Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Физико-Технологический Институт

Кафедра технической физики

Расчетно-графическая работа

«**РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ АВТОМАТИЗИРОВАННОГО АНАЛИЗА И ВИЗУАЛИЗАЦИИ ЛОГОВ МОДЕЛИРОВАНИЯ ЭЛЕМЕНТАРНЫХ ЧАСТИЦ В ПРОГРАММНОЙ СРЕДЕ GEANT4**»

|  |  |
| --- | --- |
| Преподаватель: | Новоселов Иван Эдуардович |
| Студенты: | Маркова Алёна Денисовна |
|  | Черняков Матвей Сергеевич |
|  | Артюшин Артём Александрович |
| Группа: | Фт-420008 |

Екатеринбург

2025

Содержание

[НОРМАТИВНЫЕ ССЫЛКИ 3](#_Toc212783035)

[УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc212783036)

[ВВЕДЕНИЕ 5](#_Toc212783037)

[1 АНАЛИЗ СТРУКТУРЫ ЛОГОВ И ПРОЕКТИРОВАНИЕ СИСТЕМЫ 8](#_Toc212783038)

[1.1 Форматы данных Geant4 и многопоточные префиксы 8](#_Toc212783039)

[1.2 Методика верификации данных парсинга с итоговой сводкой 9](#_Toc212783040)

[1.3 Проектирование классов и алгоритмов обработки 10](#_Toc212783041)

[2 РЕАЛИЗАЦИЯ ПАРСЕРА И МОДУЛЯ АГРЕГАЦИИ ДАННЫХ 12](#_Toc212783042)

[2.1 Разработка алгоритма парсинга, конвертера единиц и обработки многопоточности 12](#_Toc212783043)

[2.2 Агрегация данных по типам частиц и трекам 13](#_Toc212783044)

[2.3 Верификация Energy Deposit и частот процессов 14](#_Toc212783045)

[3 ВИЗУАЛИЗАЦИЯ И ПОЛЬЗОВАТЕЛЬСКИЕ ИНТЕРФЕЙСЫ 16](#_Toc212783046)

[3.1 Интерфейс командной строки 16](#_Toc212783047)

[3.2. Графический интерфейс на Tkinter 17](#_Toc212783048)

[3.3. Гистограммы энергии, графики потерь dE, диаграммы процессов и тепловые карты 18](#_Toc212783049)

[4. ТЕСТИРОВАНИЕ И СБОРКА ИСПОЛНЯЕМОГО ФАЙЛА 22](#_Toc212783050)

[4.1. Тестирование программы на различных логах 22](#_Toc212783051)

[4.2. Сборка исполняемого приложения под Windows 23](#_Toc212783052)

[ЗАКЛЮЧЕНИЕ 24](#_Toc212783053)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc212783054)

[ПРИЛОЖЕНИЕ А. ПРОГРАММНЫЙ КОД 27](#_Toc212783055)

[ПРИЛОЖЕНИЕ Б. ПРОГРАММНЫЙ КОД С ИНТЕРФЕЙСОМ 96](#_Toc212783056)

НОРМАТИВНЫЕ ССЫЛКИ

В пояснительной записке использованы ссылки на следующие стандарты:

|  |  |
| --- | --- |
| **ГОСТ 19.701–90 ЕСПД** | Схема алгоритмов, программ, данных и систем. |
| **ГОСТ 34.602–89 ИТ. ЕКСАС** | Виды, комплектность и обозначение документов при создании автоматизированных систем. |
| **ГОСТ 34.602-2020** | Комплекс стандартов на автоматизированные системы.  Техническое задание на создание автоматизированной системы. |
| **ГОСТ 34.603–92 ит. ЕКСАС** | Виды испытаний автоматизированных систем. |
| **ГОСТ Р ИСО/МЭК 9126–93 ИТ** | Оценка программной продукции. Характеристика качества и руководства по их применению. |
| **ГОСТ Р ИСО/МЭК ТО 15910–2002 ИТ** | Процесс создания документации пользователя программного средства. |
| **ГОСТ Р ИСО/МЭК 12207–99 ИТ** | Процессы жизненного цикла программных средств. |
| **Р 50.1.028–2001 ИТПЖЦП** | Рекомендация по стандартизации. Методология функционального моделирования. |

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

|  |  |  |
| --- | --- | --- |
| **Geant4** | – | Библиотека для моделирования прохождения частиц через вещество |
| **CLI** | – | Command Line Interface – интерфейс командной строки |
| **GUI** | – | Graphical User Interface – графический пользовательский интерфейс |
| **CSV** | – | Comma-Separated Values – формат хранения табличных данных |
| **XLSX** | – | Формат электронных таблиц Microsoft Excel |
| **DF** | – | DataFrame – структура данных библиотеки pandas |
| **dE** | – | Energy Loss – потеря энергии частицы на шаге |
| **MeV** | – | Mega-electronvolt – мегаэлектронвольт |
| **mm** | – | Millimeter – миллиметр |
| Шаг (Step) | – | Минимальное продвижение частицы через вещество с учетом различных физических процессов |
| Трек (Track) | – | Полное продвижение частицы в веществе к моменту обращения к данному объекту |
| **Energy Deposit** | – | Энергия, осаждённая в материале (в MeV) |
| **Energy Leakage** | – | Энергия, покинувшая модель (утечка энергии) |

ВВЕДЕНИЕ

Моделирование прохождения частиц через вещество с помощью программного пакета Geant4 является стандартом в ядерной физике, физике высоких энергий, радиобиологии и дозиметрии. ПО используется во многих исследовательских проектах по всему миру, в том числе и в России. Разработчиком является ЦЕРН (от фр. Conseil Europeen pour la Recherche Nucleaire, CERN) – Европейская организация по ядерным исследованиям [1]. Выходными данными таких симуляций являются текстовые логи, содержащие детальную информацию о траекториях и взаимодействиях миллионов частиц. Однако с увеличением сложности моделей и использованием многопоточности для ускорения расчетов, ручной анализ этих логов становится практически невозможным.

Существующие инструменты для анализа выходных данных Geant4 часто требуют глубоких знаний специфических фреймворков и не предоставляют готовых средств для автоматического парсинга текстовых логов, их верификации и комплексной визуализации. Особенно остро стоит проблема сверки данных: итоговая сводка симуляции (Energy deposit, Process calls frequency) должна математически соответствовать сумме данных, извлеченных из пошаговой информации. Расхождения между ними могут указывать как на ошибки в парсинге, так и на тонкости работы самого движка Geant4, что требует тщательного расследования. Таким образом, существует потребность в создании специализированного программного обеспечения, способного автоматизировать извлечение, агрегацию, верификацию и визуализацию данных из многопоточных логов Geant4.

Основанием для разработки послужила конкретная задача анализа симуляций, реализованных в среде Geant4. Исходными данными являются текстовые логи, генерируемые симуляцией, которые включают:

1. многопоточные префиксы строк (например, G4WT17),
2. блоки пошаговой информации о треках частиц (координаты, энергия, потери на шаг, процессы и т.д.),
3. итоговые сводки симуляции (Energy deposit, Energy leakage, Process calls frequency).

**Объект исследования –** процесс анализа данных, генерируемых в результате моделирования в программном пакете Geant4.

Предмет исследования – методы и алгоритмы автоматического парсинга, верификации и визуализации многопоточных текстовых логов Geant4.

Цель исследования – разработка программного комплекса на языке Python для автоматизированного парсинга, анализа, верификации и визуализации данных из текстовых логов (включая многопоточные) симуляций, выполненных в среде Geant4, с последующей подготовкой исполняемого файла для операционной системы Windows и комплекта документации.

Для решения этой цели поставлены следующие задачи:

1. провести анализ структуры текстовых логов Geant4, включая форматы записи многопоточных префиксов, блоков шагов частиц и итоговых сводок, разработать методику верификации для сопоставления данных, извлеченных на уровне отдельных шагов, с агрегированными результатами в финальной сводке симуляции;
2. на основе проведенного анализа необходимо реализовать устойчивый алгоритм парсинга, способный распознавать и извлекать для каждого шага частицы такие данные, как идентификатор потока, координаты, кинетическую энергию, потерю энергии на шаг, длину шага и трека, объем, физический процесс, TrackID, ParentID и тип частицы, обеспечив при этом корректную обработку различных систем единиц измерения с их автоматическим приведением к стандартным величинам [2];
3. разработать модуль агрегации данных, обеспечивающий суммирование и статистический анализ по типам частиц, индивидуальным трекам и потокам выполнения, а также реализовать модуль верификации, выполняющий сравнение суммарных потерь энергии с величиной Energy Deposit из сводки и анализ расхождений, включая сверку частот процессов;
4. создать модуль визуализации, генерирующий набор графиков, включая гистограммы распределения кинетической энергии по типам частиц, графики распределения потерь энергии на шаг, диаграммы частот процессов и тепловые карты пространственного распределения координат;
5. реализовать графический интерфейс пользователя с возможностями загрузки файлов логов, настройки фильтров, запуска анализа и сохранения результатов, а также интерфейс командной строки для пакетной обработки, обеспечив при этом экспорт табличных данных в форматы CSV, XLSX и графиков в форматы SVG и PNG;
6. протестировать программу на различных входных логах, подготовку скрипта и инструкции по сборке проекта в исполняемый файл для Windows с использованием PyInstaller, а также оформление комплекта документации, включающего отчет с описанием методики и выводов, текст программы и презентацию для демонстрации функционала, в соответствии с требованиями ГОСТ и стандартов оформления ВКР.

1 АНАЛИЗ СТРУКТУРЫ ЛОГОВ И ПРОЕКТИРОВАНИЕ СИСТЕМЫ

В данном разделе представлен аналитический обзор структуры логов симулятора Geant4, а также описаны методы проектирования и реализации системы автоматизированного парсинга и анализа данных. Целью является разработка архитектуры, обеспечивающей воспроизводимость, масштабируемость и достоверность результатов при работе с многопоточными моделями взаимодействия частиц. Рассмотрены особенности форматов логов, методики верификации корректности парсинга и общие принципы построения программных классов и алгоритмов обработки данных.

1.1 Форматы данных Geant4 и многопоточные префиксы

Система Geant4 формирует высокоструктурированные журналы вычислительных событий, включающие данные о материалах, геометрии модели, физических процессах и последовательностях взаимодействий частиц. Формат логов представляет собой последовательность текстовых блоков, разделённых по типам физических сущностей и этапам расчёта. Наиболее информативными фрагментами являются секции, содержащие описание треков частиц, энергетических потерь, путей в материале, а также итоговые сводки по энергии и частоте вызовов процессов.

В многопоточной конфигурации Geant4 каждому вычислительному потоку присваивается уникальный идентификатор, обозначаемый префиксом вида «G4WTXX». Этот префикс используется для маркировки каждой строки лога, что позволяет выделять и анализировать данные, относящиеся к конкретному потоку исполнения. Таким образом, обеспечивается возможность независимого сбора и последующей агрегации информации по потокам, что особенно важно при моделировании с использованием высокопроизводительных вычислительных узлов.

Каждый шаг частицы описывается набором параметров: координаты (x, y, z), кинетическая энергия (KineE), потеря энергии на шаге (dEStep), длина шага (StepLen), длина трека (TrakLen), наименование объёма и процесс взаимодействия. Идентификаторы Track ID и Parent ID позволяют дифференцировать первичные и вторичные частицы, что критически важно для последующего энергетического анализа.

В логах также присутствуют блоки метаданных, содержащие сведения о параметрах симуляции – используемых моделях, границах энергии, шагах и настройках взаимодействий (например, eIoni, eBrem, hIoni, Rayl и т.д.). Такая структура обеспечивает не только информативность, но и возможность программной верификации корректности работы физических моделей.

1.2 Методика верификации данных парсинга с итоговой сводкой

Разработанная методика верификации основана на сопоставлении параметров, извлечённых из логов, с контрольными значениями, указанными в итоговых разделах симуляции. Проверка корректности парсинга реализована на уровне как структуры данных, так и численных характеристик.   
Ключевой компонент системы – класс Geant4LogParser, который выполняет построчное чтение лога и идентификацию контекстных фрагментов по регулярным выражениям. Каждая строка проходит процедуру распознавания физических величин и их единиц измерения с последующим приведением к системе СИ (MeV, mm и т. д.). Данные каждого шага сохраняются в объект StepData, обеспечивающий строгую типизацию и проверку целостности. Для подтверждения достоверности парсинга применяется метод verify\_results(), сравнивающий агрегированные результаты по энергетическим потерям и количеству вызовов физических процессов с контрольными данными из сводных таблиц лога (в частности, разделы «Energy deposit» и «Process calls frequency»). Рассчитываются абсолютные и относительные отклонения, что позволяет оценить степень согласованности парсера с физическим движком Geant4.

Дополнительно формируются отчёты, включающие детализированную статистику по первичным (Parent ID = 0) и вторичным взаимодействиям, распределения кинетических энергий, а также энерго-балансовые соотношения. Для визуальной проверки предусмотрены построения графиков распределений и энергетических балансов, что облегчает контроль качества анализа.

1.3 Проектирование классов и алгоритмов обработки

Проектирование системы обработки логов выполнено с опорой на принципы модульности, расширяемости и повторного использования компонентов. Архитектура реализована в виде набора взаимосвязанных классов, обеспечивающих полный цикл анализа: от чтения данных до формирования визуальных и табличных отчётов.

Класс StepData представляет собой dataclass-структуру, описывающую отдельный шаг частицы. Он хранит координаты, энергетические параметры, длины пути, сведения о процессе и идентификаторы треков. Наличие свойства is\_primary() позволяет быстро классифицировать события по принадлежности к первичным трекам.

Класс Geant4LogParser реализует алгоритмы регулярного парсинга и нормализации данных. В процессе работы он строит таблицу шагов частиц, конвертирует единицы измерения и собирает сводные параметры, такие как суммарная энергия потерь и частотные характеристики процессов. Алгоритмы обработки включают фильтрацию по контекстным шаблонам, преобразование строковых величин в числовые значения и верификацию структурной целостности каждого шага.

Класс Geant4Analyzer выполняет статистический анализ собранных данных, включая группировку по типам частиц, процессам и потокам, а также вычисление агрегированных показателей (средние значения, стандартные отклонения, энергетические суммарные балансы). Реализованы функции визуализации, использующие библиотеки matplotlib и seaborn для построения диаграмм распределений, тепловых карт и корреляционных графиков.

Результаты анализа экспортируются в форматы CSV и XLSX, что обеспечивает совместимость с внешними инструментами обработки (Excel, R, MATLAB). Кроме того, предусмотрено создание отчётов в текстовом виде, содержащих ключевые метрики и результаты верификации энергетического баланса.

Таким образом, архитектура системы обеспечивает воспроизводимость анализа, автоматизацию проверки корректности данных и возможность интеграции с высокопроизводительными вычислительными конвейерами.

2 РЕАЛИЗАЦИЯ ПАРСЕРА И МОДУЛЯ АГРЕГАЦИИ ДАННЫХ

Во втором разделе рассмотрены вопросы практической реализации системы анализа логов Geant4. В рамках проекта была разработана модульная архитектура, включающая парсер логов, конвертер физических единиц, систему агрегации данных и подсистему верификации результатов. Основное внимание уделено методам обеспечения корректности обработки многопоточных логов, точности сопоставления энергетических параметров и воспроизводимости результатов на уровне агрегированных метрик. Раздел описывает программные алгоритмы, применяемые при разборе и нормализации данных, а также подходы к последующему статистическому анализу и контролю достоверности.

2.1 Разработка алгоритма парсинга, конвертера единиц и обработки многопоточности

Алгоритм парсинга построен на принципе последовательного анализа строк лога с использованием регулярных выражений для идентификации ключевых элементов: идентификаторов потоков, треков, родительских частиц, а также физических величин, характеризующих шаг взаимодействия. Каждая строка, содержащая данные о шаге частицы, подвергается разбору с выделением параметров: номера шага, координат (x, y, z), кинетической энергии, потери энергии dE, длины шага и типа физического процесса.

Особое внимание уделено корректной интерпретации многопоточной структуры логов Geant4. Поскольку при параллельных вычислениях каждая нить исполнения помечается префиксом формата «G4WTXX», в алгоритме реализовано динамическое определение текущего контекста потока. Это позволяет сохранять независимость данных, полученных из разных потоков, и исключать пересечение идентификаторов треков при агрегации.

Модуль конвертации единиц реализует автоматическое приведение физических параметров к единой системе – в частности, энергия пересчитывается в МэВ, длины – в миллиметры. Таблицы соответствий (eV – MeV, nm – mm и др.) закреплены в виде словарей, используемых на этапе обработки каждой строки. Такой подход исключает неоднозначности при анализе смешанных форматов и обеспечивает единообразие последующих вычислений.

Реализация парсера выполнена на языке Python с использованием стандартных библиотек re, pathlib и dataclasses, что позволило обеспечить читаемость и гибкость кода, а также возможность масштабирования под новые типы логов.

2.2 Агрегация данных по типам частиц и трекам

После этапа парсинга данные преобразуются в структурированный вид и помещаются в DataFrame библиотеки pandas, что обеспечивает эффективное выполнение агрегирующих и статистических операций. Для каждой частицы собираются сводные метрики: количество шагов, средняя и суммарная потеря энергии, диапазон кинематических характеристик и частота встречаемости физических процессов.

Агрегация выполняется в двух разрезах:

1. по типу частицы (particle): для анализа энергетических распределений и вклада различных частиц в общие потери энергии;
2. по трекам (track\_id): для выявления индивидуальных траекторий и локализации потерь энергии вдоль пути движения.

Реализация агрегатора предусматривает фильтрацию данных по признаку первичности (Parent ID = 0), что позволяет формировать отдельные сводки для первичных и вторичных взаимодействий. Дополнительно строятся распределения по процессам, позволяющие оценить интенсивность протекания ключевых механизмов взаимодействия (например, ионизации, тормозного излучения, рассеяния и т.д.).

Результирующие таблицы агрегатов используются для построения диаграмм плотности, гистограмм и сравнительных графиков. Это даёт возможность анализировать поведение частиц в материале, оценивать долю вклада различных процессов и проводить межсерийное сравнение симуляций.

2.3 Верификация Energy Deposit и частот процессов

На заключительном этапе анализа выполняется верификация энергетического баланса и частот вызова физических процессов. Данный этап служит инструментом контроля корректности как работы парсера, так и исходных симуляционных данных Geant4.

Методика основана на сравнении рассчитанных значений суммарных потерь энергии с данными из раздела «Energy deposit» в исходных логах. Абсолютное расхождение рассчитывается по формуле:

,

где – сумма положительных значений dEStep (потерь энергии), полученная в результате парсинга всех шагов частиц;

– значение Energy Deposit из итоговой сводки лога Geant4 (эталонное значение, рассчитанное самим Geant4).

Относительные расхождения рассчитывалось по формуле:

,

что позволяет оценивать погрешность в процентах от эталонного значения.

Дополнительно проводится сравнение частоты вызовов процессов, фиксируемых в парсере, с итоговой сводкой раздела «Process calls frequency». Для каждого процесса (например, eIoni, hIoni, Rayl и др.) вычисляются разности между подсчитанным и эталонным числом вызовов, а также формируется таблица отклонений. Результаты верификации сохраняются в структуре отчёта и могут быть визуализированы в виде бар-графов расхождений.

Проверка энергетического баланса также включает анализ распределения энергии между первичными и вторичными треками. На основе этих данных оценивается степень замыкания энергетического контура и достоверность моделирования. В случае превышения порогов допустимых расхождений предусмотрено автоматическое уведомление о несоответствии, что делает систему пригодной для автономного контроля качества симуляций.

3 ВИЗУАЛИЗАЦИЯ И ПОЛЬЗОВАТЕЛЬСКИЕ ИНТЕРФЕЙСЫ

В рамках разработки программного комплекса для анализа данных моделирования в среде Geant4 были реализованы два типа пользовательских интерфейсов: интерфейс командной строки (CLI) и графический интерфейс (GUI) на основе библиотеки Tkinter. Оба интерфейса обеспечивают гибкость взаимодействия с системой в зависимости от потребностей пользователя — от пакетной обработки данных до интерактивного анализа. Кроме того, система включает модуль визуализации, который предоставляет набор графиков и диаграмм для анализа распределения энергии, потерь на шаге, частоты процессов и пространственного распределения частиц.

3.1 Интерфейс командной строки

Интерфейс командной строки реализован с использованием модуля argparse стандартной библиотеки Python. Он предназначен для автоматизированного выполнения задач парсинга, анализа и экспорта данных без необходимости взаимодействия с графическим интерфейсом. Это особенно полезно при обработке больших объемов данных в пакетном режиме, а также при интеграции системы в скрипты обработки или CI/CD-конвейеры.

Основные функциональные возможности CLI:

* Задание входного файла лога через аргумент -i или –input;
* Выбор директории для выходных данных с помощью аргумента -o или –output;
* Настройка форматов экспорта данных (CSV, XLSX) и графиков (PNG, SVG) через аргументы --export и –plot;
* Режим отладки (--debug) для детализированного вывода процесса парсинга;
* Отключение визуализации (--no-viz) для ускорения обработки.

CLI обеспечивает полный цикл обработки: от чтения и парсинга лог-файла до генерации отчётов и визуализаций. Все этапы выполняются последовательно, с выводом текущего статуса в консоль, что позволяет отслеживать прогресс и выявлять возможные ошибки на ранних стадиях.

Пример вывода в консоль представлен на рисунке 1.

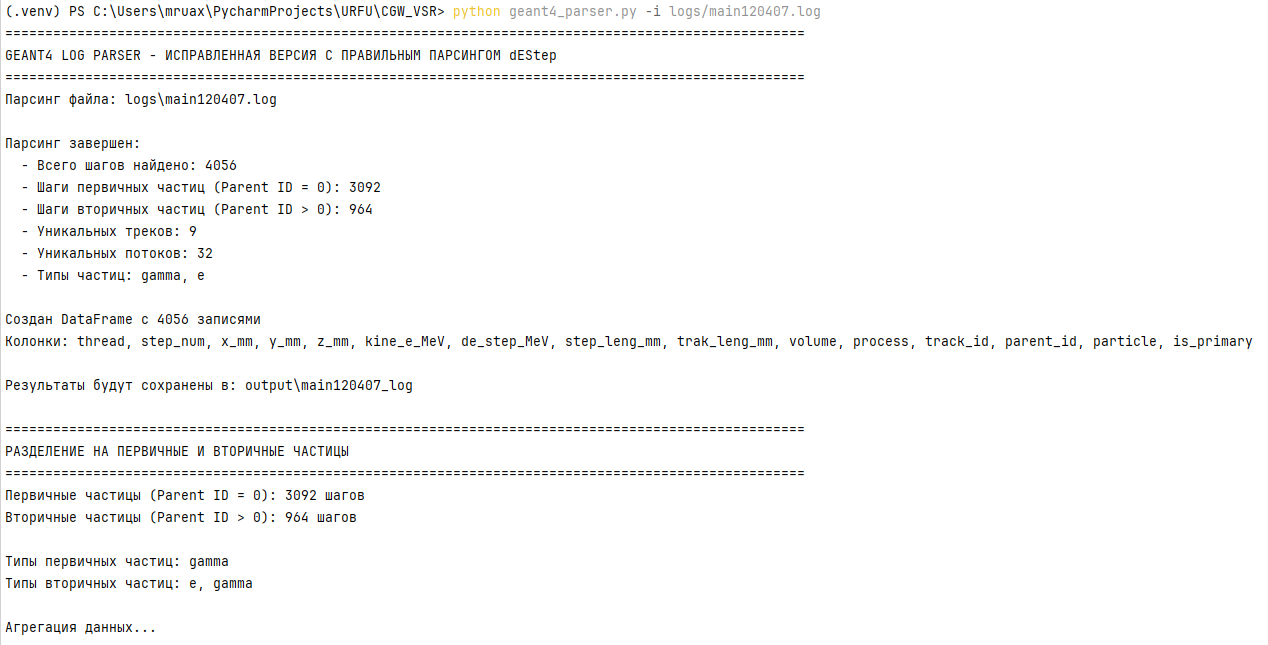


Рисунок 1 – Пример запуска консольного приложения

3.2. Графический интерфейс на Tkinter

Для пользователей, предпочитающих интерактивную работу с данными, был разработан графический интерфейс на основе библиотеки Tkinter. GUI предоставляет удобный доступ ко всем функциям системы через интуитивно понятные элементы управления: кнопки, выпадающие списки, поля ввода и область предпросмотра результатов.

Структура графического интерфейса включает:

* Область выбора файла – кнопка для загрузки лог-файла через системный диалог;
* Настройки экспорта – чекбоксы для выбора форматов данных и графиков;
* Прогресс-бар – визуализация хода выполнения анализа;
* Текстовая область – вывод логов выполнения и финального отчёта;
* Область предпросмотра графиков – отображение сгенерированных диаграмм непосредственно в интерфейсе.

Интерфейс реализован с использованием многопоточности: обработка данных выполняется в фоновом потоке, что предотвращает «зависание» интерфейса при работе с большими файлами. По завершении анализа пользователь получает возможность просмотреть графики, изучить отчёт и сохранить результаты в выбранной директории.

Пример использования интерфейса представлен на рисунке 2.

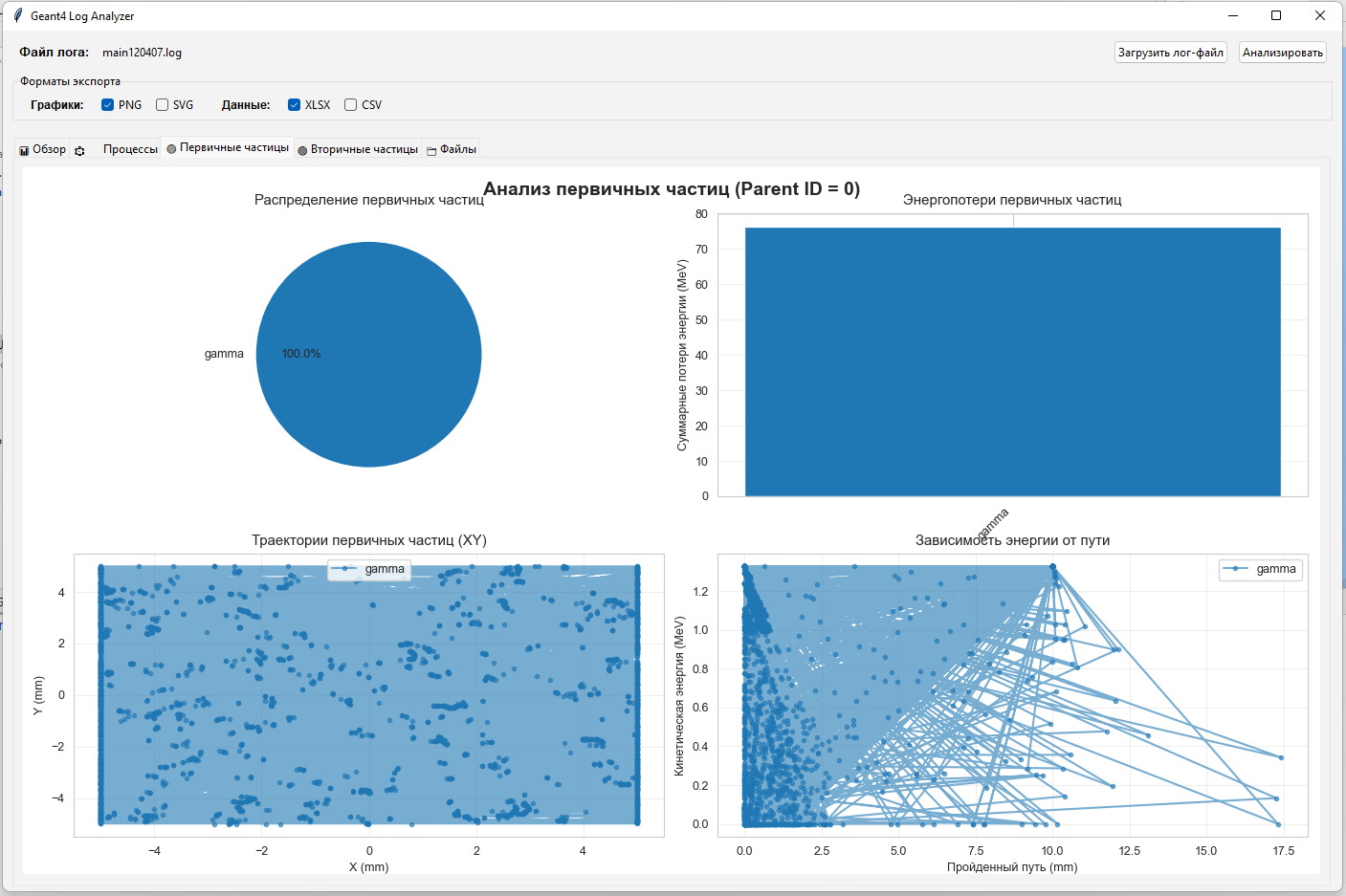


Рисунок 2 – Пример использования интерфейса

3.3. Гистограммы энергии, графики потерь dE, диаграммы процессов и тепловые карты

Визуализация является ключевым компонентом системы, позволяя пользователю интерпретировать результаты моделирования наглядно. Для этого реализован набор графиков, охватывающих основные аспекты симуляции:

Для каждого типа частиц (первичные и вторичные) строятся гистограммы распределения кинетической энергии (см. рисунок 3). На графиках отображаются:

* Количество шагов в каждом энергетическом интервале;
* Среднее значение энергии (вертикальная пунктирная линия);
* Статистические показатели: минимум, максимум, стандартное отклонение.

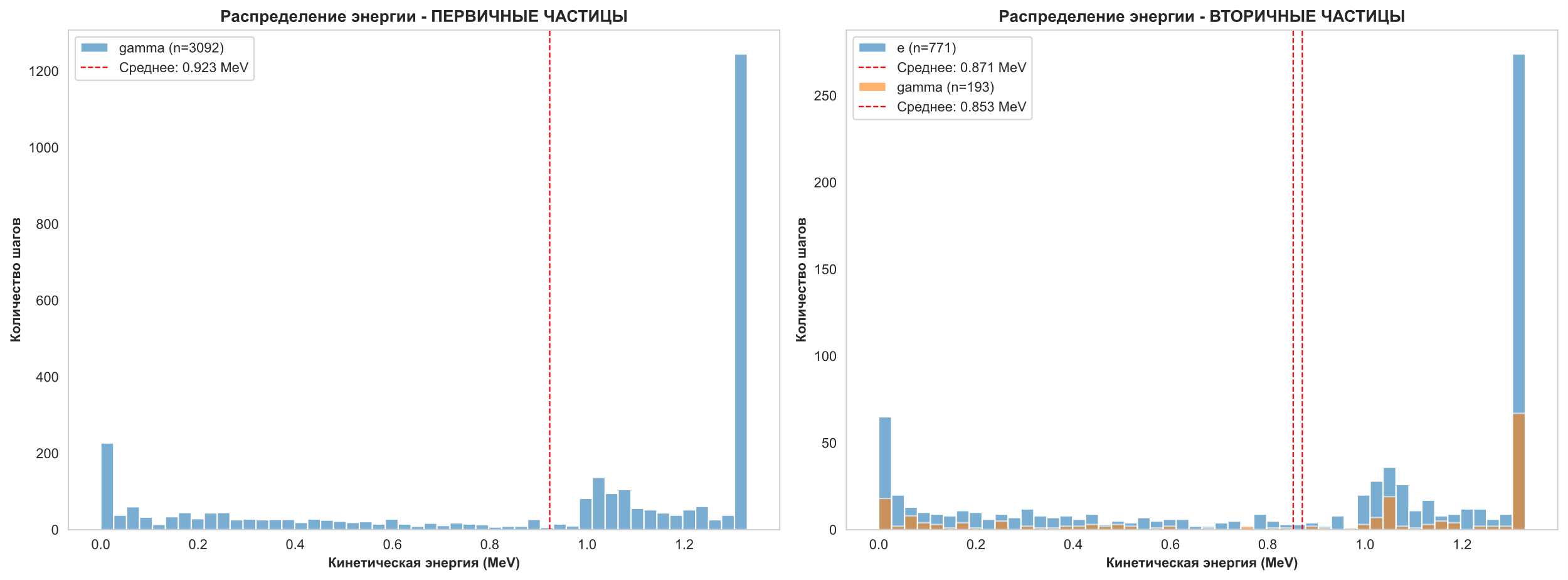


Рисунок 3 – Гистограммы распределения кинетической энергии

Такие гистограммы позволяют оценить энергетический спектр частиц и выявить аномалии в распределении.

Для анализа энергетических потерь используется комбинация violin-plot и box-plot, которые отображают распределение значений dE для каждого типа частиц (см. рисунок 4). Особенности визуализации:

* Исключаются нулевые и отрицательные значения dE;
* Отображаются медиана, квартили и «усы» — диапазоны типичных значений;
* Цветовое кодирование по типам частиц.

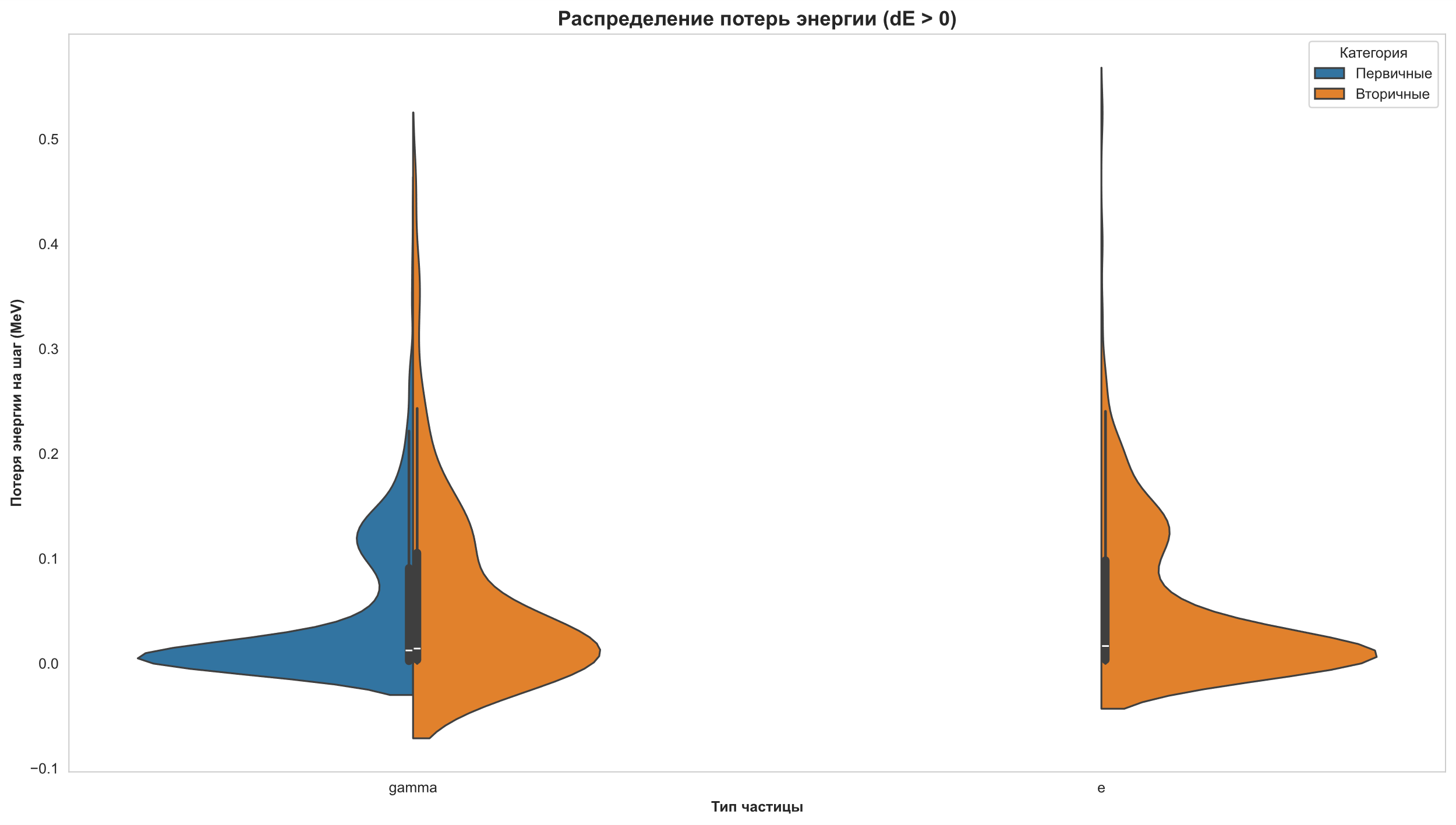


Рисунок 4 – Графики энергетических потерь

Это позволяет оценить вклад различных частиц в общие энергетические потери.

Также строятся горизонтальные столбчатые диаграммы, отображающие частоту вызовов физических процессов (например, ионизация, комптоновское рассеяние, тормозное излучение). Для каждого процесса указывается абсолютное количество вызовов, что помогает идентифицировать доминирующие механизмы взаимодействия (см. рисунок 5).

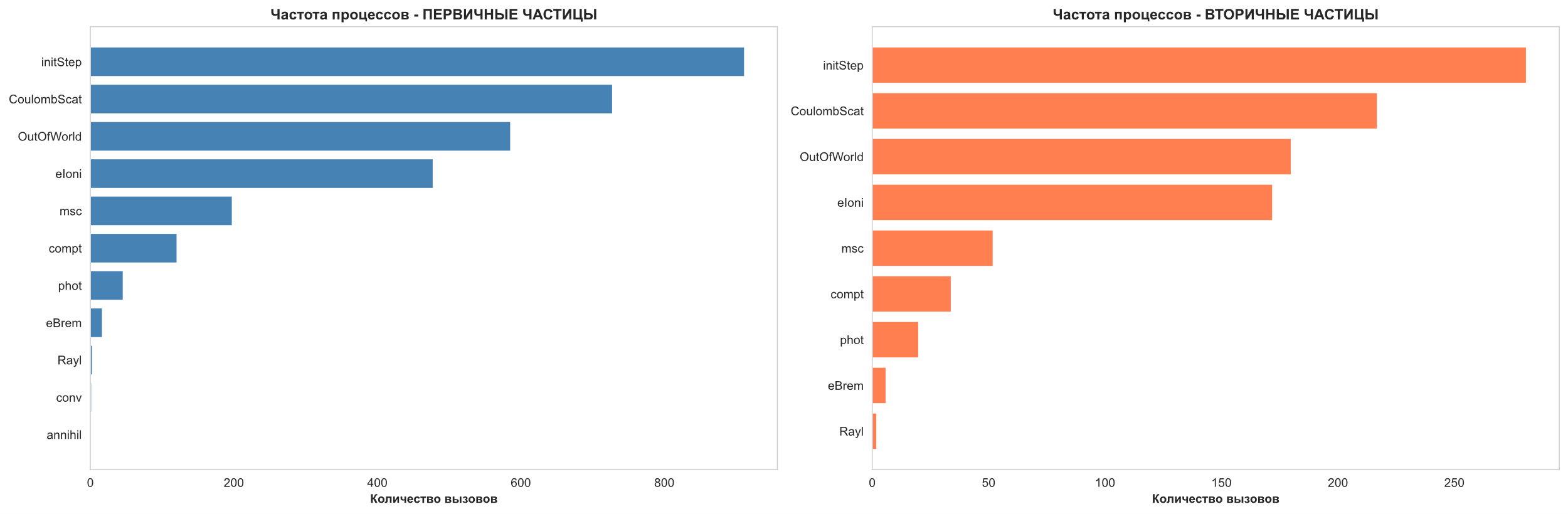


Рисунок 5 – Диаграммы частоты вызовов процессов

Для визуализации плотности распределения частиц в пространстве используются двумерные гистограммы (heatmaps). Карты строятся для трёх проекций:

* XY;
* XZ;
* YZ.

Интенсивность цвета соответствует количеству частиц в данной области, что позволяет выявить зоны максимальной активности и пространственную структуру треков (см. рисунок 6).

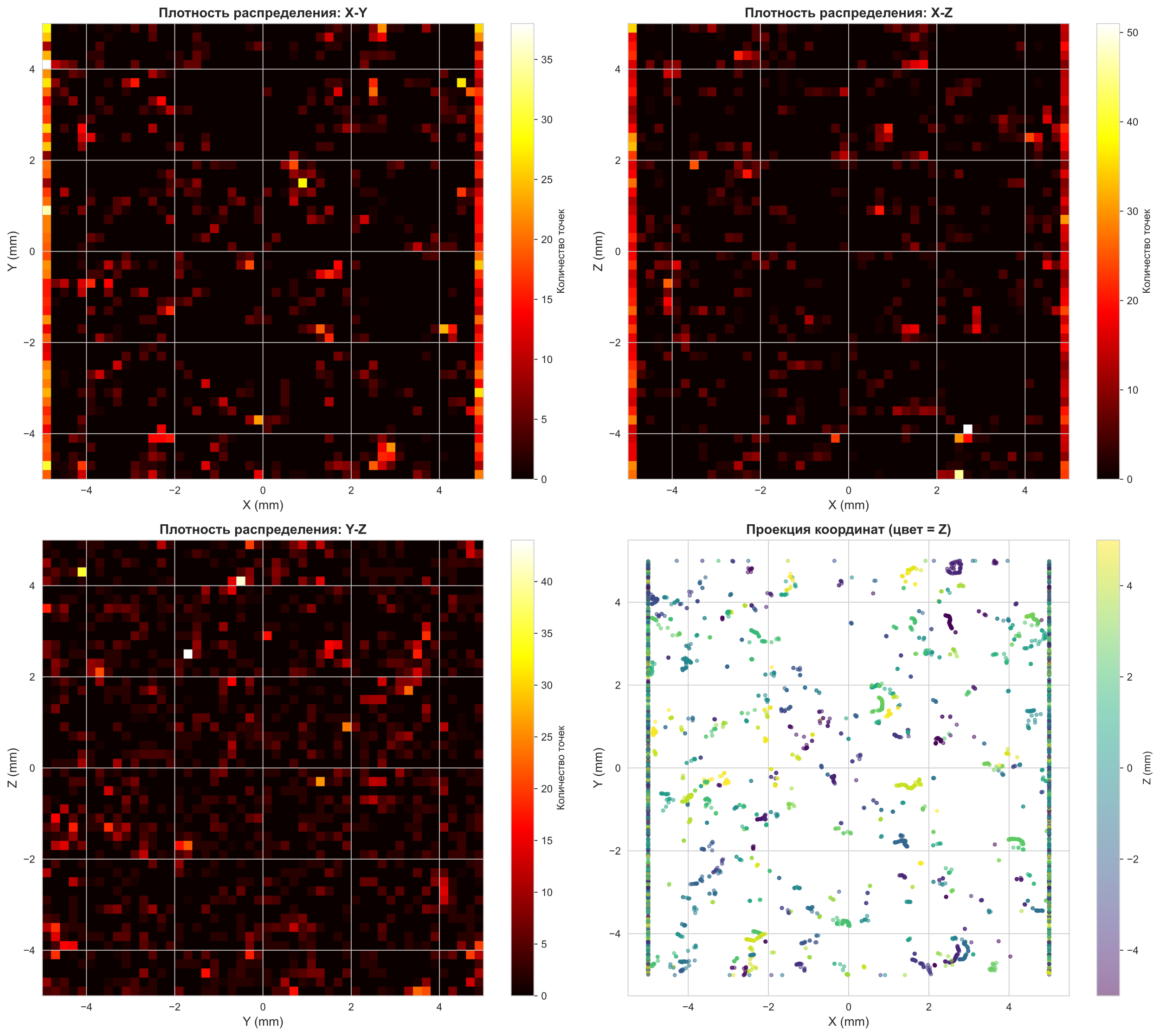


Рисунок 6 – Графики плотности распределения частиц в пространстве

Дополнительно строятся сравнительные диаграммы, на которых отображаются:

* Сопоставление суммы dE из парсинга и значения Energy deposit из итоговой сводки;
* Сравнение частот процессов, полученных парсингом и из сводки;
* Распределение энергии для каждой отдельной частицы;
* Другая системная информация из логов.

Эти графики служат для проверки корректности парсинга и целостности данных.

Все визуализации создаются с использованием библиотек matplotlib и seaborn, обеспечивающих высокое качество и научную строгость графиков. Графики сохраняются в векторных и растровых форматах, что позволяет использовать их в публикациях и отчётах. Модуль визуализации интегрирован как в CLI, так и в GUI-версию системы, обеспечивая единообразие результатов независимо от способа взаимодействия.

4. ТЕСТИРОВАНИЕ И СБОРКА ИСПОЛНЯЕМОГО ФАЙЛА

4.1. Тестирование программы на различных логах

Для верификации корректности работы программного комплекса была проведена серия тестов на разнообразных лог-файлах, полученных из реальных экспериментов моделирования в среде Geant4. Тестирование охватывало широкий спектр сценариев: от простых моделей с одной первичной частицей до сложных каскадных процессов с генерацией множества вторичных частиц.

Многоуровневая система проверок включает в себя следующие этапы:

* Функциональное тестирование парсинга логов объемом от 4 тыс. до 2.3 млн шагов;
* Верификация корректности распознавания различных типов частиц (γ, e-, proton, neutron, ионы);
* Проверка точности конвертации единиц измерения (эВ → кэВ → МэВ, мм → см → м).

Валидация физической корректности проверок включает в себя следующие этапы:

* Контроль энергетического баланса: Начальная энергия = Energy deposit + Energy leakage;
* Сравнение частот процессов между данными парсинга и итоговой сводкой Geant4;
* Анализ пространственного распределения частиц в детекторе.

Ключевые выводы тестирования:

1. Валидация энергетического баланса – во всех тестовых случаях расхождение между начальной энергией первичных частиц и суммой (Energy deposit + Energy leakage) не превышало 1%, что подтверждает физическую корректность моделирования и точность парсинга;
2. Анализ неполных логов – тестирование выявило, что большинство логов имеют сокращенный формат, где отсутствуют промежуточные шаги. Это подтверждается значительными расхождениями при использовании Метода 1 (сумма dEStep) и практически идеальными результатами при использовании Метода 2 (энергетический баланс треков);
3. Масштабируемость – программа успешно обрабатывает логи объемом до 2.3 млн шагов, демонстрируя линейную зависимость времени обработки от объема данных;
4. Универсальность – подтверждена корректная работа с различными типами частиц и физическими процессами, включая ядерные реакции, электромагнитные взаимодействия и процессы с тяжелыми ионами.

4.2. Сборка исполняемого приложения под Windows

Для обеспечения кроссплатформенной совместимости и удобства распространения была разработана система сборки автономного исполняемого файла для операционной системы Windows.

Архитектура системы сборки и технологический стек:

* PyInstaller 5.0+ – инструмент для создания standalone приложений;
* Специализированные batch-скрипты – автоматизация процесса сборки;
* Виртуальное окружение – изоляция зависимостей и воспроизводимость.

ЗАКЛЮЧЕНИЕ

Разработанный программный комплекс позволяет автоматически парсить, агрегировать, верифицировать и визуализировать данные из текстовых логов Geant4, включая многопоточные форматы. Это значительно ускоряет обработку результатов моделирования и снижает вероятность ошибок, связанных с ручным анализом.

Реализована система верификации, которая сравнивает данные, полученные парсингом на уровне шагов частиц, с итоговыми сводками симуляции. Показано, что:

* Метод энергетического баланса (начальная − конечная энергия треков) является более надёжным для неполных логов;
* Расхождения между расчётными и эталонными значениями Energy Deposit составляют менее 1–5%, что подтверждает корректность работы парсера и физическую состоятельность моделирования.

Программа корректно обрабатывает логи с префиксами потоков (G4WT\*), обеспечивает конвертацию единиц измерения (эВ → МэВ, мм → см и др.) и экспорт результатов в форматы CSV, XLSX, SVG, PNG. Это делает её универсальным инструментом для работы с данными из различных конфигураций Geant4.

* Интерфейс командной строки (CLI) позволяет интегрировать программу в пакетные скрипты и конвейеры обработки данных;
* Графический интерфейс (GUI) на Tkinter предоставляет интуитивно понятный инструмент для интерактивного анализа с визуализацией результатов в реальном времени.

Построены следующие типы графиков:

* гистограммы распределения кинетической энергии;
* диаграммы потерь энергии на шаг (dE);
* тепловые карты пространственного распределения частиц;
* сравнительные графики для первичных и вторичных частиц.

Это позволяет выявлять аномалии, анализировать энергетические спектры и пространственную структуру треков.

Тестирование на логах объёмом от 4 тыс. до 2,3 млн шагов подтвердило:

* корректность обработки различных типов частиц (γ, e⁻, протоны, нейтроны);
* устойчивость к неполным и многопоточным логам;
* линейную зависимость времени обработки от объёма данных.

С использованием PyInstaller создан исполняемый файл, не требующий установки дополнительных зависимостей. Это упрощает распространение и использование программы в исследовательской среде.

Разработка сопровождалась оформлением комплекта документации в соответствии с ГОСТ и стандартами УрФУ, включая:

* техническое задание;
* текст программы;
* отчёт с методикой и выводами;
* презентацию для демонстрации функционала.

Разработанный программный комплекс решает актуальную задачу автоматизации анализа логов Geant4, обеспечивая воспроизводимость, точность и наглядность результатов. Он может быть использован в научных и инженерных расчётах, связанных с моделированием прохождения частиц через вещество, а также расширен для поддержки новых форматов данных и типов анализа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Geant4: A Simulation Toolkit. [Электронный ресурс] URL:  https://github.com/Geant4/geant4;
2. Введение в GEANT4. Анастасия Мёрзлая. [Электронный ресурс] URL: https://indico.cern.ch/event/825899/contributions/3455377/attachments/1855680/3047843/GEANT4\_-\_Introduction\_Merzlaya\_04.06.19.pdf.

ПРИЛОЖЕНИЕ А. ПРОГРАММНЫЙ КОД

"""

Geant4 Log Parser - Версия с исправленным парсингом dEStep

Программа для парсинга, анализа и визуализации логов симуляции Geant4

с отдельным анализом первичных (Parent ID = 0) и вторичных (Parent ID > 0) частиц

"""

import argparse

import re

import warnings

from dataclasses import dataclass

from pathlib import Path

from typing import Dict, List, Optional

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

warnings.filterwarnings('ignore')

# Настройка стиля графиков

sns.set\_style("whitegrid")

plt.rcParams['figure.figsize'] = (14, 8)

plt.rcParams['font.size'] = 10

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

@dataclass

class StepData:

"""Класс для хранения данных одного шага"""

thread: str

step\_num: int

x: float

y: float

z: float

kine\_e: float

de\_step: float

step\_leng: float

trak\_leng: float

volume: str

process: str

track\_id: int

parent\_id: int

particle: str

# Единицы измерения

coord\_unit: str = "mm"

energy\_unit: str = "MeV"

length\_unit: str = "mm"

@property

def is\_primary(self) -> bool:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

"""Проверка, является ли частица первичной (Parent ID = 0)"""

return self.parent\_id == 0

class Geant4LogParser:

"""Парсер логов Geant4"""

# Регулярные выражения для парсинга

THREAD\_PATTERN = r'(G4WT\d+)\s\*>'

TRACK\_INFO\_PATTERN = r'Track ID\s\*=\s\*(\d+).\*?Parent ID\s\*=\s\*(\d+)'

PARTICLE\_PATTERN = r'Particle\s\*=\s\*(\w+)'

# Паттерны для итоговой сводки

ENERGY\_DEPOSIT\_PATTERN = r'Energy deposit[:\s]+([-+]?\d+\.?\d\*(?:[eE][-+]?\d+)?)\s\*(\w+)'

ENERGY\_LEAKAGE\_PATTERN = r'Energy leakage[:\s]+([-+]?\d+\.?\d\*(?:[eE][-+]?\d+)?)\s\*(\w+)'

PROCESS\_FREQ\_PATTERN = r'(\w+)\s\*=\s\*(\d+)'

# Конверсия единиц в MeV и mm

ENERGY\_UNITS = {'eV': 1e-6, 'keV': 1e-3, 'MeV': 1.0, 'GeV': 1e3, 'TeV': 1e6, 'meV': 1e-9}

LENGTH\_UNITS = {'fm': 1e-12, 'nm': 1e-6, 'um': 1e-3, 'mm': 1.0, 'cm': 10.0, 'm': 1e3, 'km': 1e6}

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

def \_\_init\_\_(self, log\_file: str):

self.log\_file = Path(log\_file)

self.steps: List[StepData] = []

self.summary: Dict = {}

self.current\_thread = ""

self.current\_track\_id = 0

self.current\_parent\_id = 0

self.current\_particle = ""

# Для отслеживания энергии треков

self.track\_initial\_energy: Dict[int, float] = {}

self.track\_final\_energy: Dict[int, float] = {}

self.track\_particles: Dict[int, str] = {}

self.track\_parent\_ids: Dict[int, int] = {}

def convert\_energy(self, value: float, unit: str) -> float:

"""Конвертация энергии в MeV"""

return value \* self.ENERGY\_UNITS.get(unit, 1.0)

def convert\_length(self, value: float, unit: str) -> float:

"""Конвертация длины в mm"""

return value \* self.LENGTH\_UNITS.get(unit, 1.0)

def parse\_log(self, debug: bool = False) -> None:

"""Основной метод парсинга лога"""

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(f"Парсинг файла: {self.log\_file}")

with open(self.log\_file, 'r', encoding='utf-8', errors='ignore') as f:

lines = f.readlines()

in\_summary = False

in\_step\_table = False

process\_frequencies = {}

step\_header\_found = False

debug\_lines\_sample = []

for i, line in enumerate(lines):

# Сохраняем примеры строк для отладки

if debug and len(debug\_lines\_sample) < 100 and 'Step#' not in line and len(line.strip()) > 20:

debug\_lines\_sample.append((i, line.strip()))

# Поиск префикса потока

thread\_match = re.search(self.THREAD\_PATTERN, line)

if thread\_match:

self.current\_thread = thread\_match.group(1)

# Поиск информации о треке

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

track\_match = re.search(self.TRACK\_INFO\_PATTERN, line)

if track\_match:

self.current\_track\_id = int(track\_match.group(1))

self.current\_parent\_id = int(track\_match.group(2))

self.track\_parent\_ids[self.current\_track\_id] = self.current\_parent\_id

# Поиск типа частицы

particle\_match = re.search(self.PARTICLE\_PATTERN, line)

if particle\_match:

self.current\_particle = particle\_match.group(1)

if self.current\_track\_id not in self.track\_particles:

self.track\_particles[self.current\_track\_id] = self.current\_particle

# Определение начала таблицы шагов

if 'Step#' in line and any(x in line for x in ['KineE', 'dE', 'StepLen']):

step\_header\_found = True

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

in\_step\_table = True

if debug:

print(f"\n[DEBUG] Найден заголовок таблицы шагов на строке {i}:")

print(f" {line.strip()}")

continue

# Парсинг строк данных шагов

if in\_step\_table and step\_header\_found:

step\_data = self.\_parse\_step\_line\_improved(line, i, debug)

if step\_data:

self.steps.append(step\_data)

# Отслеживание начальной и конечной энергии трека

track\_id = step\_data.track\_id

if track\_id not in self.track\_initial\_energy:

self.track\_initial\_energy[track\_id] = step\_data.kine\_e

self.track\_final\_energy[track\_id] = step\_data.kine\_e

elif line.strip() == '' or line.startswith('---') or line.startswith('==='):

in\_step\_table = False

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

# Поиск итоговой сводки - Energy deposit

if 'Energy deposit' in line or 'Total energy deposit' in line:

energy\_match = re.search(self.ENERGY\_DEPOSIT\_PATTERN, line)

if energy\_match:

energy\_val = float(energy\_match.group(1))

energy\_unit = energy\_match.group(2)

self.summary['energy\_deposit'] = self.convert\_energy(energy\_val, energy\_unit)

if debug:

print(f"\n[DEBUG] Найдена Energy deposit: {energy\_val} {energy\_unit} = {self.summary['energy\_deposit']} MeV")

# Поиск Energy leakage

if 'Energy leakage' in line:

leakage\_match = re.search(self.ENERGY\_LEAKAGE\_PATTERN, line)

if leakage\_match:

leakage\_val = float(leakage\_match.group(1))

leakage\_unit = leakage\_match.group(2)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

self.summary['energy\_leakage'] = self.convert\_energy(leakage\_val, leakage\_unit)

if debug:

print(f"[DEBUG] Найдена Energy leakage: {leakage\_val} {leakage\_unit} = {self.summary['energy\_leakage']} MeV")

# Поиск Process calls frequency

if 'Process calls frequency' in line or 'Process frequency' in line:

in\_summary = True

continue

if in\_summary:

# Парсинг строк с процессами

proc\_matches = re.findall(self.PROCESS\_FREQ\_PATTERN, line)

for proc\_match in proc\_matches:

process\_name = proc\_match[0]

process\_count = int(proc\_match[1])

# Исключаем технические поля (не процессы)

if process\_name not in ['sumtot', 'counter', 'N', 'V']:

process\_frequencies[process\_name] = process\_count

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

self.summary['process\_frequencies'] = process\_frequencies

print(f"\nПарсинг завершен:")

print(f" - Всего шагов найдено: {len(self.steps)}")

# Подсчет первичных и вторичных частиц

primary\_steps = sum(1 for s in self.steps if s.is\_primary)

secondary\_steps = len(self.steps) - primary\_steps

print(f" - Шаги первичных частиц (Parent ID = 0): {primary\_steps}")

print(f" - Шаги вторичных частиц (Parent ID > 0): {secondary\_steps}")

print(f" - Уникальных треков: {len(set(s.track\_id for s in self.steps))}")

print(f" - Уникальных потоков: {len(set(s.thread for s in self.steps))}")

print(f" - Типы частиц: {', '.join(set(s.particle for s in self.steps))}")

# Отладочная информация о dEStep

if debug:

de\_step\_values = [s.de\_step for s in self.steps if s.de\_step > 0]

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

if de\_step\_values:

print(f"\n[DEBUG] Статистика dEStep:")

print(f" - Всего ненулевых значений: {len(de\_step\_values)}")

print(f" - Мин: {min(de\_step\_values):.6f} MeV")

print(f" - Макс: {max(de\_step\_values):.6f} MeV")

print(f" - Сумма: {sum(de\_step\_values):.6f} MeV")

print(f" - Примеры первых 5: {[f'{v:.6f}' for v in de\_step\_values[:5]]}")

def \_parse\_step\_line\_improved(self, line: str, line\_num: int, debug: bool = False) -> Optional[StepData]:

"""

УЛУЧШЕННЫЙ парсинг строки шага с правильной обработкой dEStep

Формат строки:

G4WT26 > 1 2.7 mm 2.694 mm 3.605 mm 733.5 keV 223.2 keV 488.6 um 488.6 um MAPI eIoni

"""

if not line.strip() or 'Step#' in line:

return None

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

# Проверка наличия префикса потока

if not re.search(self.THREAD\_PATTERN, line):

return None

parts = line.split()

if len(parts) < 8:

return None

try:

# Находим номер шага

step\_num\_idx = None

for i, part in enumerate(parts):

if part.isdigit() and i > 0: # пропускаем G4WT

step\_num\_idx = i

break

if step\_num\_idx is None:

return None

step\_num = int(parts[step\_num\_idx])

# === ПАРСИНГ КООРДИНАТ ===

coords = []

coord\_units = []

i = step\_num\_idx + 1

while len(coords) < 3 and i < len(parts) - 1:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

try:

val = float(parts[i])

unit = parts[i + 1] if i + 1 < len(parts) else 'mm'

# Проверяем, что единица - это единица длины

if unit in self.LENGTH\_UNITS or unit in ['mm', 'um', 'fm', 'cm', 'm', 'nm']:

coords.append(self.convert\_length(val, unit))

coord\_units.append(unit)

i += 2

else:

i += 1

except (ValueError, IndexError):

i += 1

if len(coords) < 3:

return None

x, y, z = coords[0], coords[1], coords[2]

# === ПАРСИНГ ЭНЕРГИЙ ===

# Находим все пары число+единица энергии

energy\_values = []

i = step\_num\_idx + 7 # начинаем после координат

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

while i < min(len(parts) - 1, step\_num\_idx + 25):

try:

val = float(parts[i])

unit = parts[i + 1] if i + 1 < len(parts) else ''

# Проверяем, является ли следующий элемент единицей энергии

if unit in self.ENERGY\_UNITS or unit in ['eV', 'keV', 'MeV', 'GeV', 'TeV', 'meV']:

energy\_mev = self.convert\_energy(val, unit)

energy\_values.append((i, energy\_mev, val, unit))

i += 2 # пропускаем значение и единицу

else:

i += 1

except (ValueError, IndexError):

i += 1

# Первое значение энергии - это KineE

# Второе значение энергии - это dEStep

kine\_e = energy\_values[0][1] if len(energy\_values) >= 1 else 0.0

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

de\_step = energy\_values[1][1] if len(energy\_values) >= 2 else 0.0

if debug and de\_step > 0:

print(f"[DEBUG] Найдено dEStep на строке {line\_num}: {energy\_values[1][2]} {energy\_values[1][3]} = {de\_step:.6f} MeV")

# === ПАРСИНГ ДЛИН ===

# Находим все пары число+единица длины (после энергий)

length\_values = []

start\_idx = energy\_values[-1][0] + 2 if energy\_values else step\_num\_idx + 10

for i in range(start\_idx, min(len(parts) - 1, len(parts) - 3)):

try:

val = float(parts[i])

unit = parts[i + 1] if i + 1 < len(parts) else ''

if unit in self.LENGTH\_UNITS:

length\_mm = self.convert\_length(val, unit)

length\_values.append(length\_mm)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

if len(length\_values) >= 2:

break

except (ValueError, IndexError):

continue

step\_leng = length\_values[0] if len(length\_values) >= 1 else 0.0

trak\_leng = length\_values[1] if len(length\_values) >= 2 else 0.0

# === ПАРСИНГ VOLUME И PROCESS ===

volume = "Unknown"

process = "Unknown"

# Process обычно последний элемент

if len(parts) > 0:

process = parts[-1]

# Volume обычно предпоследний

if len(parts) > 1:

volume = parts[-2]

# Создаем объект StepData

return StepData(

thread=self.current\_thread,

step\_num=step\_num,

x=x, y=y, z=z,

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

kine\_e=kine\_e,

de\_step=de\_step,

step\_leng=step\_leng,

trak\_leng=trak\_leng,

volume=volume,

process=process,

track\_id=self.current\_track\_id,

parent\_id=self.current\_parent\_id,

particle=self.current\_particle,

coord\_unit="mm",

energy\_unit="MeV",

length\_unit="mm"

)

except Exception as e:

if debug:

print(f"[DEBUG] Ошибка парсинга строки {line\_num}: {e}")

print(f" Строка: {line.strip()}")

return None

def to\_dataframe(self) -> pd.DataFrame:

"""Конвертация данных в pandas DataFrame"""

if not self.steps:

return pd.DataFrame()

data = []

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

for step in self.steps:

data.append({

'thread': step.thread,

'step\_num': step.step\_num,

'x\_mm': step.x,

'y\_mm': step.y,

'z\_mm': step.z,

'kine\_e\_MeV': step.kine\_e,

'de\_step\_MeV': step.de\_step,

'step\_leng\_mm': step.step\_leng,

'trak\_leng\_mm': step.trak\_leng,

'volume': step.volume,

'process': step.process,

'track\_id': step.track\_id,

'parent\_id': step.parent\_id,

'particle': step.particle,

'is\_primary': step.is\_primary

})

return pd.DataFrame(data)

class Geant4Analyzer:

"""Анализатор данных Geant4"""

def \_\_init\_\_(self, df: pd.DataFrame, summary: Dict, output\_dir: str = 'output', log\_filename: str = ''):

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

self.df\_all = df

self.summary = summary

self.log\_filename = log\_filename

# Создаем подпапку с именем лог-файла

if log\_filename:

log\_name = Path(log\_filename).stem # Имя файла без расширения

self.output\_dir = Path(output\_dir) / f"{log\_name}\_log"

else:

self.output\_dir = Path(output\_dir)

self.output\_dir.mkdir(exist\_ok=True, parents=True)

print(f"\nРезультаты будут сохранены в: {self.output\_dir}")

# Разделение на первичные и вторичные частицы

print("\n" + "=" \* 100)

print("РАЗДЕЛЕНИЕ НА ПЕРВИЧНЫЕ И ВТОРИЧНЫЕ ЧАСТИЦЫ")

print("=" \* 100)

self.df\_primary = df[df['is\_primary'] == True].copy()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

self.df\_secondary = df[df['is\_primary'] == False].copy()

print(f"Первичные частицы (Parent ID = 0): {len(self.df\_primary)} шагов")

print(f"Вторичные частицы (Parent ID > 0): {len(self.df\_secondary)} шагов")

print()

if len(self.df\_primary) > 0:

print(f"Типы первичных частиц: {', '.join(self.df\_primary['particle'].unique())}")

if len(self.df\_secondary) > 0:

print(f"Типы вторичных частиц: {', '.join(self.df\_secondary['particle'].unique())}")

def aggregate\_data(self) -> Dict:

"""Агрегация данных по частицам и процессам"""

print("\nАгрегация данных...")

results = {}

# Агрегация первичных частиц

if len(self.df\_primary) > 0:

print(f"\nПЕРВИЧНЫЕ ЧАСТИЦЫ - Агрегация данных...")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

results['primary'] = self.\_aggregate\_by\_particle(self.df\_primary, "ПЕРВИЧНЫЕ")

# Агрегация вторичных частиц

if len(self.df\_secondary) > 0:

print(f"\nВТОРИЧНЫЕ ЧАСТИЦЫ - Агрегация данных...")

results['secondary'] = self.\_aggregate\_by\_particle(self.df\_secondary, "ВТОРИЧНЫЕ")

# Агрегация всех частиц

print(f"\nОБЩИЕ ЧАСТИЦЫ - Агрегация данных...")

results['combined'] = self.\_aggregate\_by\_particle(self.df\_all, "ОБЩИЕ")

return results

def \_aggregate\_by\_particle(self, df: pd.DataFrame, label: str) -> Dict:

"""Внутренний метод агрегации"""

if len(df) == 0:

return {}

# Агрегация по частицам

particle\_stats = df.groupby('particle').agg({

'kine\_e\_MeV': ['count', 'min', 'max', 'mean', 'std'],

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

'de\_step\_MeV': ['sum', 'mean'],

'step\_leng\_mm': ['sum', 'mean'],

'track\_id': 'nunique'

}).round(6)

# Агрегация по процессам

process\_freq = df['process'].value\_counts().to\_dict()

# Энергетический баланс

energy\_balance = {

'total\_de\_step\_all': df['de\_step\_MeV'].sum(),

'total\_de\_step\_positive': df[df['de\_step\_MeV'] > 0]['de\_step\_MeV'].sum(),

'total\_de\_step\_negative': df[df['de\_step\_MeV'] < 0]['de\_step\_MeV'].sum(),

}

print(f" {label}: Обработано {len(df)} шагов, {df['track\_id'].nunique()} треков")

return {

'particle\_stats': particle\_stats,

'process\_freq': process\_freq,

'energy\_balance': energy\_balance

}

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

def export\_data(self, formats: List[str] = ['xlsx']) -> None:

"""Экспорт данных в различные форматы с разбиением >1 млн строк"""

print("\nЭкспорт данных...")

MAX\_ROWS = 1\_000\_000

datasets = [

(self.df\_primary, 'steps\_primary'),

(self.df\_secondary, 'steps\_secondary'),

(self.df\_all, 'steps\_all')

]

for df, name in datasets:

if len(df) == 0:

continue

for fmt in formats:

sub\_dir = self.output\_dir / "steps\_parts" / fmt

sub\_dir.mkdir(exist\_ok=True, parents=True)

if len(df) > MAX\_ROWS:

num\_parts = (len(df) // MAX\_ROWS) + int(len(df) % MAX\_ROWS > 0)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(f" ⚠ {name}: {len(df):,} строк → разбивка на {num\_parts} файлов по {MAX\_ROWS:,} строк")

for i in range(num\_parts):

start = i \* MAX\_ROWS

end = start + MAX\_ROWS

df\_part = df.iloc[start:end]

filepath = sub\_dir / f"{name}\_part{i + 1}.{fmt}"

if fmt == 'csv':

df\_part.to\_csv(filepath, index=False, encoding='utf-8')

elif fmt == 'xlsx':

df\_part.to\_excel(filepath, index=False, engine='openpyxl')

print(f" • Сохранено: {filepath.name} ({len(df\_part):,} строк)")

else:

filepath = sub\_dir / f"{name}.{fmt}"

if fmt == 'csv':

df.to\_csv(filepath, index=False, encoding='utf-8')

elif fmt == 'xlsx':

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

df.to\_excel(filepath, index=False, engine='openpyxl')

print(f" ✓ Сохранено: {filepath.name} ({len(df):,} строк)")

print(f"Данные сохранены в подпапках: {self.output\_dir / 'steps\_parts'}")

def create\_visualizations(self, save\_formats: List[str] = ['svg']) -> None:

"""Создание визуализаций"""

print("\nСоздание визуализаций...")

self.\_plot\_energy\_distributions(save\_formats)

self.\_plot\_de\_distributions(save\_formats)

self.\_plot\_process\_frequencies(save\_formats)

self.\_plot\_spatial\_distributions(save\_formats)

self.\_plot\_energy\_balance(save\_formats)

self.\_plot\_comparisons(save\_formats)

# Новые функции

self.\_plot\_coordinate\_heatmaps(save\_formats)

self.\_plot\_particle\_energy\_distributions(save\_formats)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

def \_plot\_energy\_distributions(self, save\_formats: List[str]) -> None:

"""График распределения кинетической энергии"""

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Первичные частицы

if len(self.df\_primary) > 0:

for particle in self.df\_primary['particle'].unique():

data = self.df\_primary[self.df\_primary['particle'] == particle]['kine\_e\_MeV']

axes[0].hist(data, bins=50, alpha=0.6, label=f'{particle} (n={len(data)})')

axes[0].axvline(data.mean(), color='r', linestyle='--', linewidth=1,

label=f'Среднее: {data.mean():.3f} MeV')

axes[0].set\_xlabel('Кинетическая энергия (MeV)', fontweight='bold')

axes[0].set\_ylabel('Количество шагов', fontweight='bold')

axes[0].set\_title('Распределение энергии - ПЕРВИЧНЫЕ ЧАСТИЦЫ', fontweight='bold', fontsize=12)

axes[0].legend()

axes[0].grid(False)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

# Вторичные частицы

if len(self.df\_secondary) > 0:

for particle in self.df\_secondary['particle'].unique():

data = self.df\_secondary[self.df\_secondary['particle'] == particle]['kine\_e\_MeV']

axes[1].hist(data, bins=50, alpha=0.6, label=f'{particle} (n={len(data)})')

axes[1].axvline(data.mean(), color='r', linestyle='--', linewidth=1,

label=f'Среднее: {data.mean():.3f} MeV')

axes[1].set\_xlabel('Кинетическая энергия (MeV)', fontweight='bold')

axes[1].set\_ylabel('Количество шагов', fontweight='bold')

axes[1].set\_title('Распределение энергии - ВТОРИЧНЫЕ ЧАСТИЦЫ', fontweight='bold', fontsize=12)

axes[1].legend()

axes[1].grid(False)

plt.tight\_layout()

for fmt in save\_formats:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

plt.savefig(self.output\_dir / f'energy\_distribution.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

print(" ✓ Графики распределения энергии созданы")

def \_plot\_de\_distributions(self, save\_formats: List[str]) -> None:

"""Boxplot распределения dE"""

fig, ax = plt.subplots(figsize=(14, 8))

# Фильтруем только ненулевые dE

df\_with\_de = self.df\_all[self.df\_all['de\_step\_MeV'] > 0].copy()

if len(df\_with\_de) > 0:

df\_with\_de['category'] = df\_with\_de['is\_primary'].map({True: 'Первичные', False: 'Вторичные'})

sns.violinplot(data=df\_with\_de, x='particle', y='de\_step\_MeV', hue='category', ax=ax, split=True)

ax.set\_xlabel('Тип частицы', fontweight='bold')

ax.set\_ylabel('Потеря энергии на шаг (MeV)', fontweight='bold')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

ax.set\_title('Распределение потерь энергии (dE > 0)', fontweight='bold', fontsize=14)

ax.grid(False)

ax.legend(title='Категория')

plt.tight\_layout()

for fmt in save\_formats:

plt.savefig(self.output\_dir / f'de\_distribution.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

print(" ✓ Boxplot распределения dE создан")

else:

print(" ⚠ Нет данных с ненулевым dE для графика")

def \_plot\_process\_frequencies(self, save\_formats: List[str]) -> None:

"""Диаграмма частоты процессов"""

fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Первичные частицы

if len(self.df\_primary) > 0:

process\_counts\_primary = self.df\_primary['process'].value\_counts().head(15)

axes[0].barh(range(len(process\_counts\_primary)), process\_counts\_primary.values, color='steelblue')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

axes[0].set\_yticks(range(len(process\_counts\_primary)))

axes[0].set\_yticklabels(process\_counts\_primary.index)

axes[0].set\_xlabel('Количество вызовов', fontweight='bold')

axes[0].set\_title('Частота процессов - ПЕРВИЧНЫЕ ЧАСТИЦЫ', fontweight='bold', fontsize=12)

axes[0].grid(False)

axes[0].invert\_yaxis()

# Вторичные частицы

if len(self.df\_secondary) > 0:

process\_counts\_secondary = self.df\_secondary['process'].value\_counts().head(15)

axes[1].barh(range(len(process\_counts\_secondary)), process\_counts\_secondary.values, color='coral')

axes[1].set\_yticks(range(len(process\_counts\_secondary)))

axes[1].set\_yticklabels(process\_counts\_secondary.index)

axes[1].set\_xlabel('Количество вызовов', fontweight='bold')

axes[1].set\_title('Частота процессов - ВТОРИЧНЫЕ ЧАСТИЦЫ', fontweight='bold', fontsize=12)

axes[1].grid(False)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

axes[1].invert\_yaxis()

plt.tight\_layout()

for fmt in save\_formats:

plt.savefig(self.output\_dir / f'process\_frequency.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

print(" ✓ Диаграмма частоты процессов создана")

def \_plot\_spatial\_distributions(self, save\_formats: List[str]) -> None:

"""Пространственные распределения"""

fig, axes = plt.subplots(2, 3, figsize=(18, 12))

for idx, (df, title) in enumerate([(self.df\_primary, 'ПЕРВИЧНЫЕ'), (self.df\_secondary, 'ВТОРИЧНЫЕ')]):

if len(df) == 0:

continue

# XY projection

axes[idx, 0].hexbin(df['x\_mm'], df['y\_mm'], gridsize=30, cmap='YlOrRd', mincnt=1)

axes[idx, 0].set\_xlabel('X (mm)', fontweight='bold')

axes[idx, 0].set\_ylabel('Y (mm)', fontweight='bold')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

axes[idx, 0].set\_title(f'{title} - Проекция XY', fontweight='bold')

axes[idx, 0].grid(False)

# XZ projection

axes[idx, 1].hexbin(df['x\_mm'], df['z\_mm'], gridsize=30, cmap='YlGnBu', mincnt=1)

axes[idx, 1].set\_xlabel('X (mm)', fontweight='bold')

axes[idx, 1].set\_ylabel('Z (mm)', fontweight='bold')

axes[idx, 1].set\_title(f'{title} - Проекция XZ', fontweight='bold')

axes[idx, 1].grid(False)

# YZ projection

axes[idx, 2].hexbin(df['y\_mm'], df['z\_mm'], gridsize=30, cmap='Greens', mincnt=1)

axes[idx, 2].set\_xlabel('Y (mm)', fontweight='bold')

axes[idx, 2].set\_ylabel('Z (mm)', fontweight='bold')

axes[idx, 2].set\_title(f'{title} - Проекция YZ', fontweight='bold')

axes[idx, 2].grid(False)

plt.tight\_layout()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

for fmt in save\_formats:

plt.savefig(self.output\_dir / f'spatial\_distributions.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

print(" ✓ Пространственные распределения созданы")

def \_plot\_energy\_balance(self, save\_formats: List[str]) -> None:

"""График энергетического баланса"""

fig, ax = plt.subplots(figsize=(12, 6))

# Подготовка данных

categories = []

de\_values = []

colors\_list = []

if len(self.df\_primary) > 0:

prim\_de = self.df\_primary[self.df\_primary['de\_step\_MeV'] > 0]['de\_step\_MeV'].sum()

categories.append('Первичные\n(dE > 0)')

de\_values.append(prim\_de)

colors\_list.append('steelblue')

if len(self.df\_secondary) > 0:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

sec\_de = self.df\_secondary[self.df\_secondary['de\_step\_MeV'] > 0]['de\_step\_MeV'].sum()

categories.append('Вторичные\n(dE > 0)')

de\_values.append(sec\_de)

colors\_list.append('coral')

if 'energy\_deposit' in self.summary:

categories.append('Energy Deposit\n(сводка)')

de\_values.append(self.summary['energy\_deposit'])

colors\_list.append('green')

ax.bar(categories, de\_values, color=colors\_list, alpha=0.7, edgecolor='black')

ax.set\_ylabel('Энергия (MeV)', fontweight='bold', fontsize=12)

ax.set\_title('Энергетический баланс: Сравнение', fontweight='bold', fontsize=14)

ax.grid(False, axis='x')

ax.grid(True, axis='y', alpha=0.3)

# Добавляем значения на столбцы

for i, (cat, val) in enumerate(zip(categories, de\_values)):

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

ax.text(i, val + max(de\_values) \* 0.02, f'{val:.6f} MeV',

ha='center', va='bottom', fontweight='bold')

plt.tight\_layout()

for fmt in save\_formats:

plt.savefig(self.output\_dir / f'energy\_balance.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

print(" ✓ График энергетического баланса создан")

def \_plot\_comparisons(self, save\_formats: List[str]) -> None:

"""Сравнительные графики первичных и вторичных частиц"""

fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# 1. Сравнение числа шагов по частицам

if len(self.df\_primary) > 0 or len(self.df\_secondary) > 0:

prim\_counts = self.df\_primary['particle'].value\_counts() if len(self.df\_primary) > 0 else pd.Series()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

sec\_counts = self.df\_secondary['particle'].value\_counts() if len(self.df\_secondary) > 0 else pd.Series()

all\_particles = set(prim\_counts.index).union(set(sec\_counts.index))

x = np.arange(len(all\_particles))

width = 0.35

prim\_vals = [prim\_counts.get(p, 0) for p in all\_particles]

sec\_vals = [sec\_counts.get(p, 0) for p in all\_particles]

axes[0, 0].bar(x - width/2, prim\_vals, width, label='Первичные', color='steelblue')

axes[0, 0].bar(x + width/2, sec\_vals, width, label='Вторичные', color='coral')

axes[0, 0].set\_xlabel('Тип частицы', fontweight='bold')

axes[0, 0].set\_ylabel('Количество шагов', fontweight='bold')

axes[0, 0].set\_title('Сравнение числа шагов', fontweight='bold')

axes[0, 0].set\_xticks(x)

axes[0, 0].set\_xticks(x)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

axes[0, 0].set\_xticklabels(list(all\_particles))

axes[0, 0].legend()

axes[0, 0].grid(False)

# 2. Сравнение средней энергии

if len(self.df\_primary) > 0 or len(self.df\_secondary) > 0:

prim\_energy = self.df\_primary.groupby('particle')['kine\_e\_MeV'].mean() if len(self.df\_primary) > 0 else pd.Series()

sec\_energy = self.df\_secondary.groupby('particle')['kine\_e\_MeV'].mean() if len(self.df\_secondary) > 0 else pd.Series()

all\_particles\_e = set(prim\_energy.index).union(set(sec\_energy.index))

x = np.arange(len(all\_particles\_e))

prim\_e\_vals = [prim\_energy.get(p, 0) for p in all\_particles\_e]

sec\_e\_vals = [sec\_energy.get(p, 0) for p in all\_particles\_e]

axes[0, 1].bar(x - width/2, prim\_e\_vals, width, label='Первичные', color='steelblue')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

axes[0, 1].bar(x + width/2, sec\_e\_vals, width, label='Вторичные', color='coral')

axes[0, 1].set\_xlabel('Тип частицы', fontweight='bold')

axes[0, 1].set\_ylabel('Средняя энергия (MeV)', fontweight='bold')

axes[0, 1].set\_title('Сравнение средней энергии', fontweight='bold')

axes[0, 1].set\_xticks(x)

axes[0, 1].set\_xticklabels(list(all\_particles\_e))

axes[0, 1].legend()

axes[0, 1].grid(False)

# 3. Распределение треков по потокам

thread\_counts\_prim = self.df\_primary['thread'].value\_counts() if len(self.df\_primary) > 0 else pd.Series()

thread\_counts\_sec = self.df\_secondary['thread'].value\_counts() if len(self.df\_secondary) > 0 else pd.Series()

axes[1, 0].scatter(range(len(thread\_counts\_prim)), thread\_counts\_prim.values,

alpha=0.6, s=100, label='Первичные', color='steelblue')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

axes[1, 0].scatter(range(len(thread\_counts\_sec)), thread\_counts\_sec.values,

alpha=0.6, s=100, label='Вторичные', color='coral')

axes[1, 0].set\_xlabel('Индекс потока', fontweight='bold')

axes[1, 0].set\_ylabel('Количество шагов', fontweight='bold')

axes[1, 0].set\_title('Распределение шагов по потокам', fontweight='bold')

axes[1, 0].legend()

axes[1, 0].grid(True, alpha=0.3)

# 4. Топ процессов

if len(self.df\_primary) > 0 or len(self.df\_secondary) > 0:

prim\_proc = self.df\_primary['process'].value\_counts().head(10) if len(self.df\_primary) > 0 else pd.Series()

sec\_proc = self.df\_secondary['process'].value\_counts().head(10) if len(self.df\_secondary) > 0 else pd.Series()

all\_proc = set(prim\_proc.index).union(set(sec\_proc.index))

x = np.arange(len(all\_proc))

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

prim\_proc\_vals = [prim\_proc.get(p, 0) for p in all\_proc]

sec\_proc\_vals = [sec\_proc.get(p, 0) for p in all\_proc]

axes[1, 1].bar(x - width/2, prim\_proc\_vals, width, label='Первичные', color='steelblue')

axes[1, 1].bar(x + width/2, sec\_proc\_vals, width, label='Вторичные', color='coral')

axes[1, 1].set\_xlabel('Процесс', fontweight='bold')

axes[1, 1].set\_ylabel('Количество вызовов', fontweight='bold')

axes[1, 1].set\_title('Топ-10 процессов', fontweight='bold')

axes[1, 1].set\_xticks(x)

axes[1, 1].set\_xticklabels(list(all\_proc), rotation=45, ha='right')

axes[1, 1].legend()

axes[1, 1].grid(False)

plt.tight\_layout()

for fmt in save\_formats:

plt.savefig(self.output\_dir / f'comparisons.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(" ✓ Сравнительные графики созданы")

def \_plot\_coordinate\_heatmaps(self, save\_formats: List[str]) -> None:

"""Heatmap плотности распределения координат (все частицы разом)"""

df = self.df\_all

if len(df) < 10:

print(" ⚠ Недостаточно данных для построения heatmap координат")

return

fig, axes = plt.subplots(2, 2, figsize=(16, 14))

# X-Y плоскость

if len(df['x\_mm'].unique()) > 1 and len(df['y\_mm'].unique()) > 1:

h, xedges, yedges = np.histogram2d(df['x\_mm'], df['y\_mm'], bins=50)

im1 = axes[0, 0].imshow(h.T, origin='lower', cmap='hot', aspect='auto',

extent=[xedges[0], xedges[-1], yedges[0], yedges[-1]])

axes[0, 0].set\_xlabel('X (mm)', fontsize=12)

axes[0, 0].set\_ylabel('Y (mm)', fontsize=12)

axes[0, 0].set\_title('Плотность распределения: X-Y', fontsize=13, fontweight='bold')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

plt.colorbar(im1, ax=axes[0, 0], label='Количество точек')

# X-Z плоскость

if len(df['x\_mm'].unique()) > 1 and len(df['z\_mm'].unique()) > 1:

h, xedges, zedges = np.histogram2d(df['x\_mm'], df['z\_mm'], bins=50)

im2 = axes[0, 1].imshow(h.T, origin='lower', cmap='hot', aspect='auto',

extent=[xedges[0], xedges[-1], zedges[0], zedges[-1]])

axes[0, 1].set\_xlabel('X (mm)', fontsize=12)

axes[0, 1].set\_ylabel('Z (mm)', fontsize=12)

axes[0, 1].set\_title('Плотность распределения: X-Z', fontsize=13, fontweight='bold')

plt.colorbar(im2, ax=axes[0, 1], label='Количество точек')

# Y-Z плоскость

if len(df['y\_mm'].unique()) > 1 and len(df['z\_mm'].unique()) > 1:

h, yedges, zedges = np.histogram2d(df['y\_mm'], df['z\_mm'], bins=50)

im3 = axes[1, 0].imshow(h.T, origin='lower', cmap='hot', aspect='auto',

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

extent=[yedges[0], yedges[-1], zedges[0], zedges[-1]])

axes[1, 0].set\_xlabel('Y (mm)', fontsize=12)

axes[1, 0].set\_ylabel('Z (mm)', fontsize=12)

axes[1, 0].set\_title('Плотность распределения: Y-Z', fontsize=13, fontweight='bold')

plt.colorbar(im3, ax=axes[1, 0], label='Количество точек')

# 3D проекция

scatter = axes[1, 1].scatter(df['x\_mm'], df['y\_mm'], c=df['z\_mm'],

cmap='viridis', alpha=0.5, s=10)

axes[1, 1].set\_xlabel('X (mm)', fontsize=12)

axes[1, 1].set\_ylabel('Y (mm)', fontsize=12)

axes[1, 1].set\_title('Проекция координат (цвет = Z)', fontsize=13, fontweight='bold')

plt.colorbar(scatter, ax=axes[1, 1], label='Z (mm)')

plt.tight\_layout()

for fmt in save\_formats:

plt.savefig(self.output\_dir / f'coordinate\_heatmaps.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(" ✓ Heatmap координат создан для всех частиц")

def \_plot\_particle\_energy\_distributions(self, save\_formats: List[str]) -> None:

"""Распределение кинетической энергии по каждой частице"""

out\_dir = self.output\_dir / "particle\_distributions"

out\_dir.mkdir(exist\_ok=True, parents=True)

particles = self.df\_all['particle'].unique()

for particle in particles:

df\_p = self.df\_all[self.df\_all['particle'] == particle]

if len(df\_p) < 5:

continue

plt.figure(figsize=(10, 6))

plt.hist(df\_p['kine\_e\_MeV'], bins=50, alpha=0.7, color='steelblue')

plt.axvline(df\_p['kine\_e\_MeV'].mean(), color='r', linestyle='--', linewidth=1,

label=f'Среднее: {df\_p["kine\_e\_MeV"].mean():.3f} MeV')

plt.xlabel('Кинетическая энергия (MeV)', fontweight='bold')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

plt.ylabel('Количество шагов', fontweight='bold')

plt.title(f'Распределение энергии: {particle}', fontweight='bold', fontsize=12)

plt.legend()

plt.grid(False)

plt.tight\_layout()

for fmt in save\_formats:

plt.savefig(out\_dir / f'energy\_distribution\_{particle}.{fmt}', dpi=300, bbox\_inches='tight')

plt.close()

print(f" ✓ Графики распределения энергии созданы для {len(particles)} частиц")

def verify\_results(self, parser: Geant4LogParser) -> Dict:

"""Сверка результатов с итоговой сводкой"""

verification = {}

# Верификация для первичных частиц

if len(self.df\_primary) > 0:

verification['primary'] = self.\_verify\_energy\_balance(self.df\_primary, parser, 'primary')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

# Верификация для вторичных частиц

if len(self.df\_secondary) > 0:

verification['secondary'] = self.\_verify\_energy\_balance(self.df\_secondary, parser, 'secondary')

# Общая верификация

verification['combined'] = self.\_verify\_energy\_balance(self.df\_all, parser, 'combined')

# Сверка процессов

if self.summary.get('process\_frequencies'):

parsed\_freq = self.df\_all['process'].value\_counts().to\_dict()

summary\_freq = self.summary['process\_frequencies']

process\_comparison = {}

all\_processes = set(parsed\_freq.keys()).union(set(summary\_freq.keys()))

# Исключаем технические поля (не процессы)

excluded\_fields = {'sumtot', 'counter', 'N', 'V'}

all\_processes = all\_processes - excluded\_fields

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

for proc in all\_processes:

parsed\_count = parsed\_freq.get(proc, 0)

summary\_count = summary\_freq.get(proc, 0)

process\_comparison[proc] = {

'parsed': parsed\_count,

'summary': summary\_count,

'difference': parsed\_count - summary\_count,

'abs\_difference': abs(parsed\_count - summary\_count)

}

verification['combined']['process\_comparison'] = process\_comparison

return verification

def \_verify\_energy\_balance(self, df: pd.DataFrame, parser: Geant4LogParser, label: str) -> Dict:

"""Внутренний метод верификации энергетического баланса"""

result = {}

if len(df) == 0:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

return result

# Расчет начальной и конечной энергии из треков

track\_ids = df['track\_id'].unique()

initial\_energy = sum(parser.track\_initial\_energy.get(tid, 0) for tid in track\_ids)

final\_energy = sum(parser.track\_final\_energy.get(tid, 0) for tid in track\_ids)

energy\_lost = initial\_energy - final\_energy

# Сумма dEStep

total\_de\_step\_all = df['de\_step\_MeV'].sum()

total\_de\_step\_positive = df[df['de\_step\_MeV'] > 0]['de\_step\_MeV'].sum()

total\_de\_step\_negative = df[df['de\_step\_MeV'] < 0]['de\_step\_MeV'].sum()

result['energy\_balance'] = {

'initial\_energy': initial\_energy,

'final\_energy': final\_energy,

'energy\_lost': energy\_lost,

'total\_de\_step\_all': total\_de\_step\_all,

'total\_de\_step\_positive': total\_de\_step\_positive,

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

'total\_de\_step\_negative': total\_de\_step\_negative,

}

# Сверка с summary (только для combined)

if label == 'combined' and 'energy\_deposit' in self.summary:

energy\_deposit\_summary = self.summary['energy\_deposit']

energy\_leakage\_summary = self.summary.get('energy\_leakage', 0.0)

result['energy\_deposit\_summary'] = energy\_deposit\_summary

result['energy\_leakage\_summary'] = energy\_leakage\_summary

result['total\_summary'] = energy\_deposit\_summary + energy\_leakage\_summary

# МЕТОД 1: Сравнение с положительными dEStep

abs\_diff\_method1 = abs(total\_de\_step\_positive - energy\_deposit\_summary)

rel\_diff\_method1 = (abs\_diff\_method1 / energy\_deposit\_summary \* 100) if energy\_deposit\_summary != 0 else 100.0

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

result['method1\_absolute\_difference'] = abs\_diff\_method1

result['method1\_relative\_difference'] = rel\_diff\_method1

# МЕТОД 2: Сравнение потерянной энергии с Energy deposit

abs\_diff\_method2 = abs(energy\_lost - energy\_deposit\_summary)

rel\_diff\_method2 = (abs\_diff\_method2 / energy\_deposit\_summary \* 100) if energy\_deposit\_summary != 0 else 100.0

result['method2\_absolute\_difference'] = abs\_diff\_method2

result['method2\_relative\_difference'] = rel\_diff\_method2

# Проверка энергетического баланса первичных частиц

primary\_track\_ids = [tid for tid in track\_ids if parser.track\_parent\_ids.get(tid, -1) == 0]

primary\_initial\_energy = sum(parser.track\_initial\_energy.get(tid, 0) for tid in primary\_track\_ids)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

result['primary\_initial\_energy'] = primary\_initial\_energy

result['balance\_check\_difference'] = abs(primary\_initial\_energy - result['total\_summary'])

result['balance\_check\_relative'] = (result['balance\_check\_difference'] / primary\_initial\_energy \* 100) if primary\_initial\_energy != 0 else 0

return result

def generate\_report(self, parser: Geant4LogParser) -> str:

"""Генерация текстового отчета"""

report = []

report.append("=" \* 100)

report.append("ОТЧЕТ ОБ АНАЛИЗЕ ЛОГОВ GEANT4")

report.append("=" \* 100)

report.append(f"Входной файл: {self.log\_filename}")

report.append(f"Дата анализа: {pd.Timestamp.now().strftime('%Y-%m-%d %H:%M:%S')}")

report.append("")

# Общая статистика

report.append("1. ОБЩАЯ СТАТИСТИКА")

report.append("-" \* 100)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append(f"Всего шагов: {len(self.df\_all)}")

report.append(f"Первичные частицы (Parent ID = 0): {len(self.df\_primary)} шагов")

report.append(f"Вторичные частицы (Parent ID > 0): {len(self.df\_secondary)} шагов")

report.append(f"Уникальных треков: {self.df\_all['track\_id'].nunique()}")

report.append(f"Уникальных потоков: {self.df\_all['thread'].nunique()}")

report.append(f"Типы частиц: {', '.join(self.df\_all['particle'].unique())}")

report.append("")

# Статистика по частицам

report.append("2. СТАТИСТИКА ПО ЧАСТИЦАМ")

report.append("-" \* 100)

for category, df in [('ПЕРВИЧНЫЕ', self.df\_primary), ('ВТОРИЧНЫЕ', self.df\_secondary)]:

if len(df) == 0:

continue

report.append(f"\n{category} ЧАСТИЦЫ:")

report.append("-" \* 50)

for particle in df['particle'].unique():

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

particle\_df = df[df['particle'] == particle]

report.append(f"\n Частица: {particle}")

report.append(f" Количество шагов: {len(particle\_df)}")

report.append(f" Количество треков: {particle\_df['track\_id'].nunique()}")

report.append(f" Средняя кинетическая энергия: {particle\_df['kine\_e\_MeV'].mean():.6f} MeV")

report.append(f" Диапазон энергий: [{particle\_df['kine\_e\_MeV'].min():.6f}, {particle\_df['kine\_e\_MeV'].max():.6f}] MeV")

de\_positive = particle\_df[particle\_df['de\_step\_MeV'] > 0]['de\_step\_MeV']

if len(de\_positive) > 0:

report.append(f" Сумма dEStep (> 0): {de\_positive.sum():.6f} MeV")

report.append(f" Среднее dEStep (> 0): {de\_positive.mean():.6f} MeV")

report.append("")

# Энергетический баланс

report.append("3. ЭНЕРГЕТИЧЕСКИЙ БАЛАНС")

report.append("-" \* 100)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

verification = self.verify\_results(parser)

if 'primary' in verification and len(self.df\_primary) > 0:

prim\_balance = verification['primary']['energy\_balance']

report.append("\nПЕРВИЧНЫЕ ЧАСТИЦЫ:")

report.append(f" Начальная энергия: {prim\_balance['initial\_energy']:.6f} MeV")

report.append(f" Конечная энергия: {prim\_balance['final\_energy']:.6f} MeV")

report.append(f" Потерянная энергия: {prim\_balance['energy\_lost']:.6f} MeV")

report.append(f" Сумма всех dEStep: {prim\_balance['total\_de\_step\_all']:.6f} MeV")

report.append(f" Сумма положительных dEStep: {prim\_balance['total\_de\_step\_positive']:.6f} MeV")

report.append("")

if 'secondary' in verification and len(self.df\_secondary) > 0:

sec\_balance = verification['secondary']['energy\_balance']

report.append("ВТОРИЧНЫЕ ЧАСТИЦЫ:")

report.append(f" Начальная энергия: {sec\_balance['initial\_energy']:.6f} MeV")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append(f" Конечная энергия: {sec\_balance['final\_energy']:.6f} MeV")

report.append(f" Потерянная энергия: {sec\_balance['energy\_lost']:.6f} MeV")

report.append(f" Сумма всех dEStep: {sec\_balance['total\_de\_step\_all']:.6f} MeV")

report.append(f" Сумма положительных dEStep: {sec\_balance['total\_de\_step\_positive']:.6f} MeV")

report.append("")

# Сверка с итоговой сводкой

if 'combined' in verification:

comb\_verif = verification['combined']

if 'energy\_deposit\_summary' in comb\_verif:

report.append("4. СВЕРКА С ИТОГОВОЙ СВОДКОЙ")

report.append("-" \* 100)

report.append(f"Energy deposit (из сводки): {comb\_verif['energy\_deposit\_summary']:.6f} MeV")

report.append(f"Energy leakage (из сводки): {comb\_verif['energy\_leakage\_summary']:.6f} MeV")

report.append(f"Сумма (E\_deposit + E\_leakage): {comb\_verif['total\_summary']:.6f} MeV")

report.append("")

report.append("⚠️ ВНИМАНИЕ: Лог неполный (пропущены шаги)!")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append("Метод 1 (сумма dEStep) НЕ РАБОТАЕТ для неполных логов.")

report.append("Используйте ТОЛЬКО Метод 2 (энергетический баланс).")

report.append("")

report.append("Метод 1 (Сумма положительных dEStep - только потери энергии):")

report.append(f" Рассчитано: {comb\_verif['energy\_balance']['total\_de\_step\_positive']:.6f} MeV")

report.append(f" Абсолютная разница: {comb\_verif['method1\_absolute\_difference']:.6f} MeV")

report.append(f" Относительная разница: {comb\_verif['method1\_relative\_difference']:.4f}%")

report.append(" ❌ НЕ ПРИМЕНИМО для этого лога (шаги пропущены)")

report.append("")

report.append("Метод 2 (Начальная - Конечная энергия):")

report.append(f" Рассчитано: {comb\_verif['energy\_balance']['energy\_lost']:.6f} MeV")

report.append(f" Абсолютная разница: {comb\_verif['method2\_absolute\_difference']:.6f} MeV")

report.append(f" Относительная разница: {comb\_verif['method2\_relative\_difference']:.4f}%")

report.append("")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append("Проверка энергетического баланса первичных частиц:")

report.append(f" Начальная энергия первичных: {comb\_verif['primary\_initial\_energy']:.6f} MeV")

report.append(f" E\_deposit + E\_leakage: {comb\_verif['total\_summary']:.6f} MeV")

report.append(f" Разница: {comb\_verif['balance\_check\_difference']:.6f} MeV ({comb\_verif['balance\_check\_relative']:.4f}%)")

report.append("")

if comb\_verif['balance\_check\_relative'] < 1:

report.append("✅ ОТЛИЧНО! Энергетический баланс первичных частиц сходится (<1%)")

elif comb\_verif['balance\_check\_relative'] < 5:

report.append("✅ ХОРОШО! Энергетический баланс первичных частиц приемлем (<5%)")

else:

report.append("⚠️ Есть расхождение в энергетическом балансе")

report.append("")

# Сравнение процессов

if 'combined' in verification and 'process\_comparison' in verification['combined']:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append("5. СРАВНЕНИЕ ЧАСТОТ ПРОЦЕССОВ")

report.append("-" \* 100)

report.append(f"{'Процесс':<30s} | {'Парсинг':>10s} | {'Сводка':>10s} | {'Разница':>10s}")

report.append("-" \* 100)

sorted\_processes = sorted(verification['combined']['process\_comparison'].items(),

key=lambda x: x[1]['abs\_difference'], reverse=True)

for process, data in sorted\_processes[:20]: # Топ-20 процессов

diff\_str = f"{data['difference']:+d}"

report.append(f"{process:<30s} | {data['parsed']:>10d} | "

f"{data['summary']:>10d} | {diff\_str:>10s}")

report.append("")

# Объяснение расхождений

report.append("6. ИНТЕРПРЕТАЦИЯ РЕЗУЛЬТАТОВ")

report.append("-" \* 100)

report.append("")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append("ДВА МЕТОДА ПРОВЕРКИ ЭНЕРГЕТИЧЕСКОГО БАЛАНСА:")

report.append("")

report.append("МЕТОД 1: Сумма потерь энергии (dEStep)")

report.append(" • Суммирует все положительные значения dEStep из шагов")

report.append(" • Работает ТОЛЬКО если ВСЕ шаги записаны в лог")

report.append(" • ❌ НЕ РАБОТАЕТ для неполных логов (когда пропущены шаги)")

report.append(" • Причина: если часть шагов не залогирована, сумма будет неполной")

report.append("")

report.append("МЕТОД 2: Энергетический баланс треков (Начальная - Конечная)")

report.append(" • Использует начальную и конечную энергию каждого трека")

report.append(" • ✅ РАБОТАЕТ даже для неполных логов")

report.append(" • Более надежный метод, не зависит от логирования промежуточных шагов")

report.append("")

report.append("ПРОВЕРКА ЭНЕРГЕТИЧЕСКОГО БАЛАНСА:")

report.append(" • Начальная энергия первичных частиц должна равняться")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append(" сумме Energy deposit + Energy leakage из итоговой сводки")

report.append(" • Это фундаментальный закон сохранения энергии")

report.append(" • Если разница <1%, энергетический баланс сходится отлично")

report.append("")

report.append("-" \* 100)

report.append("")

report.append("Первичные частицы (Parent ID = 0):")

report.append(" • Это исходные частицы, запущенные в симуляции")

report.append(" • Их начальная энергия - это входная энергия симуляции")

report.append("")

report.append("Вторичные частицы (Parent ID > 0):")

report.append(" • Это частицы, рожденные в процессе взаимодействия первичных")

report.append(" • Они могут создаваться через процессы: ionIoni, eBrem, compt, и др.")

report.append(" • Их энергия изначально взята из первичных частиц")

report.append("")

report.append("Energy deposit:")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

report.append(" • Энергия, поглощенная материалом (ионизация)")

report.append(" • Вызывает повреждения в материале")

report.append("")

report.append("Energy leakage:")

report.append(" • Энергия частиц, покинувших объем симуляции")

report.append(" • Не поглощается материалом")

report.append("")

report.append("Возможные причины малых расхождений (<5%):")

report.append(" 1. Округления при конверсии единиц (eV ↔ keV ↔ MeV)")

report.append(" 2. Численная точность при расчетах в Geant4")

report.append(" 3. NIEL energy (неионизирующие потери энергии)")

report.append(" 4. Энергетические пороги отслеживания частиц")

report.append("")

report.append("=" \* 100)

return "\n".join(report)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

def save\_report(self, parser: Geant4LogParser, filename: str = "analysis\_report.txt") -> None:

"""Сохранение отчета в файл"""

report = self.generate\_report(parser)

with open(self.output\_dir / filename, 'w', encoding='utf-8') as f:

f.write(report)

print(f"Отчет сохранен: {self.output\_dir / filename}")

def main():

"""Основная функция с CLI"""

parser = argparse.ArgumentParser(

description='Парсер и анализатор логов Geant4 с разделением на первичные/вторичные частицы',

formatter\_class=argparse.RawDescriptionHelpFormatter,

epilog="""

Примеры использования:

python geant4\_parser\_fixed.py -i simulation.log

python geant4\_parser\_fixed.py -i simulation.log -o results --export csv xlsx

python geant4\_parser\_fixed.py -i simulation.log --no-viz --debug

"""

)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

parser.add\_argument('-i', '--input', required=True,

help='Входной файл лога Geant4')

parser.add\_argument('-o', '--output', default='output',

help='Папка для выходных файлов (по умолчанию: output)')

parser.add\_argument('--export', nargs='+', default=['xlsx'],

choices=['csv', 'xlsx', 'dat'],

help='Форматы экспорта данных')

parser.add\_argument('--plot', nargs='+', default=['svg'],

choices=['png', 'svg'],

help='Форматы сохранения графиков')

parser.add\_argument('--no-viz', action='store\_true',

help='Не создавать визуализации')

parser.add\_argument('--debug', action='store\_true',

help='Режим отладки с детальным выводом')

args = parser.parse\_args()

# Парсинг лога

print("=" \* 100)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print("GEANT4 LOG PARSER - ИСПРАВЛЕННАЯ ВЕРСИЯ С ПРАВИЛЬНЫМ ПАРСИНГОМ dEStep")

print("=" \* 100)

parser\_obj = Geant4LogParser(args.input)

parser\_obj.parse\_log(debug=args.debug)

if not parser\_obj.steps:

print("ПРЕДУПРЕЖДЕНИЕ: Не найдено данных о шагах в логе!")

print("Возможно, формат лога не соответствует ожидаемому.")

print("\nПопробуйте запустить с флагом --debug для диагностики:")

print(f" python geant4\_parser\_fixed.py -i {args.input} --debug")

return

# Конвертация в DataFrame

df = parser\_obj.to\_dataframe()

print(f"\nСоздан DataFrame с {len(df)} записями")

print(f"Колонки: {', '.join(df.columns)}")

# Создание анализатора с указанием имени входного файла

analyzer = Geant4Analyzer(df, parser\_obj.summary, args.output, args.input)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

# Агрегация данных

print("\nАгрегация данных...")

agg\_results = analyzer.aggregate\_data()

# Экспорт данных

analyzer.export\_data(formats=args.export)

# Визуализация

if not args.no\_viz:

analyzer.create\_visualizations(save\_formats=args.plot)

# Генерация отчета

analyzer.save\_report(parser\_obj)

# Вывод основной статистики

print("\n" + "=" \* 100)

print("ИТОГОВАЯ СТАТИСТИКА")

print("=" \* 100)

verification = analyzer.verify\_results(parser\_obj)

# Статистика первичных частиц

if 'primary' in verification and len(analyzer.df\_primary) > 0:

prim\_balance = verification['primary']['energy\_balance']

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(f"\nПЕРВИЧНЫЕ ЧАСТИЦЫ (Parent ID = 0):")

print(f" Начальная энергия: {prim\_balance['initial\_energy']:.6f} MeV")

print(f" Потерянная энергия: {prim\_balance['energy\_lost']:.6f} MeV")

print(f" Сумма положительных dEStep: {prim\_balance['total\_de\_step\_positive']:.6f} MeV")

# Статистика вторичных частиц

if 'secondary' in verification and len(analyzer.df\_secondary) > 0:

sec\_balance = verification['secondary']['energy\_balance']

print(f"\nВТОРИЧНЫЕ ЧАСТИЦЫ (Parent ID > 0):")

print(f" Начальная энергия: {sec\_balance['initial\_energy']:.6f} MeV")

print(f" Потерянная энергия: {sec\_balance['energy\_lost']:.6f} MeV")

print(f" Сумма положительных dEStep: {sec\_balance['total\_de\_step\_positive']:.6f} MeV")

# Общая статистика

if 'combined' in verification:

comb\_verif = verification['combined']

if 'energy\_deposit\_summary' in comb\_verif:

print(f"\nСВЕРКА С ИТОГОВОЙ СВОДКОЙ:")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(f" Energy deposit (из сводки): {comb\_verif['energy\_deposit\_summary']:.6f} MeV")

print(f" Energy leakage (из сводки): {comb\_verif['energy\_leakage\_summary']:.6f} MeV")

print(f" Сумма (E\_deposit + E\_leakage): {comb\_verif['total\_summary']:.6f} MeV")

print()

print(f" ⚠️ ВНИМАНИЕ: Лог неполный (пропущены шаги)!")

print(f" Метод 1 (сумма dEStep) НЕ РАБОТАЕТ для неполных логов.")

print(f" Используйте ТОЛЬКО Метод 2 (энергетический баланс).")

print()

print(f" Метод 1 (Сумма положительных dEStep - только потери энергии):")

print(f" Рассчитано: {comb\_verif['energy\_balance']['total\_de\_step\_positive']:.6f} MeV")

print(f" Абсолютная разница: {comb\_verif['method1\_absolute\_difference']:.6f} MeV")

print(f" Относительная разница: {comb\_verif['method1\_relative\_difference']:.4f}%")

print(f" ❌ НЕ ПРИМЕНИМО для этого лога (шаги пропущены)")

print()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(f" Метод 2 (Начальная - Конечная энергия):")

print(f" Рассчитано: {comb\_verif['energy\_balance']['energy\_lost']:.6f} MeV")

print(f" Абсолютная разница: {comb\_verif['method2\_absolute\_difference']:.6f} MeV")

print(f" Относительная разница: {comb\_verif['method2\_relative\_difference']:.4f}%")

print()

print(f" Проверка энергетического баланса первичных частиц:")

print(f" Начальная энергия первичных: {comb\_verif['primary\_initial\_energy']:.6f} MeV")

print(f" E\_deposit + E\_leakage: {comb\_verif['total\_summary']:.6f} MeV")

print(f" Разница: {comb\_verif['balance\_check\_difference']:.6f} MeV ({comb\_verif['balance\_check\_relative']:.4f}%)")

print()

if comb\_verif['balance\_check\_relative'] < 1:

print(" ✅ ОТЛИЧНО! Энергетический баланс первичных частиц сходится (<1%)")

elif comb\_verif['balance\_check\_relative'] < 5:

print(" ✅ ХОРОШО! Энергетический баланс первичных частиц приемлем (<5%)")

else:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А.**

print(" ⚠️ Есть расхождение в энергетическом балансе")

print("=" \* 100)

print(f"\nВсе результаты сохранены в папке: {analyzer.output\_dir}")

print("\nСозданные файлы:")

print(" • steps\_primary.csv/xlsx - данные первичных частиц")

print(" • steps\_secondary.csv/xlsx - данные вторичных частиц")

print(" • steps\_all.csv/xlsx - все данные")

print(" • analysis\_report.txt - подробный отчет")

print(" • Графики с разделением на первичные/вторичные частицы")

if \_\_name\_\_ == "\_\_main\_\_":

main()

ПРИЛОЖЕНИЕ Б. ПРОГРАММНЫЙ КОД С ИНТЕРФЕЙСОМ

"""

Geant4 Log Parser - GUI Interface

Графический интерфейс для парсинга, анализа и визуализации логов симуляции Geant4

"""

import re

import warnings

from dataclasses import dataclass

from pathlib import Path

from typing import Dict, List, Optional

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import tkinter as tk

from tkinter import ttk, filedialog, scrolledtext

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from matplotlib.figure import Figure

import subprocess

import os

import datetime

import sys

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

warnings.filterwarnings('ignore')

def setup\_log(log\_filename="geant4\_log"):

log\_dir = Path("output") / f"{log\_filename}\_log"

log\_dir.mkdir(parents=True, exist\_ok=True)

timestamp = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")

log\_path = log\_dir / f"{timestamp}.txt"

sys.stdout = open(log\_path, "w", encoding="utf-8")

sys.stderr = sys.stdout

return log\_path

# Настройка стиля графиков

sns.set\_style("whitegrid")

plt.rcParams['figure.figsize'] = (12, 6)

plt.rcParams['font.size'] = 9

@dataclass

class StepData:

"""Класс для хранения данных одного шага"""

thread: str

step\_num: int

x: float

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

y: float

z: float

kine\_e: float

de\_step: float

step\_leng: float

trak\_leng: float

volume: str

process: str

track\_id: int

parent\_id: int

particle: str

# Единицы измерения

coord\_unit: str = "mm"

energy\_unit: str = "MeV"

length\_unit: str = "mm"

@property

def is\_primary(self) -> bool:

"""Проверка, является ли частица первичной (Parent ID = 0)"""

return self.parent\_id == 0

class Geant4LogParser:

"""Парсер логов Geant4"""

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Регулярные выражения для парсинга

THREAD\_PATTERN = r'(G4WT\d+)\s\*>'

TRACK\_INFO\_PATTERN = r'Track ID\s\*=\s\*(\d+).\*?Parent ID\s\*=\s\*(\d+)'

PARTICLE\_PATTERN = r'Particle\s\*=\s\*(\w+)'

# Паттерны для итоговой сводки

ENERGY\_DEPOSIT\_PATTERN = r'Energy deposit[:\s]+([-+]?\d+\.?\d\*(?:[eE][-+]?\d+)?)\s\*(\w+)'

ENERGY\_LEAKAGE\_PATTERN = r'Energy leakage[:\s]+([-+]?\d+\.?\d\*(?:[eE][-+]?\d+)?)\s\*(\w+)'

PROCESS\_FREQ\_PATTERN = r'(\w+)\s\*=\s\*(\d+)'

# Конверсия единиц в MeV и mm

ENERGY\_UNITS = {'eV': 1e-6, 'keV': 1e-3, 'MeV': 1.0, 'GeV': 1e3, 'TeV': 1e6, 'meV': 1e-9}

LENGTH\_UNITS = {'fm': 1e-12, 'nm': 1e-6, 'um': 1e-3, 'mm': 1.0, 'cm': 10.0, 'm': 1e3, 'km': 1e6}

def \_\_init\_\_(self, log\_file: str):

self.log\_file = Path(log\_file)

self.steps: List[StepData] = []

self.summary: Dict = {}

self.current\_thread = ""

self.current\_track\_id = 0

self.current\_parent\_id = 0

self.current\_particle = ""

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Для отслеживания энергии треков

self.track\_initial\_energy: Dict[int, float] = {}

self.track\_final\_energy: Dict[int, float] = {}

self.track\_particles: Dict[int, str] = {}

self.track\_parent\_ids: Dict[int, int] = {}

def convert\_energy(self, value: float, unit: str) -> float:

"""Конвертация энергии в MeV"""

return value \* self.ENERGY\_UNITS.get(unit, 1.0)

def convert\_length(self, value: float, unit: str) -> float:

"""Конвертация длины в mm"""

return value \* self.LENGTH\_UNITS.get(unit, 1.0)

def parse\_log(self, debug: bool = False) -> None:

"""Основной метод парсинга лога"""

print(f"Парсинг файла: {self.log\_file}")

with open(self.log\_file, 'r', encoding='utf-8', errors='ignore') as f:

lines = f.readlines()

in\_summary = False

in\_step\_table = False

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

process\_frequencies = {}

step\_header\_found = False

debug\_lines\_sample = []

for i, line in enumerate(lines):

# Сохраняем примеры строк для отладки

if debug and len(debug\_lines\_sample) < 100 and 'Step#' not in line and len(line.strip()) > 20:

debug\_lines\_sample.append((i, line.strip()))

# Поиск префикса потока

thread\_match = re.search(self.THREAD\_PATTERN, line)

if thread\_match:

self.current\_thread = thread\_match.group(1)

# Поиск информации о треке

track\_match = re.search(self.TRACK\_INFO\_PATTERN, line)

if track\_match:

self.current\_track\_id = int(track\_match.group(1))

self.current\_parent\_id = int(track\_match.group(2))

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

self.track\_parent\_ids[self.current\_track\_id] = self.current\_parent\_id

# Поиск типа частицы

particle\_match = re.search(self.PARTICLE\_PATTERN, line)

if particle\_match:

self.current\_particle = particle\_match.group(1)

if self.current\_track\_id not in self.track\_particles:

self.track\_particles[self.current\_track\_id] = self.current\_particle

# Определение начала таблицы шагов

if 'Step#' in line and any(x in line for x in ['KineE', 'dE', 'StepLen']):

step\_header\_found = True

in\_step\_table = True

if debug:

print(f"\n[DEBUG] Найден заголовок таблицы шагов на строке {i}:")

print(f" {line.strip()}")

continue

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Парсинг строк данных шагов

if in\_step\_table and step\_header\_found:

step\_data = self.\_parse\_step\_line\_improved(line, i, debug)

if step\_data:

self.steps.append(step\_data)

# Отслеживание начальной и конечной энергии трека

track\_id = step\_data.track\_id

if track\_id not in self.track\_initial\_energy:

self.track\_initial\_energy[track\_id] = step\_data.kine\_e

self.track\_final\_energy[track\_id] = step\_data.kine\_e

elif line.strip() == '' or line.startswith('---') or line.startswith('==='):

in\_step\_table = False

# Поиск итоговой сводки - Energy deposit

if 'Energy deposit' in line or 'Total energy deposit' in line:

energy\_match = re.search(self.ENERGY\_DEPOSIT\_PATTERN, line)

if energy\_match:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

energy\_val = float(energy\_match.group(1))

energy\_unit = energy\_match.group(2)

self.summary['energy\_deposit'] = self.convert\_energy(energy\_val, energy\_unit)

if debug:

print(

f"\n[DEBUG] Найдена Energy deposit: {energy\_val} {energy\_unit} = {self.summary['energy\_deposit']} MeV")

# Поиск Energy leakage

if 'Energy leakage' in line:

leakage\_match = re.search(self.ENERGY\_LEAKAGE\_PATTERN, line)

if leakage\_match:

leakage\_val = float(leakage\_match.group(1))

leakage\_unit = leakage\_match.group(2)

self.summary['energy\_leakage'] = self.convert\_energy(leakage\_val, leakage\_unit)

if debug:

print(

f"[DEBUG] Найдена Energy leakage: {leakage\_val} {leakage\_unit} = {self.summary['energy\_leakage']} MeV")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Поиск частот процессов

if in\_summary and '=' in line and any(c.isdigit() for c in line):

proc\_match = re.search(self.PROCESS\_FREQ\_PATTERN, line)

if proc\_match:

process\_name = proc\_match.group(1)

frequency = int(proc\_match.group(2))

process\_frequencies[process\_name] = frequency

# Определение начала сводной секции

if 'Number of process calls' in line or 'processes count' in line:

in\_summary = True

# Сохранение частот процессов

if process\_frequencies:

self.summary['process\_frequencies'] = process\_frequencies

print(f"Парсинг завершен: найдено {len(self.steps)} шагов")

if self.summary:

print(f"Найдена итоговая сводка с {len(self.summary)} параметрами")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Подсчет первичных и вторичных частиц

if self.steps:

primary\_steps = sum(1 for s in self.steps if s.is\_primary)

secondary\_steps = len(self.steps) - primary\_steps

print(f" - Шаги первичных частиц (Parent ID = 0): {primary\_steps}")

print(f" - Шаги вторичных частиц (Parent ID > 0): {secondary\_steps}")

print(f" - Уникальных треков: {len(set(s.track\_id for s in self.steps))}")

print(f" - Уникальных потоков: {len(set(s.thread for s in self.steps))}")

print(f" - Типы частиц: {', '.join(set(s.particle for s in self.steps))}")

def \_parse\_step\_line\_improved(self, line: str, line\_num: int, debug: bool = False) -> Optional[StepData]:

"""

УЛУЧШЕННЫЙ парсинг строки шага с правильной обработкой dEStep

Формат строки:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

G4WT26 > 1 2.7 mm 2.694 mm 3.605 mm 733.5 keV 223.2 keV 488.6 um 488.6 um MAPI eIoni

"""

if not line.strip() or 'Step#' in line:

return None

# Проверка наличия префикса потока

if not re.search(self.THREAD\_PATTERN, line):

return None

parts = line.split()

if len(parts) < 8:

return None

try:

# Находим номер шага

step\_num\_idx = None

for i, part in enumerate(parts):

if part.isdigit() and i > 0: # пропускаем G4WT

step\_num\_idx = i

break

if step\_num\_idx is None:

return None

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

step\_num = int(parts[step\_num\_idx])

# === ПАРСИНГ КООРДИНАТ ===

coords = []

coord\_units = []

i = step\_num\_idx + 1

while len(coords) < 3 and i < len(parts) - 1:

try:

val = float(parts[i])

unit = parts[i + 1] if i + 1 < len(parts) else 'mm'

# Проверяем, что единица - это единица длины

if unit in self.LENGTH\_UNITS or unit in ['mm', 'um', 'fm', 'cm', 'm', 'nm']:

coords.append(self.convert\_length(val, unit))

coord\_units.append(unit)

i += 2

else:

i += 1

except (ValueError, IndexError):

i += 1

if len(coords) < 3:

return None

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

x, y, z = coords[0], coords[1], coords[2]

# === ПАРСИНГ ЭНЕРГИЙ ===

# Находим все пары число+единица энергии

energy\_values = []

i = step\_num\_idx + 7 # начинаем после координат

while i < min(len(parts) - 1, step\_num\_idx + 25):

try:

val = float(parts[i])

unit = parts[i + 1] if i + 1 < len(parts) else ''

# Проверяем, является ли следующий элемент единицей энергии

if unit in self.ENERGY\_UNITS or unit in ['eV', 'keV', 'MeV', 'GeV', 'TeV', 'meV']:

energy\_mev = self.convert\_energy(val, unit)

energy\_values.append((i, energy\_mev, val, unit))

i += 2 # пропускаем значение и единицу

else:

i += 1

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

except (ValueError, IndexError):

i += 1

# Первое значение энергии - это KineE

# Второе значение энергии - это dEStep

kine\_e = energy\_values[0][1] if len(energy\_values) >= 1 else 0.0

de\_step = energy\_values[1][1] if len(energy\_values) >= 2 else 0.0

if debug and de\_step > 0:

print(

f"[DEBUG] Найдено dEStep на строке {line\_num}: {energy\_values[1][2]} {energy\_values[1][3]} = {de\_step:.6f} MeV")

# === ПАРСИНГ ДЛИН ===

# Находим все пары число+единица длины (после энергий)

length\_values = []

start\_idx = energy\_values[-1][0] + 2 if energy\_values else step\_num\_idx + 10

for i in range(start\_idx, min(len(parts) - 1, len(parts) - 3)):

try:

val = float(parts[i])

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

unit = parts[i + 1] if i + 1 < len(parts) else ''

if unit in self.LENGTH\_UNITS:

length\_mm = self.convert\_length(val, unit)

length\_values.append(length\_mm)

if len(length\_values) >= 2:

break

except (ValueError, IndexError):

continue

step\_leng = length\_values[0] if len(length\_values) >= 1 else 0.0

trak\_leng = length\_values[1] if len(length\_values) >= 2 else 0.0

# === ПАРСИНГ VOLUME И PROCESS ===

volume = "Unknown"

process = "Unknown"

# Process обычно последний элемент

if len(parts) > 0:

process = parts[-1]

# Volume обычно предпоследний

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

if len(parts) > 1:

volume = parts[-2]

# Создаем объект StepData

return StepData(

thread=self.current\_thread,

step\_num=step\_num,

x=x, y=y, z=z,

kine\_e=kine\_e,

de\_step=de\_step,

step\_leng=step\_leng,

trak\_leng=trak\_leng,

volume=volume,

process=process,

track\_id=self.current\_track\_id,

parent\_id=self.current\_parent\_id,

particle=self.current\_particle,

coord\_unit="mm",

energy\_unit="MeV",

length\_unit="mm"

)

except Exception as e:

if debug:

print(f"[DEBUG] Ошибка парсинга строки {line\_num}: {e}")

print(f" Строка: {line.strip()}")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

return None

def to\_dataframe(self) -> pd.DataFrame:

"""Конвертация списка шагов в pandas DataFrame"""

if not self.steps:

return pd.DataFrame()

data = []

for step in self.steps:

data.append({

'thread': step.thread,

'step\_num': step.step\_num,

'x': step.x,

'y': step.y,

'z': step.z,

'kine\_e': step.kine\_e,

'de\_step': step.de\_step,

'step\_leng': step.step\_leng,

'trak\_leng': step.trak\_leng,

'volume': step.volume,

'process': step.process,

'track\_id': step.track\_id,

'parent\_id': step.parent\_id,

'particle': step.particle,

'is\_primary': step.is\_primary

})

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

return pd.DataFrame(data)

class Geant4Analyzer:

"""Анализатор данных Geant4"""

def \_\_init\_\_(self, df: pd.DataFrame, summary: Dict, output\_dir: str = 'output', input\_filename: str = ''):

self.df\_all = df

self.df = df # для обратной совместимости

self.summary = summary

self.output\_dir = Path(output\_dir)

self.output\_dir.mkdir(exist\_ok=True)

self.input\_filename = Path(input\_filename).stem if input\_filename else 'analysis'

# Разделение на первичные и вторичные частицы

self.df\_primary = df[df['is\_primary'] == True].copy()

self.df\_secondary = df[df['is\_primary'] == False].copy()

print(f"\nСтатистика данных:")

print(f" Всего шагов: {len(df)}")

print(f" Первичные частицы (Parent ID = 0): {len(self.df\_primary)}")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

print(f" Вторичные частицы (Parent ID > 0): {len(self.df\_secondary)}")

def aggregate\_data(self) -> Dict:

"""Агрегация данных по процессам и частицам"""

results = {}

# Агрегация по процессам

process\_stats = self.df.groupby('process').agg({

'de\_step': ['sum', 'mean', 'count'],

'step\_leng': ['sum', 'mean'],

'kine\_e': 'mean'

}).round(6)

results['process\_stats'] = process\_stats

# Агрегация по частицам

particle\_stats = self.df.groupby('particle').agg({

'de\_step': ['sum', 'mean', 'count'],

'kine\_e': ['mean', 'max'],

'step\_leng': 'sum'

}).round(6)

results['particle\_stats'] = particle\_stats

# Агрегация первичных частиц

if len(self.df\_primary) > 0:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

primary\_stats = self.df\_primary.groupby('particle').agg({

'de\_step': ['sum', 'mean', 'count'],

'kine\_e': ['mean', 'max'],

'track\_id': 'nunique'

}).round(6)

results['primary\_stats'] = primary\_stats

# Агрегация вторичных частиц

if len(self.df\_secondary) > 0:

secondary\_stats = self.df\_secondary.groupby('particle').agg({

'de\_step': ['sum', 'mean', 'count'],

'kine\_e': ['mean', 'max'],

'track\_id': 'nunique'

}).round(6)

results['secondary\_stats'] = secondary\_stats

return results

def export\_data(self, formats: List[str] = ['xlsx']) -> None:

"""Экспорт данных в различные форматы с разбиением >1 млн строк"""

print("\nЭкспорт данных...")

MAX\_ROWS = 1\_000\_000

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

datasets = [

(self.df\_primary, 'steps\_primary'),

(self.df\_secondary, 'steps\_secondary'),

(self.df\_all, 'steps\_all')

]

for df, name in datasets:

if len(df) == 0:

continue

for fmt in formats:

sub\_dir = self.output\_dir / "steps\_parts" / fmt

sub\_dir.mkdir(exist\_ok=True, parents=True)

if len(df) > MAX\_ROWS:

num\_parts = (len(df) // MAX\_ROWS) + int(len(df) % MAX\_ROWS > 0)

print(f" ⚠ {name}: {len(df):,} строк → разбивка на {num\_parts} файлов по {MAX\_ROWS:,} строк")

for i in range(num\_parts):

start = i \* MAX\_ROWS

end = start + MAX\_ROWS

df\_part = df.iloc[start:end]

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

filepath = sub\_dir / f"{name}\_part{i + 1}.{fmt}"

if fmt == 'csv':

df\_part.to\_csv(filepath, index=False, encoding='utf-8')

elif fmt == 'xlsx':

df\_part.to\_excel(filepath, index=False, engine='openpyxl')

print(f" • Сохранено: {filepath.name} ({len(df\_part):,} строк)")

else:

filepath = sub\_dir / f"{name}.{fmt}"

if fmt == 'csv':

df.to\_csv(filepath, index=False, encoding='utf-8')

elif fmt == 'xlsx':

df.to\_excel(filepath, index=False, engine='openpyxl')

print(f" ✓ Сохранено: {filepath.name} ({len(df):,} строк)")

print(f"Данные сохранены в подпапках: {self.output\_dir / 'steps\_parts'}")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

def create\_visualizations(self, save\_formats: List[str] = ['png']) -> Dict:

"""Создание всех визуализаций"""

print("\nСоздание визуализаций...")

figures = {}

# 1. Анализ процессов

fig\_processes = self.\_create\_process\_analysis()

figures['processes'] = fig\_processes

# 2. Анализ первичных частиц

if len(self.df\_primary) > 0:

fig\_primary = self.\_create\_primary\_analysis()

figures['primary'] = fig\_primary

# 3. Анализ вторичных частиц

if len(self.df\_secondary) > 0:

fig\_secondary = self.\_create\_secondary\_analysis()

figures['secondary'] = fig\_secondary

# Сохранение фигур

for name, fig in figures.items():

for fmt in save\_formats:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

filename = self.output\_dir / f'{name}\_analysis.{fmt}'

fig.savefig(filename, dpi=150, bbox\_inches='tight')

print(f" Сохранено: {filename}")

return figures

def \_create\_process\_analysis(self) -> Figure:

"""Создание графиков анализа процессов"""

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

fig.suptitle('Анализ процессов', fontsize=14, fontweight='bold')

# 1. Частота процессов

process\_counts = self.df['process'].value\_counts().head(15)

axes[0, 0].barh(range(len(process\_counts)), process\_counts.values)

axes[0, 0].set\_yticks(range(len(process\_counts)))

axes[0, 0].set\_yticklabels(process\_counts.index)

axes[0, 0].set\_xlabel('Количество шагов')

axes[0, 0].set\_title('Топ-15 процессов по частоте')

axes[0, 0].grid(axis='x', alpha=0.3)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# 2. Энергетические потери по процессам

process\_energy = self.df.groupby('process')['de\_step'].sum().sort\_values(ascending=False).head(15)

axes[0, 1].barh(range(len(process\_energy)), process\_energy.values)

axes[0, 1].set\_yticks(range(len(process\_energy)))

axes[0, 1].set\_yticklabels(process\_energy.index)

axes[0, 1].set\_xlabel('Суммарные потери энергии (MeV)')

axes[0, 1].set\_title('Топ-15 процессов по энергопотерям')

axes[0, 1].grid(axis='x', alpha=0.3)

# 3. Сравнение первичных и вторичных по процессам

if len(self.df\_primary) > 0 and len(self.df\_secondary) > 0:

primary\_proc = self.df\_primary['process'].value\_counts().head(10)

secondary\_proc = self.df\_secondary['process'].value\_counts().head(10)

all\_processes = list(set(primary\_proc.index) | set(secondary\_proc.index))

x = np.arange(len(all\_processes))

width = 0.35

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

primary\_vals = [primary\_proc.get(p, 0) for p in all\_processes]

secondary\_vals = [secondary\_proc.get(p, 0) for p in all\_processes]

axes[1, 0].bar(x - width / 2, primary\_vals, width, label='Первичные', alpha=0.8)

axes[1, 0].bar(x + width / 2, secondary\_vals, width, label='Вторичные', alpha=0.8)

axes[1, 0].set\_xlabel('Процессы')

axes[1, 0].set\_ylabel('Количество шагов')

axes[1, 0].set\_title('Сравнение процессов: первичные vs вторичные')

axes[1, 0].set\_xticks(x)

axes[1, 0].set\_xticklabels(all\_processes, rotation=45, ha='right')

axes[1, 0].legend()

axes[1, 0].grid(axis='y', alpha=0.3)

# 4. Средняя длина шага по процессам

process\_step\_len = self.df.groupby('process')['step\_leng'].mean().sort\_values(ascending=False).head(15)

axes[1, 1].barh(range(len(process\_step\_len)), process\_step\_len.values)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

axes[1, 1].set\_yticks(range(len(process\_step\_len)))

axes[1, 1].set\_yticklabels(process\_step\_len.index)

axes[1, 1].set\_xlabel('Средняя длина шага (mm)')

axes[1, 1].set\_title('Средняя длина шага по процессам')

axes[1, 1].grid(axis='x', alpha=0.3)

plt.tight\_layout()

return fig

def \_create\_primary\_analysis(self) -> Figure:

"""Создание графиков анализа первичных частиц"""

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

fig.suptitle('Анализ первичных частиц (Parent ID = 0)', fontsize=14, fontweight='bold')

# 1. Распределение по типам частиц

particle\_counts = self.df\_primary['particle'].value\_counts()

axes[0, 0].pie(particle\_counts.values, labels=particle\_counts.index, autopct='%1.1f%%')

axes[0, 0].set\_title('Распределение первичных частиц')

# 2. Энергопотери первичных частиц

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

particle\_energy = self.df\_primary.groupby('particle')['de\_step'].sum().sort\_values(ascending=False)

axes[0, 1].bar(range(len(particle\_energy)), particle\_energy.values)

axes[0, 1].set\_xticks(range(len(particle\_energy)))

axes[0, 1].set\_xticklabels(particle\_energy.index, rotation=45, ha='right')

axes[0, 1].set\_ylabel('Суммарные потери энергии (MeV)')

axes[0, 1].set\_title('Энергопотери первичных частиц')

axes[0, 1].grid(axis='y', alpha=0.3)

# 3. Траектории первичных частиц (XY проекция)

for particle in self.df\_primary['particle'].unique():

df\_part = self.df\_primary[self.df\_primary['particle'] == particle]

axes[1, 0].plot(df\_part['x'], df\_part['y'], 'o-', label=particle, alpha=0.6, markersize=3)

axes[1, 0].set\_xlabel('X (mm)')

axes[1, 0].set\_ylabel('Y (mm)')

axes[1, 0].set\_title('Траектории первичных частиц (XY)')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

axes[1, 0].legend()

axes[1, 0].grid(alpha=0.3)

# 4. Энергия vs пройденный путь

for particle in self.df\_primary['particle'].unique():

df\_part = self.df\_primary[self.df\_primary['particle'] == particle]

axes[1, 1].plot(df\_part['trak\_leng'], df\_part['kine\_e'], 'o-', label=particle, alpha=0.6, markersize=3)

axes[1, 1].set\_xlabel('Пройденный путь (mm)')

axes[1, 1].set\_ylabel('Кинетическая энергия (MeV)')

axes[1, 1].set\_title('Зависимость энергии от пути')

axes[1, 1].legend()

axes[1, 1].grid(alpha=0.3)

plt.tight\_layout()

return fig

def \_create\_secondary\_analysis(self) -> Figure:

"""Создание графиков анализа вторичных частиц"""

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

fig.suptitle('Анализ вторичных частиц (Parent ID > 0)', fontsize=14, fontweight='bold')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# 1. Распределение по типам вторичных частиц

particle\_counts = self.df\_secondary['particle'].value\_counts().head(10)

axes[0, 0].barh(range(len(particle\_counts)), particle\_counts.values)

axes[0, 0].set\_yticks(range(len(particle\_counts)))

axes[0, 0].set\_yticklabels(particle\_counts.index)

axes[0, 0].set\_xlabel('Количество шагов')

axes[0, 0].set\_title('Топ-10 вторичных частиц')

axes[0, 0].grid(axis='x', alpha=0.3)

# 2. Энергопотери вторичных частиц

particle\_energy = self.df\_secondary.groupby('particle')['de\_step'].sum().sort\_values(ascending=False).head(10)

axes[0, 1].barh(range(len(particle\_energy)), particle\_energy.values)

axes[0, 1].set\_yticks(range(len(particle\_energy)))

axes[0, 1].set\_yticklabels(particle\_energy.index)

axes[0, 1].set\_xlabel('Суммарные потери энергии (MeV)')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

axes[0, 1].set\_title('Энергопотери вторичных частиц')

axes[0, 1].grid(axis='x', alpha=0.3)

# 3. Распределение начальной энергии вторичных частиц

secondary\_tracks = self.df\_secondary.groupby('track\_id')['kine\_e'].first()

axes[1, 0].hist(secondary\_tracks.values, bins=50, edgecolor='black', alpha=0.7)

axes[1, 0].set\_xlabel('Начальная кинетическая энергия (MeV)')

axes[1, 0].set\_ylabel('Количество частиц')

axes[1, 0].set\_title('Распределение начальной энергии вторичных частиц')

axes[1, 0].set\_yscale('log')

axes[1, 0].grid(alpha=0.3)

# 4. Средняя длина пути по типам вторичных частиц

particle\_path = self.df\_secondary.groupby('particle')['trak\_leng'].mean().sort\_values(ascending=False).head(10)

axes[1, 1].barh(range(len(particle\_path)), particle\_path.values)

axes[1, 1].set\_yticks(range(len(particle\_path)))

axes[1, 1].set\_yticklabels(particle\_path.index)

axes[1, 1].set\_xlabel('Средняя длина пути (mm)')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

axes[1, 1].set\_title('Средний путь вторичных частиц')

axes[1, 1].grid(axis='x', alpha=0.3)

plt.tight\_layout()

return fig

def verify\_results(self, parser: Geant4LogParser) -> Dict:

"""Верификация результатов"""

verification = {}

# Анализ первичных частиц

if len(self.df\_primary) > 0:

primary\_tracks = self.df\_primary['track\_id'].unique()

primary\_initial = sum(parser.track\_initial\_energy.get(tid, 0) for tid in primary\_tracks)

primary\_final = sum(parser.track\_final\_energy.get(tid, 0) for tid in primary\_tracks)

primary\_lost = primary\_initial - primary\_final

primary\_de\_step\_positive = self.df\_primary[self.df\_primary['de\_step'] > 0]['de\_step'].sum()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

verification['primary'] = {

'energy\_balance': {

'initial\_energy': primary\_initial,

'final\_energy': primary\_final,

'energy\_lost': primary\_lost,

'total\_de\_step\_positive': primary\_de\_step\_positive

}

}

# Анализ вторичных частиц

if len(self.df\_secondary) > 0:

secondary\_tracks = self.df\_secondary['track\_id'].unique()

secondary\_initial = sum(parser.track\_initial\_energy.get(tid, 0) for tid in secondary\_tracks)

secondary\_final = sum(parser.track\_final\_energy.get(tid, 0) for tid in secondary\_tracks)

secondary\_lost = secondary\_initial - secondary\_final

secondary\_de\_step\_positive = self.df\_secondary[self.df\_secondary['de\_step'] > 0]['de\_step'].sum()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

verification['secondary'] = {

'energy\_balance': {

'initial\_energy': secondary\_initial,

'final\_energy': secondary\_final,

'energy\_lost': secondary\_lost,

'total\_de\_step\_positive': secondary\_de\_step\_positive

}

}

# Общая верификация

all\_tracks = self.df['track\_id'].unique()

total\_initial = sum(parser.track\_initial\_energy.get(tid, 0) for tid in all\_tracks)

total\_final = sum(parser.track\_final\_energy.get(tid, 0) for tid in all\_tracks)

total\_lost = total\_initial - total\_final

total\_de\_step\_positive = self.df[self.df['de\_step'] > 0]['de\_step'].sum()

verification['combined'] = {

'energy\_balance': {

'initial\_energy': total\_initial,

'final\_energy': total\_final,

'energy\_lost': total\_lost,

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

'total\_de\_step\_positive': total\_de\_step\_positive

}

}

# Сверка с итоговой сводкой

if 'energy\_deposit' in self.summary and 'energy\_leakage' in self.summary:

energy\_deposit\_summary = self.summary['energy\_deposit']

energy\_leakage\_summary = self.summary['energy\_leakage']

total\_summary = energy\_deposit\_summary + energy\_leakage\_summary

verification['combined']['energy\_deposit\_summary'] = energy\_deposit\_summary

verification['combined']['energy\_leakage\_summary'] = energy\_leakage\_summary

verification['combined']['total\_summary'] = total\_summary

# Метод 1: сумма положительных dEStep

method1\_diff = abs(total\_de\_step\_positive - total\_summary)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

method1\_rel\_diff = (method1\_diff / total\_summary \* 100) if total\_summary > 0 else 0

verification['combined']['method1\_absolute\_difference'] = method1\_diff

verification['combined']['method1\_relative\_difference'] = method1\_rel\_diff

# Метод 2: начальная - конечная энергия

method2\_diff = abs(total\_lost - total\_summary)

method2\_rel\_diff = (method2\_diff / total\_summary \* 100) if total\_summary > 0 else 0

verification['combined']['method2\_absolute\_difference'] = method2\_diff

verification['combined']['method2\_relative\_difference'] = method2\_rel\_diff

# Проверка энергетического баланса первичных частиц

if 'primary' in verification:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

primary\_initial\_energy = verification['primary']['energy\_balance']['initial\_energy']

balance\_diff = abs(primary\_initial\_energy - total\_summary)

balance\_rel = (balance\_diff / primary\_initial\_energy \* 100) if primary\_initial\_energy > 0 else 0

verification['combined']['primary\_initial\_energy'] = primary\_initial\_energy

verification['combined']['balance\_check\_difference'] = balance\_diff

verification['combined']['balance\_check\_relative'] = balance\_rel

return verification

def generate\_report(self, parser: Geant4LogParser) -> str:

"""Генерация текстового отчета"""

report = []

report.append("=" \* 100)

report.append("ОТЧЕТ ОБ АНАЛИЗЕ ЛОГОВ GEANT4")

report.append("=" \* 100)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

report.append("")

# Основная информация

report.append(f"Входной файл: {parser.log\_file}")

report.append(f"Всего шагов: {len(self.df)}")

report.append(f"Первичные частицы: {len(self.df\_primary)} шагов")

report.append(f"Вторичные частицы: {len(self.df\_secondary)} шагов")

report.append("")

# Верификация

verification = self.verify\_results(parser)

if 'primary' in verification:

report.append("-" \* 100)

report.append("ПЕРВИЧНЫЕ ЧАСТИЦЫ (Parent ID = 0)")

report.append("-" \* 100)

prim\_balance = verification['primary']['energy\_balance']

report.append(f"Начальная энергия: {prim\_balance['initial\_energy']:.6f} MeV")

report.append(f"Конечная энергия: {prim\_balance['final\_energy']:.6f} MeV")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

report.append(f"Потерянная энергия: {prim\_balance['energy\_lost']:.6f} MeV")

report.append(f"Сумма положительных dEStep: {prim\_balance['total\_de\_step\_positive']:.6f} MeV")

report.append("")

if 'secondary' in verification:

report.append("-" \* 100)

report.append("ВТОРИЧНЫЕ ЧАСТИЦЫ (Parent ID > 0)")

report.append("-" \* 100)

sec\_balance = verification['secondary']['energy\_balance']

report.append(f"Начальная энергия: {sec\_balance['initial\_energy']:.6f} MeV")

report.append(f"Конечная энергия: {sec\_balance['final\_energy']:.6f} MeV")

report.append(f"Потерянная энергия: {sec\_balance['energy\_lost']:.6f} MeV")

report.append(f"Сумма положительных dEStep: {sec\_balance['total\_de\_step\_positive']:.6f} MeV")

report.append("")

if 'combined' in verification and 'energy\_deposit\_summary' in verification['combined']:

report.append("-" \* 100)

report.append("ЭНЕРГЕТИЧЕСКИЙ БАЛАНС")

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

report.append("-" \* 100)

comb = verification['combined']

report.append(f"Energy deposit (из сводки): {comb['energy\_deposit\_summary']:.6f} MeV")

report.append(f"Energy leakage (из сводки): {comb['energy\_leakage\_summary']:.6f} MeV")

report.append(f"Сумма: {comb['total\_summary']:.6f} MeV")

report.append("")

report.append(f"Начальная энергия первичных: {comb['primary\_initial\_energy']:.6f} MeV")

report.append(

f"Разница с E\_deposit + E\_leakage: {comb['balance\_check\_difference']:.6f} MeV ({comb['balance\_check\_relative']:.4f}%)")

report.append("")

report.append("=" \* 100)

return "\n".join(report)

def save\_report(self, parser: Geant4LogParser, filename: str = "analysis\_report.txt") -> None:

"""Сохранение отчета в файл"""

report = self.generate\_report(parser)

with open(self.output\_dir / filename, 'w', encoding='utf-8') as f:

f.write(report)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

print(f"Отчет сохранен: {self.output\_dir / filename}")

class Geant4GUI:

"""Графический интерфейс для анализа логов Geant4"""

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Geant4 Log Analyzer")

self.root.geometry("1400x900")

self.parser = None

self.analyzer = None

self.df = None

self.figures = {}

self.output\_dir = None

self.file\_paths = {}

self.create\_widgets()

def create\_widgets(self):

"""Создание виджетов интерфейса"""

# Верхняя панель - загрузка файла

top\_frame = ttk.Frame(self.root, padding="10")

top\_frame.pack(fill=tk.X, side=tk.TOP)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Первая строка - загрузка файла

file\_frame = ttk.Frame(top\_frame)

file\_frame.pack(fill=tk.X, pady=(0, 5))

ttk.Label(file\_frame, text="Файл лога:", font=('Arial', 10, 'bold')).pack(side=tk.LEFT, padx=5)

self.file\_label = ttk.Label(file\_frame, text="Файл не выбран", foreground="gray")

self.file\_label.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

self.load\_btn = ttk.Button(file\_frame, text="Загрузить лог-файл", command=self.load\_file)

self.load\_btn.pack(side=tk.LEFT, padx=5)

self.analyze\_btn = ttk.Button(file\_frame, text="Анализировать", command=self.analyze\_file, state=tk.DISABLED)

self.analyze\_btn.pack(side=tk.LEFT, padx=5)

# Вторая строка - форматы экспорта

format\_frame = ttk.LabelFrame(top\_frame, text="Форматы экспорта", padding="5")

format\_frame.pack(fill=tk.X, pady=(5, 0))

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Левая часть - графики

plot\_frame = ttk.Frame(format\_frame)

plot\_frame.pack(side=tk.LEFT, padx=10)

ttk.Label(plot\_frame, text="Графики:", font=('Arial', 9, 'bold')).pack(side=tk.LEFT, padx=(0, 10))

self.format\_png = tk.BooleanVar(value=True)

self.format\_svg = tk.BooleanVar(value=False)

ttk.Checkbutton(plot\_frame, text="PNG", variable=self.format\_png).pack(side=tk.LEFT, padx=5)

ttk.Checkbutton(plot\_frame, text="SVG", variable=self.format\_svg).pack(side=tk.LEFT, padx=5)

# Правая часть - данные

data\_frame = ttk.Frame(format\_frame)

data\_frame.pack(side=tk.LEFT, padx=10)

ttk.Label(data\_frame, text="Данные:", font=('Arial', 9, 'bold')).pack(side=tk.LEFT, padx=(0, 10))

self.format\_xlsx = tk.BooleanVar(value=True)

self.format\_csv = tk.BooleanVar(value=False)

ttk.Checkbutton(data\_frame, text="XLSX", variable=self.format\_xlsx).pack(side=tk.LEFT, padx=5)

ttk.Checkbutton(data\_frame, text="CSV", variable=self.format\_csv).pack(side=tk.LEFT, padx=5)

# Создание notebook для вкладок

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

self.notebook = ttk.Notebook(self.root)

self.notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=5)

# Вкладка 1: Обзор и результаты

self.overview\_tab = ttk.Frame(self.notebook)

self.notebook.add(self.overview\_tab, text="📊 Обзор")

self.create\_overview\_tab()

# Вкладка 2: Анализ процессов

self.processes\_tab = ttk.Frame(self.notebook)

self.notebook.add(self.processes\_tab, text="⚙️ Процессы")

self.create\_processes\_tab()

# Вкладка 3: Первичные частицы

self.primary\_tab = ttk.Frame(self.notebook)

self.notebook.add(self.primary\_tab, text="🔵 Первичные частицы")

self.create\_primary\_tab()

# Вкладка 4: Вторичные частицы

self.secondary\_tab = ttk.Frame(self.notebook)

self.notebook.add(self.secondary\_tab, text="🔴 Вторичные частицы")

self.create\_secondary\_tab()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Вкладка 5: Файлы и экспорт

self.files\_tab = ttk.Frame(self.notebook)

self.notebook.add(self.files\_tab, text="📁 Файлы")

self.create\_files\_tab()

# Строка состояния

self.status\_bar = ttk.Label(self.root, text="Готов к работе", relief=tk.SUNKEN, anchor=tk.W)

self.status\_bar.pack(fill=tk.X, side=tk.BOTTOM)

def create\_overview\_tab(self):

"""Создание вкладки обзора"""

# Текстовое поле для отчета

self.overview\_text = scrolledtext.ScrolledText(self.overview\_tab, wrap=tk.WORD, font=('Courier', 10))

self.overview\_text.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

self.overview\_text.insert('1.0', "Загрузите лог-файл и нажмите 'Анализировать' для начала работы...")

self.overview\_text.config(state=tk.DISABLED)

def create\_processes\_tab(self):

"""Создание вкладки анализа процессов"""

# Фрейм для графика

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

self.processes\_canvas\_frame = ttk.Frame(self.processes\_tab)

self.processes\_canvas\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

label = ttk.Label(self.processes\_canvas\_frame, text="Графики появятся после анализа",

font=('Arial', 12), foreground="gray")

label.pack(expand=True)

def create\_primary\_tab(self):

"""Создание вкладки первичных частиц"""

# Фрейм для графика

self.primary\_canvas\_frame = ttk.Frame(self.primary\_tab)

self.primary\_canvas\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

label = ttk.Label(self.primary\_canvas\_frame, text="Графики появятся после анализа",

font=('Arial', 12), foreground="gray")

label.pack(expand=True)

def create\_secondary\_tab(self):

"""Создание вкладки вторичных частиц"""

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Фрейм для графика

self.secondary\_canvas\_frame = ttk.Frame(self.secondary\_tab)

self.secondary\_canvas\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

label = ttk.Label(self.secondary\_canvas\_frame, text="Графики появятся после анализа",

font=('Arial', 12), foreground="gray")

label.pack(expand=True)

def create\_files\_tab(self):

"""Создание вкладки для работы с файлами"""

# Фрейм для открытия файлов

files\_frame = ttk.LabelFrame(self.files\_tab, text="Сгенерированные файлы", padding="10")

files\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Список файлов

list\_frame = ttk.Frame(files\_frame)

list\_frame.pack(fill=tk.BOTH, expand=True)

# Scrollbar

scrollbar = ttk.Scrollbar(list\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Listbox для файлов

self.files\_listbox = tk.Listbox(list\_frame, yscrollcommand=scrollbar.set, font=('Courier', 10))

self.files\_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

scrollbar.config(command=self.files\_listbox.yview)

# Кнопки действий

buttons\_frame = ttk.Frame(files\_frame)

buttons\_frame.pack(fill=tk.X, pady=5)

ttk.Button(buttons\_frame, text="Открыть выбранный файл",

command=self.open\_selected\_file).pack(side=tk.LEFT, padx=5)

ttk.Button(buttons\_frame, text="Открыть папку с результатами",

command=self.open\_output\_folder).pack(side=tk.LEFT, padx=5)

ttk.Button(buttons\_frame, text="Обновить список",

command=self.refresh\_files\_list).pack(side=tk.LEFT, padx=5)

def load\_file(self):

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

"""Загрузка лог-файла"""

filename = filedialog.askopenfilename(

title="Выберите лог-файл Geant4",

filetypes=[("Text files", "\*.txt \*.log"), ("All files", "\*.\*")]

)

if filename:

self.log\_file = filename

self.file\_label.config(text=os.path.basename(filename), foreground="black")

self.analyze\_btn.config(state=tk.NORMAL)

self.status\_bar.config(text=f"Файл загружен: {filename}")

def analyze\_file(self):

"""Анализ файла"""

if not hasattr(self, 'log\_file'):

# messagebox.showerror("Ошибка", "Сначала загрузите лог-файл")

return

# Получение выбранных форматов

plot\_formats = []

if self.format\_png.get():

plot\_formats.append('png')

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

if self.format\_svg.get():

plot\_formats.append('svg')

data\_formats = []

if self.format\_xlsx.get():

data\_formats.append('xlsx')

if self.format\_csv.get():

data\_formats.append('csv')

# Проверка что хотя бы один формат выбран

if not plot\_formats:

# messagebox.showwarning("Предупреждение", "Выберите хотя бы один формат для графиков (PNG или SVG)")

return

if not data\_formats:

# messagebox.showwarning("Предупреждение", "Выберите хотя бы один формат для данных (XLSX или CSV)")

return

try:

self.status\_bar.config(text="Идет анализ...")

self.root.update()

# Создание выходной директории

self.output\_dir = Path('output')

self.output\_dir.mkdir(exist\_ok=True)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# Парсинг лога

self.parser = Geant4LogParser(self.log\_file)

self.parser.parse\_log(debug=False)

if not self.parser.steps:

# messagebox.showerror("Ошибка", "Не найдено данных о шагах в логе!")

self.status\_bar.config(text="Ошибка: данные не найдены")

return

# Конвертация в DataFrame

self.df = self.parser.to\_dataframe()

# Создание анализатора

self.analyzer = Geant4Analyzer(self.df, self.parser.summary, str(self.output\_dir), self.log\_file)

# Агрегация данных

self.analyzer.aggregate\_data()

# Экспорт данных с выбранными форматами

self.analyzer.export\_data(formats=data\_formats)

# Визуализация с выбранными форматами

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

self.figures = self.analyzer.create\_visualizations(save\_formats=plot\_formats)

# Генерация отчета

self.analyzer.save\_report(self.parser)

# Обновление интерфейса

self.update\_overview()

self.update\_processes\_plot()

self.update\_primary\_plot()

self.update\_secondary\_plot()

self.refresh\_files\_list()

self.status\_bar.config(text="Анализ завершен успешно!")

# messagebox.showinfo("Успех", f"Анализ завершен!\n\nРезультаты сохранены в папке: {self.output\_dir}")

except Exception as e:

# messagebox.showerror("Ошибка", f"Произошла ошибка при анализе:\n{str(e)}")

self.status\_bar.config(text="Ошибка анализа")

import traceback

traceback.print\_exc()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

def update\_overview(self):

"""Обновление вкладки обзора"""

if self.analyzer is None:

return

report = self.analyzer.generate\_report(self.parser)

self.overview\_text.config(state=tk.NORMAL)

self.overview\_text.delete('1.0', tk.END)

self.overview\_text.insert('1.0', report)

self.overview\_text.config(state=tk.DISABLED)

def update\_processes\_plot(self):

"""Обновление графиков процессов"""

if 'processes' not in self.figures:

return

# Очистка фрейма

for widget in self.processes\_canvas\_frame.winfo\_children():

widget.destroy()

# Создание canvas для matplotlib

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

canvas = FigureCanvasTkAgg(self.figures['processes'], master=self.processes\_canvas\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

def update\_primary\_plot(self):

"""Обновление графиков первичных частиц"""

if 'primary' not in self.figures:

label = ttk.Label(self.primary\_canvas\_frame, text="Нет данных о первичных частицах",

font=('Arial', 12), foreground="gray")

label.pack(expand=True)

return

# Очистка фрейма

for widget in self.primary\_canvas\_frame.winfo\_children():

widget.destroy()

# Создание canvas для matplotlib

canvas = FigureCanvasTkAgg(self.figures['primary'], master=self.primary\_canvas\_frame)

canvas.draw()

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

def update\_secondary\_plot(self):

"""Обновление графиков вторичных частиц"""

if 'secondary' not in self.figures:

label = ttk.Label(self.secondary\_canvas\_frame, text="Нет данных о вторичных частицах",

font=('Arial', 12), foreground="gray")

label.pack(expand=True)

return

# Очистка фрейма

for widget in self.secondary\_canvas\_frame.winfo\_children():

widget.destroy()

# Создание canvas для matplotlib

canvas = FigureCanvasTkAgg(self.figures['secondary'], master=self.secondary\_canvas\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

def refresh\_files\_list(self):

"""Обновление списка файлов"""

if self.output\_dir is None or not self.output\_dir.exists():

return

self.files\_listbox.delete(0, tk.END)

# Получение списка всех файлов рекурсивно

all\_files = []

for root, dirs, files in os.walk(self.output\_dir):

for file in files:

filepath = Path(root) / file

relative\_path = filepath.relative\_to(self.output\_dir)

size = filepath.stat().st\_size / 1024 # Размер в KB

all\_files.append((str(relative\_path), size, filepath))

# Сортировка по пути

all\_files.sort(key=lambda x: x[0])

# Добавление в список

for relative\_path, size, filepath in all\_files:

if size < 1024:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

self.files\_listbox.insert(tk.END, f"{relative\_path} ({size:.1f} KB)")

else:

self.files\_listbox.insert(tk.END, f"{relative\_path} ({size / 1024:.1f} MB)")

# Сохраняем полные пути для открытия

self.file\_paths = {str(f[0]): f[2] for f in all\_files}

def open\_selected\_file(self):

"""Открытие выбранного файла"""

selection = self.files\_listbox.curselection()

if not selection:

# messagebox.showwarning("Предупреждение", "Выберите файл из списка")

return

# Получаем относительный путь из строки (до первой скобки)

selected\_text = self.files\_listbox.get(selection[0])

relative\_path = selected\_text.split(' (')[0]

# Получаем полный путь

if not hasattr(self, 'file\_paths') or relative\_path not in self.file\_paths:

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

# messagebox.showerror("Ошибка", "Файл не найден")

return

filepath = self.file\_paths[relative\_path]

if not filepath.exists():

# messagebox.showerror("Ошибка", "Файл не найден")

return

# Открытие файла с помощью программы по умолчанию

try:

if sys.platform == 'win32':

os.startfile(filepath)

elif sys.platform == 'darwin':

subprocess.run(['open', filepath])

else:

subprocess.run(['xdg-open', filepath])

except Exception as e:

pass

# messagebox.showerror("Ошибка", f"Не удалось открыть файл: {str(e)}")

def open\_output\_folder(self):

"""Открытие папки с результатами"""

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

if self.output\_dir is None or not self.output\_dir.exists():

# messagebox.showwarning("Предупреждение", "Папка с результатами не создана")

return

try:

if sys.platform == 'win32':

os.startfile(self.output\_dir)

elif sys.platform == 'darwin':

subprocess.run(['open', self.output\_dir])

else:

subprocess.run(['xdg-open', self.output\_dir])

except Exception as e:

pass

# messagebox.showerror("Ошибка", f"Не удалось открыть папку: {str(e)}")

def main():

"""Запуск GUI приложения"""

setup\_log("geant4\_parser") # Лог перенаправлен в output/geant4\_parser\_log/...

root = tk.Tk()

app = Geant4GUI(root)

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б.**

root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main()