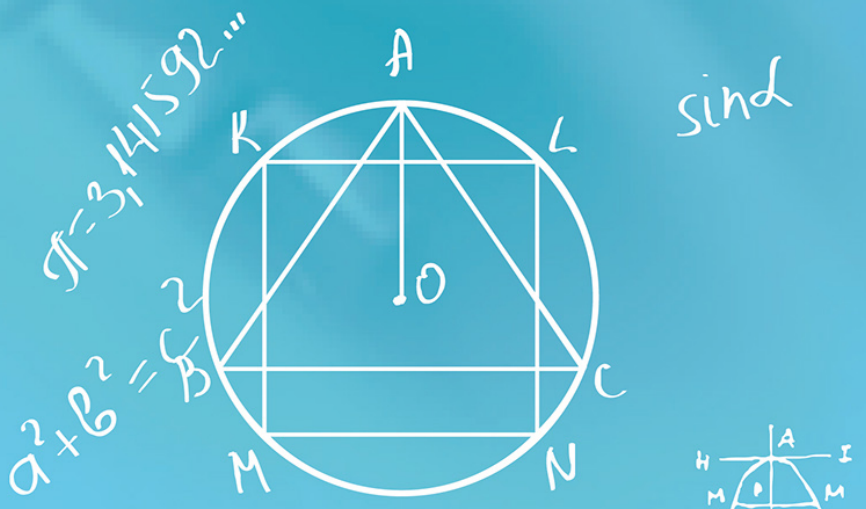
www.aurora-group.eu
www.nbpublish.com

ПРОГРАММНЫЕ СИСТЕМЫ

и вычислительные методы

научный журнал



AURORA Group s.r.o.
nota bene

Выходные данные

Номер подписан в печать: 04-04-2023
Учредитель: Даниленко Василий Иванович, w.danilenko@nbpublish.com
Издатель: ООО <НБ-Медиа>
Главный редактор: Морозов Михаил Николаевич, кандидат технических наук, mikhail.n.morozov@gmail.com
ISSN: 2454-0714
Контактная информация:
Выпускающий редактор - Зубкова Светлана Вадимовна
E-mail: info@nbpublish.com
тел.+7 (966) 020-34-36
Почтовый адрес редакции: 115114, г. Москва, Павелецкая набережная, дом 6А, офис 211.
Библиотека журнала по адресу: http://www.nbpublish.com/library_tariffs.php

Publisher's imprint

Number of signed prints: 04-04-2023
Founder: Danilenko Vasiliy Ivanovich, w.danilenko@nbpublish.com
Publisher: NB-Media ltd
Main editor: Morozov Mikhail Nikolaevich, kandidat tekhnicheskikh nauk, mikhail.n.morozov@gmail.com
ISSN: 2454-0714
Contact:
Managing Editor - Zubkova Svetlana Vadimovna
E-mail: info@nbpublish.com
тел.+7 (966) 020-34-36
Address of the editorial board : 115114, Moscow, Paveletskaya nab., 6A, office 211 .
Library Journal at : http://en.nbpublish.com/library_tariffs.php

РЕДАКЦИОННЫЙ СОВЕТ

Н. Э. Баумана, Москва, Россия E-mail: v.suzev@bmstu.ru

Леухин Анатолий Николаевич - доктор физико-математических наук, профессор, заведующий кафедрой Информационной безопасности Поволжского государственного технологического университета, Йошкар-Ола, Россия E-mail: code@volgatech.net

Гвинианидзе Темур Николаевич - Доктор технических наук, профессор, Государственный университет им. Ак. Церетели Грузия, г. Кутаиси, пр. Тамар-мепе 59. П.и 4600. temuri1951@mail.ru

COUNCIL OF EDITORS

University Georgia, Kutaisi, 59 Tamar-mepe Ave., and 4600. temuri1951@mail.ru

Требования к статьям

Журнал является научным. Направляемые в издательство статьи должны соответствовать тематике журнала (с его рубрикатором можно ознакомиться на сайте издательства), а также требованиям, предъявляемым к научным публикациям.

Рекомендуемый объем от 12000 знаков.

Структура статьи должна соответствовать жанру научно-исследовательской работы. В ее содержании должны обязательно присутствовать и иметь четкие смысловые разграничения такие разделы, как: предмет исследования, методы исследования, апелляция к оппонентам, выводы и научная новизна.

Не приветствуется, когда исследователь, трактуя в статье те или иные научные термины, вступает в заочную дискуссию с авторами учебников, учебных пособий или словарей, которые в узких рамках подобных изданий не могут широко излагать свое научное воззрение и заранее оказываются в проигрышном положении. Будет лучше, если для научной полемики Вы обратитесь к текстам монографий или диссертационных работ оппонентов.

Не превращайте научную статью в публицистическую: не наполняйте ее цитатами из газет и популярных журналов, ссылками на высказывания по телевидению.

Ссылки на научные источники из Интернета допустимы и должны быть соответствующим образом оформлены.

Редакция отвергает материалы, напоминающие реферат. Автору нужно не только продемонстрировать хорошее знание обсуждаемого вопроса, работ ученых, исследовавших его прежде, но и привнести своей публикацией определенную научную новизну.

Не принимаются к публикации избранные части из диссертаций, книг, монографий, поскольку стиль изложения подобных материалов не соответствует журнальному жанру, а также не принимаются материалы, публиковавшиеся ранее в других изданиях.

В случае отправки статьи одновременно в разные издания автор обязан известить об этом редакцию. Если он не сделал этого заблаговременно, рискует репутацией: в дальнейшем его материалы не будут приниматься к рассмотрению.

Уличенные в плагиате попадают в «черный список» издательства и не могут рассчитывать на публикацию. Информация о подобных фактах передается в другие издательства, в ВАК и по месту работы, учебы автора.

Статьи представляются в электронном виде только через сайт издательства http://www.e-notabene.ru кнопка "Авторская зона".

Статьи без полной информации об авторе (соавторах) не принимаются к рассмотрению, поэтому автор при регистрации в авторской зоне должен ввести полную и корректную информацию о себе, а при добавлении статьи - о всех своих соавторах.

Не набирайте название статьи прописными (заглавными) буквами, например: «ИСТОРИЯ КУЛЬТУРЫ…» — неправильно, «История культуры…» — правильно.

При добавлении статьи необходимо прикрепить библиографию (минимум 10–15 источников, чем больше, тем лучше).

При добавлении списка использованной литературы, пожалуйста, придерживайтесь следующих стандартов:

- ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления.
- ГОСТ 7.0.5-2008 Библиографическая ссылка. Общие требования и правила составления

В каждой ссылке должен быть указан только один диапазон страниц. В теле статьи ссылка на источник из списка литературы должна быть указана в квадратных скобках, например, [1]. Может быть указана ссылка на источник со страницей, например, [1, с. 57], на группу источников, например, [1, 3], [5-7]. Если идет ссылка на один и тот же источник, то в теле статьи нумерация ссылок должна выглядеть так: [1, с. 35]; [2]; [3]; [1, с. 75-78]; [4]....
А в библиографии они должны отображаться так:
[1]
[2]
[3]
[4]....

Постраничные ссылки и сноски запрещены. Если вы используете сноску, не содержащую ссылку на источник, например, разъяснение термина, включите сноску в текст статьи.

После процедуры регистрации необходимо прикрепить аннотацию на русском языке, которая должна состоять из трех разделов: Предмет исследования; Метод, методология исследования; Новизна исследования, выводы.

Прикрепить 10 ключевых слов.

Прикрепить саму статью.

Требования к оформлению текста:

- Кавычки даются уголками (« ») и только кавычки в кавычках — лапками (“ ”).
- Тире между датами дается короткое (Ctrl и минус) и без отбивок.
- Тире во всех остальных случаях дается длинное (Ctrl, Alt и минус).
- Даты в скобках даются без г.: (1932–1933).
- Даты в тексте даются так: 1920 г., 1920-е гг., 1540–1550-е гг.
- Недопустимо: 60-е гг., двадцатые годы двадцатого столетия, двадцатые годы XX столетия, 20-е годы XX столетия.
- Века, король такой-то и т.п. даются римскими цифрами: XIX в., Генрих IV.
- Инициалы и сокращения даются с пробелом: т. е., т. д., М. Н. Иванов. Неправильно: М.Н. Иванов, М.Н. Иванов.

ВСЕ СТАТЬИ ПУБЛИКУЮТСЯ В АВТОРСКОЙ РЕДАКЦИИ.

По вопросам публикации и финансовым вопросам обращайтесь к администратору
Зубковой Светлане Вадимовне
E-mail: info@nbpublish.com
или по телефону +7 (966) 020-34-36

Подробные требования к написанию аннотаций:

Аннотация в периодическом издании является источником информации о содержании статьи и изложенных в ней результатах исследований.

Аннотация выполняет следующие функции: дает возможность установить основное

содержание документа, определить его релевантность и решить, следует ли обращаться к полному тексту документа; используется в информационных, в том числе автоматизированных, системах для поиска документов и информации.

Аннотация к статье должна быть:

- информативной (не содержать общих слов);
- оригинальной;
- содержательной (отражать основное содержание статьи и результаты исследований);
- структурированной (следовать логике описания результатов в статье);

Аннотация включает следующие аспекты содержания статьи:

- предмет, цель работы;
- метод или методологию проведения работы;
- результаты работы;
- область применения результатов; новизна;
- выводы.

Результаты работы описывают предельно точно и информативно. Приводятся основные теоретические и экспериментальные результаты, фактические данные, обнаруженные взаимосвязи и закономерности. При этом отдается предпочтение новым результатам и данным долгосрочного значения, важным открытиям, выводам, которые опровергают существующие теории, а также данным, которые, по мнению автора, имеют практическое значение.

Выводы могут сопровождаться рекомендациями, оценками, предложениями, гипотезами, описанными в статье.

Сведения, содержащиеся в заглавии статьи, не должны повторяться в тексте аннотации. Следует избегать лишних вводных фраз (например, «автор статьи рассматривает...», «в статье рассматривается…»).

Исторические справки, если они не составляют основное содержание документа, описание ранее опубликованных работ и общеизвестные положения в аннотации не приводятся.

В тексте аннотации следует употреблять синтаксические конструкции, свойственные языку научных и технических документов, избегать сложных грамматических конструкций.

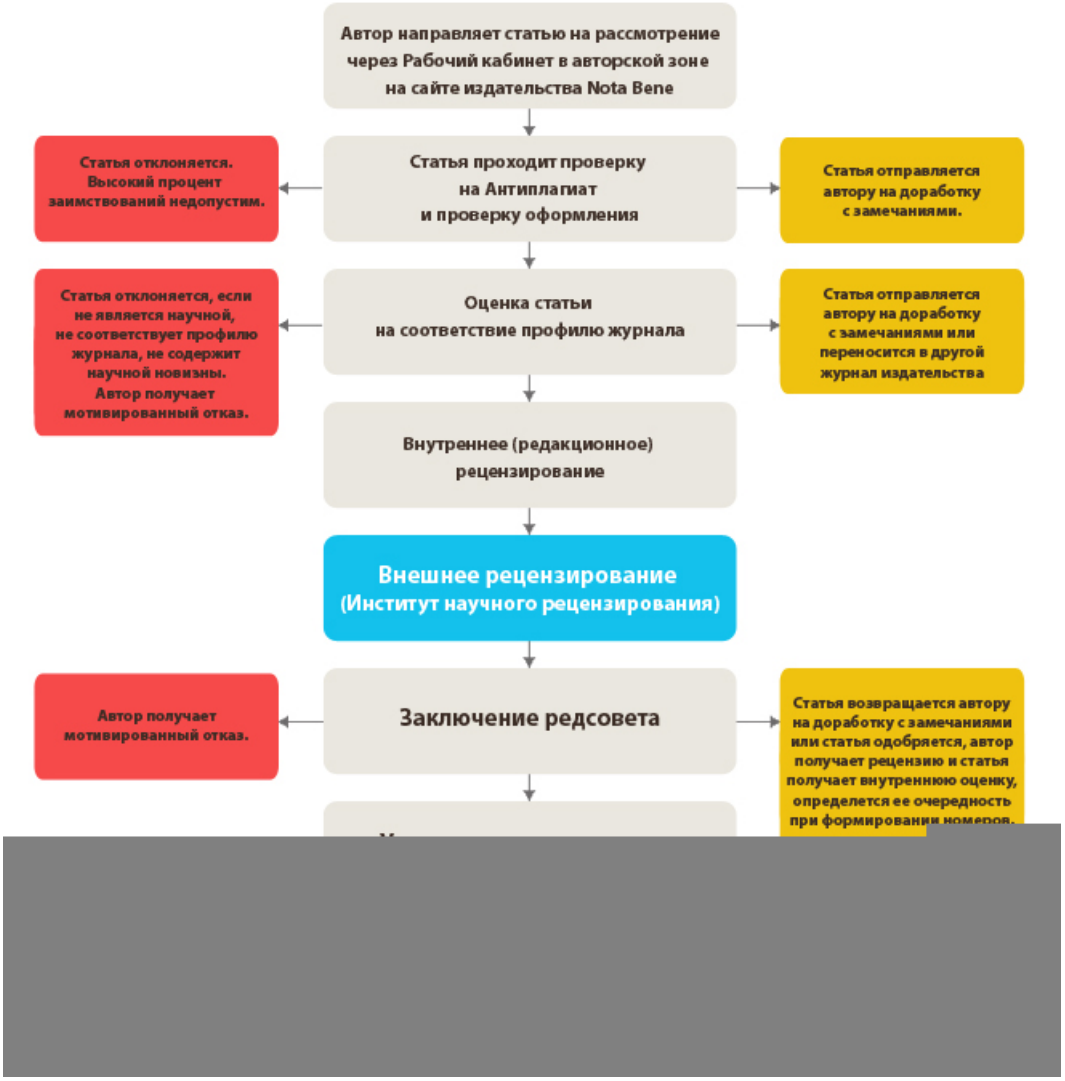
Гонорары за статьи в научных журналах не начисляются.

Материалы журналов включены:

- в систему Российского индекса научного цитирования;
- отображаются в крупнейшей международной базе данных периодических изданий Ulrich's Periodicals Directory, что гарантирует значительное увеличение цитируемости;
- Всем статьям присваивается уникальный идентификационный номер Международного регистрационного агентства DOI Registration Agency. Мы формируем и присваиваем всем статьям и книгам, в печатном, либо электронном виде, оригинальный цифровой код. Префикс и суффикс, будучи прописанными вместе, образуют определяемый, цитируемый и индексируемый в поисковых системах, цифровой идентификатор объекта — digital object identifier (DOI).

Отправить статью в редакцию

Этапы рассмотрения научной статьи в издательстве NOTA BENE.



Содержание

Contents

Программные системы и вычислительные методы
Правильная ссылка на статью:
Нуриев М.Г., Белашова Е.С., Барабаш К.А. — Конвертер Markdown-файлов в LaTeX-документ // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.39547 EDN: SNAYLQ URL: https://nbpublish.com/library_read_article.php?id=39547

Конвертер Markdown-файлов в LaTeX-документ

Нуриев Марат Гумерович

кандидат технических наук

старший преподаватель, кафедра автоматизированных систем обработки информации и управления, Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55

marat_nu1@mail.ru

Белашова Елена Семеновна

кандидат физико-математических наук

доцент, кафедра компьютерных систем, Казанский национальный исследовательский технический университет им. А.Н.Туполева-КАИ

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55

bel_lena@mail.ru

Барабаш Константин Алексеевич

студент кафедры компьютерных систем Казанского национального исследовательского технического университета им. А.Н.Туполева-КАИ

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55

kostyandriy@mail.ru

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2023.1.39547

EDN:

SNAYLQ

Дата направления статьи в редакцию:

25-12-2022

Дата публикации:

01-01-2023

Аннотация: Привычные пользователю текстовые редакторы такие как Microsoft Word, Notepad++ и другие являются «громоздкими». При своем огромном функционале они не

исключают риска неправильной конвертации документа, например, при открытии тех же Word-файлов на более старых или наоборот более новых версиях Microsoft Word. Выходом является применение языков разметок, которые позволяют маркировать блоки текста в целях их представления в нужной стилистике. В настоящее время большую популярность имеют LaTeX (набор макрорасширений системы компьютерной вёрстки TeX) и Markdown (облегчённый язык разметки, созданный с целью обозначения форматирования в простом тексте). Поэтому актуален вопрос преобразования Markdown-документа в LaTeX-документ. Существуют различные инструменты конвертации Markdown-файлов в LaTeX-документ, например, библиотека Pandoc, Markdown.lua, Lunamark и другие. Но большинство из них имеют избыточные шаги по формированию выходного документа. В данной статье освещается метод решения посредством интеграции Markdown-файла в LaTeX-документ, который потенциально позволит сократить время формирования выходного документа в отличие от существующих решений. Разработанный конвертер Markdown-файлов в LaTeX-документ позволит автоматически получить результирующий документ и снизить вероятность ошибок при ручном преобразовании текста из Markdown-формата в LaTeX-формат.

Ключевые слова:

Markdown, LaTeX, программирование, конвертер, Python, язык разметки, Overleaf, преобразование текста, Word, регулярные выражения

Введение

Привычные пользователю текстовые редакторы такие как Microsoft Word, Notepad++ и другие являются «громоздкими». При своем огромном функционале они не исключают риска неправильной конвертации документа, например, при открытии тех же Word-файлов на более старых или наоборот более новых версиях Microsoft Word. Текстовый редактор от компании Microsoft имеет два расширения: doc и docx. Хоть первый из них уже не актуален, однако старые версии Microsoft Word до сих пор используют его. Doc-файл является форматом документа используемым текстовым редактором Microsoft Word, а .docx это следующая версия, она более эффективная, создает файлы, в меньшей степени подверженные повреждению. Для большей надежности сохранения форматов и редакции в документе можно воспользоваться языком разметок [1]. Язык Markdown – один из представителей языков разметок. Он обладает следующими преимуществами:

- Универсальность: Документы с синтаксисом Markdown это простые текстовые файлы, которые можно открыть в любом текстовом редакторе.
- Простота: Язык Markdown очень прост для освоения, для этого не требуется никакие дополнительные знания.
- Большой выбор инструментов: Благодаря высокой универсальности с языком разметки Markdown можно работать в любом редакторе, выбор пользователя почти ничем не ограничивается.
- Конвертируемость: Документы Markdown легко экспортировать в любые форматы: PDF, DOC, ODT. При этом их форматирование остаётся неизменным [2].

Благодаря использованию языков разметки, в частности, Markdown можно достичь высокой сохранности редакции файла чем обусловлена актуальность разработки ПО для конвертации Markdown файлов в LaTeX документы. LaTeX документ – текстовый файл содержащий команды языка разметки.

Язык упрощённой разметки Markdown

Целью языка Markdown является легкая запись и чтение. Внешне Markdown напоминает язык HTML, однако не является им и не может его заменить так как имеет очень мало типов синтаксиса, идея Markdown состоит в том, чтобы упростить чтение, запись и изменение документов. В отличие от HTML, в связи с огромным количеством разнообразных тегов, является сложным для чтения и понимания того, как будет выглядеть результат, Markdown не обладает этой проблемой, потому что является настолько легким для чтения на сколько это возможно. Для использования Markdown достаточно просто применить простые теги к тексту. Язык разметки Markdown обладает блочными (рис. 1) и строчными элементами (рис. 2). Как видно из примера Markdown, в отличие от HTML, не требует больших каскадных выделений для создания отформатированного абзаца.

````
Here's an example:

```js
function test() {
  console.log("notice the blank line before this function?");
}
```

syntax highlighting:
```ruby
require 'redcarpet'
markdown = Redcarpet.new("Hello World!")
puts markdown.to_html
```
````

Рисунок 1. Пример блочного кода в Markdown

```
# This is an H1

## This is an H2

###### This is an H6
```

Рисунок 2. Пример строчного кода в Markdown

Структура документов LaTeX

LaTeX – набор макрорасширений системы компьютерной вёрстки TeX, который облегчает набор сложных документов. Основная идея LaTeX заключается в том, что авторам нужно думать только о содержании, не беспокоясь о конечном визуальном облике. Разрабатывая свой документ автор указывает логическую структуру текста (разбивая его на главы, разделы, таблицы, изображения), а LaTeX решает вопросы его отображения. Так содержание отделяется от оформления. Оформление при этом или определяется заранее (стандартное), или разрабатывается для конкретного документа.

Документ LaTeX гораздо менее читаемый чем документ Markdown и обладает жесткой структурой, которая сравнима с программным кодом. Этот документ содержит специальные команды языка разметки и делится на преамбулу и тело. Преамбула содержит информацию: о классе документа, об авторе, о дате создания и прочее.

Тело документа содержит текст документа и команды разметки, оно ограничивается командами begin{document} и end{document}.

Веб-редактор Overleaf

Веб-редактор Overleaf – редактор LaTeX файлов, однако помимо формата .tex, являющегося стандартным форматом LaTeX документом, может читать формат .sty, в котором можно описать стиль текста, что позволяет не тратить время на редакцию текста, отступы и прочее – достаточно описать текст, заключив его в блоки LaTeX документа. Помимо вышеуказанного преимущества имеет удобный интерфейс (рис. 3).

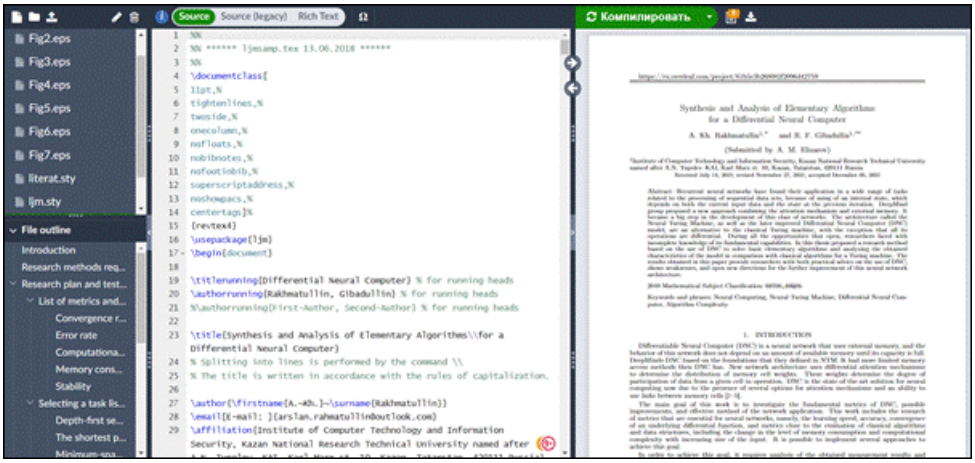


Рисунок 3. Интерфейс Overleaf

Также позволяет редактировать один и тот же файл нескольким пользователям, поддерживает практически все функции LaTeX и позволяет скомпилировать документа в формат .pdf. Overleaf написан на языке CoffeScript, использует Node.js и такие СУБД как MangoDB и Redis [3].

Предпосылки к разработке конвертера

Разработка конвертера позволит автоматизировать процесс преобразования Markdown-документов в LaTeX-документы, что устранит или сведет к минимуму процедуру ручного преобразования и позволит избежать ошибок, вызванных человеческим фактором.

Разрабатываемый конвертер Markdown-файлов в LaTeX-документ будет предназначен для:

- Подготовки выбранного документа к конвертации.
- Снабжения результирующего файла необходимыми библиотеками, требуемыми для корректного отображения всех функций исходного документа.
- Конвертации документа .md в формат .tex.
- Создания результирующих файлов, а именно отфильтрованного .md и финального .tex файла.

При этом конвертер должен выполнять обработку выбранного файла за короткий промежуток времени (не более 30 секунд на конвертацию документа размера 500 КБ), иметь интерактивный пользовательский интерфейс.

Структурная схема конвертера Markdown-файлов в LaTeX-документ

На основании поставленной задачи разработана структурная схема конвертера Markdown-файлов в LaTeX-документ (рис. 4).

На этапе ввода файла пользователь выбирает исходный файл, который прежде чем стать частью LaTeX-документа должен пройти предварительную обработку. Основным этапом создания конвертера Markdown-файлов в LaTeX-документ является этап обработки файлов, в котором исходный файл должен быть подготовлен к включению в шаблон LaTeX. Он представляет собой стандартный документ LaTeX, в котором включены все

необходимые компоненты. На этом этапе документ будет обрабатываться программой, написанной на Python, после чего файл будет подготовлен к выводу.

Вывод будет осуществлен в виде двух файлов. Первый файл – это Markdown-документ, в котором будет храниться текст. Вторым файлом является шаблон LaTeX, в котором хранятся данные о разметки текста.

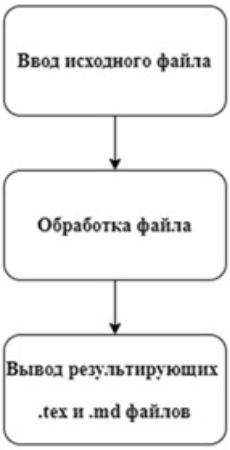


Рисунок 4. Структурная схема обработки документа

Алгоритм работы конвертера Markdown-файлов в LaTeX-документ

Схема алгоритма работы конвертера Markdown-файла в LaTeX-документ показан на рисунке 5.

Рисунок 5. Алгоритм работы конвертера Markdown-файла в LaTeX-документ

Пользователь выбирает Markdown-файл и одновременно с этим происходит создание LaTeX-документа, в котором хранится информация о стиле и библиотеках, используемых для корректного отображения файла в формате .tex. Tex-компоненты позволяют использовать русский язык, обозначают кодировку символов, включают корректное

отображение программного кода, а также делают его цветным.

С помощью следующих ключевых слов:

- usepackage{packages/sleek-title}
- usepackage{packages/sleek-theorems}
- usepackage{packages/sleek-listings}

задается стиль текста.

На этапе очистки из исходного файла удаляются объекты, автоматическое добавление которых в LaTeX-документ затруднено или не является возможным, это является узким местом выбранного метода решения задачи. На рисунке 6 показано как выглядит титул в Markdown-файле, этот текст не читается в LaTeX-документе, конвертер выдаст ошибку при обработке. На рисунке 7 показано как выглядит титул в LaTeX-документе.

```
---
layout: post
title: "Параллельная программа проверки орфографии"
date: 2021-05-01
---
```

Рисунок 6. Пример титульного листа в markdown файле

```
\title{Тест}
\author{тест}
\date{22.11.22}
\begin{document}
\maketitle
\markdownInput{result.md}
\end{document}
```

Рисунок 7. Пример титульного листа в LaTeX-документе

Внутри фигурных скобок ключевых слов title {Заголовок}, author{автор} и date{дата} вводятся соответствующий текст заголовка, ФИО автора и дата публикации, а ключевое слово maketitle в теле LaTeX-документа создает титульный лист. Однако в заголовке Markdown-файла лежит информация, которая считывается с помощью программы и записывается в начало файла .md. Далее исходный Markdown-файл также требуется очистить от списка источников, так как в LaTeX-документе ссылки представляются иной языковой конструкцией. Далее Markdown-файл добавляется в шаблон LaTeX. С помощью ключевого слово markdownInput{пример.md} файл markdown присоединяется к LaTeX-документу. Данное ключевое слово должно следовать после команды maketitle, в противном случае заголовок будет создан после основного текста [4].

Допустим и другой способ, когда содержимое Markdown-файла находится в теле LaTeX-документа вместо ключевого слова markdownInput{пример.md}.

Программная реализация конвертера Markdown-файлов в Latex-документ

Программная реализация состоит из нескольких этапов.

В рамках первого этапа с помощью блока очистки выполняется выделение полезной информации и запись в промежуточный файл. Второй этап заключается в повторной очистке, который полностью подготовит документ к конвертации, устранив из файла не

поддающие преобразованию фрагменты. После обработки модулем очистки результирующий для него файл будет прикреплен к шаблону LaTeX с помощью ключевого слова MarkdownInput{файл}, где файл шаблона создается в момент, когда будет готов очищенный файл. Последним этапом работы является вывод двух результирующих файлов: один – LaTeX-файл, который хранит в себе ссылку на второй файл формата Markdown, а также данные о разметки текста и необходимые LaTeX библиотеки для корректной работы и отображения результата. На рисунке 8 приведена общая структурная схема программного обеспечения.

Рисунок 8. Структурная схема программного обеспечения конвертера Markdown-файлов в LaTeX-документ

Таким образом, программное обеспечение можно разделить на несколько модулей, таких как:

1. Модуль очистки;
2. Модуль создания LaTeX шаблона;
3. Модуль вывода результата.

Модуль очистки открывает файл и построчно с помощью оператора while обрабатывает текст исходного файла. Первая строка обрабатываемого файла содержит значение “---”, в кодировке Markdown оно создает прямую горизонтальную линию внутри блока, огражденного парой таких значений. В данной строке находится непереводимый в LaTeX заголовок файла. С помощью Python-выражения *file = open ("имя_файла.md", "r", encoding='utf-8')* открываем файл "имя_файла.md", для чтения в кодировке UTF-8. Далее с помощью инструкции file.readline считываем первую строку файла и сдвигаем первую строку, предварительно записав ее значение в переменную. Это требуется для того, чтобы найти конец заголовка. На листинге 1 приведен код программы, выполняющей первый этап очистки файла.

Листинг 1. Первый этап очистки

```
    del_pattern_list = r"title:"
    first_line = in_file.readline()
    line_index=0
    skip = False
    while True:
    # считываем строку
        line = in_file.readline()
    # прерываем цикл, если строка пустаяif not line:
        print("конец", line_index)break
    # выделение заголовка
        if re.search(del_pattern_list, line):
            test = re.split(r'\"', line)
            header= "# "+ test[1] +"\n\n"
            filter_file.writelines(header)
    # поиск лишнего заголовка
        if line == first_line:
            skip = True
        if skip:
            filter_file.writelines(in_file.readlines())
        line_index+=1
    filter_file.close
```

В результате выполнения данного кода (см. листинг 1) файл очищается от заголовка, при этом в переменную *header* записывается значение заголовка, написанного в исходном файле, с пометкой *tittle* . Поиск происходит при помощи использования регулярного выражения, которое является удобным инструментом при работе с текстом. Текст исходного файла записывается в промежуточный файл начиная со следующей строки после нахождения второй границы заголовка. Данное решение позволяет удалять заголовок за 4 итерации, что увеличивает скорость обработки, так как не нужно исследовать каждую строку файла.

Второй этап обработки требуется для удаления списка источников. Здесь происходит обработка файла *filter_file* , созданного в первом этапе. Задачей данного этапа является определение и удаление списка источников, так как в Markdown-файле он формируется с помощью непереводимых в LaTeX команд, вызывающих ошибку при конвертации. На листинге 2 приведен код программы, решающий задачу второго этапа очистки файла.

Листинг 2. Второй этап очистки

```
    def sourse_list_check(line, line_index):
        sourse_pattern = r'\[\^\d\]'
        sourse_list =re.match(sourse_pattern, line)
        if sourse_list:
            return line_index


    line_index=0
    while True:
        line = filter_file.readline()
        if not line:
            print("конец", line_index)
            break
        if sourse_list_check(line, line_index) != None:
            sourse_index = line_index
            print(sourse_index)
            break
        line_index+=1
    filter_file.close
    filter_file = open("_posts/test_filter.md", "r",encod-
ing='utf-8')
    result = open("results/result.md", "w", encoding='utf-8')
    result.writelines(filter_file.readlines()[:sourse_index-1])
```

Здесь также применяются регулярные выражения. С помощью функции

sourse_list_check(line, line_index) , которая принимает текущую строку и её номер, происходит поиск по заданному шаблону с возвращением номера строки при совпадении. Номер строки запоминается и происходит выход из цикла, далее данные из промежуточного файла записываются в результирующий файл до строки с началом списка источников. Для этого из найденного функцией значения вычитаем 1, иначе первая строка со списком источников войдет в файл.

В результате выполнения двух этапов очистки создается результирующий Markdown-файл.

Модуль создания LaTeX шаблона

Для создания LaTeX шаблона использовался пустой шаблон LaTeX документа, в который были добавлены требуемые библиотеки. На листинге 3 показан готовый LaTeX шаблон.

Листинг 3. LaTeX шаблон

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[english,russian]{babel}
\usepackage[T1]{fontenc}
\usepackage[fencedCode,inlineFootnotes,citations,definitionL
ists,hashEnumerators,smartEllipses,hybrid]{markdown}
\usepackage{packages/sleek-title}
\usepackage{packages/sleek-theorems}
\usepackage{packages/sleek-listings}
\usepackage[noheader]{packages/sleek}
\usepackage{minted}
\title{Тест}
\author{тест}
\date{22.11.22}
\begin{document}
\maketitle
\markdownInput{result.md}
\end{document}
```

documentclass[12pt, letterpaper]{article} - определяет тип документа. В команду можно передать некоторые дополнительные параметры, заключенные в скобки и разделенные запятыми. В примере дополнительные параметры задают размер шрифта (12pt), размер по умолчанию – 10pt и размер бумаги (letterpaper).

usepackage[utf8]{inputenc} – это кодировка документа, позволяющая использовать в тексте символы за пределами ASCII (например, à, ü, č...). Его можно опустить или изменить на другую кодировку, но рекомендуется использовать utf-8.

usepackage[english,russian]{babel} определяет используемые языки по умолчанию, русский язык не доступен.

usepackage[T1]{fontenc} – это кодировка шрифта (определяет, какой шрифт используется.

usepackage[fencedCode,inlineFootnotes,citations,definitionLists,hashEnumerators,smartEllipses,hybrid]{markdown} – множество библиотек для Markdown, требуемых для корректного отображения его функций.

fencedCode – используется для корректного отображения кода.

hashEnumerators – используется для создания упорядоченных списков.

inlineFootnotes – позволяет создать ссылки на внешние сайты.

hybrid – позволяет использовать LaTeX код внутри Markdown документа.

Строки:

usepackage{packages/sleek-title}

usepackage{packages/sleek-theorems}

usepackage{packages/sleek-listings}

usepackage[noheader]{packages/sleek}

подключают соответствующий шаблон, параметр noheader выключает стандартный заголовок стиля.

Внутри файлов шаблона описываются: разметка текста, стили вывода уравнений, кода, заголовков, обычного текста.

Результат выводится после выполнения работы блока очистки, однако он не является читаемым для пользователя, так как закодирован. С помощью сервиса overleaf происходит чтение результирующего файла и конвертация в формат pdf.

Это решение имеет ряд преимуществ, например, высокая производительность, возможность редактировать Markdown-файл после включения его в LaTeX документ, что сильно повышает гибкость решения.

Заключение

По результатам проделанной работы был разработан конвертер Markdown-файлов в LaTeX-документ, нацеленный на исключение рутинной ручной работы по преобразованию Markdown-файлов в LaTeX-документ.

Решены задачи:

1. Представлен программный проект конвертера;
2. Разработан и реализован алгоритм работы модулей очистки;
3. Разработан модуль формирования выходного документа с заданным LaTeX-шаблоном.

В будущем планируется развивать данный проект посредством расширения приложения новыми стилями выходного документа, оптимизацией кода, по критерию быстродействия [5,6], добавлением новых возможностей, таких как пакетная обработка файлов, интеграция программы в качестве расширения для браузера и обеспечения доступа к базе данных с результирующими документами с соблюдением принципов информационной безопасности [7,8].

Библиография

1. Мехмонов И. Н. Инструментарии автоматизированного формирования динамических документов //Прикладная математика и информатика: Современные исследования в области естественных и технических наук. – 2020. – С. 883-886.
2. Павлов Д. А. Автоматическая вёрстка и оформление научной и программной документации //Компьютерные инструменты в образовании. – 2018. – №. 6. – С. 39-46.

3. B. Luo, W. Zhu, P. Li and Z. Han, "Distributed Dynamic Cuckoo Filter System Based on Redis Cluster," 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), 2018, pp. 244-247, doi: 10.1109/BDS/HPSC/IDS18.2018.00059.
4. J. Tippayachai and S. Kiattisin, "Academic Publishing Solution Based on LATEX Class Package Implementation for ITMSOC Journal," 2018 3rd Technology Innovation Management and Engineering Science International Conference (TIMES-iCON), 2018, pp. 1-5, doi: 10.1109/TIMES-iCON.2018.8621689.
5. Gibadullin, R.F., Lekomtsev, D.V. & Perukhin, M.Y. Analysis of Industrial Network Parameters Using Neural Network Processing. Sci. Tech. Inf. Proc. 48, 446–451 (2021). https://doi.org/10.3103/S0147688221060046.
6. Гибадуллин Р.Ф. Потокобезопасные вызовы элементов управления в обогащенных клиентских приложениях // Программные системы и вычислительные методы. – 2022. – № 4. – С. 1-19. DOI: 10.7256/2454-0714.2022.4.39029 EDN: IAXOMA URL: https://nbpublish.com/library_read_article.php?id=39029.
7. Гибадуллин Р.Ф. Организация защищенной передачи данных в сенсорной сети на базе микроконтроллеров AVR // Кибернетика и программирование. – 2018. – № 6. – С. 80-86. DOI: 10.25136/2306-4196.2018.6.24048 URL: https://nbpublish.com/library_read_article.php?id=24048.
8. Гибадуллин Р. Ф. Развитие единообразного формализма защиты точечных, линейных и площадных объектов картографии // Вестник Казанского государственного технического университета им. А.Н. Туполева. – 2010. – №. 2. – С. 101-105.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Предмет исследования – разработка конвертера Markdown-файлов в LaTeX-документ.

Методология исследования основана на сочетании теоретического и эмпирического подходов с применением методов анализа, обобщения, сравнения, синтеза, программирования.

Актуальность исследования определяется широким распространением информационно-коммуникационных технологий, важностью проектирования и реализации соответствующих программных продуктов.

Научная новизна связана с разработкой автором программного продукта (конвертера), который включая алгоритм работы модулей очистки, а также модуля формирования выходного документа с заданным LaTeX-шаблоном.

Статья написана русским литературным языком. Стиль изложения научный.

Структура рукописи включает следующие разделы: Введение ("громоздкие" текстовые редакторы Microsoft Word, Notepad++ и др., риск неправильной конвертации документа, язык разметок, преимущества языка Markdown), Язык упрощённой разметки Markdown

(цель языка, отличия от языка HTML), Структура документов LaTeX (основная идея LaTeX, документ LaTeX), Веб-редактор Overleaf (формат .sty, удобный интерфейс, редактирование файла нескольким пользователям), Предпосылки к разработке конвертера (структурная схема конвертера Markdown-файлов в LaTeX-документ, структурная схема обработки документа, алгоритм работы конвертера Markdown-файлов в LaTeX-документ, пример титульного листа в Markdown-файле, пример титульного листа в LaTeX-документе),
Программная реализация конвертера Markdown-файлов в Latex-документ (структурная схема программного обеспечения конвертера Markdown-файлов в LaTeX-документ, модули очистки, создания LaTeX шаблона, вывода результата), Модуль создания LaTeX шаблона (разметка текста, стили вывода уравнений, кода, заголовков, обычного текста), Заключение (выводы), Библиография.

Текст включает восемь рисунков, три листинга. Листинги также можно обозначить рисунками.

Содержание в целом соответствует названию. Представлено описание конвертера Markdown-файлов в LaTeX-документ, нацеленный на исключение рутинной ручной работы, а также определены перспективы развития проекта (расширение новыми стилями, оптимизация кода, увеличение быстродействия, добавление пакетной обработки файлов и пр.).

Библиография включает восемь источника зарубежных авторов (научные статьи). Библиографические описания некоторых источников требуют корректировки в соответствии с ГОСТ и требованиями редакции, например:
3. Luo B., Zhu W., Li P., Han Z. Distributed Dynamic Cuckoo Filter System Based on Redis Cluster // IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS). Место издания ???, 2018. P. 244–247.
Возможно излишнее самоцитирование (Гибадуллин Р. Ф. с соавторами).

Апелляция к оппонентам (Мехмонов И. Н., Павлов Д. А., Luo B., Zhu W., Li P.,Han Z., Tippayachai J., Kiattisin S. и др.) имеет место.

В целом материал представляет интерес для читательской аудитории и после доработки может быть опубликован в журнале «Программные системы и вычислительные методы».

Программные системы и вычислительные методы
Правильная ссылка на статью:
Викторов И.В., Гибадуллин Р.Ф. — Разработка синтаксического дерева для автоматизированного транслятора последовательного программного кода в параллельный код для многоядерных процессоров // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.38483 EDN: ANMSZI URL: https://nbpublish.com/library_read_article.php?id=38483

Разработка синтаксического дерева для автоматизированного транслятора последовательного программного кода в параллельный код для многоядерных процессоров

Викторов Иван Владимирович

аспирант кафедры компьютерных систем Казанского национального исследовательского технического университета им. А.Н. Туполева - КАИ (КНИТУ-КАИ)

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55, оф. 432

victorov.i.vl@yandex.ru

Гибадуллин Руслан Фаршатович

ORCID: 0000-0001-9359-911X

кандидат технических наук

доцент кафедры компьютерных систем Казанского национального исследовательского технического университета им. А.Н. Туполева-КАИ (КНИТУ-КАИ)

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55, каб. 432

CSharpCooking@gmail.com

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2023.1.38483

EDN:

ANMSZI

Дата направления статьи в редакцию:

19-07-2022

Дата публикации:

28-01-2023

Аннотация: Появление многоядерных архитектур чрезвычайно стимулировало область параллельных вычислений. Однако разработка параллельной программы и ручное распараллеливание унаследованных последовательных программных кодов являются трудоемкой работой, программист должен обладать хорошими навыками применения

методов параллельного программирования. Данное обстоятельство определяет актуальность предмета исследования работы – разработка транслятора последовательного кода в параллельный. В статье приводится обзор существующих решений в рамках выбранного направления исследований, рассматриваются их преимущества и недостатки. Предлагается принцип формирования синтаксического дерева, который основан на JSON формате (текстовый формат обмена данными, основанный на JavaScript), и разбирается пример формирования синтаксического дерева на основе данного принципа. Результатом работы является подход к построению программной платформы трансляции последовательного кода в параллельный. Отличительной особенностью разработанной платформы является web-сервис, который потенциально позволяет расширить транслятор другими языками программирования. Взаимодействие с программной средой осуществляется посредством REST-запросов (HTTP-запросов, предназначенных для вызова удаленных процедур). Разработанная программная платформа состоит из трёх модулей: модуль обработки запросов, обеспечивающий взаимодействие с внешними системами посредством REST-запросов; модуль построения дерева, служащий для формирования синтаксического дерева на основе исходного программного кода; модуль преобразования кода, получающий параллельный программный код на основе синтаксического дерева.

Ключевые слова:

многоядерные процессоры, параллельные вычисления, параллельное программирование, многопоточное программирование, автоматизированный транслятор, JSON формат, языки программирования, синтаксическое дерево, веб-сервис, REST-запросы

Введение

В последнее время в целях повышения производительности вычислительных систем наблюдается увеличение числа мультипроцессорных комплексов, позволяющих оперировать большим объёмом вычислений. Для решения вычислительно сложных задач за приемлемое время требуется разработка программных модулей с применением многопоточного программирования. Для написания параллельных программ нужны высококвалифицированные программисты: в работе требуется участие аналитика, программиста, специалиста по тестированию. Компании несут существенные финансовые затраты, оплачивая работу IT-специалистов подобного класса.

Подспорьем для параллельного программирования являются специализированные языки, например, Cilk. С применением данного языка можно обеспечить вызовы функций асинхронно до тех пор, пока в родительской функции не будет достигнута точка синхронизации. Этот стиль параллелизма называется "разделяй и властвуй" (divide and conquer), или fork-join parallelism. В нём программа подразделяется на более мелкие задачи, которые планируются к параллельному выполнению. Помимо Cilk существуют специализированные библиотеки, например, Java Fork Join.

Зачастую программисту из-за сложности отладки изначально приходится писать последовательную (однопоточную) программу, а далее адаптировать решение в параллельный (многопоточный) эквивалент. Поэтому актуален вопрос по автоматизированному (или автоматическому) преобразованию последовательного кода в параллельный. Автоматическое преобразование не требует участия человека, но

сопровождается рядом трудностей по достижению однозначного решения (последовательный код может иметь различные параллельные эквиваленты). Автоматизированный транслятор использует диалог с программистом для уточнения свойств последовательной программы в целях получения наилучшего параллельного эквивалента, который потокобезопасен и оптимизирован по использованию аппаратных ресурсов и по критерию быстродействия.

В настоящее время отсутствуют программы, позволяющие успешно справляться с автоматической параллелизацией. Это связано, например, с наличием в программах вложенных функций, сложных выражений формирования циклов при которых усложняется анализ зависимостей по данным. Наиболее известными программными продуктами, реализующими автоматизированный подход, являются [1,2]: Parawise, Polaris, Cetus, Pluto, Polly, Par4All, CAPO, WPP, SUIF, VAST/PARALLEL, OSCAR, САПФОР.

Проведем сравнительный анализ на примере Parawise. Рассмотрим шаги алгоритма, отвечающего за преобразования последовательного в параллельный код в программном продукте Parawise (см. рис. 1):

1. Получение последовательного кода.
2. Анализ зависимостей последовательного кода на основе пользовательских данных и запись информации в БД.
3. Разделение входного кода на токены (объекты, создающиеся из лексемы в процессе лексического анализа, например, “{”, “}”, “case”, “do”, “else”, “for”, “if”, “sizeof” и т.д.) и запись информации в БД при участии пользователя.
4. Контроль масок (контроль максимальных и минимальных значений для условий) и запись информации в БД при участии пользователя.
5. Вычисление и генерация структуры синтаксического дерева и запись информации в БД.
6. Оптимизация кода (снижение использования памяти) и запись информации в БД.
7. Генерация кода и переход к следующему разделу кода в п.3, либо формирование результата преобразования последовательного кода в параллельный код.

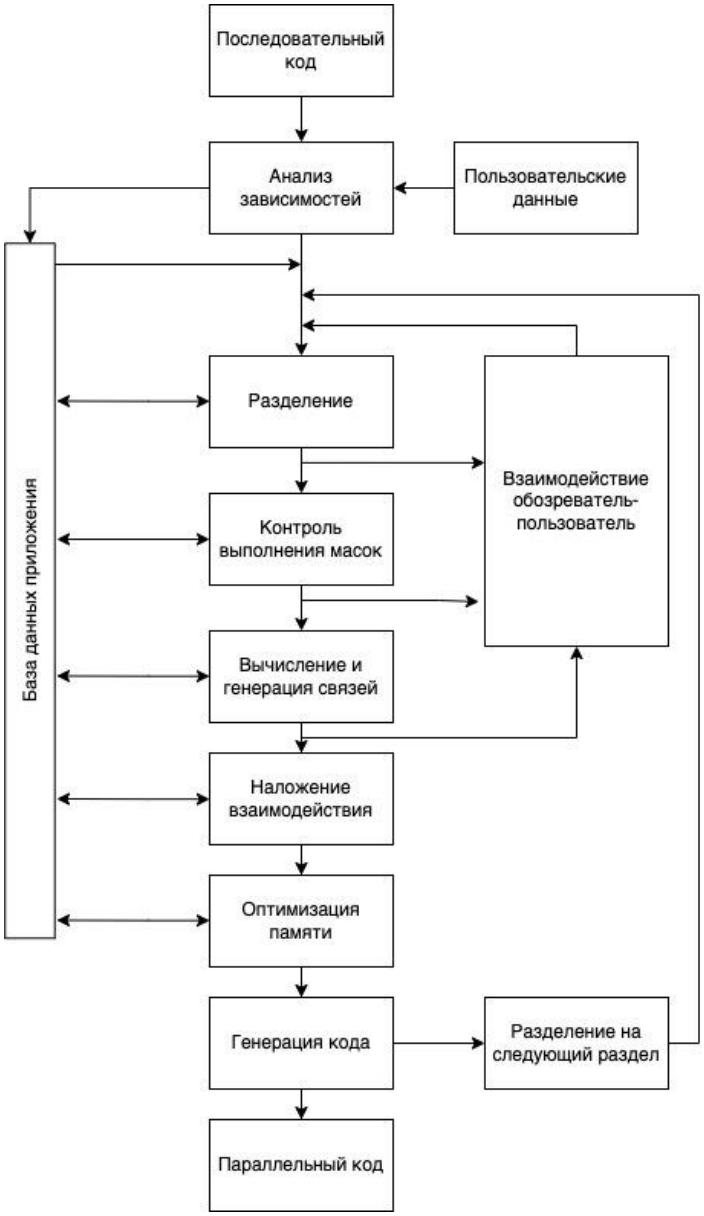


Рисунок 1. Алгоритм работы Parawise

На рисунке 2 представлены шаги преобразования последовательного кода в параллельный для программного продукта CAPO.

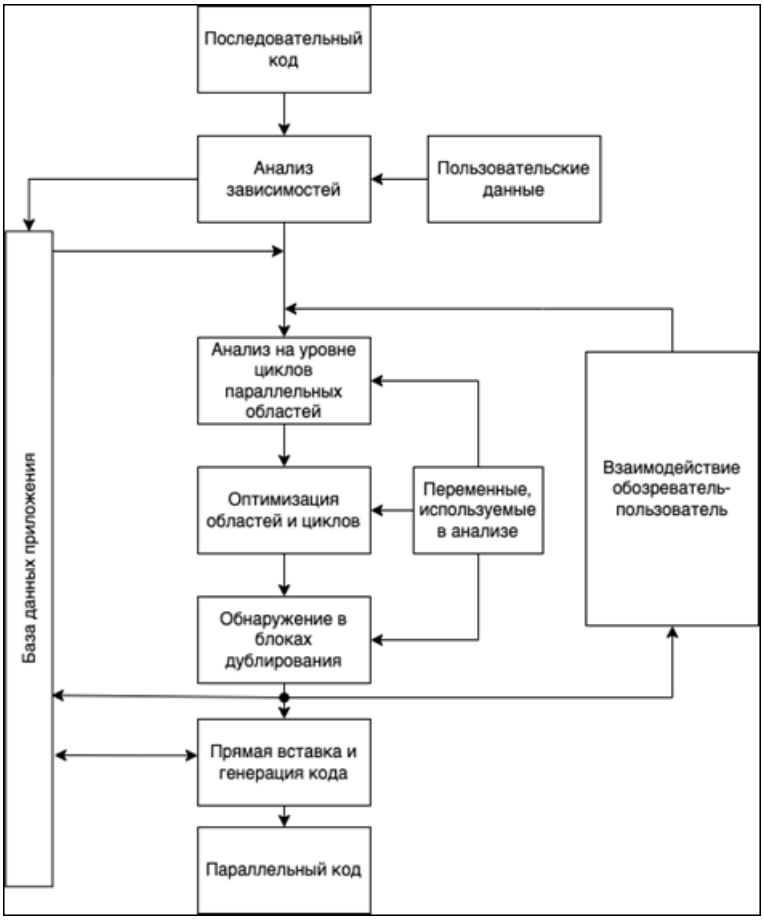


Рисунок 2. Алгоритм работы CAPO

1. Получение последовательного кода, который требуется распараллеливать.
2. Анализ зависимостей в полученном последовательном коде с обрабатываемыми пользовательскими данными и запись данных в БД.
3. На основе входных зависимостей, БД, данных взаимодействия с пользователем и переменных, используемых в анализе, формирование анализа на уровне циклов параллельных областей.
4. Оптимизация областей и циклов на основе анализа на уровне циклов параллельных областей и переменных, используемых в анализе, формирование оптимизации областей и циклов.
5. На основе полученных данных производится обнаружение дублирования в блоках. В БД вносятся промежуточные данные и происходит опрос пользователя на корректировку данных.
6. Выполнение прямой вставки полученными данными, генерация кода, обмен данными с БД.

Заметим, что Parawise и CAPO работают по схожему алгоритму, как и программы Cetus, Pluto, Polly, Par4All, CAPO, WPP, SUIF, VAST/PARALLEL, OSCAR, САПФОР. Одним из минусов является то, что данные программные продукты десктопные, т.е. имеют зависимость от целевой платформы (нельзя запустить на разных операционных системах), также требуется отслеживать обновление программного продукта. Отдельно следует выделить блок взаимодействия с пользователем, при котором пользователь выбирает требуемое решение. Реализация данного блока, представленная в CAPO и Parawise, исключает возможность автоматического параллелизма. Это требует от программиста наличие знаний в области параллельного программирования [3,4].

Предлагаемый в статье подход позволяет перейти к автоматизации перевода параллельного программного кода с наименьшим участием оператора, так как блок взаимодействия с пользователем является само обучаемым, основанным на глубоком обучении (deep learning) [5,6].

Архитектура транслятора

При разработке транслятора позволяющего преобразовать последовательный код в параллельный было принято решение сделать веб-приложение. Данное решение вызвано следующими факторами:

- веб-приложение не требует установки;
- все обновления происходят на сервере, доставляются пользователю сразу (достаточно перезагрузить веб-страницу);
- веб-приложение доступно из любой точки мира, с любого устройства, а пользовательские файлы всегда будут под рукой при наличии Интернет-соединения.

Проект транслятора имеет микросервисную архитектуру (рис. 3). Приложение состоит из нескольких программ (сервисов):

- сервис обработки запросов (обработчик запросов),
- сервис построения синтаксического дерева,
- сервис преобразования кода,
- база данных на базе СУБД PostgreSQL.

Сервис обработки запросов позволяет принимать REST-запросы пользователя, а именно запрос на преобразование последовательного кода в параллельный [7,8]. В запросе указывается последовательный код, требуемый для распараллеливания, а также язык программирования, на котором он написан. Также сервис позволяет делать запросы к базе данных для получения данных о синтаксическом дереве и преобразованном коде.

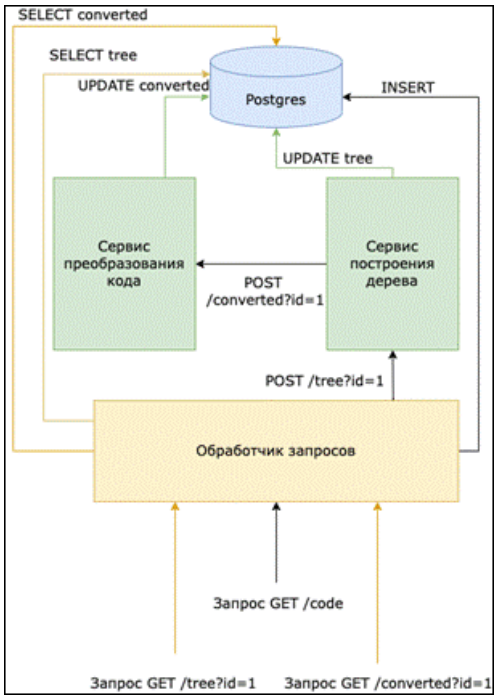


Рисунок 3. Архитектура транслятора

Сервис построения дерева преобразует последовательный код в синтаксическое дерево. Записывает данные о синтаксическом дереве в базу данных, после записи данных в БД отправляет запрос о завершении построения синтаксического дерева в сервис преобразования кода.

Сервисы формирования параллельной программы

Сервис преобразования кода забирает данные о построенном синтаксическом дереве из БД, строит различные схемы распараллеливания для исходной программы. Далее производится оценка построенной схемы по набору правил полученных из БД

соответствующего синтаксису языка преобразования текста из синтаксического дерева, и строит текст программы с применением принципов параллельного программирования и записывает в БД [9,10]. Создание параллельной программы можно рассмотреть на диаграмме последовательностей (рис. 4). Дадим описание данной диаграммы.

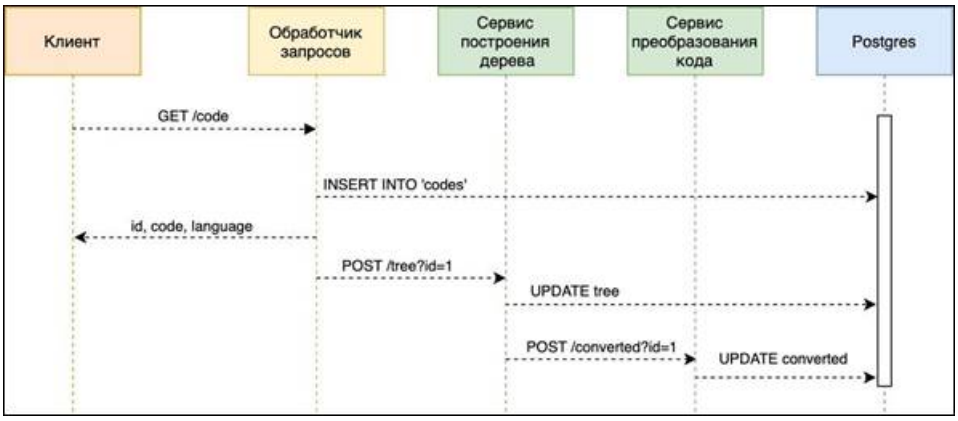


Рисунок 4. Диаграмма последовательностей создания параллельной программы

Клиент отправляет REST (GET '/code') запрос на преобразование последовательного кода в параллельный. Обработчик запросов вставляет новые данные в таблицу *code* базы данных и получает значение идентификатора нового поля. Следующим этапом обработчик запросов отправляет данные о записанном идентификаторе в сервис построения синтаксического дерева POST запросом. Сервис построения дерева получает данные о последовательном коде из БД и преобразует их в синтаксическое дерево. Результат преобразования записывается в БД и передается сервису преобразования кода POST запросом с идентификатором. Сервис преобразования кода преобразует синтаксическое дерево в параллельный код и записывает его в БД.

Построение программы выполняется в соответствии с микросервисной архитектурой. В качестве плюса микросервисов можно выделить наличие дополнительных возможностей по добавлению нового функционала программы для работы с другими языками: добавление синтаксического дерева или преобразование в параллельный код. В случае появления большей нагрузки со стороны пользователей добавляется брокер сообщений RabbitMQ или Kafka, позволяющий преобразовывать сообщение (для RabbitMQ – AMQP, Kafka – Kafka protocol) от приложения-источника в приложение-приемник, тем самым выступая между ними посредником. В этом случае клиент отправляет запрос на преобразование кода в сервис обработчика запросов. Обработчик запросов записывает данные в БД, отправляет код в брокер сообщений с идентификатором и языком программирования, который требуется преобразовать в синтаксическое дерево. Сервис построения дерева преобразовывает синтаксическое дерево в требуемый код для данного языка программирования. После этого происходит запись данных в БД и отправка идентификатора и названия языка программирования в брокер сообщений (система, преобразующая сообщение от источника данных "producer" в сообщение принимающей стороны "consumer"). Сервис преобразования кода через брокер сообщений преобразует код в параллельный эквивалент требуемого языка программирования.

Рисунок 5. Архитектура транслятора с использованием брокера сообщений

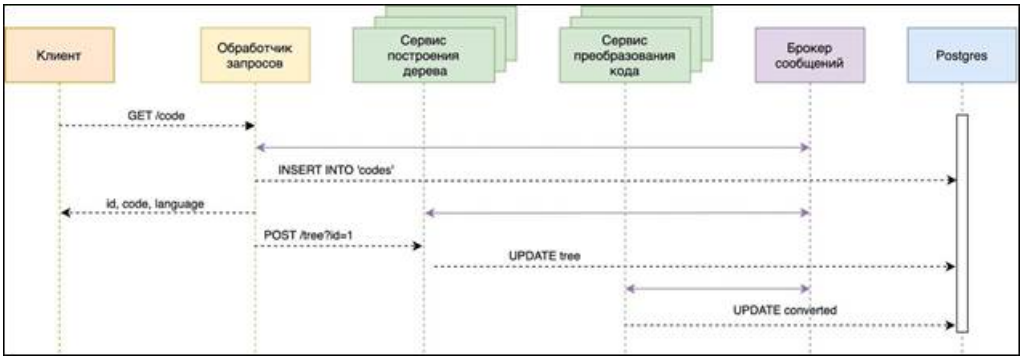


Рисунок 6. Диаграмма последовательностей создания параллельной программы с использованием брокера сообщений

Построение синтаксического дерева

В ходе проектирования были рассмотрены различные форматы синтаксического дерева.

Наиболее популярным является ANTLR4 это библиотека лексического анализа для Another Tool for Language Recognition (ANTLR). Принцип работы ANTLR основан на синтаксическом анализе входного потока данных, построения синтаксического дерева, обхода этого дерева и предоставления сгенерированного API для внедрения логики для каждого из событий обхода. API генерируется на основе *"грамматики "*, которая определяет структуру предложения, отдельных фраз/слов и конкретных символов. Однако данная библиотека написана давно, вследствие чего в ней отсутствует поддержка современных языков программирования и требуются дополнительные библиотеки во время выполнения.

Следующим распространенным форматом синтаксического дерева является Babel AST. К минусам данного формата можно отметить отсутствие поддержки языка программирования Си и большое количество передаваемых данных.

Исходя из выявленных недостатков рассмотренных решений принято решение разрабатывать синтаксическое дерево с собственным форматом.

Разработанное синтаксическое дерево характеризуется следующими этапами:

- Лексический анализ входной программы. Данный процесс характеризуется аналитическим разбором входной последовательности символов на распознанные группы – лексемы – с целью получения на выходе идентифицированных последовательностей, называемых «токенами» (подобно группировке букв в словах).
- Синтаксический анализ или синтаксический разбор – процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой.
- К полученной структуре применяется семантический анализ, процесс выявления «смысла» синтаксических конструкций. Примером является связывание идентификатора с его объявлением, типом данных, определение результирующего типа данных выражения.
- Далее производится преобразование в промежуточный код (в нашем случае это код в формате JSON).

Рассмотрим простой пример преобразования в синтаксическое дерево.

На вход отправляем JSON посылку с последовательным кодом:


```
{
"code": "int main()n {n for(int i=0; i<=4; i++)n{n printf("Текст"); n} n return 0;n}",
"language": "C"
}
```


Где code – последовательный код, предназначенный для параллелизации; language – язык программирования, на котором написан код.

Получаем сформированное синтаксическое дерево:


```
{
"tree": [
{
"type": "Program"
},
{
"type": "FunctionDeclaration",
"typeVar": "int",
"id": "main",
"params": [],
"body": [
{
"type": "ForStatement",
"init": [
{
"type": "VariableDeclaration",
"id": "i",
"value": "0",
"variable": "int"
},
{
"type": "CompareExpression",
"id": "i",
"mark": "<=",
"number": "4"
},
```

```
{
"type": "UpdateExpression",
"id": "i",
"operator": "++",
"prefix": "false"
}
],
"body": [
{
"type": "StringFunctionDeclared",
"value": "Текст"
}
]
},
{
"type": "ReturnStatement",
"argument": {
"type": "Literal",
"value": "0"
}
}
]
}
]
}
```


Описание используемых полей:

- tree – корневое дерево проекта;
- type – тип описываемого блока;
- typeVar – тип данных;
- id – название идентификатора;
- params – параметры описываемого объекта;
- body – тело описываемого блока;
- init – инициализируемые параметры;
- value – значение заданного параметра;
- variable – тип переменной;
- mark – значение знака;
- number – значение числа;
- operator – значение оператора;
- prefix – использование префикса;
- argument – значение аргумента.

Заключение

В данной статье проанализированы существующие реализации трансляторов последовательного кода в параллельный код, представлена архитектура принципиально нового транслятора, дано описание синтаксического дерева. Также продемонстрирован пример формирования синтаксического дерева, построение которого обеспечивается выполнением лексического, синтаксического и семантического анализов.

В будущем планируется расширить структурное представление синтаксического дерева,

дополнить сервис преобразования кода модулями преобразования последовательного кода в параллельный код с применением нейросетевого подхода [11,12].

Библиография

1. P. Czarnul, J. Proficz and K. Drypczewski, "Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems," Scientific Programming, vol. 2020, pp. 1058–9244, 2020.
2. D. B. Changdao, I. Firmansyah and Y. Yamaguchi, "FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method," in Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th Int. Symp., ARC 2020, Toledo, Spain, Springer Nature. vol. 12083, pp. 232, 2020.
3. D. Wang and F. Yuan, "High-performance computing for earth system modeling," High Performance Computing for Geospatial Applications, vol. 23, pp. 175–184, 2020.
4. M. U. Ashraf, F. A. Eassa, A. Ahmad and A. Algarni, "Empirical investigation: Performance and power-consumption based dual-level model for exascale computing systems," IET Software, vol. 14, no. 4, pp. 319–327, 2020.
5. B. Brandon, "Message passing interface (mpi)," in Workshop: High Performance Computing on Stampede, Cornell University Center for Advanced Computing (CAC), vol. 262, 2015.
6. A. Podobas and S. Karlsson, "Towards unifying openmp under the task-parallel paradigm," in Int. Workshop on OpenMP. Springer International Publishing, Springer International Publishing, Nara, Japan, vol. 9903, pp. 116–129, 2016.
7. M. U. Ashraf, F. Fouz and F. A. Eassa, "Empirical analysis of hpc using different programming models," International Journal of Modern Education & Computer Science, vol. 8, no. 6, pp. 27–34, 2016.
8. J. A. Herdman, W. P. Gaudin, S. Smith, M. Boulton, D. A. Beckingsale et al., "Accelerating hydrocodes with openacc, opencl and cuda," High Performance Computing, Networking, Storage and Analysis (SCC), vol. 66, pp. 465–471, 2012.
9. H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang et al., "High performance computing using mpi and openmp on multi-core parallel systems," Parallel Computing, vol. 37, no. 9, pp. 562–575, 2011.
10. M. Marangoni and T. Wischgoll, "Togpu: Automatic source transformation from c++ to cuda using clang/llvm," Electronic Imaging, vol. 1, pp. 1–9, 2016.
11. X. Xie, B. Chen, L. Zou, Y. Liu, W. Le et al., "Automatic loop summarization via path dependency analysis," IEEE Transactions on Software Engineering, vol. 45, no. 6, pp. 537–557, 2017.
12. M. S. Ahmed, M. A. Belarbi, S. Mahmoudi, G. Belalem and P. Manneback, "Multimedia processing using deep learning technologies, high-performance computing cloud resources, and big data volumes," Concurrency and Computation: Practice and Experience, vol. 32, no. 17, pp. 56–99, 2020

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Предмет исследования – разработка синтаксического дерева для автоматизированного транслятора последовательного программного кода в параллельный код для

многоядерных процессоров.

Методология исследования основана на теоретическом подходе с применением методов анализа, моделирования, алгоритмизации, схематизации, обобщения, сравнения, синтеза.

Актуальность исследования определяется широким распространением технологий параллельных вычислений, необходимостью разработки соответствующих программных систем и вычислительных методов, включая разработку синтаксического дерева для автоматизированного транслятора для многоядерных процессоров.

Научная новизна связана с предложенной автором архитектурой транслятора нового транслятора по переводу последовательного кода в параллельный, описанием синтаксического дерева, построение которого обеспечивается выполнением лексического, синтаксического и семантического анализов.

Статья написана русским литературным языком. Стиль изложения научный.

Структура рукописи включает следующие разделы: Введение (увеличение числа мультипроцессорных комплексов, решение вычислительно сложных задач за приемлемое время, разработка программных модулей с применением многопоточного программирования, участие аналитика, программиста, специалиста по тестированию, специализированный язык Cilk, точка синхронизации, стиль "разделяй и властвуй" (divide and conquer), или fork-join parallelism, специализированные библиотеки Java Fork Join, вопрос по автоматизированному (или автоматическому) преобразованию последовательного кода в параллельный, автоматизированный транслятор, диалог с программистом, оптимизация алгоритма по использованию аппаратных ресурсов и по критерию быстродействия, известные программные продукты Polaris, Cetus, Pluto, Polly, Par4All, CAPO, WPP, SUIF, VAST/PARALLEL, OSCAR, САПФОР, сравнительный анализ н примере Parawise, алгоритм работы CAPO), Архитектура транслятора (решение сделать веб-приложение, микросервисная архитектура транслятора, сервис обработки запросов, сервис построения дерева, сервисы формирования параллельной программы, сервис преобразования кода, диаграмма последовательностей создания параллельной программ, использованием брокера сообщений), Построение синтаксического дерева (форматы синтаксического дерева, ANTLR4, Babel AST, собственный формат, лексический анализ входной программы, синтаксический анализ, семантический анализ, преобразование в промежуточный код в формате JSON, пример преобразования в синтаксическое дерево, описание используемых полей), Заключение, Библиография.

Текст шесть рисунков. Для рисунков 5, 6 необходимо упоминание в предшествующем тексте.

Содержание в целом соответствует названию. Автором проанализированы известные реализации трансляторов последовательного кода в параллельный, представлена оригинальная архитектура транслятора, дано описание синтаксического дерева. Определены перспективы работы, связанные с применением нейросетевого подхода.

Библиография включает 12 источников зарубежных авторов – главы монографий, научные статьи, материалы научных мероприятий и пр. Библиографические описания некоторых источников требуют корректировки в соответствии с ГОСТ и требованиями

редакции, например:

1. Czarnul P. , Proficz J. , Drypczewski K. Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems // Scientific Programming. 2020. Vol. 2020. P. 1058–9244.

2. Changdao D. B., Firmansyah I., Yamaguchi Y. FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method // Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th Int. Symp. ARC 2020. Toledo, Spain : Springer Nature, 2020. Vol. 12083. P. 232.

4. Ashraf M. U., Eassa F. A, Ahmad A., Algarni A. Empirical investigation: Performance and power-consumption based dual-level model for exascale computing systems // IET Software, 2020. Vol. 14. № 4. P. 319–327.

Обращает внимание отсутствие ссылок на работы отечественных специалистов.

Апелляция к оппонентам (P. Czarnul, J. Proficz, K. Drypczewski, D. B. Changdao, I. Firmansyah, Y. Yamaguchi, D. Wang, F. Yuan, M. U. Ashraf, F. A. Eassa, A. Ahmad, A. Algarni, B. Brandon, A. Podobas, S. Karlsson, M. U. Ashraf, F. Fouz, F. A. Eassa, J. A. Herdman, W. P. Gaudin, S. Smith, M. Boulton, D. A. Beckingsale, H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, M. Marangoni, T. Wischgoll, X. Xie, B. Chen, L. Zou, Y. Liu, W. Le, M. S. Ahmed, M. A. Belarbi, S. Mahmoudi, G. Belalem, P. Manneback и др.) имеет место.

В целом материал представляет интерес для читательской аудитории и после доработки может быть опубликован в журнале «Программные системы и вычислительные методы».

Программные системы и вычислительные методы
Правильная ссылка на статью:
Дагаев Д.В. — Ограничительная семантика языка в системе МультиОберон // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.36217 EDN: IWIODR URL: https://nbpublish.com/library_read_article.php?id=36217

Ограничительная семантика языка в системе МультиОберон

Дагаев Дмитрий Викторович

Главный Эксперт, АО "РАСУ"

115230, Россия, г. Москва, шоссе Каширское, 3, корп. 2, стр.16

dvdagaev@oberon.org

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2023.1.36217

EDN:

IWIODR

Дата направления статьи в редакцию:

03-08-2021

Аннотация: Язык и системы на базе Оберон в реализации демонстрируют минималистский подход для достижения надежности, существенного отличающийся от большинства программных систем, стремящихся к максимизации числа поддерживаемых функций. Требования к критически важным системам для АЭС категории А запрещают использование еще большего числа практик программирования. Для соответствия требованию категории А стабильного числа итераций устанавливается запрет использования операторов цикла по условию. Для обеспечения эргодичности используется запрет использования динамической памяти и рекурсии. Уязвимость типа переполнения буфера закрывается запрещением модуля системных операций SYSTEM. Ограничения могут быть установлены на выявление проблемы хрупкого базового класса, операции смены типа и использование вложенных процедур. Отмечено, что переход к диалекту Оберон-07 в основном коснулся дополнительных ограничений и хорошо встраивается в рамки ограничительной семантики. Вместо языков и диалектов под каждый набор требований автором предложен подход ограничительной семантики, при котором используется один язык с системой ограничений. В язык введен один оператор RESTRICT как декларация ограничений на данный модуль. Реализован компилятор МультиОберон с одним фронтендом, включающем систему ограничений, и несколькими сменными бэкендами. Продемонстрирован синтаксический анализ

компилятора на примерах. Показана стратегия масштабирования компилятора в зависимости от системы требований. Новизной подхода ограничительной семантики является достижение набора минимально необходимых свойств, соответствующих требованиям к системе. Использование разработчиками систем подхода «от ограничений» является преимуществом, т.к. декларирует действительно необходимые свойства системы, увязанные с требованиями.

Ключевые слова:

Оберон, Компонентный Паскаль, эргодичность, надежность ПО, ограничение, компилятор, синтаксическое дерево, семантический анализ, модульность, хрупкий базовый класс

Введение

Язык программирования и система Оберон [1] реализовали крайне минималистский подход, при котором из ПО исключались все функции, не являющиеся жизненно необходимыми. Этот вектор движения сохранился во всех последующих реализациях Оберонов, что является их существенным отличием от общепринятого подхода к построению программных систем, при котором объем заявленных функций является необходимой коммерческой особенностью каждого нового продукта.

Не только проект Оберон развивался в сторону уменьшения функциональности. Аналогичные подходы применяются в формировании требований к системам для АЭС, выполняющим функции категории А [2]. Это – наивысшая категория в части критически важного ПО, применяемая для систем защит реакторов. ПО категории А должно гарантировать не наличие, а отсутствие многих стандартных решений, применяемых программистами. В данном ПО необходимо отсутствие стандартной ОС и библиотек сторонних разработчиков, за исключением сертифицированных для категории А. Требование неизменности времени прогона вне зависимости от входных данных приводит к исключению циклов с переменным числом итераций и выходом по условию. Требование защиты содержимого памяти приводит к запрету выделения динамической памяти во время работы и контроля. Известные реализации ПО категории А используют специализированные под данные требования языки и компиляторы, например, язык ΨС [3].

Описанные подходы демонстрируют, что требования влияют существенным образом на программную реализацию систем. При ужесточении требований может возникнуть проблема структурных изменений в ПО. Более того, при очень жестких требованиях может измениться и язык программирования реализации, и компилятор. Это, естественно, приведет не к доработке существующей, а к разработке новой системы.

Автором предложен подход **ограничительной семантики**, который состоит в следующем: **вместо нескольких языков и компиляторов использовать один язык и фронтенд компилятора с системой семантических ограничений**.

Ниже данный подход будет детально описан, равно как и программная реализация на языке Оберон в системе МультиОберон.

Все программные примеры будут представлены в Паскале-подобном синтаксисе, данная разработка продолжает традиции школы Никлауса Вирта в части разработки языков

Паскаль/Модула/Оберон.

Оператор RESTRICT как декларирование ограничений

Механизм ограничительной семантики очень прост – используется только оператор RESTRICT, декларирующий систему ограничений для данного программного модуля.

RESTRICT -, -*, +, ..;

где - представляет собой некоторую лексему, которую компилятор интерпретирует как дополнительное ограничение для данного модуля (знак '-' означает исключить лексему). В свою очередь, + представляет собой включение имеющейся функциональности, которая исключена по умолчанию. Область действия ограничения – программный модуль. Оберон является модульным языком программирования, где модуль находится в одном файле и является единицей компиляции, загрузки и выполнения. Следовательно, ограничения распространяются только на данный программный модуль. Если необходимо систему ограничений распространять на несколько программных модулей, то создается файл ограничений с внешней областью видимости (знак '*'), который необходимо импортировать в каждый программный модуль. Например, оператором IMPORT RestrictIEC880 импортируются все ограничения, записанные в модуле RestrictIEC880 как внешние. Например, если в нем записаны RESTRICT –WHILE*, -LOOP*, это запрети использование операторов WHILE, LOOP.

При несоответствии кода модуля системе ограничений не должен проходить семантический анализ, и, как следствие, выдаваться ошибка компиляции.

Ограничение числа итераций цикла

Допустим, к разрабатываемой прикладной системе реального времени предъявлено требование ограничения числа итераций цикла. В стандарте [2] это сформулировано как «время прогона не должно существенно изменяться в результате изменения входных данных». Это означает, что в тексте разрабатываемой программы должны быть только циклы FOR с постоянным числом итераций. Следовательно, откомпилированный бинарный код данного модуля также будет с фиксированным числом итераций, вне зависимости от конфигурации или условий запуска данного кода. Ограничительная семантика в части постоянных итераций цикла будет означать запрет остальных типов циклов.

RESTRICT –WHILE, -REPEAT, -UNTIL, -LOOP;

В результате останется только цикл FOR j := 1 TO N-1 DO .. END, который имеет постоянное число итераций N, вне зависимости от внешних условий. Но даже для цикла FOR есть ряд случаев, когда от внешних условий может зависеть параметр N. Это может привести к варьированию времени прогона. В таком случае следует ограничить использование пределов цикла FOR константами, т.е. запретить переменные для пределов.

RESTRICT –FOR(VAR);

Обработка данного ограничения компилятором требует уровня семантического анализа. В отличие от запрещения ключевого слова (например, WHILE) целиком, исключение пределов цикла-переменных приводит к необходимости анализа синтаксического дерева [4] с учетом семантической информации о типах. Рисунок 1 иллюстрирует дерево для проведения подобного анализа.

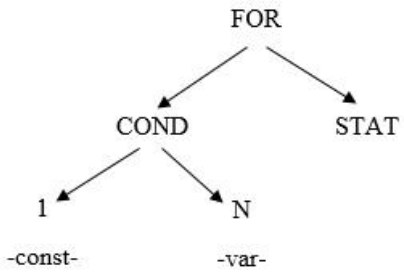


Рис. 1. Абстрактное синтаксическое дерево цикла FOR

Fig.1 Abstract Syntax Tree of FOR cycle

При установленном ограничении –FOR(VAR) компилятор должен активировать проверки токена предела FOR->COND->N данным ограничением. Поскольку токен предела на рисунке является переменной, то проверка завершается неудачей, и компилятор выдает сообщение об ошибке.

Время прогона модуля может зависеть также от времени прогона процедур других, импортируемых модулей. Например, может быть такой вызов процедуры OtherModule.LongCalculations(). Это означает, что в модуле OtherModule требования ограничения числа итераций циклов могут не выполняться. Следовательно, аналогичным образом необходимо рассматривать и модуль OtherModule. Если семантические ограничения на модуль не установлены, то отсутствует подтвержденная декларация о выполнении требований.

Ограничение динамической памяти

В работе [5] показано, как свойства компьютерной системы в части памяти могут со временем деградировать, и что важным критерием является эргодичность, т.е. стабильность этих свойств, что требует доказательств, в том числе и архитектурного характера. Там же показано, что нужно исходить из презумпции неэргодичности, т.е. предполагать, что характеристики разрабатываемого ПО могут деградировать со временем. Система семантических ограничений служит хорошим инструментом такого доказательства эргодичности.

Для многих систем реального времени нужен запрет выделения динамической памяти. Внутри модуля это осуществляется ограничением встроенной функции NEW.

```
RESTRICT –NEW;
```

Это означает, что в данном модуле непосредственно функция NEW быть вызвана не может. При этом аллокирование динамической памяти может быть вызвано функцией из другого, импортируемого модуля, например, фабричного Factory.New(). Помимо рассмотрения ограничений импортируемого модуля, можно запретить использование указателей.

```
RESTRICT –NEW, -POINTER;
```

Тогда проверка не будет пройдена и завершится синтаксической ошибкой.

```
VAR ptr: POINTER TO FactoryElem;

ptr := Factory.NewElem;
```

Следующим способом обойти ограничение будет использование библиотечных функций malloc, calloc, strdup и т.д. Данные функции возвращают безтэговый (untagged) указатель на память, который может потом преобразовываться в используемый в Обероне указатель с помощью модуля SYSTEM. Для исключения такого рода указателей требует запрета использования модуля SYSTEM.

RESTRICT –NEW, -POINTER, -SYSTEM;

Итоговое условие ограничения динамической памяти модуля будет иметь все три составляющие.

Ограничение стековой памяти

В Оберон-системах стековая память фиксируется для каждого процесса и не может быть увеличена впоследствии. Выход за пределы стековой памяти может происходить при глубокой рекурсии в используемом ПО. Следствием выхода за пределы является фатальная ошибка памяти. Этого нужно избежать. Запрет использования рекурсии выглядит следующим образом.

RESTRICT –PROCEDURE(PROCEDURE);

При установленном ограничении -PROCEDURE(PROCEDURE) производится анали синтаксического дерева на предмет выявления рекурсий. Выявленные случаи приводят к синтаксической ошибке.

Способом обойти эти ограничения может стать принудительная установка указателя стека за счет регистровых операций, например, SYSTEM.PUTREG(SP, address). Во избежание этого нужно запретить функцию PUTREG, работающую непосредственно с регистрами.

Другим способом обойти ограничение будет использование библиотечной функции alloca или аналогичных. Способом борьбы также будет запрещение SYSTEM.

RESTRICT –PROCEDURE(PROCEDURE), -SYSTEM;

Использование гарантий ограничений стековой памяти имеет практическую пользу. Автор сталкивался с ситуацией, когда нужно было организовать быструю сортировку динамически поступающих сигналов в системе мягкого реального времени. Была обнаружена редкая ситуация, что при пиковом потоке сигналов программа быстрой сортировки аварийно завершилась из-за выхода за границы стековой памяти. В результате был найден и реализован алгоритм не рекурсивной быстрой сортировки [6], который в последующем показал свою стабильную работу. Если бы имелись разработанные 2 модуля, один – без ограничений, а второй – с запретом рекурсии, то выбор был бы однозначен в пользу варианта 2.

Атаки на переполнение буфера

Атака на переполнение буфера является атакой на самую распространенную уязвимость в области безопасности ПО [7]. Авторы [7] рекомендуют «обратить внимание на языки, обеспечивающих проверку и сохранение типов, как в Java, исключающие переполнение буфера. Однако, не следует забывать, что виртуальная машина Java написана на C и, таким образом, может иметь уязвимости».

Оберон представляет собой статически жестко типизированную систему. Исполняющая

среда Оберона написана на Обероне, который является статически типизированным языком. Поэтому на чистом Обероне атака на переполнение буфера невозможна, поскольку операции копирования производятся между жестко типизированными объектами с фиксированной длиной.

Однако есть модуль SYSTEM,который декларирует использование низкоуровневых системных операций. Например, при его использовании можно копировать, используя адреса и длины.

SYSTEM.MOVE(srcAddr, dstAddr, length);

Кроме этого, процедурами SYSTEM.PUT(addr, value) можно записывать информацию по произвольному адресу, например, изменять адрес возврата или указатель на функцию. Есть еще способ – получить адрес стека и работать с ним как с массивом указателей.

rawAddr := SYSTEM.ADR(localVar); rawAdr[16 (* offset *)] := x;

Способом борьбы с уязвимостями в виде переполнения буфера будет, как минимум, запрещение указанных функций.

RESTRICT -MOVE, -PUT, -ADR;

Более точным решением будет запрещение модуля SYSTEM.

RESTRICT -SYSTEM;

Следует заметить, что использование команд из SYSTEM в стандартном Обероне требует явной декларации импортирования этого модуля.

IMPORT SYSTEM;

Такая декларация явно указывает на необходимость использования в разрабатываемом программном модуле потенциально небезопасных функций (в смысле безопасности типов). Если разработчик указал импорт SYSTEM, то он декларирует потенциально уязвимые места. Если при этом имеется ограничение использования SYSTEM (например, путем импорта модуля с RESTRICT -SYSTEM), то это приводит к синтаксической ошибке при компиляции.

Разумеется, к данной уязвимости приведет использование функций стандартной библиотеки memcpy, strcpy, работающие с безтэговыми (untagged) указателями. Однако, работа с данными указателями возможна только при использовании модуля SYSTEM.

Проблема хрупкого базового класса

Проблема хрупкого базового класса [8] заключается в скрытых зависимостях при наследовании неабстрактного класса сторонними разработчиками. Рассмотрим такой базовый класс, имеющий реализацию метода inc2. Эта реализация наследуется, что определено ключевым словом EXTENSIBLE.

TYPE Base = EXTENSIBLE RECORD counter: INTEGER END;

PROCEDURE (VAR b: Base) inc2(), NEW, EXTENSIBLE;

BEGIN INC(b.counter) END inc2;

PROCEDURE (VAR b: Base) inc1(), NEW; BEGIN b.inc2 END inc1;

Производный класс, зная только интерфейс базового, может предложить свою реализацию inc2.

```
TYPE Ext = RECORD (B.Base) END;

PROCEDURE (VAR e: Ext) inc2(); BEGIN e.inc1 END inc2;
```

Такая внешне правильная конструкция приведет к появлению бесконечной рекурсии, inc1->inc2->inc1.

Проблема данного примера и хрупкого базового класса заключается в наследовании реализации. В Обероне (здесь и далее используется диалект Oberon-CP, иногда упоминающийся под коммерческим именем Компонентный Паскаль, англ. Component Pascal, https://blackboxframework.org/index.php?cID=home-ru,ru,) наследование реализации явно отмечается ключевым словом EXTENSIBLE. Если процедура EXTENSIBLE, то это указывает на наследование реализации. Использование EXTENSIBLE, равно как и SYSTEM – это декларация последующих нарушений нормального порядка вещей в программном коде. В других языках, например, Java, для этих целей было не совсем удачно введено ключевое слово final, отсутствие которого возникает при ненормальном проектировании системы классов.

Декларация запрещения наследования реализации выглядит как

```
RESTRICT -EXTENSIBLE;
```

Это обеспечивает отсутствие проблемы хрупкого базового класса для типов всего модуля. Такой удачный, по сравнению с Java, синтаксис Оберона позволяет просто запретить ключевое слово, а не выстраивать сложные семантические схемы проверок.

Проблема вложенных процедур

Вложенные процедуры располагаются внутри других процедур и имеют доступ к их локальным переменным. Проблема вложенных процедур [9] возникает при выходе из области действия родительского фрейма, что иллюстрирует следующий код.

```
VAR vProc = PROCEDURE(val: INTEGER);

PROCEDURE Parent();

VAR loc_val: INTEGER;

PROCEDURE Child(val: INTEGER);

BEGIN loc_val := val END Child;

BEGIN Child(12);

vProc := Child;

vProc(13) (* --- CRASH --- *)

END Parent;
```

Присваивание глобальной переменной vProc адреса процедуры Child не запрещено. Но активация этой процедуры vProc(13) приводит к ошибке выполнения из-за отсутствия корректного доступа к стеку процедуры Parent.

Способом решения данной проблемы является запрет использования локальных процедур, определяемый как

RESTRICT –BEGIN(PROCEDURE);

При таком запрете процедура примера выше переместится в глобальную область с прототипом PROCEDURE Child(val: INTEGER; VAR loc_val: INTEGER), в котором буд добавлен выходной параметр.

Проблема оператора WITH

Проблема оператора WITH заключается в том, что он одновременно реализует функциональность переключателя и смены типа. Это приводит к неоднозначности кода и неясности в механизмах проверки при компиляции. Рассмотрим следующий пример.

```
TYPE Base = POINTER TO ABSTRACT RECORD END;

Ext1 = POINTER TO RECORD (Base) END;

Ext2 = POINTER TO RECORD (Base) END;

VAR vBase: Base;

vExt2: Ext2;

WITH vBase: Ext1 DO (* vBase IS Ext1 *)

vBase := vExt2 (* vBase IS Ext2 ??? *)

END;

vBase := vExt2; (* vBase IS Ext2 – Possible? *)
```

В данном примере переменная vBase до входа в область WITH рассматривается как указатель на тип Base, а после входа рассматривается как указатель на тип Ext1. Поэтому присваивание вне области WITH допустимо, а внутри – нет.

Способом решения является запрет WITH.

RESTRICT –WITH;

При этом можно использовать оператор проверок типа IS и вводить локальные переменные типов Ext1, Ext2 с явным преобразованием типов ext1 := vBase(Ext1).

Расширяемость синтаксиса языка

Приводимые выше лексемы оператора RESTRICT работали в сторону ограничения функциональности. Однако данный оператор позволяет реактивировать отключенную по умолчанию функциональность.

Например, при переходе с 32 на 64 бита потребовалось вводить целочисленный тип размером в адрес, 4 байта для 32-битных сред и 8 байт для 64-битных. Компилятор был расширен типом ADRINT, но, поскольку данный тип не входил в сообщение о языке, он стал деактивирован по умолчанию. Реактивировать его можно операцией RESTRICT.

```
RESTRICT +ADRINT;

VAR ai: ADRINT;
```

ai := SYSTEM.VAL(ADRINT, Api.GetAddr());

Данное число сохраняет целочисленный адрес, полученный из указателя, возвращенного Api.GetAddr().

Языки программирования тоже эволюционируют со временем. Однако при переходе на новые версии происходит часто прекращение поддержки старых. Например, Питон 2.7 прекратили поддерживать с 2020 года (https://legacy.python.org/dev/peps/pep-0373/).

Вместо расширения и изменения версий синтаксиса языка предлагается с помощью механизма ограничительной семантики использовать поэтапное явно декларируемое расширение добавлением новых ключевых слов. Декларация RESTRICT +ADRINT добавляет новое ключевое слово и тип ADRINT. Расширение языка программирования за счет новых ключевых слов и понятий не может происходить бесконечно. Такой путь является тупиковым. Поэтому допускается расширение за счет очень ограниченного количества ключевых слов.

Динамика развития диалектов, Оберон-07/13/16

Оберон от автора развивался в динамике, были созданы диалекты Оберон от 2007, 2013 года и уточнения в версии 2016. Развитие шло главным образом в сторону исключения не являющейся необходимой функциональности [10].

Основные изменения в языке сведены автором в таблицу на рисунке 2.

1	Изменены синтаксис/семантика WHILE, который поднят до цикла Дейкстры добавлением ELSE	+ELSE(WHILE)
2	Цикл LOOP исключен	-LOOP, -EXIT
3	Исключены типы SHORTINT, LONGINT, SHORTREAL	-SHORTINT, -LONGINT, -SHORTREAL
4	Исключен переключатель WITH	-WITH
5	Изменен синтаксис переключателя CASE	Изменение CASE
6	Оператор RETURN может быть только в конце функции, возвращающей значение	-RETURN(PROCEDURE)
7	Запрет передачи структурированных параметров по значению	- «Структура по значению»
8	Исключение возможности импортировать экспортируемые переменные по записи	- «Импорт по записи»
9	Исключено понятие указателей на массивы	-POINTER(ARRAY)
10	Отсутствие методов, относящихся к объектам, введенным в диалекте Оберон-КП	-ABSTRACT, -EXTENSIBLE
11	Добавлен процедурный тип для обработки прерываний	Изменение PROCEDURE
12	Исключены вложенные процедуры	-BEGIN(PROCEDURE)

Рис. 2. Изменения Н.Вирта в языке Оберон

Fig 2. N.Wirths changes in Oberon language

Анализ приведенных изменений показывает, что 9 из 12 изменений представляют собой запреты использования в языке некоторых синтаксических и семантических конструкций. Например, для выполнения п.6 (оператор RETURN может быть только в конце функции, возвращающей значение) необходимо добавить семантическое ограничение.

RESTRICT –RETURN(PROCEDURE);

Это позволит сделать соответствующую проверку на узлах синтаксического дерева.

В то же время 3 из 12 изменений несут за собой изменение синтаксиса операторов ELSE(WHILE), CASE, PROCEDURE. Изменение оператора CASE связанно с тем, ч операция проверки типа переходит от оператора WITH к оператору CASE, в то же время преобразование типов исключается вместе с оператором WITH. Оставшиеся изменения вносят дополнительную функциональность. Для данных изменений следует определить лексемы и предусмотреть их активацию в компиляторе, возможный вариант приведен.

RESTRICT +ELSE(WHILE), +CASE(POINTER), +PROCEDURE(SYSTEM);

Если есть поддержка новых измененных синтаксических конструкций, то можно выстраивать компилятор для диалекта 07/13/16.

По результатам оценки внесенных в Оберон 07/13/16 изменений можно сделать вывод об исключении потенциально проблемных мест. Среди них – WITH с преобразованием типа переменной, цикл LOOP с неявно определенным условием выхода, исключение неоднозначности при приведении целочисленных и действительных типов, возврат значения в конце процедуры и т.д. Исключение проблемных мест минимизирует возможность написания проблемного кода, что, в свою очередь, даст возможность утверждать о повышении надежности ПО.

Особенности реализации компилятора

Заложенные в систему свойства расширяемости и ограничиваемости диктуют структуру реализации компилятора, удовлетворяющего указанным требованиям. В условиях такой вариативности наилучшим представляется решение с разделением фронтенда и бэкенда[11]. Фронтенд занимается построением синтаксического дерева, бэкенд выполняет кодогенерацию на целевую платформу.

Предложенный во введении подход с ограничительной семантикой предлагает использовать один язык и компилятор вместо нескольких. При этом необходимо отметить, что для различных сред могут быть различные, и отличающиеся в существенной части бэкенды. Бинарные коды для встроенного ПО на ARM будут существенно отличаться от ПО для Windows, а байт-код LLVM будет отличаться от байт-кода Java Virtual Machine.

В части реализации предложенного подхода был разработан компилятор МультиОберон с набором сменных бэкендов.

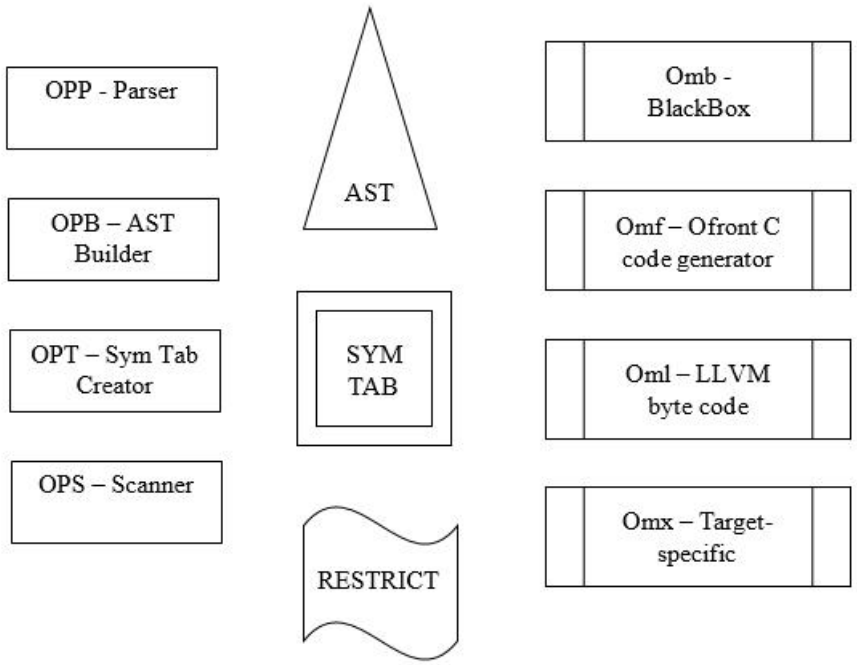


Рис. 3. Мульти-компилятор с набором бэкендов

Fig.3. Multi-compiler with a set of backends

Концепция универсального фронтенда для разработки портируемого компилятора OP2 была предложена в [12] и получила несколько программных реализаций.

Реализация диалекта Oberon-CP в виде системы, построенной на основе компонентного ПО была осуществлена в системе BlackBox Component Builder[13]. Компилятор указанной системы был переработан и подключен в МультиОберон в качестве сменного бэкенда Omb. Построенный на основе OP2 фронтенд системы BlackBox был переработан и лег в основу фронтенда МультиОберона.

Другой компилятор на основе OP2 Ofront (https://github.com/jtempl/ofront) создает на выходе код на языке C. Он уточняет систему проверок корректности типов указателей по сравнению с базовой в OP2 [14]. Ofront был существенно доработан и подключен в систему в качестве бэкенда Omf.

Бэкенд Oml, имеющий на выходе байт-код LLVM, был полностью разработан автором для системы МультиОберон.

Предполагается возможность расширения бэкендами для целевых платформ.

На выходе фронтенда создается синтаксическое дерево AST, таблица символов Sym Tab и набор использованных ограничений RESTRICT. Набор использованных ограничений должен соответствовать возможностям бэкенда. Если такого соответствия нет, то выдается ошибка при компиляции.

Вся система МультиОберон со сменными бэкендами доступна по https://github.com/dvdagaev/Mob . Текущая версия 1.2 поддерживает архитектуры X86, X64, ARM32, Aarch64 для Linux, Windows, Raspberry Pi OS.

Пример работы компилятора

Пример анализа компилятором с установленными ограничениями из графической системы BlackBox приведен на рисунке 4.

```
RESTRICT -PROCEDURE(PROCEDURE), -RETURN(PROCEDURE), -BEGIN(PROCEDURE);

PROCEDURE Fact (n: INTEGER): INTEGER;
   VAR x1234: INTEGER; nested procedure is not allowed
   PROCEDURE One(): INTEGER;
   BEGIN
      RETURN 1
   END One;
BEGIN
   IF n <= 1 THEN RETURN One() END; unexpected RETURN position
   x1234 := n * Fact(n-1 recursion is not allowed );
   RETURN x1234
END Fact;
```

Рис. 4. Листинг с нарушениями ограничений

Fig.4. Listing with restrictions violations

В данном листинге видно, что, несмотря на установленное ограничение на использование вложенных процедур –BEGIN(PROCEDURE), процедура One находится внутри родительской Fact. Следствием этого является синтаксическая ошибка «nested procedure is not allowed». Кроме этого, несмотря на установленное ограничение –RETURN(PROCEDURE), в функции Fact реализован возврат из середины кода. Следствием этого является синтаксическая ошибка «unexpected RETURN position». И, наконец, несмотря на ограничение использования рекурсии –PROCEDURE(PROCEDURE), функция Fact разработана рекурсивной. Следствием этого является синтаксическая ошибка «recursion is not allowed».

Стратегия масштабирования/демасштабирования МультиОберона

Предложенная реализация системы МультиОберон на данном этапе представляет собой эталонный уровень в части функциональности. При этом возможно как и уменьшение функциональности за счет дополнительных ограничений, так и лимитированное увеличение функциональности. Рисунок 5 иллюстрирует направления движения развития.

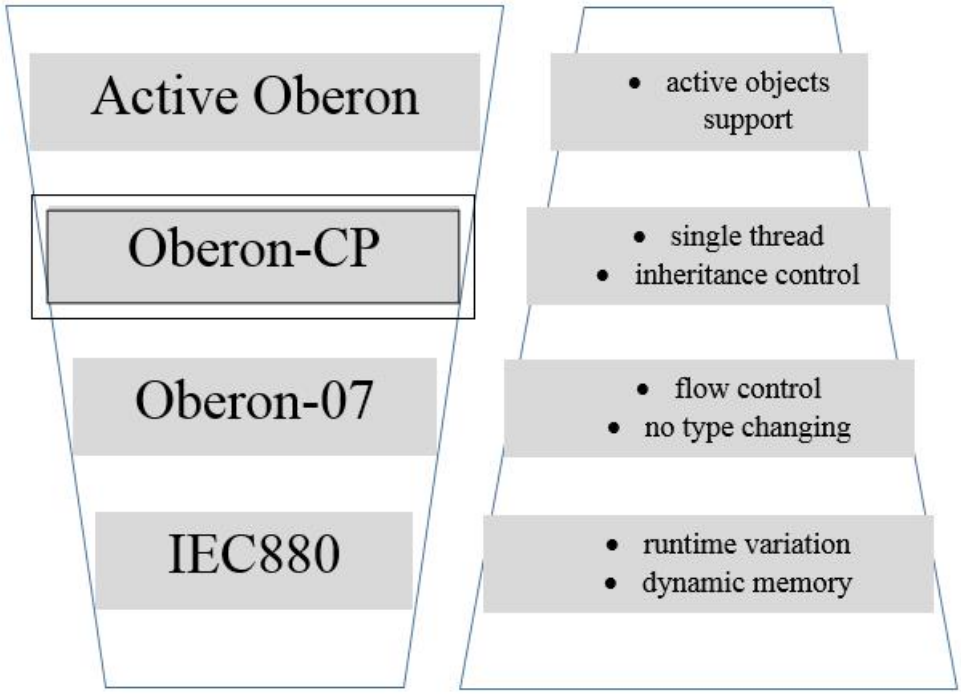


Рис. 5. Масштабирование/демасштабирование МультиОберона

Fig.5. MultiOberon scaling up and down

Oberon-CP представляет собой эталонный уровень, не требующий установки ограничений. При переходе к функциональности Оберона-07 необходимо использовать систему ограничений, при этом получается более определенная картина в управлении типами и потоком программы (flow control, no type changing). Введение дополнительных ограничений к Оберону 07 для соответствия стандарту IEC880 потребует запрета динамической памяти и вариативности выполнения циклов (runtime variation, dynamic memory). Это будет означать демасштабирование. При демасштабировании (на рисунке – движение вниз) число ограничений растет, а число функций уменьшается. При этом уменьшается число уязвимостей и увеличивается надежность.

Масштабирование от эталонного уровня означает явное введение дополнительных функций, например, введения активных объектов языка Active Oberon [15]. Используемый бэкенд должен поддерживать данные дополнительно вводимые функции.

Система ограничений языка ADA

При проектировании системы ограничений учитывался опыт компилятора GNAT языка Ada в создании ограничений (https://docs.adacore.com/gnat_rm-docs/html/gnat_rm/gnat_rm/standard_and_implementation_defined_restrictions.html). Подход Ada акцентирован на использование ряда семантических проверок (рекурсия, выделение динамической памяти) и большого числа настроек исполняющей системы. Высокая надежность Ada достигается за счет избыточности программных решений.

Подход МультиОберона нацелен на расширение и демасштабирование возможностей самого языка Оберон. Высокая надежность Оберона достигается за счет минимализма используемых программных решений.

Заключение

Предложенный в данной статье подход ограничительной семантики позволяет выбирать набор минимально необходимых средств для решения задач с разной шкалой требований. Для критически-важных систем и требований необходимо разрабатывать код с максимальной системой ограничений. Для задач реального времени необходимо отказаться от решений, потенциально блокирующих поток выполнения.

Гарантии выполнения требований ограничительной семантики предоставляются на основе синтаксического и семантического анализа программ при разработке, а не последующими экспертными средствами анализа кода на уязвимость. Соответствия модуля системе ограничений является необходимым условием успешной компиляции программы.

Подход «от ограничений» предполагает декларирование разработчикам соответствия необходимым требованиям проектирования. И наоборот, отказ от ограничений подразумевает несоответствие в полном объеме системе требований.

При построении эргодичных систем отказ от ограничений означает отказ от доказательства эргодичности. Такая практика не является общепринятой. Однако, учитывая презумпцию неэргодичности, постулируется, что характеристики разрабатываемого ПО деградируют со временем. И если в ПО нет ограничений за использованием динамической памяти, то она будет использоваться по худшему сценарию из тех, что позволит ядро ОС и система своппинга.

Общее правило построения: чем больше ограничений на использование ПО, тем данное ПО будет более надежно и предсказуемо.

Библиография

1. Н.Вирт, Ю.Гуткнехт. Разработка операционной системы и компилятора. Проект Оберон. ДМК-Пресс, 2015.
2. ГОСТ Р МЭК 60880, Программное обеспечение компьютерных систем, выполняющих функции категории А, 2009 / GOST R IEC 60880, Software for computer systems performing category A functions, 2009 (in Russian).
3. S. Louise, M. Lemerre, C. Aussagues and V. David. The OASIS Kernel: A Framework for High Dependability Real-Time Systems. In Proc. of the IEEE 13th International Symposium on High-Assurance Systems Engineering, 2011, pp. 95-103.
4. А.Ахо, М.Лам, Р.Сети, Д.Ульман. Компиляторы: принципы, технологии и инструментарий, второе издание, 2008, с. 137-144.
5. Дагаев Д.В. О разработке Оберон-системы с заданными свойствами эргодичности. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 67-78. DOI: 10.15514/ISPRAS–2020–32(6)–5
6. Вирт Н., Алгоритмы и структуры данных. ДМК-Пресс, 2016.
7. В. Н. Гугнин, Д. В. Сотник. Атака с использованием переполнения буфера. Вестник Нац. техн. ун-та "ХПИ" : сб. науч. тр. Темат. вып. : Информатика и моделирование. – Харьков : НТУ "ХПИ". – 2004. – № 34. – С. 52-57.
8. Ермаков И.Е. Объектно-ориентированное программирование: прояснение принципов? //Объектные системы — 2010: Материалы I Международной научно-практической конференции — г. Ростов-на-Дону, Южно-Российский ГТУ — 2010. С. 130-135
9. Keller R. Improved Stackmanagement in Active Oberon Kernel / Master Thesis, ETH, march 2006, pp. 40-41.

10. Wirth N. The Programming Language Oberon. Revision 1.10.2013 / 3.5.2016. – pp. 1-17.
11. Вирт Н., Построение компиляторов, ДМК-Пресс, 2016.
12. Crelier R. OP2: A portable Oberon Compiler / ETH Zurich, Department Informatik, 1990, pp. 4-19.
13. Szyperski C. Component Software: Beyond Object-Oriented Programming / 01/2002, 2nd edition, Addison Wesley, ISBN: 0-201-745572-0
14. Templ J., Metaprogramming in Oberon, diss. ETH No 10655, 1994, pp. 120-121
15. Pieter J. Muller, The Active Object System Design and Multiprocessor Implementation. Diss. ETH No.14755, for the degree of Doctor of Technical Sciences, ETH Zurich 2002, 197 p.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемый материал посвящен совершенствованию конструкции языка программирования путем реализации минималистского подхода, при котором из программного обеспечения исключаются функции, не являющиеся жизненно необходимыми. Это может быть актуальным для критически важного программного обеспечения, например, при защите реакторов атомных электростанций.
Методология исследования базируется на обобщении современных научных публикаций зарубежных о российских авторов по разрабатываемой теме с учетом государственных стандартов, регламентирующих требования к программному обеспечению компьютерных систем, выполняющих функции категории А.
Научная новизна результатов представленного исследования, по мнению рецензента, состоит в предложенном подходе к ограничительной семантике, в котором вместо нескольких языков и компиляторов предлагается использовать один язык и фронтенд компилятора с системой семантических ограничений.
В статье выделены следующие разделы: Введение; Оператор RESTRICT как декларирование ограничений: Ограничение числа итераций цикла; Ограничение динамической памяти; Ограничение стековой памяти; Атаки на переполнение буфера; Проблема хрупкого базового класса; Проблема вложенных процедур; Проблема оператора WITH; Расширяемость синтаксиса языка; Динамика развития диалектов, Оберон-07/13/16; Особенности реализации компилятора; Пример работы компилятора; Стратегия масштабирования/демасштабирования МультиОберона; Система ограничений языка ADA; Заключение; Библиография. Во введении кратко изложена необходимость и суть проводимого исследования. В последующих разделах, наименования которых ёмко отражают их содержание, излагаются способы осуществления конкретных ограничений с целью сужения функционала для обеспечения надежности и безопасности программного обеспечения в соответствии с требованиями к компьютерным системам, выполняющим функции категории А. Повествование сопровождается иллюстрациями программных реализаций на языке Оберон в системе МультиОберон в Паскале-подобном синтаксисе.
В Заключении представлены преимущества предложенного подхода, состоящие, по мнению авторов, в возможности выбирать набор минимально необходимых средств для решения задач с разной шкалой требований: для критически важных систем и требований разрабатывать код с максимальной системой ограничений, для задач реального времени -отказываться от решений, потенциально блокирующих поток

выполнения.

Библиографический список статьи включает 15 наименований, на приведенные источники в тексте имеются адресные ссылки, наличие которых свидетельствует об апелляции к оппонентам.

По рецензируемой статье следует высказать некоторые замечания.

Во-первых, вряд ли стоит в статье, подготовленной на русском языке, дублировать наименование каждого рисунка на английском.

Во-вторых, в разделе «Динамика развития диалектов, Оберон-07/13/16» при оформлении ссылки на рисунок 2 «Изменения Н. Вирта в языке Оберон» фраза «Основные изменения в языке сведены автором в таблицу на рисунке 2» нуждается в корректировке, поскольку иллюстрации могут быть представлены либо таблицей, либо рисунком.

Представленный материал соответствует тематике журнала «Кибернетика и программирование», содержит оригинальные разработки, ориентированные на повышение надёжности программного обеспечения компьютерных систем, выполняющих функции категории А, и рекомендуется к опубликованию.

Программные системы и вычислительные методы
Правильная ссылка на статью:
Вяткин С.И., Долговесов Б.С. — Прямой рендеринг трехмерных объектов на основе функций возмущения с использованием графических процессоров // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.38263 EDN: IWRNCU URL: https://nbpublish.com/library_read_article.php?id=38263

Прямой рендеринг трехмерных объектов на основе функций возмущения с использованием графических процессоров

Вяткин Сергей Иванович

кандидат технических наук

старший научный сотрудник, Институт автоматики и электрометрии СО РАН

630090, Россия, г. Новосибирск, ул. Коптюга, 1

sivser@mail.ru

Долговесов Борис Степанович

ORCID: 0000-0002-6255-9315

кандидат технических наук

Заведующий Лабораторией, Институт автоматики и электрометрии СО РАН

630090, Россия, г. Новосибирск, ул. Ак. Коптюга, 1, каб. 318

bsd@iae.nsk.su

Статья из рубрики "Автоматизация проектирования и технологической подготовки производства"

DOI:

10.7256/2454-0714.2023.1.38263

EDN:

IWRNCU

Дата направления статьи в редакцию:

13-06-2022

Аннотация: Объектом исследования является метод прямого рендеринга сложных трехмерных объектов на основе функций возмущения с применением графических процессоров, с использованием множества потоковых мультипроцессоров. Прямой рендеринг означает, что визуализация функционально заданных моделей происходит без их предварительного преобразования в другие форматы, например, в сетки треугольников.Метод исследования базируется на аналитической геометрии в пространстве, дифференциальной геометрии, теории интерполяции и теории матриц, опирается на математическое моделирование и теорию вычислительных систем. Основными выводами проведенного исследования являются: возможность прямого

рендеринга функционально заданных объектов, при рендеринге важно, чтобы вычислительные процессоры не простаивали. Первая проблема, которая была решена, заключалась в том, что у разных графических процессоров разное количество потоковых мультипроцессоров. Поэтому необходимо было выбрать во время исполнения оптимальную стадию, с которой начиналась работа. Таким образом, можно частично избавиться от проблемы с неиспользуемыми вычислительными ресурсами. Вторая проблема - задача балансировки, была решена с помощью использования большого количества вычислительных процессоров. Для реализации была использована модель параллельного программирования CUDA, которая вместе с набором программных средств позволяет реализовывать программы на языке C для исполнения на графическом процессоре. Полученная в результате система визуализирует сложные функционально заданные объекты с высоким разрешением в интерактивном режиме. Исследована зависимость производительности от вычислительной мощности графических процессоров.

Ключевые слова:

функционально заданный объект, функции возмущения, конструктивная твердотельная геометрия, прямой рендеринг, графический процессор, потоковые мультипроцессоры, параллельные вычисления, модель параллельного программирования, иерархия групп потоков, ускорение вычислений

1. Введение

Функционально заданные поверхности - это представление замкнутых многообразий для моделирования и визуализации. Функциональные представления являются способом представления твердотельных моделей [1, 2]. По сравнению с треугольными сетками и NURBS, они обеспечивают простые операции с CSG. В полигональные модели сложно внести атрибутивную информацию, а алгоритмы визуализации выполнения топологических операций достаточно трудоемки. Полигональные модели позволяют визуализировать только внешний слой объекта, в то время, как для многих приложений необходима внутреннее строение. Функциональные представления обеспечивают однозначную проверку внутри и снаружи объектов, и произвольное разрешение [3, 4]. Основная задача твердотельного моделирования состоит в том, чтобы знать, находится ли точка внутри или снаружи объекта [5-8]. Также важно знать, находится ли точка в пределах допуска к границе.

Для моделей с представлениями границ (B-rep) ответ на этот вопрос означает проверку точки по каждой части топологии в модели [9-12]. Общий подход заключается в создании луча, который начинается с рассматриваемой точки и движется в любом направлении. Если число пересечений нечетное, точка находится внутри объекта, в противном случае она находится снаружи. Такое вычисление является дорогостоящим и подверженным ошибкам, так как грани могут быть пропущены или пересчитаны дважды, если они нанесены вблизи их краев или вблизи касательной к краям или граням. Для моделей B-rep и сетки – это одна из основных проблем.

При моделировании функционально заданных поверхностей используют информацию о знаках для обозначения внутреннего и внешнего региона, поэтому вычисления являются простыми [13]. Значения, близкие к нулю, находятся ближе к границе, поэтому можно

установить, находится ли точка в пределах допуска к границе. Как с точки зрения точности, так и с точки зрения производительности, использование функционально заданных поверхностей лучше подходит для такой задачи.

Визуализировать функционально заданные не простая задача. В работе [14] описан метод визуализации поверхностей, заданных алгебраически полиномами высокой степени. При этом моделировать объекты с помощью полиномов сложно, не гарантируется то, насколько точно начальная функция будет приближена к кривой Безье. Кроме этого перевод объекта в другую систему координат – не простая задача, поэтому создание динамических сцен проблематично. Метод визуализации аналитически заданных объектов [15], основанный на не постоянном размере шага, вычисляет сферу с центром в текущей точке на луче, однако нахождение подходящего радиуса сферы не простая задача. Кроме того, в алгоритме использована предварительная обработка, поэтому, как и в предыдущем методе, визуализация объектов, которые изменяют свою форму и положение во времени требует больших вычислений.

Для быстрой визуализации обычно применяется преобразование функционально заданных поверхностей в сетки треугольников.

Однако сделать надежными такие преобразования очень сложно. Например, для топологических пространств, локально сходных с евклидовым, а для обнаружения тонких объектов требуется высокое разрешение выборки. Поэтому разработка метода прямого рендеринга функционально заданных объектов является актуальной задачей.

В нашей работе одной из главных задач является эффективное нахождение первого пересечения луча с поверхностью. Задание объектов с использованием функций возмущения позволяет эффективно осуществлять поиск точек поверхности. Для вычисления пересечения лучей с поверхностями трехмерных объектов предлагается метод, в котором отсутствуют недостатки, характерные для известных подходов. Уменьшение времени на визуализацию достигается за счёт эффективного использования вычислительных ресурсов графического процессора с архитектурой CUDA.

Целью данной работы является разработка метода прямого рендеринга 3D объектов на основе функций возмущения с применением графических процессоров.

2. Прямой рендеринг моделей на основе функций возмущения

Задание моделей с использованием функций возмущения описано в работе [13].

Для реализации алгоритма рендеринга на графическом процессоре (GPU), учитывалась его архитектура. Например, условные переходы осуществлять в такой архитектуре очень дорого, как с точки зрения времени выполнения операций, так и эффективности вычислений. Для графического процессора эффективно сразу "зарядить" его данными, чтобы он начал выполнять параллельные вычисления, иначе параллельные вычисления не эффективны и требуется синхронизация. Кроме этого графические процессоры имеют разное количество потоковых мультипроцессоров, поэтому необходимо учитывать оптимальную стадию с которой надо начинать работать. Благодаря таким мерам можно лишь частично решить проблему неиспользуемых вычислительных ресурсов, поскольку существует еще задача балансировки.

Рассмотрим сцену, находящуюся в единичном трёхмерном кубе. Наблюдатель смотрит на куб вдоль оси Z. С точки зрения наблюдателя исходят лучи такие, что каждый луч соответствует пикселю на изображении. Каждый из лучей делится вдоль оси Z. Таким

образом, вычисляем функцию плотности вдоль луча, которая зависит от одной переменной. Задача состоит в нахождении первой точки, в которой функция обращается в ноль. Поскольку мы имеем аналитически заданную функцию плотности, то это позволяет эффективно проводить поиск точки пересечения луча с поверхностью. После вычисления точки пересечения с поверхностью каждого луча, определяем глубину кадра. Затем в каждом пикселе вычисляем нормаль. Используя данные глубины и нормали в каждом пикселе, используем модель локального освещения. В результате получаем изображение гладкого объекта с учётом освещения.

Для реализации была использована архитектура CUDA). Это позволилоиспользовать большое количество вычислительных процессоров, которые одновременно вычисляют несколько лучей. Также учитывалось влияние скорости работы с памятью. Для ускорения вычислений максимально использованы регистры, так как это самый быстрый вид доступной памяти. Следующая по скорости - это совместно используемая память. Во всех остальных случаях использовалась общая память графического процессора.

В результате было реализовано приложение, с помощью которого можно осуществлять прямой рендеринг функционально заданных моделей. С помощью графического процессора вычисляются глубина кадра, нормали и освещение. В функции центрального процессора входят геометрические преобразования. Для отображения изображения использовалась DirectX. Тестирование производилось на процессоре Intel Core2 CPU E8400 3.0 GHz, GPU 9800 GT и 470 GTX. В таблице 1 приведены время вычислений, количество операций в секунду и ускорение.

Табл. 1. Время вычислений, количество кадров в секунду и ускорение

		0	1	2	3	4	6	7
Время	9800 GT	0,1164	0,39797	0,68046	0,3689	0,17703	0,07047	0,09109
	470 GTX	0,0178	0,06266	0,07235	0,04251	0,02109	0,01203	0,01125
	E8400	0,9015	3,17578	3,27562	2,0336	0,80265	0,59422	0,42344
Кол-во кадров в сек.	9800 GT	42,955	12,5637	7,34797	13,5538	28,2438	70,9521	54,8907
	470 GTX	280,58	79,7957	69,1085	117,619	237,079	415,627	444,444
	E8400	5,5459	1,57441	1,52642	2,45869	6,22936	8,41439	11,8080
Ускорение	9800 GT	7,7453	7,97994	4,81383	5,51260	4,53397	8,43224	4,64858
	470 GTX	50,592	50,6827	45,27464	47,83816	38,05832	49,39485	37,6391

На рисунке 1 показана диаграмма: по оси абсцисс отображен номер теста, по оси ординат – количество кадров в секунду для разных тестов. На рисунке 2 показано среднее время на кадр для разных тестов. На рисунке 3 – ускорение относительно E8400.

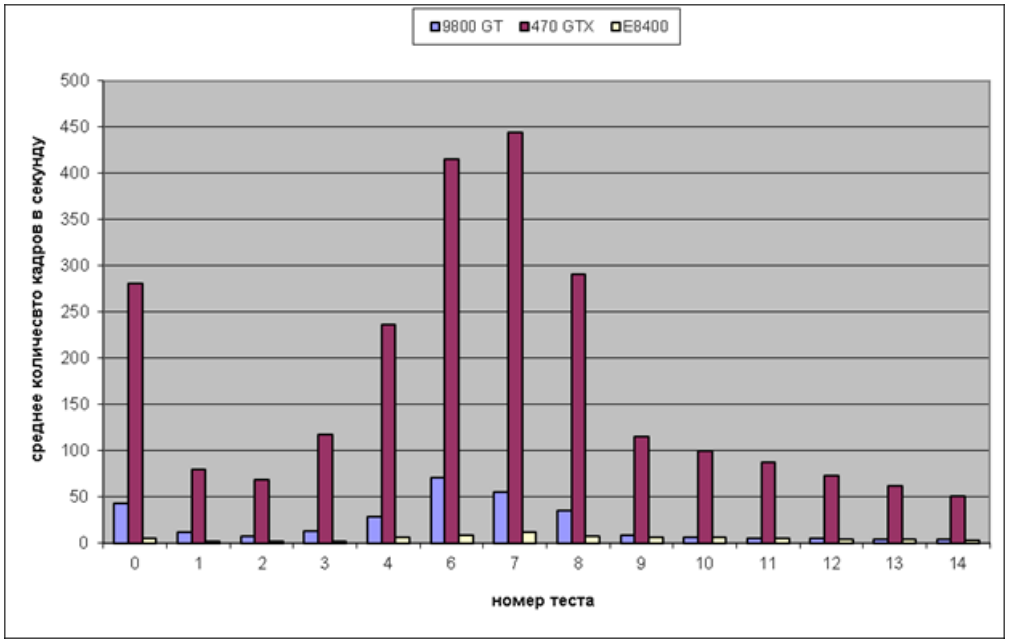


Рис. 1. Количество кадров в секунду для разных тестов

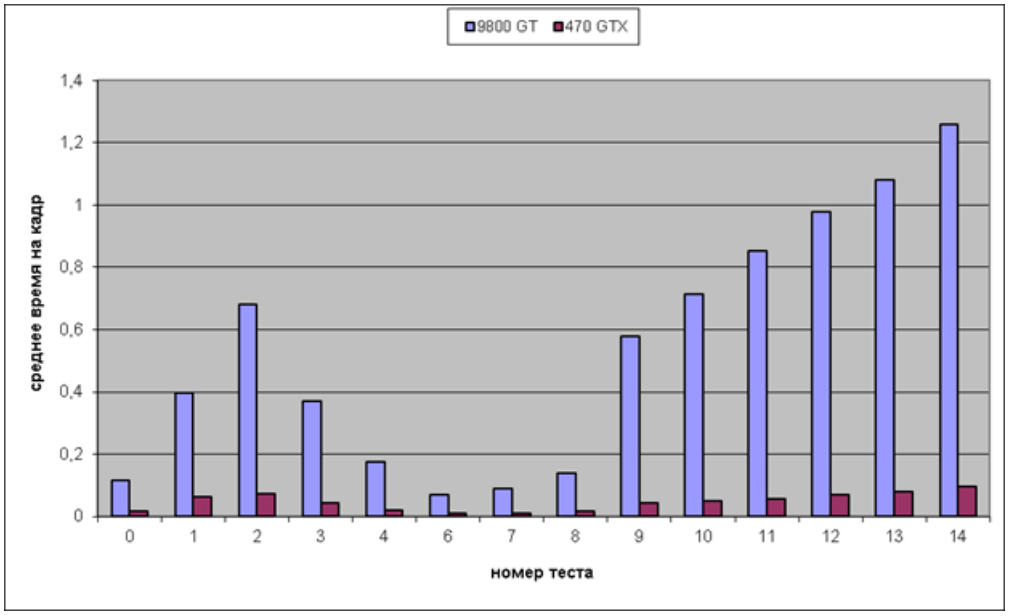


Рис. 2. Среднее время кадра для разных тестов

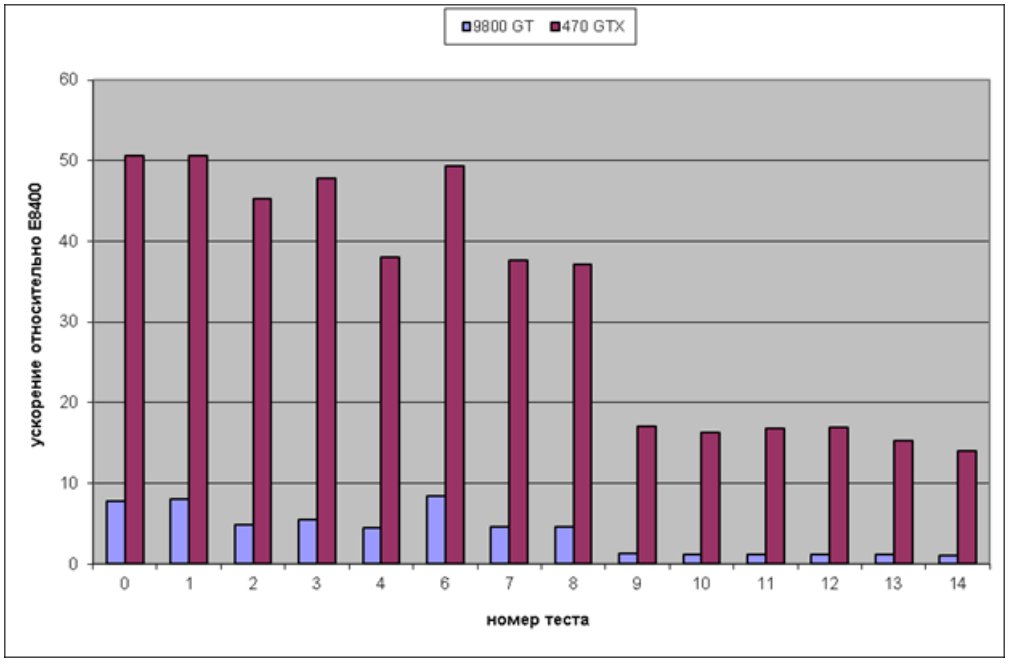


Рис. 3. Ускорение относительно центрального процессора E8400 для разных тестов

Мы сравнили предлагаемый метод прямого рендеринга с известными подходами.

Так в работе [14] описан метод рендеринга алгебраически заданных поверхностей. Однако разновидность поверхностей, которые можно описать с помощью функций возмущения намного шире, чем алгебраически заданными поверхностями. Кроме этого метод [14] не гарантирует то, насколько точно начальная функция приближена к кривой Безье. Также недостатком этого метода является то, что не просто преобразовать объект в другую систему координат. Поэтому создание динамических сцен проблематично.

В работе [15] описан метод с переменным размером шага на луче. Однако недостатком является нахождение подходящего радиуса, так как он эффективен только для статичных сцен. Для этого используется предварительные вычисления. Таким образом этот метод можно применять только тогда, когда объекты статичны. Поэтому, как и в предыдущем методе, рендеринг объектов, которые изменяют свою форму и положение во времени, не эффективно.

Заключение

Предложенные способ задания функциональных объектов и метод прямого рендеринга имеют преимущества перед известными подходами.

К основным достоинствам предлагаемых способа задания объектов на основе функций возмущения и метода прямого рендеринга относятся: простота и эффективность вычисления точек пересечения поверхности с лучами; компактное описание криволинейных объектов (задание объектов функционально заданными поверхностями сокращает в 100 и более раз описание базы данных по сравнению с описанием их полигонами); простота анимации и деформации поверхностей.

Функциональное задание объектов особенно актуально в ряде задач компьютерной графики, включая моделирование мягких или органических объектов, трехмерного морфинга, определения столкновений объектов и конструктивной твердотельной геометрии. Области применения функционально заданных объектов: молекулярная биология, интерактивные графические системы визуализации, CAD-системы, системы 3D-

моделирования, 3D веб-визуализация, аддитивное производство и т. д.

Библиография

1. Sigg, C. Representation and rendering of implicit surfaces. Diss. ETH No. 16664, Dipl. Rechn. Wiss. ETH Zurich, Switzerland, 2006, pages 162. DOI:10.3929/ETHZ-A-005267203 Corpus ID: 124349594
2. Dekkers, D., Overveld, K.V. Golsteijn, R. Combining CSG modeling with soft blending using Lipschitz-based implicit surfaces. Visual Computer, vol. 20, no. 6, 2004. pp. 380–391.
3. Vyatkin, S. An Interactive System for Modeling, Animating and Rendering of Functionally Defined Objects. American Journal of Computer Science and Engineering Survey, 2014, vol. 2, no. 3. pp. 102-108.
4. Vyatkin, S. Perturbation functions for compact database. Review of computer engineering research. 2017. vol. 4., no. 1. pp. 30–37. doi: 10.18488/journal.76.2017.41.30.37
5. Farin, G., Hoschek, J., Kim, M.S. Handbook of Computer Aided Geometric Design. [electronic resource]. 2002 Elsevier. ISBN 978-0-444-51104-1.
6. Pottmann, H., Brell-Cokcan, S., Wallner, J. Discrete Surfaces for Architectural Design. Archived 2009-08-12 at the Wayback Machine, pp. 213–234 in Curve and Surface Design, Patrick Chenin, Tom Lyche and Larry L. Schumaker (eds.), 2007. Nashboro Press, ISBN 978-0-9728482-7-5.
7. Farin, G. Curves and Surfaces for CAGD. A Practical Guide, Morgan-Kaufmann, 2002 ISBN 1-55860-737-4.
8. Sturm, T. An Algebraic Approach to Offsetting and Blending of Solids in Computer Algebra in Scientific Computing. CASC 2000, V.G. Ganzha, E.W.Mayr and E.V.Vorozhtsov (Eds.), Springer-Verlag, Berlin (2000), pp. 367-381.
9. Stroud, I. Boundary Representation Modelling Techniques. Publisher: Springer, January 2006, ISBN: 978-1-84628-312-3 DOI:10.1007/978-1-84628-616-2
10. Kainz, W., Neufeld, E., Bolch, W.E., Graff, C.G. Advances in Computational Human Phantoms and Their Applications in Biomedical Engineering-A Topical Review. IEEE Transactions on Radiation and Plasma Medical Sciences PP(99):1-1. December 2018 DOI:10.1109/TRPMS.2018.2883437
11. Arioli, C., Shamanskiy, A., Klinkel, S., Simeon, B. Scaled boundary parametrizations in isogeometric analysis. Computer Methods in Applied Mechanics and Engineering 349, March 2019, DOI:10.1016/j.cma.2019.02.022
12. Leidinger, L.F, Breitenberger, M., Bauer, A.M., Hartmann, S. Explicit dynamic isogeometric B-Rep Analysis of penalty-coupled trimmed NURBS shells. Computer Methods in Applied Mechanics and Engineering 351 April 2019, DOI:10.1016/j.cma.2019.04.016
13. Vyatkin, S.I. Complex Surface Modeling Using Perturbation Functions. Optoelectronics, Instrumentation and Data Processing, vol. 43, no. 3, 2007. pp. 40-47.
14. Reimers, M., Seland, J. Ray Casting Algebraic Surfaces using the Frustum Form. Eurographics, vol. 27 (2008), no. 2, pp. 361-370.
15. Liktor, G. Ray Tracing Implicit Surfaces on the GPU. Computer Graphics and Geometry, vol. 10, no. 3, 2008, pp. 36-53. http://www.cgg-journal.com/2008-3/04.htm

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Предмет исследования – разработка метода прямого рендеринга 3D-объектов на основе функций возмущения с применением графических процессоров.

Методология исследования основана на сочетании теоретического и эмпирического подходов с применением методов анализа, моделирования, численного эксперимента, обобщения, сравнения, синтеза.

Актуальность исследования определяется широким распространением технологий компьютерной графики, необходимостью разработки соответствующих программных систем и вычислительных методов, включая рендеринг трёхмерных объектов с использованием графических процессоров.

Научная новизна связана с предложенным автором способом задания функциональных объектов и метод прямого рендеринга, который имеет ряд преимуществ перед известными аналогами (простота и эффективность вычисления точек пересечения поверхности с лучами,
компактное описание криволинейных объектов, простота анимации и деформации поверхностей).

Статья написана русским литературным языком. Стиль изложения научный.

Структура рукописи включает следующие разделы: 1. Введение (функционально заданные поверхности, представление замкнутых многообразий для моделирования и визуализации, твердотельные модели, треугольные сетки и NURBS, простые операции с CSG, полигональные модели, модели с представлениями границ (B-rep), задача визуализации, метод визуализации поверхностей, заданных алгебраически полиномами высокой степени, метод визуализации, основанный на не постоянном размере шага, преобразование функционально заданных поверхностей в сетки треугольников, разработка метода прямого рендеринга функционально заданных объектов, эффективное нахождение первого пересечения луча с поверхностью), 2. Прямой рендеринг моделей на основе функций возмущения (задание моделей с использованием функций возмущения, реализация алгоритма рендеринга на графическом процессоре (GPU), сцена в единичном трёхмерном кубе, получение изображения гладкого объекта с учётом освещения, архитектура CUDA, приложение для прямого рендеринга функционально заданных моделей (вычисляются глубина кадра, нормали и освещение), отображение изображения DirectX, тестирование на процессоре Intel Core2 CPU E8400 3.0 GHz, GPU 9800 GT и 470 GTX, количество кадров в секунду, среднее время кадра и ускорение (относительно центрального процессора E8400) для разных тестов, сравнение предлагаемого метода прямого рендеринга с известными подходами – метод рендеринга алгебраически заданных поверхностей, метод с переменным размером шага на луче), Заключение (выводы), Библиография.

Разделы «Введение», «Заключение» следует единообразно пронумеровать либо оставить без нумерации. Дублирование данных в таблице и на рисунках не целесообразно, что следует исправить.

Текст включает одну таблицу, три рисунка. Дублирование данных в таблице и на рисунках не целесообразно, что следует исправить. Для представленных величин нужно указать размерность.

Содержание в целом соответствует названию. Встречающиеся в тексте аббревиатуры (CSG, CUDA и др.) при первом упоминании следует привести полностью.

Библиография включает15 источников отечественных и зарубежных авторов – монографии, научные статьи, Интернет-ресурсы и пр. Библиографические описания некоторых источников требуют корректировки в соответствии с ГОСТ и требованиями редакции, например:
1. Sigg C. Representation and rendering of implicit surfaces. Zurich, Switzerland : Dipl. Rechn. Wiss. ETH, 2006. 162 p.
2. Dekkers D., Overveld K. V., Golsteijn R. Combining CSG modeling with soft blending using Lipschitz-based implicit surfaces // Visual Computer. 2004. Vol. 20. № 6. P. 380–391.
5. Farin G., Hoschek J., Kim M.S. Handbook of Computer Aided Geometric Design.Место издания ??? : Elsevier, 2002. ??? p.
Возможно излишнее самоцитирование (Vyatkin S.I. с соавторами).

Апелляция к оппонентам (Sigg C., Dekkers D., Overveld K. V., Golsteijn R., Farin G., Hoschek J., Kim M. S., Pottmann H., Brell-Cokcan S., Wallner J., Farin G., Sturm T., Stroud I., Kainz W., Neufeld E., Bolch W.E., Graff C. G., Arioli C., Shamanskiy A., Klinkel S., Simeon B., Leidinger L. F, Breitenberger M., Bauer A. M., Hartmann S., Reimers M., Seland J., Liktor G. и др.) имеет место.

В целом материал представляет интерес для читательской аудитории и после доработки может быть опубликован в журнале «Программные системы и вычислительные методы».

Программные системы и вычислительные методы
Правильная ссылка на статью:
Боревич Е.В. — Ай-трекинговое исследование влияния композиции на восприятие кинокадра // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.39634 EDN: IWYBNX URL: https://nbpublish.com/library_read_article.php?id=39634

Ай-трекинговое исследование влияния композиции на восприятие кинокадра

Боревич Екатерина Владиславовна

ORCID: 0000-0001-6263-3901

ассистент высшей школы дизайна и архитектуры, Санкт-Петербургский политехнический университет Петра Великого

195251, Россия, Санкт-Петербург область, г. Санкт-Петербург, ул. Политехническая, 29

plasma5210@mail.ru

Статья из рубрики "Компьютерная графика, обработка изображений и распознавание образов"

DOI:

10.7256/2454-0714.2023.1.39634

EDN:

IWYBNX

Дата направления статьи в редакцию:

18-01-2023

Аннотация: Представленные исследования направлены на изучение элементов, влияющих на визуальное восприятие кинокадра с целью выработки методических рекомендаций для гармонизации кадра в процессе кинопроизводства. Объект исследования – кинокадр. Предмет исследования – технологии обработки кинокадра. Цель данной работы – получить экспериментальные данные шаблона рассматривания кинокадра и выявить статистические закономерности для подтверждения или опровержения сформулированной гипотезы. Задача – провести экспериментальное исследование влияния композиции на параметры шаблона рассматривания кинокадра. Исследуется влияние фактор взаимного отношения площадей центров интереса к фону на параметры шаблона рассматривания стимульного материала. В результате проделанной работы разработана методика проведения экспериментальных исследований восприятия визуальной информации человеком с применением программно-аппаратного комплекса ай-трекинга. Полученные результаты показывают, что при условии, когда объекты занимают незначительную площадь кадра, наблюдателю требуется больше времени, чтобы рассмотреть этот кадр. Также как и в случае, когда объекты занимают большую часть кадра (более 40%). В первом случае, ввиду небольших размеров объектов, наблюдателю становится труднее отыскать объекты в

пространстве кадра. Во втором случае, требуется время, для идентификации объектов, так как они, в виду больших размеров, тяготеют к тому, чтобы быть восприняты как фон.

Ключевые слова:

Композиция, Визуальная привлекательность, Кинокадр, Ай-трекер, Цветовое решение, Эксперимент, Статистический анализ данных, Гештальтпсихология, Зрительная система человека, Кинематограф

Введение

Кинематограф, как явление культуры с достаточно специфическими особенностями, сформировался, пройдя много этапов своего развития. Прогресс кино шел не только в техническом плане, который состоял в развитии и совершенствовании средств съемки и монтажа фильма, но и в художественном. Элемент творчества не сразу вошел в режиссуру фильма, для этого потребовалось продолжительное время становления кинематографа как предмета искусства. В настоящее время, можно говорить о наличии в кинематографе своего, уникального художественного языка повествования, который имеет свой синтаксис [1].

Восприятие художественного произведения, и кинокадра в том числе, тесно связано с физиологией восприятия визуальной информации человеком. Законы построения кадра в кинематографе, аналогичны произведениям живописи и стремятся к такому отражению действительности, которое способствовало бы более глубокому погружению зрителя в происходящее действие на экране. Важную роль в процессе погружения зрителя в атмосферу фильма играет правильно и качественно спроектированный кинокадр, являющийся минимальной структурной единицей кинофильма.

Существует несколько альтернативных подходов к шаблону восприятия информации. Один из них рассматривает этот процесс как интеграцию путем пространственного сопоставления информации путём суммирования информации от последовательных фиксаций [2]. Альтернативный подход предполагает, что информация из последовательных фиксаций агрегируется не путем объединения «снимков» фиксаций, а путем интеграции более сложных визуальных атрибутов на средних и высоких уровнях анализа [3].

Представленные исследования направлены на изучение элементов, влияющих на визуальное восприятие кинокадра с целью выработки методических рекомендаций для гармонизации кадра в процессе кинопроизводства.

Объект исследования – кинокадр. Предмет исследования – технологии обработки кинокадра.

1. Теоретическая модель

В настоящее время не существует общепринятой единицы измерения визуальной привлекательности кинокадра, что не позволяет сравнивать кадры между собой сравнением их параметров. Для оценки качества выполненной работы используется достаточно субъективный экспертный анализ.

Под *Визуальной привлекательностью кинокадра,* понимаем свойство кинокадра,

направленное на привлечение и удержание внимания зрителя, в результате совокупного воздействия его элементов (рис.1). Композиционное построение кадра состоит из трех основных элементов. Первый – это композиция. Второй элемент кинокадра – это наличие движения. И третий – это цветовое решение [4].

На визуальную привлекательность кинокадра влияет «информативность» и композиционное построение. В свою очередь «информативность» кадра определяется замыслом режиссера, а реализация замысла происходит с помощью композиционного построения, состоящего из трех составляющих – композиции, динамики и цветового решения (рис.1). В результате экспериментального исследования было выявлено, что «информативность» кадра влияет на шаблон рассматривания кинокадра [5]. Под шаблоном рассматривания понимается набор количественных параметров глазодвигательной активности, получаемый с помощью программно-аппаратного комплекса eye-tracker при рассматривании испытуемым стимульного материала [6]. Композиционное построение кадра, как его неотъемлемая составляющая, тоже влияет на шаблон рассматривания. Анализируя параметры шаблона рассматривания методами математической статистики, можно сравнивать визуальную привлекательность кадров, используемых в качестве стимульного материала в проводимом эксперименте.

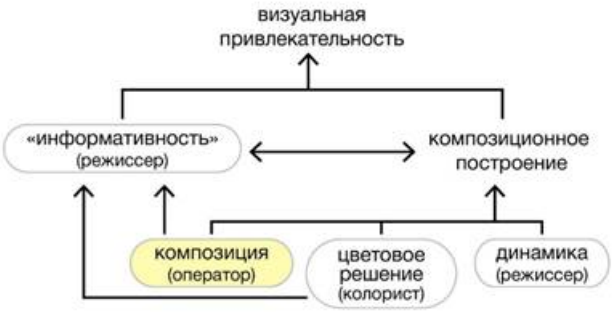


Рис. 1. Схема проектирования кадра. Элементы кадра, влияющие на визуальную привлекательность

В рамках настоящего исследования был проведен ряд экспериментов, целью которых было исследование значимости влияния цветового решения [7,8], «информативности» и стилизации [9] кинокадра на восприятие его зрителем. Работа зрительной системы по распознаванию образа происходит в три этапа (рис.2). Первый этап – это механическое сканирование изображения глазом. Он происходит бессознательно и заключается в беглом «ощупывании» изображения глазом. Второй этап – это распознавание увиденного образа – подключается работа головного мозга. Третий этап – это эмоциональный отклик зрителя. Человек составляет собственное впечатление от увиденного. В результате экспериментального исследования было установлено, что требуемое для сканирования изображения время – величина постоянная, а время распознавания стимула зависит от содержания.

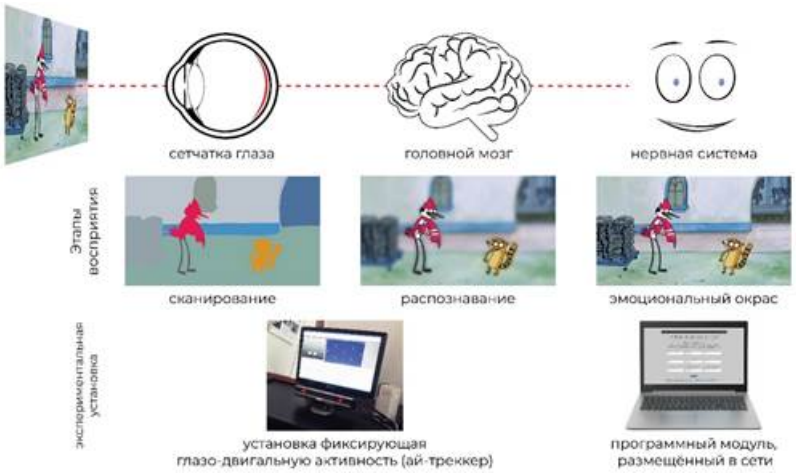


Рис.2. Схема работы зрительной системы человека по распознаванию образа

Цель данной работы – получить экспериментальные данные шаблона рассматривания кинокадра и выявить статистические закономерности для подтверждения или опровержения сформулированной гипотезы.

Постановка задачи исследования . Провести экспериментальное исследование влияния композиции на параметры шаблона рассматривания кинокадра.

2. Стимульный материал

Для составления визуального ряда, стимульного материала, выбраны кадры из фильмов. Содержимое кадров отвечало ряду требований, таких как: эмоциональная нейтральность и наличие минимальной смысловой нагрузки, однако предметность должна сохраняться. Кроме того, кадры были выбраны таким образом, чтобы изображение имело два центра интереса, имеющих разные взаимные отношения площадей. В итоге была создана база из 120 соответствующих поставленным условиям кадров (рис.3).

Рис. 3. Пример стимульного материала

Целью настоящего эксперимента являлось исследование влияния фактора композиции на восприятие кинокадра. В описываемом эксперименте было уделено внимание композиционному построению кадра – взаимному соотношению площадей центров интереса к фону. Стимульный материал эксперимента подобран так, чтобы взаимное отношение площадей центров интереса к фону можно было разделить на три группы. В результате были получены изображения, которые однозначно можно разделить на три группы (рис.4): объекты занимают менее 25% площади кадра, от 25 до 40% и группа, в которой объекты занимают более 40% площади кадра. Вычислялось процентное соотношение суммы площадей объектов (центров интереса) к фону.

А)  б)  в) 

Рис.4. Стимульный материал с процентным соотношением площади, занимаемой центрами интереса, к фону А) Менее 25%; б) от 25 до 40%; в) более 40%

Следующий фактор — это расстояние между объектами (центрами интереса). Стимулы разделены на две группы: в первой группе объекты располагаются в кадре так близко, что тяготеют быть воспринятыми как один гештальт (рис.5а). Во второй группе стимулов объекты расположены на расстоянии большем, чем сам объект, что позволяет их воспринимать как отдельные сущности(рис.5б). Поле зрения определяется как «количество градусов угла зрения при стабильной фиксации глаза [10]. При увеличении расстояния между объектами увеличивается задействованная область поля зрения у испытуемого, что требует от него осуществлять более значительные перемещения взгляда по пространству кадра для идентификации объектов.

А)  б) 

Рис. 5. Стимульный материал с расстоянием между объектами (центрами интереса): А) меньшем, б) большем; чем большее из двух измерений объекта

3. Постановка эксперимента

В настоящее время активно развиваются исследования, использующие технологию фиксирующую глазодвигательную активность [11]. Исследователи представляют новейшие технологии отслеживания взгляда и оценки взгляда. Вслед за недавними технологическими достижениями и появлением доступных ай-трекеров растет интерес к широко распространенным технологиям, ориентированным на внимание системы и интерфейсы, которые могут произвести революцию в общепринятом человеческо-технологическом взаимодействии [12].

Для проведения текущего эксперимента использовался аппаратный комплекс, фиксирующий глазодвигательную активность – программно-аппаратный комплекс ай-трекинга SMI RED 250 (рис. 6). Для каждого испытуемого настраивалось кресло и подставка для фиксации головы. Также производилась калибровка ай-трекера индивидуально с помощью встроенной в комплекс функции.

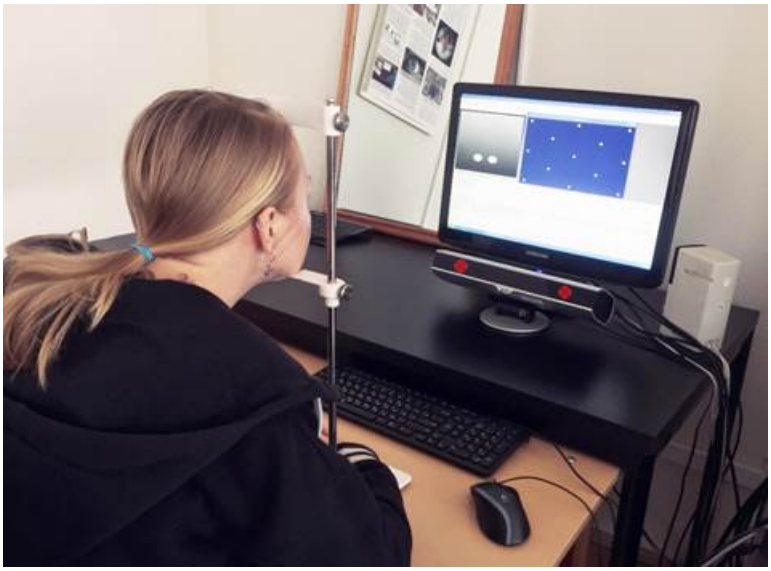


Рис. 6. Экспериментальная установка с ай-трекером: компьютерный монитор, ай-трекер и подголовник

В настоящем эксперименте испытуемым было предложено решить задачу по идентификации демонстрировавшихся им на первом этапе стимулов, среди множества других. В эксперименте фиксировалось изменение шаблона рассматривания кадра в зависимости от двух факторов: площадь главных объектов в кадре и расстояние между объектами.Результаты эксперимента по влиянию цветового решения описаны в статье[4].

Эксперимент состоял из двух этапов. На первом этапе испытуемому предъявлялись 10 стимулов. Необходимо было запомнить изображения. Время для запоминания не ограничивалось. Через временной промежуток времени (30 мин.) испытуемому предъявлялись 50 стимулов, которые включали все кадры из первой части эксперимента и случайно выбранные из базы 40 стимулов. На втором этапе испытуемому предлагалось решает задачу по распознавания стимула. Каждому участнику эксперимента предлагается выбрать из 50 кадров те стимулы, которые он уже видел ранее. Порядок демонстрации материала на обоих этапах последовательный и случайный. Продолжительность рассматривания стимула испытуемым на этапе распознавания испытуемый также устанавливал самостоятельно.

4. Результаты

В эксперименте участвовало 44 человека от 18 до 25 лет из студентов Санкт-Петербургского политехнического университета Петра Великого. Статистическая обработка результатов эксперимента производилась посредством дисперсионного анализа ANOVA[13]. Анализировались следующие параметры шаблона рассматривания: время наблюдения стимула, средняя длительность фиксаций при наблюдении одного стимула, среднее количество фиксаций при наблюдении одного стимула, среднее время саккад и среднее количество саккад при наблюдении одного стимула. Задачей анализа было выявить влияние факторов размера объектов и их удаленности друг от друга. Значение критерия значимости p-value для принятия гипотезы было выбрано 0,05. Значения p-value, полученные в результате выполнения вычислительной процедуры представлены, в таблице 1: фактор Square – процентное отношение суммы площадей объектов к фону; фактор dist – расстояние между объектами.

Таблица 1. Значения критерия значимости p-value для исследуемых факторов

фактор	p-value
square	0.042829
dist	0.040432

График плотности распределения времени рассматривания стимула в зависимости от фактора размера представлен на рисунке 7.

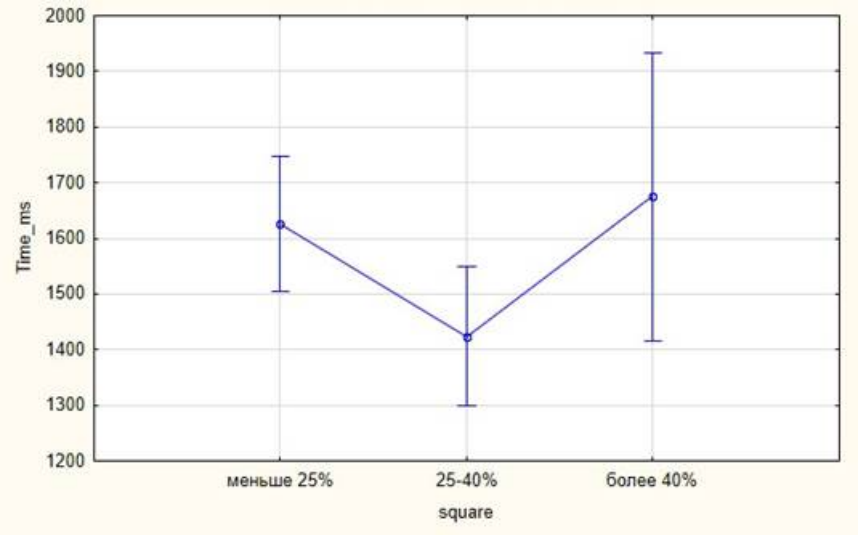


Рис. 7. График плотности распределения времени рассматривания стимула в зависимости от фактора размера объектов

По оси ординат расположены значения параметра длительности рассматривания стимула в миллисекундах. Представлены значения для стимулов, разделённых на три группы по фактору взаимного отношения площадей: менее 25% - размеры объектов незначительные по сравнению с площадью кадра, такие объекты сложнее разглядеть, потому что они теряются на большом пространстве фона; 25-40% - для рассматривания этой группы стимулов испытуемым потребовалось наименьшее количество времени, из чего можно сделать вывод, что эта группа стимулов оказалась наиболее удобной для рассматривания; более 40% - размеры объектов настолько значительны, что тяготеют быть воспринятыми как фон, поэтому для их идентификации требуется больше времени.

Экспериментальные данные показали, что фактор удалённости объектов друг от друга влияет на восприятие кинокадра (рис. 8).

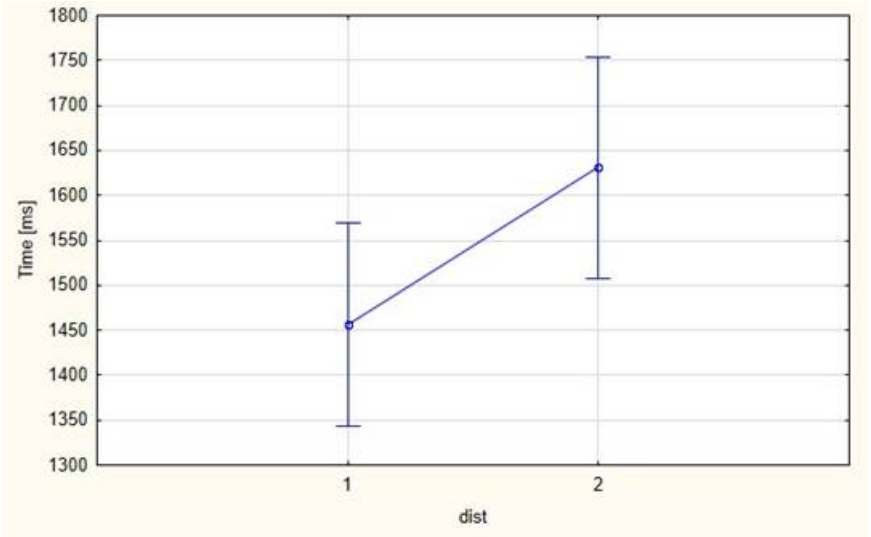


Рис.8. График плотности распределения времени рассматривания стимула в зависимости от фактора расстояния между объектами: 1 – расстояние между объектами меньше; 2 – расстояние между объектами больше, чем объект

По оси ординат расположены значения параметра длительности рассматривания стимула в миллисекундах. Представлены значения для стимулов, разделённых на две группы по фактору расстояния между объектами: 1 – расстояние между объектами меньше, чем объект; 2 – расстояние между объектами больше, чем объект, поэтому испытуемым требовалось больше времени для рассматривания большей площади кадра.

По результатам статистического анализа можно утверждать, что параметры шаблона рассматривания имеют статистически значимую зависимость от факторов размера и удаленности объектов друг от друга в кадре.

5. Выводы

При анализе данных о влиянии фактора размера объектов, полученные результаты показывают, что при условии, когда объекты занимают незначительную площадь кадра, наблюдателю требуется больше времени, чтобы рассмотреть этот кадр. Также как и в случае, когда объекты занимают большую часть кадра (более 40%). Можно предположить, что, в первом случае, ввиду небольших размеров объектов, наблюдателю становится труднее отыскать объекты в пространстве кадра, а во втором случае, требуется время, для идентификации объектов, так как они, в виду больших размеров, тяготеют к тому, чтобы быть восприняты как фон.

Если рассматривать фактор удалённости объектов друг от друга, то стоит отметить, что при большем расстоянии между объектами испытуемому приходится переводить взгляд по площади кадра, что заставляет его дольше рассматривать кадр.

Композиция влияет на шаблон рассматривания кадра человеком. Когда два объекта расположены на большем расстоянии друг от друга в кадре, то зритель рассматривает дольше такой кадр.

В результате проделанной работы разработана методика проведения экспериментальных исследований восприятия визуальной информации человеком с применением программно-аппаратного комплекса ай-трекинга.

Разработанная методика была апробирована на 44 испытуемых из числа студентов Санкт-Петербургского политехнического университета Петра Великого и показала свою состоятельность.

Кинокадр – это условный интерфейс передачи визуальной информации [14]. Разработанная методика может быть использована при проведении исследований по восприятию визуальной информации в графических интерфейсах систем удаленного управления динамическими объектами.

Библиография

1. Лотман Ю.М. Семиотика кино и проблемы киноэстетики // Лотман Ю.М. Об искусстве (СПб., 1998)
2. Jonides J., Irwin D.E., Yantis S. Integrating visual information from successive fixations // Science. 1982. V.215.P. 192-194.
3. Burr D., Morrone M.C. Eye movements: building a stable world from glance to glance //Curr Biol. 2005. V.15. №20.P. 839-840.

4. Mescheryakov S.V., Yanchus V.E., Borevich E.V. Experimental Research of Digital Color Correction Models and Their Impact on Visual Fixation of Video Frames // Humanities and Science University Journal. 2017. V. 27. P. 15-24.
5. Borevich E.V., Yanchus V.E., Mescheryakov S.V. Computer Eye-Tracking Model to Investigate Influence of the Viewer's Perception of the Graphic Information // GraphiCon 2021: труды 31-й Междунар. конф. по компьютерной графике и машинному зрению. P. 720-728
6. Орлов П.А., Лаптев В.В., Иванов В.М. К вопросу об использовании ай-трекинг систем // Научно-технические ведомости СПбГПУ. 2014. Т. 5 (205). С. 84–94.
7. Янчус В.Э., Боревич Е.В. Исследование значения цветового решения в процессе гармонизации кинокадра // Научно-технический вестник СПбГПУ. 2016. Т. 4. С. 53–68.
8. Borevich E.V., Mescheryakov S.V., Yanchus V.E. Statistical Model of Computing Experiment on Digital Color Correction //DCCN.2019.P.140-150.
9. Янчус В.Э., Боревич Е.В., Авдеева А.А. — Применение технологии ай-трекинга в вопросах исследования восприятия графической информации // Программные системы и вычислительные методы. – 2021. – № 1. DOI: 10.7256/2454-0714.2021.1.33378
10. Страсбургер, Ганс; Пёппель, Эрнст (2002). Поле зрения. В G. Adelman & BH Smith (Eds): Encyclopedia of Neuroscience; 3-е издание на компакт-диске. Elsevier Science BV, Амстердам, Нью-Йорк.
11. Norman D.A., Draper S.W. User Centered System Design. Erlbaum: London, 1986. 526 p.
12. Majaranta P., Bulling A. Eye Tracking and Eye-Based Human-Computer Interaction // Advances in Physiological ComputingHuman-Computer Interaction Series, 2014. P. 39-59.
13. Гланц, С. Медико-биологическая статистика / Стентон Гланц; Пер. с англ. д.ф.-м.н. Ю.А. Данилова под ред. Н.Е. Бузикашвили и Д.В. Самойлова. – М.: Практика, 1999. – 459 с.
14. Железняков В. Н. Цвет и контраст. Технология и творческий выбор: Учебное пособие. — М.: ВГИК, 2001. — 286 с.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемая статья посвящена актуальной задаче анализа визуального восприятия сцен в зависимости от структуры кинокадра. Авторы обосновывают практическую значимость данного исследования, выбор метода исследования, анализируют элементы кадра, определяющие его визуальную привлекательность. Авторы провели исследование с участием 44 добровольцев, приводят иллюстрации эксперимента, подробно описывают стимульный материал что является важным критерием в экспериментах с ай-трекером, выполнили статистический анализ полученных данных. Авторы делят все стимулы на 2 группы, четко обосновывая критерий разделения. Планирование экспериментальной части выполнено корректно, качество изображений достаточное, оцениваемые объекты имеют различную форму.
Недостатком статьи является достаточное краткое упоминание аналогичных

экспериментов других авторов, минимальное количество численных оценок.

Собственное исследование Авторами выполнено корректно, приводится обоснование выбора аппарата, планирование эксперимента, имеются иллюстрации использования аппаратного комплекса с соблюдением конфиденциальности данных, выполнен анализ зависимости времени рассматривания стимула от размера и расстояния между объектами.

Стиль изложения соответствует научной статье, корректно используется профессиональная терминология, стилистические не выявлены. Имеются иллюстрации с подрисуночными подписями, данные представлены в высоком качестве.

Структура статьи отвечает требованиям к научной публикации.

Библиография содержит 14 источников, в отечественных и зарубежных рецензируемых изданиях, из них 3 за последние 5 лет. Ссылки по тексту имеются. Оформлена в соответствии с требованиями Журнала, однако в некоторых позициях встречаются опечатки.

Замечания.

Статья содержит единичные орфографические ошибки.

Рис. 2 содержит пояснительный текст в теле изображения, однако выбор шрифта с учетом алгоритмов сжатия не позволяет ожидать высокого качества при публикации в Журнале. Необходимо использовать больший кегль.

Желательно в разделе Стимульный материал пояснить, почему при разделении кадров на 3 группы были выбраны пороги 25% и 40%.

В описании эксперимента желательно отметить характеристики комплекса, отвечающие за пространственное разрешение или иные технические характеристики оборудования (например, чувствительность). Отсутствуют данные о размерах изображений (в пикселях), формате данных (что позволит косвенно оценить глубину цветопередачи). Желательно отметить, что измеряемой величиной было время, необходимое для визуального восприятия и идентификации зрительного стимула.

Графические зависимости времени рассматривания объекта от его размера и расстояния имеют минимальное количество точек. Вероятно результаты расчетов для расстояния лучше представить в табличной форме или дополнить. Большим количество градаций (по оси X).

Необходимо в тексте более конкретно сформулировать результаты обработки экспериментальных данных, представленных на рис. 7-8.

Необходимо внести некоторые технические правки в выходные данные использованных источников (см. требования Журнала и ГОСТ).

Статья рекомендуется к публикации с небольшими техническими правками, в повторном рецензировании не нуждается.

Англоязычные метаданные

Markdown File Converter to LaTeX Document

Nuriev Marat Gumerovich

PhD in Technical Science

Senior Lecturer, Department of Automated Information Processing and Control Systems, Kazan National Research Technical University named after A.N. Tupolev-KAI

420015, Russia, Republic of Tatarstan, Kazan, Bolshaya Krasnaya str., 55

marat_nu1@mail.ru

Belashova Elena Semenovna

PhD in Physics and Mathematics

Associate Professor of the Computer Systems Department of Kazan National Research Technical University named after A.N. Tupolev-KAI

420015, Russia, Republic of Tatarstan, Kazan, Bolshaya Krasnaya str., 55

bel_lena@mail.ru

Barabash Konstantin Alekseevich

Student, Computer Systems Department, Kazan National Research Technical University named after A.N.Tupolev-KAI

420015, Russia, Republic of Tatarstan, Kazan, Bolshaya Krasnaya str., 55

kostyandriy@mail.ru

Abstract. Common text editors such as Microsoft Word, Notepad++ and others are cumbersome. Despite their enormous functionality, they do not eliminate the risk of incorrectly converting the document, for example, when opening the same Word files on older or, conversely, newer versions of Microsoft Word. The way out is the use of markup languages, which allow you to mark up text blocks in order to present them in the desired style. Currently, very popular are LaTeX (a set of macro-extensions of the TeX typesetting system) and Markdown (a lightweight markup language, designed to denote formatting in plain text). So the question of converting a Markdown document into a LaTeX document is relevant. There are various tools to convert Markdown files to LaTeX document, such as Pandoc library, Markdown.lua, Lunamark and others. But most of them have redundant steps to generate the output document. This paper highlights a solution method by integrating a Markdown file into a LaTeX document, which will potentially reduce the output document generation time unlike existing solutions. The developed Markdown to LaTeX document converter will automatically generate the output document and reduce the possibility of errors when manually converting text from Markdown format to LaTeX format.

Keywords: text conversion, Overleaf, markup language, Python, converter, programming, LaTeX, Markdown, Word, regular expressions

References (transliterated)

1. Mekhmonov I. N. Instrumentarii avtomatizirovannogo formirovaniya dinamicheskikh dokumentov //Prikladnaya matematika i informatika: Sovremennye issledovaniya v oblasti estestvennykh i tekhnicheskikh nauk. – 2020. – S. 883-886.

2. Pavlov D. A. Avtomaticheskaya verstka i oformlenie nauchnoi i programmnoi dokumentatsii //Komp'yuternye instrumenty v obrazovanii. – 2018. – №. 6. – S. 39-46.
3. B. Luo, W. Zhu, P. Li and Z. Han, "Distributed Dynamic Cuckoo Filter System Based on Redis Cluster," 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), 2018, pp. 244-247, doi: 10.1109/BDS/HPSC/IDS18.2018.00059.
4. J. Tippayachai and S. Kiattisin, "Academic Publishing Solution Based on LATEX Class Package Implementation for ITMSOC Journal," 2018 3rd Technology Innovation Management and Engineering Science International Conference (TIMES-iCON), 2018, pp. 1-5, doi: 10.1109/TIMES-iCON.2018.8621689.
5. Gibadullin, R.F., Lekomtsev, D.V. & Perukhin, M.Y. Analysis of Industrial Network Parameters Using Neural Network Processing. Sci. Tech. Inf. Proc. 48, 446–451 (2021). https://doi.org/10.3103/S0147688221060046.
6. Gibadullin R.F. Potokobezopasnye vyzovy elementov upravleniya v obogashchennykh klientskikh prilozheniyakh // Programmnye sistemy i vychislitel'nye metody. – 2022. – № 4. – S. 1-19. DOI: 10.7256/2454-0714.2022.4.39029 EDN: IAXOMA URL: https://nbpublish.com/library_read_article.php?id=39029.
7. Gibadullin R.F. Organizatsiya zashchishchennoi peredachi dannykh v sensornoi seti na baze mikrokontrollerov AVR // Kibernetika i programmirovanie. – 2018. – № 6. – S. 80-86. DOI: 10.25136/2306-4196.2018.6.24048 URL: https://nbpublish.com/library_read_article.php?id=24048.
8. Gibadullin R. F. Razvitie edinoobraznogo formalizma zashchity tochechnykh, lineinykh i ploshchadnykh ob''ektov kartografii // Vestnik Kazanskogo gosudarstvennogo tekhnicheskogo universiteta im. A.N. Tupoleva. – 2010. – №. 2. – S. 101-105.

Syntax Tree Development for Automated Serial-to-Parallel Code Translator for Multicore Processors

Viktorov Ivan Vladimirovich

Postgraduate Student of the Computer Systems Department of Kazan National Research Technical University named after A.N. Tupolev-KAI (KNITU-KAI)

420015, Russia, Republic of Tatarstan, Kazan, Bolshaya Krasnaya str., 55, office 432

victorov.i.vl@yandex.ru

Gibadullin Ruslan Farshatovich

PhD in Technical Science

Associate Professor of the Computer Systems Department of Kazan National Research Technical University named after A.N. Tupolev-KAI (KNITU-KAI)

420015, Russia, Republic of Tatarstan, Kazan, Bolshaya Krasnaya str., 55, office 432

rfgibadullin@kai.ru

Abstract. The emergence of multicore architectures has extremely stimulated the area of parallel computing. However, developing a parallel program and manually paralleling inherited sequential program codes are time-consuming work. The programmer should have good skills in using parallel programming methods. This fact determines the relevance of the subject of the research – the development of a serial-to-parallel code translator. The article gives a review of existing solutions in the chosen direction of research and considers their advantages

and disadvantages. The principle of formation of a syntactic tree which is based on JSON format (the text format of data exchange based on JavaScript) is offered and an example of formation of a syntactic tree on the basis of this principle is considered. The result of the work is an approach for building a program platform for translating sequential code into parallel code. The distinctive feature of the developed platform is the web-service, which potentially allows you to extend the translator with other programming languages. The interaction with the programming environment is realized by means of REST-requests (HTTP-requests designed to call remote procedures). The developed software platform consists of three modules: the query processing module, which provides interaction with external systems through REST-requests; the tree building module, which forms a syntax tree on the basis of the source program code; the code conversion module, which obtains parallel program code on the basis of the syntax tree.

Keywords: programming languages, JSON format, automated translator, multithreaded programming, parallel programming, parallel computing, multicore processors, syntax tree, web service, REST requests

References (transliterated)

1. P. Czarnul, J. Proficz and K. Drypczewski, "Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems," Scientific Programming, vol. 2020, pp. 1058–9244, 2020.
2. D. B. Changdao, I. Firmansyah and Y. Yamaguchi, "FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method," in Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th Int. Symp., ARC 2020, Toledo, Spain, Springer Nature. vol. 12083, pp. 232, 2020.
3. D. Wang and F. Yuan, "High-performance computing for earth system modeling," High Performance Computing for Geospatial Applications, vol. 23, pp. 175–184, 2020.
4. M. U. Ashraf, F. A. Eassa, A. Ahmad and A. Algarni, "Empirical investigation: Performance and power-consumption based dual-level model for exascale computing systems," IET Software, vol. 14, no. 4, pp. 319–327, 2020.
5. B. Brandon, "Message passing interface (mpi)," in Workshop: High Performance Computing on Stampede, Cornell University Center for Advanced Computing (CAC), vol. 262, 2015.
6. A. Podobas and S. Karlsson, "Towards unifying openmp under the task-parallel paradigm," in Int. Workshop on OpenMP. Springer International Publishing, Springer International Publishing, Nara, Japan, vol. 9903, pp. 116–129, 2016.
7. M. U. Ashraf, F. Fouz and F. A. Eassa, "Empirical analysis of hpc using different programming models," International Journal of Modern Education & Computer Science, vol. 8, no. 6, pp. 27–34, 2016.
8. J. A. Herdman, W. P. Gaudin, S. Smith, M. Boulton, D. A. Beckingsale et al., "Accelerating hydrocodes with openacc, opencl and cuda," High Performance Computing, Networking, Storage and Analysis (SCC), vol. 66, pp. 465–471, 2012.
9. H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang et al., "High performance computing using mpi and openmp on multi-core parallel systems," Parallel Computing, vol. 37, no. 9, pp. 562–575, 2011.
10. M. Marangoni and T. Wischgoll, "Togpu: Automatic source transformation from c++ to cuda using clang/llvm," Electronic Imaging, vol. 1, pp. 1–9, 2016.
11. X. Xie, B. Chen, L. Zou, Y. Liu, W. Le et al., "Automatic loop summarization via path

dependency analysis," IEEE Transactions on Software Engineering, vol. 45, no. 6, pp. 537–557, 2017.

12. M. S. Ahmed, M. A. Belarbi, S. Mahmoudi, G. Belalem and P. Manneback, "Multimedia processing using deep learning technologies, high-performance computing cloud resources, and big data volumes," Concurrency and Computation: Practice and Experience, vol. 32, no. 17, pp. 56–99, 2020

Restrictive language semantics in the Multioberon system

Dagaev Dmitry Viktorovich

Chief Expert, JSC "RASU"

115230, Russia, Moscow, Kashirskoe highway, 3, bldg. 2, p.16

dvdagaev@oberon.org

Abstract. The Oberon-based language and systems in implementation demonstrate a minimalist approach to achieving reliability, significantly different from most software systems that seek to maximize the number of supported functions. The requirements for critical systems for Category A nuclear power plants prohibit the use of even more programming practices. In order to meet the category A requirement of a stable number of iterations, the use of conditional loop operators is prohibited. To ensure ergodicity, the prohibition of the use of dynamic memory and recursion is used. A buffer overflow type vulnerability is closed by prohibiting the system operations module SYSTEM. Restrictions can be set to identify the problem of a fragile base class, type change operations, and the use of nested procedures. It is noted that the transition to the Oberon-07 dialect mainly concerned additional restrictions and fits well into the framework of restrictive semantics. Instead of languages and dialects for each set of requirements, the author proposes an approach of restrictive semantics, in which one language with a system of restrictions is used. A single RESTRICT statement has been introduced into the language as a declaration of restrictions on this module. The Multioberon compiler is implemented with one frontend, including a system of restrictions, and several replaceable backends. The syntactic analysis of the compiler is demonstrated by examples. The strategy of scaling the compiler depending on the system requirements is shown. The novelty of the restrictive semantics approach is the achievement of a set of minimum necessary properties that meet the requirements for the system. The use of the "from limitations" approach by system developers is an advantage, because it declares the really necessary properties of the system, linked to the requirements.

Keywords: Oberon, Component Pascal, ergodicity, SW reliability, restriction, compiler, syntax tree, semantic analysis, modularity, fragile base class

References (transliterated)

1. N.Virt, Yu.Gutknekht. Razrabotka operatsionnoi sistemy i kompilyatora. Proekt Oberon. DMK-Press, 2015.
2. GOST R MEK 60880, Programmnoe obespechenie komp'yuternykh sistem, vypolnyayushchikh funktsii kategorii A, 2009 / GOST R IEC 60880, Software for computer systems performing category A functions, 2009 (in Russian).
3. S. Louise, M. Lemerre, C. Aussagues and V. David. The OASIS Kernel: A Framework for High Dependability Real-Time Systems. In Proc. of the IEEE 13th International Symposium on High-Assurance Systems Engineering, 2011, pp. 95-103.

4. A.Akho, M.Lam, R.Seti, D.Ul'man. Kompilyatory: printsipy, tekhnologii i instrumentarii, vtoroe izdanie, 2008, s. 137-144.
5. Dagaev D.V. O razrabotke Oberon-sistemy s zadannymi svoistvami ergodichnosti. Trudy ISP RAN, tom 32, vyp. 6, 2020 g., str. 67-78. DOI: 10.15514/ISPRAS–2020–32(6)–5
6. Virt N., Algoritmy i struktury dannykh. DMK-Press, 2016.
7. V. N. Gugnin, D. V. Sotnik. Ataka s ispol'zovaniem perepolneniya bufera. Vestnik Nats. tekhn. un-ta "KhPI" : sb. nauch. tr. Temat. vyp. : Informatika i modelirovanie. – Khar'kov : NTU "KhPI". – 2004. – № 34. – S. 52-57.
8. Ermakov I.E. Ob''ektno-orientirovannoe programmirovanie: proyasnenie printsipov? //Ob''ektnye sistemy — 2010: Materialy I Mezhdunarodnoi nauchno-prakticheskoi konferentsii — g. Rostov-na-Donu, Yuzhno-Rossiiskii GTU — 2010. S. 130-135
9. Keller R. Improved Stackmanagement in Active Oberon Kernel / Master Thesis, ETH, march 2006, pp. 40-41.
10. Wirth N. The Programming Language Oberon. Revision 1.10.2013 / 3.5.2016. – pp. 1-17.
11. Virt N., Postroenie kompilyatorov, DMK-Press, 2016.
12. Crelier R. OP2: A portable Oberon Compiler / ETH Zurich, Department Informatik, 1990, pp. 4-19.
13. Szyperski C. Component Software: Beyond Object-Oriented Programming / 01/2002, 2nd edition, Addison Wesley, ISBN: 0-201-745572-0
14. Templ J., Metaprogramming in Oberon, diss. ETH No 10655, 1994, pp. 120-121
15. Pieter J. Muller, The Active Object System Design and Multiprocessor Implementation. Diss. ETH No.14755, for the degree of Doctor of Technical Sciences, ETH Zurich 2002, 197 p.

Direct Rendering of Three-Dimensional Objects Based on Perturbation Functions Using GPUs

Vyatkin Sergei Ivanovich

PhD in Technical Science

Senior Research Fellow, Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences

630090, Russia, g. Novosibirsk, ul. Koptyuga, 1

sivser@mail.ru

Dolgovesov Boris Stepanovich

PhD in Technical Science

Head of the Laboratory, Institute of Automation and Electrometry SB RAS

630090, Russia, g. Novosibirsk, ul. Ak. Koptyuga, 1, kab. 318

bsd@iae.nsk.su

Abstract. The object of the study is a method of direct rendering of complex three-dimensional objects based on perturbation functions using graphics processors, using a variety of streaming multiprocessors. Direct rendering means that the visualization of functionally defined models takes place without their preliminary conversion to other formats, for example, into triangle grids. The research method is based on analytical geometry in space, differential

geometry, interpolation theory and matrix theory, based on mathematical modeling and the theory of computing systems. The main conclusions of the study are: the possibility of direct rendering of functionally specified objects, when rendering it is important that the computing processors are not idle. The first problem that was solved was that different GPUs have different numbers of streaming multiprocessors. Therefore, it was necessary to choose during execution the optimal stage from which the work began. Thus, you can partially get rid of the problem with unused computing resources. The second problem, the balancing problem, was solved by using a large number of computing processors. For implementation, the CUDA parallel programming model was used, which, together with a set of software tools, allows implementing programs in the C language for execution on a GPU. The resulting system visualizes complex functionally defined objects with high resolution interactively. The dependence of performance on the computing power of graphics processors is investigated.

Keywords: acceleration of calculations, parallel programming model, hierarchy of thread groups, parallel computing, streaming multiprocessors, graphics processor, constructive solid-state geometry, direct rendering, perturbation functions, functionally defined object

References (transliterated)

1. Sigg, C. Representation and rendering of implicit surfaces. Diss. ETH No. 16664, Dipl. Rechn. Wiss. ETH Zurich, Switzerland, 2006, pages 162. DOI:10.3929/ETHZ-A-005267203 Corpus ID: 124349594
2. Dekkers, D., Overveld, K.V. Golsteijn, R. Combining CSG modeling with soft blending using Lipschitz-based implicit surfaces. Visual Computer, vol. 20, no. 6, 2004. pp. 380–391.
3. Vyatkin, S. An Interactive System for Modeling, Animating and Rendering of Functionally Defined Objects. American Journal of Computer Science and Engineering Survey, 2014, vol. 2, no. 3. pp. 102-108.
4. Vyatkin, S. Perturbation functions for compact database. Review of computer engineering research. 2017. vol. 4., no. 1. pp. 30–37. doi: 10.18488/journal.76.2017.41.30.37
5. Farin, G., Hoschek, J., Kim, M.S. Handbook of Computer Aided Geometric Design. [electronic resource]. 2002 Elsevier. ISBN 978-0-444-51104-1.
6. Pottmann, H., Brell-Cokcan, S., Wallner, J. Discrete Surfaces for Architectural Design. Archived 2009-08-12 at the Wayback Machine, pp. 213–234 in Curve and Surface Design, Patrick Chenin, Tom Lyche and Larry L. Schumaker (eds.), 2007. Nashboro Press, ISBN 978-0-9728482-7-5.
7. Farin, G. Curves and Surfaces for CAGD. A Practical Guide, Morgan-Kaufmann, 2002 ISBN 1-55860-737-4.
8. Sturm, T. An Algebraic Approach to Offsetting and Blending of Solids in Computer Algebra in Scientific Computing. CASC 2000, V.G. Ganzha, E.W.Mayr and E.V.Vorozhtsov (Eds.), Springer-Verlag, Berlin (2000), pp. 367-381.
9. Stroud, I. Boundary Representation Modelling Techniques. Publisher: Springer, January 2006, ISBN: 978-1-84628-312-3 DOI:10.1007/978-1-84628-616-2
10. Kainz, W., Neufeld, E., Bolch, W.E., Graff, C.G. Advances in Computational Human Phantoms and Their Applications in Biomedical Engineering-A Topical Review. IEEE Transactions on Radiation and Plasma Medical Sciences PP(99):1-1. December 2018 DOI:10.1109/TRPMS.2018.2883437
11. Arioli, C., Shamanskiy, A., Klinkel, S., Simeon, B. Scaled boundary parametrizations in

isogeometric analysis. Computer Methods in Applied Mechanics and Engineering 349, March 2019, DOI:10.1016/j.cma.2019.02.022

12. Leidinger, L.F, Breitenberger, M., Bauer, A.M., Hartmann, S. Explicit dynamic isogeometric B-Rep Analysis of penalty-coupled trimmed NURBS shells. Computer Methods in Applied Mechanics and Engineering 351 April 2019, DOI:10.1016/j.cma.2019.04.016
13. Vyatkin, S.I. Complex Surface Modeling Using Perturbation Functions. Optoelectronics, Instrumentation and Data Processing, vol. 43, no. 3, 2007. pp. 40-47.
14. Reimers, M., Seland, J. Ray Casting Algebraic Surfaces using the Frustum Form. Eurographics, vol. 27 (2008), no. 2, pp. 361-370.
15. Liktor, G. Ray Tracing Implicit Surfaces on the GPU. Computer Graphics and Geometry, vol. 10, no. 3, 2008, pp. 36-53. http://www.cgg-journal.com/2008-3/04.htm

The Eye-Tracking Study of the Film Frame Composition Influence on the Visual Perception

Borevich Ekaterina Vladislavovna

Assistant of the Higher School of Design and Architecture, Peter the Great St. Petersburg Polytechnic University

195251, Russia, Sankt-Peterburg oblast', g. Saint Petersburg, ul. Politekhnicheskaya, 29

plasma5210@mail.ru

Abstract. The research is aimed at studying the elements that affect the visual perception of the film frame in order to develop methodological recommendations for the process harmonization of the film frame. The object of research is a film frame. The subject of the research is the technology of film frame processing. The purpose of this work is to obtain experimental data of the film frame viewing pattern and to identify statistical patterns to confirm or refute the formulated hypothesis. The goal of the study is to conduct an experimental study of the influence of composition on the parameters of the film frame viewing pattern. The influence of the factor of the mutual ratio of the areas of the centers of interest to the background on the parameters of the template for viewing the stimulus material is investigated. As a result, the methodology has been developed for conducting experimental studies of human perception of visual information using an eye-tracking software and hardware complex. When analyzing data on the influence of the objects size factor. The results obtained show that under the condition that objects occupy a small area of the frame, the observer needs more time to consider this frame. As well as in the case when objects occupy most of the frame (more than 40%). In the first case, due to the small size of the objects, it becomes more difficult for the observer to find objects in the frame space. In the second case, it takes time to identify objects, since they tend to be perceived as a background due to their large size.

Keywords: Cinematograph, Human visual system, Gestalt psychology, Statistical analysis of data, Experiment, Color scheme, Eye-tracker, Film frame, Visual appeal, Composition

References (transliterated)

1. Lotman Yu.M. Semiotika kino i problemy kinoestetiki // Lotman Yu.M. Ob iskusstve (SPb., 1998)
2. Jonides J., Irwin D.E., Yantis S. Integrating visual information from successive fixations // Science. 1982. V.215.P. 192-194.

3. Burr D., Morrone M.C. Eye movements: building a stable world from glance to glance //Curr Biol. 2005. V.15. №20.P. 839-840.
4. Mescheryakov S.V., Yanchus V.E., Borevich E.V. Experimental Research of Digital Color Correction Models and Their Impact on Visual Fixation of Video Frames // Humanities and Science University Journal. 2017. V. 27. P. 15-24.
5. Borevich E.V., Yanchus V.E., Mescheryakov S.V. Computer Eye-Tracking Model to Investigate Influence of the Viewer's Perception of the Graphic Information // GraphiCon 2021: trudy 31-i Mezhdunar. konf. po komp'yuternoi grafike i mashinnomu zreniyu. P. 720-728
6. Orlov P.A., Laptev V.V., Ivanov V.M. K voprosu ob ispol'zovanii ai-treking sistem // Nauchno-tekhnicheskie vedomosti SPbGPU. 2014. T. 5 (205). S. 84–94.
7. Yanchus V.E., Borevich E.V. Issledovanie znacheniya tsvetovogo resheniya v protsesse garmonizatsii kinokadra // Nauchno-tekhnicheskii vestnik SPbGPU. 2016. T. 4. S. 53–68.
8. Borevich E.V., Mescheryakov S.V., Yanchus V.E. Statistical Model of Computing Experiment on Digital Color Correction //DCCN.2019.P.140-150.
9. Yanchus V.E., Borevich E.V., Avdeeva A.A. — Primenenie tekhnologii ai-trekinga v voprosakh issledovaniya vospriyatiya graficheskoi informatsii // Programmnye sistemy i vychislitel'nye metody. – 2021. – № 1. DOI: 10.7256/2454-0714.2021.1.33378
10. Strasburger, Gans; Peppel', Ernst (2002). Pole zreniya. V G. Adelman & BH Smith (Eds): Encyclopedia of Neuroscience; 3-e izdanie na kompakt-diske. Elsevier Science BV, Amsterdam, N'yu-Iork.
11. Norman D.A., Draper S.W. User Centered System Design. Erlbaum: London, 1986. 526 p.
12. Majaranta P., Bulling A. Eye Tracking and Eye-Based Human-Computer Interaction // Advances in Physiological ComputingHuman-Computer Interaction Series, 2014. P. 39-59.
13. Glants, S. Mediko-biologicheskaya statistika / Stenton Glants; Per. s angl. d.f.-m.n. Yu.A. Danilova pod red. N.E. Buzikashvili i D.V. Samoilova. – M.: Praktika, 1999. – 459 s.
14. Zheleznyakov V. N. Tsvet i kontrast. Tekhnologiya i tvorcheskii vybor: Uchebnoe posobie. — M.: VGIK, 2001. — 286 s.