МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
«Гомельский государственный технический университет
имени П.О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 01 02-01 «Информационные системы и технологии (в проектировании и производстве)»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломной работе

на тему «Компьютерное моделирование распределения электромагнитных полей в тонких стеклянных пластинках с включением частиц восстановленных металлов»

Разработал ст. гр. ИТ-51 _____ Кухаренко А. А.
(подпись) (Ф.И.О.)

Руководитель работы _____ доцент, к.т.н., Курочка К. С.
(подпись) (учёное звание, учёная степень, Ф.И.О.)

Консультант по
экономической части _____ доцент, к.э.н., Кожевников Е. А.
(подпись) (учёное звание, учёная степень, Ф.И.О.)

Консультант по охране труда и
технике безопасности _____ к.т.н. Кротенок В.В.
(подпись) (учёное звание, учёная степень, Ф.И.О.)

Нормоконтроль _____ доцент, д.т.н. Мурашко И.А.
(подпись) (учёное звание, учёная степень, Ф.И.О.)

Рецензент _____ _____
(подпись) _____
_____
_____
(учёное звание, учёная степень, должность, организация Ф.И.О.)

Дипломная работа (_____стр.,_____листов) допущена
к защите в Государственной экзаменационной комиссии.

Зав. кафедрой
«Информационные технологии» _____ доцент, к.т.н. Курочка К.С.
(подпись) (учёное звание, учёная степень, Ф.И.О.)

Гомель 2012

РЕФЕРАТ

Дипломная работа: 199 страниц, 23 рисунка, 20 таблиц, 82 источника, 13 приложений.

Ключевые слова: компьютерное моделирование, метод конечных элементов (МКЭ), нанокомпозитные материалы, наночастицы, электромагнитные поля, программное обеспечение, математическое моделирование.

Объектом исследования является пластинка, выполненная из нанокомпозитного материала с включением частиц восстановленных металлов, предметом изучения является распределение электромагнитного поля в материале.

Цель работы: разработка программного продукта для компьютерного моделирования стеклянных пластинок с включением частиц восстановленных металлов.

В процессе работы было сделано: выполнен анализ существующих методов исследования распределения электромагнитных полей, построена математическая модель на основе векторного метода конечных элементов, разработана программа для компьютерного моделирования (EMFFM) и спецификация языка описания модели для разработанного программного продукта (Input API).

Основными функциями разработанного ПО являются:

– унифицированный и стандартизированный формат ввода данных;

– возможность задания автоматического сценария выполнения программы;

– реализация компьютерного моделирования с использованием векторного метода конечных элементов;

– возможность визуализации исследуемой геометрии.

Элементами научной новизны является применение векторного метода конечных элементов для компьютерного моделирования нанокомпозитных материалов с включением частиц восстановленных металлов.

Областью возможного практического применения является синтез материалов с заданными свойствами.

Приведенный в дипломной работе расчетно-аналитический материал получен самостоятельно и объективно отражает состояние исследуемого процесса, все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на их авторов.

ЗАДАНИЕ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ

РЕЗЮМЕ

Тема работы – компьютерное моделирование распределения электромагнитных полей в стеклянных пластинках с включением частиц восстановленных металлов.

Объектом исследования является пластинка, выполненная из нанокомпозитного материала с включением частиц восстановленных металлов, исследуемая с помощью моделирования распределения электромагнитного поля.

Целью работы является разработка программного продукта для компьютерного моделирования стеклянных пластинок с включением частиц восстановленных металлов.

Основными результатами являются математическая модель и созданный программный продукт для исследования распределений ЭМП на основе ВМКЭ.

РЭЗЮМЭ

Тэма працы – кампутарнае мадэляванне размеркавання электрамагнітных палёў у шкляных пласцінках з уключэннем часціц адноўленых металаў.

Аб'ектам даследавання з'яўляецца пласцінка, выкананая з нанокомпазіту з уключэннем часціц адноўленых металаў з дапамогай мадэлявання размеркавання электрамагнітнага поля.

Мэтай працы з'яўляецца распрацоўка праграмнага прадукта да кампутарнага мадэлявання шкляных пласцінак з уключэннем часціц адноўленых металаў.

Асноўнымі вынікамі з'яўляюцца матэматычная мадэль і створаны праграмны прадукт для даследавання размеркаванняў ЭМП на аснове ВМКЭ.

Abstract

R & D – computer simulation of electromagnetic field distribution in the glass plates with the inclusion of particles of reduced metal.

The object of this study is the plate made of nanocomposite with the inclusion of particles of reduced metal by simulation of the electromagnetic field distribution.

The aim is to develop software for the simulation of glass plates with the inclusion of particles of reduced metal.

The main results are a mathematical model and a software product developed to study the distributions of electromagnetic fields (EMF) based on vector finite element method (VFEM).

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

В настоящей пояснительной записке применяются следующие термины, обозначения и сокращения.

Нанооптика – наука, занимающаяся исследование оптических эффектов в наноструктурных объектах.

Наноплазмоника – раздел нанооптики, предметами которого являются оптические свойства металлических частиц и наноструктур, которые обусловлены колебаниями электронов проводимости относительно кристаллической решетки.

Нанотехнология – это совокупность методов и приемов структурирования вещества на атомном и молекулярном уровнях с целью производства конечных продуктов с заранее заданной атомной структурой.

ЕСПД – единая система программной документации.
ММ – математическая модель.
НДС – налог на добавленную стоимость.
ООП – объектно-ориентированное программирование.
ОС – операционная система.
ПК – персональный компьютер.
ПО – программное обеспечение.
ПП – программный продукт.
СЛАУ – система линейных алгебраических уравнений.
ЭМП – электромагнитное поле.

ABC (Absorbing Boundary Conditions) – поглощающие граничные условия.

CAD (Computer-Aided Design) – средства автоматизированного проектирования, предназначенные для автоматизации двумерного и/или трехмерного геометрического проектирования.

CAE (Computer-Aided Engineering) – общее название для программ, предназначенных для решения различных инженерных задач: расчётов, анализа и симуляции физических процессов.

CDA (Coupled Dipole Approximation) – приближение взаимодействующих диполей.

DDA (Discrete Dipole Approximation) – приближение дискретных диполей.

FDTD (Finite-Difference Time-Domain) – метод конечных разностей во временной области.

FEM (Finite Element Method) – метод конечных элементов (МКЭ).

GPGPU (General-purpose graphics processing units – GPU общего назначения) – методика использования графического процессора видеокарты, который обычно имеет дело с вычислениями только для компьютерной

графики, для выполнения общих вычислений, которые обычно производятся центральным процессором.

GUI (Graphical User Interface) – графический интерфейс пользователя. Представляет собой визуальную среду, через которую производится взаимодействие пользователя с функциями компьютера.

IDE (Integrated development environment или integrated debugging environment) – интегрированная среда разработки, которая позволяет упростить процесс разработки и проектирования, связать используемые технологии и средства в едином месте.

LINQ (Language Integrated Query – язык интегрированных запросов) – компонент Microsoft .NET Framework, встраивает синтаксис запросов в язык программирования C# (VB.NET) и обеспечивает возможность доступа к различным источникам данных, используя один и тот же код, причем делает это возможным за счет дополнительного уровня абстракций.

MOM (Method of Moments – метод моментов) – метод граничных элементов.

MPI (Message Passing Interface, интерфейс передачи сообщений) – программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу.

OpenCL (Open Computing Language – открытый язык вычислений) – фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических (GPU) и центральных процессорах (CPU). В фреймворк OpenCL входят язык программирования, который базируется на стандарте C99, и интерфейс программирования приложений (англ. API). OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных и является реализацией техники GPGPU. OpenCL является полностью открытым стандартом, его использование не облагается лицензионными отчислениями.

PML (Perfectly Matched Layer) – идеально согласованный слой.

TPL (Task Parallel Library) – библиотека параллельных задач, появившаяся в .NET 4, предоставляющая новые удобные средства для использования параллелизма в программах.

VFEM (Vector Finite Element Method) – векторный метод конечных элементов (ВМКЭ).

XAML (eXtensible Application Markup Language – расширяемый язык разметки приложений) – основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft.

WPF (Windows Presentation Foundation, кодовое название – Avalon) – система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), имеющая прямое отношение к XAML.

ВВЕДЕНИЕ

Современное развитие науки и техники приводит ко все большему использованию нанокомпозитов с частицами восстановленных металлов.

В настоящее время они используются в качестве:

– оптических сенсоров элементов для селективного детектирования газообразных и (или) жидких веществ;
– спектральных фильтров покрытия;
– преобразователей и усилителей излучений;
– управляющих оптических и (или) оптоэлектронных элементов;
– приборов для регистрации оптических сигналов;
– и других.

Однако этот процесс не останавливается и промышленность постоянно требует новых материалов. Для создания любого нанообъекта нужно сначала разработать его структуру и технологию. При этом приходится сталкиваться с дорогостоящими и затратными методами: методами нанохимии, газофазного синтеза, нанолитографическими способами, сфокусированными электронными пучками и др. [1] Чтобы избежать бесполезного конструирования и сборки многочисленных прототипов наносистем, понять, какая из них будет работать, а какая нет, используют методы компьютерного моделирования. С помощью него, в основе которого большое количество экспериментальной и теоретической информации, можно в рамках определенных границ описать поведение проектируемых систем. В ряде случаев именно оно служит катализатором для экспериментальных исследований и производства. Если ранее оно было направлено на количественное описание процессов в материалах, то сейчас большое внимание уделяется созданию новых перспективных материалов и прогнозированию их свойств.

Основным предметом исследования наноструктурных объектов является электромагнитное поле (ЭМП), которое позволяет исследовать оптические и физические эффекты, возникающие при воздействии света.

Среди типовых задач моделирования распределения ЭМП в нанокомпозитах можно выделить следующие:

– расчет распределения ЭМП оптического диапазона в 2D и 3D в ближней зоне металлической наночастицы;
– расчет распределения ЭМП в 2D и 3D в ближней зоне наноконцентраторов полей – ансамблей близкорасположенных наночастиц одинакового или разного состава, имеющих различную форму;
– расчет эффективных характеристик в 2D и 3D в объемах, содержащих большое количество наночастиц, распределенных по известному закону и имеющих различную форму;
– расчет распределения ЭМП, созданного одним или несколькими точечными источниками (диполями), вблизи металлических наночастиц.

В настоящее время не существует унифицированного метода и программного обеспечения (ПО), позволяющего решать все данные классы задач, что требует от исследователя не только знаний в области физики и нанотехнологии, но и умение ориентироваться в современном ПО и быстро обучаться работе с ним.

Согласно вышеизложенного можно отметить следующее:

– существует необходимость в создании математической модели, позволяющей адекватно описать нанокомпозиты с включениями частиц восстановленных металлов;

– необходимо выбрать метод, позволяющий эффективно реализовать решение построенной модели.

Для проведения компьютерного моделирования необходима разработка программного обеспечения, позволяющего реализовать модель задачи и выбранный метод для ее решения.

В результате проделанной работы были рассмотрены существующие методы решения задач о распределении ЭМП, проанализированы существующие программные продукты (ПП), разработана структура программного комплекса и выполнена его разработка.

1 ИССЛЕДОВАНИЕ ЭЛЕКТРОМАГНИТНЫХ ПОЛЕЙ С ПОМОЩЬЮ МОДЕЛИРОВАНИЯ

1.1 Общие принципы моделирования

Моделирование – процесс исследования моделей, которые представляют собой абстрактные объекты, заменяющие реальные. В зависимости от вида исследуемого объекта и средств, можно выделить следующие типы моделирования:

- математическое;
- имитационное
- компьютерное.

Выделяют предмет, объект и средства моделирования. Объект представляет собой изучаемую систему, которую описывают в виде модели. Предметом является некоторое явление или процесс, воздействие которого на объект является целью проведения моделирования. Средства представляют собой конкретные методы, которые могут быть применены для моделирования. Кроме того, присутствует и субъект или исследователь, который проводит выполнение всех этапов и проводит анализ результатов.

Моделирование всегда предполагает принятие допущений той или иной степени важности. При этом должны удовлетворяться следующие требования к моделям:

- *адекватность*, то есть соответствие модели исходной реальной системе и учет, прежде всего, наиболее важных качеств, связей и характеристик. Оценить адекватность выбранной модели, особенно, например, на начальной стадии проектирования, когда вид создаваемой системы ещё неизвестен, очень сложно. В такой ситуации часто полагаются на опыт предшествующих разработок или применяют определенные методы, например, метод последовательных приближений;
- *точность*, то есть степень совпадения полученных в процессе моделирования результатов с заранее установленными, желаемыми. Здесь важной задачей является оценка с требуемой точностью результатов и имеющейся точности исходных данных, согласование их как между собой, так и с точностью используемой модели;
- *универсальность*, то есть применимость модели к анализу ряда однотипных систем в одном или нескольких режимах функционирования, что позволяет расширить область применимости модели для решения большего круга задач;
- *целесообразная экономичность*, то есть точность получаемых результатов и общность решения задачи должны увязываться с затратами на моделирование. Удачный выбор модели, как показывает практика – результат компромисса между отпущенными ресурсами и особенностями используемой модели;

– *расширяемость*, которая предусматривает возможность модификации модели и введение в нее новых параметров и свойств.

В зависимости от сложности и степени теоретического обоснования используются следующие модели: “белый ящик”, “черный ящик”, “серый ящик” [2].

Если физические процессы, протекающие на рассматриваемых объектах, описываются множеством нелинейных уравнений, когда теоретическая основа для расчетных моделей является прозрачной, построенной на известных физических и химических законах и свойствах, то такая модель может быть представлена как “белый ящик” [2].

Модели ”черный ящик” основаны на наличии экспериментальных данных и не требуют никакой априорной информации. Они достаточно хорошо изучены и просты для работы в реальном масштабе времени. В то же время такие модели должны регулярно обновляться с появлением новых экспериментальных данных.

Моделирование на основе “черного ящика” иногда используется как синоним понятия идентификации системы. Иными словами, идентификация системы – это теория разработки математических моделей динамических систем по результатам измерений.

Модель “серый ящик” является сбалансированной системой, которая по своей сущности не что иное, как компромисс между сложностью модели “белый ящик” и возможностями модели “черный ящик” по прогнозированию процессов.

Одним из существенных понятий в модели “серый ящик” являются так называемые базисные элементы, включающие имеющуюся информацию о поведении системы в виде простых аналитических функций и выражений. Вид этих элементарных функций увязывается с ее поведением. Базисные элементы могут иметь в модели системы разнообразные формы и подвергаться изменениям (мутациям).

Важно обратить внимание на то, что выбор методов моделирования очень сильно зависит от типа исследуемой системы и задачи, для которой эта модель создается. Непрерывные системы лучше всего моделируются дифференциальными уравнениями, в случае необходимости дополненными алгебраическими связями.

Процесс моделирования можно разделить на 5 этапов (рисунок 1.1) [2].

Рисунок 1.1 – Схема процесса моделирования

Первый этап – построение модели. Сначала выбирается физическая модель и проводится разделение всех действующих в рассматриваемом явлении факторов на главные, обязательные для учета и второстепенные (на данном этапе исследования они могут быть отброшены). Одновременно формулируются допущения или рамки применимости модели, в которых будут справедливы полученные на ее основе результаты. Эта модель записывается в математических терминах. В результате получается абстракция реального объекта или системы.

На втором этапе модель выступает как самостоятельный объект исследования. Одной из форм такого исследования является проведение модельных экспериментов, при которых сознательно изменяются условия функционирования модели и систематизируются данные о её “поведении”. Конечным результатом этого этапа является множество (совокупность) знаний о модели.

На третьем этапе осуществляется перенос знаний с модели на оригинал – формирование множества знаний. Одновременно происходит переход с “языка” модели на “язык” оригинала, т.е. полученные результаты с абстрактной модели переносятся на исходную физическую систему реального мира. Процесс переноса знаний проводится по определенным правилам. Знания о модели должны быть скорректированы с учетом тех свойств объекта-оригинала, которые не нашли отражения или были изменены при построении модели.

Четвёртый этап – практическая проверка получаемых с помощью моделей знаний и их использование для построения обобщающей теории объекта, его преобразования или управления им. Здесь используются

полученные аналитические данные, результаты других аналогичных исследований, существующие экспериментальные данные и т.д., которые можно использовать для подтверждения адекватности проведенного моделирования.

Пятый этап – производится корректировка модели в случае необходимости и вводятся новые параметры.

Моделирование – циклический процесс. Это означает, что за первым четырёхэтапным циклом может последовать второй, третий и т. д. При этом знания об исследуемом объекте расширяются и уточняются, а исходная модель постепенно совершенствуется. Недостатки, обнаруженные после первого цикла моделирования, обусловленные малым знанием объекта или ошибками в построении модели, можно исправить в последующих циклах.

1.1.1 Для моделирования сложных систем и объектов сегодня применяют компьютерное моделирование, которое позволяет значительно сократить расходы на проведение исследований и уменьшить время получения результата. Оно построено на основе алгоритмического представления моделей, что обусловлено необходимостью хранения информации в дискретном виде, воспроизводимом на компьютере.

Большинство применяемых аналитических средств, таких как дифференциальное исчисление, больше всего подходят для исследования линейных задач. Формально это уравнения, в которые переменные входят только в первой степени; реально они описывают процессы, которые протекают одинаково при разных воздействиях. Область применения линейных уравнений необычайно широка. Она охватывает классическую и квантовую механику, электродинамику и теорию волн. Для их решения разработано достаточно методов, которые обладают высокой эффективностью.

Однако множество природных процессов – нелинейны, так что малые изменения одной величины могут привести к неожиданно большим изменениям в другой и будет иметь место качественно иное поведение системы. Поскольку нелинейные задачи удается решать аналитическими методами только в редких случаях, то появляется необходимость в численных методах, позволяющих исследовать нелинейные явления.

1.2 Моделирование электромагнитных полей в наноструктурах

Исследования наноструктурных систем показали, что для их изучения можно применять электромагнитное поле, которое описывается уравнениями классической электродинамики. В частности, при рассмотрении оптических свойств они также могут использоваться. К этому обстоятельству приводит и тот факт, что распространение света можно описать как электромагнитную волну, с той лишь разницей, что элементарными частицами для света принимаются фотоны, а для электродинамики электроны.

При исследовании наночастиц необходимо учитывать их особые свойства [3, раздел 3], для которых были выявлены аналитические зависимости. Здесь находят свое применение теории Друде-Зоммерфельда [3], Друде-Лоренца [4], которые описывают влияние частоты волны на электрическую постоянную $\varepsilon(\omega)$. По мере увеличения частоты света поглощение увеличивается, и в оптической области это приближение оказывается не вполне адекватным. В ультрафиолетовой области некоторые металлы оказываются прозрачными (Na), в то время как другие (Ag, Au) из-за межзонных переходов являются сильно поглощающими. Физической причиной столь сильной зависимости диэлектрической проницаемости от частоты является изменение в фазе индуцированных в металле токов относительно фазы падающего света в области частот, близких к так называемой плазмонной частоте. Такие эффекты рассматриваются в наноплазмонике [3].

Классическая электродинамика [5, 6], основанная на уравнениях Максвелла, лежит в основе многочисленных приложений электро- и радиотехники, СВЧ и оптики. До настоящего времени не было обнаружено ни одного эффекта, который потребовал бы видоизменения уравнений. Они оказываются применимы и в квантовой механике, когда рассматривается движение, например, заряженных частиц во внешних электромагнитных полях. Поэтому уравнения Максвелла [3] являются основой микроскопического описания электромагнитных свойств вещества.

В дифференциальной форме уравнения Максвелла можно записать в виде:

– теорема о циркуляции магнитного поля

$$\frac{d}{dt} D(x,t) + J(x,t) = \nabla \times H(x,t), \qquad (1.1)$$

– закон индукции Фарадея

$$\frac{d}{dt} B(x,t) = \nabla \times E(x,t), \qquad (1.2)$$

– закон Гаусса

$$\nabla \cdot D(x,t) = \rho, \qquad (1.3)$$

– закон Гаусса для магнитного поля

$$\nabla \cdot B(x,t) = 0, \qquad (1.4)$$

где $E(x,t)$ – электрическое поле;

$H(x,t)$ – вспомогательное магнитное поле (часто его называют просто магнитным полем);

$D(x,t)$ – электрическая индукция;

$J(x,t)$ – плотность электрического тока (плотность тока проводимости);

ρ – плотность стороннего электрического заряда.

В среде сторонние электрические и магнитные поля вызывают поляризацию и намагничивание вещества, которые макроскопически

описываются соответственно вектором поляризации P и вектором намагниченности M вещества, и вызваны появлением связанных зарядов ρ_b и токов j_b. В результате поле в среде оказывается суммой внешних полей и полей, вызванных связанными зарядами и токами:

$$\rho_b = -\nabla \cdot P, \tag{1.5}$$

$$j_b = \nabla \times M + \frac{\partial P}{\partial t}. \tag{1.6}$$

Поляризация P и намагниченность вещества M связаны с векторами напряжённости и индукции электрического и магнитного поля следующими соотношениями:

$$D = \varepsilon_0 E + P, \tag{1.7}$$

$$B = \mu_0 (H + M), \tag{1.8}$$

где ε_0 – диэлектрическая проницаемость в вакууме (электрическая постоянная);

μ_0 – магнитная постоянная.

Уравнения Максвелла содержат в себе законы сохранения заряда и энергии электромагнитного поля. Источники полей (ρ, J) не могут быть заданы произвольным образом. Уравнение непрерывности для зарядов и токов может быть записано в виде:

$$\nabla \cdot J + \frac{\partial \rho}{\partial t} = 0. \tag{1.9}$$

На практике в материальных уравнениях обычно используются экспериментально определяемые коэффициенты (зависящие в общем случае от частоты электромагнитного поля), которые собраны в различных справочниках физических величин.

В слабых электромагнитных полях, сравнительно медленно меняющихся в пространстве и во времени, в случае изотропных, неферромагнитных и несегнетоэлектрических сред справедливо приближение, в котором поляризуемость и намагниченность линейно зависят от приложенных полей:

$$P = \varepsilon_0 \chi_e E, \tag{1.10}$$

$$M = \chi_m H, \tag{1.11}$$

где введены безразмерные константы: χ_e – диэлектрическая восприимчивость и χ_m – магнитная восприимчивость вещества (в системе единиц СИ эти константы в 4π раз больше, чем в гауссовой системе СГС). Соответственно, материальные уравнения для электрической и магнитной индукций записываются в следующем виде:

$$D = \varepsilon_0 \varepsilon E = \varepsilon_0 (1 + \chi_e) E, \tag{1.12}$$

$$B = \mu_0 \mu E = \mu_0 (1 + \chi_m) H, \tag{1.13}$$

где ε – относительная диэлектрическая проницаемость;

μ – относительная магнитная проницаемость.

Размерные величины $\varepsilon_0\varepsilon$ (в единицах СИ – Ф/м) и $\mu_0\mu$ (в единицах СИ – Гн/м), возникающие в системе СИ, называются абсолютная диэлектрическая проницаемость и абсолютная магнитная проницаемость соответственно.

Рассмотренные выше уравнения относятся к “классическим” материальным уравнениям. В настоящее время активно развиваются технологии изготовления искусственных, наноструктурных материалов, которые получили название – метаматериалы [4, 7, 8]. В них материальные соотношения могут иметь более сложный, чем представленный в выражениях (1.12) и (1.13) вид. Такие материалы еще до конца не исследованы, но исследования ведутся почти во всех странах мира.

Для решения уравнений Максвелла было разработано достаточно методов, но основной популярностью среди них пользуются FDTD, DDA, Т-матрицы. Эти методы позволяют проводить моделирование распределения ЭМП и позволяют проводить процесс на компьютере. Применение методов описано в [3, 4, 8 – 41]. Краткое описание методов приведено в приложении А.

1.3 Существующее ПО для компьютерного моделирования распределения электромагнитного поля

Для решения задач о распределении ЭМП существует целый ряд программных комплексов, которые используются в производстве и исследованиях. Среди них выделяются как свободные, так и коммерческие продукты. Существующее ПО построено на основе численных методов, рассмотренных в приложении А. В таблице 1.1 представлены некоторые из популярных продуктов.

Таблица 1.1 – Популярное ПО для решения задач о распределении ЭМП

Название	Метод	Язык реализации	Тип ПО
MEEP	FDTD	C, C++	GPL v.2
EMAP	FEM, VFEM, VFEM / MOM	C	Open Source
ELCUT	FEM	C++	Commercial, Academic
A-DDA	DDA	Fortran 90, C99	GPL v.3
DDSCAT	DDA	Fortran 90	GPL
Elmer	FEM	C++, Fortran 90, C	GPL
Maxwell	FEM	C++	Commercial
Femlab	FEM	C++	Commercial

Основными языками программирования, на которых ведется разработка являются C/C++ и Fortran.

1.3.1 Среди существующего ПО имеется достаточное количество коммерческих продуктов. Можно отметить следующие: Ansys Maxwell, Vector Field Opera, Ansys Emag, Ansys Multiphysics, COMSOL Multiphysics Femlab, Abaqus Unified FEA. Коммерческие продукты обладают большим количеством возможностей, являются довольно универсальными, имеют развитые средства для визуализации и проведения моделирования. Такие пакеты основаны на хорошо зарекомендовавших себя методах (приложение А). В них реализованы довольно качественные алгоритмы.

Стоимость подобных пакетов начинается от нескольких тысяч долларов. Самые многофункциональные могут стоить десятки и сотни тысяч. Программы обладают возможность интеграции в производственный процесс, что также требует денежных и временных затрат. Масштабируемость позволяет использовать их на различных этапах производства и для решения разнообразных задач.

Для разработки таких комплексов используются математические модели, которые хорошо изучены. Проведение моделирования основано на известных закономерностях и зависимостях, хорошо изученных и апробированных на различных задачах. Это означает то, что при решении определенного класса задач, которые имеют специфичные (уникальные) особенности, а также содержат неизученные объекты, результаты нельзя интерпретировать адекватно. Так как коды программ не предоставляются и вносить изменения в модели невозможно, то это сразу ограничивает спектр применимости такого ПО. Именно поэтому коммерческие продукты не используются при исследованиях и в моделировании новых материалов и структур.

Все пакеты для конструирования и проектирования разнообразных систем включают в себя пакеты для проведения конечноэлементного моделирования (CAE), использующие метод конечных элементов (МКЭ). Этот метод является наиболее популярным для моделирования в различных областях.

Кроме приведенных недостатков можно добавить:

– большие денежные и временные затраты на интеграцию и изучение продукта, а также подготовку сотрудников и обслуживающего персонала;

– большие затраты времени на подготовку задачи;

– иногда, требуются сторонние коммерческие проекты (например, для рисования геометрии задачи в CAD пакете).

При выборе продукта необходимо исходить из требований и типа задач, которые должен решать продукт. С этой целью необходимо производить маркетинговые исследования, позволяющие выявить наиболее эффективный и подходящий комплекс, обеспечивающий наилучшие показатели по необходимым критериям.

1.3.1.1 ELCUT широко используется в научных исследованиях, промышленности и образовании. ELCUT [42] – это современный комплекс программ для компьютерного моделирования электромагнитных, тепловых и механических задач методом конечных элементов (МКЭ). Программа предлагает дружественный пользовательский интерфейс, возможность простого описания моделей, широкие аналитические возможности комплекса и высокую степень автоматизации всех операций. Начать работу с ELCUT можно практически сразу, не отвлекаясь на изучение математических основ вычислительных алгоритмов и особенностей их реализации. В англоязычном варианте программа носит название Quick Field. Программы полностью совместимы друг с другом. Как по интерфейсу, так и по формату файлов. Текущая версия программы Elcut – 5.9, а Quick Field – 5.10.

Elcut позволяет решать следующие типы задач (определять следующие типы полей):

– магнитостатика (в этой задаче рассчитывается магнитное поле постоянных магнитов, а также проводников с постоянным током в среде с заданными магнитными свойствами);

– электростатика (в этой задаче рассчитывается электрическое поле зарядов, заданных значений потенциала в среде с заданными электрическими свойствами);

– растекание токов (в этой задаче рассчитывается распределение электрического потенциала и тока в системах проводников);

– магнитное поле переменных токов (расчёт электрического и магнитного поля, возбуждённого приложенными переменными синусоидальными токами или внешним магнитным переменным полем);

– температурное поле (расчёт температурного поля в среде с заданной теплопроводностью и граничными условиями первого-четвёртого рода в статике;

– нестационарная теплопередача (расчёт динамики тепловых процессов);

– задачи теории упругости.

Программа Elcut позволяет также производить связанные расчёты, т.е. вычисленные параметры в одной задаче передать в другую задачу в качестве исходных данных.

Расчёты производятся в двумерной плоской или осесимметричной постановке задач. В плоской постановке задачи геометрическая модель представляет собой сечение бесконечно протяжённой в плоскость чертежа системы, в осесимметричной – половину осевого сечения тела вращения. При этом ось симметрии располагается на линии с координатами $r = 0$.

Результаты расчета представляются в достаточно понятной и полной форме для определенного типа задач.

Среди особенностей и недостатков можно выделить:

– строго двухмерная постановка решаемой задачи;

– высокая стоимость программы (неоправданно высокая);

– закрытость исходного кода, что не дает полноценной возможности использования его в процессе исследования;

– достаточно сложное задание задачи, путем выполнения множества этапов.

1.3.2 Для проведения научных исследований наибольшее применение нашли свободно распространяемые продукты. Эти продукты предоставляют возможность модификации исходных кодов, их использование в своих приложениях, а также поддержку со стороны разработчиков. Причем в виде свободного ПО есть достаточное количество библиотек, которые предоставляют возможность использования их для разработки полноценного программного комплекса. Среди наиболее применяемых методов выделяются FDTD и DDA.

Большинство разработанных программ явились результатами научно-исследовательской деятельности и разработка их велась достаточно большой командой.

1.3.2.1 Elmer [43] это программа для компьютерного моделирования на основе МКЭ задач механики, электродинамики, гидродинамики. Программа предоставляет графический интерфейс, поддержку нескольких стандартов для сеток, собственный генератор сеток, решатель (в программе предусмотрено несколько типов решателей, которые построены на разных типах уравнений). Интерфейс пользователя позволяет визуально анализировать задачу, просматривать сетку, заданные условия на границах и получаемые результаты.

Решение основано на узловом методе конечных элементов (МКЭ). Для аппроксимации элементов используются полиномиальные функции, описанные для каждого типа элемента.

Задание задачи и ее условий основано на алгоритмическом способе – путем описания небольшой программы. Для такого описания используется ECMA Script [44].

Решение получаемых в процессе моделирования систем уравнений можно производить с применением распределенных вычислений на базе технологии MPI.

Среди недостатков можно отметить достаточно своеобразное описание решаемой задачи, которое не позволяет быстро ее задавать, для этого требуется изучение документации в достаточном объеме. Для построения сеток нет удобного средства, позволяющего единожды описать геометрию, но использовать для генерации различные типы сеток. Достаточно упрощенная формулировка основных решающих уравнений по типам задач. Примеры, идущие в комплекте с программой, не доведены до того, чтобы их можно было сразу использовать для решения тестовых задач.

1.3.2.2 MEEP (MIT Electromagnetic Equation Propagation) [45] – программное обеспечение, предназначенное для исследования электромагнетизма методом конечных разностей во временной области (FDTD). Программа разработана в Массачусетском технологическом институте (MIT) и использует наработки, созданные в институте на протяжении исследований электромагнетизма. Проект изначально предназначался для исследования фотонных кристаллов, но затем нашел применение в других задачах электродинамики (в качестве основы использовался проект MBP (MIT Photonic Bands) [46]).

Среди основных особенностей программы можно отметить следующие:

– распространение по лицензии GNU GPL;

– возможность моделирования задач в одно-, двух- и трехмерном пространствах, а также с использованием цилиндрических координат;

– использование параллельных вычислений благодаря поддержке стандарта MPI. Работает на любых UNIX-подобных системах;

– возможность задания произвольных значений диэлектрической проницаемости ε и магнитной проницаемости μ, включая дисперсию (зависимость этих величин от частоты $\varepsilon(\omega)$ и $\mu(\omega)$), включая потери (как положительные, так и отрицательные) и нелинейность (типа Керра [7] или Покельса [7]) диэлектрических и магнитных материалов, а также электрическую и магнитную проводимость σ;

– имеется возможность использования PML в качестве поглощающих граничных условий, а также периодические условия Блоха [47];

– позволяет использовать симметрию задачи для уменьшения размера расчетной области – четную и нечетную зеркальную симметрию, а также симметрию вращения на 90 или 180 градусов;

– поддерживает создание и использование скриптов – как с использованием языка Scheme [48] (через библиотеку libctl или MPB), так и вызываемых через библиотеки, написанные на C++. Имеется также интерфейс для Python;

– поддерживает вывод рассчитанных полей в стандартном научном формате данных HDF5, который поддерживается многими средствами визуализации;

– позволяет задавать произвольное распределение материалов и источников;

– включает полезные надстройки для анализа рассчитанных полей, такие как анализ спектра, извлечение данных на заданных частотах и вычисление интегралов энергии. Функции полностью программируемые;

– поддерживает многопараметрическую оптимизацию, поиск корней, интегрирование и т.п.

Конечно-разностное моделирование электромагнитного поля во временной области основано на пошаговом режиме, когда за один шаг выполняется расчет параметров поля в определенный период времени. При

этом рассматривается некоторая ограниченная область, в которой и рассчитываются параметры. Иначе говоря, это разновидность численного эксперимента. Его можно использовать для расчета большого количества характеристик, основные из которых можно описать следующим образом:

– расчет спектров отражения и пропускания – проводится путем моделирования прохождения по системе короткого импульса и последующим вычислением Фурье-преобразования рассчитанных полей. Один единственный прогон импульса может дать информацию об амплитудах рассеянных полей в широком диапазоне частот;

– расчет резонансных частот и соответствующих им мод – проводится путем моделирования прохождения по системе короткого импульса с последующим вычислением скорости затухания на различных частотах и расчетом полей гармонических мод системы (включая как волноводы, так и резонаторы, а также включая системы с потерями);

– расчет распределений полей, возникающих в результате воздействия произвольных источников.

Учитывая многочисленные возможности программы, она является одним из самых популярных комплексов, применяемых для решения задач электромагнетизма. Комплекс находит свое применение гораздо чаще, чем любые коммерческие программы. Среди недостатков можно отметить следующие:

– отсутствие в комплексе средства визуализации, причем как для задания входных данных, так и анализа выходных;

– сложность описания файлов *.ctl с условиями задачи, необходимо хорошее знание документации и тонкостей программы;

– тяжеловесность самой программы и многочисленность ее модулей также вносят сложности в использование;

– в существующей документации, которая представлена в виде сайта, плохо освещены вопросы использования программы, а также тонкости, которые необходимо знать при составлении моделей для решаемых задач;

– скорость работы и решаемые задачи имеют свои ограничения (не смотря на возможности), как в связи с возможностями самого метода FDTD, так и реализацией в программе;

– нет поддержки моделирования в частотной области;

– сложность в описании параметров для дисперсионных материалов, например, для наночастиц, исследуемых в оптическом диапазоне волн. Трудно описать свойства металлов с использованием теории Друде-Зоммерфельда и других для известных металлов (Ag, Au, Cu) [3];

– применение своей системы единиц измерения, которая не является близкой ни к СИ, ни к СГС. Требуется время на переход к подобной системе и умение конвертировать задачу из системы СИ в систему, применяемую в Meep;

– большая степень абстракции для создаваемых моделей, необходимость достаточно большой подгонки исследуемой модели под комплекс;

– нет возможности гибко управлять выводом результатов, что для сложных задач приводит к генерации файлов больших объемов;

– повышенные требования к аппаратным ресурсам, в частности к оперативной памяти (речь идет о физической памяти, если начинается использование виртуальной, расчеты можно прекращать, так как они не будут выполнены). Из-за высоких требований приходится прибегать к использованию параллельной версии meep-mpi (или meep-openmpi);

– параллельная реализация программы имеет сложности в настройке и управлении выводом. Для правильной работы необходима специальная настройка параллельных библиотек, а также библиотеки для работы с HDF5 файлами.

1.3.2.3 EMAP (ElectroMagnetic Analysis Program) [49] представляет собой семейство программ на основе МКЭ с исходными кодами, которые могут быть применены для анализа задач электромагнетизма. Исходники могут быть изучены и использованы в других проектах.

EMAP разрабатывался не как конкурент существующим продуктам. Программа не имеет сложного генератора сетки, графического интерфейса и визуализации начальных и результирующих данных, либо неограниченной технической поддержки. Основная цель разработки программы была простота использования, скромные требования к ресурсам, а также точное моделирование в простой трехмерной конфигурации в широком диапазоне частот (частотное моделирование).

В EMAP1 используется вариационная формулировка, описанная Maile [50]. EMAP-2 использует МКЭ и вариационный принцип Галеркина, разработка описана в работах Paulsen and Lynch [51, 52]. Оба EMAP1 и EMAP2 основаны на скалярной формулировке МКЭ (узловые элементы) [53]. Формулировка, применяемая в EMAP2, не проявляет ложных мод [54]. EMAP3 реализует векторный МКЭ (используя векторные (реберные) элементы). Векторные конечноэлементные коды, как правило, не зависят от ложных режимов источников и имеют другие преимущества, присущие векторному методу. Все версии EMAP написаны на языке программирования C и могут быть скомпилированы на любой ОС и любом типе компьютеров.EMAP4 является улучшенной версией EMAP3. Она является более эффективной, в нее добавлены граничные поглощающие условия PML типа.

EMAP5 представляет собой гибридный FEM / MOM код [55, 56].

В качестве входных и выходных данных программы анализируют структуру, определенную в виде прямоугольников или параллелепипедов. Данные считываются из входного файла, который имеет простой вид в виде текста и определяются форматом SIF (Standard Input File) [57], после чего

производится анализ и решение задачи, в результате создается файл с выходными данными. По умолчанию формат вывода представляет собой простой список каждого из элементов сетки и соответствующих электрических величин поля.

Достоинством программы EMAP можно назвать простоту, умеренно понятный исходный код, а также кроссплатформенность (благодаря использованию языка C). Имеется также вариант для параллельного расчета с использованием технологии MPI.

Исходные коды программ частично документированы (относится только к EMAP4 и EMAP5). Описание структуры программы сводится к ее описанию в статьях научных журналов, в которых приводятся используемые формулы для математических моделей.

Недостатки:

– плохая документация, которую тяжело разобрать и каким-то образом полностью понять код программы и формулы, используемые для реализации математической модели;

– используется решение за один проход, т.е. чтобы выполнить моделирование с изменением параметров, необходимо многократно запускать программу и создавать каждый раз свой входной файл;

– начальные условия задачи необходимо полностью вводить в ручном режиме, причем нет какого либо упрощения для задания динамических параметров;

– невозможно решать задачи, использующие материалы с дисперсией. Для этого необходимо внесение изменений в программный код и соответственно вносение поправок в модель.

1.3.2.4 ADDA [58] представляет собой пакет программного обеспечения для расчета рассеяния и поглощения электромагнитных волн на частицах произвольной геометрии с использованием метода приближения дискретных диполей (DDA). Его главной особенностью является возможность работать на многопроцессорных системах или многоядерных процессорах (распараллеливание одной инстанции решения методом DDA). ADDA позиционируется как универсальный инструмент, пригодный для широкого спектра задач, начиная от моделирования межзвездной пыли и атмосферных аэрозолей до моделирования металлических наночастиц и биологических клеток. Его применение ограничено только имеющимися ресурсами компьютера. Как предполагается, ADDA должна быть использована и доступна для многих приложений без необходимости внесения изменений. Программа разработана в модульной форме, что позволяет изменять ее отдельные модули в случае необходимости.

ADDA написана на C99, имеет небольшой объем кода, поддерживает множество частиц предопределенной геометрии (эллипсоид, прямоугольные твердые тела, сферы, эритроциты и т.д.) и позволяет импортировать частицы произвольной геометрии из файла. ADDA автоматически вычисляет

экстинкции и сечения поглощения, а также полной матрицы Мюллера [59] для одной плоскости рассеяния. Частица может быть повернута по отношению к падающей волне, или результаты могут быть усреднены с учетом ориентации. Изначально ADDA возникла в Амстердамском университете, но затем превратилась в ПО с открытым исходным кодом и статусом международного проекта.

Программа позволяет производить решение с использованием параллельных вычислений. Для этого можно использовать стандарт MPI. Кроме того, среди важных преимуществ, можно отметить наличие возможности использования OpenCL, нового стандарта GPGPU вычислений, которые позволяют задействовать видеоускорители (сегодня OpenCL поддерживают также процессоры AMD APU и процессоры Intel на архитектуре Ivy Bridge) и их вычислительный ресурс, который сегодня значительно больше процессорного.

Среди недостатков можно отметить использование комбинации языков разработки, сложность задания исходных данных, почти полное отсутствие документации (описание программы более или менее присутствует в сборниках научных трудов в виде статей).

Для решения обозначенного класса задач тяжело выбрать необходимое ПО, которые бы позволяло достаточно адекватно и быстро их решать с минимальными затратами времени и ресурсов. Поэтому стоит задача проектирования и разработки нового ПО, а также соответствующей ему математической модели.

2 КОНЕЧНОЭЛЕМЕНТНОЕ МОДЕЛИРОВАНИЕ РАСПРЕДЕЛЕНИЯ ЭЛЕКТРОМАГНИТНОГО ПОЛЯ

Из рассмотренных методов [3, 9 – 41] (приложение А) для построения модели выбран векторный метод конечных элементов (ВМКЭ), который позволяет более точно и просто описывать природу электромагнитного поля, нежели “классический” узловой метод. Он позволяет описывать характер поля в виде векторов, за которые принимаются ребра элементов, а далее находить по ним поле в произвольной точке элемента. В отличие от других методов, ему не нужна очень мелкая сетка. Алгоритм решения хорошо алгоритмизируется.

Построение математической модели включает в себя:

– постановку краевой задачи для области решения в виде уравнений Максвелла и волнового уравнения;

– конечноэлементная формулировка – иначе говоря, переход к вариационной постановке;

– преобразование полученных уравнений к дискретному виду для записи в виде основного уравнения МКЭ.

Для решения задачи также необходимо:

– выполнять учет условий на границах исследуемых объектов;

– рассчитывать параметры воздействий на основе различных типов источников, испускающих электромагнитные волны.

2.1 Конечноэлементный анализ векторных полей

2.1.1 Напряженность электрического поля, создаваемого источником с плотностью зарядов J_{imp} в области Ω, характеризуемой электрической ε и магнитной μ постоянными может быть описана с помощью уравнений Максвелла (раздел 1.2). Исследуемая область может быть как двухмерной, так и трехмерной. Для определения напряженности электрического поля E необходимо решить уравнения Максвелла [39] с учетом граничных условий.

За счет исключения напряженности магнитного поля H в (1.1) и преобразования (1.2) можно получить волновое уравнение, называемое уравнением Гельмгольца:

$$\nabla \times \left(\frac{1}{\mu_r} \nabla \times E \right) - k_0^2 \varepsilon_r E = -jk_0 Z_0 J_{imp} \text{ на } \Omega, \qquad (2.1)$$

где $\mu_r = \mu / \mu_0$ и $\mu_r = \varepsilon / \varepsilon_0$ – относительные магнитная и электрическая постоянные;

$k_0 = \omega\sqrt{\mu_0 \varepsilon_0}$ и $Z_0 = \sqrt{\mu_0 / \varepsilon_0}$ – волновое число и волновое сопротивление (импеданс вакуума).

Типовые граничные условия для электрических полей включают однородные условия Дирихле на идеально проводящей поверхности, а также

смешанные на границе поверхности, обладающей волновым сопротивлением. Формулировку этих граничных условия можно записать в виде:

$$\vec{n} \times E = P, \tag{2.2}$$

$$\vec{n} \times \left(\frac{1}{\mu_r} \nabla \times E \right) + \frac{jk_0}{\eta_r} \vec{n} \times \left(\vec{n} \times E \right) = K_n, \tag{2.3}$$

где P – установленные значения для тангенциальных компонент поля на Γ_D;

η_r – нормальный импеданс поверхности на Γ_N;

K_n – известные функции, описанные на границе источника.

2.1.2 Для приближенного численного решения краевых задач используются методы Галеркина и Ритца [39]. Так называемая вариационная постановка представляет собой использование одного из них для разложения функционала задачи по бизису.

Используя метод Галеркина [41], умножим выражение (2.1) на весовую функцию W_i и проинтегрируем по области решения Ω, что даст:

$$\int_\Omega W_i \cdot \left[\nabla \times \left(\frac{1}{\mu_r} \nabla \times E \right) - k_0^2 \varepsilon_r E \right] d\Omega = -jk_0 Z_0 \int_\Omega W_i \cdot J_{imp} d\Omega. \tag{2.4}$$

Применяя свойства векторного произведения [60]

$$\nabla \cdot \left[W_i \times \left(\frac{1}{\mu_r} \nabla \times E \right) \right] = \frac{1}{\mu_r} \cdot \left(\nabla \times W_i \right) \cdot \left(\nabla \times E \right) - W_i \cdot \left[\nabla \times \left(\frac{1}{\mu_r} \nabla \times E \right) \right], \tag{2.5}$$

и векторную теорему Гаусса [60]

$$\int_\Omega \nabla \cdot \left[W_i \times \left(\frac{1}{\mu_r} \nabla \times E \right) \right] d\Omega = \oint_\Gamma \vec{n} \cdot \nabla \cdot \left[W_i \times \left(\frac{1}{\mu_r} \nabla \times E \right) \right] d\Gamma, \tag{2.6}$$

а также применяя граничные условия (2.3), получим вариационную форму волнового уравнения:

$$\begin{aligned} \int_\Omega \left[\frac{1}{\mu_r} \left(\nabla \times W_i \right) \cdot \left(\nabla \times E \right) - k_0^2 \varepsilon_r W_i \cdot E \right] d\Omega = \int_{\Gamma_D} \frac{1}{\mu_r} \left(\vec{n} \times W_i \right) \left(\nabla \times E \right) d\Gamma - \\ - \int_{\Gamma_N} \left[\frac{jk_0}{\eta_r} \left(\vec{n} \times W_i \right) \left(\nabla \times E \right) + W_i \cdot K_N \right] d\Gamma - jk_0 Z_0 \int_\Omega W_i \cdot J_{imp} d\Omega. \end{aligned} \tag{2.7}$$

2.1.3 Для нахождения численного решения уравнения (2.7) методом конечных элементов (МКЭ) внутри рассматриваемой области Ω, вначале нужно разделить область на малые части, которые называют конечными элементами (КЭ). Среди таких элементов можно отметить треугольники для двухмерной области и тетраэдры для трехмерной. На каждом КЭ поле E можно интерполировать используя дискретные значения. Один из подходов заключается в присвоении значения поля в нескольких узлах элемента и затем интерполяция на всем элементе с помощью скалярных функций. Этот

подход называется узловым МКЭ и применяется для скалярных полей, но является весьма проблематичным из-за ряда трудностей, связанных с применением требуемых граничных условий. Лучшим подходом является присвоение тангенциальной компоненты поля E для каждого ребра элемента и затем интерполяция на элементе, используя векторные базисные функции. Например, поле на треугольном элементе можно описать в следующем виде:

$$E^e(x,y) = N_{12}^e(x,y)E_{12}^e + N_{13}^e(x,y)E_{13}^e + N_{23}^e(x,y)E_{23}^e, \qquad (2.8)$$

а поле на тетраэдральном элементе может быть интерполировано как

$$E^e(x,y,z) = N_{12}^e(x,y,z)E_{12}^e + N_{13}^e(x,y,z)E_{13}^e + N_{14}^e(x,y,z)E_{14}^e + \\ + N_{23}^e(x,y,z)E_{23}^e + N_{24}^e(x,y,z)E_{24}^e + N_{34}^e(x,y,z)E_{34}^e, \qquad (2.9)$$

где E_{lk}^e обозначает тангенциальную компоненту поля E на ребре, соединяющем узлы l и k элемента e, а N_{lk}^e соответствующую интерполируемую базисную функцию. Обозначая линейную скалярную интерполяцию функции ассоциированную с узлами l и k для треугольного или третраэдрального элемента как N_l^e и N_k^e соответственно, векторная базисная функция в (2.12) и (2.13) может быть записана в виде:

$$N_{lk}^e(r) = \left[N_l^e \nabla N_k^e - N_k^e \nabla N_l^e\right] \cdot l_{lk}^e, \qquad (2.10)$$

где l_{lk}^e – длина ребра, соединяющего узлы l и k, с условием что $l < k$.

В соответствии с (2.7), базисные функции определенные как (2.10) являются векторными функциями, а соответствующие элементы называются векторными (vector) или реберными (edge) (рисунок 2.1). Очевидно, что такие базисные функции имеют тангенциальную компоненту только вдоль ассоциированного ребра, таким образом обеспечивая непрерывность тангенциального интерполированного поля, при этом позволяя нормальной составляющей быть прерывистой, что позволяет использовать их для точного решения векторного поля E.

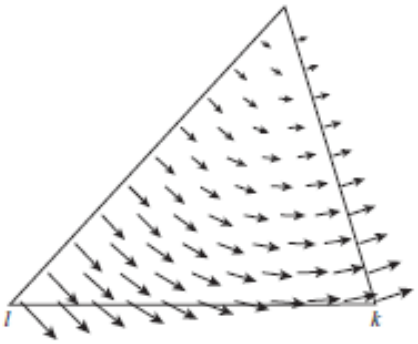


Рисунок 2.1 – Векторная базисная функция N_{lk}^e для линейного треугольного элемента

Когда электрическое поле интерполируется для каждого элемента сетки используя тангенциальные значения на ребрах элементов, поле E в области Ω может быть найдено по формуле:

$$E = \sum_{i=1}^{N_{edge}} N_i E_i + \sum_{i=1}^{N_D} N_i^D E_i^D , \qquad (2.11)$$

где N_{edge} – число уникальных ребер элементов в дискретизованной области, исключая те ребра, которые расположены на Γ_D;

E_i –тангенциальная компонента поля на *i*-ом ребре;

N_i – векторная базисная функция для соответствующего *i*-ого ребра;

N_D – количество ребер на Γ_D;

N_i^D и E_i^D – векторная базисная функция и тангенциальное значение поля соответственно.

Очевидно, что для ребер внутри области Ω, N_j насчитывается несколько соседних элементов. На рисунке 2.2 показана векторная базисная функция для внутреннего ребра в треугольной сетке. Также отметим, что со вторым слагаемым уравнения (2.11), интерполированное поле удовлетворяет необходимым условиям на границе в соответствии с условием (2.2).

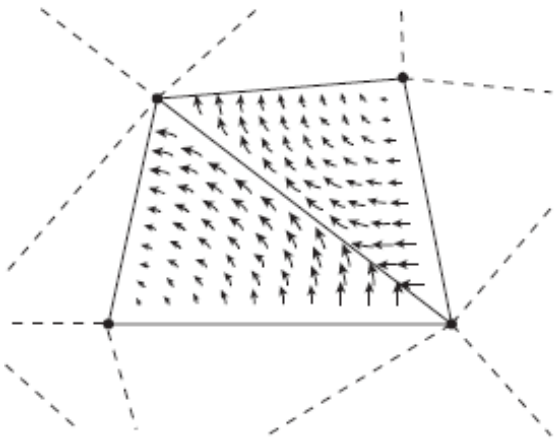


Рисунок 2.2 – Векторная базисная функция N_i для линейных треугольных элементов

Подставив (2.11) в (2.7), где в качестве W_i использована векторная базисная функция N_i для ребер элементов, получим:

$$\sum_{j=0}^{N_{edge}} K_{ij} E_j = b_i , \; i = 1,2,\ldots, N_{edge} , \qquad (2.12)$$

где

$$K_{ij} = \int_{\Omega}\left[\frac{1}{\mu_r}(\nabla\times N_i)\cdot(\nabla\times N_j) - k_0^2\varepsilon_r\cdot N_i\cdot N_j\right]d\Omega + \\ + jk_0\int_{\Gamma_N}\frac{1}{\eta_r}(\vec{n}\times N_i)(\nabla\times N_j)d\Gamma, \tag{2.13}$$

$$b_i = -jk_0Z_0\int_{\Omega} N_i\cdot J_{imp}d\Omega - \int_{\Gamma_N} N_i\cdot K_N d\Gamma - \\ - \sum E_j^D\int_{\Omega}\left[\frac{1}{\mu_r}(\nabla\times N_i)\cdot(\nabla\times N_j^D) - k_0^2\varepsilon_r\cdot N_i\cdot N_j^D\right]d\Omega\,. \tag{2.14}$$

Заметим, что интеграл по Γ_D в уравнении (2.7) исчезает из-за условия $\hat{n}\times N_i = 0$ на Γ_D.

В матричном виде уравнение (2.12) можно записать как

$$[K]\{E\} = \{b\}, \tag{2.15}$$

решая которое может быть получен вектор $\{E\}$. В нем будет находиться сведения о локальных компонентах поля, собранных на ребрах элементов.

2.1.4 Связь локального и глобального представлений поля.

В представлении в виде (2.11) вектор поля задается 6-ю скалярными компонентами, каждая из которых определяет вклад одной из базисных функций N_i. Найденные коэффициенты вектора неизвестных равны норме вектора, направленного вдоль одного из ребер элемента, образующего дискретизацию расчетной области. Однако для интерпретации и графического отображения полученных результатов необходимо представление поля не в виде (2.11), а в обычном виде

$$E = E_x i + E_y j + E_z k\,, \tag{2.16}$$

который называют глобальным представлением поля.

Возникает задача нахождения зависимости, позволяющей осуществить переход от $(E_1, E_2, E_3, E_4, E_5, E_6)^T$ к $(E_x, E_y, E_z)^T$. Для этого необходимо воспользоваться формулой:

$$E = \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = W_{12}l_{12}E_{12} + W_{13}l_{13}E_{13} + W_{14}l_{14}E_{14} + W_{23}l_{23}E_{23} + \\ + W_{42}l_{42}E_{42} + W_{34}l_{34}E_{34} = \\ = \begin{bmatrix} W_{12} & W_{13} & W_{14} & W_{23} & W_{42} & W_{34} \end{bmatrix}\times \\ \times\begin{bmatrix} l_{12}E_{12} & l_{13}E_{13} & l_{14}E_{14} & l_{23}E_{23} & l_{42}E_{42} & l_{34}E_{34} \end{bmatrix}^T. \tag{2.17}$$

В формуле (2.17) приведенные обозначения соответствуют векторному конечному элементу тетраэдр, имеющему 4 узла в вершинах и 6 ребер (раздел 2.1.1).

Для решения (2.17) необходимо выразить векторные базисные функции W_{ij} в декартовой системе (x,y,z). В случае трехмерных L-координат это довольно сложно сделать вручную. В приложении Б представлена программа для аналитического вычисления объемных L-координат для тетраэдрального элемента.

2.1.5 Во многих задачах можно предположить, что временная зависимость E и B от источников ведет себя таким образом, что ее можно представить в виде конечной суммы из N временных спектральных компонент (Фурье-компонент) [61], или, другими словами, в виде временной серии Фурье. В таком случае достаточно изучить свойства одного произвольного члена из набора спектральных компонент $\{\omega_n : n = 0,1,2,...N-1\}$, т.е.

$$E(x,t) = E_n \cos(\omega_n t) = E_n(x)\,\mathrm{Re}\{e^{-i\omega_n t}\} \equiv E_n(x)e^{-i\omega_n t}, \qquad (2.18)$$

$$B(x,t) = B_n \cos(\omega_n t) = B_n(x)\,\mathrm{Re}\{e^{-i\omega_n t}\} \equiv B_n(x)e^{-i\omega_n t}. \qquad (2.19)$$

Такая запись возможна из того факта, что уравнения Максвелла-Лоренца является линейным, подразумевая что общее решение может быть получено с помощью линейной суперпозиции (суммирования) результата для каждой спектральной компоненты, где вес каждого спектрального получается из амплитуд Фурье, $E_n(x)$ и $B_n(x)$ соответственно.

В физической системе, временная спектральная компонента точно определяется по круговой частоте ω_n. Волна, содержащая только конечное число таких компонент, называется гармонически изменяемой во времени (time-harmonic wave). Когда число компонент ограничено одной, говорят о монохроматической волне, которой в реальности не существует.

Если подставить (2.16) в выражение для волнового уравнения (2.1) [61] и решать его, то полученный результат необходимо умножить на $e^{-i\omega_n t}$ и просуммировать для всех Фурье (спектральных) компонент с частотами ω_n, на которых работает источник.

2.2 Узловые конечные элементы

После дискретизации области выбранным типом элементов, свойства области присваиваются узлам элементов. В качестве КЭ для трехмерного анализа используется тетраэдр.

Неизвестную функцию на элементе можно представить как:

$$\phi^e(\mathrm{x},\mathrm{y},\mathrm{z}) = a^e + b^e x + c^e y + d^e z. \qquad (2.20)$$

Коэффициенты a^e, b^e, c^e, d^e могут быть определены путем определения значения искомой функции (2.20) в узлах:

$$\phi_1^e = a^e + b^e x_1^e + c^e y_1^e + d^e z_1^e,$$

$$\phi_2^e = a^e + b^e x_2^e + c^e y_2^e + d^e z_2^e,$$
$$\phi_3^e = a^e + b^e x_3^e + c^e y_3^e + d^e z_3^e,$$
$$\phi_4^e = a^e + b^e x_4^e + c^e y_4^e + d^e z_4^e,$$

откуда получаются следующие значения для коэффициентов:

$$a^e = \frac{1}{6V^e}\begin{vmatrix} \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix} = \frac{1}{6V^e}(a_1^e\phi_1^e + a_2^e\phi_2^e + a_3^e\phi_3^e + a_4^e\phi_4^e),$$

$$b^e = \frac{1}{6V^e}\begin{vmatrix} 1 & 1 & 1 & 1 \\ \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix} = \frac{1}{6V^e}(b_1^e\phi_1^e + b_2^e\phi_2^e + b_3^e\phi_3^e + b_4^e\phi_4^e),$$

$$c^e = \frac{1}{6V^e}\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix} = \frac{1}{6V^e}(c_1^e\phi_1^e + c_2^e\phi_2^e + c_3^e\phi_3^e + c_4^e\phi_4^e),$$

$$d^e = \frac{1}{6V^e}\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \end{vmatrix} = \frac{1}{6V^e}(d_1^e\phi_1^e + d_2^e\phi_2^e + d_3^e\phi_3^e + d_4^e\phi_4^e),$$

где

$$V^e = \frac{1}{6}\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix}$$ - объем тетраэдра. (2.21)

Подставив выражения для a^e, b^e, c^e, d^e в (2.20), получается, что исходную функцию можно представить в виде:

$$\phi^e(x, y, z) = \sum_{j=1}^{4} N_j^e(x, y, z)\varphi_j^e, \quad (2.22)$$

где интерполяционная функция $N_j^e(x, y, z)$ равна:

$$N_j^e(x, y, z) = \frac{1}{6V^e}(a + b_j^e x + c_j^e y + d_j^e z). \quad (2.23)$$

Эта функция обладает следующим свойством:

$$N_j^e(x,y,z) = \delta_{ij} = \begin{cases} 1, i = j, \\ 0, i \neq j. \end{cases} \quad (2.24)$$

Которое означает то, что $N_j^e(x,y,z)$ пропадает, когда точка на поверхности тетраэдра расположена в противоположной вершине относительно *j*-ой. В результате гарантируется непрерывность интерполяционной функции.

2.3 Векторные конечные элементы

2.3.1 Тетраэдр.

Среди наиболее используемых элементов для анализа трехмерных объектов можно выделить тетраэдр (рисунок 2.3) [39]. Простой тетраэдр описывается с помощью 4 узлов и 6 ребер, соединяющих узлы.

Как и треугольный элемент, тетраэдр удобно записывать через так называемые *L*-координаты (приложение Б), которые ассоциируются с 4-мя узлами элемента. Тогда векторную функцию для ребра (1,2) можно записать:

$$W_{12} = L_1^e \nabla L_2^e - L_2^e \nabla L_1^e . \quad (2.25)$$

Во-первых, просто показать что

$$\nabla \cdot W_{12} = 0, \qquad \nabla \times W_{12} = 2\nabla L_1^e \times \nabla L_2^e . \quad (2.26)$$

Во-вторых, e_1 является единичным вектором из узла 1 в узел 2. Так как L_1^e является линейной функцией, которая варьируется в значениях от 1 в узле 1 до нуля в узле 2 и L_2^e – аналогично, от 1 в узле 2 до 0 в узле 1, $e_1 \cdot \nabla L_1^e = -1/l_1^e$ и $e_1 \cdot \nabla L_2^e = -1/l_1^e$, где l_1^e – длина ребра, соединяющего узлы 1 и 2.

Поэтому получается выражение

$$e_1 \cdot W_{12} = \left(L_1^e + L_2^e\right)/l_1^e = 1/l_1^e , \quad (2.27)$$

которое означает, что W_{12} имеет постоянную тангенциальную компоненту вдоль ребра (1,2). Далее, так как L_1^e исчезает вдоль ребер (2,3), (2,4) и (3,4) и L_2^e – вдоль (1,3), (1,4) и (3,4), W_{12} не имеет тангенциальных компонент вдоль этих пяти ребер. Кроме того, поскольку L_1^e исчезает на плоскости элемента (2,3,4), а L_2^e на плоскости (1,3,4), W_{12} не имеет тангенциальных компонент на любой из этих поверхностей. Эти тангенциальные компоненты появятся только на поверхностях, которые содержат ребро (1,2): (1,2,3) и (1,2,4). Таким образом, W_{12} обладает всеми необходимыми свойствами присущими векторной базисной функции для описания поля на ребре (1,2). Если определить его с номером 1, то получим:

$$N_1^e = W_{12} l_1^e = \left(L_1^e \nabla L_2^e - L_2^e \nabla L_1^e\right) l_1^e . \quad (2.28)$$

Аналогично получаются векторные базисные функции для остальных ребер тетраэдрального элемента:

$$N_i^e = W_{i1.i2} l_i^e = \left(L_{i1}^e \nabla L_{i2}^e - L_{i2}^e \nabla L_{i1}^e \right) l_i^e . \qquad (2.29)$$

Вид векторного элемента в виде тетраэдра представлен на рисунке 2.3, а нумерация его узлов и ребер – в таблице 2.1.

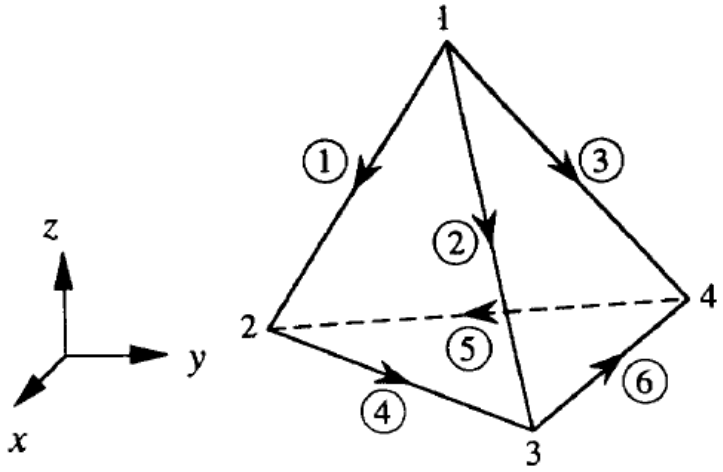


Рисунок 2.3 – Тетраэдр с 4-мя узлами и 6-ю ребрами

Таблица 2.1 – Нумерация узлов и ребер тетраэдра

Номер ребра i	Номер узла i_1	Номер узла i_2
1	1	2
2	1	3
3	1	4
4	2	3
5	4	2
6	3	4

2.3.2 Вычисление локальных матриц элемента.

Результирующие матрицы для элементов имеют вид:

$$E_{ij}^e = \iiint_V (\nabla \times N_i^e) \cdot (\nabla \times N_j^e) dV , \qquad (2.30)$$

$$F_{ij}^e = \iiint_V N_i^e \cdot N_j^e dV . \qquad (2.31)$$

Эти интегралы могут быть аналитически определены для тетраэдра и треугольника.

$$\nabla \times N_i^e = 2l_i^e \nabla L_{i1}^e \times \nabla L_{i2}^e =$$

$$= \frac{l_i^e}{(6V^e)^2}\begin{bmatrix} \left(c_{i1}^e d_{i2}^e - d_{i1}^e c_{i2}^e\right)\left(c_{j1}^e d_{j2}^e - d_{j1}^e c_{j2}^e\right) + \\ + \left(d_{i1}^e b_{i2}^e - b_{i1}^e d_{i2}^e\right)\left(d_{j1}^e b_{j2}^e - b_{j1}^e d_{j2}^e\right) + \\ + \left(b_{i1}^e c_{i2}^e - c_{i1}^e b_{i2}^e\right)\left(b_{j1}^e c_{j2}^e - c_{j1}^e b_{j2}^e\right) \end{bmatrix}. \quad (2.32)$$

Теперь можно записать для элементов из (2.25):

$$E_{ij}^e == \frac{l_i^e l_j^e V^e}{(6V^e)^4}\begin{bmatrix} \left(c_{i1}^e d_{i2}^e - d_{i1}^e c_{i2}^e\right)\left(c_{j1}^e d_{j2}^e - d_{j1}^e c_{j2}^e\right) + \\ + \left(d_{i1}^e b_{i2}^e - b_{i1}^e d_{i2}^e\right)\left(d_{j1}^e b_{j2}^e - b_{j1}^e d_{j2}^e\right) + \\ + \left(b_{i1}^e c_{i2}^e - c_{i1}^e b_{i2}^e\right)\left(b_{j1}^e c_{j2}^e - c_{j1}^e b_{j2}^e\right) \end{bmatrix}. \quad (2.33)$$

Для определения (2.26):

$$N_i^e \cdot N_j^e = \frac{l_i^e l_j^e}{(6V^e)^2}\begin{bmatrix} L_{i1}^e L_{j1}^e f_{i2.j2} - L_{i1}^e L_{j2}^e f_{i2.j1} - \\ - L_{i2}^e L_{j1}^e f_{i1.j2} + L_{i2}^e L_{j2}^e f_{i1.j1} \end{bmatrix}, \quad (2.34)$$

где $f_{ij} = b_i^e b_j^e + c_i^e c_j^e + d_i^e d_j^e$. Подстановка выражения (2.34) в (2.31) дает следующий результат:

$$F_{11}^e = \frac{(l_1)^2}{360 \cdot V^e}\left(f_{22} - 2 f_{12+} f_{11}\right),$$

и так далее (остальные формулы приведены в [39]).

После вычисления аналитических матриц, можно составить полную матрицу для единичного элемента (локальную матрицу жесткости):

$$[K]^e = \frac{1}{\mu_r}[E]^e - k^2 \varepsilon_r [F]^e. \quad (2.35)$$

На основе рассмотренной математической модели и МКЭ можно сформулировать требования к разработке программного комплекса:

– наличие универсального алгоритма для построения сеток, т. е. алгоритма дискретизации области на КЭ;

– возможность задания произвольной геометрии и материалов для всех объектов задачи;

– возможность задания различных параметров источников электромагнитного излучения (источников воздействующих волн);

– реализация построения матриц, в соответствии с МКЭ;

– реализация метода для решения СЛАУ, получаемого в результате реализации математической модели по МКЭ;

– возможность как единичного решения, так и многократного решения задачи с пересчетом параметров;

– интерпретация результатов и предоставление их в удобном для дальнейшей обработки виде.

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПО ДЛЯ МОДЕЛИРОВАНИЯ РАПРЕДЕЛЕНИЯ ЭЛЕКТРОМАГНИТНОГО ПОЛЯ

Для компьютерного моделирования распределения электромагнитных полей (ЭМП) в нанокомпозитах необходимо сформулировать требования, предъявляемые к программному продукту:

– возможность точного описания модели системы, наиболее близко описывающей ее реальную структуру;

– адекватность результатов расчетов по модели с реальной системой;

– универсальный язык описания моделей (метаязык);

– возможность поддержки программного кода, а также внесение изменений в используемые модели;

– экономическая эффективность, т. е. стоимость разработки и использования вычислительных ресурсов должны быть оправданы (целесообразны).

Процесс решения задачи, построенной на основе математической модели из раздела 2, можно описать в виде схемы на рисунке 3.1.

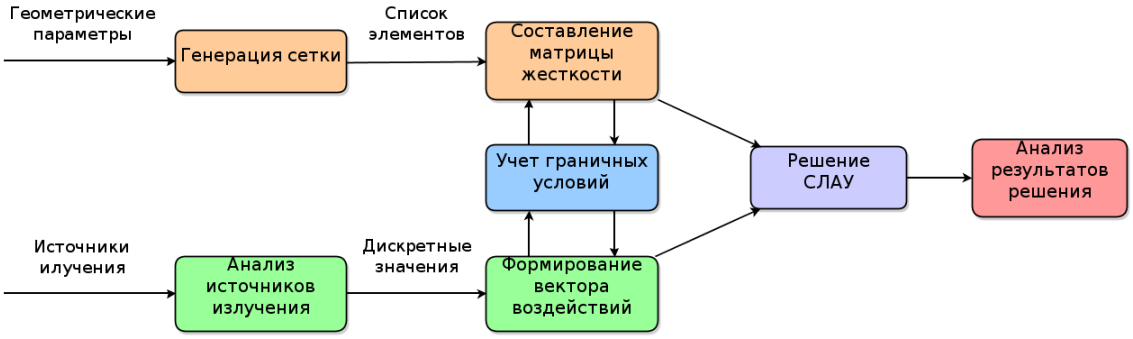


Рисунок 3.1 – Процесс выполнения функций

Для реализации каждой функции системы необходимо выполнить ряд этапов, которые идут по порядку. Такой процесс разработки можно назвать итерационным [62], когда на каждой итерации производится выполнение определенной поставленной задачи.

3.1 Выбор средств для разработки программного обеспечения

На основании предъявляемых требований и функций (рисунок 3.1) выбираются средства, с помощью которых будет производиться разработка ПО для компьютерного моделирования распределения ЭМП. Среди выбранных средств можно выделить:

– язык C# 4 и платформа .NET [63];

– программа Netgen;

– формат XML для описания входных данных (условий и параметров решаемой задачи);

– диаграммы UML [64] для описания работы программы.

3.1.1 Для компьютерного моделирования распределения ЭМП была выбрана платформа .NET, на которой будет построено разрабатываемое ПО. В качестве языка программирования выбран язык C# 4 [63]. Такой выбор обусловлен тем, что C# позволяет полноценно реализовывать принципы ООП, обладает простым и понятным синтаксисом в стиле C, имеет новые возможности, обладает технологиями, облегчающих разработку. На языке возможна реализация основных паттернов проектирования (GoF) [65], используемых в разработке. Для платформы разработано множество дополнений и библиотек, позволяющих расширить возможности самой платформы .NET. Наличие большого сообщества разработчиков позволяет находить ответы на возникающие при разработке проблемы и вопросы.

Скорость работы программ достаточно высокая, требования к аппаратной и программной части умеренные, что позволяет использовать разрабатываемое ПО на различных конфигурациях (в приложении К приведено руководство системного программиста). При необходимости в масштабировании ПО, можно использовать параллельные библиотеки.

Особенно стоит отметить возможность разработки программы в виде библиотеки, которую затем можно использовать с любым интерфейсом (UI), которые предлагает платформа .NET (WPF, Silverlight, ASP.NET WebForms, ASP. NET MVC, ASP.NET WebPages). Таким образом, возможно обеспечение универсальности, получаемой при создании одного ядра приложения и использующих его интерфейсов. При необходимости переноса приложения в облачную инфраструктуру или в Интернет, можно использовать веб-интерфейс и платформу Windows Azure.

3.1.2 Для решения задачи о генерации сетки было принято решение использовать стороннюю программу, которая хорошо решает задачу и проста в обращении, умеет работать с двух- и трехмерной геометрией. Таким выбором стала программа *Netgen*[66], позволяющая генерировать конечноэлементные сетки для двух- и трехмерных задач. Программа распространяется по лицензии GPL [67]. Это позволяет использовать не только саму программу, но и исходные коды. Программа генерирует сетки из треугольников и тетраэдров, а также поддерживает экспорт в различные форматы для других программ КЭ анализа. Описание работы с программой и примеры использования приведены в приложении Г.

3.1.3 При разработке программы были использованы сторонние библиотеки, предоставляющие возможность логирования работы программы, а также мощную математическую библиотеку, необходимую для реализации некоторых алгоритмов линейной алгебры.

В качестве библиотеки для работы с логами был выбран пакет *NLog 2* [68], который свободно распространяется и имеет расширенные возможности по созданию и ведению логов. С помощью него можно организовать ведение логов в файл, в БД, отправку логов по почте и прочее. Среди преимуществ можно отметить простоту использования и возможность гибко управлять правилами вывода лога. Распространяется по лицензии BSD [69].

В качестве библиотеки для математических операций и алгоритмов была выбрана библиотека *Math.NET Numerics [70]*, которая содержит большое число качественных математических алгоритмов. *Numerics* стал результатом объединения *dnAnalytics* с *Math.NET Iridium* и предназначен для замены обоих. Изначально, разработан под .NET 4.0, существует реализация для Mono, а также поддерживает некоторую аппаратную оптимизацию. Открытый исходный код позволяет вносить изменения, создавать и дополнять свои более эффективные алгоритмы. Проект постоянно развивается. Распространяется по лицензии MIT [71].

3.1.4 При выполнении разработки проекта применяли следующие программы:

– *Microsoft Visual Studio 2012 Ultimate* – интегрированная среда разработки (IDE), предоставляющая современные средства и инструменты для разработки высокопроизводительных, качественных программ на различных языках и платформах (VC C++, C#, HTML, .NET, ASP.NET, IronRuby, IronPython) [63], средства для моделирования архитектуры ПО, проведения тестирования разрабатываемых продуктов и т.д. Используется в качестве основного инструмента для разработки программы на языке C#.

– *Sublime Text 2* – лучший текстовый редактор на сегодняшний день, который предоставляет обширнейшие возможности по редактированию файлов различных форматов, поддержку сборки и компиляции проектов, подсветку синтаксиса, поддержку плагинов и многое другое. Редактор написан на Python, имеет гибкую систему настроек, позволяющую подстроить редактор под себя. Расширения придают редактору множество возможностей, которые предоставляют далеко не все IDE.

– *Diagramly* – расширение для браузера Google Chrome, позволяющее рисовать различные типы диаграмм (блок-схемы, UML диаграммы, организационную структуру, Workflow т.д.), и поддерживает работу с Google Drive (облачным хранилищем). Расширения применялось для рисования схем, описывающих разработанную программу.

– *Microsoft Word 2013* – многофункциональный текстовый редактор, который позволяет создавать документы различной сложности, имея расширенные возможности по редактированию. Является лучшим редактором на платформе Windows и Mac OS X.

– *Matlab R2012b* – мощнейший математический пакет, позволяющий решать сложные математические уравнения, системы и проблемы. Содержит

множество дополнений и имеет развитую систему документации. Использовался для сложных аналитических вычислений.

3.2 Структура разработанного ПО

В соответствии с функциями ПО, была разработана структура блоков программы [62], которая использована при разработке (рисунок 3.2).

Программа разработана с использованием некоторых правил именования классов, модулей пространств имен. Таким образом структурные части программы разделены между собой и имеют свою область (пространство имен). Это исключает конфликты именования, возникающие в процессе разработки и при разработке классов с идентичными именами.

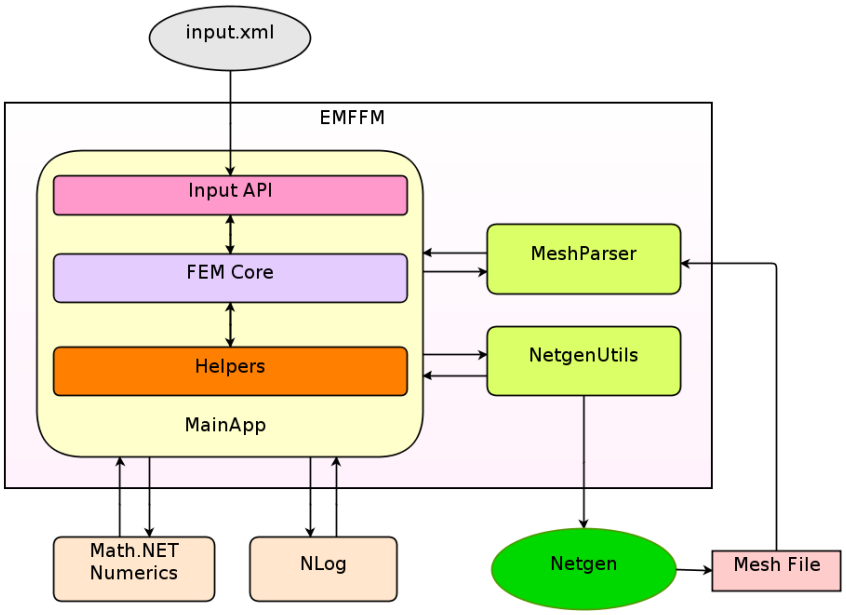


Рисунок 3.2 – Структура модулей программы

3.2.1 Для реализации работы с программой *Netgen*, используемой для построения конечноэлементной сетки были разработаны специальные интерфейсы в виде двух модулей:

– *NetgenUtils* – библиотека, содержащая средства для работы с программой *Netgen*, в частности для препроцессинга исходной задачи (генерации конечноэлементной сетки);

– *MeshParser* – библиотека для обработки файлов сетки, генерируемых программой *Netgen*.

Описание модуля *NetgenUtils* приведено в приложении Д, а модуля *MeshParser* – в приложении Д.

3.2.2 При разработке программы в соответствии с требованием к простоте задания начальных данных и параметров, необходимых для решения задачи и ее описания, спроектирован специальный стандарт для

входных файлов, который получил название Input API. Этот стандартный интерфейс предназначен для работы с входными файлами в формате XML и описания в них условий задачи для передачи ее программе на выполнение. Разработка Input API и структуры входных файлов представляет собой целую задачу, целью которой является создание унифицированного формата для описания решаемых задач, позволяющего ввести общеупотребимые форматы файлов, которые могут быть использованы в других программах. Его еще можно назвать – метаязык.

Разработанный в виде Input API, формат применяется в программе в качестве входного по умолчанию. Статус проекта – в разработке. Описание и спецификации стандарта приведены в приложении Е.

Главная особенность формата заключается в том, что он подразумевает полное описание задачи, включая сценарий ее выполнения. Это позволяет полностью избавиться от необходимости ввода каких-либо дополнительных данных в процессе выполнения, а также исключить визуальный интерфейс.

3.2.3 Для реализации математической модели, использующей МКЭ, была разработана часть системы, получившая общее название FEM Core (рисунок 3.3). Эта часть программы является главным блоком, который производит решение задачи и анализ получаемых результатов.

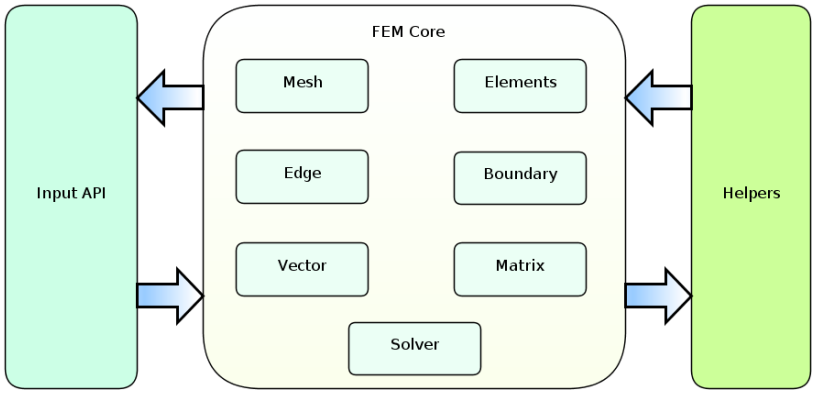


Рисунок 3.3 – Схема модуля FEM Core

3.2.4 Разработанное приложение построено с использованием принципов ООП и паттернов проектирования. Программа разрабатывалась таким образом, чтобы можно было внести изменения в отдельные ее части, перепроектировать их, добавить функциональные возможности.

Таким образом, функционально программу можно описать в виде диаграммы прецедентов UML (Use Case) [64] на рисунке 3.4.

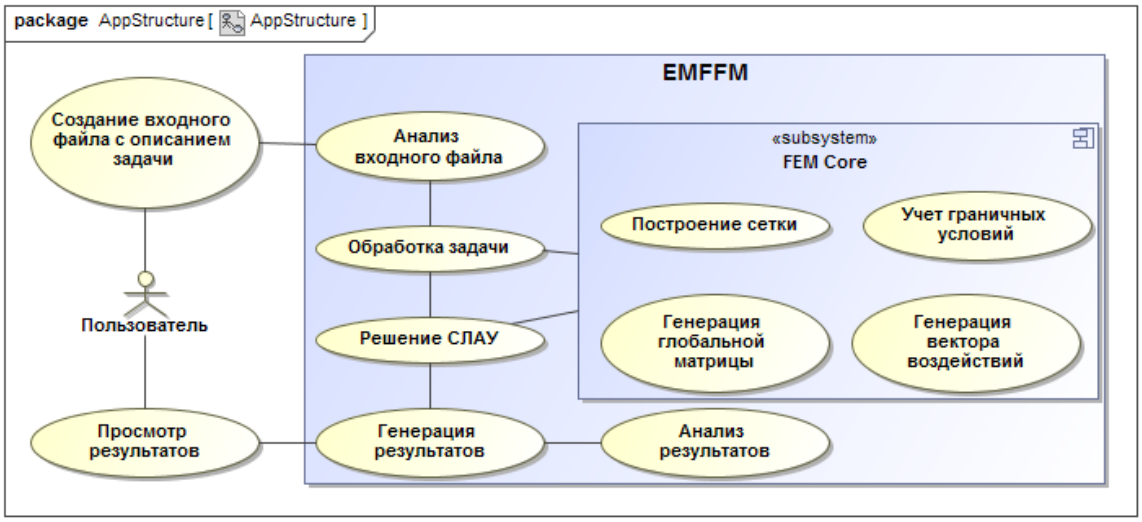


Рисунок 3.4 – Диаграмма прецедентов (Use Case)

Схему работы программы, использующую Input API, можно представить на рисунке 3.5 в виде диаграммы активности UML (Activity Diagram) [64].

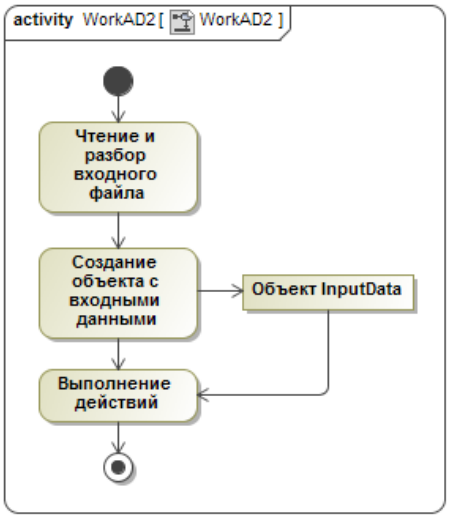


Рисунок 3.5 – Схема работы программы с Input API

На рисунке 3.5 представлено мало действий. Это связано с тем, что идея самого Input API заключается в создании простого средства для решения задачи, когда предполагается минимальное участие пользователя. Вся работа программы сводится к выполнению заданных в сценарии (входном файле) действий. Они выполняются по порядку, при этом каждое действие может вызывать и задействовать различные блоки программы.

3.3 Реализация блоков программы

3.3.1 Встроенные алгоритмы построения сетки.

В FEM Core предусмотрены алгоритмы для генерации конечноэлементной сетки в рассматриваемой области. Для генерации сетки используются специальные классы, каждый из которых производит генерацию сетки из того или иного типа элементов. Для каждого типа элементов предусмотрен свой класс генератора сеток. Генерируются регулярные сетки, основанные на прямоугольниках для 2D случая и параллелепипедах для 3D. В 2D случае сначала генерируются прямоугольные элементы, а затем треугольные, в 3D – параллелепипеды, потом они делятся на 5 тетраэдров. Классы для генерации сетки реализуют интерфейс IMesh.

Построение сетки основано на использовании объектов и работе с ними. Суть построения сетки – разбиение области на конечные элементы. На вход поступают исходные данные о геометрии задачи, на выходе получается список элементов. Каждый элемент характеризуется своими узлами, а узлы в свою очередь – координатами и номером. Узел определяется двумя номерами: локальным и глобальным. При генерации элементов, узлы получают глобальную нумерацию. Локальная нумерация может быть легко получена из списка узлов элемента, где они расположены по порядку.

В программе реализованы 4 типа генераторов сетки для 4-х типов элементов:

- прямоугольные элементы (ортогональные) – Rectangle – RectMesh;
- треугольные элементы (ортогональные) – Triangle – TriMesh;
- параллелепипеды (ортогональные) – Parallelepiped – ParMesh;
- тетраэдры – Tetrahedron – TetMesh.

3.3.2 Для работы с элементами сеток реализована иерархия классов для описания параметров элементов (рисунок 3.6). Все элементы наследуются от базового класса Element, который содержит общее описание параметров элемента. К ним относятся:

- список узлов элемента;
- список ребер элемента;
- номер элемента;
- материал;
- код области;
- код граничного условия.

Кроме того конкретные типы элементов реализуют следующие интерфейсы:

– IElement – интерфейс, описывающий методы для вычисления локальных матриц;

– I2DElement – интерфейс, описывающий методы, общие для двухмерных элементов (нахождение площади и периметра);

– I3DElement – интерфейс, описывающий методы, общие для трехмерных элементов (вычисление объема).

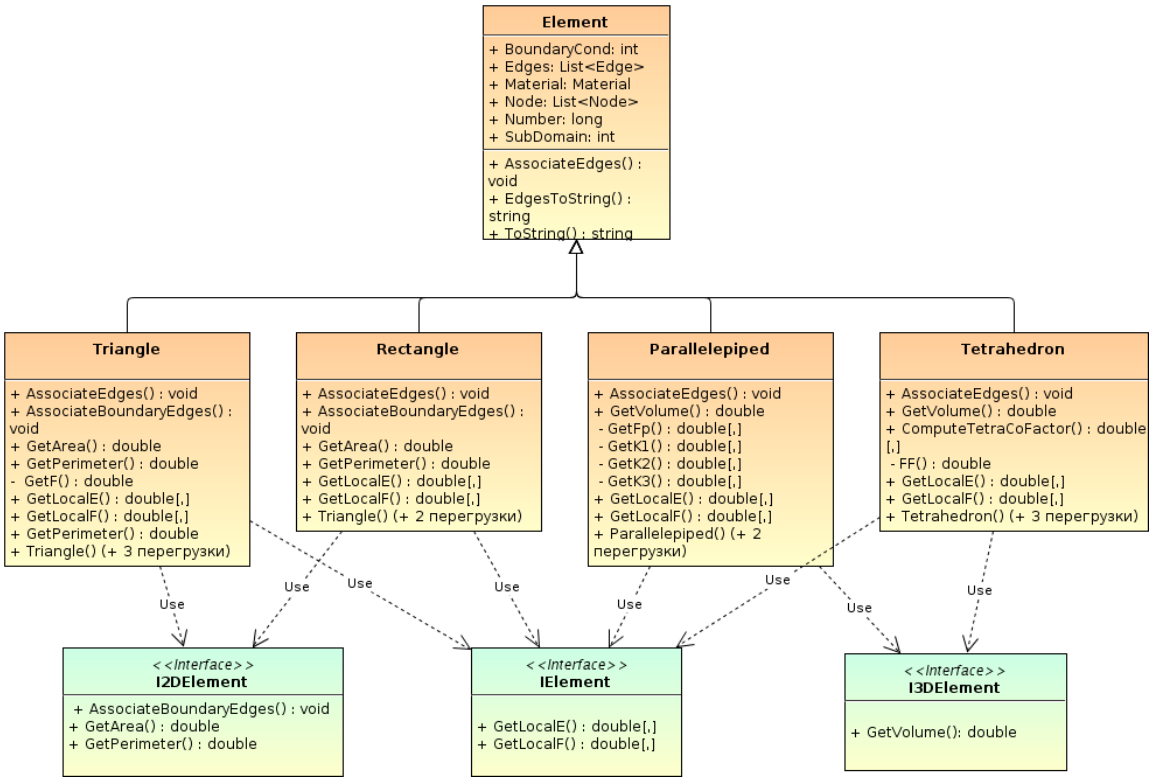


Рисунок 3.6 – Схема классов для описания элементов

3.3.3 Для обеспечения работы основных частей программы необходимы дополнительные, вспомогательные модули, которые позволяют отдельно разместить некоторые функции, логически и тематически не связанные с основными частями программы. Для таких функций применена концепция так называемых Helper классов, которые являются статическими с часто используемыми простыми функциями.

Среди вспомогательных классов в программе можно отметить следующие:

– OutputHelper – класс для реализации вывода в файл результатов работы программы (генерирует текстовые и XML файлы);

– StringHelper – класс для реализация вспомогательных функций по обработке строк;

– IOHelper – класс для работы с файлами (очистка папок, удаление файлов);

– ActionHelper – класс для описания ассоциаций между действиями из Input API и функциями с программе;

– NetgenMeshHelper – вспомогательный класс для работы с генерацией сетки с помощью программы Netgen;

– MeshHelper – вспомогательный класс для работы с сеткой;

– MathHelper – вспомогательные математические функции;

– MatrixHelper – вспомогательный класс для работы с матрицами.

3.3.4 Разработанное ПО взаимодействует с программой *Netgen* [66]. Работа с ней построена на основе двух модулей: *NetgenUtils* и *MeshParser*. Это позволяет задавать довольно сложную геометрию и строить нерегулярную сетку из тетраэдров. С помощью модулей производится запуск программы, передача в нее файла с геометрией, а затем разбор полученного файла с данными о построенной сетке.

Результатом выполнения действий, показанных на рисунке 3.7, является объект класса TMesh<TType, TType2>, который содержит сведения о построенной сетке:

– список объемных элементов;
– список поверхностных элементов;
– список узлов;
– количество ребер.

Данный объект содержит результаты препроцессинга задачи, которая выполняет генерацию конечноэлементной сетки – список элементов сетки.

Дальнейшая работа будет осуществляться в приложении с использованием созданного объекта обобщенного класса TMesh<>, содержащем сведения о построенной сетке для заданной геометрии.

Взаимодействие с программой Netgen осуществляется через вспомогательный класс NetgenMeshHelper. Его функции описаны в таблице 3.1.

Таблица 3.1 – Методы класса NetgenMeshHelper

Название	Аргументы	Описание
GenerateNetgenMesh	InputData data, string fileName	Генерация сетки с использованием программы Netgen
GenerateVolumeElementsFromMeshData	MeshData data, bool isBuiltEdges	Сбор сведений об объемных элементах
GenerateSurfaceElementsFromMeshData	MeshData data, bool isBuiltEdges	Сбор сведений о поверхностных элементах
GenerateBoundaryEdges	TMesh<Tetrahedron, Triangle> mesh	Генерация граничных ребер

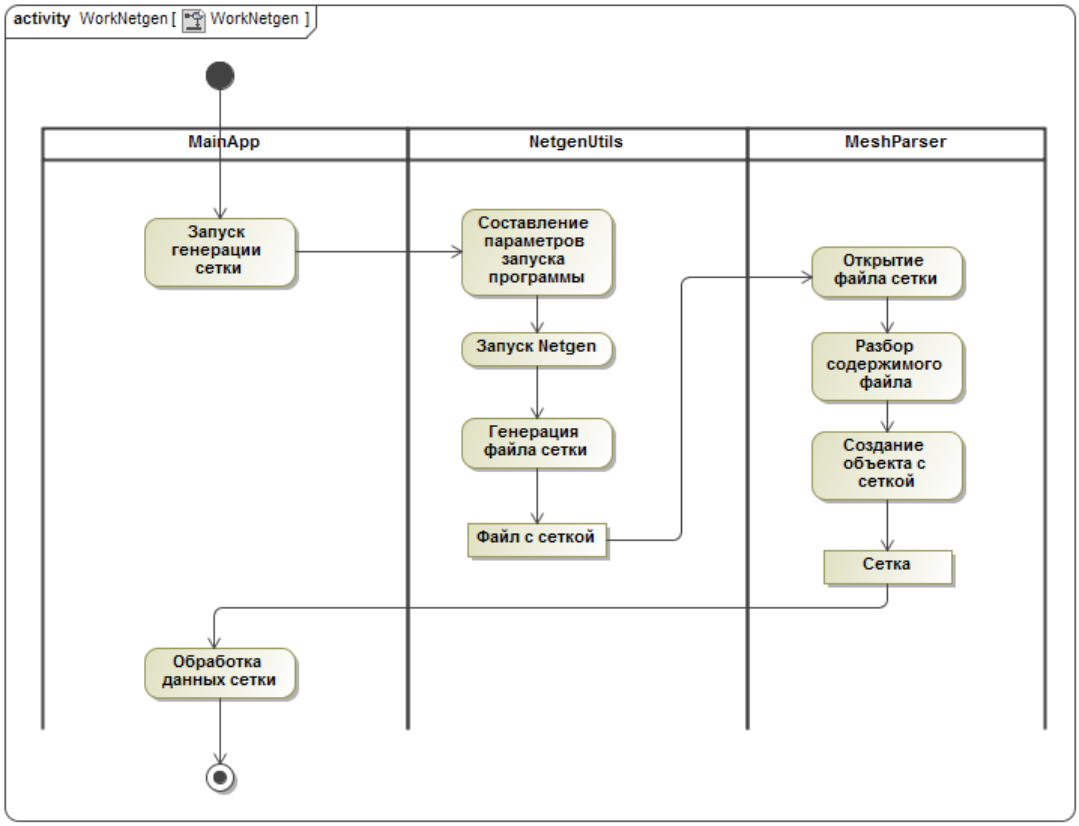


Рисунок 3.7 – Схема работы с программой Netgen

3.3.5 После построения сетки необходимо выполнить обработку списка элементов, которая связанна с особенностью используемого МКЭ. Классический метод подразумевает обработку элементов по узлам. В ВМКЭ используется, так называемая, реберная сетка, в которой основное значение играют ребра – стороны элементов. Ребра представляют собой элементы, которые состоят из:

– двух узлов (вершин элементов);
– номера ребра (точнее, двух: локального и глобального).

Проблема в использовании ребер заключается в том, что их глобальная нумерация должна быть уникальна и автоматически она не может быть получена путем обхода всех элементов и нумерации ребер подряд. Для этого нужен специальный алгоритм, который произведет анализ ребер у всех элементов, а затем, путем их обработки, исключит повторяющиеся, после чего их нумерация будет уникальной. Ребра будут иметь свой уникальный глобальный номер.

Для этого в программе используется класс EdgeMesh, который производит обработку ребер. Схема процесса представлена на рисунке 3.8. Суть алгоритма заключается в нумерации уникального набора ребер, а затем установление соответствия его с ребрами элементов.

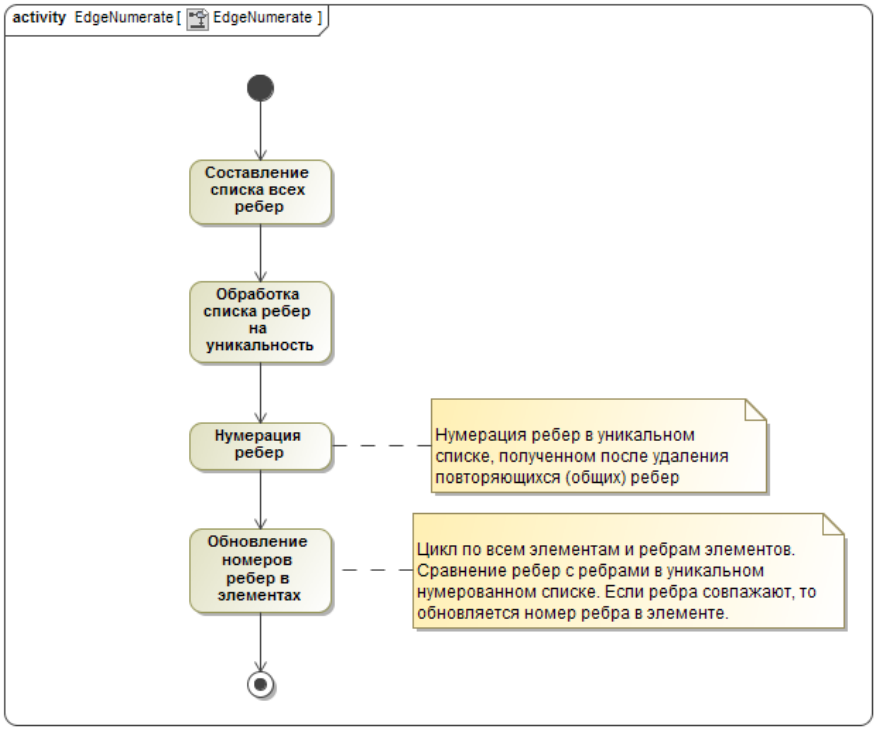


Рисунок 3.8 – Схема процесса нумерации ребер

В результате работы алгоритма получается исходный список элементов, но только теперь, у каждого элемента ребра имеют уникальную нумерацию.

Для работы с граничными условиями, необходимо определить ребра, которые будут граничными. При работе с программой Netgen, эта операция выполняется с помощью класса NetgenMeshHelper, сразу же при генерации основной сетки (метод GenerateBoundaryEdges) и обработке ребер для границ. В результате этого ребра получают дополнительные свойства: код границы и параметр, определяющий является ли ребро граничным.

Определение граничных ребер основано на том, что ребра у поверхностных элементов имеют одни и те же узлы, что и у объемных. Следовательно, выполнив сравнение всех ребер у элементов, можно получить номера границ. Так как границ, т.е. поверхностей может быть много, то ребра получат соответствующие коды.

Реализация алгоритма в программном коде выглядит очень просто благодаря использованию технологии LINQ [63] и класса Edge с описанными методами GetHashCode() и Equals(), а также для сортировки списка метода CompareTo().

В приведенном коде (рисунок 3.9) показан пример обработки списка ребер, в котором содержатся повторяющиеся ребра (List<Edge> edges), собранные по всем элементам.

```
edges = edges.AsParallel().Distinct().AsParallel().ToList();
edges.Sort();
```

Рисунок 3.9 – Пример обработки списка с использованием LINQ

Сначала производится обработка исходного списка. В результате этой операции в списке остаются только уникальные значения. Причем если не выполнять расширяющий метод ToList(), то список не обновится, т.к. по умолчанию метод возвращает объект класса IEnumerable<T>, который поддерживает отложенное выполнение. В таком случае обработка произойдет лишь при следующем обращении к нему. Так как исходный объект, подвергаемый обработке: список (List <T>), выполняем ToList() и сразу получаем обновленный список.

3.3.6 Программа использует МКЭ, который сводит решение к системе линейных уравнений (СЛАУ). Для их получения необходимо вычисление локальных матриц жесткости (ЛМЖ) элементов, а также глобальной матрицы жесткости (ГМЖ) ансамбля элементов. В программе реализовано составление ЛМЖ в зависимости от типа элементов сетки. В общем случае все элементы реализуют интерфейс IElement. Вид используемых матриц описан в разделе 2.

Каждый элемент имеет свою локальную матрицу жесткости. Ее составление и вычисление реализовано посредством описания методов для их вычисления в классах элементов. Для этого есть два метода: GetLocalE() и GetLocalF(). Оба возвращают матрицы, имеющие размерность $[N_{el.edge} \times N_{el.edge}]$, то есть по количеству ребер элемента ($N_{el.edge}$). Такая реализация основана на принципах ООП, когда каждый объект наделяется присущими ему свойствами и методами (абстракция).

Составление ГМЖ основано на сборке матрицы из ЛМЖ (рисунок 3.10). Для этого выполняется цикл по всем элементам сетки, в котором для каждого элемента вызывается построение его собственной локальной матрицы. Каждая такая матрица ассоциирована с ребрами элемента, которые, в свою очередь, имеют номера. Размерность глобальной матрицы $[N_{edge} \times N_{edge}]$, то есть по общему количеству уникальных ребер элементов сетки.

При помещении элементов из локальной матрицы в глобальную следует учесть то, что некоторые элементы ГМЖ будут содержать сумму значений. Причем эта сумма может включать как два, так и более значений. Это следует из того факта, что ребро может быть общим для нескольких соседних элементов. В зависимости от сгенерированной сетки, их может быть довольно много. Помещение данных из ЛМЖ в ГМЖ является простым алгоритмом.

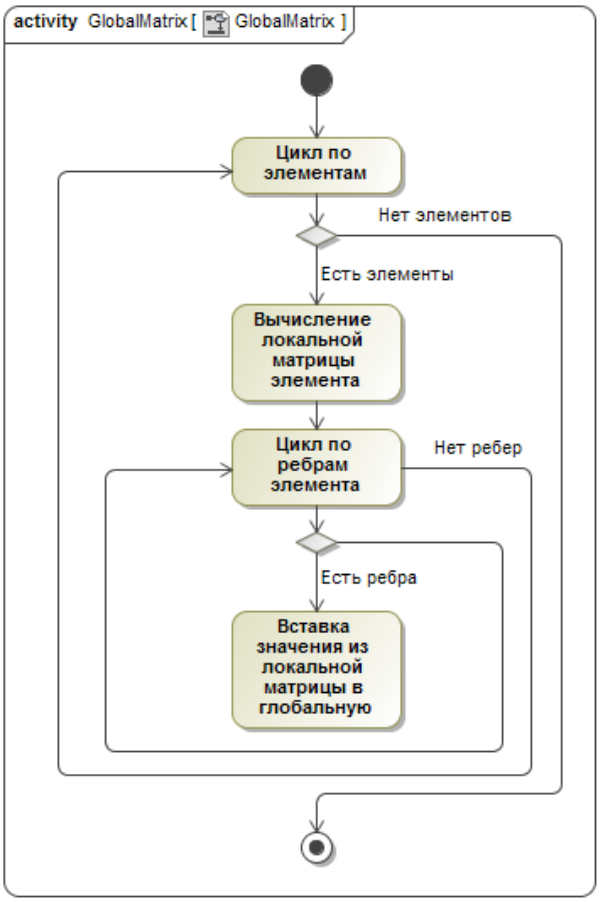


Рисунок 3.10 – Схема алгоритма построения глобальной матрицы

При построении локальной матрицы (рисунок 3.11) используются такие параметры элементов, как материал. ЛМЖ связывает основные параметры, описывающие электростатические свойства рассматриваемого объекта и составляется так, чтобы в ней были только известные параметры. Среди таких параметров можно отметить магнитную и электрическую постоянную, коэффициент проводимости и т.д. Для элемента эти параметры записываются в виде материала. Кроме того, если материал обладает дисперсными параметрами (например, электрическая проводимость зависит от круговой частоты ($\varepsilon(\omega)$)), то производится пересчет ее значения.

Алгоритмы построения и сборки локальной и глобальной матрицы реализованы в классе MatrixBuilder.

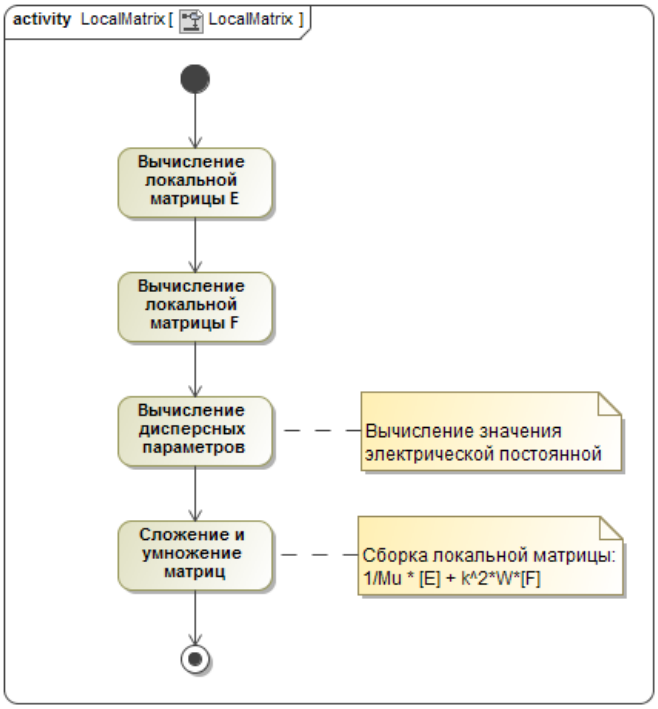


Рисунок 3.11 – Схема алгоритма построения локальной матрицы элемента

В зависимости от расположения, элемент будет обладать различными свойствами. Эти свойства получаются в процессе построения сетки. Алгоритм установки материала к элементу работает следующим образом:

– просматриваются все объемные элементы сетки;

– каждый элемент обладает номером области, в которой он расположен;

– каждая область имеет ассоциированный с ней материал (электростатические параметры), который определяется во входном файле;

– для каждого элемента из соответствующей области назначается свой материал.

В итоге элементы сетки получают обновление для электростатических свойств, ассоциированных с элементом в виде материала.

3.3.7 Учет граничных условий (ГУ) в программе реализован в виде алгоритмов модификации ГМЖ ансамбля векторных (реберных) элементов, а также составлении вектора воздействий.

Граничные условия Дирихле учитывают условия на границе области. Для их задания используются граничные ребра, номера которых получаются на основе выбора номеров из списка ребер в элементах. Так как ребер изначально много, то для них применяется обработка в виде сортировки по возрастанию номера ребра и удаления повторов.

3.3.7.1 Алгоритм применения ГУ Дирихле выглядит следующим образом.

Имеется *N1* ребер на границе *Г1*. Номера граничных ребер находятся в массиве *nd*, а предопределенные значения тангенциальной компоненты поля на границе находятся в массиве *p*. Тогда:

$$b_{nd(i)} = p(i), \qquad K_{nd(i),nd(i)} = 1, \qquad K_{nd(i),j} = 0 \text{ для } j \neq nd(i)$$

и

$$b_j \leftarrow b_j - K_{j,nd(i)} \cdot p(i), \qquad K_{j,nd(i)} = 0 \text{ для } j \neq nd(i),$$

где *b* – вектор воздействий, *K* – глобальная матрица жесткости.

3.3.8 МКЭ в конечном итоге сводится к решению системы линейных уравнений (СЛАУ). Для решения применяется метод биспоряженных градиентов (BiCGStab).

Решение организовано таким образом, что его можно проводить многократно. Параметры задаются во входном файле.

Многократное решение заключается в том, что происходит изменение исходного параметра (или параметров) решаемой задачи (изменение частоты источника при моделировании в частной области), это приводит к тому, что необходимо произвести перестроение глобальной матрицы и вектора воздействий. Такое решение генерирует больше результатов, а также требует значительно больше времени на решение. Основные затраты связаны с решением СЛАУ, которое может иметь достаточно большую размерность и потребует соответственно больших затрат вычислительных ресурсов и времени.

Для решения СЛАУ применяются классы для хранения разряженных матриц (SparseMatrix) и разряженных векторов (SparseVector).

Кроме затрат на решение СЛАУ достаточно много ресурсов требуется для обработки сетки и построения списка элементов, а также на операции вводы вывода (запись результатов в файлы). Для сетки необходимо хранить структуры данных, которые описывают каждый элемент и его свойства.

На рисунке 3.12 показана схема работы программы при решении задачи. Эта схема описывает действие Solve, которое определено в Input API и интерпретируется как решение задачи от начала до конца с выводом результатов. Это наиболее требовательное и полное действие.

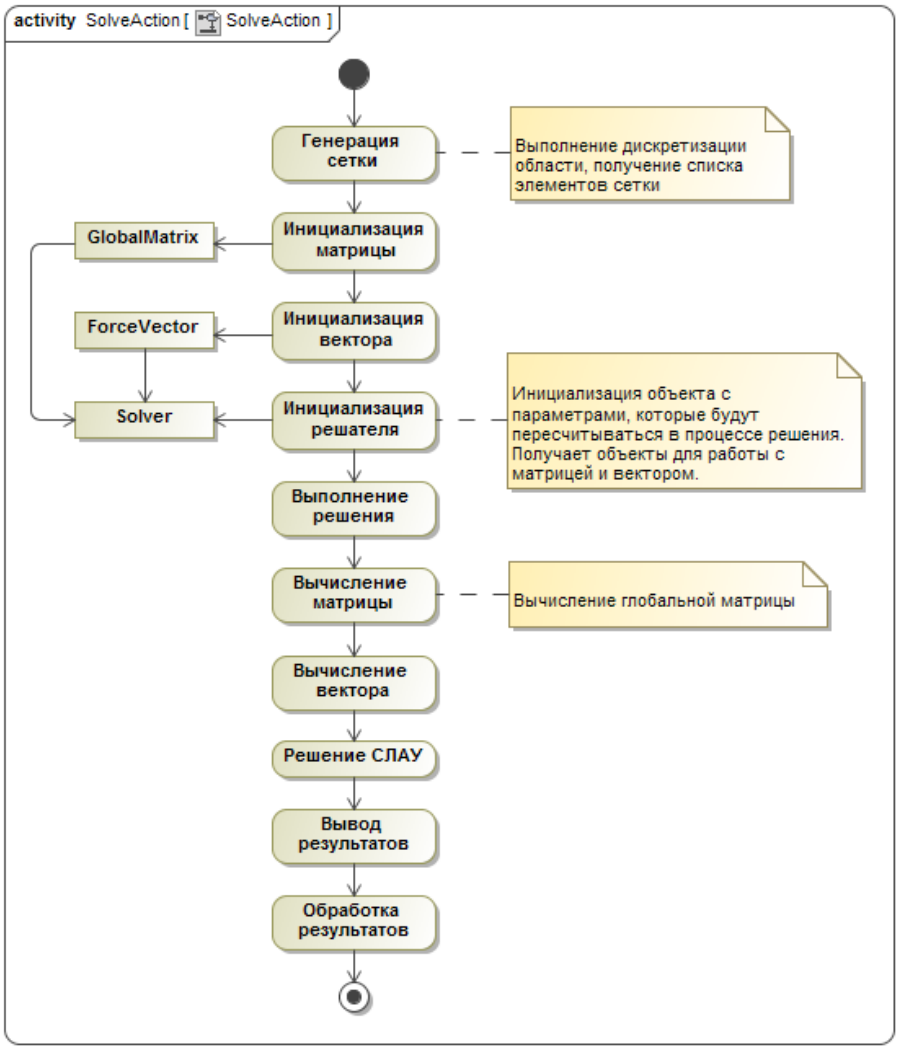


Рисунок 3.12 – Схема решения задачи

3.3.9 Обработка результатов решения основана на сборе информации из файлов, в которые были записаны результаты решения, полученные после решения СЛАУ, которые содержат компоненты поля на ребрах элементов сетки.

Эти данные представляют собой локальные компоненты поля на элементах. Необходимо выполнить обработку их таким образом, чтобы можно было узнать полное поле, глобальное на элементе и в любой точке области. Для перехода к глобальному представлению, необходимо воспользоваться исходными формулами, которыми определяют векторные компоненты на ребре. Это связано с довольно сложными вычислениями *L*-координат для трехмерного случая, а также операциями с матрицами по переводу локального поля на каждом элементе, найденного в виде тангенциальных составляющих поля на ребрах, к представлению в направлениях в декартовой системе координат ($E(x,y,z)$).

Для анализа полей используется класс FieldAnalysis <TType, TType2>, использующий сгенерированную сетку со списками элементов, построение

которой было выполнено в начале выполения задачи, а также результаты, записанные в файлы после решения СЛАУ.

3.4 Особенности реализации

При разработке структуры ПО особое внимание было уделено построению объектно-ориентированного программного кода, который как можно лучше бы описывал предметную область и ее объекты [62]. Это позволило выделить атомарные объекты задачи в виде отдельных классов. Среди таких объектов можно отметить: ребро (Edge), узел (Node), вектор (Vector) и другие. Они определяют реальные объекты, которыми приходится манипулировать при решении.

Для удобства работы с некоторыми объектами, которые нужно сортировать и сравнивать между собой предусмотрены интерфейсы IComparable<> и IEquatable<>. Первый помогает реализовать сортировку объектов класса. Для этого реализуется метод CompareTo(), который возвращает значение целого типа (только -1, 0 и 1). Второй реализует выполнение структурного сравнения объектов путем описания метода Equals(). Кроме того, для реализации полного набора методов для сортировки и поиска уникальных значений выполняется перегрузка метода GetHashCode(), который используется при сравнении объектов путем сравнения их хэш-значений.

Код программы имеет встроенную документацию, построенную на основе комментариев C#. Текст документации представлен на русском языке. Причем, данная документация позволяет использовать функции программы с описанием их содержания и аргументов, а также возвращаемых результатов. Документация может быть также сгенерирована в виде отдельных файлов с помощью специальных утилит (VsDocman, Sandcastle).

Кроме того, для уменьшения повторов однотипного кода используются возможности языка C#, такие как обобщения [63]. Это позволяет написать одну функцию для одного класса объектов, например, для обработки типов элементов, которые наследуются от класса Element, и не только. Обобщения позволяют указать произвольный тип, что позволяет создавать универсальные функции, принимающие в виде аргумента множество типов, но выполняющие одинаковые функции. Они применяются в программе как к классам, так и к методам. Некоторые классы и методы имеют более одного обобщенного типа параметра (например, класс TMesh<TType, TType2>).

Для хранения данных широко применяются коллекции, в которых хранятся объекты собственных типов, реализованных в программе. Это позволяет применять обработку списков с помощью LINQ [63], синтаксис которого упрощает код и делает его более понятным. По скорости работы он не уступает обычным циклам. Использование PLINQ позволяет выполнять некоторые запросы гораздо быстрее, используя параллельные библиотеки (TPL). Применение LINQ тесно связано с лямбда-выражениями, которые

позволяют записывать задания в простом виде без необходимости написания дополнительных операторов сравнения. При работе с LINQ используются расширяющие методы, позволяющие записывать необходимые действия в одну строку. Выполнение запросов происходит сразу, так как данные представляются в виде списков, которые не поддерживают отложенное выполнение LINQ запросов, а сразу получают нужные данные и знают их количество (емкость списка).

При реализации обработки действий в программе используются делегаты. В данном случае используется обобщенный делегат Action<T> [63], который принимает функции, возвращающие значение void. Это упрощает обработку действий до того, что в зависимости от ассоциации с методом, они будут выполнены автоматически. Ассоциации действий с методами, которые их реализуют, основано на использовании словарей (Dictionary<TKey,TValue>), в которых ключами выступают название действия (ActionTypes), а значениями – метод реализации (Action<InputData>).

Кроме применения делегата Action<T>, применяется делегат Func<Tin, Tout> [63], который, в отличие от Action, возвращает значение указанного типа. Это дает возможность задавать через него функции, которые имеют однотипное описание, но при этом возвращают значения заданного типа и имеют разное содержание.

Решение, созданное в Visual Studio 2010, состоит из нескольких проектов, которые следуют принципу модульности. Это позволяет редактировать коды отдельных модулей, тестировать и проверять их работу по отдельности. При этом зависимость модулей от основной программы отсутствуют.

Разработанная программа получила название ElectroMagnetic Field FEM Modeler (EMFFM). Исходные коды приведены в приложении В.

4 ОПЫТНАЯ ЭКСПЛУАТАЦИЯ И ВЕРИФИКАЦИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1 Эксплуатация разработанного ПО

Программа представляет собой консольное приложение. Для его запуска необходимо в командной строке выполнить:

dir> app.exe input.xml

где app.exe – название исполняемого файла программы (MainApp.exe), input.xml – входной файл с данными задачи.

Если входной файл не указан, будет открыт файл по умолчанию, который расположен в папке input и называется in.xml.

После запуска приложения начнется его выполнение в соответствии с описанным сценарием во входном файле, если он будет безошибочно прочитан. В процессе выполнения программы на экран будут выводиться сообщения (рисунок 4.1). Эти сообщения реализуются с помощью библиотеки NLog [68]. Причем они дублируются в файлах с логами, которые, в случае необходимости, можно изучить подробнее (ведется два лога: полный, в который пишутся все сообщения, и отладочный (debug)).

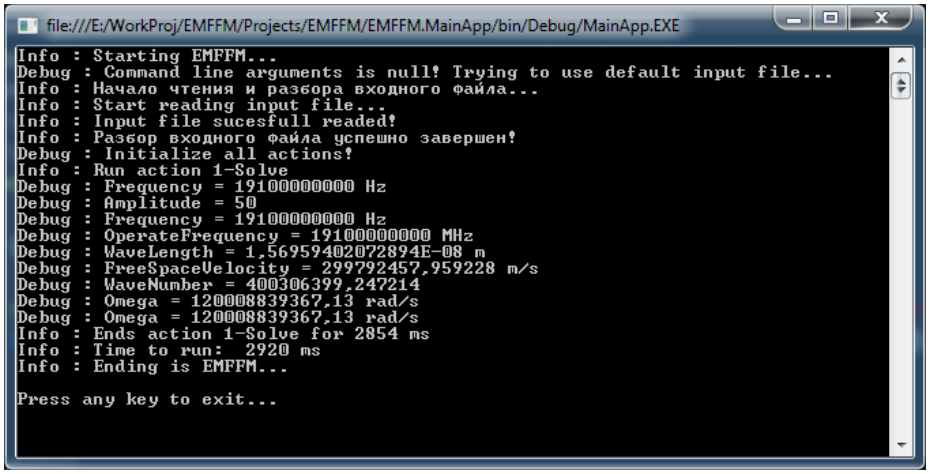


Рисунок 4.1 – Вид окна приложения

Выполняемые действия соответствуют сценарию, описанному во входном файле. Порядок выполнения и сами операции будут определяться после обработки входного файла.

Запуск программы сопровождается открытие окна приложения Netgen, консольного окна, в котором будут выводиться сведения о процессе генерации сетки. Netgen завершится автоматически после выполнения построения сетки и вывода результата в заданный файл.

В конце работы приложения программа производит несколько файлов, которые будут содержать сведения о сетке, а также результаты решения СЛАУ. Основные данные будут содержаться в файле field-full.txt, в котором

содержатся сведения по каждому элементу, о полях на его ребрах и значение полного поля в центре элемента. Поле представлено в точке в виде полных его компонент по каждому направлению (*X*, *Y*, *Z*).

Если указаны параметры для пробных областей, то будут файлы, которые содержат сведения о полях для элементов, расположенных в указанной области.

4.1.1 Порядок выполнения действий по подготовке задачи к решению.

Для решения задачи необходимо:

– сформировать полные сведения о решаемой задаче (определить значения переменных, выбрать источники излучения и их параметры и т.д.);

– выполнить построение геометрии и сформировать в виде специального файла (приложение Д);

– создать входной файл на основе Input API Specification с данными о задаче (приложение З);

– указать параметры решения задачи, в том числе выполняемые действия;

– запустить программу с созданными файлами на выполнение.

Если программа успешно выполнит все операции и не возникнет никаких ошибок, то получатся несколько файлов с результатами. Если при указании параметров задачи была включена опция вывода, то в папке с результатами будут файлы сетки и другие файлы.

4.1.2 Для анализа полученных результатов необходимы:

– файлы с результатами (один итоговый файл с полями по элементам);

– файл сетки, в котором перечислены координаты узлов и список элементов сетки.

Результирующие поля на элементах определяются в их центре, поэтому в зависимости от размера элемента и положения его центра, значения могут несколько отличаться между собой.

При необходимости нахождения поля в точке, необходимо произвести поиск элемента, в котором располагается искомая точка. Далее, в результирующих файлах найти нужный элемент и принять значение в его центре.

Проведения анализа и его интерпретация является сложной задачей. При наличии множества элементов анализ затруднителен. Программа приводит лишь значения в центре элемента. Однако, при наличии сетки, формул для получения представления поля на элементе (раздел 2.1.4), набора точек, в которых ищутся поля, а также значения полей на ребрах элементов можно выполнить необходимый анализ.

4.1.3 Построение файла с геометрией.

После определения размеров и формы объектов задачи, необходимо сформировать рассматриваемую область таким образом, чтобы можно было построить конечноэлементную сетку. Для создания файла геометрии есть два пути:

– написание XML файла с последующим его преобразованием в *.geo файл, необходимый программе Netgen;

– написание файла геометрии *.geo.

В не зависимости от выбранного способа, они оба поддерживаются в программе. Для обработки XML файла разработана специальная библиотека NetgenUtils (приложение Д), позволяющая выполнять необходимые преобразования.

Для иллюстрации примера создания геометрии, на рисунке 4.2 приведен файл XML, который является более универсальным средством описания, а также рисунок 4.3, который является результатом автоматической генерации на основе XML файла из рисунка 4.2.

```
<?xml version="1.0" encoding="utf-8"?>
<geometrydata>
 <points>
  <point name="p1" x="0" y="0" z="0"/>
  <point name="p2" x="490" y="490" z="490"/>
  <point name="p3" x="245" y="245" z="245"/>
 </points>
 <solids>
  <orthobrick name="cube1" p1="p1" p2="p2" boundary="1"/>
  <sphere name="sp" p="p3" radius="40" boundary="2"/>
  <complex name="main" data="cube1 and not sp"/>
 </solids>
 <colors>
  <color name="col1" r="0" g="0" b="1"/>
  <color name="col2" r="0" g="1" b="0"/>
 </colors>
 <output>
  <tlo name="main" transparent="true" color="col1"/>
  <tlo name="sp" transparent="false" color="col2"/>
 </output>
</geometrydata>
```

Рисунок 4.2 – Пример XML файла с геометрией geom1.xml

Здесь видно, что формат XML позволяет существенно упростить описание геометрии, путем описания ее по частям, где каждая из них логически связана с другими. При этом, каждый тип объектов помещен в свою секцию (точки, вектора, тела, цвета, объекты для отрисовки). Для правильного формирования файла геометрии имеется XML Schema (приложение Е).

```
#
# Geometry file generated by NetgenUtils at 9:03
#
algebraic3d
solid cube1 = orthobrick (0, 0, 0; 490, 490, 490) -bc=1;
solid sp = sphere (245, 245, 245; 40) -bc=2;
solid main = cube1 and not sp;
tlo main -transparent -col=[0,0,1];
tlo sp -col=[0,1,0];
# End of file
```

Рисунок 4.3 – Пример файла геометрии geom1.geo

При использовании первого варианта, XML, во входном файле необходимо указать параметр о генерации файла геометрии (IsGenerateGeomFile=true). В таком случае, перед генерацией самой сетки и вызовом программы Netgen произойдет обработка XML файла и формирование на его основе *.geo файла, который уже в свою очередь будет передан в Netgen.

4.1.4 Результаты работы программы.

После выполнения всех действий, программа сгенерирует следующие файлы:

- файлы сетки (geom1.neu, geom1.xml, mesh.xml);
- файл с результатами (field-full-0.txt, field-full-domain-2-0.txt);
- файл с вектором воздействий (field-vector-0.txt);
- файл с результатом решения СЛАУ (field-result-0.txt).

Это файлы, в случае, если вывод в опциях включен, при решении без изменения параметров решения. При решении с разными значениями параметров для каждой итерации результаты будут различны и в большом количестве.

Файлы сеток важны при обработке результатов. Если необходимо взять построенную сетку для другой программы, то можно воспользоваться XML файлом, обработать который очень просто. Самый полный вариант сетки представлен в файле mesh.xml.

```
<?xml version="1.0" encoding="utf-8" ?>
 <mesh>
 <elements count="2445">
  <element number="1">
  <nodes>
  <node number="71" x="254.562312" y="223.933526" z="277.63075" />
  <node number="82" x="241.167404" y="233.239388" z="283.039443" />
  <node number="397" x="238.189663" y="211.915086" z="290.480389" />
  <node number="85" x="238.748204" y="215.916427" z="271.740622" />
  </nodes>
  <edges>
```

```
  <edge number="773" vetr1="71" vert2="82" isBoundary="true" boundary="2" />
  <edge number="1" vetr1="71" vert2="397" />
  <edge number="2" vetr1="71" vert2="85" isBoundary="true" boundary="2" />
  <edge number="2337" vetr1="82" vert2="397" />
  <edge number="2712" vetr1="85" vert2="82" isBoundary="true" boundary="2" />
  <edge number="1930" vetr1="397" vert2="85" />
  </edges>
  <center x="243.16689574999998" y="221.25110675" z="280.722801" />
  </element>
  <!-- перечисление остальных элементов сетки -->
</elements>
</mesh>
```

Рисунок 4.4 – Файл mesh.xml

Кроме сетки, важными являются файлы с результатами вычислений поля на элементах. Вместе с данными об элементах, можно определить величины поля в различных точках области.

# Element	FieldX	FieldY	FieldZ
1	1,351718042429E-06	6,16414780658398E-07	-2,92471423408298E-06
2	-5,78280228934E-09	2,02670958697583E-08	1,22146823206513E-08
3	-1,772143919371E-09	-1,010562615660E-09	-5,606104478339E-10
4	2,613954058788E-10	-1,51146564373541E-11	-3,56737678003309E-10

Рисунок 4.5 – Файл с полными полями на элементах

На рисунке 4.5 приведена малая часть файла с результатами. Для элементов приведены только полные поля, значения на ребрах не выводились (указана опция WithEdgeFields = false во входном файле). Если включена опция вывода полей на ребрах, то для каждого элемента также будут выведены значения тангенциальных компонент на каждом ребре.

4.2 Верификация программного обеспечения

В качестве тестовой задачи воспользуемся задачей о рассеянии падающей плоской монохроматической волны идеально проводящей сферической частицей. Материалом частицы является серебро (Ag).

Размер области выбирается так, чтобы от сферы в центре граница была равноудалена на расстоянии 1λ (λ – длина волны).

Геометрия области решения задачи показана на рисунке 4.6. Для ее определения используется XML файл (приложение Н). При этом в геометрию вводится специальный объект, который будет представлять собой небольшую область, в которой будут определяться ближнее поле вокруг частицы (достаточно малая по сравнению с основной областью). Этой областью является куб со сторонами равноудаленный от поверхности шара на $0{,}375D$ (D – диаметр сферической частицы).

Построенная сетка имеет вид, представленный на рисунке 4.7, где в центре отмечена частица и ее ближняя область. Ближней областью также считается вся область задачи в целом, но в целях уменьшения рассматриваемых результатов, введение дополнительных объектов сокращает количество изучаемых элементов сетки.

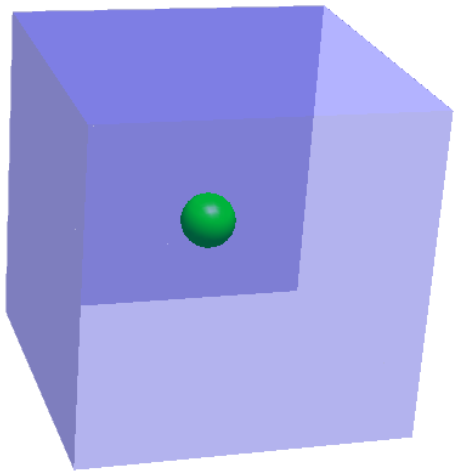


Рисунок 4.6 – Вид геометрии рассматриваемой области

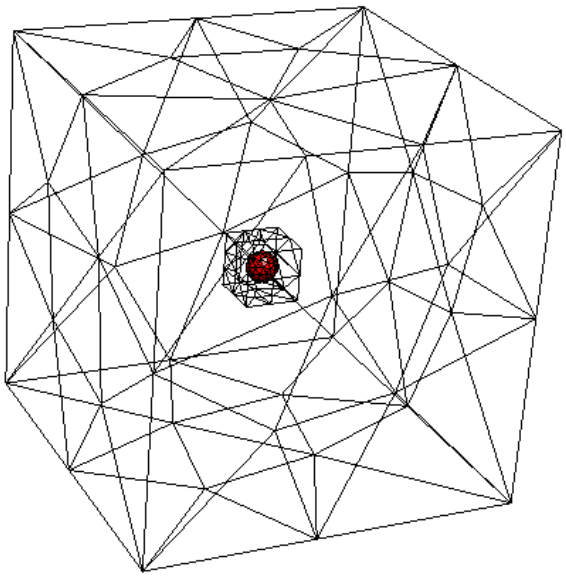


Рисунок 4.7 – Поверхностная сетка

Для описания входных данных задачи спользовался XML файл, созданный в соответствии с Input API (приложение Н).

Частица рассматривается как плазмон, поэтому к ней применяется закон дисперсии, по которому определяется зависимость диэлектрической проводимости от частоты работы источника ($\varepsilon(\omega)$).

Решение данного класса задач имеет и аналитический вид, который получен Ми [72] и описано во многих книгах [3, 4] и др.

Полученные с помощью разработанной программы результаты дают сведения о ближней области вокруг частицы, а также во всей рассматриваемой области. Расхождение решений составляет не более 15-25%.

Разработанное ПО показывает результаты с достаточной точностью и степенью адекватности. На сегодня в программе реализован анализ в частотном диапазоне. Для решения более сложных задач будет производиться совершенствование математической модели, добавление в нее законов дисперсии, создание модели для решени задач во временной области и многого другого.

Дальнейшее развитие будет включать: разработку пользовательского интерфейса, создание базы материалов и их свойств, развитие модуля анализа результатов.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ

5.1 Технико-экономическое обоснование целесообразности разработки ПП и оценка его конкурентоспособности

Сфера применения разрабатываемого проекта – моделирование и синтез новых материалов. Учреждения и организации, занимающиеся исследованием материалов, используют моделирование электромагнитных полей, разрабатывают свои продукты для проведения исследований. Эти продукты распространяются бесплатно и имеют открытые исходные коды. Они доступны другим организациям и пользователям, которые сталкиваются со сложными задачами и пытаются их решить с помощью разработанных программных комплексов. Часто ведутся разработки отдельных программных продуктов (ПП) для решения узкого круга задач. Это происходит тогда, когда существующие ПП не применимы к решаемой задаче.

Обычное явление в научно-исследовательской сфере это разработка новых специализированных продуктов, предназначенных для решения задач в рамках исследовательской программы. Разработка одного из таких проектов и ведется в данной работе.

Для разработки продукта необходим минимальный набор инструментов, которые для учреждений образования и научно-исследовательских организаций могут быть доступны по специальным программам. Дополнительные программы и библиотеки, используемые при разработке, распространяются бесплатно (по лицензиям GNU GPL, Common Creative, MIT и др.).

Среди особенностей использования разрабатываемого продукта можно отметить следующее:

- простота работы с приложением;
- возможность визуализации входной геометрии;
- возможность расчета как двухмерных, так и трехмерных задач;
- открытый исходный код.

В качестве базового продукта берется программа Elcut (Quick Field). В таблице 5.1 представлены некоторые параметры, по которым можно сравнить существующий коммерческий пакет и разрабатываемый. Сравнение этих продуктов не совсем корректное, так как разрабатываемый продукт попадает под категорию свободных пакетов, типа Meep, EMAP и др. Поэтому в таблице 5.1 также приведен продукт Meep.

Таблица 5.1 – Основные показатели сравнительного анализа вариантов ПП

Показатели	Варианты			Результаты сравнения: повышение (+), понижение (-)
	Базовый 1 (Elcut)	Базовый 2 (Meep)	проектируемый	
Расчет 2D и 3D полей	только 2D (в разрезе 3D)	2D и 3D	2D и 3D	-/+/+
Открытая платформа	нет	да	да	-/+/+
Визуальная среда проектирования	да	нет	нет	+/-/-
Расширяемость	возможность написания дополнений	да (сложна)	да	-/+/+
Метод	FEM	FDTD	VFEM	+/-/+

Выполним бальную оценку основных функциональных возможностей продуктов. Для сравнения возьмем бесплатный пакет Meep, с которым было проще ознакомиться и опробовать его в действии. Сравнение показателей приведены в таблице 5.2. Интервал оценок: от 1 до 5. Наилучшему значению присваивается максимальный балл 5.

Коэффициент изменения функциональных возможностей ($K_{\text{ф.в}}$) определяется по формуле:

$$K_{\text{ф.в}} = \frac{K_{\text{ф.в.н}}}{K_{\text{ф.в.б}}}, \qquad (5.1)$$

где $K_{\text{ф.в.н}}$ и $K_{\text{ф.в.б}}$ – бальная оценка неизмеримых показателей разрабатываемого и базового изделия соответственно.

Таблица 5.2 – Коэффициент измерения функциональных возможностей

Наименование показателя	Балльная оценка базового ПП	Балльная оценка нового ПП
Объем памяти	5	4
Функциональные возможности	4	5
Быстродействие	3	4
Удобство интерфейса	4	5
Удобства формулировки исходных данных	4	5
Производительность труда	4	4
Итого:	24	27
Коэффициент функциональных возможностей	27/24=1,13	

Конкурентоспособность нового ПП по отношению к базовому можно оценить с помощью интегрального коэффициента конкурентоспособности, учитывающего все ранее рассчитанные показатели:

$$K_{\text{и}} = \frac{K_{\text{ф.в}} \times K_{\text{н}}}{K_{\text{ц}}}, \qquad (5.2)$$

где $K_{\text{н}}$ – коэффициент соответствия нового ПО нормативам ($K_{\text{н}} = 1$);

$K_{\text{ц}}$ – коэффициент цены потребления ($K_{\text{ц}} = 1$).

Таблица 5.3 – Рассчитанный уровень конкурентоспособности нового ПП

Коэффициенты	Значение
Коэффициент изменения функциональных возможностей ($K_{\text{ф.в}}$)	1,13
Коэффициент соответствия нормативам ($K_{\text{н}}$)	1
Коэффициент цены потребления ($K_{\text{ц}}$)	1
Интегральный коэффициент конкурентоспособности	(1,13*1)/1=1,13

По результатам проведенной оценки установлено, что разрабатываемый проект обладает техническими преимуществами по сравнению с конкурентными аналогами, что определяет его сильные позиции на рынке программных продуктов. Для более полного обоснования целесообразно провести оценку его экономической эффективности.

5.2 Оценка трудоемкости работ по созданию программного обеспечения

В соответствии с [73] основой для определения общей трудоемкости разработки программного обеспечения (далее – ПО), объемов финансирования на стадии его технико-экономического обоснования используются укрупненные нормы затрат труда. На основе общей трудоемкости разработки ПО составляется смета затрат, а также определяется численность исполнителей и трудоемкость выполняемых ими работ по этапам разработки ПО.

Стадиями разработки ПО согласно ГОСТам Единой системы программной документации (ЕСПД) являются: техническое задание (ТЗ), эскизный проект (ЭП), технический проект (ТП), рабочий проект (РП), ввод в действие (ВН).

5.2.1 Определение общего объема функций ПО.

В качестве единицы измерения объема ПО может быть использована строка исходного кода (LOC). На основании рекомендаций и нормативных

документов произведем выбор набора функций и определим показатели общего объема кода.

Для расчета суммарного объема ПО в строках исходного кода (V_o) используем формулу:

$$V_o = \sum_{i=1}^{n} V_i , \qquad (5.3)$$

где V_i - объем отдельной функции ПО;

n – общее число функций.

Для уточненного объема воспользуемся теми же данными, что и для объема функций, указанных в каталоге, так как не указано каких-либо дополнительных сведений по уточнению объемов функций. Уточненный объем ($V_{\text{у}}$) определяется по формуле:

$$V_{\text{у}} = \sum_{i=1}^{n} V_{\text{у},i} , \qquad (5.4)$$

где $V_{\text{у},i}$ – уточненный объем отдельной функции ПО (LOC).

По результатам расчета полученные данные запишем в виде таблицы 5.4. Сведения по числу строк приведены в [73, приложение 1].

Таблица 5.4 – Перечень функций и объем кода (в LOC)

Код функции	Наименование (содержание) функции	Объем строк исходного кода (LOC)	
		По каталогу ($V_{\text{о}}$)	Уточненный ($V_{\text{у}}$)
101	Организация ввода информации	330	330
104	Обработка входного языка и формирование таблиц	1040	1040
303	Обработка файлов	1050	1050
506	Обработка ошибочных и сбойных ситуаций	1540	1540
703	Расчет показателей	420	420
	Итого:	4180	4180

В качестве языка, для которого принимаются данные по объему строк исходного кода для той или иной функции выбран Java, который наиболее близко подходит к языку C#, на котором производится разработка ПО.

5.2.2 Расчет поправочных коэффициентов, учитывающих организационно-технические условия разработки программного обеспечения.

Сложность ПО определим в соответствии с [73, приложение 2]. Примем 2-ую категорию сложности, т. к. ПО относится к области моделирования объектов и явлений.

На основании принятого к расчету (уточненного) объема кода ($V_{\text{у}}$) и категории сложности – 2, определим нормативную трудоемкость ($T_{\text{н}}$) выполняемых работ по разработке ПО в соответствии с [73, приложение 3] (таблица 5.5).

Таблица 5.5 – Нормативная трудоемкость выполняемых работ

Объем ПО (строки исходного кода (LOC)), $V_{\text{у}}$	Категория сложности ПО	Номер нормы
	2-я	
4110	213	45

Определим коэффициент повышения сложности разрабатываемого ПО ($K_{\text{с}}$), который определяет дополнительные затраты труда.

Коэффициент повышения сложности определяется по формуле:

$$K_{\text{с}} = 1 + \sum_{i=1}^{n} K_i , \qquad (5.5)$$

где K_i – коэффициент соответствующий степени повышения сложности;

n – количество учитываемых характеристик.

В соответствии с [73, приложение 4] коэффициент $K_1 = 0{,}12$, соответствующий двум характеристикам ПО. Тогда, итоговый коэффициент будет равен

$$K_{\text{с}} = 1 + 0{,}12 = 1{,}12 .$$

Определение степени (категории) новизны основано на выборе области, в которую попадает разрабатываемый ПП. Так как ПО это развитие существующих моделей и концепций и не является принципиально новым, то степень новизны устанавливается в виде категории В: ПО, являющееся развитием определенного параметрического ряда ПО, разработанное для ранее освоенных типов конфигураций ПК и ОС.

Тогда коэффициент $K_{\text{н}} = 0{,}63$ в соответствии с [73, приложение 5].

В разрабатываемом приложении используются готовые модули (в виде библиотек), которые обеспечивают работу отдельных частей ПО. Их использование не велико, поэтому коэффициент, учитывающий использование стандартных модулей ($K_{\text{т}}$) принимается по таблице 5.6 в соответствии с [73, приложение 6].

Таблица 5.6 – Коэффициент, учитывающий степень использования стандартных модулей ($K_{\text{т}}$)

Степень охвата реализуемых функций разрабатываемого ПО стандартными модулями	Значение $K_{\text{т}}$
до 20%	0,9

Коэффициент, учитывающий средства разработки ПО ($K_{\text{ур}}$) определяется в соответствии с [73, приложение 7]. В расчет принимается $K_{\text{ур}} = 1{,}0$, который соответствует платформе IBM PC Windows и средствам разработки в виде процедурных языков высокого уровня.

Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости определяются с учетом установленной новизны ПО согласно [73, приложение 8] (таблица 5.7). При этом, сумма значений удельных весов должны быть равна 1:

$$\sum_{i=1}^{n} K_{\text{уд}i} = 1. \qquad (5.6)$$

Таблица 5.7 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости

Категория новизны ПО	Без применения CASE-средств				
	Стадии разработки ПО				
	ТЗ	ЭП	ТП	РП	ВН
	Значения коэффициентов				
	$K_{\text{тз}}$	$K_{\text{эп}}$	$K_{\text{тп}}$	$K_{\text{рп}}$	$K_{\text{вн}}$
В	0,08	0,19	0,28	0,34	0,11

5.3 Расчет трудоемкости выполняемых работ по стадиям разработки программного обеспечения

Рассчитаем нормативную трудоёмкость ПО ($T_{\text{н}}$), скорректированную по стадиям, с использованием коэффициентов $K_{\text{тз}}$, $K_{\text{эп}}$, $K_{\text{тп}}$, $K_{\text{рп}}$, $K_{\text{вн}}$, определенных ранее:

$$T_{\text{утз}} = T_{\text{н}} \cdot K_{\text{тз}} \cdot K_{\text{с}} \cdot K_{\text{н}} \cdot K_{\text{ур}} = 213 \cdot 0{,}08 \cdot 1{,}12 \cdot 0{,}63 \cdot 1{,}0 = 12{,}02, \qquad (5.7)$$

$$T_{\text{уэп}} = T_{\text{н}} \cdot K_{\text{эп}} \cdot K_{\text{с}} \cdot K_{\text{н}} \cdot K_{\text{ур}} = 213 \cdot 0{,}19 \cdot 1{,}12 \cdot 0{,}63 \cdot 1{,}0 = 28{,}55, \qquad (5.8)$$

$$T_{\text{утп}} = T_{\text{н}} \cdot K_{\text{тп}} \cdot K_{\text{с}} \cdot K_{\text{н}} \cdot K_{\text{ур}} = 213 \cdot 0{,}23 \cdot 1{,}12 \cdot 0{,}63 \cdot 1{,}0 = 34{,}57, \qquad (5.9)$$

$$T_{\text{урп}} = T_{\text{н}} \cdot K_{\text{рп}} \cdot K_{c} \cdot K_{\text{н}} \cdot K_{\text{ур}} \cdot K_{\text{т}} = \\ = 213 \cdot 0{,}34 \cdot 1{,}12 \cdot 0{,}63 \cdot 1{,}0 \cdot 0{,}9 = 45{,}99, \qquad (5.10)$$

$$T_{\text{увн}} = T_{\text{н}} \cdot K_{\text{вн}} \cdot K_{\text{с}} \cdot K_{\text{н}} \cdot K_{\text{ур}} = 213 \cdot 0{,}11 \cdot 1{,}12 \cdot 0{,}63 \cdot 1{,}0 = 16{,}53. \qquad (5.11)$$

5.4 Расчет общей трудоемкости разработки программного обеспечения

Общая трудоёмкость получается путем суммирования нормативной (скорректированной) трудоемкости по стадиям разработки ПО:

$$T_{\text{о}} = \sum_{i=1}^{n} T_{\text{у}i}\,, \qquad (5.12)$$

где $T_{\text{у}i}$ – нормативная трудоемкость разработки ПО на i-ой стадии;

n – число стадий разработки.

Все полученные данные по трудоемкости сводятся в таблицу 5.8.

Таблица 5.8 – Результаты расчета общей трудоемкости разработки ПО

№ п/п	Показатели	Стадии разработки					Итого
		ТЗ	ЭП	ТП	РП	ВН	
1	Общий объем ПО ($V_{\text{о}}$), кол-во строк LOC	-	-	-	-	-	4180
2	Общий уточненный объем ПО ($V_{\text{у}}$), кол-во строк LOC	-	-	-	-	-	4180
3	Категория сложности разрабатываемого ПО	-	-	-	-	-	2
4	Нормативная трудоемкость разработки ПО ($T_{\text{н}}$), чел.-дн.	-	-	-	-	-	213
5	Коэффициент повышения сложности ПО ($K_{\text{с}}$)	1,12	1,12	1,12	1,12	1,12	-
6	Коэффициент, учитывающий новизну ПО ($K_{\text{н}}$)	0,63	0,63	0,63	0,63	0,63	-
7	Коэффициент, учитывающий степень использования стандартных модулей ($K_{\text{т}}$)	-	-	-	-	-	0,63
8	Коэффициент, учитывающий средства разработки ПО ($K_{\text{ур}}$)	1,0	1,0	1,0	1,0	1,0	-
9	Коэффициенты удельных весов трудоемкости стадий разработки ПО ($K_{\text{ТЗ}}$, $K_{\text{ЭП}}$, $K_{\text{ТП}}$, $K_{\text{РП}}$, $K_{\text{ВН}}$)	0,08	0,19	0,28	0,34	0,11	1,0
10	Распределение нормативной трудоемкости ПО по стадиям, чел.-дн.	17,04	40,47	59,64	72,42	23,43	213
11	Распределение скорректированной (с учетом $K_{\text{с}}$, $K_{\text{н}}$, $K_{\text{т}}$, $K_{\text{ур}}$) трудоемкости ПО по стадиям, чел.-дн.	12,02	28,55	34,57	45,99	16,53	137,66
12	Общая трудоемкость разработки ПО ($T_{\text{о}}$), чел.-дн.	-	-	-	-	-	137,66

5.5 Расчет затрат на разработку (себестоимость) программного продукта

В состав затрат на разработку ПП входят следующие статьи расходов:

– затраты труда на создание ПП (затраты по основной, дополнительной заработной плате и соответствующие отчисления) ($З_{\text{тр}}$);

– затраты на технологию (затраты на приобретение и освоение программных средств, используемых при разработке ПП; затраты на ПО, используемое как эталон) ($З_{тех}$);

– затраты на машинное время (расходы на содержание и эксплуатацию технических средств разработки, эксплуатации и сопровождения) ($З_{м.в}$);

– затраты на материалы (информационные носители) ($З_{мат}$);

– затраты на энергию, на использование каналов связи (для отдельных видов);

– общепроизводственные расходы (затраты на управленческий персонал, на содержание помещений) ($З_{общ.пр}$);

– непроизводственные (коммерческие) расходы (затраты связанные с рекламой, поиском заказчиков, поставками конкретных экземпляров) ($З_{непр}$).

В случае текущей работы, затраты на средства разработки и инструменты отсутствуют вследствие того, что все предоставляется как бесплатное ПО в рамках программы DreamSpark от Microsoft для учебных заведений.

Суммарные затраты на разработку ПО ($З_р$) определяются по формуле:

$$З_р = З_{тр} + З_{тех} + З_{м.в} + З_{мт} + З_{общ.пр} + З_{непр}. \quad (5.13)$$

5.5.1 Расчет затрат на оплату труда разработчиков.

Разработчик – один человек, автор дипломной работы, расчет будем производить только для него.

Основная заработная плата рассчитывается на основе среднечасовой ставки, которая рассчитывается на основании базовой ставки 1-ого разряда и коэффициента, соответствующего разряда.

Основная заработная плата разработчика рассчитывается по формуле:

$$ЗП_{осн} = С_{ср.час} \times T_о \times K_{ув}, \quad (5.14)$$

где $С_{ср.час}$ – средняя часовая тарифная ставка, руб./час;

$T_о$ – общая трудоемкость разработки, чел.-час;

$K_{ув}$ – коэффициент, учитывающий доплаты стимулирующего характера ($K_{ув} = 1{,}5$).

Базовая ставка установлена в размере 210000 руб. с 01.05.2012 в соответствии с [74].

Предполагается, что разработчиком является инженер-программист второй категории. В качестве коэффициента для инженера-программиста 2-ой категории в соответствии с ЕТС [75] примем 13 тарифную ставку $T_к = 3{,}04$. Тогда:

$$С_{ср.ч} = \frac{С_{м1} \cdot T_к}{F_{мес}} = \frac{210000 \cdot 3{,}04}{168} = 3800 \text{ руб.} \quad (5.15)$$

Основная заработная плата будет равна:

$$ЗП_{осн} = С_{ср.час} \times T_о \times K_{ув} = 3800 \times 137{,}66 \times 1{,}5 \times 8 = 8369728 \text{ руб.}$$

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по формуле:

$$ЗП_{доп} = ЗП_{осн} \times Н_{доп} / 100\%, \quad (5.16)$$

где $Н_{доп}$ – норматив на дополнительную заработную разработчиков, который равен 10%.

$$ЗП_{доп} = 8369728 \times 10 / 100\% = 836972{,}8 \text{ руб.}$$

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитываются по формуле:

$$ОТЧ_{сн} = (ЗЗ_{осн} + ЗП_{доп}) \times Н_{зп} / 100\%, \quad (5.17)$$

где $Н_{зп}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ($Н_{зп} = 34\%$).

$$ОТЧ_{сн} = (8369728 + 836972{,}8) \times 34 / 100 = 3130278{,}272 \text{ руб.}$$

Расходы на оплату труда разработчиков с отчислениями ($З_{тр}$) определяются с использованием формулы:

$$З_{тр} = ЗП_{осн} + ЗП_{доп} + ОТЧ_{сп}. \quad (5.18)$$

Итоговая заработная разработчика будет равна:

$$З_{тр} = 8369728 + 836972{,}8 + 3130278{,}272 = 12336979{,}07 \text{ руб.}$$

5.5.2 Расчет затрат на машинное время.

Затраты машинного времени ($З_{м.в}$) определяются по формуле:

$$З_{м.в} = С_ч \times K_т \times t_{эвм}, \quad (5.19)$$

где $K_т = 1$ – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ;

$t_{эвм}$ – машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

Стоимость 1 машино-часа определяется по формуле:

$$С_ч = \frac{ЗП_{об} + З_{ар} + З_{ам} + З_{эл} + З_{в.м} + З_{т.р} + З_{пр}}{F_{эвм}}. \quad (5.20)$$

Затраты на заработную плату обслуживающего персонала определяются по формулам (5.21) – (5.24):

$$ЗП_{об} = \frac{ЗП_{осн.об} + ЗП_{доп.об} + ОТЧ_{эп.об}}{Q_{эвм}}, \quad (5.21)$$

$$ЗП_{осн.об} = 12 \times \sum_i \left(С_{м.об.i} \times n_i\right), \quad (5.22)$$

$$ЗП_{доп.об} = ЗП_{осн.об} \times Н_{доп} / 100\%, \quad (5.23)$$

$$ОТЧ_{зп.об} = \left(ЗП_{осн.об} + ЗП_{доп.об}\right) \times Н_{зп} / 100\%. \quad (5.24)$$

Данные по обслуживающим работникам приведены в таблице 5.9.

Таблица 5.9 – Расчет основной заработной платы обслуживающего персонала

№ п/п	Категория обслуживающего персонала (i)	Кол-во человек (n_i)	Тарифный коэффициент ($T_{к.i}$)	Месячная тарифная ставка ($С_{м.i}$)
1.	Оператор	1	2,32	487200

Заработная плата обслуживающего персонала:

$$ЗП_{осн.об} = 12 \times 487200 = 5846400 \text{ руб.},$$

$$ЗП_{доп.об} = 5846400 \times 10/100 = 584640 \text{ руб.},$$

$$ОТЧ_{зп.об} = (5846400 + 584640) \times 34/100 = 2186553{,}6 \text{ руб.},$$

$$ЗП_{об} = \frac{5846400 + 584640 + 2186553{,}6}{1} = 8617593{,}6 \text{ руб.}$$

Годовые затраты на аренду помещения ($З_{ар}$) определяются по формуле:

$$З_{ар} = \frac{С_{ар} \times S}{Q_{эвм}}, \quad (5.25)$$

где $С_{ар} = 54000$ – средняя годовая ставка арендных платежей, руб./м^2;

S – площадь помещения, м^2,

$$З_{ар} = \frac{54000 \times 5}{1} \times 12 = 3240000 \text{ руб.}$$

Амортизационные отчисления рассчитываются на основании стоимости оборудования и срока его эксплуатации. Так, примем в качестве срока эксплуатации 5 лет для ПЭВМ и 4 года для монитора. Следовательно, имеем 20% и 25% в качестве нормы амортизации. Данные для расчета амортизационных отчислений представлены в таблице 5.10.

Таблица 5.10 – Расчет расходов на амортизацию

№ п/п	Наименование	Кол-во (m_i)	Стоимость приобретения ($З_{пр.i}$), руб.	Балансовая стоимость единицы ЭВМ($З_{пр.i}$*(1+ $К_{доп}$), руб.	Суммарная стоимость ЭВМ, руб.	Норма амортизации ($Н_{ам.i}$), %	Сумма амортизационных отчислений ($З_{ам.i}$), руб.
1	ПЭВМ Intel Q6600 N680SLI 4Gb	1	4590000	5140800	4590000	20	1028160
2	Монитор Asus PA238Q	1	3510000	3931200	3510000	25	982800
3	Принтер HP Laserjet 1020	1	1215000	1360800	1215000	20	272160
Итого:		3	-	-	9315000	65	2283120

Стоимость электроэнергии, потребляемой за год, ($З_{эл}$) определяется по формуле:

$$З_{эл} = \frac{М_{сум} \times F_{эвм} \times С_{эл} \times A}{Q_{эвм}}, \qquad (5.26)$$

где $С_{эл} = 238{,}5$ руб. – стоимость одного кВт-ч электроэнергии;

$A = 0.95$ – коэффициент интенсивного использования мощности;

$М_{сум}$ − суммарная мощность всей применяемой для разработки техники (таблица 5.11),

$$З_{эл} = \frac{0{,}783 \times 1672{,}8 \times 238{,}5 \times 0{,}95}{1} = 296768{,}48 \text{ руб.}$$

Таблица 5.11 – Расчет суммарной мощности ЭВМ и периферийных устройств

№ п/п	Наименование оборудования	Кол-во	Мощность 1 ед., кВт	Суммарная мощность, кВт
1.	ПЭВМ Intel Q6600 N680SLI 4Gb	1	0,5	0,5
2.	Монитор Asus PA238Q	1	0,033	0,033
3.	Принтер HP Laserjet 1020	1	0,25	0,25
Итого:		3	-	0,783

Действительный фонд рабочего времени работы рассчитывается по формуле:

$$F_{\text{эвм}} = \left(Д_{\text{г}} - Д_{\text{вых}} - Д_{\text{пр}}\right) \times F_{\text{см}} \times K_{\text{см}} \times \left(1 - K_{\text{пот}}\right). \qquad (5.27)$$

Подставив значения получим:

$$F_{\text{эвм}} = (365 - 119) \times 1 \times (1 - 0{,}15) \times 8 = 1672{,}8 \text{ ч.}$$

Затраты на материалы определяются как % от балансовой стоимости ПЭВМ. Этот процент равен 1 %:

$$З_{\text{в.м}} = \frac{8100000 \times (1 + 0{,}12) \times 1}{1} \times 0{,}01 = 90720 \text{ руб.}$$

Затраты на текущий и профилактически ремонт принимаются равными 5% от балансовой стоимости ПЭВМ:

$$З_{\text{т.р}} = \frac{8100000 \times (1 + 0{,}12) \times 1}{1} \times 0{,}05 = 453600 \text{ руб.}$$

Прочие затраты, связанные с эксплуатацией ЭВМ состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют 5% от балансовой стоимости:

$$З_{\text{пр}} = \frac{8100000 \times (1 + 0{,}12) \times 1}{1} \times 0{,}05 = 453600 \text{ руб.}$$

Машинное время ЭВМ включает: время, требуемое на составление программы по блок-схеме; время на отладку программы на ЭВМ; время на редактирование, печать и оформление документации. Для расчета машинного времени ЭВМ ($t_{\text{эвм}}$, час), необходимого для разработки и отладки проекта следует использовать формулу:

$$t_{\text{эвм}} = \left(t_{\text{рп}} + t_{\text{вн}}\right) \times F_{\text{см}} \times K_{\text{см}}. \qquad (5.28)$$

Значения для времени по стадиям берутся из таблицы 5.8:

$$t_{\text{эвм}} = \left(45{,}99 + 16{,}53\right) \times 8 \times 1 = 500{,}16 \text{ ч.}$$

Теперь можем рассчитать стоимость 1 машино-часа:

$$С_{\text{ч}} = \frac{8617593{,}6 + 3240000 + 2283120 + 296768{,}48 + 90720 + 453600 + 453600}{1672{,}8} =$$

$$= \frac{15435402{,}08}{1672{,}8} = 9227{,}28 \text{руб.}$$

Затраты машинного времени определим по формуле (5.19):

$$З_{\text{м.в}} = 9227{,}28 \times 1 \times 500{,}16 = 4615116{,}365 \text{ руб.}$$

Результаты всех расчетов сводятся в таблицу 5.12, в которой представлены все значения по затратам на разработку ПО.

Таблица 5.12 – Результаты расчета суммарных затрат на разработку ПО

№ п/п	Статьи затрат	Итого, руб
1.	**Затраты на оплату труда разработчиков ($З_{\text{тр}}$)**	12336979,07
1.1	Основная заработная плата разработчиков	8369728

Продолжение таблицы 5.12.

№ п/п	Статьи затрат	Итого, руб
1.2	Дополнительная заработная плата разработчиков	836972,8
1.3	Отчисления от основной и дополнительной заработной платы	3130278,272
2.	**Затраты машинного времени ($З_{мв}$)**	4615116,365
2.1	Стоимость машино-часа	9227,28
	Затраты на заработную плату обслуживающего персонала	8617593,6
	Годовые затраты на аренду помещения	3240000
	Сумма годовых амортизационных отчислений	2283120
	Стоимость электроэнергии, потребляемой за год	296768,48
	Действительный годовой фонд времени работы ПЭВМ	1672,8
	Затраты на материалы	90720
	Затраты на текущий и профилактический ремонт	453600
	Прочие затраты, связанные с эксплуатацией ЭВМ	453600
2.2	Машинное время ЭВМ	500,16
3.	**Затраты на материалы ($З_{мат}$)**	90720
4.	**Производственная себестоимость ($З_{общ.пр}$)**	16952095,43
5.	**Коммерческие (непроизводственные) затраты ($З_{непр}$)**	847604,77
6.	**Полная себестоимость ($З_{р}$)**	17799700,20

5.6 Расчет договорной (отпускной) цены разрабатываемого программного продукта

5.6.1 Расчет оптовой цены программного продукта.

Оптовая цена ($Ц_{опт}$) определяется следующим образом:

$$Ц_{опт} = С(З_р) + П_р, \quad (5.29)$$

$$П_р = \frac{С(З_р) \times У_р}{100}, \quad (5.30)$$

где $С(З_р)$ – себестоимость ПО, руб.;

$П_р$ – прибыль от реализации ПП, руб.;

$У_р$ – уровень рентабельности ($У_р = 30\%$),

$$П_р = \frac{17799700,20 \times 30}{100} = 5339910,06 \text{ руб.},$$

$$Ц_{опт} = 17799700,20 + 5339910,06 = 23139610,26 \text{ руб.}$$

5.6.2 Расчет отпускной цены программного продукта.

Прогнозируемая отпускная цена ПП берется как оптовая цена без НДС:

$$Ц_{отп} = 23139610,26 \text{ руб.}$$

Налог на добавленную стоимость (НДС) ($Р_{ндс}$) рассчитывается по формуле:

$$Р_{ндс} = \left(С(З_р) + П_р\right) \times \frac{Н_{ндс}}{100}, \quad (5.31)$$

где $Н_{ндс}$ – ставка налога на добавленную стоимость ($Н_{ндс} = 20\%$),

$$P_{ндс} = 23139610,26 \times \frac{20}{100} = 4627922,05 \text{ руб.}$$

Прогнозируемая отпускная цена ПО с НДС рассчитывается как:

$$Ц_{отп.ндс} = С(З_р) + П_р + Р_{ндс}. \quad (5.32)$$

$$Ц_{отп.ндс} = 17799700,20 + 5339910,06 + 4627922,05 = 27767532,31 \text{ руб.}$$

5.6.3 Расчет розничной цены программного продукта.

Розничная цена на ПП ($Ц_{розн}$) определяется следующим образом:

$$Ц_{розн} = Ц_{отп.ндс} + Т_н, \quad (5.33)$$

где $Т_н$ – торговая наценка при реализации программного обеспечения через специализированные магазины (торговых посредников), ее значение принимается в размере 10-20% от отпускной цены с НДС ($Т_н = 10\%$),

$$Ц_{розн} = 27767532,31 \times 1,1 = 30544285,54 \text{ руб.}$$

Полученные результаты представлены в таблице 5.13.

Таблица 5.13 – Плановая калькуляция разработки программного продукта

№	Статья наименование статьи расходов	Усл. обозн.	Значение, руб.
1.	**Затраты на оплату труда разработчиков**	**$З_{тр}$**	**12336979,07**
1.1	Основная заработная плата разработчиков		8369728
1.2	Дополнительная заработная плата разработчиков		836972,8
1.3	Отчисления от основной и дополнительной заработной платы		3130278,272
2.	Затраты машинного времени	$З_{м.в}$	4615116,365
3.	Затраты на материалы	$З_{мат}$	90720
4.	**Производственная себестоимость**		**16952095,43**
5.	Непроизводственных (коммерческих) затрат	$З_{непр}$	847604,77
6.	**Полная себестоимость (суммарные затраты на разработку ПО)**	**$З_р$**	**17799700,20**
7.	Прибыль от реализации ПО	$П_р$	5339910,06
8.	**Отпускная цена ПО без НДС**	**$Ц_{отп}$**	**23139610,26**
9.	Налог на добавленную стоимость	$Р_{ндс}$	4627922,05
10.	**Отпускная цена ПО с НДС**	**$Ц_{отп.ндс}$**	**27767532,31**
11.	Торговая наценка	$Т_н$	**2776753,23**
12.	**Розничная цена ПО**	**$Ц_{розн}$**	**30544285,54**

5.6.4 Расчет цены на программный продукт при условии копирования.

Для изготовления и реализации нескольких копий ПП его минимальную стоимость ($Ц'_{опт}$) можно определить по формуле:

$$Ц'_{опт} = \left(\frac{З_р}{N} + З_{коп}\right) \times \left(1 + \frac{У_р}{100}\right). \quad (5.34)$$

Затраты на копирование рассчитываются по следующей формуле:

$$З_{коп} = \frac{(Т_{коп} + Т_{под}) \times С_ч}{60} + Ц_н + З_{док}, \quad (5.35)$$

где $Т_{коп}$ – время копирования (2 мин.);

$Т_{под}$ – время подготовки носителя (1 мин.);

$Ц_н$ – розничная цена носителя (5000 руб.);

$З_{док}$ – затраты на копирование или печать документации и других сопровождающих материалов (10000 руб.),

$$З_{коп} = \frac{(1+2) \times 9227{,}28}{60} + 5000 + 10000 = 15461{,}364 \text{ руб.}$$

Результаты расчетов стоимости с учетом копирования представлены в виде таблице 5.14 и отображены в виде графика на рисунке 5.1.

Таблица 5.14 – Зависимость между минимальной ценой и числом реализуемых копий ПО

Число копий, шт.	Оптовая цена ПП (цена создания), руб.	Отпускная цена с НДС (цена реализации), руб.	Розничная цена, руб.
1	23159710,02	30107623,03	33118385,34
5	4648021,82	6042428,37	6646671,2
10	2334060,79	3034279,03	3337706,94
50	482891,97	627759,57	690535,52
100	251495,87	326944,63	359639,09

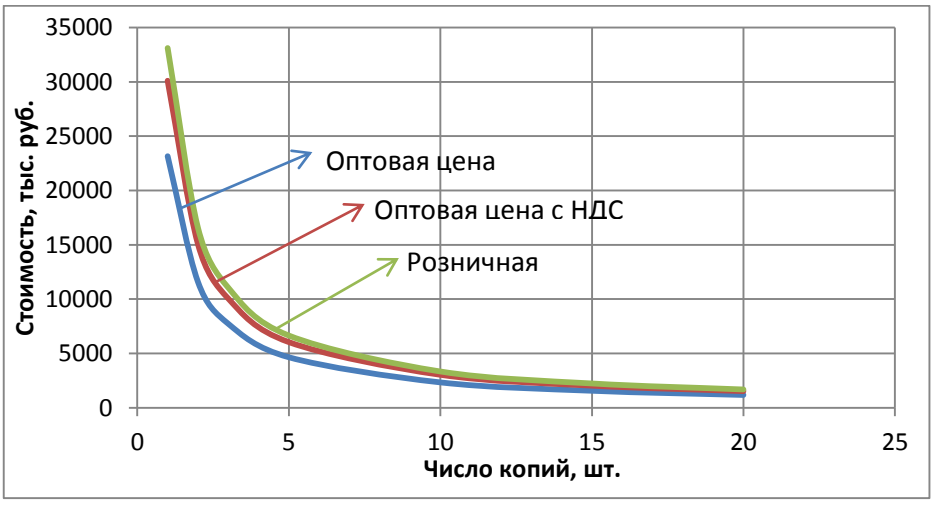


Рисунок 5.1 – График изменения стоимости ПП от количества копий

5.7 Определение экономической эффективности разработки программного продукта

Произведем статическую (простую) оценку продукта, которую выполним на основе сравнения с базовым продуктом.

За базовый продукт примем программу Elcut (QuickField). Стоимость программы составляла 123120 рос. руб. в ценах 2010 года. Для определения цены на начало 2012 года воспользуемся индексами повышения цены.

Индекс повышения цены на научно-техническую продукцию на период 2010/2011 – 1,221, на период 2011/2012 – 2,54. Для получения итогового индекса повышения цены необходимо перемножить два индекса, в результате, итоговый индекс $K_{\text{инд}} = 3{,}04$.

Курс российского рубля принимается на период начала 2010 года $C = 98$ руб.

Рассчитаем цену базового продукта по формуле:

$$З_{\text{баз}} = З_{\text{баз.2010}} \times K_{\text{инд}} \times C, \qquad (5.36)$$

$$З_{\text{баз}} = 123120 \times 3{,}04 \times 98 = 36679910{,}4 \text{ руб.}$$

Эффект (прибыль) рассчитывается по формуле:

$$Э(П) = З_{\text{баз}} - З_{\text{нов}}, \qquad (5.37)$$

где за $З_{\text{нов}}$ примем значение полной себестоимости продукта, $З_{\text{нов}} = 17799700{,}2$ руб.,

$$Э(П) = 36679910{,}4 - 17799700{,}2 = 18880210{,}2 \text{ руб.}$$

На основе полученного показателя прибыли рассчитаем такие показатели как рентабельность и срок окупаемости.

Рентабельность затрат рассчитывается по формуле:

$$Р = \frac{Э(П)}{З(И)} \times 100\%, \qquad (5.38)$$

где $З(И)$ – полная себестоимость разработки программного продукта, $З(И) = 17799700{,}2$ руб.,

$$Р = \frac{18880210{,}2}{17799700{,}2} \times 100\% = 1{,}06 \times 100\% = 106\%.$$

Срок окупаемости служит для определения степени рисков реализации проекта и ликвидности инвестиций. Различают простой срок окупаемости и динамический. Простой срок окупаемости проекта – это период времени, по окончании которого чистый объем поступлений (доходов) перекрывает объем инвестиций (расходов) в проект, и соответствует периоду, при котором накопительное значение чистого потока наличности изменяется с отрицательного на положительное:

$$Т_{\text{пр}} = \frac{З(И)}{Э(П)}, \qquad (5.39)$$

$$Т_{пр} = \frac{17799700,2}{18880210,2} = 0,94 \text{ года или } 11,3 \text{ месяца.}$$

Годовой экономический эффект определятся по формуле:

$$ГЭЭ = Э(П) - Р_{баз} \times З(И), \quad (5.40)$$

где $Р_{баз}$ – рентабельность затрат базового варианта ($Р_{баз} = 30\%$),

$$ГЭЭ = 18880210,2 - 0,3 \times 17799700,2 = 13540300,14 \text{ руб.}$$

Таким образом, можно отметить, что срок окупаемости продукта не велик и составляет менее года. Учитывая то, что продукт будет развиваться и набирать функционал в области постановки исходной задачи, это неплохие показатели. Рентабельность продукта положительная величина.

После выполнения расчетов, все полученные результаты представляются в виде итоговой таблицы, которая содержит сведения о рассчитанных показателях и их величины (таблица 5.15).

Таблица 5.15 – Технико-экономические показатели проекта

№ п/п	Наименование показателя	Ед. изм.	Базовый вариант	Проектный вариант
Показатели конкурентоспособности				
1.	Интегральный коэффициент конкурентоспособности	-	×	1,13
1.1	Коэффициент функциональных возможностей	-	×	1,13
1.2	Коэффициент соответствия нормативам	-	×	1
1.3	Коэффициент цены потребления	-	×	1
Показатели затрат на разработку				
2.	Общая трудоемкость разработки ПО	чел.-дн.	×	137,66
3.	Затраты на разработку программы	руб.	×	17799700,2
3.1	Затраты на оплату труда разработчиков	руб.	×	12336979,07
3.2	Затраты машинного времени	ч.	×	500,16
3.3	Затраты на материалы	руб.	×	90720
3.4	Общепроизводственные затраты	руб.	×	16952095,43
3.5	Непроизводственных (коммерческих) затрат	руб.	×	847604,77
Показатели экономической эффективности				
4.	Рентабельность затрат	%	×	106
5.	Простой срок окупаемости проекта	лет	×	0,94
6.	Годовой экономический эффект	руб.	×	13540300,14

6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ

6.1 Требования безопасности, производственной санитарии, пожаро- и взрывоопасности

6.1.1 Требования к персоналу, эксплуатирующему средства вычислительной техники и периферийное оборудование.

К самостоятельной эксплуатации электроаппаратуры допускается только специально обученный персонал не моложе 18 лет, пригодный по состоянию здоровья и квалификации к выполнению указанных работ.

Перед допуском к работе персонал должен пройти вводный и первичный инструктаж по технике безопасности с показом безопасных и рациональных приемов работы. Затем не реже одного раза в 6 месяцев проводится повторный инструктаж, возможно, с группой сотрудников одинаковой профессии в составе не более 20 человек. Внеплановый инструктаж проводится при изменении правил по охране труда, при обнаружении нарушений персоналом инструкции по технике безопасности, изменении характера работы персонала.

В помещениях, в которых постоянно эксплуатируется электрооборудование должны быть вывешены в доступном для персонала месте “Инструкции по технике безопасности”, в которых также должны быть определены действия персонала в случае возникновения аварий, пожаров, электротравм.

Руководители структурных подразделений несут ответственность за организацию правильной и безопасной эксплуатации средств вычислительной техники и периферийного оборудования, эффективность их использования; осуществляют контроль за выполнением персоналом требований настоящей инструкции по технике безопасности.

Требования безопасности перед началом работы, а также во время работы и после ее завершения регламентируются типовой межотраслевой инструкцией [76].

6.1.2 Требования безопасности во время работы.

Для снижения или предотвращения влияния опасных и вредных факторов необходимо соблюдать [76, 77].

Во избежание повреждения изоляции проводов и возникновения коротких замыканий не разрешается: вешать что-либо на провода, закрашивать и белить шнуры и провода, закладывать провода и шнуры за газовые и водопроводные трубы, за батареи отопительной системы, выдергивать штепсельную вилку из розетки за шнур, усилие должно быть приложено к корпусу вилки.

Для исключения поражения электрическим током запрещается: часто включать и выключать компьютер без необходимости, прикасаться к экрану и к тыльной стороне блоков компьютера, работать на средствах вычислительной техники и периферийном оборудовании мокрыми руками, работать на средствах вычислительной техники и периферийном оборудовании, имеющих нарушения

целостности корпуса, нарушения изоляции проводов, неисправную индикацию включения питания, с признаками электрического напряжения на корпусе, класть на средства вычислительной техники и периферийное оборудование посторонние предметы.

Запрещается под напряжением очищать от пыли и загрязнения электрооборудование.

Запрещается проверять работоспособность электрооборудования в неприспособленных для эксплуатации помещениях с токопроводящими полами, сырых, не позволяющих заземлить доступные металлические части.

Недопустимо под напряжением проводить ремонт средств вычислительной техники и периферийного оборудования. Ремонт электроаппаратуры производится только специалистами-техниками с соблюдением необходимых технических требований.

Во избежание поражения электрическим током, при пользовании электроприборами нельзя касаться одновременно каких-либо трубопроводов, батарей отопления, металлических конструкций, соединенных с землей.

При пользовании электроэнергией в сырых помещениях соблюдать особую осторожность.

6.1.3 Требования безопасности в аварийных ситуациях.

При обнаружении неисправности немедленно обесточить электрооборудование, оповестить администрацию. Продолжение работы возможно только после устранения неисправности.

При обнаружении оборвавшегося провода необходимо немедленно сообщить об этом администрации, принять меры по исключению контакта с ним людей. Прикосновение к проводу опасно для жизни.

Во всех случаях поражения человека электрическим током немедленно вызывают врача. До прибытия врача нужно, не теряя времени, приступить к оказанию первой помощи пострадавшему.

Необходимо немедленно начать производить искусственное дыхание, наиболее эффективным из которых является метод “рот в рот” или “рот в нос”, а также наружный массаж сердца.

Искусственное дыхание пораженному электрическим током производится вплоть до прибытия врача.

6.1.4 Требования безопасности по окончании работы [76].

После окончания работы необходимо обесточить все средства вычислительной техники и периферийное оборудование. В случае непрерывного производственного процесса необходимо оставить включенными только необходимое оборудование.

6.2 Опасные и вредные факторы при работе с ПЭВМ

Эксплуатирующий средства вычислительной техники и периферийное оборудование персонал может подвергаться опасным и вредным

воздействия, которые по природе действия подразделяются на следующие группы:

- поражение электрическим током;
- механические повреждения;
- электромагнитное излучение;
- инфракрасное излучение;
- опасность пожара;
- повышенный уровень шума и вибрации.

Для снижения или предотвращения влияния опасных и вредных факторов необходимо соблюдать “Санитарные правила и нормы. Гигиенические требования к видеодисплейным терминалам, персональным электронно-вычислительным машинам и организации работы” [77].

6.3 Требования по обеспечению пожарной безопасности

На рабочем месте запрещается иметь огнеопасные вещества [78].

В помещениях запрещается:

- зажигать огонь;
- включать электрооборудование, если в помещении пахнет газом;
- курить;
- сушить что-либо на отопительных приборах;
- закрывать вентиляционные отверстия в электроаппаратуре.

Источниками воспламенения являются:

- искра при разряде статического электричества;
- искры от электрооборудования;
- искры от удара и трения;
- открытое пламя.

При возникновении пожароопасной ситуации или пожара персонал должен немедленно принять необходимые меры для его ликвидации, одновременно оповестить о пожаре администрацию.

Помещения с электрооборудованием должны быть оснащены огнетушителями типа ОУ-2 или ОУБ-3.

6.4 Требования к производственной санитарии

6.4.1 Гигиенические требования [79] к ПЭВМ и ВДТ заключаются в следующем:

– ПЭВМ и ВДТ должны иметь гигиенический сертификат соответствия требованиям экологических стандартов безопасности, установленным в нормативных документах, например, Energy Star, Nutek, TCO 1992, VESA DPMS (энергоснабжение), ISO 924, MPR-II, TCO 2000 (защита от излучений), FCC класс B (радиочастотные помехи) и т.д.;

– при работе с ПЭВМ и ВДТ необходимо обеспечить наилучшие значения визуальных параметров в пределах оптимального диапазона, указанного в таблице 6.1;

Таблица 6.1 – Допустимые визуальные параметры устройств отображения информации

Параметры	Допустимые значения
Яркость белого поля	Не менее 35 кд/м^2
Неравномерность яркости рабочего поля	Не более ±20%
Контрастность (для монохромного режима)	Не менее 3:1
Временная нестабильность изображения (непреднамеренное изменение во времени яркости изображения на экране дисплея)	Не должна фиксироваться

– допустимые параметры неионизирующих электромагнитных полей (ЭМП) и ионизирующих излучений при работе ПЭВМ и ВДТ должны удовлетворять требованиям таблицы 7 [80]. При больших значениях этих излучений следует применять экранные фильтры. Фильтрами полной защиты пользователей являются фильтры Ergostat, UNUS и UMAX MP – 196, а также отечественные фильтры “Русский щит” и Dehender Ergan.

6.4.2 Санитарно-гигиенические требования к помещениям для эксплуатации ПЭВМ и ВДТ согласно [77, 80] следующие:

– запрещено располагать рабочие места с ПЭВМ и ВДТ в подвальных помещениях;

– пол помещения должен быть ровный, с антистатическим покрытием;

– отделка помещения полимерными материалами нежелательна;

– расстояние между боковыми поверхностями мониторов должно быть не менее 1,2 м;

– требуемые параметры *микроклимата* для помещений с ПЭВМ указаны в таблице 6.2;

Таблица 6.2 – Параметры микроклимата для помещений с ПЭВМ

Период года	Параметр микроклимата	Величина
Холодный и переходный	Температура воздуха в помещении	22 – 24 °С
	Относительная влажность	40 – 60 %
	Скорость движения воздуха	до 0,1 м/с
Теплый	Температура воздуха в помещении	23 – 25°С
	Относительная влажность	40 – 60 %
	Скорость движения воздуха	0,1 – 0,2 м/с

– *освещенность* на поверхности стола в зоне размещения рабочего документа должна быть 300 – 500 лк. Освещенность поверхности экрана не должна быть более 300 лк;

– помещения с компьютерами должны иметь *площадь* не менее 6 м^2 на одного работающего. При использовании монитора с жидкокристаллическими мониторами, или работе на компьютере не более 4 часов в день допускается площадь не менее 4,5 м^2.

6.4.3 Организация и оборудование рабочих мест с ВДТ и ПЭВМ для различных категорий пользователей [80]:

– рабочее место должно располагаться так, чтобы естественный свет падал сбоку, преимущественно слева;

– окна в помещениях с ВДТ и ПЭВМ должны быть оборудованы регулируемыми устройствами (жалюзи, занавески, внешние козырьки и т.д.);

– расстояние между рабочими столами с видеомониторами должны быть не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м;

– монитор, клавиатура и корпус компьютера должны находиться прямо перед оператором; высота рабочего стола с клавиатурой должна составлять 680 – 800 мм над уровнем стола; а высота экрана (над полом) – 900–1280 см;

– монитор должен находиться от оператора на расстоянии 60 – 70 см на 20 градусов ниже уровня глаз;

– пространство для ног должно быть: высотой не менее 600 мм, шириной не менее 500 мм, глубиной не менее 450 мм. Должна быть предусмотрена подставка для ног работающего шириной не менее 300 мм с регулировкой угла наклона. Ноги при этом должны быть согнуты под прямым углом. Рабочее место с ВДТ должно иметь легко перемещаемые пюпитры для документов.

6.4.4 Гигиенические критерии оценки тяжести и напряженности трудового процесса пользователей ПЭВМ.

Организация работы с ПЭВМ осуществляется в зависимости от вида и категории трудовой деятельности. Виды трудовой деятельности разделяются на 3 группы: группа А – работа по считыванию информации с экрана ВДТ с предварительным запросом; группа Б – работа по вводу информации; группа В – творческая работа в режиме диалога с ПЭВМ. При выполнении в течение рабочей смены работ, относящихся к разным видам трудовой деятельности, за основную работу с ПЭВМ следует принимать такую, которая занимает не менее 50 % времени в течение рабочей смены или рабочего дня.

Для видов трудовой деятельности устанавливается 3 категории тяжести и напряженности работы с ПЭВМ, которые определяются: для группы А – по суммарному числу считываемых знаков за рабочую смену, но не более 60000 знаков за смену; для группы Б – по суммарному числу считываемых

или вводимых знаков за рабочую смену, но не более 40000 знаков за смену; для группы В – по суммарному времени непосредственной работы с ПЭВМ за рабочую смену, но не более 6 ч за смену.

В зависимости от категории трудовой деятельности и уровня нагрузки за рабочую смену при работе с ПЭВМ устанавливается суммарное время регламентированных перерывов, которое указано в таблице 6.3.

В случаях, когда характер работы требует постоянного взаимодействия с ВДТ (набор текстов или ввод данных и т. п.) с напряжением внимания и сосредоточенности, при исключении возможности периодического переключения на другие виды трудовой деятельности, не связанные с ПЭВМ, рекомендуется организация перерывов на 10-15 мин через каждые 45-60 мин работы. Продолжительность непрерывной работы с ВДТ без регламентированного перерыва не должна превышать 1 ч. Ежедневная работа высокой интенсивности и с нервно-эмоциональным напряжением по 12 и более часов не допускается.

Таблица 6.3 – Суммарное время регламентированных перерывов в зависимости от продолжительности работы, вида и категории трудовой деятельности с ПЭВМ

Категория работы с ПЭВМ	Уровень нагрузки за рабочую смену при видах работ с ПЭВМ			Суммарное время регламентиро-ванных перерывов, мин	
	группа А, кол-во знаков	группа Б, кол-во знаков	группа В, ч	при 8-часовой смене	при 12-часовой смене
I	до 20000	до 15000	до 2	50	80
II	до 40000	до 30000	до 4	70	110
III	до 60000	до 40000	до 6	90	140

6.5 Шум

Одним из вредных производственных факторов является шум – беспорядочное сочетание звуков различной частоты и интенсивности (силы), возникающих при механических колебаниях в твердых, жидких и газообразных средах. Шум отрицательно влияет на организм человека, в первую очередь на его центральную нервную и сердечно-сосудистую системы. Длительное воздействие шума снижает остроту слуха и зрения, повышает кровяное давление, утомляет центральную нервную систему, в результате чего ослабляется внимание, увеличивается количество ошибок в действиях работающего, снижается производительность труда. Воздействие шума приводит к появлению профессиональных заболеваний и может явиться также причиной несчастного случая.

Источниками производственного шума являются машины, оборудование и инструмент.

Органы слуха человека воспринимают звуковые волны с частотой от 16 до 20 000 Гц. Колебания с частотой ниже 20 Гц (инфразвук) и выше 20 000 Гц (ультразвук) не вызывают слуховых ощущений, но оказывают биологическое воздействие на организм.

При звуковых колебаниях частиц среды в ней возникает переменное давление, которое называют звуковым давлением P.

Распространение звуковых волн сопровождается переносом энергии, величина которой определяется интенсивностью звука I. Минимальное звуковое давление P_0 и минимальная интенсивность звука I_0, различаемые ухом человека, называются пороговыми. Интенсивности едва слышимых звуков (порог слышимости) и интенсивность звуков, вызывающих болевые ощущения (болевой порог), отличаются друг от друга более чем в миллион раз. Поэтому для оценки шума удобно измерять не абсолютные значения интенсивности и звукового давления, а относительные их уровни в логарифмических единицах, взятые по отношению к пороговым значениям P_0 и I

За единицу измерения уровней звукового давления и интенсивности звука принят децибел (дБ). Диапазон звуков, воспринимаемых органом слуха человека, от 0 до 140 дБ.

Звуковые колебания различных частот при одинаковых уровнях звукового давления по-разному воздействуют на органы слуха человека. Наиболее благоприятно воздействие звуков более высоких частот.

По частоте шумы подразделяются на низкочастотные (максимум звукового давления в диапазоне частот ниже 400 Гц), среднечастотные (400 – 1000 Гц) и высокочастотные (свыше 1000 Гц).

Для определения частотной характеристики шума звуковой диапазон по частоте разбивают на октавные полосы частот, где верхняя граничная частота равна удвоенной нижней частоте.

По характеру спектра шум подразделяется на широкополосный с непрерывным спектром шириной более одной октавы и тональный, в спектре которого имеются выраженные дискретные тона.

По временным характеристикам шум подразделяется на постоянный и непостоянный (колеблющийся во времени, прерывистый, импульсный).

Нормирование шума и его параметры определены в ГОСТ 12.1.003 [81] и СанПиН 2.2.4/2.1.8.10-32-2002 [82].

Постоянным считается шум, уровень которого за восьмичасовой рабочий день изменяется во времени не более чем на 5 дБ, непостоянным – более чем на 5 дБ. ГОСТ 12.1.003–83 [81] устанавливает предельно-допустимые условия постоянного шума на рабочих местах, при которых шум, действуя на работающего в течение восьмичасового рабочего дня, не приносит вреда здоровью. Нормирование ведется в октавных полосах частот со среднегеометрическими частотами 63, 125, 250, 500, 1000, 2000, 4000, 8000 Гц.

Для измерения уровней шума на рабочих местах в октавных полосах частот и общего уровня шума применяют различные типы шумоизмерительной аппаратуры. Наибольшее распространение получили шумомеры, состоящие из микрофона, воспринимающего звуковую энергию и преобразующего ее в электрические сигналы, усилителя, корректирующих фильтров, детектора и стрелочного индикатора со шкалой, градуированной в децибелах.

Производственный шум нарушает информационные связи, что вызывает снижение не только эффективности, но и безопасности деятельности человека, так как высокий уровень шума мешает услышать предупреждающий сигнал опасности. Кроме того, шум вызывает обычную усталость. При действии шума снижаются способность сосредоточения внимания, точность выполнения работ, связанных с приемом и анализом информации, и производительность труда. При постоянном воздействии шума работники жалуются на бессонницу, нарушение зрения, вкусовых ощущений, расстройство органов пищеварения и т. д. У них отмечается повышенная склонность к неврозам. Энергозатраты организма при выполнении работы в условиях шума больше, т. е. работа оказывается более тяжелой. Шум, отрицательно воздействуя на слух человека, может вызвать три возможных исхода: временно (от минуты до нескольких месяцев) снизить чувствительность к звукам определенных частот, вызвать повреждение органов слуха или мгновенную глухоту. Уровень звука в 130 дБ вызывает болевое ощущение, а в 150 дБ приводит к поражению слуха при любой частоте. Предельно допустимые уровни (ПДУ) действия шума на человека гарантируют, что остаточное понижение слуха после 50 лет работы у 90 % работающих будет менее 20 дБ, т. е. ниже того предела, когда это начинает мешать человеку в повседневной жизни. Потеря слуха на 10 дБ практически не замечается.

6.6 Расчет шума

Человек ощущает звук в широком диапазоне звуковых давлений $p_{зв}$ (интенсивностей I).

Стандартным порогом слышимости называют эффективное значение звукового давления (интенсивности), создаваемого гармоническим колебанием с частотой f = 1000 Гц, едва слышимым человеком со средней чувствительностью слуха.

Стандартному порогу слышимости соответствует звуковое давление $p_o=2\cdot10^{-5}$ Па или интенсивность звука $I_o=10^{-12}$ Вт/м^2. Верхний предел звуковых давлений, ощущаемых слуховым аппаратом человека, ограничивается болевым ощущением и принят равным p_{max} = 20 Па и I_{max}= 1 Вт/м^2.

Величина слухового ощущения Λ при превышении звуковым давлением $p_{зв}$ стандартного порога слышимости определяется по закону психофизики Вебера - Фехнера:

$$\Lambda = q \lg\left(\frac{p_{зв}}{p_0}\right), \qquad (6.1)$$

где q – некоторая постоянная, зависящая от условий проведения эксперимента.

С учетом психофизического восприятия звука человеком для характеристики значений звукового давления $p_{зв}$ и интенсивности I были введены логарифмические величины – уровни L (с соответствующим индексом), выраженные в безразмерных единицах – децибелах, дБ, названных в честь Грейма – Бела (увеличение интенсивности звука в 10 раз соответствует 1 Белу (Б) – 1Б = 10 дБ):

$$L_p = 10 \lg\left(\frac{p}{p_0}\right)^2 = 20 \lg\left(\frac{p}{p_0}\right), \qquad (6.2)$$

$$L_I = 10 \lg\left(\frac{I}{I_0}\right). \qquad (6.3)$$

Следует отметить, что при нормальных атмосферных условиях $L_p = L_I$.

По аналогии были введены также и уровни звуковой мощности

$$L_W = 10 \lg\left(\frac{W}{W_0}\right), \qquad (6.4)$$

где $W_0 = I_0 * S_0$ = 10-12Вт – пороговая звуковая мощность на частоте 1000 Гц, S_0 = 1 м^2.

Безразмерные величины L_p, L_I, L_w достаточно просто измеряются приборами, поэтому их полезно использовать для определения абсолютных значений p, I, W по обратным к (6.2) и (6.3) зависимостям

$$p^2 = p_0^2 \cdot 10^{0.1 \cdot L_p}, \qquad (6.5)$$

$$I = I_0 \cdot 10^{0.1 \cdot L_p}, \qquad (6.6)$$

$$W = W_0 \cdot 10^{0.1 \cdot L_W}. \qquad (6.7)$$

Уровень суммы нескольких величин определяется по их уровням L_i, i = 1, 2, ..., n соотношениемо

$$L_\Sigma = 10 \lg \sum_{i=1}^{n} 10^{0.1 \cdot L_i}, \qquad (6.8)$$

где n – количество складываемых величин.

Если складываемые уровни одинаковы ($L_i = L$), то

$$L_\Sigma = L + 10 \lg n. \qquad (6.9)$$

6.6.1 Пример решения задачи.

Определить общий уровень шума от n агрегатов с заданными характеристиками уровня звукового давления L_i при соответствующей частоте спектра (f = 250 Гц). Сделать вывод о допустимом уровне шума по ГОСТ 12.1.003 [82] для помещений для преподавания и разработчиков.

Источники: $L_1 = 50$ дБ, $L_1 = 45$ дБ, $L_1 = 40$ дБ.

Определим суммарный уровень шума от 3 агрегатов:

$$L = 10\lg\left(10^{0.1\cdot 50} + 10^{0.1\cdot 45} + 10^{0.1\cdot 40}\right) = 51.5 \text{ дБ}.$$

Допустимый уровень шума для частоты f = 250 Гц установлен в 54 дБ. Полученный уровень шума укладывается в допустимое значение, следовательно шум в помещении с тремя аппаратами с указанными характеристиками позволяет работать в помещении установленный рабочий срок. При превышении уровня шума над допустимым, можно использовать акустические экраны и другие средства для шумоизоляции.

7 ЭНЕРГО- И РЕСУРСОСБЕРЕЖЕНИЕ

Ресурсосбережение – совокупность мер по бережливому и эффективному использованию фактов производства (капитала, земли, труда). Иначе можно сказать, что ресурсосбережение это деятельность (организационная, экономическая, техническая, научная, практическая, информационная), а также комплекс организационно-технических мер и мероприятий, сопровождающих все стадии жизненного цикла объектов и направленных на рациональное использование и экономное расходование ресурсов.

Ресурсы – ценности, запасы, возможности, источники дохода в государственном бюджете. В общем виде ресурсы делятся на природные и экономические (материальные, трудовые, финансовые).

Ресурсосбережение обеспечивается посредством:

– использования ресурсосберегающих и энергосберегающих технологий;
– снижения фондоемкости и материалоемкости продукции;
– повышения производительности труда;
– сокращения трудовых и материальных затрат;
– повышения качества продукции;
– рационального применения труда менеджеров и маркетологов;
– использования выгод международного разделения труда;

Ресурсосбережение способствует росту эффективности экономики, повышению ее конкурентоспособности.

Энергосбережение – реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на вовлечение в хозяйственный оборот возобновляемых источников энергии.

Энергосбережение – одна из наиболее приоритетных задач экономики. Без радикального снижения затрат топлива на выработку электрической энергии и теплоты и затрат энергоресурсов на единицу выпускаемой промышленной продукции невозможно добиться экономического благополучия и конкурентоспособности. Региональные энергетические компании практически полностью определяют политику энергосбережения в производстве и транспорте электрической и тепловой энергии, а также в значительной мере влияют на энергетическую эффективность потребления энергии.

Получение новых материалов, их синтез, в частности нанокомпозитов на основе частиц восстановленных металлов, является чрезвычайно сложным и дорогостоящим процессом, связанным с использованием различного сложного оборудования, а также дорогостоящими ресурсами. Для частиц используются такие металлы как золото, платина, серебро и др.

Существующие методы получения новых предметов и материалов основаны на методиках проектирования “сверху-вниз” и “снизу-верх”. Метод “сверху-вниз” основан на миниатюризации существующих предметов, путем

уменьшения техпроцесса и соответствующих элементов изделия. Проектирование “снизу-вверх” основано на составлении цепочек из атомов в определенном порядке. Это наиболее современное направление, которое позволяет конструировать как бы из отдельных кубиков нужные материалы и системы с заданными свойствами. Создание материалов с необходимыми свойствами принесет ощутимые выгоды в экономии энергетических и материальных ресурсов.

Среди наиболее популярных направлений в области синтеза материалов является получение высококремнеземных стекол, которые синтезируются золь-гель методом. Для улучшения оптических характеристик, а также для их применение в новых сферах используются нанотехнологии, позволяющие внедрять в стекло частицы восстановленных металлов, меняя при этом и исходные свойства стекол, а также получая новые. Подобные нововведения позволяют применять оптические элементы в космических отраслях, военной технике, электронике и т.д.

Золь-гель метод является весьма перспективным методом [1], разработанным для оксидной керамики. Полученная керамика приобретает свойства, подобные монокристаллам. Процесс включает в себя приготовление растворов алкоксидов, их каталитическое взаимодействие с последующим гидролизом, после чего следует конденсационная полимеризация с дальнейшим гидролизом. В результате обработки образуется оксидный полимер (гель), который затем подвергается старению, промывке, сушке и термообработке. Данный метод характеризуется высокой частотой синтезированных соединений (99,95%), однородностью и разнообразием получаемых компактных материалов. В основном таким методом получают субмикронные порошки, а также порошки соединений $SrZrO_3$ и $SrTiO_3$ с размером частиц 5-10 нм, ZrO_2, ZrO_2-Y_2O_3 с размером кристаллов 8-10 нм.

Размещение частиц металлов сопряжено с трудностями получения самих частиц, а также внедрением их в нужные места в синтезируемом материале. Обычно, разброс частиц случаен и в зависимости от применяемого оборудования и технологии может отличаться, как и свойства получаемых материалов.

Таким образом, видно, что процесс является очень затратным и требовательным к энергии, ресурсам и оборудованию. Применение компьютерного моделирования позволяет сократить расходы на изготовление пробных образов, а также сократить затраты на подбор оптимальных параметров для синтезируемых материалов с заданными характеристиками. Моделирование обеспечивает большую гибкость при синтезе, а также позволяет сократить время на подбор оптимальных форм частиц и их материалов. Кроме того, оно позволяет повысить скорость обработки большого количества образцов и выбора из них наиболее оптимального.

Основные преимущества при использовании компьютерного моделирования для синтеза новых материалов:

– сокращение затрат на материалы, ресурсы;

– сведение к минимуму количество изготавливаемых опытных образцов с заданными характеристиками, получаемых без предварительной оценки, которую можно узнать при помощи моделирования;

– экономия редких и дорогостоящих металлов (из которых изготавливают как частицы, так и части оборудования);

– сокращение времени на проектирование и подбор параметров для требуемых характеристик;

– облегчение труда исследователям, работа которых производится в безопасных условиях и в контролируемой обстановке;

– возможность варьирования большим числом параметров в моделях;

– возможность автоматического подбора частиц (элементов) материалов из каталога строительных элементов (сборка конструктора).

Основные ограничения при использовании компьютерного моделирования накладываются доступными вычислительными ресурсами. При необходимости и правильном выборе платформ и технологий их можно гибко масштабировать, оплачивая только фактически использованные ресурсы.

ЗАКЛЮЧЕНИЕ

Нанотехнологии все больше проникают во все сферы. Процесс их развития идет большими темпами. Сегодня стоят задачи синтеза новых материалов с заданными свойствами и эксплуатационными параметрами. Для этого применяют компьютерное моделирование распределения электромагнитного поля (ЭМП).

Исследование ЭМП основано на решении уравнений Максвелла. Для их решения существует достаточное количество методов, из которых были рассмотрены FDTD, DDA, Т-матрицы, FEM, а также существующее ПО, которое их использует.

Для компьютерного моделирования распределения электромагнитных полей (ЭМП) в нанокомпозитах были сформулировать требования, предъявляемые к программному продукту:

– возможность точного описания модели системы, наиболее близко описывающей ее реальную структуру,

– адекватность результатов расчетов по модели с реальной системой,

– универсальный язык описания моделей (метаязык),

– возможность поддержки программного кода, а также внесение изменений в используемые модели.

В результате выполнения работы было выполнено следующее:

– разработан формат для описания задач (Input API) и геометрии;

– разработаны программные модули для реализации интерфейса взаимодействия с программой Netgen;

– построена математическая модель для исследования распределения ЭМП с помощью векторного метода конечных элементов (ВКМЭ);

– спроектирован и разработан каркас приложения для решения задач электродинамики с помощью ВКМЭ;

– разработано ПО для компьютерного моделирования на основе построенной модели.

Продукт написан под платформу .NET Framework 4 на языке C# и состоит из нескольких модулей. Программа получила название ElectroMagnetics Field FEM Modeler (EMFFM).

На разработанный продукт и составляющие его модули была написана документация (приложения Г – И), а также руководства программиста и пользователя (приложения К – М). Исходные тексты программы прилагаются к работе в приложении В.

Произведена технико-экономическая оценка, которая показывает эффективность проведенной работы и основные показатели разработанного ПО.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ковшов А. Н. Основы нанотехнологии в технике.: Учебное пособие / А. Н. Ковшов, Ю. Ф. Назаров, И. М. Ибрагимов. – М.: Издательство МГОУ, 2006. – 244 с.: ил.
2. Ибрагимов И. М. Основы компьютерного моделирования наносистем: Учебное пособие / И. М. Ибрагимов, А. Н. Ковшов, Ю. Ф. Назаров. – СПб.: Издательство “Лань”, 2010. – 384 с.: ил.
3. Климов В. В. Наноплазмоника / В. В. Климов. – М.: Физматлит, 2009. – 480 с.
4. Виноградов А. П. Электродинамика композитных материалов / А. П. Виноградов – М: УРСС Едиториал, 2004. – 505 с.
5. Jackson J. D. Classical electrodynamics / J. D. Jackson – New York: Wiley, 1998. – 808 p.
6. Ландау Л. Д. Электродинамика сплошных сред / Л. Д. Ландау, Е. М. Лифшиц – М.: Наука, 1982. – 656 с.
7. Lindell I. V. Electromagnetic waves in chiral and bi-isotropic media / I. V. Lindell, A. H. Sihvola, S. A. Tretyakov and A. J.Viitanen – London, Boston: Artech House, 1994.
8. Smith D. R. Metamaterials and negative reflection index. / Smith D. R., Pendry J. B., Wiltshire M. C. K. // Science. – 2004. – 205. – P.788-792.
9. Purcell E. M. Scattering and absorption of light by nonspherical dielectric grains / E. M. Purcell, C. R. Pennypacker // Astrophysical Journal. – 1973 – 186. – P. 705-714.
10. Collinge M.J. Discrete dipole approximation with polarizabilities that account for both finite wavelength and target geometry / Collinge M.J., Draine B. T. // arXiv: astro-ph/0311304 – Vol 1. – 2003.
11. Draine B. T. Discrete dipole approximation for scattering calculations. / Draine B.T., Flatau P.J. //J. Opt. Soc. Am. – 2004. – A 11 (4). – P.1491–1499.
12. Draine B. T. Beyond Clausius-Mossotti: wavy propagation on a polarizable point lattice and discrete dipole approximation / Draine B. T. and Goodman I. // Astrophys. J. – 1994. – 405. – P. 685-697.
13. Yang W.-H. Discrete dipole approximation for calculating extinction and Raman intensities for small particles with arbitrary shapes / Yang W.-H., George C. Schatz, and Richard P. Van Duyne // J. Chem. Phys. – 1995. – 103. – P. 869-875.
14. Jensen T. R. Electrodynamics of noble metal nanoparticles and nanoparticle clusters. / Jensen T. R., Kelly K. L., Lazarides A, Schatz G. C. // J. Cluster Sci. – 1999. – 10. – P. 295–317.
15. Cristensen D. A. Analysis of near field tip pattern including object interaction using finite-difference time-domain calculations / Cristensen D. A. // Ultramicroscopy. – 1995. – 57. – P.189-195.

16. Kann J.L. Linear behavior of a near-field optical scanning system / Kann J.L., Milster T.D., Froehlich F.F., Ziolkowski R.W., and Judkins J. B.Y. // J. Opt. Soc. Am. A. – 1995. – 12. – P. 1677-1682.

17. Furukawa H. Near-field optical microscope images of a dielectric flat substrate with subwavelength strips / Furukawa H., Kawata S.Y. // Opt. Commun. – 2001. – 196. – P. 93-102.

18. Furukawa H. Analysis of image formation in a near-field scanning optical microscope: Effects of multiple scattering. / Furukawa H., Kawata S. Y. // Opt. Commun. – 1996. – 132. – P. 170-178.

19. Kane Yee Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media / Kane Yee // IEEE Transactions on Antennas and Propagation. – 1966. – 14 (3). – P. 302–307.

20. Taflove A. Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd ed. / Taflove A., and Hagnes S.C. – Artech House Publishers, ISBN 1-58053-832-0. – 2005.

21. Sadiku M.N.O. Numerical Techniques in Electromagnetics, Second Edition / Sadiku M.N.O. – CRC Press, London, 2003.

22. Berenger J.P. An Effective PML for the Absorption of Evanescent Waves in Waveguides / Berenger J.P. // IEEE Microwave and guided wave letters. – 1988. – 8. – P. 188-190.

23. Young J.L. A summary and systematic analysis of FDTD algorithms for linearly dispersivemedia/ Young J.L., Nelson R.O. // IEEE Antennas Propagation Mag., 2001. – 43 (1). – P. 61-77.

24. Quinten E. M. Absorption and elastic scattering of light by particles aggregates / Quinten E. M., Kreibig U. // Appl, Opt. – 1993. – 32. – P. 6173-6182.

25. Zhao L.L. The extinction spectra of silver nanoparticle arrays: Influence of array structure on plasmon resonance wavelength and width / Zhao L.L., Kelly K.L., Schatz G.C. // J. Phys. Chem. B. – 2003. – 107. – P.7343-7350.

26. Waterman P. C. Matrix formulation of electromagnetic scattering / Waterman, P. C. // Proc. IEEE. – 1965. – Vol. 53. – P. 805–812.

27. Waterman P. C. Symmetry, unitarity, and geometry in electromgnetic scattering / P. C. Waterman // Physical Review. – 1973. – D 3(4). – P. 825–839.

28. Mishchenko M. I. T-matrix computations of light scattering by nonspherical particles: A review / M. I. Mishchenko, L. D. Travis, D. W. Mackowski // J. Quant. Spectrosc. Radiat. Transfer. 1996. – 55. – P. 535-575.

29. Calculation of total cross selection of multiple –sphere clusters / Mackowski D. W. // J. Opt. Soc, Am. A. – 1994. – 11. – P. 2851-2861.

30. Mishchenko M. I. Scattering, Absorption, and Emission of Light by Small Particles / M. I. Mishchenko, L. D. Travis, A. A. Lacis. – Cambridge University Press, Cambridge, 2003.

31. Courant, R. L. Variational methods for the solution of problems of equilibrium and vibration / R. L. Courant // Bull. Amer. Math. Soc. – 49 (1): 1 – 23. – 1943.

32. Silvester P. P. Finite element solution of homogeneous waveguide problems / P. P. Silvester // Alta Freq. – 1969. – 38. – P. 313 – 317.

33. Mei K. K. Unimoment method of solving antenna and scattering problems / K. K. Mei // IEEE Trans. Antennas Propagat. – 1974. – 22 (6). – P. 760 – 766.

34. Marin S. P. Computing scattering amplitudes for arbitrary cylinders under incident plane waves / Marin S. P. // IEEE Trans. Antennas Propagat. – 1982. – 30 (6). P. 1045 – 1049.

35. Nedelec J. C. Mixed finite elements in R 3 / Nedelec J. C. // Numer. Math. – 1980. – 35. – P. 315 – 341.

36. Bossavit A. A mixed FEM – BIEM method to solve 3 - D eddy current problems / A. Bossavit, J. C. Verite // IEEE Trans. Magn. – 1982. – 18 (2) – P. 431 – 435.

37. Batron M. L. New vector finite elements for three - dimensional magnetic field computation / M. L. Barton and Z. J. Cendes // J. Appl. Phys. – 1987. – 61. – P. 3919 – 3921.

38. Silvester P.P. Finite Elements for Electrical Engineers. 3rd edition. / P. P. Silvester and R. L. Ferrari. – Cambridge, UK : Cambridge University Press, 1996.

39. Jianming, J. The Finite Element Method in Electromagnetics. 2nd edition. / J. - M. Jin – New York : Wiley, 2002. – 780 p.

40. Jianming, J. Finite element analysis of antenna and arrays / J. Jianming – John Wiley & Sons, 2008. – 452 p.

41. Jianming, J. Theory And Computation Of Electromagnetic Fields / J. Jianming – John Wiley & Sons, 2010. – 616 p.

42. ELCUT программа моделирования электромагнитных, тепловых и механических задач [Электронный ресурс]. – Режим доступа: http://elcut.ru/. – Дата доступа: 20.05.2012.

43. Elmer. Open Source Finite Element Software for Multiphysical Problems [Электронный ресурс]. – Режим доступа: http://www.csc.fi/english/pages/elmer. Дата доступа: 21.05.2012.

44. Elmer Solver Manual [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.nic.funet.fi/pub/sci/physics/elmer/doc/ElmerSolverManual.pdf.

45. MEEP [Электронный ресурс]. – Режим доступа: http://ab-initio.mit.edu/wiki/index.php/Meep – Дата доступа: 20.05.2012.

46. MIT Photonic Bands (MPB) [Электронный ресурс]. – Режим доступа: http://ab-initio.mit.edu/wiki/index.php/MIT_Photonic_Bands. – Дата доступа: 20.05.2012.

47. Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis / Johnson S/, Joannopoulos J. // Optics Express. – 2001. – Vol. 8, Issue 2. – P. 173-190.

48. Википедия. Язык Scheme. [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/wiki/Scheme. – Дата доступа: 20.05.2012.

49. The EMAP Finite Element Modeling Codes (EMAP) [Электронный ресурс]. – Режим доступа http://www.cvel.clemson.edu/modeling/EMAG/EMAP/.

50. Maile G. L. Three-dimensional analysis of electromagnetic problems by finite element methods, Ph.D. Dissertation. / G. L. Maile – University of Cambridge, U.K., Dec. 1979.

51. Lynch D. R. Origin of vector parasites in numerical Maxwell solutions / D. R. Lynch, K. D. Paulsen // IEEE Trans. on Microwave Theory and Techniques. – Mar. 1991. – Vol. 39, no. 3. – P. 383-394.

52. Lynch D. R. Elimination of vector parasites in finite element Maxwell solutions / D. R. Lynch and K. D. Paulsen // IEEE Trans. on Microwave Theory and Techniques. – Mar. 1991. – Vol. 39, no. 3. P. 395-404.

53. Hubing T. H. EMC Applications of EMAP-2: A 3D Finite Element Modeling Code / T. H. Hubing, M. W. Ali // Proc. of the 1993 IEEE International EMC Symposium, August. – 1993. – P. 279-283.

54. Hubing T. H. EMAP: a 3-D, finite element modeling code for analyzing time-varying electromagnetic fields / T. H. Hubing, M. W. Ali, and G. K. Bhat, // Journal of the Applied Computational Electromagnetics Society. – 1993. – Vol. 8, no. 1.

55. Ali M. A Hybrid FEM/MOM Technique for Electromagnetic Scattering and Radiation from Dielectric Objects with Attached Wires / M. Ali, T. H. Hubing, J. Drewniak // IEEE Transactions on Electromagnetic Compatibility. – November 1997. – Vol. EMC-39, no. 4. – P. 304-314.

56. Ji Y. EMAP5: A 3D Hybrid FEM/MOM Code / Y. Ji, T. Hubing // Journal of the Applied Computational Electromagnetics Society. – March 2000. Vol. 15, no. 1. – P. 1-12.

57. The Standard Input File (SIF) Format [Электронный ресурс]. – Режим доступа: http://www.cvel.clemson.edu/modeling/EMAG/EMAP/sif.html. Дата доступа: 28.05.2012.

58. A-DDA [Электронный ресурс]. – Режим доступа: http://code.google.com/p/a-dda/. Дата доступа: 28.05.2012.

59. The discrete-dipole-approximation code ADDA: Capabilities and known limitations / Maxim A. Yurkin, Alfons G. Hoekstra // Journal of Quantitative Spectroscopy and Radiative Transfer. – September 2011. – P. 2234-2247.

60. Бронштейн И. Н. Справочник по математике для инженеров и учащихся вузов. / И. Н. Бронштейн. – М.: Наука, главная редакция физико-математической литературы. – 1981. – 722 с.

61. Bo Thidé Electromagnetic Field Theory. Second Edition [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.plasma.uu.se/CED/Book/EMFT_Book.pdf.

62. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. / С. Макконнелл. – М.: Русская редакция, 2011. – 896 с.: ил.

63. Нейгел К. C# 4.0 и платформа .NET 4 для профессионалов. / Пер. с англ. / Нейгел К., Ивьен Б., Глинн Д., Уотсон К., Скиннер М. – М.: Диалектика, 2011. – 1440 стр.: ил.

64. Арлоу Д. UML 2 и унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-ое издание. / Пер. с англ. / Арлоу Д., Нейштадт А. – СПб.: Символ-плюс, 2007. – 624 с.: ил.

65. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Д., Влиссидес Д. – СПб.: Питер, 2010. – 368 с.; ил.

66. Netgen Mesher [Электронный ресурс] – Режим доступа: http://sourceforge.net/projects/netgen-mesher/. Дата доступа: 11.06.2012.

67. GNU General Public License [Электронный ресурс]. – Режим доступа: http://www.opensource.org/licenses/gpl-2.0.php. Дата доступа: 12.06.2012.

68. Nlog Project [Электронный ресурс]. – Режим доступа: http://nlog-project.org/. Дата доступа: 12.06.2012.

69. BSD License [Электронный ресурс]. – Режим доступа: http://www.opensource.org/licenses/BSD-2-Clause. Дата доступа: 12.06.2012.

70. Math.NET Numerics is an opensource numerical library for .Net, Silverlight and Mono [Электронный ресурс] – Режим доступа: http://numerics.mathdotnet.com/. Дата доступа: 12.06.2012.

71. The MIT License (MIT) [Электронный ресурс] – Режим доступа: http://www.opensource.org/licenses/mit-license.php. Дата доступа: 12.06.2012.

72. Beitrage zur Optik truber Medien, Speziell killoidaler Metallosungen / G. Mie // Ann. d. Physik. – 1908. – Vol. IV 25. – P. 377.

73. Об утверждении укрупненных норм затрат труда на разработку программного обеспечения: постановление Министерства труда и социальной защиты Республики Беларусь, 27 июня 2007 г., № 91.

74. Об установлении размера тарифной ставки первого разряда для оплаты труда работников бюджетных организаций и иных организаций, получающих субсидии, работники которых приравнены по оплате труда к работникам бюджетных организаций: постановление Совета министров Республики Беларусь, 4 мая 2012 г., №410.

75. О внесении изменений и дополнений в Инструкцию о порядке применения Единой тарифной сетки работников Республики Беларусь: постановление Министерства труда и социальной защиты Республики Беларусь, 23 марта 2009, №40.

76. Межотраслевая типовая инструкция по охране труда при работе с персональными компьютерами: утв. Постановлением Министерства Труда и Социальной Защиты Республики Беларусь 30.11.2004 г. № 138: текст по состоянию на 01.01.2005 г.

77. СанПиН 2.2.2.542-96 Санитарно-эпидемиологические правила и нормативы “Гигиенические требования к видеодисплейным терминалам,

персональным электронно-вычислительным машинам и организации работы”. – М.: Госкомсанэпиднадзор, 1996.

78. Пожарная безопасность. Общие требования: ГОСТ 12.1.004-91 ССБТ.

79. Общие санитарно-гигиенические требования к воздуху рабочей зоны: ГОСТ 12.1.005-88 ССБТ.

80. Санитарно-эпидемиологические правила и нормативы «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы»: СанПиН 2.2.2/2.4.1340-03. – М.: Госкомсанэпиднадзор, 2003.

81. Шум. Общие требования безопасности: ГОСТ 12.1.003.

82. Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории жилой застройки (с изменениями и дополнениями, утвержденными Постановлением Главного государственного санитарного врача Республики Беларусь от 12.12.2005 г. № 220): СанПиН 2.2.4/2.1.8.10-32-2002.

ПРИЛОЖЕНИЕ А
(обязательное)
Численные методы решения уравнений Максвелла

На сегодня разработан ряд эффективных методов, позволяющих решать уравнения Максвелла в различных постановках. В частности, методы позволяют решать задачи в присутствии наночастиц и наноструктур. Методы позволяют найти отраженное и рассеянное поля, плазмонные резонансы, или определить функцию Грина, а также характеристики самих наночастиц и молекул вблизи них [3].

Методы нашли свое широкое применение для задач электродинамики, электростатики, нанооптики, наноплазмонике, а также применяются при проектировании и расчете электрических схем.

К основным методам можно отнести:

– приближение дискретных диполей (discrete dipole approximation (DDA));
– метод Т-матриц;
– метод конечных разностей во временной области (finite difference time domain (FDTD) method).

Методы имеют свои преимущества и недостатки. Их применение всегда обусловлено решаемой задачей. Так, изначально методы DDA и Т-матриц разрабатывались первоначально для решения задач рассеяния. При этом оказывается, что эти методы не очень хорошо описывают поле вблизи границ раздела.

Сегодня находит свое широкое применение в исследовании электромагнитных полей метод конечных элементов (МКЭ).

1 Метод DDA

Приближение дискретными диполями является одним из методов расчета рассеяния электромагнитного излучения в телах произвольной формы или в периодических структурах. Для заданной геометрии метод стремится вычислить собственные параметры рассеяния и поглощения. Точные решения уравнений Максвелла известны лишь для основных геометрических объектов, таких как сферы, сфероиды, поэтому требуются приближенные методы, которые позволяют решить задачу для нерегулярных форм объектов.

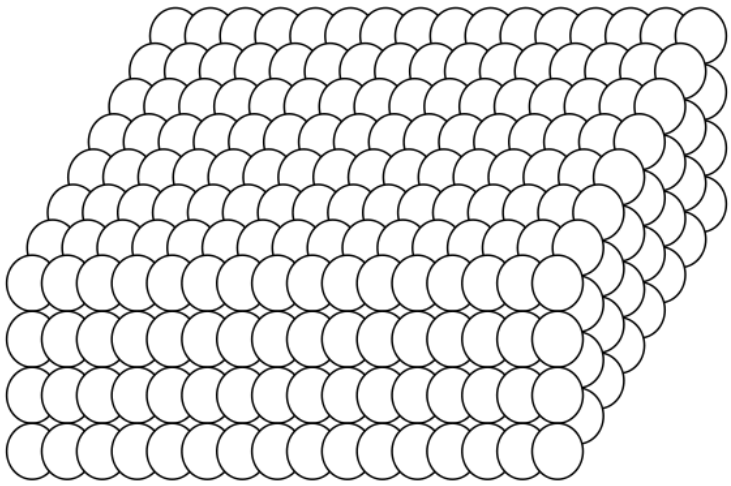


Рисунок А.1 – Вид большого объета, представленного в виде дискретных диполей

Метод DDA – это численный метод, в котором изучаемый объект представляется кубической решеткой с периодом d из N точек (рисунок А.1), которые обладают

свойством поляризуемости [9]. Иногда это метод называют CDA (приближение взаимодействующих диполей – coupled dipole approximation) методом. В рамках этого метода нет ограничений на то, занят ли узел ячейки диполем или нет. Это означает, что в принципе метод может быть использован для приближенного описания тел любой формы и состава. Предполагается, что *i*-ый элемент расположен в точке с радиусом-вектором r_i и обладает дипольной поляризуемостью a_i. Поляризуемости более высоких порядков (квадрупольная и т.д.) считаются равными нулю. Дипольный момент каждого элемента возникает в результате воздействия на него локального поля E_{loc}, которое предполагается монохроматическим (e^{-iwt}):

$$P_i = a_i E_{loc}(r_i)\,. \qquad \text{(A.1)}$$

В свою очередь, E_{loc} является суммой падающего и обусловленных всеми остальными диполями полей:

$$E_{loc}(r_i) = E_{loc,i} = E_{in,i} + E_{dip,i} = E_0 \exp(ik \cdot r_i) - \sum_{j \neq i} A_{ij} P_j\,, \qquad \text{(A.2)}$$

где E_0 и k – амплитуда и волновой вектор падающей волны (которая предполагается плоской, но может, в общем случае, быть любой), а матрица взаимодействия A имеет следующую форму (для $j \neq i$):

$$A_{ij}P_j = \frac{\exp(ikr_{ij})}{r_{ij}^3} \times \left\{ k^2 r_{ij} \times (r_{ij} \times P_j) + \frac{1 - ikr_{ij}}{r_{ij}^2}\left[r_{ij}^2 \cdot P_j - 3r_{ij} \cdot (r_{ij} \cdot P_j) \right] \right\}, \qquad \text{(A.3)}$$

где $k = w/c$.

Фактически матрица A – это функция Грина уравнений Максвелла с дипольным источником в свободном пространстве.

Важно иметь в виду, что диэлектрическая проницаемость металлической частицы и окружающей среды входят в виде отношения $\varepsilon_i/\varepsilon_0$, которое содержится в выражении для поляризуемости. Кроме того, волновой вектор должен быть домножен на $\sqrt{\varepsilon_0}$ если частица находится не в вакууме. Явные выражения для поляризуемости a_i были разработаны [9,11,12] так, чтобы бесконечная решетка диполей точно описывала диэлектрические свойства сплошного тела

$$a_m^{CM} = \frac{3}{n}\frac{\varepsilon(r_m) - 1}{\varepsilon(r_m) + 2}\,, \qquad \text{(A.4)}$$

где $n = 1/d^3$ – концентрация диполей, а $\varepsilon(r_m)$ – значение диэлектрической проницаемости в месте расположения *m*-го диполя. Еще лучшие результаты дает выражение, которое учитывает радиационные поправки [9].

$$a_m = \frac{a_m^{CM}}{1 + \frac{a_m^{CM}}{d^3}\left[(b_1 + b_2 \cdot \varepsilon(r_m) + b_3 \cdot S \cdot \varepsilon(r_m)) \cdot (kd)^2 - \frac{2}{3} i (kd)^3 \right]}. \qquad \text{(A.5)}$$

В (A.5) b_1 = -1,8915316, b_2 = 0,1648469 и b_3 = -1,7700004, а *S* – функция направления плоской волны *a* и ее поляризации *e*:

$$S = \sum_{j=1}^{3} (e_j a_j)^2\,. \qquad \text{(A.6)}$$

Применение такой процедуры для наночастиц является приближенным и на практике спектры экстинкции металлических наночастиц обычно отличаются на 10% от точного результата независимо от формы, состава и размера моделируемых частиц.

Подставляя уравнения (А.3) и (А.2) в (А.1), после простых преобразований приходим к матричному уравнению,

$$\sum_{j\beta}[A']_{ia,j\beta}\cdot P_{ia}=[E]_{in,ia}\,, \tag{A.7}$$

где матрица A' получается из матрицы A в (А.3) и греческие индексы обозначают декартовы координаты. Для системы из N диполей E и P в (А.7) являются векторами размерности $3N$, а матрица A' является $3N\times 3N$ матрицей. Решая систему $3N$ комплексных линейных уравнений, можно найти вектор дипольных моментов и полный дипольный момент системы и другие оптические свойства. В частности, сечение экстинкции и сечение поглощения выражаются через вектор поляризация и имеют

$$\begin{aligned} C_{ext} &= \frac{4\pi k}{|E_0|^2}\sum_{j=1}^{N}\mathrm{Im}\left(E_{j,in}\cdot p_j\right), \\ C_{abs} &= \frac{4\pi k}{|E_0|^2}\sum_{j=1}^{N}\leq\left\{\mathrm{Im}\left(p_j\cdot\left(\overleftrightarrow{a}_j^{-1}\right)\cdot p_j\right)-\frac{2}{3}k^3\left|p_j\right|^2\right\}. \end{aligned} \tag{A.8}$$

На практике вычисления существенно ускоряются, если при суммировании в (А.2) использовать быстрое преобразование Фурье и решать уравнение (А.7) методом комплексно сопряженных градиентов. Все эти упрощения реализованы в работах [9,11,12]. Дальнейшие детали вычислений и примеры можно найти, например, в работах [13,14], а также [3, гл. 9].

При вычислении спектров экстинкции и рассеяния на наночастицах различной формы удобно их нормировать на эффективное геометрическое сечение A:

$$Q_{ext}=\frac{C_{ext}}{A};Q_{sca}=\frac{C_{sca}}{A};Q_{abs}=\frac{C_{abs}}{A}, \tag{A.9}$$

которое определяется как $A=\pi a_{eff}^2$, где эффективный радиус определяется из равенства объема частицы объему сферы с радиусом a_{eff}.

2 Метод FDTD

В последние несколько лет этот численный метод был успешно применен к решению различных проблем в нанонауке [15 – 18]. В рамках этого метода непосредственно уравнения Максвелла во временной области решаются методом конечных разностей. Говоря точнее, применение этого метода начинается с выписывания уравнений Ампера и Фарадея. Например, если рассматривать наносистему с диэлектрической проницаемостью $\varepsilon(x,y,z)$, эти два закона имеют вид:

$$rotE(x,y,z,t)=-\frac{1}{c}\frac{\partial B(x,y,z,t)}{\partial t}, \tag{A.10}$$

и

$$rotH(x,y,z,t)=\frac{\varepsilon(x,y,z,t)}{c}\frac{\partial E(x,y,z,t)}{\partial t}, \tag{A.11}$$

соответственно.

Если эти векторные уравнения аппроксимировать с помощью конечных разностей во временной и пространственной областях, то мы получим 6 скалярных конечно-разностных уравнений:

$$\begin{aligned} &\frac{E_z(x,y+\Delta y,z,t)-E_z(x,y-\Delta y,z,t)}{2\Delta y}-\frac{E_y(x,y,z+\Delta z,t)-E_y(x,y,z-\Delta z,t)}{2\Delta z}= \\ &-\frac{B_x(x,y,z,t+\Delta t)-B_x(x,y,z,t)}{c\Delta t}, \end{aligned} \tag{A.12}$$

$$\frac{E_x(x,y,z+\Delta z,t)-E_x(x,y,z-\Delta z,t)}{2\Delta z}-\frac{E_z(x+\Delta x,y,z,t)-E_z(x-\Delta x,y,z,t)}{2\Delta x}=$$
$$-\frac{B_y(x,y,z,t+\Delta t)-B_y(x,y,z,t)}{c\Delta t}, \qquad (A.13)$$

$$\frac{E_y(x+\Delta x,y,z,t)-E_y(x-\Delta x,y,z,t)}{2\Delta x}-\frac{E_x(x,y+\Delta y,z,t)-E_x(x,y-\Delta y,z,t)}{2\Delta y}=$$
$$-\frac{B_z(x,y,z,t+\Delta t)-B_z(x,y,z,t)}{c\Delta t}, \qquad (A.14)$$

и

$$\frac{H_x(x,y+\Delta y,z,t)-H_x(x,y-\Delta y,z,t)}{2\Delta y}-\frac{H_y(x,y,z+\Delta z,t)-H_y(x,y,z-\Delta z,t)}{2\Delta z}=$$
$$\varepsilon(x,y,z)\left(\frac{E_x(x,y,z,t+\Delta t)-E_x(x,y,z,t)}{c\Delta t}\right), \qquad (A.15)$$

$$\frac{H_y(x+\Delta x,y,z,t)-H_y(x-\Delta x,y,z,t)}{2\Delta x}-\frac{H_x(x,y+\Delta y,z,t)-H_x(x,y-\Delta y,z,t)}{2\Delta y}=$$
$$\varepsilon(x,y,z)\left(\frac{E_z(x,y,z,t+\Delta t)-E_z(x,y,z,t)}{c\Delta t}\right), \qquad (1.16)$$

$$\frac{H_x(x,y,z+\Delta z,t)-H_x(x,y,z-\Delta z,t)}{2\Delta z}-\frac{H_z(x+\Delta x,y,z,t)-H_z(x-\Delta z,y,z,t)}{2\Delta x}=$$
$$\varepsilon(x,y,z)\left(\frac{E_y(x,y,z,t+\Delta t)-E_y(x,y,z,t)}{c\Delta t}\right). \qquad (A.17)$$

Из этих уравнений, зная значения полей в дискретных точках пространственной сетки, можно найти значения этих полей в этих же точках пространства в момент времени $t+\Delta t$.

Приведенная выше численная схема имеет лишь первый порядок точности по Δt и $\Delta x, \Delta y, \Delta z$, и для достижения требуемой точности необходимо брать очень мелкие сетки как по времени, так и по пространству, что, в свою очередь, приводит к недопустимо большим размерностям матриц.

В современных реализациях метода FDTD используется явная схема второго порядка точности по Δt и $\Delta x, \Delta y, \Delta z$, предложенная Йи [19]. В рамках этого подхода используется специальная пространственная сетка, так называемая сетка Йи, где каждая *E*-компонента окружена 4-мя *H*-компонентами и наоборот (рисунок А.2). Второй порядок точности временных производных достигается простой симметричной разностью.

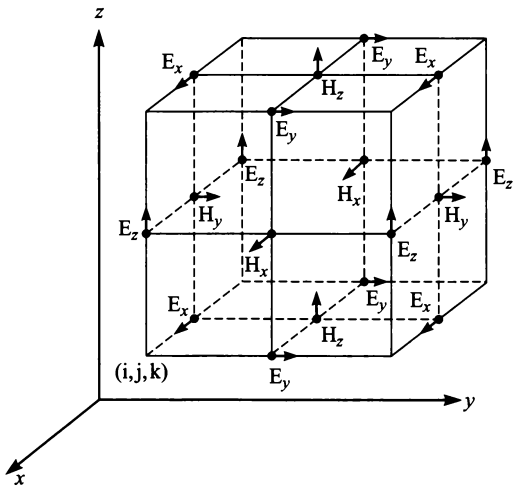


Рисунок А.2 – Расположение полевых компонент в единой ячейке Йи

В результате такого выбора пространственно-временной сетки дискретный вариант, например, *x*-компоненты уравнения (А.10) примет вид

$$H_x\Big|_{i,j,k}^{n+1/2} = H_x\Big|_{i,j,k}^{n+1/2} + \\ + c\Delta t \times \left[\frac{E_y\Big|_{i,j,k+1/2}^{n} - E_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} - \frac{E_x\Big|_{i,j+1/2,k}^{n} - E_x\Big|_{i,j-1/2,k}^{n}}{\Delta y} \right]. \qquad \text{(A.18)}$$

где верхний индекс соответствует временной сетке, а нижние 3 индекса соответствуют пространственной сетке вдоль *x*-, *y*- и *z*- направлений соответственно. Из этого уравнения видно, что как временные, так и пространственные производные аппроксимированы центральными производными, и поэтому эта аппроксимация имеет второй порядок точности. Аналогичные уравнения имеют место и для других компонент уравнений (А.10) и (А.11). Таким образом, весь метод конечных разностей во временной области в формулировке Йи [19] является методом второго порядка в областях с однородными значениями диэлектрической и магнитной проницаемости.

При конкретных вычислениях необходимо также учитывать, что для стабильности между значением шага во временной области и значением шага пространственной решетки должно выполняться так называемое условие Куранта-Фридрихса-Леви [20]:

$$\Delta t \le \frac{1}{v\sqrt{1/\Delta x^2 + 1/\Delta y^2 + 1/\Delta z^2}}, \qquad \text{(A.19)}$$

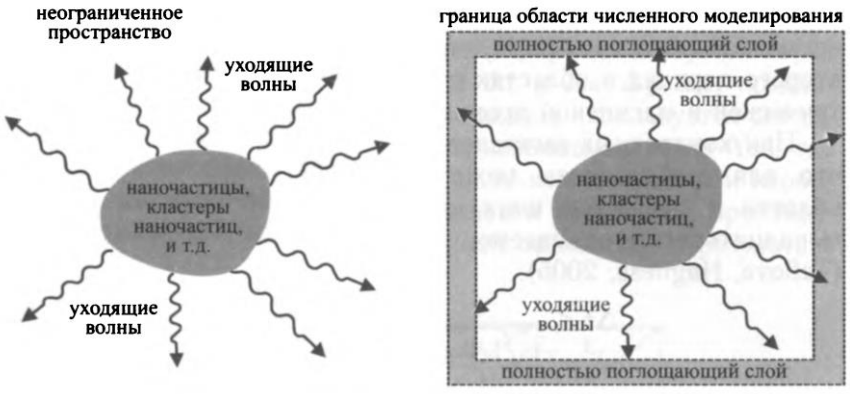
где $v = \dfrac{c}{\sqrt{\varepsilon\mu}}$ – скорость света в среде.

При выполнении этого условия метод конечных разностей во временной области (FDTD) становится очень надежной вычислительной схемой. Важной особенностью применения метода (FDTD) является наличие информации о состоянии поля во все времена моделирования, что позволяет легко визуализировать процесс распространения импульсов в наночастицах и наноструктурах, перед тем как установится стационарный монохроматический режим.

Важной особенностью метода FDTD является необходимость знать значения полей не только в области наночастиц и наноструктур, но и во всем пространстве, что, конечно, невозможно по чисто техническим причинам. Для того чтобы ограничить вычисления конечными объемами и конечными сетками, в методе FDTD используются так называемые граничные условия поглощения (ABC – absorbing boundary conditions, рисунок А.3). Наложение этих условий позволяет избежать возникновения волн, отраженных от границы рассматриваемой области пространства, которая содержит в себе рассматриваемые наночастицы и наноструктуры. Поэтому можно рассматривать изучаемую систему расположенной в неограниченном пространстве.

Детальная классификация различных типов граничных условий поглощения приведена в книге [20]. Более детальное объяснение того, почему нужны граничные условия поглощения, приводится в книге [21].

Большинство граничных условий поглощения может быть разбито на две категории. Одна категория АВС выводится из самих дифференциальных уравнений. Другая категория АВС связана с введением в систему гипотетических материальных поглотителей.



а б

Рисунок А.3 – Иллюстрация метода граничного условия полного поглощения на примере идеально поглощающего слоя: а – неограниченная область, б – область, ограниченная слоем PML

В настоящее время при моделировании методом FDTD наиболее часто используется граничное условие идеально согласованного слоя (PML), которое основано на введении поглотителя и было впервые предложено в работе [22].

Граничное условие поглощения PML реализуется путем окружения области вычисления слоем поглощающего материала, который ослабляет уходящие волны. При соответствующим образом подобранных параметрах поглощающего слоя падающие на этот слой волны не испытывают отражения при любых углах падения и быстро затухают при распространении внутри этого поглощающего слоя.

Коэффициент отражения равен нулю, если параметры поглощающего слоя связаны соотношением

$$\frac{\sigma_e}{\varepsilon} = \frac{\sigma_m}{\mu}, \qquad \text{(A.20)}$$

где σ_e, σ_m – электрическая и магнитная проводимости, а ε, μ – электрическая и магнитная проницаемости поглощающего слоя, которые, вообще говоря, являются анизотропными тензорами.

Внешняя граница поглощающего слоя считается идеально проводящей, т.е. она отражает все волны внутрь поглощающего слоя. Заметим, что условие (А.20) является схематичным и применяется лишь для определенных компонент соответствующих тензоров, различных для разных сторон поглощающего слоя (рисунок А.3).

Одним из наиболее важных моментов при использовании метода FDTD для описания плазмонных наночастиц и наноструктур является корректное описание законов дисперсии в металлах. В отличие от методов в частотной области, включение законов дисперсии в метод FDTD возможно только при использовании той или иной аналитической модели закона дисперсии в металле, например, при использовании модели Друде или модели Друде-Лоренца [3, гл. 3, разд. 3.2]. Наиболее часто при моделировании наночастиц с дисперсией методом FDTD используется метод дополнительного дифференциального уравнения (auxiliary differential equation (ADE) method) [23]. В частотной области соотношение между индукцией и напряженностью электрического поля имеет вид:

$$D = \varepsilon(\omega)E\,. \qquad \text{(A.21)}$$

Метод ADE основан на переводе соотношения (А.21) во временную область и использовании его вместе с остальными дифференциальными уравнениями Максвелла во временной области. Если взять дисперсионную зависимость в простейшей форме Друде-Лоренца

$$\varepsilon(\omega) = \frac{\omega_0^2}{\omega_0^2 - \omega^2 - i\omega\gamma_0}, \qquad \text{(A.22)}$$

подставить ее в (А.21) и выполнить обратное преобразование Фурье, то получится дифференциальное уравнение, которое связывает *D* и *E*

$$\omega_0^2 D + \gamma_0 \frac{\partial D}{\partial t} + \frac{\partial^2 D}{\partial t^2} = \omega_0^2 E\,. \qquad \text{(A.23)}$$

Затем это уравнение дискретизируется с использованием центральных разностей так, чтобы получить аппроксимацию второго порядка точности.

При использовании метода вспомогательного уравнения электрическое поле вычисляется в два этапа. На первом этапе вектор индукции получается из дискретизированной версии уравнения

$$rotH = \frac{1}{c}\frac{\partial D}{\partial t}, \qquad \text{(A.24)}$$

которая, например, для *x*-компоненты индукции имеет вид

$$D_x\Big|_{i,j,k}^{n+1} = D_x\Big|_{i,j,k}^{n} + \\ + c\Delta t \times \left[\frac{H_y\Big|_{i,j,k+1/2}^{n+1/2} - H_y\Big|_{i,j,k-1/2}^{n+1/2}}{\Delta z} - \frac{H_x\Big|_{i,j+1/2,k}^{n+1/2} - H_x\Big|_{i,j-1/2,k}^{n+1/2}}{\Delta y} \right]. \qquad \text{(A.25)}$$

Для остальных компонент имеем аналогичные уравнения.

На втором этапе электрическое поле получается решением дискретизированной формы уравнения (А.23). При этом для нахождения поля в момент времени $t = (n+1)\Delta t, \left(E^{n+1}\right)$ необходимо знать индукцию как в момент времени $t = (n+1)\Delta t, \left(D^{n+1}\right)$, так и в 2 предшествующих момента времени $t = (n-1)\Delta t, t = (n-2)\Delta t, \left(D^{n-1}, D^{n-2}\right)$.

3 Метод Т-матриц

Метод Т-матриц часто применяется для описания кластеров сферически наночастиц, в принципе является точным и базируется на теории Ми для каждой частицы и теореме о сложении для векторных сферических гармоник [24, 25].

Этот метод был разработан Waterman в 1965 году [26] и дополнен в 1971 [27] и значительно улучшен далее Mishchenko, Travis, Mackowski [28]. Процедура аналитического вычисления Т-матрицы для кластера из сфер была описана в деталях в [29]. Подробная библиография метода описана в [30].

Краткое изложение метода основано на работе [25]. Рассмотрим кластер произвольной формы из *N* частиц произвольной формы, на который падает плоская электромагнитная волна. Поле, рассеянное этим кластером, является суперпозицией полей, рассеянных отдельными частицами кластера:

$$E_s = \sum_{i=1}^{N_s} E_{s,i}\,, \qquad \text{(A.26)}$$

где поле каждой частицы $E_{s,i}$ в свою очередь может быть разложено по векторным сферическим гармоникам связанным с центами соответствующих частиц:

$$E_{s,i} = \sum_{n=1}^{\infty} \sum_{m=-n}^{n} \sum_{p=1}^{2} a_{mnp}^{i} h_{mnp}(r_i)\,. \qquad \text{(A.27)}$$

Здесь, *h* обозначает расходящуюся сферическую волну степени *n* и порядка *m*, a^i – коэффициенты разложения и p означает поляризацию поля, *p=1* и *2* обозначают TM- и TE-поляризации соответственно.

Так как все сферы взаимодействуют друг с другом, то поле рассеянное *i*-ой сферой, обусловлено как непосредственно падающим полей, так и полями, рассеянными другими сферами. Использую теорему сложения для векторных гармоник, все поля можно преобразовать в ряды специфических гармоник с центром в рассматриваемой *i*-ой сфере. Обрывая суммирование по *n* в (А.27) на $n = N_{O,i}$,получаем систему линейных уравнений типа

$$a^i_{mnp} + \bar{a}^i_{np} \sum_{j=1,\, j\neq i}^{N_s} \sum_{l=1}^{N_{0,j}} \sum_{k=-1}^{l} \sum_{q=1}^{2} H^{ij}_{mnpklq} a^j_{klq} = \bar{a}^i_{klq} p^i_{mnp}\,, \qquad \text{(А.28)}$$

где матрица H_{ij} формируется из коэффициентов входящих в теорему сложения для сферических гармоник (на основе уравнения Ханкеля) и зависит только от расстояния и направления вектора, соединяющего центры *j*-ой и *i*-ой сфер, $\bar{a}^i$ – коэффициенты рассеяния сферических волн различной мультипольности *i*-ой сферой, которые даются теорией Ми, и, наконец, p^i – коэффициенты разложения падающей плоской волны по сферическим гармоникам с центром в центре *i*-ой сферы. Формальное обращение этой системы уравнений дает Т-матрицу кластера

$$a^i_{nmp} = \sum_{j=1}^{N_s} \sum_{l=1}^{N_{0,j}} \sum_{k=-1}^{l} \sum_{q=1}^{2} T^{ij}_{mnpklq} p^j_{klq}\,, \qquad \text{(А.29)}$$

где матрица $T^{i,j}$ связана с центом *i*-ой сферы. Ее можно преобразовать к Т-матрице, связанной с центром кластера.

$$T_{nl} = \sum_{i=1}^{N_s} \sum_{j=1}^{N_s} \sum_{n'=1}^{N_{0,j}} \sum_{l'=1}^{N_{0,i}} J^{0i}_{nn}, T^{ij}_{n'l''}, J^{j0}_{l'l'}\,, \qquad \text{(А.30)}$$

где J^{0i} и J^{0j} – матрицы, сформированные из коэффициентов теоремы сложения для сферических функций Бесселя, и индексы nn' и ll' обозначают тройной индекс, характеризующий сферическую волну. Знание Т-матрицы позволяет найти все интересующие оптические свойства кластера наночастиц. Например, для сечения экстинкции выражение имеет вид

$$C_{ext} = \frac{2\pi}{k^2} \mathrm{Re}\left(\sum_{nmp} T_{mnpmnp}\right). \qquad \text{(А.31)}$$

Метод Т-матрицы может быть также применен и к несферическим частицам. В этом случае он становится приближенным, но весьма эффективным для не очень вытянутых частиц.

4 Метод конечных элементов

Как и метод конечных разностей (FDTD), метод конечных элементов (МКЭ) является численным методом, в котором производится преобразование уравнений в частных производных к системе линейных алгебраических уравнений (СЛАУ) для получения приближенных решений на границе – краевых задач математической физики. Вместо аппроксимации дифференциальных операторов, метод конечных элементов сводится к решению уравнения в частных производных. Поскольку метод был впервые предложен в 1943 году Курант для решения вариационных задач в теории потенциалов [31], то он был хорошо разработан и широко применяется для решения проблем структурного анализа и в других областях. В настоящее время метод конечных элементов

признается в качестве выдающегося метода, применимого для широкого круга инженерных и математических задач, в том числе и в СВЧ-технике и электромагнетизме.

Первое применение метода конечных элементов для СВЧ-техники и в электромагнетизме появилось в 1969 году, когда Silvester использовал его для анализа волн в полом волноводе [32]. Значимость метода быстро признали, после чего получили и успешно применили его для анализа различных электростатических, магнитостатических проблем в волноводе, а также в заполненном диэлектриком волноводе. В 1974 году Ми разработал технику, которая в сочетании МКЭ с собственным расширением, имела дело с открытой областью электромагнитных задач, таких как антенны и анализ рассеяния и [33]. В 1982 году Marin разработал альтернативный метод, который в сочетании МКЭ и метода граничных интегральных уравнений был применен для решения открытой проблемы региона рассеяния [34].

Важным прорывом в конечноэлементом анализе векторных задач электромагнитного поля стало появление в 1980-ых годах так называемых векторных элементов, основанных на ребрах (edge based elements) [35 – 37]. Эти новые элементы позволили точно моделировать характер электрических и магнитных полей и устранить многие проблемы, связанные с традиционными узловыми скалярными элементам. Это было особо примечательно тем, что сам по себе характер и свойства электромагнитных полей являлись до этого момента преградой для их изучения. С развитием векторных элементов, МКЭ стал очень мощным численным методом для решения задач электромагнетизма. Сегодня этот метод используется в качестве основного инструмента для разработки антенн и СВЧ устройств. Его основные принципы и различные приложения были описаны во многих книгах, таких, как Silvester и Ferrari [38] и Jin [39 – 41].

ПРИЛОЖЕНИЕ Б
(справочное)
L-координаты

Нерегулярные сетки, содержащие ячейки переменной формы и размера, а также ячейки с различным числом узлов, строятся обычно без использования отображений сразу в пространстве образа (в актуальной области решения). Для нерегулярных сеток применяется кусочно-полиномиальная (конечно-элементная) интерполяция. Для этого на простейших одномерных (отрезок), двухмерных (треугольник) или трехмерных (тетраэдр) конечных элементах используются так называемые *L*-координаты.

Четыре относительных координаты определяют как отношения расстояний от выбранной точки внутри тетраэдра до одной из его граней к длине высоты, опущенной на эту грань из противоположной вершины. Связь трехмерных *L*-координат можно записать в виде системы уравнений вида:

$$x = L_1X_1 + L_2X_2 + L_3X_3 + L_4X_4,$$
$$y = L_1Y_1 + L_2Y_2 + L_3Y_3 + L_4Y_4,$$
$$z = L_1Z_1 + L_2Z_2 + L_3Z_3 + L_4Z_4,$$
$$1 = L_1 + L_2 + L_3 + L_4.$$

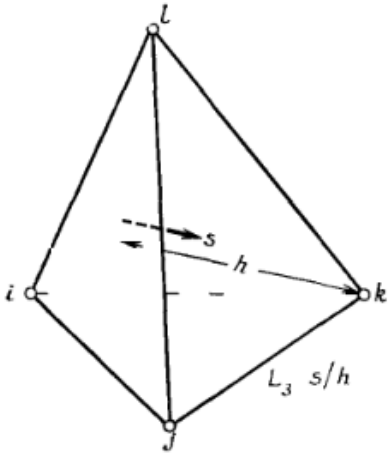


Рисунок Б.1 – Объемные L-координаты тетраэдра

В трехмерном случае *L*-координаты применяют для нахождения объемных интегралов вида:

$$\int_V L_1^a L_2^b L_3^c L_4^d dV = \frac{a!b!c!d!}{(a+b+c+d)!}6V, \qquad (Б.1)$$

где V – объем тетраэдра.

$$V = \frac{1}{6}\begin{vmatrix} 1 & x1 & y1 & z1 \\ 1 & x2 & y2 & z2 \\ 1 & x3 & y3 & z3 \\ 1 & x4 & y4 & z4 \end{vmatrix}. \qquad (Б.2)$$

Вычисление трехмерных (объемных) *L*-координат довольно сложно в аналитическом виде. Двухмерные координаты вычислены аналитически и часто приведены в книгах по конечно-элементному моделированию. Для аналитического

получения координат в трехмерном случае была разработана программа в Matlab (рисунок Б.1).

```
function TatraLCords()
   syms x y z x1 x2 x3 x4 y1 y2 y3 y4 z1 z2 z3 z4 L1 L2 L3 L4;

   % built system of equation and solve system - get all L's
   [L1, L2, L3, L4] = solve('1=L1+L2+L3+L4',...
      'x=x1*L1+x2*L2+x3*L3+x4*L4',...
      'y=y1*L1+y2*L2+y3*L3+y4*L4',...
      'z=z1*L1+z2*L2+z3*L3+z4*L4', L1, L2, L3, L4);

   % calculate determinant
   DMat =[1,x1,y1,z1;
      1,x2,y2,z2;
      1,x3,y3,z3;
      1,x4,y4,z4];
   D = det(DMat);

   L1=L1*D;
   L2=L2*D;
   L3=L3*D;
   L4=L4*D;

   % calculates differencials
   [L1x] = diff(L1,'x')*D;
   [L1y] = diff(L1,'y')*D;
   [L1z] = diff(L1,'z')*D;

   [L2x] = diff(L2,'x')*D;
   [L2y] = diff(L2,'y')*D;
   [L2z] = diff(L2,'z')*D;

   [L3x] = diff(L3,'x')*D;
   [L3y] = diff(L3,'y')*D;
   [L3z] = diff(L3,'z')*D;

   [L4x] = diff(L4,'x')*D;
   [L4y] = diff(L4,'y')*D;
   [L4z] = diff(L4,'z')*D;

   % built vector function
   W12=L1*[L2x;L2y;L2z]-L2*[L1x;L1y;L1z];
   W13=L1*[L3x;L3y;L3z]-L3*[L1x;L1y;L1z];
   W14=L1*[L4x;L4y;L4z]-L4*[L1x;L1y;L1z];
   W23=L2*[L3x;L3y;L3z]-L3*[L2x;L2y;L2z];
   W42=L4*[L2x;L2y;L2z]-L2*[L4x;L4y;L4z];
   W34=L3*[L4x;L4y;L4z]-L4*[L3x;L3y;L3z];
end
```

Рисунок Б.1 – Программа для вычисления трехмерных *L*-координат

Программа вычисляет:

- аналитический вид определителя матрицы;
- аналитические значения *L* координанат;
- векторные базисные функции для тетраэдра.

ПРИЛОЖЕНИЕ В
(обязательное)
Листинги программного обеспечения

MainApp

Program.cs:

```
using System;
using System.Linq;
using EMFFM.MainApp.Helpers;
using EMFFM.MainApp.Input;
using EMFFM.MainApp.Controllers;
using System.IO;
using NLog;

namespace EMFFM.MainApp
{
    /// <summary>
    /// Главный класс программы для обработки запуска расчетов
    /// </summary>
    class Program
    {
        static Logger logger = LogManager.GetCurrentClassLogger();

        static string DefaultInputFile = "input\\in5.xml";

        /// <summary>
        /// Входные данные для программы
        /// </summary>
        static InputData inData;

        /// <summary>
        /// Разбор входного файла с данными для задачи
        /// </summary>
        /// <param name="fname">Файл с данными</param>
        static void ParseInputFile(string fname)
        {
            if (File.Exists(fname))
            {
                logger.Info("Начало чтения и разбора входного файла...");

                // разбор входного файла с помощью специального класса
                inData = InputFileReader.ReadInputFile(fname);

                logger.Info("Разбор входного файла успешно завершен!");
            }
            else
            {
                throw new Exception(String.Format("Указанного файла с входными данными не существует! Файл: {0}",
Path.GetFullPath(fname)));
            }
        }

        /// <summary>
        /// Разбор и обработка аргументов из командной строки
        /// </summary>
        /// <param name="args">Аргументы</param>
        static void ParseArguments(string[] args)
        {
            // разбор аргументов из командной строки
            if (args.Length == 0)
            {
                logger.Debug("Command line arguments is null! Trying to use default input file...");

                ParseInputFile(DefaultInputFile);
            }
            else
            {
                ParseInputFile(args[0]);
            }
        }

        /// <summary>
        /// Главная функция консольного приложения
        /// </summary>
        /// <param name="args">Аргументы командной строки</param>
```

```
        static void Main(string[] args)
        {
            logger.Log(LogLevel.Info, "Starting EMFFM...");

            try
            {
                DateTime start = DateTime.Now;

                // разбор аргументов командной строки
                ParseArguments(args);

                // очистка рабочей папки для вывода файлов результатов
                IOHelper.ClearOutputPath(inData.Output.Path);
                // создание и запуск главного контроллера всей программы
                AppController app = new AppController(inData);
                app.Run();
                DateTime end = DateTime.Now;
                logger.Info("Time to run:  {0} ms", (end - start).TotalMilliseconds);
            }
            catch (Exception ex)
            {
                logger.Error(String.Format("Возникло исключение: {0}", ex.Message));
            }

            logger.Info("Ending is EMFFM...");
            // ждем ответа пользователя (TODO: потом убрать ожидание ответа пользователя)
            Console.WriteLine("\nPress any key to exit...");
            Console.ReadKey();
        }
    }
}
```

InputFileReader.cs:

```
using System;
using System.Linq;
using System.Xml;
using System.Collections.Generic;
using EMFFM.MainApp.Common;
using EMFFM.MainApp.Helpers;
using EMFFM.MainApp.Mesh.Elements;

namespace EMFFM.MainApp.Input
{
    /// <summary>
    /// Класс для работы со входным файлом данных (чтения из него)
    /// </summary>
    class InputFileReader
    {
        static NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();

        /// <summary>
        /// Чтение исходного файла с данными для задачи
        /// </summary>
        /// <param name="fname">Файл для чтения (путь и название файла)</param>
        public static InputData ReadInputFile(string fname)
        {
            try
            {
                logger.Info("Start reading input file...");

                // открываем и загружаем документ в переменную doc
                XmlDocument doc = new XmlDocument();
                doc.Load(fname);

                // создание объекта с входными данными для возврата
                InputData data = new InputData();

                // чтение секций входного файла и их разбор
                XmlNodeList list = doc.DocumentElement.ChildNodes;

                // вне зависимости от порядка следования элементов (секциий) разбираем их
                foreach (XmlNode item in list)
                {
                    if (item.NodeType == XmlNodeType.Element)
                    {
                        switch (item.Name)
                        {
                            // обязательные секции
```

```
                case "task": // чтение класса задачи, описываемой в файле
                    ReadTask(item, ref data);
                    break;
                case "actions": // чтение секции с действиями
                    ReadActionSection(item.ChildNodes, ref data);
                    break;
                case "output": // чтение секции с определенями выходных файлов
                    ReadOutputSection(item.ChildNodes, ref data);
                    break;
                case "options": // чтение секции с опциями и параметрами
                    ReadOptionsSection(item.ChildNodes, ref data);
                    break;
                case "boundary": // чтение секции с параметрами граничных условий и их тип
                    ReadBoundarySection(item.ChildNodes, ref data);
                    break;
                case "variables": // чтение секции с описанием переменных и констант
                    ReadVariablesSection(item.ChildNodes, ref data);
                    break;
                case "materials": // чтение секции с описанием материалов, используемых в задаче
                    ReadMaterialsSection(item.ChildNodes, ref data);
                    break;
                case "sources":
                    ReadSourcesSection(item.ChildNodes, ref data);
                    break;
                    // секции для собственного генератора сетки
                case "box": // чтение секции с описание геометрии области
                    ReadBoxSection(item.ChildNodes, ref data);
                    break;
                case "mesh": // чтение секции с параметрами сетки
                    ReadMeshSection(item.ChildNodes, ref data);
                    break;
                case "particles": // чтение секции с параметрами наночастиц
                    ReadParticlesSection(item.ChildNodes, ref data);
                    break;
                    // секция если для работы с программой Netgen
                case "geometry":
                    ReadGeometrySection(item.ChildNodes, ref data);
                    break;
                    // справочные секции
                case "plasmons": // чтение секции с описанием плазмонов
                    ReadPlasmonsSection(item.ChildNodes, ref data);
                    break;
                case "probes":
                    ReadProbesSection(item.ChildNodes, ref data);
                    break;
            }
        }
    }

    logger.Info("Input file sucesfull readed!");

    return data;
  }
  catch (Exception ex)
  {
    logger.Error(String.Format("Catch error in input file reading process.  Error message: {0}", ex.Message));
    throw new Exception("Возникло исключение при обработке входного файла! " + ex.Message);
  }
}

#region Чтение и разбор секций

/// <summary>
/// Чтение класс решаемой задачи, описываемой во входном файле
/// </summary>
/// <param name="xmlNode">Узел XML с данными</param>
/// <param name="data">Входные данные</param>
private static void ReadTask(XmlNode xmlNode, ref InputData data)
{
  data.Task = (TaskClasses)Enum.Parse(typeof(TaskClasses), xmlNode.Attributes["name"].Value);
}

/// <summary>
/// Разбор секции с геометрией области
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadBoxSection(XmlNodeList xmlNodeList, ref InputData data)
```

```
{
    // читаем координаты для левой нижней точки
    data.Box.P1 = new Point(Convert.ToDouble(xmlNodeList.Item(0).Attributes["x"].Value),
        Convert.ToDouble(xmlNodeList.Item(0).Attributes["y"].Value),
        Convert.ToDouble(xmlNodeList.Item(0).Attributes["z"].Value));
    // читаем координаты для верхней правой точки
    data.Box.P2 = new Point(Convert.ToDouble(xmlNodeList.Item(1).Attributes["x"].Value),
        Convert.ToDouble(xmlNodeList.Item(1).Attributes["y"].Value),
        Convert.ToDouble(xmlNodeList.Item(1).Attributes["z"].Value));
}

/// <summary>
/// Чтение секции с описанием сетки
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadMeshSection(XmlNodeList xmlNodeList, ref InputData data)
{
    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "size": // размер КЭ
                data.Element.Da = Convert.ToDouble(node.Attributes["x"].Value);
                data.Element.Db = Convert.ToDouble(node.Attributes["y"].Value);
                data.Element.Dc = Convert.ToDouble(node.Attributes["z"].Value);
                break;
            case "type": // тип КЭ
                string tmp = node.Attributes["value"].Value;
                data.Element.Type = (ElementType)Enum.Parse(typeof(ElementType), tmp);
                break;
        }
    }
}

/// <summary>
/// Чтение секции со списком действий
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadActionSection(XmlNodeList xmlNodeList, ref InputData data)
{
    data.Actions = new List<Action>();

    // считываем все доступные действия
    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "action":
                data.Actions.Add(new Action()
                {
                    Code = Convert.ToInt32(node.Attributes["code"].Value),
                    Value = (ActionNames)Enum.Parse(typeof(ActionNames), node.Attributes["value"].Value)
                });
                break;
        }
    }

    // сортировка действий по коду
    data.Actions.Sort();

    // TODO: добавить обработку списка на уникальность действий, т.е. действия без повтора
}

/// <summary>
/// Чтение секции с данными о выходных файлах и данных
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadOutputSection(XmlNodeList xmlNodeList, ref InputData data)
{
    data.Output.Files = new Dictionary<DatafileType, string>();

    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
```

```
            case "path":
                data.Output.Path = node.Attributes["value"].Value;
                break;
            case "datafile":
                data.Output.Files.Add(
                    (DatafileType)Enum.Parse(typeof(DatafileType), node.Attributes["type"].Value),
                    node.Attributes["name"].Value);
                break;
            case "fileprefix":
                data.Output.FilePrefix = node.Attributes["value"].Value;
                break;
        }
    }
}

/// <summary>
/// Чтение секции с опциями и параметрами
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadOptionsSection(XmlNodeList xmlNodeList, ref InputData data)
{
    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "param":
                ParseOptionParam(node.Attributes["name"].Value, node.Attributes["value"].Value, ref data);
                break;
        }
    }
}

/// <summary>
/// Чтение секции с переменными и константами
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadVariablesSection(XmlNodeList xmlNodeList, ref InputData data)
{
    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "var":
                ParseVariableParam(node.Attributes["name"].Value, node.Attributes["value"].Value, ref data);
                break;
        }
    }
}

/// <summary>
/// Чтение секции с параметрами граничных условий
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadBoundarySection(XmlNodeList xmlNodeList, ref InputData data)
{
    data.Boundaries = new List<Boundary>();

    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "condition":
                data.Boundaries.Add(new Boundary()
                {
                    Type = (BoundaryTypes)Enum.Parse(typeof(BoundaryTypes), node.Attributes["type"].Value),
                    BoundaryNumber = Convert.ToInt32(node.Attributes["boundary"].Value),
                    Value = Convert.ToDouble(node.Attributes["value"].Value)
                });
                break;
        }
    }
}

/// <summary>
/// Чтение секции с параметрами размещения частиц (наночастиц)
```

```
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadParticlesSection(XmlNodeList xmlNodeList, ref InputData data)
{
   foreach (XmlNode node in xmlNodeList)
   {
      switch (node.Name)
      {
         case "particle":
            data.Particles.Add(ParseParticleParam(node));
            break;
      }
   }
}

/// <summary>
/// Чтение секции с материалами
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadMaterialsSection(XmlNodeList xmlNodeList, ref InputData data)
{
   data.Materials = new List<Material>();

   // считываем все доступные действия
   foreach (XmlNode node in xmlNodeList)
   {
      switch (node.Name)
      {
         case "material":
            Material mat = new Material()
            {
               Name = node.Attributes["name"].Value,
               Eps = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["eps"].Value)),
               Mu = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["mu"].Value)),
            };
            if (node.Attributes["default"] != null)
            {
               bool val = Convert.ToBoolean(node.Attributes["default"].Value);
               mat.IsDefault = val;
            }
            if (node.Attributes["plasmon"] != null)
            {
               mat.IsPlasmon = true;
               mat.PlasmonMaterial =node.Attributes["plasmon"].Value ;
            }
            data.Materials.Add(mat);
            break;
      }
   }
}

/// <summary>
/// Чтение секции с описанием источников
/// </summary>
/// <param name="xmlNodeList">Часть XML документа</param>
/// <param name="data">Входные данные</param>
private static void ReadSourcesSection(XmlNodeList xmlNodeList, ref InputData data)
{
   data.Sources = new List<Source>();

   foreach (XmlNode node in xmlNodeList)
   {
      switch (node.Name)
      {
         case "source":
            Source src = new Source()
            {
               Type = SourceTypes.Simple,
               Frequency = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["freq"].Value)),
               Lambda = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["lambda"].Value)),
               Amplitude = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["amplitude"].Value)),
               Phase = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["phase"].Value)),
               Direction = (VectorDirection)Enum.Parse(typeof(VectorDirection), node.Attributes["dir"].Value),
               Boundary = Convert.ToInt32(node.Attributes["boundary"].Value)
            };
            src.Point = ReadSourcePoint(node.ChildNodes);
```

```
                data.Sources.Add(src);
                break;
            case "esource":
                data.Sources.Add(new Source()
                {
                    Type = SourceTypes.Electric,
                    Frequency = Convert.ToDouble(node.Attributes["freq"].Value)
                });
                break;
            case "msource":
                data.Sources.Add(new Source()
                {
                    Type = SourceTypes.Magnetic,
                    Frequency = Convert.ToDouble(node.Attributes["freq"].Value)
                });
                break;
        }
    }
}

/// <summary>
/// Чтение секции с описанием параметров геометрии для работы с Netgen
/// </summary>
/// <param name="xmlNodeList">Часть XML файла</param>
/// <param name="data">Входные данные</param>
private static void ReadGeometrySection(XmlNodeList xmlNodeList, ref InputData data)
{
    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "geofile":
                data.Geomerty.GometryFile = node.Attributes["name"].Value;
                break;
            case "meshfile":
                data.Geomerty.MeshFile = node.Attributes["name"].Value;
                break;
            case "domains":
                data.Geomerty.Domains = ReadGeometryDomains(node.ChildNodes);
                break;
            case "isgeneratege omfile":
                data.Geomerty.IsGenerateGeomFile = Convert.ToBoolean(node.Attributes["value"].Value);
                break;
            case "cofactor":
                data.Geomerty.CoFactor = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["value"].Value));
                break;
        }
    }
}

/// <summary>
/// Чтение секции со сведениями о плазмонах
/// </summary>
/// <param name="xmlNodeList">Часть XML документа с данными для разбора</param>
/// <param name="data">Входные данные</param>
/// <remarks>Справочные сведения о используемом в задаче плазмоне!</remarks>
private static void ReadPlasmonsSection(XmlNodeList xmlNodeList, ref InputData data)
{
    data.Plasmons = new List<PlasmonData>();

    // TODO: описать чтение всех параметров плазмона для материлов

    foreach (XmlNode node in xmlNodeList)
    {
        switch (node.Name)
        {
            case "plasmon":
                data.Plasmons.Add(new PlasmonData()
                {
                    Material = (PlasmonMaterials)Enum.Parse(typeof(PlasmonMaterials), node.Attributes["material"].Value),
                    Plasmow = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["plasmow"].Value)),
                    Relax = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["relax"].Value))
                });
                break;
        }
    }
}
```

```
/// <summary>
/// Чтение секции со списоком пробных точек
/// </summary>
/// <param name="xmlNodeList">Часть XML документа с данными для разбора</param>
/// <param name="data">Входные данные</param>
private static void ReadProbesSection(XmlNodeList xmlNodeList, ref InputData data)
{
   data.ProbePoints = new List<ProbePoint>();

   foreach (XmlNode node in xmlNodeList)
   {
      switch (node.Name)
      {
         case "probe":
            data.ProbePoints.Add(new ProbePoint()
            {
               Type = ProbeType.Point,
               Point = new Point(Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["x"].Value)),
                  Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["y"].Value)),
                  Convert.ToDouble(StringHelper.CorrectDoubleStringValue(node.Attributes["z"].Value))),
               Direction = (FieldDirection)Enum.Parse(typeof(FieldDirection), node.Attributes["dir"].Value)
            });
            break;
         case "domain":
            data.ProbePoints.Add(new ProbePoint()
            {
               Type = ProbeType.Domain,
               DomainNumber = Convert.ToInt32(node.Attributes["number"].Value)
            });
            break;
      }
   }
}

#endregion

#region Разбор параметров внутри секций

/// <summary>
/// Разбор параметра из опций
/// </summary>
/// <param name="paramName">Название параметра</param>
/// <param name="paramValue">Значение параметра</param>
/// <param name="data">Входные данные</param>
private static void ParseOptionParam(string paramName, string paramValue, ref InputData data)
{
   switch (paramName)
   {
      case "IsOutputToFile":
         data.Options.IsOutputToFile = Convert.ToBoolean(paramValue);
         break;
      case "IsBuiltEdges":
         data.Options.IsBuiltEdges = Convert.ToBoolean(paramValue);
         break;
      case "IsAppend":
         data.Options.IsAppend = Convert.ToBoolean(paramValue);
         break;
      case "WithParticles":
         data.Options.WithParticles = Convert.ToBoolean(paramValue);
         break;
         // опции работы с Netgen Mesher (препроцессинг)
      case "NetgenParams":
         data.Options.NetgenParams = paramValue;
         break;
      case "InTimeDomain": // решать во временной области (с шагами по времени)
         data.Options.InTimeDomain = Convert.ToBoolean(paramValue);
         break;
      case "InFrequencyDomain": // решать в частотной области (с шагами по частоте работы источника)
         data.Options.InFrequencyDomain = Convert.ToBoolean(paramValue);
         break;
      case "WithoutIterations": // решаем без итераций по варьируемой величине
         data.Options.WithoutIterations = Convert.ToBoolean(paramValue);
         break;
      case "WithEdgeFields":
         data.Options.WithEdgeFields = Convert.ToBoolean(paramValue);
         break;
      case "IsProbeOutput":
         data.Options.IsProbeOutput = Convert.ToBoolean(paramValue);
```

```
            break;
        }
    }

    /// <summary>
    /// Разбор параметра из переменной
    /// </summary>
    /// <param name="paramName">Название переменной (параметра)</param>
    /// <param name="paramValue">Значение переменной (параметра)</param>
    /// <param name="data">Входные данные</param>
    private static void ParseVariableParam(string paramName, string paramValue, ref InputData data)
    {
        switch (paramName)
        {
            case "Time":
                data.Variables.Time = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(paramValue));
                break;
            case "TimeDelta":
                data.Variables.TimeDelta = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(paramValue));
                break;
            case "FrequencyDelta":
                data.Variables.FrequencyDelta = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(paramValue));
                break;
            case "FrequencyStep":
                data.Variables.FrequencyStep = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(paramValue));
                break;
        }
    }

    /// <summary>
    /// Разбор параметра для распределения частиц
    /// </summary>
    /// <param name="node">Узел XML документа с данными</param>
    private static Particle ParseParticleParam(XmlNode node)
    {
        Particle particle = null;
        ParticlesType type = (ParticlesType)Enum.Parse(typeof(ParticlesType), node.Attributes["type"].Value);
        switch (type)
        {
            case ParticlesType.Law:
                ParticleD par = new ParticleD();
                ParseCustomParticleParam<ParticleD>(node, ref par);
                particle = par;
                break;
            case ParticlesType.Single:
                ParticleS par2 = new ParticleS();
                ParseCustomParticleParam<ParticleS>(node, ref par2);
                particle = par2;
                break;
            case ParticlesType.Many:
                ParticleM par3 = new ParticleM();
                ParseCustomParticleParam<ParticleM>(node, ref par3);
                particle = par3;
                break;
        }

        return particle;
    }

    /// <summary>
    /// Разбор аргументов в описании частиц
    /// </summary>
    /// <typeparam name="TType">Тип частиц</typeparam>
    /// <param name="node">Данные узла XML файла</param>
    /// <param name="particle">Данные о частицах</param>
    private static void ParseCustomParticleParam<TType>(XmlNode node, ref TType particle)
    {
        foreach (XmlAttribute attr in node.Attributes)
        {
            switch (attr.Name)
            {
                // общие параметры
                case "material":
                    (particle as Particle).Material = attr.Value;
                    break;
                case "type":
                    (particle as Particle).Type = (ParticlesType)Enum.Parse(typeof(ParticlesType), attr.Value);
                    break;
```

```
                // для распределения частиц
                case "name":
                    (particle as ParticleD).DistName = (ParticlesDistribution)Enum.Parse(typeof(ParticlesDistribution), attr.Value);
                    break;
                case "lawtype":
                    (particle as ParticleD).DistType = (DistributionTypes)Enum.Parse(typeof(DistributionTypes), attr.Value);
                    break;
                case "count":
                    (particle as ParticleD).Count = Convert.ToInt32(attr.Value);
                    break;
                case "lower":
                    (particle as ParticleD).LowerBound = Convert.ToInt32(attr.Value);
                    break;
                case "top":
                    (particle as ParticleD).TopBound = Convert.ToInt32(attr.Value);
                    break;
                // для одной частицы (один КЭ)
                case "number":
                    (particle as ParticleS).Number = Convert.ToInt32(attr.Value);
                    break;
            }
        }
    }

    /// <summary>
    /// Разбор областей геометрии
    /// </summary>
    /// <param name="childNodes">Часть XML файла</param>
    /// <returns>Возвращает список областей геометрии</returns>
    private static List<Domain> ReadGeometryDomains(XmlNodeList xmlNodeList)
    {
        List<Domain> doms = new List<Domain>();

        foreach (XmlNode node in xmlNodeList)
        {
            switch (node.Name)
            {
                case "domain":
                    doms.Add(new Domain()
                    {
                        Code = Convert.ToInt32(node.Attributes["code"].Value),
                        Name = node.Attributes["name"].Value,
                        MaterialName = node.Attributes["material"].Value
                    });
                    break;
            }
        }

        return doms;
    }

    #endregion

    #region Вспомогательные функции

    /// <summary>
    /// Чтение блока с координатами источника
    /// </summary>
    /// <param name="childNodes">Часть XML документа</param>
    /// <returns>Возвращает считанную точку Point</returns>
    private static Point ReadSourcePoint(XmlNodeList xmlChildNodes)
    {
        Point point = new Point(0, 0, 0);

        if (xmlChildNodes[0].Name == "point")
        {
            point.X = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(xmlChildNodes[0].Attributes["x"].Value));
            point.Y = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(xmlChildNodes[0].Attributes["y"].Value));
            point.Z = Convert.ToDouble(StringHelper.CorrectDoubleStringValue(xmlChildNodes[0].Attributes["z"].Value));
        }

        return point;
    }

    #endregion
  }
}
```

AppContoller.cs:

```
using System;
using System.Diagnostics;
using System.Linq;
using NLog;
using EMFFM.MainApp.Common;
using EMFFM.MainApp.Helpers;
using EMFFM.MainApp.Input;
using System.Collections.Generic;

namespace EMFFM.MainApp.Controllers
{
   /// <summary>
   /// Главный класс для управления работой программы
   /// </summary>
   class AppController
   {
      readonly Logger logger = LogManager.GetCurrentClassLogger();

      InputData _data; // исходные данные из файла для работы контроллера

      Dictionary<ActionNames, Action<InputData>> _actions;

      /// <summary>
      /// Инициализация контроллера программы
      /// </summary>
      /// <param name="data">Входные данные из файла с данными</param>
      public AppController(InputData data)
      {
         _data = data;
         SetAssociations(data.Task);
      }

      /// <summary>
      /// Установка словаря ассоциация действий InputAPI с функциями их исполняющими
      /// </summary>
      private void SetAssociations(TaskClasses task)
      {
         logger.Debug("Initialize all actions!");

         // TODO: описание привязки инициализации по типу задачи в контроллере
         // NOTE: по сути здесь для аждого типа задачи выбирается свой список ассоциаций, единственная зависимость контроллера

         switch(task)
         {
             case TaskClasses.Common:
               _actions = ActionHelper.CommonTasks();
               break;
            case TaskClasses.Netgen:
               _actions = ActionHelper.NetgenTasks();
               break;
         }
      }

      /// <summary>
      /// Запуск обработки всех действий
      /// </summary>
      public void Run()
      {
         // выполняем действия в порядке их расположения в списке (по возрастанию кода)
         foreach (var item in _data.Actions)
         {
            Run(item);
         }
      }

      /// <summary>
      /// Запуск на обработку определенного действия
      /// </summary>
      /// <param name="action">Действие для выполнения</param>
      public void Run(EMFFM.MainApp.Input.Action action)
      {
         logger.Info("Run action {0}", action);

         // засекаем время работы
         Stopwatch time = new Stopwatch();
         time.Start();
```

```
            _actions[action.Value](_data);

            time.Stop();

            logger.Info("Ends action {0} for {1} ms", action, time.ElapsedMilliseconds);
        }
    }
}
```

ActionOperations.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using EMFFM.MainApp.Common;
using EMFFM.MainApp.FEM;
using EMFFM.MainApp.FEM.Elements;
using EMFFM.MainApp.Mesh;
using EMFFM.MainApp.Input;
using EMFFM.MainApp.Helpers;
using EMFFM.MainApp.Mesh.Elements;

namespace EMFFM.MainApp.Controllers
{
    /// <summary>
    /// Класс с методами, описывающими последовательности операций для действий
    /// </summary>
    class ActionOperations
    {
        // NOTE: здесь в зависимости от ассоциаций и будут определятся классы решаемых задач! Т.е. какое действие будет присвоено для его выполнения.

        #region Действия для типа Common
        /// <summary>
        /// Выполнение последовательности действий для генерации сетки
        /// </summary>
        /// <param name="data">Исходные данные для действия</param>
        /// <remarks>Используется встроенный генератор сетки (простые ругулярные ортогональные)</remarks>
        public static void GenerateMeshAction(InputData data)
        {
            // 1. Подготовка объекта для описания геометрии области дискретизации
            GeomData geomData = InputHelper.GetGeomData(data);

            // 2. Определение типа сетки и выбор соответсвующего класса для ее генерации
            switch (data.Element.Type)
            {
                case ElementType.Rectangle: // генерация сетки для прямоугольников
                    MeshBuilder.GenerateRectangleMesh(geomData, data.Output, data.Options);
                    break;
                case ElementType.Triangle: // генерация сетки для треугольников
                    MeshBuilder.GenerateTriangleMesh(geomData, data.Output, data.Options);
                    break;
                case ElementType.Parallelepiped: // генерация сетки для параллелепипедов
                    MeshBuilder.GenerateParallepipedMesh(geomData, data.Output, data.Options);
                    break;
                case ElementType.Tetrahedron: // генерация сетки для тетраэдров
                    MeshBuilder.GenerateTetrahedronMesh(geomData, data.Output, data.Options);
                    break;
            }
        }

        /// <summary>
        /// Выполнение последовательности действий для генерации частиц
        /// </summary>
        /// <param name="data">Исходные данные для действия</param>
        public static void GenerateParticlesAction(InputData data)
        {
            // TODO: описать метод действия для генерации частиц (GenerateParticle)
            Particles.ParticleBuilder pb = new Particles.ParticleBuilder(data.Particles);
            int[] nums = pb.Numbers;

            if (data.Options.IsOutputToFile)
            {
                Output.OutputHelper.OutputVectorToFile<int>(nums, data.Output.GetFullPath(DatafileType.Particles));
            }
        }
```

```
        /// <summary>
        /// Построение глобальной матрицы
        /// </summary>
        /// <param name="data">Исходные даннве для действия</param>
        /// <remarks>Тестовый метод как и само действие!</remarks>
        public static void BuiltGlobalMatrixAction(InputData data)
        {
            // 1. Задание геометрии для разбиения
            GeomData geomData = InputHelper.GetGeomData(data);

            double[,] matrix = new double[0, 0];

            // 2. Определение типа сетки и выбор соответсвующего класса для ее генерации
            switch (data.Element.Type)
            {
                case ElementType.Rectangle: // генерация сетки для прямоугольников
                    matrix = BuiltGlobalMatrix<Rectangle, RectMesh>(data, new RectMesh(geomData));

                    matrix = BoundaryBuilder.ZeroBoundaryEdges(matrix, new int[] { 1, 2 });
                    break;
                case ElementType.Triangle: // генерация сетки для треугольников
                    matrix = BuiltGlobalMatrix<Triangle, TriMesh>(data, new TriMesh(geomData, TriangleStyle.Left));
                    break;
                case ElementType.Parallelepiped: // генерация сетки для параллелепипедов
                    matrix = BuiltGlobalMatrix<Parallelepiped, ParMesh>(data, new ParMesh(geomData));
                    break;
                case ElementType.Tetrahedron: // генерация сетки для тетраэдров
                    matrix = BuiltGlobalMatrix<Tetrahedron, TetMesh>(data, new TetMesh(geomData));
                    break;
            }

            if (data.Options.IsOutputToFile)
            {
                Output.OutputHelper.OutputMatrixToFile<double>(matrix, data.Output.GetFullPath(DatafileType.Matrix));
            }
        }
        #endregion
        #region Действия для задачи типа Netgen
        /// <summary>
        /// Генерация сетки с использованием программы Netgen
        /// </summary>
        /// <param name="data">Входные данные</param>
        public static void GenerateNetgenMesh(InputData data)
        {
            string fileName = Helpers.StringHelper.GetFileNameWithoutExtension(data.Geomerty.GometryFile);
            // используем генератор сеток Netgen
            NetgenMeshHelper.GenerateNetgenMesh(data, fileName);
        }

        /// <summary>
        /// Генерация глобальной матрицы на основании сетки из Netgen
        /// </summary>
        /// <param name="data">Входные данные</param>
        public static void BuiltGlobalMatrixNetgen(InputData data)
        {
            // 1. Генерация сетки
            string fileName = Helpers.StringHelper.GetFileNameWithoutExtension(data.Geomerty.GometryFile);
            // используем генератор сеток Netgen
            TMesh<Tetrahedron, Triangle> mesh = NetgenMeshHelper.GenerateNetgenMesh(data, fileName);

            // 2. Построение глобальной матрицы
            double[,] matrix = BuiltGlobalMatrixNetgen(data, mesh);

            if (data.Options.IsOutputToFile)
            {
                // NOTE: проба параллельной либы через таски
                var task = new Task(() => Output.OutputHelper.OutputMatrixToFile<double>(matrix,
data.Output.GetFullPath(DatafileType.Matrix)));
                task.Start();
            }
        }

        /// <summary>
        /// Решения задачи с использованием программы Netgen
        /// </summary>
        /// <param name="data">Входные данные</param>
        public static void SolveNetgen(InputData data)
        {
```

```
        // 1. Генерация сетки
        string fileName = Helpers.StringHelper.GetFileNameWithoutExtension(data.Geomerty.GometryFile);
        // используем генератор сеток Netgen
        TMesh<Tetrahedron, Triangle> mesh = NetgenMeshHelper.GenerateNetgenMesh(data, fileName);

        // формируем объект для построения глобальных матриц
        GlobalMatrix<Tetrahedron, Triangle> matrix = new GlobalMatrix<Tetrahedron, Triangle>(data, mesh);

        // формируем объект для построения векторов в воздействиями (значения падающего поля)
        WaveData wave = InputHelper.GetWaveData(data); // формируем сведения о волне от источника
        List<Edge> boundSource = MeshHelper.GetBoundaryEdges<Tetrahedron, Triangle>(mesh, data.Sources.Single(s => s.Type == SourceTypes.Simple).Boundary);
        ForceVector vector = new ForceVector(wave, SourceFunctions.MonoWave, mesh.EdgesCount, boundSource);

        // 2. Решение задачи
        int[] boundNumsBoundary = MeshHelper.GetNumbersOfBoundaryEdges<Tetrahedron, Triangle>(mesh, data.Boundaries.Single(b => b.Type == BoundaryTypes.Dirichlet).BoundaryNumber);
        Solver<Tetrahedron, Triangle> solver = new Solver<Tetrahedron, Triangle>(data, wave, matrix, vector, boundNumsBoundary);
        solver.Run(); // запуск

        if (data.Options.IsOutputToFile)
        {
            int count = 0;
             foreach (var item in solver.Results)
            {
                item.OutputToFile(String.Format("{0}resultdata-{1}.txt", data.Output.Path, count));
                count++;
            }
        }

        // 3. Обработка результатов решения
        FieldAnalysis<Tetrahedron, Triangle> analysis = new FieldAnalysis<Tetrahedron, Triangle>(solver.Results, mesh, ElementType.Tetrahedron);
        //analysis.CalculateDifference();
        analysis.CalculateFields(data.Geomerty.CoFactor);
        analysis.PrintElementsField(data.Output.Path, data.Output.FilePrefix, data.Options.WithEdgeFields);

        // 4. Вывод результатов на основе указанных проб
        if (data.Options.IsProbeOutput)
        {
            foreach (var item in data.ProbePoints)
            {
                if (item.Type == ProbeType.Domain)
                {
                    int[] nums = MeshHelper.GetElementsByDomain(mesh, item.DomainNumber);
                    analysis.PrintElementsField(nums, data.Output.Path, data.Output.FilePrefix, data.Options.WithEdgeFields, item.DomainNumber);
                }
            }
        }
    }
    #endregion

    #region Вспомогательные методы
    /// <summary>
    /// Генерация глобальной матрицы в зависимости от типов элементов
    /// </summary>
    /// <typeparam name="TType">Тип элементов сетки</typeparam>
    /// <typeparam name="TType2">Класс генератор сетки</typeparam>
    /// <param name="data">Входные данные</param>
    /// <param name="meshClass">Экземпляр класса для генерации сетки</param>
    /// <returns>Возвращает глобальную матрицу</returns>
    /// <remarks>Генерирует с учетом частиц (и без них), а также с параметром K</remarks>
    private static double[,] BuiltGlobalMatrix<TType, TType2>(InputData data, TType2 meshClass)
    {
        // NOTE: не используется!
        List<TType> elements = (meshClass as IMesh<TType>).GenerateMesh();
        int size = EdgeMesh.BuiltEdges(ref elements);

        if (data.Options.WithParticles)
        {
            Particles.ParticleBuilder pb = new Particles.ParticleBuilder(data.Particles);
            elements = MaterialsBuilder.SetMaterials(elements, data.Materials, pb.Numbers, data.Particles);
        }
        else
        {
            elements = MaterialsBuilder.SetDefaultMaterials(elements, data.Materials.Single(m => m.IsDefault == true));
        }
```

```
            double freq = ParametersBuilder.GetFrequency(data.Sources, SourceTypes.Simple);
            double K = ParametersBuilder.CalculateWaveNumber(freq);
            double omega = ParametersBuilder.CalculateOmega(freq);

            return MatrixBuilder.BuiltGlobalMatrix(size, elements, K, omega);
        }

        /// <summary>
        /// Генерация глобальной матрицы для сетки из Netgen
        /// </summary>
        /// <typeparam name="TType">Тип элементов сетки</typeparam>
        /// <param name="data">Входные данные</param>
        /// <param name="meshData">Построенная сетка</param>
        /// <returns>Возвращает глобальную матрицу</returns>
        private static double[,] BuiltGlobalMatrixNetgen<TType, TType2>(InputData data, TMesh<TType, TType2> meshData)
        {
            // NOTE: не используется!
            MaterialsBuilder.SetMaterialsByDomain(ref meshData, data.Geomerty, data.Materials, data.Plasmons);

            double freq = ParametersBuilder.GetFrequency(data.Sources, SourceTypes.Simple);
            double K = ParametersBuilder.CalculateWaveNumber(freq);
            double omega = ParametersBuilder.CalculateOmega(freq);

            return MatrixBuilder.BuiltGlobalMatrix(meshData.EdgesCount, meshData.MeshElements, K, omega);
        }
        #endregion
    }
}
```

TMesh.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.Common;

namespace EMFFM.MainApp.Mesh
{
    /// <summary>
    /// Описание сетки
    /// </summary>
    /// <remarks>Собственно все данные о сетке, которая используется для решения</remarks>
    class TMesh<TType, TType2>
    {
        public ElementType ElementType;

        /// <summary>
        /// Список элементов сетки
        /// </summary>
        /// <remarks>Объемные элементы (тетраэдры)</remarks>
        public List<TType> MeshElements;

        /// <summary>
        /// Список поверхностных элементов
        /// </summary>
        /// <remarks>Поверхностные элементы (треугольники)</remarks>
        public List<TType2> SurfElements;

        /// <summary>
        /// Количество ребер в сетке всего
        /// </summary>
        public int EdgesCount;

        /// <summary>
        /// Количество ребер на гранях
        /// </summary>
        /// <remarks>Все ребра для всех плоскостей!</remarks>
        public int BoundaryEdgesCount;

        /// <summary>
        /// Инициализация хранилища сетки
        /// </summary>
        /// <param name="type">Тип элементов сетки</param>
        public TMesh(ElementType type)
        {
            ElementType = type;
        }

        /// <summary>
```

```
        /// Инициализация хранилица сетки
        /// </summary>
        /// <param name="type">Тип элементов сетки</param>
        /// <param name="elements">Список элементов</param>
        /// <param name="edgesCount">Количество ребер</param>
        public TMesh(ElementType type, List<TType> elements, int edgesCount)
        {
            ElementType = type;
            MeshElements = elements;
            EdgesCount = edgesCount;
        }
    }
}
```

Element.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using EMFFM.MainApp.Input;
using EMFFM.MainApp.Math;

namespace EMFFM.MainApp.Mesh.Elements
{
    /// <summary>
    /// Базовый класс для всех элементов
    /// </summary>
    class Element
    {
        /// <summary>
        /// Узлы элемента
        /// </summary>
        public List<Node> Nodes { get; set; }

        /// <summary>
        /// Ребра элемента
        /// </summary>
        public List<Edge> Edges { get; set; }

        /// <summary>
        /// Материал элемента
        /// </summary>
        /// <remarks>Электромагнитные параметры области, занимаемой элементом</remarks>
        public Material Material { get; set; }

        /// <summary>
        /// Номер элемента
        /// </summary>
        public int Number { get; set; }

        /// <summary>
        /// Номер области в описываемой задаче
        /// </summary>
        /// <remarks>Область из генерации сетки с помощью Netgen</remarks>
        public int SubDomain { get; set; }

        /// <summary>
        /// Номер граничного условия
        /// </summary>
        /// <remarks>Для поверхностных плоских элементов!</remarks>
        public int BoundaryCond { get; set; }

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendFormat("{0} [", Number); // номер элемента
            // записываем список узлов элемента
            for (int i = 0; i < Nodes.Count; i++)
            {
                if (i != Nodes.Count - 1)
                {
                    sb.AppendFormat("{0}, ", Nodes[i].ToString());
                }
                else
                {
                    sb.AppendFormat("{0}", Nodes[i].ToString());
                }
            }
```

```
            sb.Append("]");

            return sb.ToString();
        }

        /// <summary>
        /// Генерация строки с данными о ребрах элемента
        /// </summary>
        public string EdgesToString()
        {
            StringBuilder sb = new StringBuilder();

            sb.AppendFormat("[ ");
            for (int i = 0; i < Edges.Count; i++)
            {
                if (i != Edges.Count - 1)
                {
                    sb.AppendFormat("{0}, ", Edges[i].ToString());
                }
                else
                {
                    sb.AppendFormat("{0}", Edges[i].ToString());
                }
            }
            sb.Append("]");

            return sb.ToString();
        }

        public virtual void AssociateEdges()
        {
        }

        /// <summary>
        /// Определение центра элемента
        /// </summary>
        /// <returns>Возвращает точку, расположенную в центре элемента</returns>
        public Point GetCenterPoint()
        {
            int count = Nodes.Count; // число вершин

            double x = MathHelper.SumOfCooodinates(Nodes, 0) / count;
            double y = MathHelper.SumOfCooodinates(Nodes, 1) / count;
            double z = MathHelper.SumOfCooodinates(Nodes, 2) / count;

            return new Point(x, y, z);
        }
    }
}
```

Tetrahedron.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.Math;

namespace EMFFM.MainApp.Mesh.Elements
{
    /// <summary>
    /// Тетраэдр
    /// </summary>
    class Tetrahedron : Element, IElement, I3DElement
    {
        #region Конструкторы

        /// <summary>
        /// Инициализация "пустого" элемента
        /// </summary>
        public Tetrahedron()
        {
            // инициализируем список узлов в размером в 3 узла
            Nodes = new List<Node>(4);
            Number = -1; // по умолчанию для пустого элемента номер его -1
        }

        /// <summary>
        /// Инициализируем элемент с данными об узлах
        /// </summary>
```

```
/// <param name="nodes">Узлы элемента</param>
/// <param name="number">Номер элемента</param>
public Tetrahedron(List<Node> nodes, int number)
{
   Nodes = nodes;
   Number = number;
}

/// <summary>
/// Инициализируем элемент с данными об узлах
/// </summary>
/// <param name="nodes">Узлы элемента</param>
/// <param name="number">Номер элемента</param>
/// <param name="subdomain">Область, где расположен элемент</param>
public Tetrahedron(List<Node> nodes, int number, int subdomain)
{
   Nodes = nodes;
   Number = number;
   SubDomain = subdomain;
}

/// <summary>
/// Копирование элемента
/// </summary>
/// <param name="tr">Тетраэдра для копирования</param>
public Tetrahedron(Tetrahedron tr)
{
   this.Nodes = tr.Nodes;
   this.Number = tr.Number;
}

#endregion

#region Реализация интерфейса I3DElement

/// <summary>
/// Определение объема элемента
/// </summary>
/// <returns>Возвращает значение объема</returns>
/// <remarks>Вычисление выполняется на основе определителя матрицы!(длинная формула)</remarks>
public double GetVolume()
{
   // определение векторов с общим узлом в точке 1 тетраэдра
   TVector v1 = new TVector(Nodes[1].Coord, Nodes[0].Coord); // 2-1
   TVector v2 = new TVector(Nodes[2].Coord, Nodes[0].Coord); // 3-1
   TVector v3 = new TVector(Nodes[3].Coord, Nodes[0].Coord); // 4-1

   // вычисление смешанного произведения векторов
   return System.Math.Abs((1.0 / 6.0) * TVector.VectorMult(v1, v2, v3));
}

#endregion

/// <summary>
/// Ассоциация ребер элемента с его узлами
/// </summary>
/// <remarks>На основании заданной локальной нумерации ребер на элементе!
/// По книге и по проге ЕМАР3!</remarks>
public override void AssociateEdges()
{
   Edges = new List<Edge>(6);
   Edges.Add(new Edge(new Node(Nodes[0], 1), new Node(Nodes[1], 2))); // 1-2
   Edges.Add(new Edge(new Node(Nodes[0], 1), new Node(Nodes[2], 3))); // 1-3
   Edges.Add(new Edge(new Node(Nodes[0], 1), new Node(Nodes[3], 4))); // 1-4
   Edges.Add(new Edge(new Node(Nodes[1], 2), new Node(Nodes[2], 3))); // 2-3
   Edges.Add(new Edge(new Node(Nodes[3], 4), new Node(Nodes[1], 2))); // 4-2
   Edges.Add(new Edge(new Node(Nodes[2], 3), new Node(Nodes[3], 4))); // 3-4
   // добавим локальную нумерацию
   Edges[0].LocalNumber = 1;
   Edges[1].LocalNumber = 2;
   Edges[2].LocalNumber = 3;
   Edges[3].LocalNumber = 4;
   Edges[4].LocalNumber = 5;
   Edges[5].LocalNumber = 6;
}

#region Вычисление локальных матриц
```

```
double[,] CoFactor;
double SizeCoFactor;

/// <summary>
/// Вычисление локальной матрицы Ek для элемента
/// </summary>
/// <returns>Возвращает вычисленную матрицу элемента</returns>
/// <remarks>Взято из EMAP3!</remarks>
public double[,] GetLocalE()
{
   double[,] S_mat = new double[6, 6];

   int i, j;
   double Length_i, Length_j, Mult1, Volume;
   {
      Volume = GetVolume() * SizeCoFactor;
      Mult1 = (1296.0 * Volume * Volume * Volume);
      for (i = 0; i <= 5; i++)
         for (j = 0; j <= 5; j++)
         {
            Length_i = Math.Geometry.GetDistance(Edges[i].Vertex1.Coord, Edges[i].Vertex2.Coord, SizeCoFactor);
            Length_j = Math.Geometry.GetDistance(Edges[j].Vertex1.Coord, Edges[j].Vertex2.Coord, SizeCoFactor);

            S_mat[i, j] = (CoFactor[2, Edges[i].Vertex1.LocalNumber - 1] * CoFactor[3, Edges[i].Vertex2.LocalNumber - 1] -
                           CoFactor[3, Edges[i].Vertex1.LocalNumber - 1] * CoFactor[2, Edges[i].Vertex2.LocalNumber - 1]) *
                          (CoFactor[2, Edges[j].Vertex1.LocalNumber - 1] * CoFactor[3, Edges[j].Vertex2.LocalNumber - 1] -
                           CoFactor[3, Edges[j].Vertex1.LocalNumber - 1] * CoFactor[2, Edges[j].Vertex2.LocalNumber - 1]) +
                          (CoFactor[3, Edges[i].Vertex1.LocalNumber - 1] * CoFactor[1, Edges[i].Vertex2.LocalNumber - 1] -
                           CoFactor[1, Edges[i].Vertex1.LocalNumber - 1] * CoFactor[3, Edges[i].Vertex2.LocalNumber - 1]) *
                          (CoFactor[3, Edges[j].Vertex1.LocalNumber - 1] * CoFactor[1, Edges[j].Vertex2.LocalNumber - 1] -
                           CoFactor[1, Edges[j].Vertex1.LocalNumber - 1] * CoFactor[3, Edges[j].Vertex2.LocalNumber - 1]) +
                          (CoFactor[1, Edges[i].Vertex1.LocalNumber - 1] * CoFactor[2, Edges[i].Vertex2.LocalNumber - 1] -
                           CoFactor[2, Edges[i].Vertex1.LocalNumber - 1] * CoFactor[1, Edges[i].Vertex2.LocalNumber - 1]) *
                          (CoFactor[1, Edges[j].Vertex1.LocalNumber - 1] * CoFactor[2, Edges[j].Vertex2.LocalNumber - 1] -
                           CoFactor[2, Edges[j].Vertex1.LocalNumber - 1] * CoFactor[1, Edges[j].Vertex2.LocalNumber - 1]);

            S_mat[i, j] = (Length_i * Length_j * S_mat[i, j]) / Mult1;
            S_mat[i, j] *= 4;
         }
   }

   return S_mat;
}

/// <summary>
/// Вычисление локальной матрицы Fk для элемента
/// </summary>
/// <returns>Возвращает вычисленную матрицу элемента</returns>
/// <remarks>Взято из EMAP3!</remarks>
public double[,] GetLocalF()
{
   double TetVolume = GetVolume() * SizeCoFactor; // определяем объем тетраэдра

   int i, j;
   double Length_i, Length_j; // длины ребер элемента

   double[,] T_mat = new double[6, 6];

   double Mult1 = 1.0 / (720.0 * TetVolume);
   // вычисление частей элементов с кофакторами
   T_mat[0, 0] = FF(1, 1) + FF(2, 2) - FF(1, 2);
   T_mat[0, 1] = 2 * FF(2, 3) - FF(2, 1) - FF(1, 3) + FF(1, 1);
   T_mat[0, 2] = 2 * FF(2, 4) - FF(2, 1) - FF(1, 4) + FF(1, 1);
   T_mat[0, 3] = -2 * FF(1, 3) - FF(2, 2) + FF(2, 3) + FF(1, 2);
   T_mat[0, 4] = 2 * FF(1, 4) - FF(2, 4) - FF(1, 2) + FF(2, 2);
   T_mat[0, 5] = FF(2, 4) - FF(2, 3) - FF(1, 4) + FF(1, 3);
   T_mat[1, 1] = FF(1, 1) + FF(3, 3) - FF(1, 3);
   T_mat[1, 2] = 2 * FF(3, 4) - FF(1, 3) - FF(1, 4) + FF(1, 1);
   T_mat[1, 3] = 2 * FF(1, 2) - FF(2, 3) - FF(1, 3) + FF(3, 3);
   T_mat[1, 4] = FF(2, 3) - FF(3, 4) - FF(1, 2) + FF(1, 4);
   T_mat[1, 5] = -2 * FF(1, 4) - FF(3, 3) + FF(1, 3) + FF(3, 4);
   T_mat[2, 2] = FF(1, 1) + FF(4, 4) - FF(1, 4);
   T_mat[2, 3] = FF(3, 4) - FF(2, 4) - FF(1, 3) + FF(1, 2);
   T_mat[2, 4] = -2 * FF(1, 2) - FF(4, 4) + FF(1, 4) + FF(2, 4);
   T_mat[2, 5] = 2 * FF(1, 3) - FF(3, 4) - FF(1, 4) + FF(4, 4);
   T_mat[3, 3] = FF(3, 3) + FF(2, 2) - FF(2, 3);
   T_mat[3, 4] = -2 * FF(3, 4) - FF(2, 2) + FF(2, 3) + FF(2, 4);
   T_mat[3, 5] = -2 * FF(2, 4) - FF(3, 3) + FF(2, 3) + FF(3, 4);
```

```
    T_mat[4, 4] = FF(2, 2) + FF(4, 4) - FF(2, 4);
    T_mat[4, 5] = -2 * FF(2, 3) - FF(4, 4) + FF(2, 4) + FF(3, 4);
    T_mat[5, 5] = FF(3, 3) + FF(4, 4) - FF(3, 4);
    // копирование симметричной части матрицы
    for (i = 0; i <= 5; i++)
        for (j = 0; j <= i - 1; j++)
        {
            T_mat[i, j] = T_mat[j, i];
        }

    for (i = 0; i <= 5; i++)
        for (j = 0; j <= 5; j++)
        {
            Length_i = Math.Geometry.GetDistance(Edges[i].Vertex1.Coord, Edges[i].Vertex2.Coord, SizeCoFactor);
            Length_j = Math.Geometry.GetDistance(Edges[j].Vertex1.Coord, Edges[j].Vertex2.Coord, SizeCoFactor);

            if (i == j) // диагональные элементы , чтобы получилось 360 в знаменателе, а не 720
                T_mat[i, j] = 2 * Length_i * Length_j * Mult1 * T_mat[i, j];
            else
                T_mat[i, j] = Length_i * Length_j * Mult1 * T_mat[i, j];
            // домножение на Epsilon осуществляется в методе сборке локальной матрицы
            // T_mat[i, j] = RELPerm[HexNum,0] * T_mat[i,j];
            //T_mat[i, j] = WaveNumber * WaveNumber * T_mat[i][j];
            // домножение на волновое число осуществляется в общем методе сборки локальной матрицы
        }

    return T_mat;
}

/// <summary>
/// Подготовительные операции для вычисления локальных матриц
/// </summary>
/// <param name="coFactor">Размерный кофактор для нормирования</param>
public void Prerequsite(double coFactor)
{
    SizeCoFactor = coFactor;
    // вычисление кофакторов для элемента
    CoFactor = ComputeTetraCoFactor();
}

#endregion

#region Вспомогательные методы

/// <summary>
/// Вычисление кофакторов для тетраэдра
/// </summary>
/// <returns>Возвращает матрицу кофакторов [4x4]</returns>
/// <remarks>Код взят из EMAP3!</remarks>
private double[,] ComputeTetraCoFactor()
{
    int i, j, RowNum, ColNum, Row, Col = 0, Count = 0;
    double[,] CoF = new double[4, 4];
    double[,] Buff = new double[4, 4];
    double[,] CoFs = new double[4, 4];

    /* This just involves some basic matrix simulations */
    for (i = 0; i <= 3; i++)
    {
        for (j = 0; j <= 3; j++)
        {
            if (j == 3)
                Buff[i, j] = 1.0;
            else
                Buff[i, j] = Nodes[i].Coord[j]; //Buff[i,j] = NodeCord[TetGlobalNodeNum[t][i] - 1][j];
        }
    }
    /* The main matrix is made up of a column of 1's and the other columns are made of the node coordinates */

    /* The co-factor matrices are formed from the main matrix by taking part of the matrix */
    for (RowNum = 0; RowNum <= 3; RowNum++)
    {
        for (ColNum = 0; ColNum <= 3; ColNum++)
        {
            /* The next few steps calculate the co-factors by taking the determinant of the sub-matrix */
            Row = 0;
            Col = 0;
            Count = 0;
```

```
                for (i = 0; i <= 3; i++)
                    for (j = 0; j <= 3; j++)
                    {
                        if ((i != RowNum) && (j != ColNum))
                        {
                            Count++;
                            if (Count < 4)
                                Row = 0;
                            if ((Count > 3) && (Count < 7))
                                Row = 1;
                            else if (Count > 6)
                                Row = 2;
                            CoF[Row, Col] = Buff[i, j];
                            Col++;
                            Col = Col % 3;
                        }
                    }
                Row = 0;
                Col = 0;
                Count = 0;
                CoFs[RowNum, ColNum] = CoF[0, 0] * ((CoF[1, 1] * CoF[2, 2]) - (CoF[1, 2] * CoF[2, 1])) -
                                       CoF[0, 1] * ((CoF[1, 0] * CoF[2, 2]) - (CoF[2, 0] * CoF[1, 2])) +
                                       CoF[0, 2] * ((CoF[1, 0] * CoF[2, 1]) - (CoF[2, 0] * CoF[1, 1]));
                if ((RowNum + ColNum) % 2 != 0)
                    CoFs[RowNum, ColNum] *= -1.0;
            }
        }
        double[,] CoFactor = new double[4, 4];
        for (RowNum = 0; RowNum <= 3; RowNum++)
        {
            if (RowNum == 0)
                Col = 3;
            else if (RowNum == 1)
                Col = 0;
            else if (RowNum == 2)
                Col = 1;
            else if (RowNum == 3)
                Col = 2;
            for (ColNum = 0; ColNum <= 3; ColNum++)
            {
                CoFactor[RowNum, ColNum] = -CoFs[ColNum, Col];
            }
        }
        return CoFactor;
    }

    /// <summary>
    /// Вычисление Fij для элемента
    /// </summary>
    /// <param name="i">Ребро i</param>
    /// <param name="j">Ребро j</param>
    /// <returns>Возвращает значение Fij</returns>
    /// <remarks>Взято из EMAP3!</remarks>
    private double FF(int i, int j)
    {
        if (CoFactor != null)
        {
            return CoFactor[1, i - 1] * CoFactor[1, j - 1] + CoFactor[2, i - 1] * CoFactor[2, j - 1] +
                CoFactor[3, i - 1] * CoFactor[3, j - 1];
        }
        else
            return 0;
    }
    #endregion
  }
}
```

Point.cs:

```
using System;
using System.Linq;

namespace EMFFM.MainApp.Mesh.Elements
{
  /// <summary>
  /// Описание точки в пространстве
  /// </summary>
  public class Point : IEquatable<Point>
  {
```

```
/// <summary>
/// Координата X
/// </summary>
public double X;
/// <summary>
/// Координата Y
/// </summary>
public double Y;
/// <summary>
/// Координата Z
/// </summary>
public double Z;

#region Конструкторы

/// <summary>
/// Инициализация точки 2D
/// </summary>
/// <param name="x">Координата X</param>
/// <param name="y">Координанта Y</param>
public Point(double x, double y)
{
   X = x;
   Y = y;
   Z = 0;
}

/// <summary>
/// Инициализация точки в 3D
/// </summary>
/// <param name="x">Координата X</param>
/// <param name="y">Координата Y</param>
/// <param name="z">Координата Z</param>
public Point(double x, double y, double z)
{
   X = x;
   Y = y;
   Z = z;
}

/// <summary>
/// Копирование координат точки
/// </summary>
/// <param name="p">Точка</param>
public Point(Point p)
{
   X = p.X;
   Y = p.Y;
   Z = p.Z;
}

#endregion

public bool Equals(Point other)
{
   return this.X == other.X && this.Y == other.Y && this.Z == other.Z;
}

public override string ToString()
{
   if (Z == 0)
   {
      return String.Format("( {0}, {1} )", X, Y);
   }
   else
   {
      return String.Format("( {0}, {1}, {2} )", X, Y, Z);
   }
}

public double this[int ind]
{
   get
   {
      double result = 0;
      switch (ind)
      {
         case 0:
```

```
                result = X;
                break;
            case 1:
                result = Y;
                break;
            case 2:
                result = Z;
                break;
        }
        return result;
    }
}

#region Перегрузка операторов

public static bool operator ==(Point lhs, Point rhs)
{
    if (lhs.X == rhs.X && lhs.Y == rhs.Y && lhs.Z == rhs.Z)
        return true;
    else
        return false;
}

public static bool operator !=(Point lhs, Point rhs)
{
    return !(lhs == rhs);
}

#endregion
}
}
```

Node.cs:

```
using System;
using System.Linq;

namespace EMFFM.MainApp.Mesh.Elements
{
    /// <summary>
    /// Описание узла сетки
    /// </summary>
    public class Node : IEquatable<Node>
    {
        /// <summary>
        /// Координаты узла
        /// </summary>
        public Point Coord { get; set; }

        /// <summary>
        /// Номер узла (глобальный)
        /// </summary>
        public int Number { get; set; }

        /// <summary>
        /// Локальный номер узла (в элементе)
        /// </summary>
        public int LocalNumber { get; set; }

        #region Конструкторы

        /// <summary>
        /// Инициализация узла
        /// </summary>
        /// <param name="p">Координаты узла</param>
        /// <param name="num">Номер узла</param>
        public Node(Point p, int num)
        {
            Coord = p;
            Number = num;
        }

        /// <summary>
        /// Инициализация узла в 2D
        /// </summary>
        /// <param name="x">Координата X</param>
        /// <param name="y">Координаната Y</param>
        /// <param name="num">Номер узла</param>
        public Node(double x, double y, int num)
```

```
{
   Coord = new Point(x, y, 0);
   Number = num;
}

/// <summary>
/// Инициализация узла в 3D
/// </summary>
/// <param name="x">Координата X</param>
/// <param name="y">Координата Y</param>
/// <param name="z">Координата Z</param>
/// <param name="num">Номер узла</param>
public Node(double x, double y, double z, int num)
{
   Coord = new Point(x, y, z);
   Number = num;
}

/// <summary>
/// Копирование узла
/// </summary>
/// <param name="node">Узел</param>
public Node(Node node)
{
   Coord = node.Coord;
   Number = node.Number;
   LocalNumber = node.LocalNumber;
}

/// <summary>
/// Копирование узла
/// </summary>
/// <param name="node">Узел</param>
/// <param name="localNumber">Локальный номер узла</param>
public Node(Node node, int localNumber)
{
   Coord = node.Coord;
   Number = node.Number;
   LocalNumber = localNumber;
}

#endregion

public bool Equals(Node other)
{
   return this.Coord == other.Coord && this.Number == other.Number;
}

public override bool Equals(object obj)
{
   return this.Equals(obj as Node);
}

public override int GetHashCode()
{
   return Number.GetHashCode();
}

public override string ToString()
{
   return String.Format("{0} {1}", Number, Coord.ToString());
}

#region Перегрузка операторов

public static bool operator ==(Node lhs, Node rhs)
{
   if (lhs.Coord == rhs.Coord && lhs.Number == rhs.Number)
      return true;
   else
      return false;
}


public static bool operator !=(Node lhs, Node rhs)
{
   return !(lhs == rhs);
}
```

```
        #endregion
    }
}
```

Edge.cs:

```
using System;
using System.Linq;

namespace EMFFM.MainApp.Mesh.Elements
{
    /// <summary>
    /// Описание ребра (стороны элемента)
    /// </summary>
    public class Edge : IEquatable<Edge>, IComparable<Edge>
    {
        /// <summary>
        /// Вершина 1
        /// </summary>
        public Node Vertex1 { get; set; } // вершина 1
        /// <summary>
        /// Вершина 2
        /// </summary>
        public Node Vertex2 { get; set; } // вершина 2
        /// <summary>
        /// Номер ребра (глобальный)
        /// </summary>
        public int Number { get; set; }

        /// <summary>
        /// Локальный номер ребра (в элементе)
        /// </summary>
        public int LocalNumber { get; set; }

        /// <summary>
        /// Граничное ли ребро
        /// </summary>
        public bool IsBoundary { get; set; }
        /// <summary>
        /// Номер граничного условия
        /// </summary>
        public int BoundaryNumber { get; set; }

        #region Конструкторы

        /// <summary>
        /// Инициализация ребра для двух узлов
        /// </summary>
        /// <param name="n1">Узел 1</param>
        /// <param name="n2">Узел 2</param>
        public Edge(Node n1, Node n2)
        {
            Vertex1 = n1;
            Vertex2 = n2;
            Number = -1; // ребро не пронумеровано!
        }

        /// <summary>
        /// Инициализация ребра с параметрами
        /// </summary>
        /// <param name="n1">Узел 1</param>
        /// <param name="n2">Узел 2</param>
        /// <param name="number">Номер ребра (глобальный)</param>
        public Edge(Node n1, Node n2, int number)
        {
            Vertex1 = n1;
            Vertex2 = n2;
            Number = number;
        }

        /// <summary>
        /// Инициализация ребра с параметрами
        /// </summary>
        /// <param name="n1">Узел 1</param>
        /// <param name="n2">Узел 2</param>
        /// <param name="isBoundary">Является ли ребро граничным</param>
        /// <param name="boundNumber">Номер границы</param>
        public Edge(Node n1, Node n2, bool isBoundary, int boundNumber)
```

```
{
    Vertex1 = n1;
    Vertex2 = n2;
    Number = -1; // ребро не нумеровано!
    IsBoundary = isBoundary;
    BoundaryNumber = boundNumber;
}

/// <summary>
/// Инициализация ребра с параметрами
/// </summary>
/// <param name="n1">Узел 1</param>
/// <param name="n2">Узел 2</param>
/// <param name="number">Номер ребра</param>
/// <param name="isBoundary">Является ли ребро граничным</param>
/// <param name="boundNumber">Номер границы</param>
public Edge(Node n1, Node n2, int number, bool isBoundary, int boundNumber)
{
    Vertex1 = n1;
    Vertex2 = n2;
    Number = number;
    IsBoundary = isBoundary;
    BoundaryNumber = boundNumber;
}

/// <summary>
/// Копирование ребер
/// </summary>
/// <param name="e">Ребро</param>
public Edge(Edge e)
{
    Vertex1 = e.Vertex1;
    Vertex2 = e.Vertex2;
    Number = -1; // ребро не пронумеровано!
}

#endregion

/// <summary>
/// Занумеровано ли ребро
/// </summary>
public bool IsNumerated
{
    get
    {
        // если занумерован, то будет номер его на -1
        return (Number != -1);
    }
}

public bool Equals(Edge other)
{
    /*if (other == null)
    {
        throw new ArgumentException("other");
    }
    return Vertex1.Equals(other.Vertex1) && Vertex2.Equals(other.Vertex2);*/
    // NOTE: сравнение для выполнения операции Distinct (выбор уникальных значений)
    return ((Vertex1 == other.Vertex1) && (Vertex2 == other.Vertex2) ||
            (Vertex1 == other.Vertex2) && (Vertex2 == other.Vertex1));
}

public int CompareTo(Edge other)
{
    if (Number > other.Number)
        return 1;
    else if (Number < other.Number)
        return -1;
    else
        return 0;
}

public override int GetHashCode()
{
    return (Vertex1.Number + Vertex2.Number).GetHashCode();
}

public override string ToString()
```

```
        {
            return String.Format("{0} [ {1} -> {2} ]", Number, Vertex1.ToString(), Vertex2.ToString());
        }

        /// <summary>
        /// Вычисление длины ребра (стороны)
        /// </summary>
        /// <returns>Возвращает вычисленное значение длины</returns>
        public double GetLenght()
        {
            return Math.Geometry.GetDistance(Vertex1.Coord, Vertex2.Coord);
        }

        /// <summary>
        /// Вычисление длины ребра (стороны)
        /// </summary>
        /// <param name="coFactor">Поправка размеров относительно 1 метра (например, для нм => 1E-9)</param>
        /// <returns>Возвращает вычисленное значение длины</returns>
        public double GetLenght(double coFactor)
        {
            return Math.Geometry.GetDistance(Vertex1.Coord, Vertex2.Coord) * coFactor;
        }

        #region Перегрузка операторов

        public static bool operator ==(Edge lhs, Edge rhs)
        {
            // NOTE: сравниваем две вершины, причем не важно какая из них 1-ая или 2-ая
            if (((lhs.Vertex1 == rhs.Vertex1) && (lhs.Vertex2 == rhs.Vertex2)) ||
                ((lhs.Vertex1 == rhs.Vertex2) && (lhs.Vertex2 == rhs.Vertex1)))
                return true;
            else
                return false;
        }

        public static bool operator !=(Edge lhs, Edge rhs)
        {
            return !(lhs == rhs);
        }

        #endregion
    }
}
```

EdgeMesh.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.Mesh.Elements;

namespace EMFFM.MainApp.Mesh
{
    /// <summary>
    /// Класс для векторизации узловых элементов (нумерация ребер)
    /// </summary>
    class EdgeMesh
    {
        /// <summary>
        /// Нумерация ребер для сетки
        /// </summary>
        /// <typeparam name="TType">Тип элементов</typeparam>
        /// <param name="elements">Список элементов</param>
        /// <returns>Возвращаемое число указывает размерность глобальной матрицы и вектора</returns>
        public static int BuiltEdges<TType>(ref List<TType> elements)
        {
            // 1. Сборка всех ребер элементов в список
            int elementEdgesCount = (elements[0] as Element).Edges.Count;
            List<Edge> edges = new List<Edge>(elements.Count * elementEdgesCount);
            foreach (var item in elements)
            {
                edges.AddRange((item as Element).Edges);
            }

            // 2. Выполнение уникальной нумерации узлов
            List<Edge> numbered = NumerateEdges(edges);

            // 3. Объединение уникальных номеров с номерами ребер в элементах
            foreach (var item in elements.AsParallel())
```

```
            {
                foreach (var edge in (item as Element).Edges)
                {
                    foreach (var edgenum in numbered.AsParallel())
                    {
                        if (edge == edgenum)
                        {
                            edge.Number = edgenum.Number;
                        }
                    }
                }
            }

            return numbered.Count;
        }

        /// <summary>
        /// Выполнение уникальной нумерации ребер (без повторов)
        /// </summary>
        /// <param name="edges">Список узлов</param>
        /// <returns>Возвращает список пронумерованных уникально ребер элементов сетки</returns>
        /// <remarks>Вход - список всех ребер сетки, выход - нумерованные ребра без повторов</remarks>
        private static List<Edge> NumerateEdges(List<Edge> edges)
        {
            List<Edge> noduplicates = edges.AsParallel().Distinct().ToList();

            int curnum = 1; // начальный номер для текущего ребра
            foreach (var item in noduplicates)
            {
                item.Number = curnum;
                curnum++;
            }
            return noduplicates;
        }
    }
}
```

MatrixBuilder.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.Mesh.Elements;

namespace EMFFM.MainApp.FEM
{
    /// <summary>
    /// Класс для построения матриц (для МКЭ)
    /// </summary>
    static class MatrixBuilder
    {
        static double _coFactor = 1;

        /// <summary>
        /// Построение глобальной матрицы для ребер элементов сетки
        /// </summary>
        /// <typeparam name="TType">Тип элементов сетки</typeparam>
        /// <param name="size">Размерность матрицы (количество ребер)</param>
        /// <param name="elements">Список элементов, полученных при дискретизации</param>
        /// <param name="k">Волновое число (K)</param>
        /// <param name="omega">Круговая частота (омега)</param>
        /// <param name="coFactor">Нормирующий коэффициент для размеров (перевод к метрам)</param>
        /// <returns>Возвращает построенную матрицу</returns>
        public static double[,] BuiltGlobalMatrix<TType>(int size, List<TType> elements, double k, double omega, double coFactor)
        {
            _coFactor = coFactor;
            return BuiltGlobalMatrix(size, elements, k, omega);
        }

        /// <summary>
        /// Построение глобальной матрицы для ребер элементов сетки
        /// </summary>
        /// <typeparam name="TType">Тип элементов сетки</typeparam>
        /// <param name="size">Размерность матрицы (количество ребер)</param>
        /// <param name="elements">Список элементов, полученных при дискретизации</param>
        /// <param name="k">Волновое число (K)</param>
        /// <param name="omega">Круговая частота (омега)</param>
        /// <returns>Возвращает построенную матрицу</returns>
        public static double[,] BuiltGlobalMatrix<TType>(int size, List<TType> elements, double k, double omega)
```

```
{
    double[,] mat = Math.MatrixHelper.ZerosMatrix(size, size);

    // бегаем по элементам сетки
    foreach (var item in elements)
    {
        // генерируем локальную матрицу элемента
        double[,] tmp = BuildLocalMatrix(item, k, omega);

        // бегаем по ребрам элемента (вставка локальной матрицы в глобальную)
        for (int i = 0; i < (item as Element).Edges.Count; i++)
        {
            for (int j = 0; j < (item as Element).Edges.Count; j++)
            {
                // BUG: хитрый баг при сборке матрицы, иначе получаются бесконечные значения
                if (System.Math.Abs(tmp[i, j]) > 1.0E-10)
                {
                    mat[(item as Element).Edges[i].Number - 1, (item as Element).Edges[j].Number - 1] += tmp[i, j];
                }
            }
        }
    }

    // проверка знака в матрице (глобальной)
    foreach (var item in elements)
    {
        for (int i = 0; i < (item as Element).Edges.Count; i++)
        {
            for (int j = 0; j < (item as Element).Edges.Count; j++)
            {
                if ((item as Element).Edges[i].Number - 1 > (item as Element).Edges[j].Number - 1)
                {
                    mat[(item as Element).Edges[i].Number - 1, (item as Element).Edges[j].Number - 1] *= -1;
                }
            }
        }
    }

    return mat;
}

/// <summary>
/// Вычисление локальной матрицы элемента
/// </summary>
/// <typeparam name="TType">Тип элементов сетки</typeparam>
/// <param name="element">Элемент</param>
/// <param name="k">Волновое число (K)</param>
/// <param name="omega">Круговая частота (омега)</param>
/// <returns>Возвращает вычисленную локальную матрицу элемента</returns>
/// <remarks>Производит вычисление матрицы вида: Ek + k*k *Fk (где k - элемент)
/// Учитывается как волновое число, так и Epsilon (диэлектрическая проводимость)</remarks>
private static double[,] BuildLocalMatrix<TType>(TType element, double k, double omega)
{
    // TODO: продумать этот метод для правильного вычисления матрицы локальной

    // для тетраэдра выполним подготовку данных для вычисления локальных матриц
    if (element is Tetrahedron)
    {
        (element as Tetrahedron).Prerequsite(_coFactor);
    }

    // вычисляем локальную матрицу E
    double[,] emat = (element as IElement).GetLocalE(); // матрица Eij
    emat = Math.MatrixHelper.Multiply(1 / (element as Element).Material.Mu, emat); // умножаем на 1/Mu

    // если волновое число не нулевое, то считаем еще и локальную матрицу F
    if (k != 0)
    {
        double[,] fmat = (element as IElement).GetLocalF(); // матрицы Fij

        // NOTE: выполнение пересчета Epsilon(w)
        double eps = CalculateEpsilon(element, omega);
        fmat = Math.MatrixHelper.Multiply(-eps * k * k, fmat); // сразу сделаем умножение на Eps*K*K и * -1

        // складываем матрицы: E+k^2*F (так как ранее умножили на -1, то будет E-F)
        // вычисляем [Ek] - k*k*eps*[Fk]
        emat = Math.MatrixHelper.Sum(emat, fmat);
    }
```

```
            return emat;
        }

        /// <summary>
        /// Вычисление дисперсного параметра Epsilon(w)
        /// </summary>
        /// <typeparam name="TType">Тип элемента сетки</typeparam>
        /// <param name="element">Элемент</param>
        /// <param name="omega">Круговая частота (омега)</param>
        /// <returns>Возвращает значение диэлектрической проводимости</returns>
        private static double CalculateEpsilon<TType>(TType element, double omega)
        {
            double epsilon = (element as Element).Material.Eps;

            if ((element as Element).Material.IsPlasmon)
            {
                epsilon = ParametersBuilder.CalculateEpsilonForPlasmon(omega, (element as Element).Material.PlasmonData);
            }

            return epsilon;
        }
    }
}
```

ForceVector.cs:

```
using System;
using System.Linq;
using System.Collections.Generic;
using EMFFM.MainApp.FEM.Elements;
using EMFFM.MainApp.Math;
using EMFFM.MainApp.Mesh.Elements;

namespace EMFFM.MainApp.FEM
{
    /// <summary>
    /// Описание класса для вектора воздействий
    /// </summary>
    class ForceVector
    {
        WaveData _wave;

        Func<double, WaveData, double, double> _function;

        int _totalEdgesCount;

        List<Edge> _edges;

        /// <summary>
        /// Инициализация вектора с известными значениями поля
        /// </summary>
        /// <param name="initValue">Начальное значение (амплитуда)</param>
        /// <param name="function">Функция для вычисления значений поля</param>
        /// <param name="totalEdges">Общее количество ребер (уникальных) в сетке</param>
        /// <param name="edges">Список ребер на границе источника</param>
        public ForceVector(WaveData wave, Func<double, WaveData, double, double> function, int totalEdges, List<Edge> edges)
        {
            _wave = wave;
            _function = function;
            _edges = edges;
            _totalEdgesCount = totalEdges;
        }
        /// <summary>
        /// Генерация вектора воздействий
        /// </summary>
        /// <param name="currentTime">Текущее значение времени</param>
        /// <returns>возвращает сгенерированный вектор</returns>
        public double[] Generate(double currentTime)
        {
            double[] result = new double[_totalEdgesCount];

            // вычисляем по функции
            for (int i = 0; i < _edges.Count; i++)
            {

                double position = CalculatePosiltion(_edges[i]);
                result[_edges[i].Number - 1] = _function(position, _wave, currentTime);
            }
```

```
            return result;
        }

        /// <summary>
        /// Вычисление скалярного произведения k*n*r
        /// </summary>
        /// <param name="edge">Ребро</param>
        /// <returns>Возвращает скалярное значение</returns>
        private double CalculatePosiltion(Edge edge)
        {
            Point p = Math.Geometry.GetCenterPoint(edge.Vertex1.Coord, edge.Vertex2.Coord);

            TVector vector = new TVector(_wave.Point, p);
            double result = vector * _wave.Vector;

            return result * _wave.WaveNumber;
        }
    }
}
```

GlobalMatrix.cs:

```
using System;
using System.Linq;
using EMFFM.MainApp.FEM.Elements;
using EMFFM.MainApp.Input;
using EMFFM.MainApp.Mesh;

namespace EMFFM.MainApp.FEM
{
    /// <summary>
    /// Класс для глобальной матрицы и ее построения и пересчета
    /// </summary>
    class GlobalMatrix<TType, TType2>
    {
        InputData _inputData;
        TMesh<TType, TType2> _mesh;

        /// <summary>
        /// Инициализация матрицы и ее данных
        /// </summary>
        /// <param name="data">Входные данные</param>
        /// <param name="mesh">Сгенерированная сетка</param>
        public GlobalMatrix(InputData data, TMesh<TType, TType2> mesh)
        {
            _inputData = data;
            _mesh = mesh;

            Init();
        }

        /// <summary>
        /// Инициализация параметров для построения матрицы
        /// </summary>
        private void Init()
        {
            // настройка параметров материала (свойств частицы)
            MaterialsBuilder.SetMaterialsByDomain(ref _mesh, _inputData.Geomerty, _inputData.Materials, _inputData.Plasmons);
        }

        /// <summary>
        /// Генерация глобальной матрицы
        /// </summary>
        /// <param name="currentFreq">Текущая частота источника</param>
        /// <returns>Возвращает сгенерированную матрицу</returns>
        public double[,] Generate(double currentFreq)
        {
            // 1. Вычисление волнового числа
            double waveNumber = ParametersBuilder.CalculateWaveNumber(currentFreq);

            // 1.1. Вычисление круговой частоты (омега)
            double omega = ParametersBuilder.CalculateOmega(currentFreq);

            // 2. Вычисление и построение матрицы глобальной
            double[,] result = MatrixBuilder.BuiltGlobalMatrix(_mesh.EdgesCount, _mesh.MeshElements, waveNumber, omega);

            return result;
        }
```

```
        /// <summary>
        /// Генерация глобальной матрицы
        /// </summary>
        /// <param name="wave">Сведения о волне</param>
        /// <returns>Возвращает сгенерированную матрицу</returns>
        public double[,] Generate(WaveData wave)
        {
            // 1. Вычисление и построение матрицы глобальной
            double[,] result = MatrixBuilder.BuiltGlobalMatrix(_mesh.EdgesCount, _mesh.MeshElements, wave.WaveNumber, wave.Omega,
_inputData.Geomerty.CoFactor);

            return result;
        }
    }
}
```

Solver.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.FEM.Elements;
using EMFFM.MainApp.Helpers;
using EMFFM.MainApp.Input;

namespace EMFFM.MainApp.FEM
{
    /// <summary>
    /// Класс для решения итоговой системы уравнений
    /// </summary>
    class Solver<TType, TType2>
    {
        NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();

        InputData _data;
        WaveData _wave;
        string _outputFilePrefix;

        ForceVector _vectorB;
        GlobalMatrix<TType, TType2> _matrix;

        public List<Result> Results { get; set; }

        int[] _boundNums;

        /// <summary>
        /// Инициализация решателя
        /// </summary>
        /// <param name="data">Входные данные</param>
        /// <param name="wave">Сведения о волне</param>
        /// <param name="matrixA">Глобальная матрица</param>
        /// <param name="vector">Объект для работы с вектором воздействий</param>
        public Solver(InputData data, WaveData wave, GlobalMatrix<TType, TType2> matrix, ForceVector vector, int[] boundaryEdges)
        {
            logger.Info("Инициализация решателя...");

            _data = data;
            _wave = wave;
            _outputFilePrefix = String.Format("{0}{1}", data.Output.Path, data.Output.FilePrefix);
            Results = new List<Result>();

            _matrix = matrix;
            _vectorB = vector;

            _boundNums = boundaryEdges;
        }

        /// <summary>
        /// Запуск многоразового решения системы
        /// </summary>
        public void Run()
        {
            if (_data.Options.WithoutIterations)
            {
                logger.Info("Решение для единичного времени...");
                SolveOne(0);
            }
            else
```

```
        {
            if (_data.Options.InTimeDomain)
            {
                logger.Info("Расчет во временной области...");
                RunInTimeDomain();
            }
            else if (_data.Options.InFrequencyDomain)
            {
                logger.Info("Расчет в частотной области...");
                RunInFrequencyDomain();
            }
        }
        logger.Info("Решатель завершил работу!");
    }

    #region Решение по времени

    /// <summary>
    /// Решение во временной области (по времени)
    /// </summary>
    private void RunInTimeDomain()
    {
        double initTime = 0;
        // конечная частота работы источника
        double timeEnd = initTime + _data.Variables.Time;

        int step = 1;

        // бегаем по временным шагам
        for (double time = initTime; time < _data.Variables.Time; time += _data.Variables.TimeDelta)
        {
            logger.Info("Solve in time domain at step {0} - time = {1}...", step, time);

            SolveTime(time, step);
            step++;
        }
    }

    /// <summary>
    /// Вычисления поля для текущего времени
    /// </summary>
    /// <param name="currentTime">Текущее время</param>
    /// <param name="stepNumber">Номер шага (итерация)</param>
    private void SolveTime(double currentTime, int stepNumber)
    {
        // 1. Вычисление вектора B
        double[] vec = _vectorB.Generate(currentTime);

        // 2. Построение глобальной матрицы
        double[,] mat = _matrix.Generate(_wave);

        // NOTE: применение граничных условий Дирихле
        BoundaryBuilder.ApplyDirichletBoundaryCondition2(ref mat, ref vec, _boundNums,
            ArrayHelper.InitValueVector<double>(_boundNums.Length, _data.Boundaries.Single(b => b.Type ==
Common.BoundaryTypes.Dirichlet).Value));

        // NOTE: или так записать вместео трех предыдущих строк
        double[] result = Math.MathNetHelper.Solve(mat, vec);

        // 4. Запись результата в файл
        string[] resfiles = PrintResultsToFile(vec, result, stepNumber);

        // 5. Добавление результата в список с результатами
        AddResultItem(resfiles[0], resfiles[1], _wave, currentTime);
    }

    #endregion

    #region Решение в частотной области

    /// <summary>
    /// Запуск решения в частной области с вариацией частот
    /// </summary>
    private void RunInFrequencyDomain()
    {
        // исходная частота работы источника
        double initFreq = ParametersBuilder.GetFrequency(_data.Sources, Common.SourceTypes.Simple);
        // стартовая частота работы источника
```

```
            double freqStart = initFreq - _data.Variables.FrequencyDelta;
            // конечная частота работы источника
            double freqEnd = initFreq + _data.Variables.FrequencyDelta;

            // номер шага
            int step = 1;

            for (double w = freqStart; w < freqEnd; w += _data.Variables.FrequencyStep)
            {
                logger.Info("Solve in frequency domain at step {0} - omega = {1}...", step, w);

                SolveFrequency(w, step);
                step++;
            }
        }

        /// <summary>
        /// Выполнение расчета для текущего шага
        /// </summary>
        /// <param name="freq">Частота источника (MHz)</param>
        /// <param name="iterateNumber">Номер итерации (шага)</param>
        private void SolveFrequency(double freq, int iterateNumber)
        {
            _wave.Omega = ParametersBuilder.CalculateOmega(freq);

            // 1. Вычисление вектора B
            double[] vec = _vectorB.Generate(0);

            // 2. Построение глобальной матрицы
            double[,] mat = _matrix.Generate(_wave);

            // NOTE: применение граничных условий Дирихле
            BoundaryBuilder.ApplyDirichletBoundaryCondition2(ref mat, ref vec, _boundNums,
                ArrayHelper.InitValueVector<double>(_boundNums.Length, _data.Boundaries.Single(b => b.Type == Common.BoundaryTypes.Dirichlet).Value));

            // NOTE: или так записать вместео трех предыдущих строк
            double[] result = Math.MathNetHelper.Solve(mat, vec);

            // 4. Запись результата в файл
            string[] resfiles = PrintResultsToFile(vec, result, iterateNumber);

            double eps = ParametersBuilder.CalculateEpsilonForPlasmon(freq, _data.Plasmons[0]);

            // 5. Добавление результата в список с результатами
            AddResultItem(resfiles[0], resfiles[1], _wave, freq, eps);
        }

        #endregion
        #region Единичное решение
        /// <summary>
        /// Решение в единственный момент времени с заданными начальными параметрами
        /// </summary>
        private void SolveOne(double currentTime)
        {
            // 1. Вычисление вектора B
            double[] vec = _vectorB.Generate(currentTime);

            // 2. Построение глобальной матрицы
            double[,] mat = _matrix.Generate(_wave);

            // NOTE: применение граничных условий Дирихле
            BoundaryBuilder.ApplyDirichletBoundaryCondition2(ref mat, ref vec, _boundNums,
                ArrayHelper.InitValueVector<double>(_boundNums.Length, _data.Boundaries.Single(b => b.Type == Common.BoundaryTypes.Dirichlet).Value));

            // NOTE: или так записать вместео трех предыдущих строк
            double[] result = Math.MathNetHelper.Solve(mat, vec);

            // 4. Запись результата в файл
            string[] resfiles = PrintResultsToFile(vec, result, 0);

            // 5. Добавление результата в список с результатами
            AddResultItem(resfiles[0], resfiles[1], _wave, currentTime);
        }
        #endregion
        #region Вспомогательные функции
```

```
        /// <summary>
        /// Вывод результата в текстовый файл
        /// </summary>
        /// <param name="result">Вектор с результатом</param>
        /// <param name="number">Номер итерации</param>
        /// <returns>Возвращает имена записанных файлов с результатами</returns>
        private string[] PrintResultsToFile(double[] vector, double[] result, int number)
        {
            // формируем имя файла в зависимости от итерации
            string fileNameVector = String.Format("{0}{1}-{2}.txt", _outputFilePrefix, "vector", number);
            string fileNameResult = String.Format("{0}{1}-{2}.txt", _outputFilePrefix, "result", number);
            // вывод файла с результатами вычисления вектора воздействий
            Output.OutputHelper.OutputVectorToFile<double>(vector, fileNameVector);
            // вывод файла с результатом решения СЛАУ
            Output.OutputHelper.OutputVectorToFile<double>(result, fileNameResult);
            return new string[] { fileNameVector, fileNameResult };
        }
        /// <summary>
        /// Добавление результата в список с данными о результатах
        /// </summary>
        /// <param name="forceFile">Имя файала с вектором воздействий ({b})</param>
        /// <param name="resultFile">Имя файла с результатами (вектором)</param>
        /// <param name="wave">Сведения о волне</param>
        /// <param name="time">Значение времени</param>
        private void AddResultItem(string forceFile, string resultFile, WaveData wave, double time)
        {
            Results.Add(new Result()
            {
                FileName = resultFile,
                ForceFileName = forceFile,
                Wave = wave,
                Time = time
            });
        }
        /// <summary>
        /// Добавление результата в список с данными о результатах
        /// </summary>
        /// <param name="forceFile">Имя файала с вектором воздействий ({b})</param>
        /// <param name="resultFile">Имя файла с результатами (вектором)</param>
        /// <param name="wave">Сведения о волне</param>
        /// <param name="time">Значение времени</param>
        /// <param name="frequency">Частота работы источника (Hz)</param>
        /// <param name="epsilon">Диэлектрическая проницаемость</param>
        private void AddResultItem(string forceFile, string resultFile, WaveData wave, double frequency, double epsilon)
        {
            Results.Add(new Result()
            {
                FileName = resultFile,
                ForceFileName = forceFile,
                Wave = wave,
                Frequency = frequency,
                Epsilon = epsilon
            });
        }
        #endregion
    }
}
```

FieldAnalisys.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.Helpers;
using EMFFM.MainApp.Mesh;
using EMFFM.MainApp.Common;
using EMFFM.MainApp.Mesh.Elements;

namespace EMFFM.MainApp.FEM
{
    /// <summary>
    /// Класс для выполнения анализа поля
    /// </summary>
    class FieldAnalysis <TType, TType2>
    {
        List<Result> _results;
        TMesh<TType, TType2> _mesh;
        ElementType _volElementType;
```

```
/// <summary>
/// Результирующие поля
/// </summary>
public List<FieldData> ResultFields;

/// <summary>
/// Поля от внешних источников
/// </summary>
public List<FieldData> IncidentFields;

/// <summary>
/// Инициализация анализатора с параметрами
/// </summary>
/// <param name="results">Сведения о результатах</param>
/// <param name="mesh">Сетка</param>
/// <param name="elementsType">Тип объемных элементов сетки</param>
public FieldAnalysis(List<Result> results, TMesh<TType, TType2> mesh, ElementType elementsType)
{
   ResultFields = new List<FieldData>(results.Count);
   IncidentFields = new List<FieldData>(results.Count);
   _results = results;
   _mesh = mesh;
   _volElementType = elementsType;

   InitData();
}

/// <summary>
/// Чтение файлов с результатами и заполнение данных о полях
/// </summary>
private void InitData()
{
   // считываем поля, полученные в результате решения СЛАУ
   foreach (var result in _results)
   {
      FieldData data = new FieldData(FieldHelper.ReadFieldFromResultFile(result.FileName), result.Wave);
      ResultFields.Add(data);
   }
   // считываем поля, полученные в результате составления вектора воздействий
   foreach (var result in _results)
   {
      FieldData data = new FieldData(FieldHelper.ReadFieldFromResultFile(result.ForceFileName), result.Wave);
      IncidentFields.Add(data);
   }
}

/// <summary>
/// Вычисление полей
/// </summary>
public void CalculateFields(double coFactor)
{
   switch (_volElementType)
   {
       case ElementType.Tetrahedron: // поля для тетраэдра описаны через базисные векторные функции
         foreach (var field in ResultFields) // для каждого полученного поля
         {
            foreach (var element in _mesh.MeshElements) // для каждого элемента ищем его глобальное поле
            {
               CalculateLocalElement<Tetrahedron>(element as Tetrahedron, field, coFactor);
            }
         }
         break;
   }
}

/// <summary>
/// Вычисление поля на элементе
/// </summary>
/// <typeparam name="TType">Тип элемента</typeparam>
/// <param name="element">Данные элемента</param>
/// <param name="field">Данные о локальном поле</param>
private void CalculateLocalElement<TType>(TType element, FieldData field, double coFactor)
{
   ElementField elf = new ElementField();

   BasisFunctions<TType> wmat = new BasisFunctions<TType>(element);
   // определяем векторные соотношения для точки в центре масс элемента
   double[,] mat = wmat.GetW((element as Element).GetCenterPoint());
```

```
            double[] vec = new double[6];
            int k = 0;
            foreach (var edge in (element as Element).Edges)
            {
                vec[k] = edge.GetLenght(coFactor) * field.EdgeFields[edge.Number - 1].TangentialField;
                elf.EdgeFields.Add(field.EdgeFields[edge.Number - 1]);
                k++;
            }

            double[] result = Math.MathNetHelper.Multiply(mat, vec);

            elf.CenterField = new Field(result);
            field.ElementFields.Add(elf);
        }

        /// <summary>
        /// Вычисление разницы между полями (между полученным и падающим)
        /// </summary>
        /// <remarks>В итоге изменяется значение результирующего поля!</remarks>
        public void CalculateDifference()
        {
            for (int i = 0; i < _results.Count; i++)
            {
                ResultFields[i].EdgeFields = FieldHelper.Difference(ResultFields[i].EdgeFields, IncidentFields[i].EdgeFields);
            }
        }

        /// <summary>
        /// Вывод результатов по всем полученным результирующим полям
        /// </summary>
        /// <param name="outputPath">Каталог для вывода результата</param>
        /// <param name="filePrefix">Префикс названия файла</param>
        public void PrintResultField(string outputPath, string filePrefix)
        {
            for (int i = 0; i < _results.Count; i++)
            {
                string fileName = string.Format("{0}\\{1}-result-{2}.txt", outputPath, filePrefix, i);
                FieldHelper.OutputFieldToFile(ResultFields[i].EdgeFields, fileName);
            }
        }

        /// <summary>
        /// Вывод полей по элементам
        /// </summary>
        /// <param name="outputPath">Каталог для вывода результата</param>
        /// <param name="filePrefix">Префикс названия файла</param>
        /// <param name="withEdgeFields">Выводить значения полея на ребрах?</param>
        public void PrintElementsField(string outputPath, string filePrefix, bool withEdgeFields)
        {
            for (int i = 0; i < _results.Count; i++)
            {
                string fileName = string.Format("{0}\\{1}full-{2}.txt", outputPath, filePrefix, i);
                FieldHelper.OutputFieldToFile(ResultFields[i].ElementFields, fileName, withEdgeFields);
            }
        }

        /// <summary>
        /// Вывод полей по элементам
        /// </summary>
        /// <param name="elementNumbers">Номера элементов из области для вывода данных по ним</param>
        /// <param name="outputPath">Каталог для вывода результата</param>
        /// <param name="filePrefix">Префикс названия файла</param>
        /// <param name="withEdgeFields">Выводить значения полея на ребрах?</param>
        /// <param name="domainNumber">Номер области, для которой выводятся сведения</param>
        public void PrintElementsField(int[] elementNumbers, string outputPath, string filePrefix, bool withEdgeFields, int domainNumber)
        {
            for (int i = 0; i < _results.Count; i++)
            {
                string fileName = string.Format("{0}\\{1}full-domain-{2}-{3}.txt", outputPath, filePrefix, domainNumber, i);
                FieldHelper.OutputFieldToFile(ResultFields[i].ElementFields, elementNumbers, fileName, withEdgeFields);
            }
        }
    }
}
```

ParametersBuilder.cs:

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using EMFFM.MainApp.Input;
using EMFFM.MainApp.Common;

namespace EMFFM.MainApp.FEM
{
   /// <summary>
   /// Класс для вычисления параметров, необходимых для решения
   /// </summary>
   /// <remarks>Вычисление всех вспомогательных параметров и величин здесь!</remarks>
   class ParametersBuilder
   {
      static NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();
      static double AbsPermeable = 1.25663706144E-06; // магнитная постоянная
      static double AbsPermitt = 8.85418782E-12; // электрическая постоянная
      static double FreeSpaceVel = 1.0 / System.Math.Sqrt(AbsPermeable * AbsPermitt); // фазовая скорость распространения волны
      static double LightSpeed = 299792458; // скорость света в вакууме(м/с)
      static double ehbar = 1.519250349719305e+15; // e/hbar where hbar=h/(2*pi) and e=1.6e-19

      /// <summary>
      /// Вычисление значения волнового числа K
      /// </summary>
      /// <param name="freq">Частота источника (MHz)</param>
      /// <returns>Возвращает значение волнового числа</returns>
      public static double CalculateWaveNumber(double freq)
      {
         // NOTE: проверенный метод для определения волнового числа
         double  WaveLength;

         double OperateFreq = freq; // частота фунционирования источника

         //WaveLength = FreeSpaceVel / (OperateFreq * 1.0E+06);
         WaveLength = FreeSpaceVel / OperateFreq ;
         double WaveNumber = 2.0 * System.Math.PI / WaveLength; // модуль волнового вектора

         double Omega = WaveNumber * FreeSpaceVel;

         logger.Debug("OperateFrequency = {0} MHz", OperateFreq);
         logger.Debug("WaveLength = {0} m", WaveLength);
         logger.Debug("FreeSpaceVelocity = {0} m/s", FreeSpaceVel);
         logger.Debug("WaveNumber = {0}", WaveNumber);
         logger.Debug("Omega = {0} rad/s", Omega);

         return WaveNumber;
      }

      /// <summary>
      /// Вычисление значения круговой частоты (омега)
      /// </summary>
      /// <param name="freq">Частота источника (Hz)</param>
      /// <returns>Возвращает значение частоты</returns>
      public static double CalculateOmega(double freq)
      {
         double operateFreq = freq; // круговая частота фунционирования источника

         //double waveLength = FreeSpaceVel / operateFreq;
         //double waveNumber = 2.0 * System.Math.PI / waveLength; // модуль волнового вектора
         //double omega = waveNumber * FreeSpaceVel;

         double omega = 2 * System.Math.PI * freq;

         //logger.Debug("WaveLength = {0} m", waveLength);
         //logger.Debug("WaveNumber = {0}", waveNumber);
         logger.Debug("Omega = {0} rad/s", omega);
         //logger.Debug("Omega2 = {0} rad/s", omega);

         return omega;
      }

      /// <summary>
      /// Получение значения частоты из входного файла
      /// </summary>
      /// <param name="sources">Список источников во входном файле</param>
      /// <param name="sourceType">Тип источника</param>
      /// <returns>Возвращает исходное значение частоты</returns>
      public static double GetFrequency(List<Source> sources, SourceTypes sourceType)
      {
```

```
            double lambda = sources.Single(s => s.Type == sourceType).Lambda;

            //freq *= 10E6; // переводим из МегаГерц и Герцы

            double freq = 2 * System.Math.PI * LightSpeed / lambda;
            logger.Debug("Frequency = {0} Hz", freq);

            return freq;
        }

        /// <summary>
        /// Получение значения круговой частоты по длине волны
        /// </summary>
        /// <param name="lambda">Длина волны (метры)</param>
        /// <returns>Возвращает значение Omega</returns>
        public static double GetFrequency(double lambda)
        {
            double freq = 2 * System.Math.PI * LightSpeed / lambda;
            logger.Debug("Frequency = {0} Hz", freq);

            return freq;
        }

        /// <summary>
        /// Получение значения амплитуды для источника
        /// </summary>
        /// <param name="sources">Список источников из входного файла</param>
        /// <param name="simple">Тип источника</param>
        /// <returns>Возвращает значение амплитуды</returns>
        public static double GetAmplitude(List<Source> sources, SourceTypes simple)
        {
            double amp = sources.Single(s => s.Type == SourceTypes.Simple).Amplitude;

            logger.Debug("Amplitude = {0}", amp);

            return amp;
        }

        /// <summary>
        /// Вычисление электрической постоянной на основе закона дисперсии для плазмона
        /// </summary>
        /// <param name="freq">Круговая частота источника (омега, рад/с)</param>
        /// <param name="data">Параметры плазмона</param>
        /// <returns>Возвращает значение Epsilon</returns>
        /// <remarks>Книга Климов Наноплазмоника, формула (3.32), действительная часть (3.33.1)</remarks>
        public static double CalculateEpsilonForPlasmon(double freq, PlasmonData data)
        {
            // NOTE: метод для вычисления дисперсного значения электрической постоянной
            double eps = 1 - ((data.Plasmow * data.Plasmow) * (data.Relax * data.Relax)) / (1 + (freq * freq) * (data.Relax * data.Relax));

            return eps;
        }
    }
}
```

NetgenUtils

MeshGenerator.cs:

```
using System;
using System.Linq;
using System.IO;
using NetgenUtils.Helpers;

namespace NetgenUtils.Preprocessing
{
    /// <summary>
    /// Класс для генерации сетки с использованием Netgen
    /// </summary>
    /// <remarks>Препроцессинг исходной задачи</remarks>
    public class MeshGenerator
    {
        Variables _vars; // параметры окружения

        /// <summary>
        /// Инициализация генератора сетки Netgen
        /// </summary>
        /// <param name="vars">Переменные окружения</param>
        public MeshGenerator(Variables vars)
        {
```

```
            _vars = vars;
        }

        /// <summary>
        /// Генерация сетки
        /// </summary>
        /// <param name="geomFile">Входной файл с геометрией задачи(*.geo)</param>
        /// <param name="outputFile">Выходной файл с данными сетки (*.mesh)</param>
        public void Generate(string geomFile, string outputFile)
        {
            // обработка файлов - полные пути к ним
            geomFile = Path.GetFullPath(geomFile);
            outputFile = String.Format("{0}\\{1}{2}", _vars.OutputPath, _vars.MeshFilePrefix, outputFile);
            // формирование строки с аргументами
            string args = GenerateArgumentsString(geomFile, outputFile, "Neutral Format");
            // запуск приложения
            LaunchHelper.ExecuteProgram(_vars.NetgenPath, _vars.AppName, args);
        }

        /// <summary>
        /// Генерация строки с аргументами для запуска Netgen
        /// </summary>
        /// <param name="geomFile">Входной файл с геометрией</param>
        /// <param name="outputFile">Выходной файл с сеткой</param>
        /// <param name="meshType">Тип файла сетки для выхода</param>
        /// <param name="withTestout">Выводить ли лог работы Netgen в файл (*.out)</param>
        /// <param name="isDefaultMesh">Генерировать ли сетку без указания типа ее экспорта (сетки по умолчанию (*.vol))</param>
        /// <returns>Возвращает результирующую строку с аргументами</returns>
        private string GenerateArgumentsString(string geomFile, string outputFile, string meshType, bool withTestout = false, bool isDefaultMesh
= false)
        {
            string str = "";
            if (isDefaultMesh)
            {
                str = String.Format("-batchmode -geofile=\"{0}\" -meshfile=\"{1}\"", geomFile, outputFile);
            }
            else
            {
                if (withTestout)
                {
                    str = String.Format("-batchmode -geofile=\"{0}\" -meshfile=\"{1}\" -meshfiletype=\"{2}\" -testout=\"{3}\\testout.out\"", geomFile,
outputFile, meshType, _vars.OutputPath);
                }
                else
                {
                    str = String.Format("-batchmode -geofile=\"{0}\" -meshfile=\"{1}\" -meshfiletype=\"{2}\"", geomFile, outputFile, meshType);
                }
            }
            return str;
        }
    }
}
```

MeshFileParser

MeshParser.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using EMFFM.MeshParser.Common;
using EMFFM.MeshParser.Elements;
using EMFFM.MeshParser.Helpers;

namespace EMFFM.MeshParser
{
    /// <summary>
    /// Парсер для файла с данными сетки
    /// </summary>
    /// <remarks>Текстовый файл, генерируемый Netgen при сохранении сетки(*.vol)</remarks>
    public class MeshFileParser
    {
        string _fileName;

        /// <summary>
        /// Инициализация парсера
        /// </summary>
        /// <param name="fname">Название файла для разбора</param>
```

```
public MeshFileParser(string fname)
{
   _fileName = fname;
}

/// <summary>
/// Разбор содержимого файла
/// </summary>
/// <returns>Возвращает данные о сетке, прочитанной из файла</returns>
public MeshData Parse()
{
   FileTypes type = CommonHelpers.GetFileType(_fileName);

   MeshData data = new MeshData();

   switch (type)
   {
      case FileTypes.NeutralFormat: // Neutral Format
         ReadNeutralFormatFile(ref data);
         break;
      case FileTypes.SurfaceTriangulation: // Surface triangulation file
         ReadSurfaceTriangulationFile(ref data);
         break;
   }

   return data;
}

#region Чтение содержимого для разных типов файлов

/// <summary>
/// Чтение файла в формате Neutral Format
/// </summary>
/// <param name="data">Данные о сетке из файла</param>
private void ReadNeutralFormatFile(ref MeshData data)
{
   try
   {
      StreamReader file = new StreamReader(_fileName);

      // 1. Чтение списка точек и их кооридинаты (nodes)
      data.Points = ReadPoints(file);

      // 2. Чтение списа объемных элементов (volume elements)
      data.VolumeElements = ReadVolumeElements(file);

      // 3. Чтение списка поверзностных элементов (surface elements)
      data.SurfaceElements = ReadSurfaceElements(file, true);

      file.Close();
   }
   catch (Exception ex)
   {
      Console.WriteLine("Ошибка причтение *.neu файла: {0}", ex.Message);
   }
}

/// <summary>
/// Чтение файла формата Surface Triangulation File Format
/// </summary>
/// <param name="data">Данные о сетке из файла</param>
private void ReadSurfaceTriangulationFile(ref MeshData data)
{
   try
   {
      StreamReader file = new StreamReader(_fileName);

      string header = file.ReadLine();
      if (header != "surfacemesh")
      {
         throw new Exception("Файл не соответствует структуре Sufrace Triangulation Mesh!");
      }

      // 1. Чтение списка точек и их кооридинаты (nodes)
      data.Points = ReadPoints(file);

      // 2. Чтение списка поверзностных элементов (surface elements)
      data.SurfaceElements = ReadSurfaceElements(file);
```

```
            file.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Ошибка причтение Surface Trianguation файла: {0}", ex.Message);
        }
    }

    #endregion

    #region Чтение блоков файла

    /// <summary>
    /// Чтение списка точек и их кординат
    /// </summary>
    /// <param name="file">Открытый файл для чтения</param>
    /// <returns>Возвращает список точек</returns>
    private List<Point> ReadPoints(StreamReader file)
    {
        // считываем количество точек
        int count = Convert.ToInt32(file.ReadLine());

        List<Point> points = new List<Point>(count);

        string line;
        for (int i = 0; i < count; i++)
        {
            line = file.ReadLine();
            // разбираем строку с координатами (x y z)
            double[] coords = CommonHelpers.ParseCoordinatesString(line);
            points.Add(CommonHelpers.ConvertArrayToPoint(coords));
        }

        return points;
    }

    /// <summary>
    /// Чтение списка объемных элементов
    /// </summary>
    /// <param name="file">Открытый файл для чтения</param>
    /// <returns>Возвращает список элементов</returns>
    private List<VolumeElement> ReadVolumeElements(StreamReader file)
    {
        // считываем количество элементов
        int count = Convert.ToInt32(file.ReadLine());

        List<VolumeElement> elements = new List<VolumeElement>(count);

        string line;
        for (int i = 0; i < count; i++)
        {
            line = file.ReadLine();
            // разбираем строку с координатами (x y z)
            int[] nums = CommonHelpers.ParseVolumeElementString(line);
            elements.Add(CommonHelpers.ConvertArrayToVolumeElement(nums));
        }

        return elements;
    }

    /// <summary>
    /// Чтение списка поверхностных элементов
    /// </summary>
    /// <param name="file">Открытый файл для чтения</param>
    /// <param name="withBoundaryCond">Читать с учетом граничных условий</param>
    /// <returns>Возвращает список элементов</returns>
    private List<SufraceElement> ReadSurfaceElements(StreamReader file, bool withBoundaryCond = false)
    {
        // считываем количество элементов
        int count = Convert.ToInt32(file.ReadLine());

        List<SufraceElement> elements = new List<SufraceElement>(count);

        string line;
        for (int i = 0; i < count; i++)
        {
            line = file.ReadLine();
```

```
            // разбираем строку с координатами (x y z)
            int[] nums = CommonHelpers.ParseSurfaceElementString(line);
            elements.Add(CommonHelpers.ConvertArrayToSurfaceElement(nums, withBoundaryCond));
        }

        return elements;
    }

    #endregion
  }
}
```

MeshData.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using EMFFM.MeshParser.Elements;

namespace EMFFM.MeshParser
{
  /// <summary>
  /// Описание сетки, прочитанной из файла Mesh
  /// </summary>
  public class MeshData
  {
    public List<SufraceElement> SurfaceElements;

    public List<VolumeElement> VolumeElements;

    public List<Point> Points;

    /// <summary>
    /// Инициализация данных сетки
    /// </summary>
    public MeshData()
    {
      SurfaceElements = new List<SufraceElement>();
      Points = new List<Point>();
      VolumeElements = new List<VolumeElement>();
    }

    public override string ToString()
    {
      return String.Format("Mesh with {0} points", Points.Count);
    }
  }
}
```

ПРИЛОЖЕНИЕ Г
(справочное)
Работа с программой Netgen

1 Общие сведения

Программа Netgen предназначена для постропроцессинга, а также препроцессинга задач, использующих конечно элементные расчеты. Препроцессинг заключается в генерации сеток. Программа имеет мощный генератор сеток, который позволяет строить их для произвольной геометрии, задаваемой пользователем с помощью входных форматов. Кроме удобного и быстрого генератора сеток, в программе присутствует возможность для обработки результатов решений, полученных в конечноэлементных программах. Особенно это удобно в случае если программа не имеет своего графического интерфейса (это зачастую связано с объемом работы и сложностью, так как это целая отдельная система, которую зачастую не предусматривают в научных и исследовательских приложениях). В таком случае результаты можно экспортировать в формат решения, который потом визуализировать с помощью Netgen. Боле подробные сведения можно получить на сайте программы и в руководстве пользователя.

2 Составление файлов геометрии

Для описания геометрии, с которой будет производиться работа, необходимо использовать такие встроенные средства, как CGS (Constructive Solid Geometry), представленной в программе. Это формат файлов, позволяющий описывать геометрию и производить генерацию сеток по ней.

Это один из вариантов задания геометрии. Кроме этого формата, есть и другие поддерживаемые форматы, например STL, IGES. Эти форматы поддерживаются на основе импорта и дополнительных библиотек для работы с ними. В случае сложной геометрии, ее можно построить в CAD пакете, после чего экспортировать в один из воспринимаемых форматов. Однако для относительно простых и несложных моделей наиболее оптимально использование родного формата.

Наиболее часто используемые построения в нашем случае, это сфера, плоскость и параллелепипед.

Рассмотрим каждый из них подробнее:

1) Параллелепипед.

Для его построения необходимо задать две точки, противоположные по трем координатам. Тогда, разница между координатами даст длины сторон, а также размеры элемента и его положение в пространстве.

orthobrick (ax, ay, az ; bx, by, bz)

2) Плоскость.

Плоскость задается в довольно произвольном виде. Сначала задается произвольная точка с координатами $p(p_x, p_y, p_z)$, а также вектор нормали, который и показывает направление. Причем вектор задается как единичный. Из его конца видна верхняя поверхность плоскости.

plane (px, py, pz ; nx, ny, nz)

3) Сфера.

Для построения сферы необходимо указать координата цента и ее радиус.

sphere (cx, cy, cz ; r)

Кроме этих примитивов, существуют еще следующие примитивы:

- цилиндр;
- цилиндр со скругленными основаниями;
- эллипсоид;
- конус.

Использование этих примитивов позволит создать не сложную геометрию, но вполне достаточную для многих модельных задач, которым потребуется генерация сеток.

2.1 Граничные условия

Граничные условия являются также важной составной частью генерации сеток, так как всегда необходимо знать для каких граничных элементов нужно произвести задание граничных условий. Граничное условие описывается его порядковым номером (начиная с 1).

Автоматически, граничные условия обрабатываются на основе порядка следования поверхностей, то есть, как их обнаружит алгоритм построения сетки. В выходном файле с данными сетки они описываются в разделе с поверхностными элементами, которыми являются треугольники.

Для того чтобы задать свои значения для граничных условий, необходимо выбрать нужную геометрию и указать флаг –bc=value, где value – значение граничных условий. Эти условия будут задавать новую нумерацию границ. Так, если для первого объекта в виде куба задать для всех его сторон условие равное 1, то счетчик плоскостей по умолчанию (который бы выдал значения от 1 до 6 для каждой из плоскости куба) не будут изменять свое значение для всех поверхностей объекта [куба]. Это позволяет использовать свои граничные условия в случае, если для всего объекта они одинаковые. Следующий за ним объект будет иметь счетчик плоскостей равный 2. Однако если ему снова задать принудительно свое значение, то повторится описанный выше случай. Иначе, для каждой новой найденной плоскости алгоритм сгенерирует свой номер граничного условия.

Если нужно задать разные условия для поверхностей куда, например, то можно построить куб из 6 плоскостей, причем для каждой плоскости задать нужное значение для граничного условия.

Например, нужно задать волновод в виде параллелепипеда, в котором одна из поверхностей имеет граничное условие 1, а все остальные 2 (рисунок Г.1).

```
algebraic3d
solid cond_wall = plane (0, 0, 0; 0, 0, -1) -bc=1;
solid wg =  cond_wall
        and plane (0, 0, 0; 0, -1, 0)
        and plane (0, 0, 0; -1, 0, 0)
        and plane (8, 4, 48; 0, 0, 1)
        and plane (8, 4, 48; 0, 1, 0)
        and plane (8, 4, 48; 1, 0, 0) -bc=2;
tlo wg -transparent;
```

Рисунок Г.1 – Файл геометрии example.geo

Из рисунка Г.1 видно, что вместо применения геометрии для параллелепипеда применено описание его в виде 6 плоскостей, которые взаимно пересекут друг друга, тем самым обрезав область нужного объема.

Таким образом, задание граничных условий может быть довольно гибким. Эту возможность можно использовать в сторонних программах, которые используют Netgen в качестве генератора сеток.

2.2 Выходные файлы сеток

Для использования в сторонних приложениях в программе предусмотрен экспорт в различные форматы, которые включают в себя как экспорт для специализированных КЭ пакетов, так и в текстовые файлы с определенной структурой.

Среди всех экспортируемых форматов наибольший интерес представляют два формата (текстовые):

– Neural Format;
– Surface triangulation file.

Эти оба формата являются текстовыми, что позволяет их легко считывать другими программами, для этого достаточно специализированного парсера (типа ENFFM.MeshParser). Данные из файлов могут быть прочитаны и структурированы, что в дальнейшем позволит более просто работать с ними.

3 Примеры

Рассмотрим несколько примеров различной геометрии и сеток для них, построенных с помощью программы Netgen.

Большое число примеров входит в комплект поставки программы. Их можно найти в папке программы tutorials.

3.1 Сфера в кубе

Разместим сферу в центре куба. Причем, для построения куба воспользуемся плоскостями, неограниченными в пространстве. Куб состоит из 6 плоскостей. Для этого построим их, а для задания направления воспользуемся вектором нормали, указывающим направление плоскости (рисунок Г.2). В результате получатся безграничные плоскости, однако при построении они отсекутся и получится куб (рисунки Г.3 – Г.5).

```
#
# Example with two sub-domains:
#
algebraic3d
solid cube = plane (0, 0, 0; 0, 0, -1)
        and plane (0, 0, 0; 0, -1, 0)
        and plane (0, 0, 0; -1, 0, 0)
        and plane (1, 1, 1; 0, 0, 1)
        and plane (1, 1, 1; 0, 1, 0)
        and plane (1, 1, 1; 1, 0, 0);
solid sph = sphere (0.5, 0.5, 0.5; 0.3);
solid rest = cube and not sph;
tlo rest -transparent -col=[0,0,1];
tlo sph -col=[1,0,0];
```

Рисунок Г.2 – Файл геометрии example1.geo

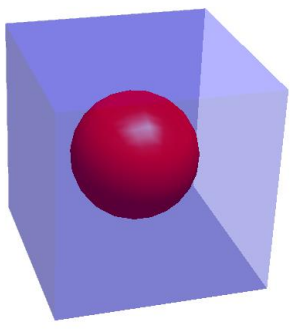


Рисунок Г.3 – Вид геометрии

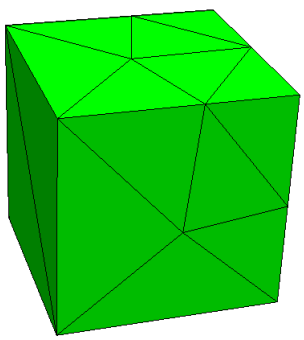


Рисунок Г.4 – Вид сетки

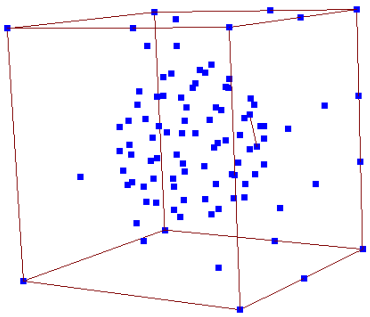


Рисунок Г.5 – Вид сетки из узлов

3.2 Сфера в кубе, куб в другом кубе

Рассмотрим следующий пример (рисунок Г.6). Поместим в центре малого куба сферу. Затем, малый куб поместим (сложим) внутрь большего (в центр). Причем расположим их так, чтобы центры всех объектов совпадали и были в центре сферы (рисунки Г.7 – Г.9).

```
#
## A cube with sphere in cube
#
algebraic3d
# cube for full box:
solid cube_main = orthobrick(0,0,0;8,8,8) -bc=1;
# cube for near field detection:
solid cube = orthobrick(2,2,2;6,6,6) -bc=2;
# sphere in center of cube
solid sp = sphere (4, 4, 4; 1);
# cube without sphere
solid main = cube_main and not cube;
# cut off small sphere:
solid rest = cube and not sp;
tlo main -transparent -col=[0,1,0];
tlo rest -transparent -col=[1,1,1];
tlo sp -col=[1,1,0];
```

Рисунок Г.6 – Файл геометрии example2.geo

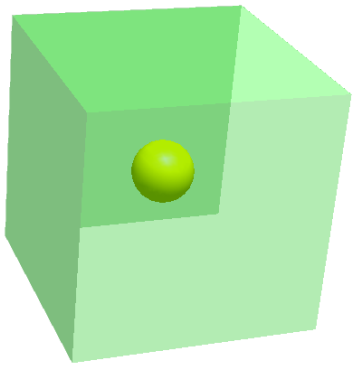


Рисунок Г.7 – Вид геометрии

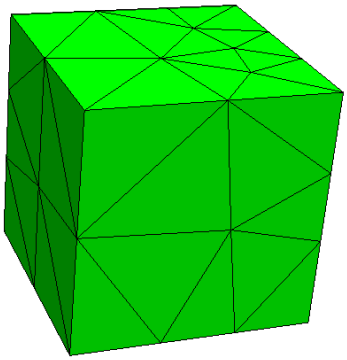


Рисунок Г.8 – Вид поверхностной сетки

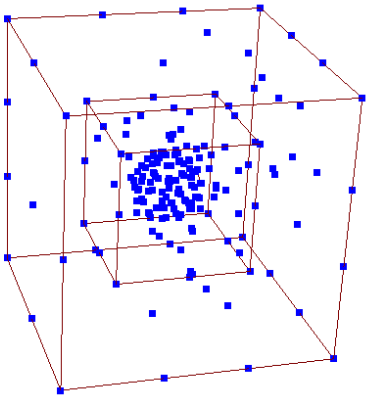


Рисунок Г.9 – Вид сетки из узлов

3.3 Сфера между двумя параллелепипедами

Для примера создадим три объекта (рисунок Г.10):

– параллелепипед 1;
– параллелепипед 2;
– сферу.

Разместим два параллелепипеда таким образом, чтобы у них была общая грань. Сферу поместим в центре, на пересечении параллелепипедов. В результате получим геометрию, показанную на рисунках Г.11 – Г.12.

```
#
## A two bricks with sphere in middle
#
algebraic3d
# cube for full box:
solid cube_main = orthobrick(0,0,0;8,8,4) -bc=1;
# cube for near field detection:
solid cube = orthobrick(0,0,4;8,8,8) -bc=2;
# sphere in center of cube:
solid sp = sphere (4, 4, 4; 1);
# cube without sphere:
solid main = cube_main and not sp;
solid main2 = cube and not sp;
# draw objects:
tlo main -transparent -col=[0,1,0];
tlo main2 -transparent -col=[0,0,1];
tlo sp -col=[1,1,0];
```

Рисунок Г.10 – Файл геометрии example3.geo

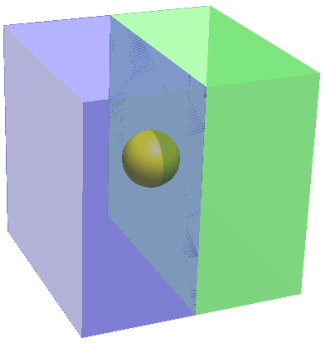


Рисунок Г.11 – Вид геометрии

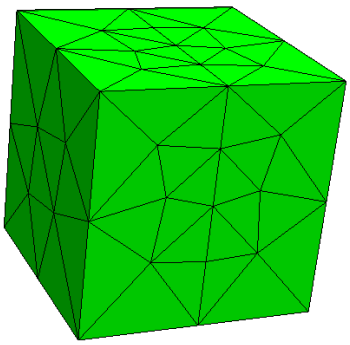


Рисунок Г.12 – Вид поверхностной сетки

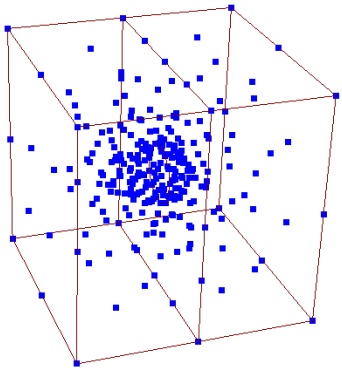


Рисунок Г.13 – Вид сетки из узлов

4 Известные проблемы

При работе с программой также возникают и некоторые проблемы, связанные отчасти с языком разработки. Например, стоит отметить, что работа с каталогами или файлами имеющими название, написанное на кириллице, будет приводить к падению приложения. Это стоит учитывать особенно тогда, когда работа производится через консоль и передаются путь к файлам. Это одна из проблем.

Кроме того, среди минусов самой среды можно отметить слабую информативность GUI, который прилагается к программе. Присутствуют и полезные опции, но реализованы они далеко не лучшим образом. Аналогичный GUI реализован в Elmer, в котором визуализация сетки более качественна и функциональна. Кроме того, программа Elmer позволяет решать различные задачи с использованием МКЭ.

Дополнительные проблемы и возможные баги можно найти в руководстве к программе, а также на сайте.

ПРИЛОЖЕНИЕ Д
(справочное)
Модуль NetgenUtils

1 Описание модуля

Модуль представляет собой расширение для работы с Netgen. Среди возможностей можно отметить:

– автоматическая генерация сетки с использованием входного файла с геометрией задачи;

– автоматическое создание файла с геометрией задачи по входным данным в формате XML (файлы *.geo).

Генерация сетки и создание файла для с геометрией, являются реализацией этапа препроцессинга, на котором происходит дискретизация области и подготовка данных для конечноэлементной модели.

1.1 Требования

Для работы необходима установленная программа Netgen. Скачать ее можно с официального сайта: http://sourceforge.net/projects/netgen-mesher/.

Кроме того, не стоит забывать, о том, что модуль используется вкупе с другими модулями, поэтому их наличие также важно. Стандартное и важнейшее требование это .Net Framework 4, Netgen 4.9 и выше.

2 Работа модуля

Модуль работает с использованием, так называемых, параметров окружения, которые говорят о конкретных путях к каталогам и названиям, характерными для конкретной системы. Таким образом, это позволяет использовать его на любых конфигурациях, стоит лишь правильно задать параметры окружения, в частности, параметры для связи с Netgen.

Параметры окружения описаны по умолчанию, а также ими можно управлять через описание параметров в файле XML. Этот файл будет иметь следующий вид, приведенный на рисунке Д.1.

```
<?xml version="1.0" encoding="utf-8" ?>
- <params>
    <param name="NetgenPath" value="d:\Program Files (x86)\Netgen-4.9.13_Win32\bin" />
    <param name="OutputPath" value="e:\WorkProj\EMFFM\Projects\EMFFM\EMFFM.MainApp\bin\Debug\output" />
    <param name="MeshFilePrefix" value="" />
  </params>
```

Рисунок Д.1 – Вид файла параметров

В зависимости от того, на каком компьютере запускается собранная программа, параметры указанные в этом файле будут использованы и программа сработает правильно.

Описание параметров идет следующим образом:

<param name=”<название параметра>” value=”<значение параметра>”/>

Работа с параметрами окружения производится с помощью специального класса *Initializer:*

Initializer init = new Initializer("../../Params.xml");

В примере показан вызов инициализатора параметров окружения (таблица Д.1), который в свою очередь считывает их из указанного XML-файла.

Таблица Д.1 – Список параметров окружения

Название	Тип	Описание
NetgenPath	String	Каталог (bin) с программой Netgen (обязательный)
OutputPath	String	Каталог для вывода результатов и файла сетки (обязательный)
MeshFilePrefix	String	Префикс для файлов с результирующей сеткой (*.mesh)
AppName	String	Название файла приложения Netgen (по умолчанию – netgen.exe)

2.1 Генерация файлов геометрии

Модуль предоставляет такую функциональность, как разбор и генерация файла геометрии, которая может быть использована для программы Netgen. Для этого были разработаны специальные классы, которые обеспечивают как генерацию файла геометрии, так и чтение XML файла с ее описанием для генерации.

Для генерации файла и описания геометрии создан формат файла XML, позволяющий довольно полно и просто описать нужную геометрию, которую необходимо сгенерировать. Структура файла может быть использована для Input API, куда интегрировать это описание. Это позволит использовать единый и универсальный формат файлов с описанием задачи, так и с описанием геометрии, которую также можно преобразовывать не только в файлы геометрии программы Netgen, но и других необходимых программ.

Фактически, разработанная структура соответствует предложенному описанию в программе Netgen через Constructive Solid Geometry (CSG). Структура XML файла стала результатам анализа проблемы и требований по автоматизации процесса генерации файлов.

2.1.1 Программная реализация выполнена в виде классов и объектов, описанных в пространстве имен NetgenUtils.GeomFiles. в нем описаны такие рабочие классы как:

– GeomFileGenerator – статический класс для генерации файла геометрии с заданными параметрами;

– Algebraic3DFile – класс, для генерации файла для описания трехмерной геометрии;

– DefinitionFileReader – реализация чтения XML файла с описанием геометрии.

Эти класс имеют публичный спецификатор и являются открытыми для использования в сторонних проектах. Также открыты и классы, описывающие отдельные элементы геометрии.

Для описания структуры файла геометрии предназначен класс GeometryData, который содержит все сведения о заданной геометрии:

– список узлов (Points);
– список ребер (Vectors);
– список тел (Solids);
– список цветов (Colors);
– список объектов для рисования (Tlos).

2.1.2 Описание формата файла.

Файл состоит из следующих секций:

– points – секция с описанием списка точек, используемых для построения примитивов;

– vectors – секция со списком единичных векторов, описывающих положение плоскости в пространстве (ее направление);

– solids – основная секция, данные из которой будут помещены в обработанном виде в генерируемый файл;

– colors – описание цветов, используемых для построения объектов модели в программе в визуальном виде;

– output – перечисление объектов, которые будут отображены.

Главным узлом файла является узел с названием <geometrydata/>, который будет использован как родительский для размещения всех нужных секций. В зависимости от описания типа задачи, секции могут быть описаны в произвольном порядке и даже не быть описанными вовсе.

2.1.3 Правила описания данных в секциях.

Описанные выше секции по факту являются обязательными и желательно иметь все секции. Однако, в зависимости от геометрии, секции с описание цветов и векторов могут отсутствовать. Это значит, что для рисования не применяются параметры цвета и то, что не используются объекты, требующие задания единичных векторов.

В секциях могут быть описаны следующие типы элементов:

– points:
  – point;
– vectors:
  – vector;
– solids:
  – sphere, orthobrick, plane, cylinder, complex, cone;
– colors:
  – color;
– output:
  – tlo.

Каждый элемент характеризуется своими атрибутами (таблицы Д.2 – Д.11), которые раскрывают сущность объекта. Основным атрибутом является атрибут *name*, который предоставляет название объекта. Это имя используется для связи элементов между собой. Так, плоскость имеет произвольную точку и вектор. Точка плоскости привязывается к точке *p1* из списка точек, а вектора *v1* – к вектору *v1* из списка векторов. Такое становится возможным благодаря ассоциациям по имени объектов.

Таблица Д.2 – Описание элемента point

Название аргумента	Тип	Описание
name	string	Название элемента
x	double	Координата x
y	double	Координата y
z	double	Координата z

Таблица Д.3 – Описание элемента vector

Название аргумента	Тип	Описание
name	string	Название элемента
direction	UnitVectorDirection	Направление вектора

В описании вектора присутствует параметр Direction, который определяет, куда будет направлен вектор. Для этого есть три единичных вектора (*i, j, k*) по каждой из осей

декартовой координатной системы. Предусмотрено автоматическое задание величин для единичного вектора. Определены следующие направления:

– вверх (top);
– вниз (bottom);
– вправо (right);
– влево (left);
– назад (back);
– вперед (forward).

Каждое направление можно описать так: смотреть на плоскость XOY так, чтобы ось OZ была направлена вдоль линии наблюдения в положительном направлении.

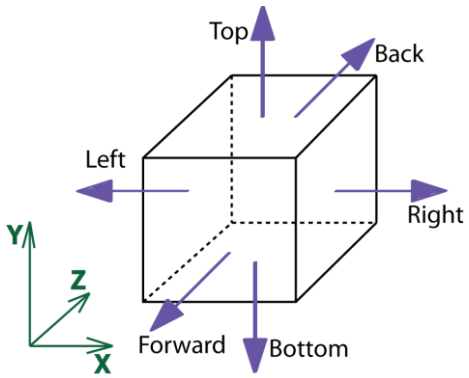


Рисунок Д.2 – Схема направлений единичных векторов

Таблица Д.4 – Описание элемента sphere

Название аргумента	Тип	Описание
name	string	Название элемента
point	string	Центр сферы
radius	double	Радиус сферы

Таблица Д.5 – Описание элемента orthobrick

Название аргумента	Тип	Описание
name	string	Название элемента
point1	string	Начальная точка отсчета элемента
point2	string	Противоположная точка

Таблица Д.6 – Описание элемента plane

Название аргумента	Тип	Описание
name	string	Название элемента
point	string	Произвольная точка
vector	string	Вектор нормали к поверхности (указание направления)

Таблица Д.7 – Описание элемента cylinder

Название аргумента	Тип	Описание
name	string	Название элемента
point1	String	Точка 1 на оси цилиндра
point2	String	Точка 2 на оси цилиндра
radius	double	Радиус основания цилиндра

Таблица Д.8 – Описание элемента cone

Название аргумента	Тип	Описание
name	string	Название элемента
point1	String	Точка 1 на оси цилиндра
radius1	double	Радиус 1-ого основания конуса
point2	string	Точка 2 на оси цилиндра
radius2	double	Радиус 2-ого основания конуса

Таблица Д.9 – Описание элемента complex

Название аргумента	Тип	Описание
name	String	Название элемента
data	String	Условие рисование сложного объекта

Таблица Д.10 – Описание элемента color

Название аргумента	Тип	Описание
name	string	Название элемента
r	int	Код цвета red
g	int	Код цвета green
b	int	Код цвета blue

Таблица Д.11 – Описание элемента tlo

Название аргумента	Тип	Описание
name	string	Название элемента
transparent	double	Прозрачен ли элемент
color	double	Цвет объекта

Для элементов из solid есть дополнительное свойство (аргумент) boundary, который указывает номер граничного условия (номер указывается с 1). Это применяется для задания особых параметров задачи, таких как ручное задание граничных условий.

В примере на рисунке Д.3 указаны все возможные элементы и секции.

```
<?xml version="1.0" encoding="utf-8"?>
<geometrydata>
  <points>
    <point name="p1" x="0" y="0" z="0"/>
    <point name="p2" x="4" y="4" z="4"/>
    <point name="p3" x="2" y="2" z="2"/>
  </points>
  <vectors>
    <vector name="v1" direction="Top"/>
    <vector name="v2" direction="Bottom"/>
    <vector name="v3" direction="Left"/>
    <vector name="v4" direction="Right"/>
    <vector name="v5" direction="Back"/>
    <vector name="v6" direction="Forward"/>
  </vectors>
  <solids>
    <plane name="plane1" p="p1" v="v3"/>
    <plane name="plane2" p="p1" v="v6"/>
    <plane name="plane3" p="p1" v="v2"/>
    <plane name="plane4" p="p2" v="v1"/>
```

```
    <plane name="plane5" p="p2" v="v4"/>
    <plane name="plane6" p="p2" v="v5"/>
    <cylinder name="cyl1" p1="p1" p2="p3" radius="5"/>
    <cone name="cone1" p1="p1" radius1="10" p2="p2" radius2="3"/>
    <orthobrick name="cube1" p1="p1" p2="p2" boundary="1"/>
    <sphere name="sp" p="p3" radius="1" boundary="2"/>
    <complex name="main" data="cube1 and not sp"/>
    <complex name="main2" data="plane1 and plane2 and plane3 and plane4 and
plane5 and plane6"/>
  </solids>
  <colors>
    <color name="col1" r="0" g="0" b="1"/>
    <color name="col2" r="0" g="1" b="0"/>
  </colors>
  <output>
    <tlo name="main2" transparent="true" color="col1"/>
  </output>
</geometrydata>
```

Рисунок Д.3 – Пример файла с описанием сложной геометрии

Некоторые примеры использования модуля, приведены в файле Samples.cs, размещенного в пространстве имен NetgenUtils.Samples.

Для правильного оформления XML файла создана схема документа на основе XML Schema 1.0. которая размещена в приложении Е.

В примере (рисунок Д.4) производится чтение исходного XML файла с описанием геометрии. Далее, после разбора данных из файла производится генерация файла для программы Netgen. В итоге будет сгенерирован файл с расширением *.geo (рисунок Д.5).

```
/// <summary>
/// Пример демонстрирует:
///     - генерацию файла геометрии по входному XML файлу.
/// </summary>
public static void Sample2()
{
        string testfile = @"E:\WorkProj\EMFFM\Defs\NetgenGeometryDefinition1.xml";
        GeomFiles.DefinitionFileReader reader = new
GeomFiles.DefinitionFileReader(testfile);
        GeomFiles.Elements.GeometryData data = reader.Data;

        GeomFiles.GeomFileGenerator.Generate("test-file.geo",
Common.FileTypes.The3D, data);
}
```

Рисунок Д.4 – Пример использования генерации файла геометрии

```
#
# Geometry file generated by NetgenUtils at 17:57
#
algebraic3d
solid cube1 = orthobrick (0, 0, 0; 4, 4, 4) -bc=1;
solid sp = sphere (2, 2, 2; 1) -bc=2;
solid main = cube1 and not sp;
tlo main -transparent -col=[0,0,1];
tlo sp -col=[0,1,0];
# End of file
```

Рисунок Д.5 – Пример сгенерированного файла с геометрией

2.2 Препроцессинг

В данном разделе описываются возможности препроцессинга, которые обеспечивает модуль. Основной возможностью является генерация конечноэлементных сеток с помощью программы Netgen (рисунок Д.6).

```
/// <summary>
/// Пример демонстрирует использование:
///     - инициализатора параметров окружения,
///     - генерацию сетки на основе некоторого входного файла геометрии.
/// </summary>
public static void Sample1()
{
    Initializer init = new Initializer("../../Params.xml");
    MeshGenerator mesh = new MeshGenerator(init.Vars);
    mesh.Generate("test1.geo", "test1.mesh");
}
```

Рисунок Д.6 – Пример использования генерации сетки

В данном примере показано как использовать возможность препроцессинга, а именно, генерации конечно-элементной сетки для заданной геометрии. Сгенерированный файл сетки далее может быть разобран модулем MeshParser и в итоге пользователь получит структурированные данные для сетки, построенной для произвольной геометрии.

При этом используется файл параметров окружения, который содержит нужные параметры для текущей системы. После этого производится вызов классов препроцессинга – построение сетки (MeshGenerator).

Построенная сетка передается не программно, а выводится в файл, указанный в параметрах запуска программы. Полученный файл можно преобразовать в структуру понятную приложению с помощью MeshParser. В таком случае получается программный объект с данными сетки.

На рисунке Д.7 представлена схема классов и структур, которые реализованы в модуле.

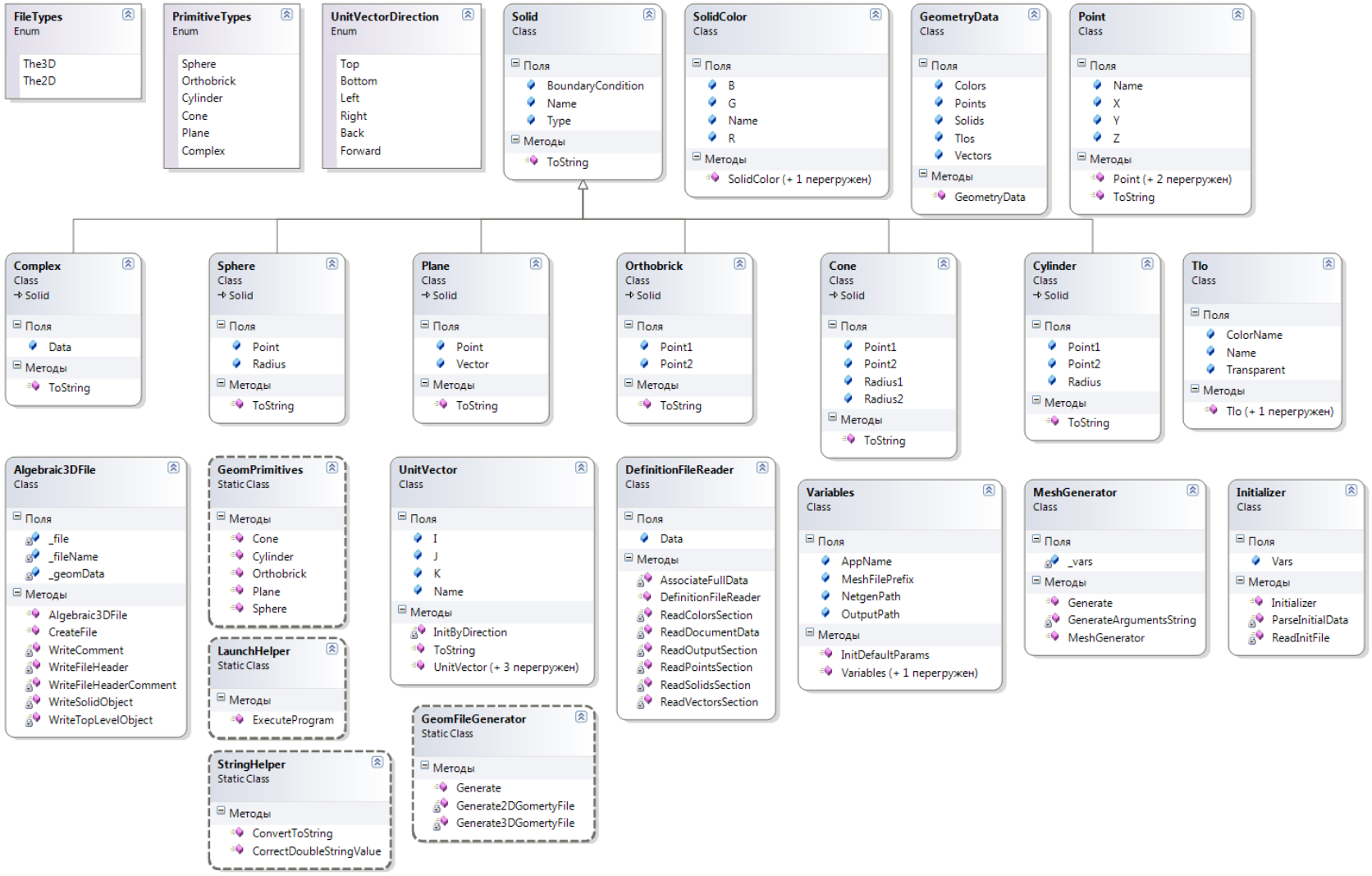


Рисунок Д.7 – Схема классов библиотеки

2.3 Запуск Netgen

Для запуска программы используется передача параметров через аргументы командной строки и сама программа запускается как новый процесс.

Есть несколько вариантов запуска для полученной строки с аргументами:

– *-batchmode -geofile=\"{0}\" -meshfile=\"{1}\" -meshfiletype=\"{2}\" -testout=\"{3}\\testout.out"*;

– *-batchmode -geofile=\"{0}\" -meshfile=\"{1}\" -meshfiletype=\"{2}\".*

Здесь: {0} – geomFile, файл с геометрией области (*.geo), {1} – outputFile, файл для вывода полученной сетки, {2} – meshType, тип сетки, {3} – OutputPath, каталог с выходными файлами (результатами).

При указании параметра –batchmode запуск приложения произойдет без графического окна, будет только консольное окно с выводом сведений о построении сетки. Построение сетки запускается в автоматическом режиме, а после чего результат записывается в файл.

3 Общие замечания

Разработанная библиотека не является самой унифицированной в плане взаимодействия со сторонним приложением. Для работы с ним используется возможность платформы .NET запускать сторонние процессы и затем дожидаться их завершения. При работе с Netgen в случае возникновения ошибки в ней, она автоматически завершит работу, при этом главное приложение, которое воспользовалось вызовом функции NetgenUtils, завершит работу с ошибкой, тогда ее перехватит обработчик исключений и выведет некоторые сведения об ошибке в лог. Дальнейшая работы программы будет невозможна. Необходимо будет найти причину и запустить заново.

Особое внимание следует уделять указания параметров геометрии, так как сама программа Netgen не всегда способно обработать некоторые е виды. Например, при создания параллелепипеда с квадратным вырезом на одной из его сторон необходимо создать еще один параллелепипед для исключения его из первого, причем его габариты по длине должны несколько превышать базовый объект. В противном случае, сетка не будет построена, т.к. произойдет неправильное распознавание геометрии.

При использовании программы следует также руководствоваться приложением В.

ПРИЛОЖЕНИЕ Е
(справочное)
Схема для документов XML с описанием геометрии

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="geometrydata">
    <xs:annotation>
      <xs:documentation>
        Описание геометрии задачи для генерации конечноэлементой сетки
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>

        <xs:element name="points">
          <xs:annotation>
            <xs:documentation>
              Список базовых точек для построения объектов
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="point">
                <xs:complexType>
                  <xs:attribute name="name" type="xs:string" use="required" />
                  <xs:attribute name="x" type="xs:unsignedShort" use="required" />
                  <xs:attribute name="y" type="xs:unsignedShort" use="required" />
                  <xs:attribute name="z" type="xs:unsignedShort" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="vectors" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              Список нормальных единичных векторов (для описания плоскостей)
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="vector">
                <xs:complexType>
                  <xs:attribute name="name" type="xs:string" use="required" />
                  <xs:attribute name="direction" type="VectorDirections" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="solids">
          <xs:annotation>
            <xs:documentation>
              Описание тел и объектов геометрии
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:choice maxOccurs="unbounded">

                <xs:element maxOccurs="unbounded" name="plane">
                  <xs:annotation>
                    <xs:documentation>
                      Описание бесконечной плоскости
                    </xs:documentation>
                  </xs:annotation>
                  <xs:complexType>
                    <xs:attribute name="name" type="xs:string" use="required" />
                    <xs:attribute name="p" type="xs:string" use="required" />
                    <xs:attribute name="v" type="xs:string" use="required" />
                    <xs:attribute name="boundary" type="xs:unsignedByte" use="optional" />
                  </xs:complexType>
```

```
      </xs:element>

      <xs:element name="sphere">
        <xs:annotation>
          <xs:documentation>
            Описание сферы
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required" />
          <xs:attribute name="p" type="xs:string" use="required" />
          <xs:attribute name="radius" type="xs:unsignedByte" use="required" />
          <xs:attribute name="boundary" type="xs:unsignedByte" use="optional" />
        </xs:complexType>
      </xs:element>

      <xs:element name="complex">
        <xs:annotation>
          <xs:documentation>
            Описание комбинированного тела
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required" />
          <xs:attribute name="data" type="xs:string" use="required" />
          <xs:attribute name="boundary" type="xs:unsignedByte" use="optional" />
        </xs:complexType>
      </xs:element>

      <xs:element name="orthobrick">
        <xs:annotation>
          <xs:documentation>
            Описание кирпича (параллелепипеда)
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required" />
          <xs:attribute name="p1" type="xs:string" use="required" />
          <xs:attribute name="p2" type="xs:string" use="required" />
          <xs:attribute name="boundary" type="xs:unsignedByte" use="optional" />
        </xs:complexType>
      </xs:element>

    </xs:choice>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="colors" minOccurs="0">
 <xs:annotation>
  <xs:documentation>
   Список цветов для отображения тел
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element maxOccurs="unbounded" name="color">
    <xs:complexType>
     <xs:attribute name="name" type="xs:string" use="required" />
     <xs:attribute name="r" type="xs:unsignedByte" use="required" />
     <xs:attribute name="g" type="xs:unsignedByte" use="required" />
     <xs:attribute name="b" type="xs:unsignedByte" use="required" />
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="output">
 <xs:annotation>
  <xs:documentation>
   Описание объектов для отображения
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element maxOccurs="unbounded" name="tlo">
    <xs:complexType>
```

```
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="transparent" type="xs:boolean" use="optional" />
            <xs:attribute name="color" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  </xs:sequence>
 </xs:complexType>
</xs:element>

<!-- собственные типы -->
<xs:simpleType name="VectorDirections">
 <xs:annotation>
  <xs:documentation>
   Направление единичного вектора в пространстве
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="xs:string">
  <xs:enumeration value="Back"/>
  <xs:enumeration value="Forward"/>
  <xs:enumeration value="Top"/>
  <xs:enumeration value="Bottom"/>
  <xs:enumeration value="Right"/>
  <xs:enumeration value="Left"/>
 </xs:restriction>
</xs:simpleType>
<!-- ENDS собственные типы -->
</xs:schema>
```

ПРИЛОЖЕНИЕ Ж
(справочное)
Модуль MeshParser

1 Описание модуля

Данный модуль предназначен для чтения текстовых файлов форматов Mesh, которые хранят описание конечноэлементных сеток. Поддерживаются форматы, создаваемые программой Netgen, которая производит генерацию конечноэлементных сеток и позволяет экспортировать их в Mesh форматы.

Модуль производит чтение и разбор файла, после чего данные могут быть использованы в другой программе, которой необходимо работать с сетками.

Список поддерживаемых форматов:

– Neutral Format;
– Surface triangulation file.

1.1 Neutral Format

Формат файла предназначен для хранения сведений об объемах и их разбиении сеткой. Файл состоит из следующих секций:

1) Nodes.

Сначала указывается число узлов в сетке. Далее следуют сами узлы в виде координат (x, y, z).

2) Volume elements.

Сначала указывается количество элементов. После количества идет перечисление элементов в следующем виде:

– номер подобласти (sub-domain number),
– номера 4-ех узлов элемента (тетраэдра).

Номера узлов начинаются с 1 (единицы). Номеру соответствует узел, описанный ранее.

3) Surface elements.

Сначала идет указание количества поверхностных элементов. Затем следует перечисление самих элементов. Каждый элемент определяется номером граничного условия и 4-мя номерами его узлов (т. к. треугольные элементы).

Номера узлов начинаются с единицы.

1.2 Surface triangulation file

Структура файла имеет следующий вид:

– Surfacemesh – файл начинается с этого ключевого слова на первой его строке;
– Количество узлов. Далее следуют координаты каждого узла (x, y, z);
– Количество поверхностных треугольников. Поверхностные треугольники ориентированы против часовой стрелки, если смотреть на объект, индекс начинается с 1.

2 Работа с модулем

Модуль разработан как библиотека классов на языке C#. Это значит, его можно собрать как динамическую библиотеку и использовать в сторонних проектах, где требуется работать с файлами сеток.

Работа модуля построена по принципу черного ящика, когда известен вход и известен выход. Причем выход может быть обработан дополнительно (вывод в XML-файл прочитанных и структурированных данных). При этом, пользователю не нужно знать какие именно данные обрабатываются, в результате работы он их получает в структурированном виде.

Рисунок Ж.1 – Схема работы модуля MeshParser

```
MeshFileParser parser = new MeshFileParser("test1.neu");
MeshData mesh = parser.Parse();
OutputHelper.OutputMeshDataToXmlFile(mesh, "test1.xml");
```

Рисунок Ж.2 – Пример вызова функций модуля MeshParser

В данном примере показан вызов парсера для разбора файла сетки, после чего результаты выводятся в файл test1.xml. Пользователя в этом коде должна интересовать только одна переменная и это mesh. Именно она и содержит данные о сетке в структурированном виде.

То есть, для работы пользователю необходимо воспользоваться строками кода из рисунка Ж.3.

```
MeshFileParser parser = new MeshFileParser("test1.neu");
MeshData mesh = parser.Parse();
```

Рисунок Ж.3 – Вызов парсера для файла сетки

Модуль описан в пространстве имен EMFFM.MeshParser.

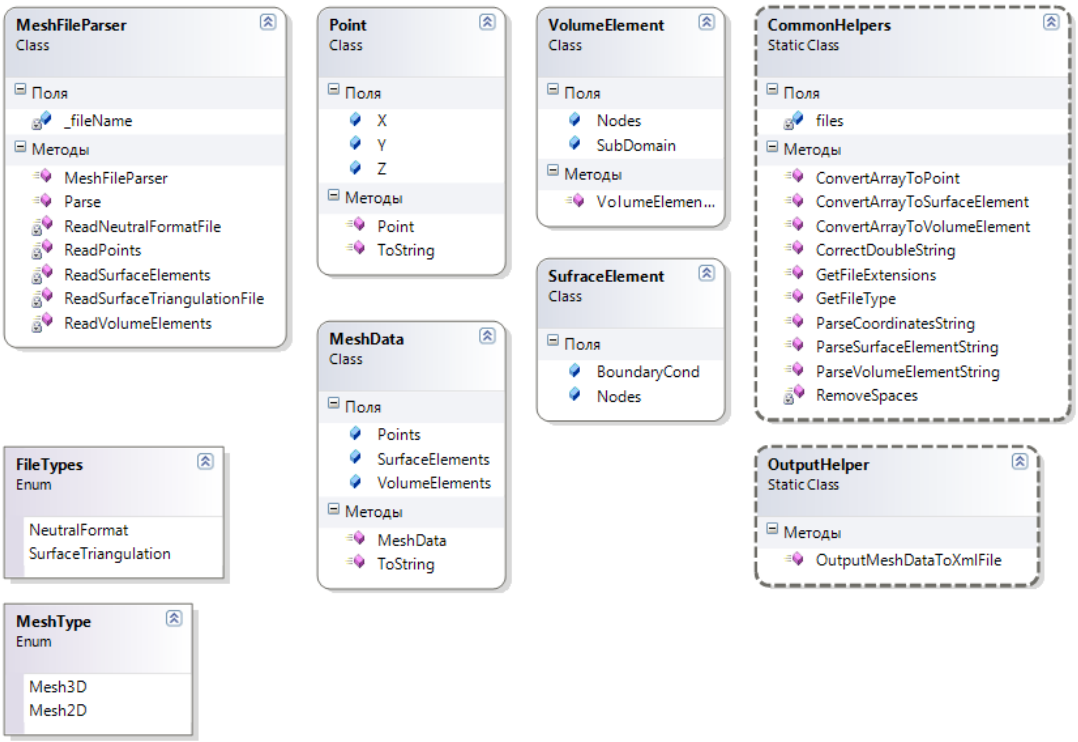


Рисунок Ж.4 – Схема классов библиотеки MeshParser

3 Дополнительные возможности

Среди дополнительных возможностей модуля можно отметить возможность экспорта данных, прочитанных из файла сеток в XML-файл (рисунок Ж.5). При этом файл становится более понятным для чтения сторонними программами, так как XML формат является стандартизированным, нежели текстовые файлы различного содержания (которым и является файл сетки, с которыми работает модель и Netgen).

```
<?xml version="1.0" encoding="utf-8"?>
<meshdata>
  <nodes count="185">
    <node number="1" x="0" y="0" z="0" />
    <!—список узлов с номерами и кординатами -->
  </nodes>
  <volelements count="860">
    <element subdomain="1" node1="40" node2="46" node3="23" node4="6" />
    <!— список элементов с номерами узлов -->
  </volelements >
  <surfelements count="282">
    <node node1="1" node2="1" node3="17" node4="47" />
...
  <!—список элементов с номерами узлов и граничными условиями (если есть) -->
  </surfelements>
</meshdata>
```

Рисунок Ж.5 – Описание структуры XML-файла

Таким образом, программа может быть использована и для более интеллектуальной обработки файлов с сетками и передачи результатов другим приложениям.

ПРИЛОЖЕНИЕ З
(обязательное)
Input API Specification

1 Введение

В спецификации приводится описание Input API, который используется в программе, структура и спецификации файлов с данными, а также программная реализация.

Входной файл представляет собой XML-файл, который поделен на секции, где каждая из секций имеет свое назначение. Для описания файла используется спецификация XML версии 1.0 и кодировка UTF-8.

Содержание файла было спроектировано таким образом, чтобы было удобно работать с поставленной задачей, которая сводится к расчету электромагнитных полей в материалах с включением наночастиц металлов с помощью метода конечных элементов (векторного МКЭ). Именно поэтому, секции направлены на наиболее точное описание целей и задач. Кроме этого, создание спецификации входного файла является одной из частей разработки унифицированного и специализированного программного комплекса.

2 Структура файлов данных

Файл приставляет собой структурированный XML-файл, который содержит в себе различные секции, каждая из которых содержит набор данных, необходимый для решения задачи и выполнения определенных действий. Описание секций начинается с ключевой секции, являющейся корневым узлом XML: <inputdata/>. В нем размещаются все остальные секции, описание которых приведено в данном разделе.

Файл состоит из следующих секций:

– Task – секция с описанием типа задачи, которая описывается в файле;

– Box – описание геометрии области решения (параллепипеда);

– Mesh – описание типа конечноэлементой сетки, которая будет применена при решении;

– Output – описание сведений для вывода;

– Actions – описание действий, которые необходимо выполнить в процессе решения;

– Options – описание опций и различных параметров, которые могут быть применены в процессе решения задачи;

– Sources – описание источников электромагнитного излучения;

– Boundary – описание граничных условий, которые применяются при решении задачи;

– Particles – описание типа, размеров и параметров генерации наночастиц, которые размещаются в исследуемой геометрии;

– Variables – описание переменных и констант, которые применяются при вычислениях в программе;

– Materials – описание материалов, которые используются в задаче;

– Geometry – описание геометрии для работы с программой Netgen;

– Plasmons – секция с описанием справочных данных по материалам, которые определяются как плазмоны (наночастицы металлов);

– Probes – описание набора пробных точек области, для изучения значений поля.

Для правильного ввода данных в файле XML используется XML Schema, которая содержит правила и наборы элементов, которые могут присутствовать в файле для Input API.

В файле есть обязательные секции, которые должны быть указаны независимо от задачи. Они будут общими и определяющими параметры выполнения задачи. Кроме того эти секции являются общими для любого типа задачи.

Список общих секций:

- Task;
- Actions;
- Materials;
- Plasmons (не обязательная, зависит от задачи);
- Sources;
- Output;
- Boundary;
- Variables.

1.1 Секция Task.

Секция содержит название типа задачи (таблица 3.1), которая будет описана во входном файле. Это позволяет программе точно знать какие секции файла понадобятся, а также выполнить правильные действия, описанные в сценарии со списком действий в секции Action. Является обязательной. Если секция не объявлена, выполнение программы будет прекращено.

Таблица 3.1 – Описание типов задач

Название	Описание
Common	Задачи общего типа, использующие чисто встроенные функции программы
Netgen	Задачи со сложной геометрией, для которой используется программа Netgen. (предпочтительный вариант)

Примечание: В данной версии программы, программа работоспособна полностью только для задачи Netgen.

1.2 Секция Box

Секция описывает общую геометрию задачи. Изначально в текущей версии API поддерживается описание одного тела в виде параллелепипеда. Он может представлять как пластину, так и плоскость (при соответствующем задании координат узлов). Описание параллелепипеда производится в виде указания координат двух точек P1 и P2 (рисунок 3.1).

Секция может описывать плоскость, которую необходимо разбить на плоские элементы (прямоугольники или треугольники). Для этого необходимо при задании координат точек указать координату Z равной нулю.

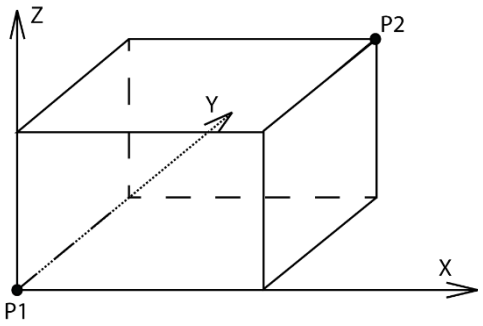


Рисунок 3.1 – Вид параллелепипеда, описанного в секции Box

Пример описания секции:

```
<box>
   <point x="0" y="0" z="0"/>
   <point x="10" y="5" z="2"/>
</box>
```

Здесь указываются 2 точки и их координаты по каждой из оси (рисунок 3.1).

Примечание: При указании координат точки необходимо указывать все 3 координаты (три атрибута XML элемента)! В противном случае данные не будут прочитаны корректно!

Примечание: Секция работает совместно с секцией Mesh, в которой описывается тип элементов сетки. Это необходимо учитывать для сохранения согласованности введенных данных. Согласованности следующие:

– плоскость для разбиения на плоские элементы: координата *Z* в секции Box равно нулю, тип элементов выбран Rectangle или Triangle. (приложение 3, раздел 1.3);

– параллелепипед для разбиения на трехмерные элементы (параллелепипеды или тетраэдры): все координаты заданы, причем координаты одной из точек больше координат другой по каждой из компонент, тип элементов выбран Parallelepiped или Tetrahedron (приложение 3, раздел 1.3).

Таким образом, секция Box работает совместно с секцией Mesh.

1.3 Секция Mesh

В данной секции приводится описание типа и размера элементов, которые будут использованы для дискретизации исходной области, заданной в секции Box. Здесь указываются типы элементов (см. список ниже), а также их размеров. В зависимости от типа элемента, его размеры могут состоять как из двух (*x, y*), так и из трех (*x, y, z*) параметров.

Типы элементов, которые присутствуют в программе:

- 2D:
  - Triangle;
  - Rectangle;
- 3D:
  - Parallelepiped;
  - Tetrahedron.

На основании описания данной секции выполняется построение регулярной сетки. В данной версии не поддерживается описание и генерация нерегулярной сетки, которая состоит из элементов различного размера.

Пример описания секции:

```
<mesh>
   <size x="1" y="1" z="1"/>
   <type value="Parallelepiped"/>
</mesh>
```

Секция используется в случае использования типа задачи *Common*, а также при генерации сетки встроенными средствами.

1.4 Секция Output

Данная секция описывает подробно файлы и их типы, а также относительные пути, которые будут использования для вывода результатов работы программы. То есть, здесь

приводится описание файлов, в которые будут помещены результаты выполнения описанных в секции Actions действий.

Описание секции построено по следующему принципу:

– указывается каталог, в который помещаются все файлы вывода (path);

– указывается список файлов и их типов, которые описывают их содержание;

– префикс для результирующих файлов, которые будут генерироваться программой (fileprefix).

Файлов может быть несколько, в зависимости от задачи, которая описывается во входном файле.

Примечание: Файл одного типа может быть описан только один раз! В противном случае возникнет ошибка! Т. е., например, необходимо вывести сетку в файл. Для этого указывается название файла и его тип в виде: <datafile type="Mesh" name="mesh.txt"/>. Это значит, что сетка будет выведена именно в этот файл и никуда более.

Таблица 3.2 – Типы файлов и их содержание

Тип файла	Описание	Стандартное расширение
Mesh	Содержит сведения о сетке, список элементов и их узлов и ребер)	txt, xml
Particles	Содержит данные о частицах, которые генерируются при собственном задании закона распределения.	txt
Matrix	Глобальная матрица, построенная для задачи (матрицы [*K*] в СЛАУ)	txt
Result	Единичный результат выполнения некоторых действий	txt

Пример описания секции:

```
<output>
   <path value="output/"/>
   <datafile type="Mesh" name="mesh.txt"/>
</output>
```

1.5 Секция Actions

Секция описывает список действий, которые необходимо выполнить в программе для получения результата. При описании действий можно задать их порядок (если их несколько) путем указания кода действия. Это описание формирует так называемый сценарий работы программы.

Пример описания действия:

```
<action code="1" value="GenerateMesh"/>
```

В описании действия присевают два параметра (аргумента XML элемента):

– Code – код или номер действия. Используется в случае описания нескольких действий и определяет порядок их выполнения;

– Value – название действия, которое необходимо выполнить. Список названий строго определен в спецификации (таблица 3.3).

Таблица 3.3 – Названия определенных в Input API действий

Название	Описание
GenerateMesh	Выполнение генерации сетки на основании данных, указанных в секциях входного файла (Box, Mesh, Output)
GenerateParticles	Генерация частиц, которые помещены в заданный объем
BuiltGlobalMatrix	Построение глобальной матрицы, на основе разбиения области на элементы
Solve	Решения всей задачи целиком с получением результатов

При генерации частиц в результате получаются номера элементов, на которые разбита область задачи, куда будут помещены частицы. Частица представляет собой элемент, наделенный особыми свойствами. В зависимости от задачи, свойства и значения могут отличаться.

1.6 Секция Options

В данной секции описываются опции и параметры, которые используются при выполнении действий и операций в программе. Они определяют параметры вывода, решения, опции логирования и прочее, определяют метод решения задачи.

Большинство параметров обязательны к описанию, т.к. определяют процесс выполнения, особенно это касается вывода результатов.

Список возможных опций и параметров приведен в таблице 3.4.

Таблица 3.4 – Список опций

Название	Значения (тип)	Описание
IsOutputToFile	True / false (boolean)	Выводить ли результаты работы в файл
IsBuiltEdges	True / false (boolean)	Строить ли сетку ребрами (на основе узловой)
IsAppend	True / false (boolean)	Дописывать ли данные в файл при выполнении нескольких действий
WithParticles	True / false (boolean)	Добавлять ли частицы к рассматриваемому объему
NetgenParams	String	Определяет название и размещение файла
InTimeDomain	True / false (boolean)	Решение задачи во временной области
InFrequencyDomain	True / false (boolean)	Решение задачи в частотной области
WithoutIterations	True / false (boolean)	Определяет решение задачи итерационным (варьирование значением) образом или для одного значения

1.7 Секция Variables

В данной секции записываются переменные и константы, которые могут быть переопределены для определенной задачи (точнее говоря, их значения, отличаются от значений, установленных по умолчанию в программе).

Таблица 3.5 – Список переменных и констант

Название	Тип данных	Значение по умолчанию
Time	double	Время решения (во временной области)
TimeDelta	double	Шаг по времени
FrequencyDelta	double	Максимальной отклонение частоты от исходного значения
FrequencyStep	double	Шаг изменения частоты

1.8 Секция Sources

В секций описываются источники электромагнитного поля, а также их параметры. Источники могут быть указаны по их типу, которые строго определены программой. Причем каждый источник имеет свои параметры и значения. Описание типов источников приведено в таблице 3.6. Если источников несколько, то указываются несколько источников, которые затем последовательно применяются при решении.

Таблица 3.6 – Список типов источников

Название	Параметры	Описание параметров	Описание источника
source	freq	Частота работы (MHz)	Параметры для задания монохроматической волны
	amplitude	Амплитуда колебаний (E_0)	
	lambda	Длина волны оптического источника (метры)	

Для указания границы, на которой определен источник, используется параметр boundary, значение для которого указывается по аналогии с граничными условиями (номер границы).

В секции можно будет размещать и другие типы источников с расширенным набором параметров.

1.9 Секция Boundary

Секция описывает список граничных условий, которые используются при решении и расчете электромагнитных полей и могут быть заданы в соответствии с требуемым типом. Среди параметров также можно задать их условия применения, то есть где именно будут применены (для всего объема задачи или же для его части).

Описание условий начинается с указания элемента <condition/>.

Таблица 3.7 – Типы граничных условий

Название	Параметры	Описание параметров	Описание источника
Dirichlet	boundary	Номер границы (поверхностей)	Граничное условие Дирихле
	value	Значение	

Пример описания условия Дирихле:

<condition type="Dirichlet" boundary="1" value="10"/>

1.10 Секция Particles

Секция описывает правила и законы, по которым размещаются частицы различных материалов в заданном объеме. Это позволяет создавать модели, которые будут уникальны каждый раз и иметь различную, структуру и свойства.

Так как пока законы распределения не определены, то секция не работоспособна и не имеет описания. Для ее использования можно создать тестовый метод, который будет случайным образом распределять частицы.

Принцип размещения частиц основан на том, что закон распределения формирует определенные номера или координаты, которые определяют положение частиц в сетке, построенной для решения задачи. В зависимости от способа задания положения частиц в разделе используются свои описания. Эти описания ассоциированы с алгоритмами, которые на основе Input API выберут нужный и сгенерируют собственные “псевдочастицы” (так называемые виртуальные описания для частиц, которые используются в программе).

Для описания расположения частиц используется один закон, определяющий формулировку для генерации номеров элементов, которые и будут представлять собой частицы.

Среди параметров генерации можно отметить следующие:

– количество генерируемых частиц;

– параметры закона распределения;

– нижняя и верхняя границы получаемых номеров (в соответствии с размерностью сетки).

Для описания закона в секции используется узел <law/> (закон), который описывает выбранный закон и его параметры.

Таблица З.8 – Законы распределения частиц в объеме

Название	Параметры		Описание
Random	count	Количество частиц	Случайное распределение частиц
	lower	Нижняя граница	
	top	Верхняя граница	
	type	Тип алгоритма (например, MersenneTwister)	
	material	Материал (название), который применяется для частиц (из раздела Materials)	

Для работы с законами распределения и использовании генераторов случайных чисел используется библиотека Math.NET Numerics.

Пример описания закона распределения:

<law name="Random" type="Random" count="10" lower="5" top="50"/>

или

<law name="Random" type="MersenneTwister" count="10" lower="5" top="50"/>

Кроме автоматического задания номеров частиц с помощью закона, присутствует и принудительное задание частиц в виде секции с описанием номера элемента сетки, в котором она задана. Здесь также задается и материал, из которого изготовлена частица.

Пример описания узла:

<particle number="10" material="MyMaterial2"/>

Описания в секции сопровождаются зависимостью от описания секции с материалами (Materials).

1.11 Секция Materials

В данной секции производится описание всех материалов и их электромагнитных параметров, которые будут использованы при решении задачи. В таблице Ж.9 представлены параметры материалов, необходимые для описания.

Стоит также отметить то, что в текущей вариации Input API необходимо описание всех материалов и их параметров, которые будут использованы при решении задачи. Это образовывает справочный материал, используемый при решении. Так, для всех элементов по умолчанию задается материал, указанный как базовый. Далее, для частиц в соответствии с номерами задается отличный материал, указанный в параметре для частиц в секции Particles.

Список материалов используется для установки параметров частиц, а также для элементов в свободном пространстве рассматриваемой области, чтобы описать ее электромагнитные характеристики.

Таблица 3.9 – Параметры материалов

Название	Описание
eps (ε)	Диэлектрическая проницаемость
mu (μ)	Магнитная постоянная
sigma (σ)	Электрическая проводимость
default	Говорит о том, что данный материал является материалом по умолчанию для всей области

Пример описания материалов:
<material name="Air" eps="1" mu="1"/>
<material name="Metall" eps="1.1" mu="1.02"/>

1.12 Секция Geometry

Данная секция используется для задания геометрии задачи, а также параметров объектов, которые располагаются в исследуемом объеме. Основное предназначение секции это описание параметров для программы Netgen

Таблица 3.10 – Элементы секции описания геометрии

Название	Описание
geofile	Файл с геометрией исследуемой задачи
meshfile	Файл сетки
isgenerategeomfile	Генерировать ли файл геометрии на основе XML файла
domains	Описание списка областей рассматриваемой геометрии и присвоение им материалов

Секция включает в себя описание всех объемов (областей), которые рассматриваются в задаче. Здесь происходит назначение материалов в соответствии с расположением области, а также названия области. Формально, они не используются (название и код), но удобно их применять для четкой формулировки рассматриваемых объектов.

Таблица 3.11 – Параметры области (domain)

Название	Описание
code	Код объекта
name	Название объекта
material	Имя материала, их которого изготовлен объект
plasmon	Указывает на то, что материал рассматривается с точки зрения плазмона (для которого известны соответствующие параметры дисперсии)

1.13 Секция Plasmons

Секция содержит сведения об используемых материалах, которые считаются плазмонами. Такие параметры выявлены для разных постановок описания дисперсии материала. Поэтому в зависимости от описания закона дисперсии описания могут меняться.

Например, для используемого закона [3] (действительная часть):

$$\varepsilon = 1 - \frac{\omega_{pl}^2 \tau^2}{1 + \omega^2 \tau^2},$$

где ω_{pl} – плазмонная частота свободного электронного газа;

τ – время релаксации свободного электронного газа.

Параметры описываются в следующем виде:

```
<plasmons>
  <plasmon material="Ag" plasmow="9.1" relax="29"/>
  <plasmon material="Au" plasmow="9.1" relax="40"/>
  <plasmon material="Cu" plasmow="8.8" relax="40"/>
 </plasmons>
```

1.14 Секция Probes

В данной секции описываются специальные точки, называемые пробными, которые предназначены для исследования поля в конкретных местах. Описание точки заключается в указании координат (*x*,*y*,*z*) размещения точки, а также направления исследуемого поля. Так как в трехмерном случае поле имеет три компоненты в пространстве, то необходимо указать одну из них или все, если нужны полные сведения.

Такие точки еще можно назвать приемниками, которые позволяют получить некоторое значение о распределении поля в рассматриваемом объеме.

Таблица 3.12 – Параметры пробных точек

Название	Тип	Описание
x, y, z	Double	Координаты точки
dir	FieldDirection (X, Y, Z, All)	Направление исследуемого поля в точке

2 Описание работы с входными файлами в программе EMFFM

В соответствии с описанием, приведенном в разделе 3.1, в программе реализованы специализированные классы для работы – Input API, а также структуры данных, которые расположены в пространстве имен (namespace) <AppName>.Input. В совокупности они обеспечивают обработку входных данных и предоставление их в том виде, в котором ядро программы их сможет обработать для выполнения поставленных задач.

В таблице 3.13 и 3.14 приведено описание классов и структур, которые используются при работе с Input API.

Таблица 3.13 – Описание классов пространства имен <AppName>.Input

Название	Описание
InputFileReader	Класс предназначен для чтения входного файла по секциям и их разбора. Расшифровывает XML данные и создает объект InputData, который хранит входные данные задачи.
InputHelper	Вспомогательный класс, который содержит справочные методы, позволяющие работать с входными данными.
InputData	Класс, который содержит поля с данными, прочитанными из файла в виде определенных структур в программе

Список структур и классов, которые используются можно менять в зависимости от требований к программе и что необходимо автоматизировать, скрыть в Input API.

Таблица 3.14 – Описание структур и классов пространства имен <AppName>.Input

Название	Описание
Box	Область решения задачи, ее размеры
MeshElement	Элементы сетки (в случае использования встроенного генератора сеток)

Продолжение таблицы 3.14.

Название	Описание
Options	Опции и параметры программы
Action	Описание действий
Output	Содержит сведения о выходных файлах, куда должны быть помещены результаты
Variable	Список переменных
Particle	Параметры частиц (общие)
ParticleD, ParticlesS, ParticlesM	Описание параметров расположения частиц в объеме
Material	Параметры материала
PlasmonData	Описание параметров плазмона (из справочных данных во входном файле)
Source	Источники излучений и их параметры
Boundary	Описание граничных условий
Geometry	Содержит параметры геометрии, которые используются для работы с программой Netgen
Domain	Описание области, ее материал, номер и название
ProbePoint	Описание пробной точки

Процесс обработки входного файла построен по следующему принципу:

– программа получает путь к файлу с входными данными;

– производится передача полного пути к файлу в класс InputFileReader;

– происходит загрузка файла в объект XmlDocument;

– для всех секций, содержащихся в XmlDocument, производится их разбор и заполнение данными объекта InputData;

– заполненный объект InputData возвращается в место вызова InputFileReader.

Таким образом, в итоге получаются данные, представленные в виде понятных и доступных структур в объекте InputData, который и является главной движущей силой работы программы, т.к. содержит все необходимые данные для решения задачи, а также сценарий ее выполнения.

Процесс чтения и разбора секций выполняется путем обработки каждой секции свой функцией (таблица 3.15).

Таблица 3.15 – Методы разбора секций входного файла

Название метода	Заполняемая секция	Аргументы
ReadBoxSection	InputData.Box	XmlNodeList xmlNodeList, ref InputData data
ReadTask	InputData.Task	--//--
ReadMeshSection	InputData.Element	--//--
ReadOutputSection	InputData.Output	--//--
ReadActionSection	InputData.Actions	--//--
ReadOptionsSection	InputData.Options	--//--
ReadVariablesSection	InputData.Variables	--//--
ReadSourcesSection	InputData.Souces	--//--
ReadBoundarySection	InputData.Boundaries	--//--
ReadParticlesSection	InputData.Particles	--//--
ReadMaterialsSection	InputData.Materials	--//--
ReadGeometrySection	InputData.Geometry	--//--

Продолжение таблицы 3.15.

Название метода	Заполняемая секция	Аргументы
ReadPlasmonsSection	InputData.Plasmons	XmlNodeList xmlNodeList, ref InputData data
ReadProbesSection	InputData.Probes	--//--

При разборе предопределенных названий, таких как тип элемента, тип файла, тип действия и др. используются перечисления, которые описаны в коде программы и содержат список возможных действий. Эти перечисления описаны в таблице 3.16.

Кроме того, для перечисленных параметров имеется описание в схеме документа, чтобы пользователь мог выбрать необходимые из определенного набора параметров.

Таблица 3.16 – Описание перечислений, используемых при работе с Input API

Название	Значения	Описание
ElementType	Triangle, Rectangle, Parallelepiped, Tetrahedron	Описание типов элементов, которые используются для построения сетки
ActionNames	Generate_Mesh, Solve, BuiltGlobalMatrix	Описание действий, которые определяются в секции Actions
DatafileType	Mesh	Описание типов файлов с данными, что и куда будет выводится в результате выполнения действий
BoundaryTypes	Dirichlet, Neumann, Mixed	Типы граничных условий
TaskClasses	Common, Netgen	Типы решаемых задач
SourceTypes	Simple, Electric, Magnetic	Типы источников
ParticlesType	Law, Single, Many	Типы частиц (для собственной генерации)
DistributionTypes	Random, MersenneTwister	Законы распределения частиц
PlasmonMaterials	Ag, Au, Cu	Типы материалов для плазмонов
FieldDirection	X, Y, Z, All	Описание направлений поля

Многие перечисления и параметры, связанные с обработкой входных данных, связаны с конкретной задачей и ее вариантами постановки. Поэтому, в дальнейшем описанные структуры и перечисления будут меняться и конкретизироваться под решаемые задачи.

Примечание: Список значений, указанный в таблице 3.16 может измениться и дополниться в новой версии спецификации! Для сохранения работоспособности программы и адекватности входных файлов изучите последнюю версию текущей спецификации!

2.1 Выполнение программы с использованием Input API

Выполнение программы построено на основе списка действий, которые были прочитаны из входного файла и на основании ассоциаций, которые устанавливают порядок выполнения каждого действия. Интегрированный в программу Input API предоставляет средства для обработки входных данных и их анализа, без необходимости применения сложных многостадийных средств ввода и описания входных параметров. Механизм работы (рисунок 3.2) с входными файлами тесно интегрирован с ядром программы, при этом, его всегда можно изменить путем внесения изменений в Input API, а также в ассоциации, которые связывают входные данные и параметры с обработчиками программы (точнее говоря, то, что в программе отвечает за выполнения конкретного действия в зависимости от типа задачи).

Работа приложения основана на том, каким образом определены ассоциации действий (Actions) из Input API. Именно они определяют сценарии вызова функций для достижения поставленной в задаче цели.

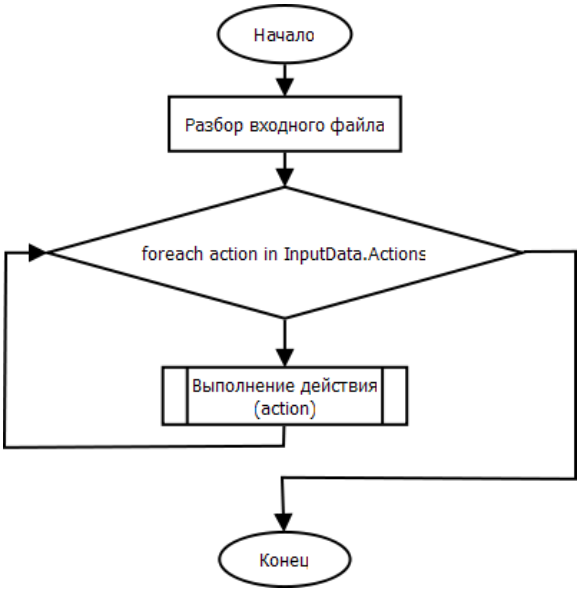


Рисунок 3.2 – Схема работы программы

Программа сначала выполняет разбор входного файла, после чего, на основании считанных данных, производит выполнение действий, которые были определены в файле (рисунок 3.2). Последовательность выполнения действий определяется их расположением в списке, который отсортирован в порядке возрастания кода. Сортировка выполняется в процессе разбора входного файла (Input API), то есть когда дело доходит до выборки действий из списка, они уже отсортированы (в методе ReadActionSection()).

Ассоциация действий с их содержанием выполнена в программе строго, что предотвращает появление ошибок в процессе выполения, связанных с непоследовательной обработкой данных. Разработчик уже протестировал и проверил работоспособность программы и ее действий. Последовательность операций, выполняемая для каждого из действий, может быть переопределена только разработчиком в случае необходимости. Это должно гарантировать целостность программы и правильный порядок выполнения операции при выполнении всех действий.

Для обработки списка действий, программа имеет специальный класс AppController, который представляет собой класс, управляющий выполнением программы. Причем, классы управления, выполнены таким образом, чтобы не было зависимости от типа приложения (веб-приложение, десктопное приложение (консольное или WinForms) и т.п.). Это сделано с целью создания универсального ядра для решения определенного типа задач, Input API должен максимально этому содействовать.

Класс AppController содержит в себе описание методов, которые ассоциированы с действиями, определяемыми Input API. Так, например, для обработки действия Generate_Mesh в ActionOperations описан метод GenerateMeshAction(), который выполняет построение сетки и вывод ее в файл на основе исходных данных.

Примечание: Текущая версия программы выполнена в виде консольного приложения и классы контроллеров имеют в текущем варианте определенны набор действий. В дальнейшем, при развитии приложения, для тестирования контроллеров можно будет создавать тесты с возможностью полноценно протестировать приложение и выполнить полное покрытие тестами всех его классов и методов.

2.2 Обработка действий в программе EMFFM

В программе сопоставлены определенные действия, которые описаны в Input API. По сути, это действия, которые определяет сама программа, то есть то, что она может сделать. Input API может быть подстроен под конкретную задачу, но для этого необходимо будет выполнить проектирование с учетом новой предметной области. Текущая версия Input API не столь универсальна.

Выполнение действия определяется разработчиком в приложении, которое использует Input API. Он описывает классы и методы, а также описывает порядок вызова тех или иный действий. При этом стоит отметить то, что написание ассоциации к действию определяется следующими правилами:

– метод, который описывает ассоциацию для действия, содержит локальные переменные и вызывает методы в заданном порядке. Данные для работы он черпает из объекта InputData, который содержит данные, считанные из входного файла;

– метод является полностью законченным и выполняет действие от начала до конца, включая вывод в файл и прочее;

– действия не зависят друг от друга. Каждый метод для действия выполняет свои операции, даже, если они одинаковые.

Таким образом, описанные требования задают строгие правила разработки ассоциаций для действий, описываемых в Input API.

Независимость от других действий позволяет выполнять разнородные операции, причем каждое из них будет выполнено в своем порядке, в том, как определено разработчиком для действия.

При обработке действия используются параметры, описанные в других секциях. Так, например, при выполнении действия по генерации сетки (встроенный генератор), используются секции с опциями (Options), параметры выхода (Output), описание геометрии берется из секции Box, данные для сетки – из Mesh. В результате получаются файлы выхода (если установлены соответствующие опции), которые содержат данные о построенной сетке. Это связано с тем, что действия могут требовать какие-то параметры для их выполнения. Секций (в текущей версии Input API) имеют глобальный характер. Это значит, что все описанные действия в секции Actions будут работать с одними и теми же параметрами. Поэтому, стоит отметить, что если выполнять однотипные действия, например два раза генерировать сетку, то данные в файле будут соответствовать последнему выполненному действию. Для установки возможности записи в файл с добавлением (Append) необходимо указать соответствующую опцию в секции Options.

Примечание: В дальнейшем развитии спецификации и Input API действия будут расширены по возможностям и существенно модифицированы. Будет добавлено возможность связи действия с остальными секциями индивидуально, то есть они смогут иметь свои параметры вывода, опции, переменные и прочее.

```
private void GenerateMeshAction()
    {
    // 1. Подготовка объекта для описания геометрии области дискретизации
       GeomData geomData = InputHelper.GetGeomData(_data);
       // 2. Определение типа сетки и выбор соответсвующего класса для ее генерации
       switch (_data.Element.Type)
       {
          case ElementType.Rectangle: // генерация сетки для прямоугольников
             MeshBuilder.GenerateRectangleMesh(geomData, _data.Output, _data.Options);
             break;
          case ElementType.Triangle: // генерация сетки для треугольников
             MeshBuilder.GenerateTriangleMesh(geomData, _data.Output, _data.Options);
             break;
          case ElementType.Parallelepiped: // генерация сетки для параллелепипедов
             MeshBuilder.GenerateParallepipedMesh(geomData, _data.Output, _data.Options);
             break;
          case ElementType.Tetrahedron: // генерация сетки для тетраэдров
             // TODO: протестировать генерацию сетки для тетраэдров!
             MeshBuilder.GenerateTetrahedronMesh(geomData, _data.Output, _data.Options);
             break;
       }
    }
```

Рисунок 3.3 – Метод, который ассоциирован с действием генерации сетки (GenerateMesh)

2.3 Использование опций и переменных

Программа, построенная на основе ядра Input API, во многом зависит от опций и параметров, которые указываются во входном файле. Они могут управлять программой дополнительно, что делает возможным выполнение дополнительных действий или запрет существующих. Например, для решения задачи один раз, без необходимости пересчитывать меняющиеся параметры, можно установить опцию WithoutIterations в значение false.

Переменные (Variables), представляют еще одну форму опций. С их помощью можно задать специальные переменные, которые необходимы для решения задачи. При этом они также будут заданы в одном входном файле, без необходимости исправлять код программы. Действие переменных должно быть совместным с опциями. Например, при решении задачи по итерациям, в переменных задается шаг изменения параметра и отклонение от базового параметра.

3 Применение Input API

Для использования Input API необходимо иметь сведения о его структуре, которая описана в части 1, а также коды приложения, которое использует его.

3.1 Создание файлов XML

Для того, чтобы использовать Input API необходимо правильно составлять файлы XML, которые являются собственно ядром. Для правильного и более удобного описания файла разработана схема документа, которая позволяет использовать предопределенные типы, описывает полный набор узлов. Который может использоваться. Схема построена на основе XML Schema 1.0 и описывается в виде XSD файла. Пример его использования показан на рисунке 3.4.

```
<?xml version="1.0" encoding="utf-8" ?>
- <inputdata xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="e:\WorkProj\EMFFM\Defs\InputApi.xsd">
    <task name="Netgen" />
  + <geometry>
  </inputdata>
```

Рисунок 3.4 – Пример подключения схемы для файла:

Так как это предварительная версия спецификация файла и будет изменяться и дополняться, схема документа реализована в простом виде.

Использование схемы позволяет использовать получать справку по узлам документа, а также четко описывать их содержание. Для значений, которые являются строго предопределенными, предусмотрено описание списка допустимых значений. Ознакомиться со схемой можно в приложении И.

ПРИЛОЖЕНИЕ И
(справочное)
Схема для документов XML из Input API

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Schema file for Input API files for EMFFM.
      XML Schema, version 0.1, June 2012.
      Copyright 2012, Digiman. MIT Public Licence.
    </xs:documentation>
  </xs:annotation>

  <xs:element name="inputdata">
    <xs:annotation>
      <xs:appinfo>
        Input file for EMFFM Input API
      </xs:appinfo>
      <xs:documentation>
        Параметры задачи и сценарий ее решения
      </xs:documentation>
    </xs:annotation>
  </xs:element>

  <!-- Описание элементов -->

  <xs:element name="task">
    <xs:annotation>
      <xs:documentation>
        Указание типа задачи
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:attribute name="name" type="TaskNames" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="geometry">
    <xs:annotation>
      <xs:documentation>
        Параметры геометрии задачи для работы с Netgen
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:all>
        <xs:element name="geofile">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="meshfile">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="isgenerategeomfile">
          <xs:complexType>
            <xs:attribute name="value" type="xs:boolean" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="domains">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="domain">
                <xs:complexType>
                  <xs:attribute name="code" type="xs:unsignedByte" use="required" />
                  <xs:attribute name="name" type="xs:string" use="required" />
                  <xs:attribute name="material" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="cofactor">
```

```
      <xs:complexType>
        <xs:attribute name="value" type="xs:double" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:all>
 </xs:complexType>
</xs:element>

<xs:element name="sources">
 <xs:annotation>
  <xs:documentation>
   Описание списка источников и их параметров
  </xs:documentation>
 </xs:annotation>
 <xs:complexType mixed="true">
  <xs:choice>
   <xs:element name="source">
    <xs:complexType>
     <xs:attribute name="freq" type="xs:double" use="required" />
     <xs:attribute name="amplitude" type="xs:double" use="required" />
     <xs:attribute name="boundary" type="xs:integer" use="optional"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="esource">
    <xs:complexType>
     <xs:attribute name="freq" type="xs:unsignedByte" use="required" />
     <xs:attribute name="amplitude" type="xs:double" use="required" />
    </xs:complexType>
   </xs:element>
   <xs:element name="msource">
    <xs:complexType>
     <xs:attribute name="freq" type="xs:unsignedByte" use="required" />
     <xs:attribute name="amplitude" type="xs:double" use="required" />
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>
</xs:element>

<xs:element name="boundary">
 <xs:annotation>
  <xs:documentation>
   Описание граничных условий
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="condition" maxOccurs="unbounded">
    <xs:complexType>
     <xs:attribute name="type" type="xs:string" use="required" />
     <xs:attribute name="boundary" type="xs:integer" use="required" />
     <xs:attribute name="value" type="xs:double" use="required" />
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="materials">
 <xs:annotation>
  <xs:documentation>
   Описание списка материалов и их параметры
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element maxOccurs="unbounded" name="material">
    <xs:complexType>
     <xs:attribute name="name" type="xs:string" use="required" />
     <xs:attribute name="eps" type="xs:decimal" use="required" />
     <xs:attribute name="mu" type="xs:decimal" use="required" />
     <xs:attribute name="sigma" type="xs:decimal" use="required" />
     <xs:attribute name="default" type="xs:boolean" use="optional" />
     <xs:attribute name="plasmon" type="PlasmonMaterials" use="optional" />
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
```

```
</xs:element>

<xs:element name="variables">
  <xs:annotation>
    <xs:documentation>
      Список переменных, используемых в программе
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="var">
        <xs:complexType>
          <xs:attribute name="name" type="Variables" use="required" />
          <xs:attribute name="value" type="xs:unsignedByte" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Список опций и параметров -->
<xs:element name="options">
  <xs:annotation>
    <xs:documentation>
      Опция и параметры работы программы
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="param">
        <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required" />
          <xs:attribute name="value" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Список действий -->
<xs:element name="actions">
  <xs:annotation>
    <xs:documentation>
      Описание списка действий (сценария работы программы)
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="action">
        <xs:complexType>
          <xs:attribute name="code" type="xs:unsignedByte" use="required" />
          <xs:attribute name="value" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Выходные параметры -->
<xs:element name="output">
  <xs:annotation>
    <xs:documentation>
      Описание параметров вывода результатов
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="path">
        <xs:complexType>
          <xs:attribute name="value" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element maxOccurs="unbounded" name="datafile">
        <xs:complexType>
          <xs:attribute name="type" type="DatafileTypes" use="required" />
          <xs:attribute name="name" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
```

```
      <xs:element name="fileprefix">
        <xs:complexType>
          <xs:attribute name="value" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="plasmons">
  <xs:annotation>
    <xs:documentation>
      Секция для описания изветных параметров плазмонов (справочные данные)
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="plasmon">
        <xs:complexType>
          <xs:attribute name="material" type="PlasmonMaterials" use="required" />
          <xs:attribute name="plasmow" type="xs:double" use="required" />
          <xs:attribute name="relax" type="xs:double" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="probes">
  <xs:annotation>
    <xs:documentation>
      Описание списка пробных точек
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="domain">
        <xs:complexType>
          <xs:attribute name="number" type="xs:unsignedByte" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="probe" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="x" type="xs:double" use="required"/>
          <xs:attribute name="y" type="xs:double" use="required"/>
          <xs:attribute name="z" type="xs:double" use="required"/>
          <xs:attribute name="dir" type="FieldDirections" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-- Конец описания элементов -->

<!-- Описание перечислений -->

<!-- 1. Названия типов задач -->
<xs:simpleType name="TaskNames">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Common"/>
    <xs:enumeration value="Netgen"/>
  </xs:restriction>
</xs:simpleType>
<!-- 2. Названия переменных -->
<xs:simpleType name="Variables">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Time"/>
    <xs:enumeration value="TimeDelta"/>
    <xs:enumeration value="FrequencyDelta"/>
    <xs:enumeration value="FrequencyStep"/>
  </xs:restriction>
</xs:simpleType>
<!-- 3. Описания типов файлов -->
<xs:simpleType name="DatafileTypes">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Mesh"/>
    <xs:enumeration value="Result"/>
```

```
      <xs:enumeration value="Matrix"/>
      <xs:enumeration value="Particles"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- 4. Описания типов материалов для плазмонов -->
  <xs:simpleType name="PlasmonMaterials">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Ag"/>
      <xs:enumeration value="Au"/>
      <xs:enumeration value="Cu"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- 5. Описание направлений исследуемого поля -->
  <xs:simpleType name="FieldDirections">
    <xs:restriction base="xs:string">
      <xs:enumeration value="X"/>
      <xs:enumeration value="Y"/>
      <xs:enumeration value="Z"/>
      <xs:enumeration value="All"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- Конец описания перечислений -->
</xs:schema>
```

ПРИЛОЖЕНИЕ К
(обязательное)
Руководство системного программиста

1. Общие сведения о программе.

Программа представляет собой консольное приложение, предназначенное для выполнения расчета электромагнитных полей (ЭМП) с помощью векторного метода конечных элементов. Имеются следующие возможности:

– автоматизированную генерацию конечноэлементных сеток на основе использования программы Netgen;
– выполнение обработки файлов сеток для использования сетки в программе;
– расчет различных типов задач по распределению ЭМП;
– преобразование текстовых файлы сеток в структурированные XML файлы.

Для нормального функционирования программы с приемлемыми эксплуатационными характеристиками необходимо оборудование, минимальная конфигурация которого выглядит следующим образом:

– процессор класса Intel Pentium III или AMD Athlon XP, Intel Core 2, AMD Athlon X2 с частотой от 1 ГГц;
– ОЗУ 1 Гб и выше;
– жесткий диск (HDD) от 40 Гб;
– видеоадаптер GeForce 6xxx или Radeon HD2xxx с объемом видеопамяти 64 Мб,
– монитор с разрешением 1027x768 и выше.

Необходимо также следующее программное и системное обеспечение:

– ОС MS Windows XP/Vista/7/8;
– MS Visual Studio 2010;
– программа для работы с XML документами.

2. Структура программы.

Программа состоит из нескольких компонентов:

– приложения для выполнения конечноэлементного расчета электромагнитного поля;
– входных файлов, описывающих параметры задачи и сценарии ее выполнения (Input API);
– программы Netgen, предназначенной для генерации конечноэлементной сетки.

Все компоненты системы обеспечивают работу системы в виде целостного комплекса. Обязательное наличие программы Netgen связано с использованием трехмерных сеток для рассматриваемых геометрических объектов, которые она эффективно разбивает на тетраэдральные элементы.

Для реализации работы со сторонней программой, имеются специальные библиотеки для взаимодействия с ней, подготовки данных и анализа результатов (в виде парсера для файлов сетки).

3. Настройка программы

Для работы программы необходимо:

– MS Windows XP и выше;
– .NET Framework 4;
– Netgen 4.9;
– MSXML 4 и выше.

Для настройки программы необходимо провести проверку работоспособности каждого из компонентов, например, путем выполнения тестовой программы для

платформы .NET. По умолчанию, все компоненты не нуждаются в дополнительной настройке.

Для настройки модуля для работы с программой Netgen необходимо внести изменения в файл параметров окружения Params.xml. В нем необходимо указать полные пути к каталогам, где расположена программа Netgen на рабочей станции и место вывода результатов работы программы. В случае, если исполняемый файл программы Netgen отличается от netgen.exe, то следует указать параметр AppName со значением используемого названия файла.

4. Проверка программы.

Для отладки программы нужно произвести проверку каждого из компонентов системы необходимо выполнить любое тестовое приложение, использующее платформу .NET. Для проверки работы программы Netgen нужно запустить ее и загрузить тестовую геометрию, для которой выполнить построение сетки и экспорт ее в формат Neutral Format.

Если необходимо выполнить проверку модулей программы, связанных с взаимодействием с Netgen, а также парсером для файлов сеток, то нужно воспользоваться исходными кодами программы и выполнить тестовые задачи, которые имеются в коде (Samples). Для этого нужно изменить тип проекта на консольную программу. После проверки можно внести изменения и вернуть проект к библиотеке классов.

5. Сообщения системному программисту.

Для правильной эксплуатации необходимо ознакомиться со следующими документами:

- Input API Specification;
- описание модуля NetgenUtils;
- описание модуля MeshParser.

При внесении изменений в код программы, а также в Input API и формат, описываемых им файлов, необходимо задокументировать внесенные изменения и сообщить разработчику.

ПРИЛОЖЕНИЕ Л
(обязательное)
Руководство программиста

1. Назначение и условия применения программы.

Программа представляет собой консольное приложение, предназначенное для выполнения расчета электромагнитных полей (ЭМП) с помощью векторного метода конечных элементов. Имеются следующие возможности:

– автоматизированную генерацию конечноэлементных сеток на основе использования программы Netgen;
– выполнение обработки файлов сеток для использования сетки в программе;
– расчет различных типов задач по распределению ЭМП;
– преобразование текстовых файлов сеток в структурированные XML файлы.

Программа может применяться как средство для компьютерного моделирования ЭМП, в качестве каркаса для решения различных задач электродинамики.

Для полноценного функционирования программы с наилучшими эксплуатационными характеристиками необходимо оборудование, рекомендуемая конфигурация которого выглядит следующим образом:

– процессор класса Intel Core i5, Core i7, AMD FX, AMD Phenom 2, AMD A6, AMD A8 с частотой от 2 ГГц на ядро, с наличием от 2 ядер;
– ОЗУ 4 Гб и выше (от DDR3-1066);
– жесткий диск (HDD) от 250 Гб со скоростью вращения 7200 об /мин и интерфейсом SATA;
– видеоадаптер Nvidia GeForce 8xxx или AMD Radeon HD4xxx с объемом видеопамяти 256 Мб;
– монитор с разрешением 1280x1024 и выше (для разработки предпочтительнее от 1680x1050).

Среди периферийных устройств можно отметить принтер, для возможности печати файлов с полученным набором результатов, входные данные и виды анализируемой геометрии.

Необходимо также следующее программное и системное обеспечение:

– ОС MS Windows XP/7/8;
– MS Office Word 2010 и выше.

Для разработки и изменения программы используют:

– MS Visual Studio 2010 Premium;
– NuGet Package Manager.

2. Характеристики программы.

Все компоненты системы обеспечивают ее работу системы в виде целостного комплекса. Обязательное наличие программы Netgen связано с использованием трехмерных сеток для рассматриваемых геометрических объектов, которые программа эффективно разбивает на тетраэдральные элементы.

Для реализации работы со сторонней программой, имеются специальные библиотеки для взаимодействия с ней, подготовки данных и анализа результатов (в виде парсера для файлов сетки).

Программа работает в автоматическом режиме в соответствии со сценарием задачи, которая задается через Input API, который предусматривает стандартизированный формат ввода данных и их распознавание приложением. Это обеспечивается благодаря применению XML файлов, а также соответствующего программного кода, который построен на использовании стандартизированных параметров и действий.

При возникновении ошибки или исключения, работа программы будет прекращена. Сведения об исключении будут выведены на экран.

3. Обращение к программе.

Для вызова программы необходимо разместить ее в любом каталоге (кроме каталогов, имеющих в наименовании русские названия(!)). После чего сформировать входной файл в формате XML в соответствии с Input API Specification. В файле описывается задача, ее параметры, а также сценарий выполнения, основанный на списке действий.

Запуск программы осуществляется через командную строку. Для этого в параметрах к ней передается имя сформированного входного файла с постановкой и сценарием задачи.

4. Входные и выходные данные.

Входной информацией в программе считаются файлы форматов XML, описываемые в Input API, которые представляют собой условия задачи и сценарии ее выполнения. Для входа необходимо два файла:

- описание самой задачи, ее типа, условий и сценария ее решения;
- описание геометрии задачи (может быть в виде *.geo файла или же в виде XML файла). В случае использования XML файла, необходимо указать соответствующий параметр, чтобы программа сгенерировала нужный файл геометрии, используемый в Netgen.

Выходной информацией являются текстовые файлы с результатами решения СЛАУ, а также вспомогательные файлы с данными об используемой геометрии и построенной сетке. Все файлы записываются и сохраняются в кодировке UTF-8.

5. Сообщения.

В процессе работы программы выводятся как информационные (Info), так и диагностические (Debug) сообщения. Их вывод производится через библиотеку логирования (Nlog), которая дублирует вывод в консоль и в лог файлы. При возникновении ошибки или изменении процесса выполнения программы все данные содержатся в папке logs программы.

ПРИЛОЖЕНИЕ М
(обязательное)
Руководство пользователя

1. Введение.

Программный продукт для компьютерного моделирования электромагнитных полей в материалах предназначен для анализа и синтеза новых материалов с требуемыми свойствами. Программа представляет собой каркас для реализации решения различного класса задач с использованием векторного метода конечных элементов (ВКМЭ). С помощью Input API можно гибко задавать условия задач, сценарий их выполнения, а также четко описывать постановку задачи. Имеются следующие возможности:

– автоматизированная генерация конечноэлементных сеток с использованием программы Netgen;
– выполнение обработки файлов сеток для использования сетки в программе;
– расчет различных типов задач по распределению ЭМП;
– преобразование текстовых файлов сеток в структурированные XML файлы.

Для работы с программой необходимы базовые знания формата XML и умение создавать такие файлы с заданной структурой. Для ознакомления со структурой входного XML файла необходимо ознакомиться с Input API Specification. Для построения геометрии задачи необходимо ознакомиться с описаниями модулей MeshParser и NetgenUtils.

Уровень подготовки пользователя необходим достаточно высокий, он должен уметь работать в среде разработки, разбираться в постановке задач электродинамики и методами их численного решениями, уметь сформулировать задачи и приводить их к нужной форме.

2. Назначение и условия применения программы.

Основные функции программы EMFFM:

– генерация конечноэлементной сетки на основе заданной геометрии;
– решение задачи о распределении электромагнитного поля с использованием векторного метода конечных элементов;
– анализ полученных в процессе решения результатов.

Для работы программы необходимо:

– MS Windows XP и выше;
– .NET Framework 4;
– Netgen 4.9;
– MSXML 4 и выше.

3. Подготовка к работе.

Для подготовки к работе программа должна быть размещена на компьютере в некотором отдельном каталоге, предназначенном целиком и полностью для нее, для хранения настроек и результатов. Программа состоит из нескольких компонентов.

Для настройки одного из компонентов – модуля для работы с программой Netgen, необходимо внести изменения в файле параметров окружения Params.xml. В нем необходимо указать полные пути к каталогам, где расположена программа Netgen на рабочей станции и место вывода результатов работы программы. В случае, если исполняемый файл программы Netgen отличается от netgen.exe, то следует указать параметр AppName со значением используемого названия файла.

Подготовка задачи включает в себя:

– описание геометрии объектов, используемых для исследования;
– описание параметров каждого объекта (электродинамические, материальные);

– преобразование источников из непрерывной формы в дискретную;
– установка граничных условий;
– описание справочных данных, необходимых для решения.

Кроме того, при создании входного файла к задаче нужно определить ее тип, параметры решения (вывод результатов, вид решения и т.д.), указать параметры выходных данных (каталог, названия файлов результатов), переменные, необходимые при решении. Формирование сценария решения основано на задании доступных действий (операций), которые выполнит программа после запуска.

4. Описание операций.

Программа разработана в виде консольного приложения. Для его запуска необходимо в командной строке выполнить команду вида:

dir> app.exe input.xml

где app.exe – название исполняемого файла программы, input.xml – входной файл с данными задачи.

Запустившись, начнется выполнение программы:

– чтение входного файла и его разбор;
– обработка списка действий;
– выполнение действий по порядку.

Работа сопровождается выводом сообщений на экран, а также их дублирование в файлы логов. Действия и соответствующие результаты работы программы будут соответствовать данным, описанными во входном файле.

При выполнении будет происходить запуск программы Netgen, сопровождаемый открытием консольного окна приложения и вывода сведении по процессу генерации сетки. Программа завершится автоматически по окончании выполнения построения сетки и вывода результатов в файл.

5. Аварийные ситуации.

При работе с программой могут возникать аварийные ситуации. При их появлении будет исключение, которое перехватывается и выводится сведения об ошибке. Работа программы при этом будет прекращена.

При частых повторах аварийных ситуаций необходимо обратиться к разработчику.

6. Рекомендации по освоению.

Для лучшего понимания процесса работы необходимо ознакомиться:

– с самой программой;
– с примерами, которые идут в комплекте с программой;
– со вспомогательными средствами, используемыми при работе;
– с документацией на модули системы.

ПРИЛОЖЕНИЕ Н
(обязательное)
Тестовая задача

```
<?xml version="1.0" encoding="utf-8"?>
<geometrydata>
 <points>
  <point name="p1" x="0" y="0" z="0"/>
  <point name="p2" x="490" y="490" z="490"/>
  <point name="p3" x="245" y="245" z="245"/>
  <point name="p4" x="210" y="210" z="210"/>
  <point name="p5" x="280" y="280" z="280"/>
 </points>
 <vectors>
  <vector name="v1" direction="Top"/>
  <vector name="v2" direction="Bottom"/>
  <vector name="v3" direction="Left"/>
  <vector name="v4" direction="Right"/>
  <vector name="v5" direction="Back"/>
  <vector name="v6" direction="Forward"/>
 </vectors>
 <solids>
  <plane name="plane1" p="p1" v="v3" />
  <plane name="plane2" p="p1" v="v6" boundary="1"/>
  <plane name="plane3" p="p1" v="v2" />
  <plane name="plane4" p="p2" v="v1" />
  <plane name="plane5" p="p2" v="v4" />
  <plane name="plane6" p="p2" v="v5" />
  <sphere name="sp" p="p3" radius="20" boundary="3"/>
  <complex name="cube1" data="plane1 and plane2 and plane3 and plane4 and plane5 and plane6" boundary="2"/>
  <orthobrick name="cube2" p1="p4" p2="p5" boundary="4"/>
  <complex name="small" data="cube2 and not sp"/>
  <complex name="big" data="cube1 and not cube2"/>
 </solids>
 <colors>
  <color name="col1" r="0" g="0" b="1"/>
  <color name="col2" r="0" g="1" b="0"/>
  <color name="col3" r="1" g="0" b="0"/>
 </colors>
 <output>
  <tlo name="big" transparent="true" color="col1"/>
  <tlo name="small" transparent="true" color="col2"/>
  <tlo name="sp" transparent="false" color="col3"/>
 </output>
</geometrydata>
```

Рисунок Н.1 – Файл с описанием геометрии задачи

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- описание входного файла для программы -->
<inputdata>
 <!-- описание класса задачи -->
 <task name="Netgen"/>

 <!-- описание геометрии задачи (для работы с Netgen) -->
 <geometry>
  <geofile name="geom3.xml"/>
  <meshfile name="mesh.neu"/>
  <isgenerategeomfile value="true"/>
  <domains>
   <domain code="1" name="Cube" material="Air"/>
   <domain code="2" name="Cube" material="Air"/>
   <domain code="3" name="Sphere" material="Dielectric" plasmon="Ag"/>
  </domains>
  <cofactor value="1E-9"/>
 </geometry>

 <!-- описание источников -->
 <sources>
  <source freq="4.654" lambda="405E-9" amplitude="50" phase="0" dir="Z" boundary="1">
   <point x="245" y="245" z="-40"/>
  </source>
 </sources>
```

```
<!-- описание граничных условий -->
<boundary>
  <condition type="Dirichlet" boundary="3" value="0"/>
</boundary>

<!-- описание списка материалов -->
<materials>
  <material name="Air" eps="1" mu="1" default="true"/>
  <material name="Dielectric" eps="10" mu="1"/>
</materials>

<!-- описание материалов для плазмонов (их известных параметров) (справочный блок) -->
<plasmons>
  <plasmon material="Ag" plasmow="9.1E-19" relax="29E-15"/>
  <plasmon material="Au" plasmow="9.1" relax="40"/>
  <plasmon material="Cu" plasmow="8.8" relax="40"/>
</plasmons>

<!-- описание переменных и констант -->
<variables>
  <var name="Time" value="1"/>
  <var name="TimeDelta" value="0.1"/>
  <var name="FrequencyDelta" value="1E6"/>
  <var name="FrequencyStep" value="5E5"/>
</variables>

<!-- описание опций и параметров решения -->
<options>
  <param name="IsOutputToFile" value="true"/>
  <param name="IsBuiltEdges" value="true"/>
  <param name="WithParticles" value="true"/>
  <param name="NetgenParams" value="input\Params.xml"/>
  <param name="InTimeDomain" value="true"/>
  <param name="InFrequencyDomain" value="false"/>
  <param name="WithoutIterations" value="false"/>
  <param name="WithEdgeFields" value="false"/>
  <param name="IsProbeOutput" value="true"/>
</options>

<!-- описание действий -->
<actions>
  <action code="1" value="Solve"/>
</actions>

<!-- описание выходных файлов -->
<output>
  <!-- путь для файлов выхода -->
  <path value="output\\"/>
  <datafile type="Mesh" name="mesh.txt"/>
  <datafile type="Particles" name="particles.txt"/>
  <datafile type="Matrix" name="matrix.txt"/>
  <datafile type="Result" name="result.txt"/>
  <fileprefix value="field-"/>
</output>

<!-- список пробных точек для исследования -->
<probes>
  <domain number="2"/>
  <probe x="2.5" y="2.3" z="1.2" dir="All"/>
</probes>
</inputdata>
```

Рисунок Н.2 – Входной файл с описанием задачи