МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»
Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Компьютерные системы конечноэлементных расчётов»

на тему: «ОПТИМИЗАЦИЯ РАЗМЕРОВ ДЕТАЛИ РОБОТА-МАНИПУЛЯТОРА»

Исполнитель: студент гр. ИТП-31
Расшивалов Н.И.

Руководитель: доцент
Комраков В.В.

Дата проверки:_____
Дата допуска к защите:_____
Дата защиты:_____
Оценка работы:_____

Подписи членов комиссии
по защите курсового проекта: _____

Гомель 2022

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ

МКР – Метод конечных разностей,
МКЭ – Метод конечных элементов,
ЭВМ – Электронно-вычислительная машина,
ПК – Персональный компьютер,
САПР – Система автоматизированного проектирования работ.
СЛАУ – Система линейных алгебраических уравнений,
ANSYS – универсальная программная система конечно-элементного анализа.
SOLIDWORKS – программный комплекс САПР для автоматизации работ промышленного предприятия на этапах конструкторской и технологической подготовки производства.
C# – объектно-ориентированный язык программирования.
C++ – компилируемый, статически типизированный язык программирования общего назначения.
Java – строго типизированный объектно-ориентированный язык программирования общего назначения.
C – компилируемый статически типизированный язык программирования общего назначения.
ECMA – это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков.
Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью
IronPython – одна из основных реализаций языка *Python*, предназначенная для платформы *Microsoft .NET* или *Mono*.
Microsoft – одна из крупнейших транснациональных компаний по производству проприетарного программного обеспечения для различного рода вычислительной техники – персональных компьютеров, игровых приставок, КПК, мобильных телефонов и прочего.
.NET – это модульная платформа для разработки программного обеспечения с открытым исходным кодом.
WPF – система для построения клиентских приложений *Windows* с визуально привлекательными возможностями взаимодействия с пользователем.
JSON – текстовый формат обмена данными, основанный на *JavaScript*.

ВВЕДЕНИЕ

На практике, когда необходимо производить какие-либо инженерные расчеты, связанные с анализом прочности различных конструкций, используют численные методы, поскольку применение аналитических методов требует высокого уровня математического аппарата. Кроме того, аналитические расчёты позволяют получить решение задач для простых тел и для простой схемы нагруженности. Применение же численных методов, к которым относятся методы конечных разностей, конечных элементов, граничных элементов, не ограничено ни способами приложения нагрузок, ни сложностью геометрии тела.

Пакеты для математического моделирования являются гибкими и надёжными средствами проектирования и анализа. Они работают в различных операционных системах, на различных устройствах – от персональных компьютеров до рабочих станций и суперкомпьютеров, однако обладают серьёзными недостатками. Они являются дорогостоящими, разносторонними и сложными продуктами. Для внедрения их на узкоспециализированном производстве необходимы большие средства.

Программа *ANSYS* – это гибкое, надежное средство проектирования и анализа. Она работает в среде операционных систем самых распространенных компьютеров – от ПК до рабочих станций и суперкомпьютеров, однако она обладает серьезным недостатком. Это дорогостоящий и многогранный, сложный продукт. Для внедрения его на узкоспециализированном малом производстве необходимы немалые средства.

Целью курсовой работы является анализ и моделирование напряженно деформированной детали робота-манипулятора, написание программного приложения на языке высокого уровня, выполняющего аналогичный расчёт конструкции методом конечных элементов.

Актуальность темы заключается в том, что расчёты подобного рода востребованы при реализации разнообразных конструкций.

1 ОБЗОР ЧИСЛЕННЫХ МЕТОДОВ МОДЕЛИРОВАНИЯ В МЕХАНИКЕ

1.1 Численные методы решения краевых задач

1.1.1 Задачи о нахождении решений дифференциальных уравнений, удовлетворяющих граничным условиям в концах интервала или на границе области, называются краевыми задачами. Эти задачи решаются с помощью математических методов. На сегодняшний день существует две основных группы таких методов: аналитические и численные.

При использовании аналитических методов решение задачи удаётся выразить с помощью формул. Например, если задача состоит в решении простейших алгебраических, тригонометрических, дифференциальных и т.д. уравнений, то использование известных из курса математики приёмов сразу приводит к цели. Преимущество аналитических методов: в результате применения аналитических методов за небольшой отрезок сразу получается точный ответ.

Недостаток аналитических методов: аналитические методы применимы лишь к небольшому числу, как правило, не очень сложных по своей структуре задач. Так, например, до сих пор не удалось решить в общем виде уравнение пятой степени.

Основным инструментом для решения сложных математических моделей и задач в настоящее время являются численные методы. Они сводят решение задачи к выполнению конечного числа арифметических действий над числами и дают результат в виде числового значения с погрешностью, приемлемой для данной задачи.

Численные методы разработаны давно. Однако при вычислениях вручную они могли использоваться лишь для решения не слишком трудоёмких задач. С появлением компьютеров, которые за короткое время могут выполнить миллиарды операций, начался период бурного развития численных методов и внедрения их в практику [1, с. 200].

Основными численными методами, используемыми в решении краевых задач, являются метод конечных разностей и метод конечных элементов. Они имеют как свои преимущества, так и недостатки.

1.1.2 Метод конечных разностей (МКР) – численный метод решения дифференциальных уравнений, основанный на замене производных разностными схемами. Является сеточным методом.

Идея метода конечных разностей известна давно, с соответствующих трудов Эйлера. Однако практическое применение этого метода тогда было весьма

ограничено из-за огромного объёма ручных вычислений, связанных с размерностью получаемых систем алгебраических уравнений, на решение которых требовались годы. В настоящее время, с появлением быстродействующих компьютеров, ситуация в корне изменилась. Этот метод стал удобен для практического использования и является одним из наиболее эффективных при решении различных задач математической физики [2, с. 524].

Главной проблемой метода является построение правильной разностной схемы, которая будет сходиться к решению. Построение схемы выполняется исходя из свойств исходного дифференциального оператора [3, с.82].

Даже при правильной схеме решение краевой задачи может быть получено с большой погрешностью, поэтому чаще всего для решения краевых задач используют другой численный метод – метод конечных элементов.

1.1.3 Метод конечных элементов (МКЭ) – основной метод современной вычислительной механики, лежащий в основе подавляющего большинства современных программных комплексов, предназначенных для выполнения расчётов инженерных конструкций на ЭВМ. МКЭ используется для решения разнообразных задач, как в области прочностных расчётов, так и во многих других сферах: гидродинамике, электромагнетизме, теплопроводности и др.

Метод конечных элементов позволяет практически полностью автоматизировать расчёт механических систем, хотя, как правило, требует выполнения значительно большего числа вычислительных операций по сравнению с классическими методами механики деформируемого твёрдого тела. Современный уровень развития вычислительной техники открывает широкие возможности для внедрения МКЭ в инженерную практику [4, с.53].

Алгоритм решения задачи прочностного расчёта методом конечных элементов похоже на МКР, однако имеет ряд ключевых отличий.

Область, занимаемая телом, разбивается на конечные элементы. Чаще всего это треугольники в плоском случае и тетраэдры в пространственном. Внутри каждого элемента задаются некоторые функции формы, позволяющие определить перемещения внутри элемента по перемещениям в узлах, т. е. в местах стыков конечных элементов. За координатные функции принимаются функции, тождественно равные нулю всюду, кроме одного конечного элемента, внутри которого они совпадают с функциями формы. В качестве неизвестных коэффициентов метода конечных элементов берутся узловые перемещения. После минимизации функционала энергии, получается система линейных алгебраических уравнений. Таким образом, ситуация здесь такая же, как и в вариационных разностных методах, в которых для получения разностной системы уравнений применяется один из вариационных принципов [5, с.225].

В данный момент метод конечных элементов – основной метод решения

задач механики, это произошло из-за ряда преимуществ относительно остальных методов:

– геометрия тела может быть абсолютно любой, так как тело разбивается на множество конечных элементов, взаимодействующих между собой. Это позволяет упростить задачу, вычисляя только взаимодействия соседних элементов, имеющих простую геометрию.

– Параметры, такие как нагрузки или закрепления можно задать в любом узле конструкции;

– так как итоговая система алгебраических уравнений учитывает взаимодействие между всеми конечными элементами, при этом применяя вариационные принципы, точность и гибкость вычислений весьма высоки.

Однако в методе конечных элементов также можно выделить ряд недостатков:

– время, необходимое для расчётов, а также требования к аппаратным средствам компьютера и объёму носителей информации в несколько раз превышают аналогичные требования для других численных методов. Для решения задач этим методом требуется высокопроизводительная ЭВМ. Однако иногда задачи теории упругости сводятся к плоским, тогда нагрузка на систему значительно уменьшается;

– сложность метода конечных элементов требует от человека, который его применяет, глубоких знаний для того, чтобы верно вычислить необходимые для анализа конструкций параметры [6, с.12].

В связи с тем, что МКЭ является самым универсальным методом решения в области прочностных расчётов существует огромное разнообразие математических пакетов, которые его используют.

1.1.4 МКР обладает следующими преимуществами:

– МКР позволяет рассчитывать геометрические конфигурации, к которым не-применимы простые расчетные методы на основе аналитических решений;

– МКР позволяет получить расчетные значения во всех точках сетки, а не только интегральные значения по поперечному сечению канала;

– точность получаемых результатов повышается за счет того, что в систему дифференциальных уравнений можно включить практически любой закон течения материала;

– возможность учета взаимосвязи дифференциальных уравнений (например, уравнений движения и энергии, зависимостей температуры и скорости сдвига от вязкости);

– так как система уравнений решается одновременно во всей расчетной области, результаты отражают все эффекты взаимодействия (это утверждение не является справедливым для явных разностных схем).

Однако наряду с перечисленными преимуществами МКР обладает рядом внутренне присущих недостатков:

– определение геометрии или рабочей точки возможно только методом итераций, поскольку уравнения необратимы;

– расчеты невозможно выполнить вручную или на карманном калькуляторе; для их осуществления необходим, как минимум, персональный компьютер;

– для выполнения расчетов требуется существенно большее время по сравнению с простыми аналитическими методами.

МКЭ предоставляет следующие преимущества по сравнению с МКР:

– рассматриваемая геометрия может быть любой, поскольку она определяется независимо от компьютерной программы. Это означает, что программы, реализующие МКЭ, работают независимо от геометрии;

– возможность определения расчетных параметров в любой точке рассматриваемой области;

– поскольку уравнения МКЭ решаются одновременно, существует возможность учесть все взаимодействия, имеющие место в системе, с высокой степенью гиб-кости и точности.

Тем не менее МКЭ тоже не свободен от недостатков:

– время, необходимое для расчетов, а также требования к аппаратным средствам компьютера и объему носителей информации в несколько раз превышают аналогичные требования для МКР. Для решения задач этим методом требуется как минимум высокопроизводительный 16- или 32-разрядный ПК. За редким исключением, применение программ, реализующих МКЭ, ограничивается плоскими задачами;

– поскольку геометрия канала, а также начальные и граничные условия задаются пользователем самостоятельно, время, необходимое для расчета, существенно больше, чем для МКР, где эти параметры более или менее фиксированы;

– большая гибкость МКЭ, касающаяся выбора геометрии, плотности сетки, вы-бора типов элементов и граничных условий требует от пользователя более глубокого понимания сущности данного метода, иначе получение надежных результатов становится проблематичным.

В связи с изложенными преимуществами и недостатками для решения поставленной задачи лучше использовать МКЭ.

1.2 Программные средства конечно-элементных расчётов

Главными факторами, почему расчёты методом конечных элементов стали настолько неотъемлемой частью любого инженерного проекта стали увеличение производительности компьютеров и количества математических пакетов, способных эффективно решать поставленные задачи.

Кроме того, большое количество документации снизило порог вхождения пользователя в инженерные расчёты, что позволило совершать сложные операции с конструкциями в математических пакетах менее квалифицированным специалистом и экономить на этом.

У каждого математического пакета есть свои сильные и слабые стороны при решении конкретной инженерной задачи. Выбор программы расчёта зависит от подготовленности пользователя в своей научной области, типа решаемой задачи, типа доступной ЭВМ, размерности задачи и других факторов.

К критериям, помогающим сделать выбор в сторону той или иной программы, относятся следующие факторы: программа широко используется; в программе используются новейшие научные достижения; программа коммерчески вполне доступна; имеется подробная и понятная документация.

Из большого разнообразия таких программ следует выделить *ANSYS* и *SOLIDWORKS*.

ANSYS является одной из самых распространённых программ для инженерных расчётов, использующей метод конечных элементов. Многоцелевая направленность программы, независимость от аппаратных средств, средства удобного геометрического моделирования, совместимость с другими популярными математическими пакетами и приятный графический интерфейс делают её одним из самых популярных средств анализа геометрических систем.
Самое важное что можно выделить, в этой программе реализовали вычисления на персональных компьютерах, что привело к большой популярности этого пакета среди многих пользователей.

Однако для использования этой программы необходимы обширные инженерные знания, так как там существует огромное количество функций, настроек и параметров, играющих ключевую роль в правильности расчёта.

SOLIDWORKS же является менее сложной в своём использовании. Эта программа имеет улучшенную версию создания геометрии различных тел по сравнению с *ANSYS* [7].

Также большинство параметров расчётов изначально имеют довольно хорошие значения, что позволяет специалисту не тратить время на их детальную настройку.

Однако минусом данных программных комплексов является их стоимость. Из-за высокой стоимости не каждое предприятие может позволить себе такой программный комплекс, особенно, если на предприятии решается узкий круг задач. Для таких задач покупка подобных программ не является рациональной. Поэтому разработка приложений, решающий какой-то конкретный вид задач и, следовательно, имеющих гораздо меньшую стоимость, является актуальной задачей.

1.3 Сравнение языков программирования для решения задачи анализа напряженно-деформированной детали робота-манипулятора

1.3.1 *C#*, также известный как *C-Sharp*, представляет собой типобезопасный язык программирования общего назначения, который следует конструкциям *C* и *C ++*. Он следует нескольким парадигмам программирования, включая объектно-ориентированное, структурированное, императивное, управляемое задачами, функциональное, управляемое событиями, параллельное, рефлексивное и общее. Андерс Хельсберг из *Microsoft* разработал C# в 2001 году [8, с.120]. Позже Европейская ассоциация производителей компьютеров (*ECMA*) утвердила его в качестве международного стандарта в 2002 году. Более того, Международная организация по стандартизации (*ISO*) одобрила его в 2003 году. *C#* синтаксически аналогичен *Java*, и его легко освоить тем, кто хорошо знает *C* и *C ++*. Как и *Java*, *C#* также является нейтральным или независимым от платформы языком, код которого может быть скомпилирован и запущен во всех операционных системах. Он обычно используется с платформой *Microsoft.NET* для *Windows*.

Преимущества *C#*:

– *C#* – простой, надежный и масштабируемый язык программирования;

– динамически типизированный характер *C#* облегчает разработчикам поиск ошибок в коде. *C#* устраняет проблему утечки памяти [9, с.512];

– он имеет знакомый синтаксис, идентичный языкам *C* и *C ++*.

Недостатки *C#*:

– У *C#* крутая кривая обучения, поэтому он не идеален для начинающих. Те, кто имеет базовые знания *C, C ++* или *Java*, могут легко изучить *C#.*

– У него плохая кроссплатформенная поддержка;

– *C#* не такой гибкий, как другие языки программирования, так как он зависит от платформы *.NET.*

1.3.2 *Python* – это универсальный интерпретируемый язык высокого уровня. Стиль *Python* – значительные отступы, которые подчеркивают удобочитаемость кода. Он следует нескольким принципам программирования, таким как объектно-ориентированное, функциональное, структурированное, рефлексивное и процедурное. *Python* включает обширную стандартную библиотеку, поэтому его часто называют языком с «включенными батареями». Кроме того, *Python* легко изучить и понять, поскольку в его синтаксисе используются простые английские ключевые слова и не используются фигурные скобки для разделения блоков. Еще одно преимущество *Python* заключается в том, что он позволяет разработчикам писать код в несколько строк по сравнению с другими языками программирования.

Преимущества *Python*:

– *Python* – это язык с динамической типизацией. Это означает, что нет необходимости определять тип данных переменной, поскольку она автоматически присваивает типы данных переменным во время выполнения.

– *Python* легко читать и изучать благодаря синтаксису, похожему на английский. Кроме того, исключается использование точки с запятой после конца оператора и разделителей для начала и конца блока.

– Поскольку *Python* является интерпретируемым языком, он выполняет код построчно, останавливает выполнение в случае ошибки и сообщает об этом;

– он бесплатный и с открытым исходным кодом, что дает возможность загружать и изменять его исходный код;

– стандартная библиотека *Python* представляет собой полный набор модулей;

– *Python* совместим и переносится в системы *Windows*, *macOS* и *Unix* / *Linux*;

Недостатки Python:

– *Python* имеет низкую скорость, потому что это интерпретируемый язык, и он выполняет код построчно;

– это не идеальный выбор для задач с интенсивным использованием памяти, поскольку он потребляет большой объем памяти из-за гибкости типов данных;

– Поскольку *Python* неэффективен с точки зрения памяти и имеет медленную вычислительную мощность, он не используется при разработке клиентских или мобильных приложений.

1.1.3 Сравнивая *Python* и C#, можно сделать вывод о том, что они являются объектно-ориентированными языками общего назначения. *Python* будет отличным вариантом, если проект связан с исследованием данных, поскольку он имеет обширную стандартную библиотеку. Выбор *C#* будет полезен для разработки адаптивных веб-сайтов, веб-сервисов и настольных приложений. Организованная структура *C#* гарантирует отсутствие несоответствий в синтаксисе и правилах форматирования. С другой стороны, можно писать код *Python* быстрее, поскольку он требует меньше строк кода, чем *C#.* Однако *C#* может делать все, что умеет *Python*, и обеспечивает лучшую производительность. Можно использовать языки *Python* и *C#* с *IronPython*, которая является реализацией *Python* с открытым исходным кодом и интегрирована с платформой *.NET*.

Таким образом, для решения задачи анализа напряженно деформированной детали робота-манипулятора наиболее подходящим является язык программирования *C#.*

2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ

2.1 Постановка задачи

Задачей является разработка приложения на языке высокого уровня, которое производит анализ напряженно-деформированного тела: детали робота-манипулятора. Также необходимо в качестве верификации провести в пакете для конечно-элементных расчётов аналогичный анализ.

Функционал приложения заключается в следующем:

– построение регулярной конечно-элементной сетки для детали робота-манипулятора;

– наложение сил и закреплений к узлам созданной сетки;

– выбор материала детали робота-манипулятора;

– построение математической модели детали робота-манипулятора, представляющей собой матрицу жёсткости системы и вектор узловых нагрузок системы;

– решение полученной СЛАУ (системы линейных алгебраических уравнений), в результате которой вычисляется вектор узловых перемещений системы;

– вычисление деформаций и напряжений системы;

– графическое отображение результатов вычислений.

На рисунке 2.1 изображен чертеж детали робота-манипулятора (вид сверху).

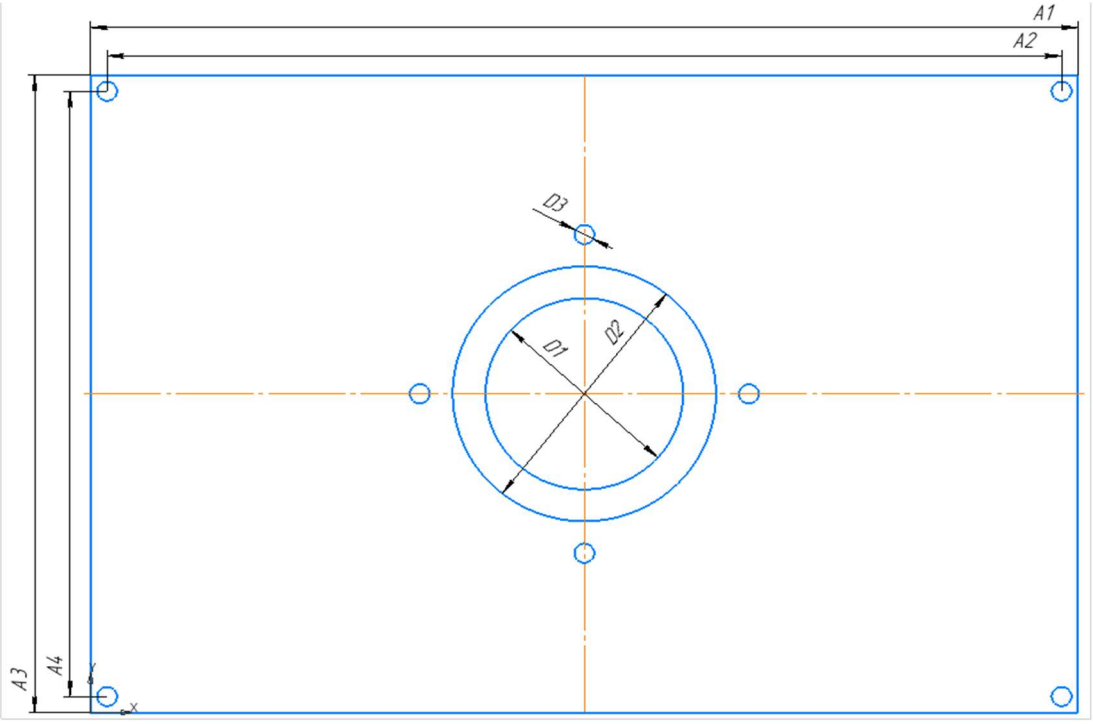

Рисунок 2.1 – Чертеж детали робота-манипулятора (вид сверху)

Исходными данными для проекта являются:

– материал детали робота манипулятора – конструкционная сталь;

– деталь – прямоугольная пластина с отверстием в центре и небольшим выступом вокруг отверстия;

– ширина детали *A3* = 200 миллиметров;

– толщина детали *S1* = 15 миллиметров;

– длинна детали *A1* = 300 миллиметров;

– радиус отверстия *D1*= 60 миллиметров;

– радиус выступа *D2* = 80 миллиметров;

– высота выступа *H2* = 10 миллиметров;

– закрепление детали – четыре отверстия в нижней грани детали, обозначенные синим цветом;

– сила, приложенная на выступ детали – 100 ньютонов, обозначена красными стрелками.

На рисунке 2.2 изображен чертеж разреза детали робота-манипулятора.

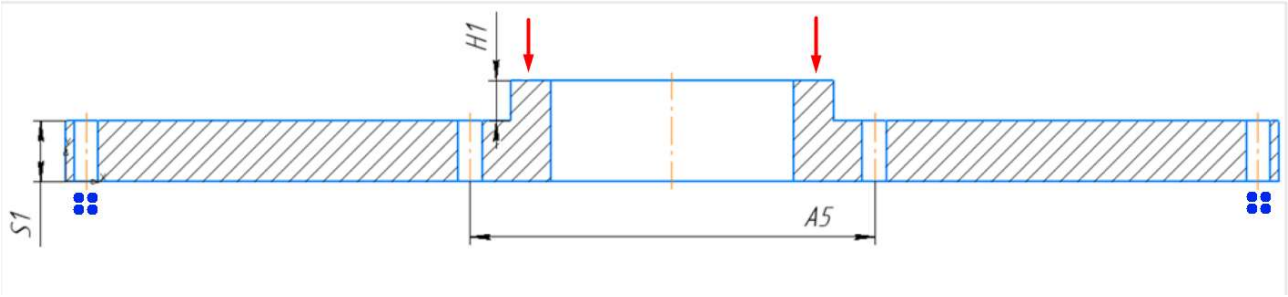

Рисунок 2.2 – Чертеж разреза детали робота-манипулятора

2.2 Математическая модель детали робота-манипулятора

Моделируемая конструкция представляет собой пластину с отверстием по середине и небольшим выступом вокруг отверстия. Распределённая нагрузка сверху на выступ равна 100 ньютонов. Закрепление применяется к нижней грани детали.

Материал, из которого сделана пластина – конструкционная сталь. Модуль Юнга для данного материала равен $2 \cdot 10^{11}$ Па, а коэффициент Пуассона равен 0,3.

Для того, чтобы построить математическую модель детали робота-манипулятора, необходимо разбить его на конечно-элементную сетку. В данном проекте построение конечно-элементной сетки происходит путем разбиения кубиков одинакового размера на шесть конечных элементов, в свою очередь кубики создаются путем наложения трехмерной сетки с заданным размером ячейки на деталь робота-манипулятора.

Применяя трехмерные, линейные конечные элементы возможно сделать довольно точную математическую модель заданной детали, с замененными окружностями на многогранные призмы.

В качестве конечных элементов используются четырехузловые тетраэдры.

Вектор узловых усилий тетраэдра представлен в виде:

$$\{R\}^T = \{X_1, Y_1, Z_1, X_2, Y_2, Z_2, X_3, Y_3, Z_3, X_4, Y_4, Z_4\}; \quad (2.1)$$

Узловым усилиям соответствуют следующие узловые перемещения:

$$\{F^e\}^T = \{u_1, \vartheta_1, w_1, u_2, \vartheta_2, w_2, u_3, \vartheta_3, w_3, u_4, \vartheta_4, w_4\}; \quad (2.2)$$

Линейные функции для перемещений узла тетраэдра имеют следующий вид:

$$\begin{gathered} u = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 z \\ \vartheta = \alpha_5 + \alpha_6 x + \alpha_7 y + \alpha_8 z \\ w = \alpha_9 + \alpha_{10} x + \alpha_{11} y + \alpha_{12} z \end{gathered}; \quad (2.3)$$

Матрица геометрических характеристик элемента:

$$[\mathrm{B}] = \frac{1}{6\mathrm{V}} \begin{bmatrix} a_i & 0 & 0 & b_j & 0 & 0 & b_m & 0 & 0 & b_n & 0 & 0 \\ 0 & c_i & 0 & 0 & c_j & 0 & 0 & c_m & 0 & 0 & c_n & 0 \\ 0 & 0 & d_i & 0 & 0 & d_j & 0 & 0 & d_m & 0 & 0 & d_n \\ c_i & b_i & 0 & c_j & b_j & 0 & c_m & b_m & 0 & c_n & b_n & 0 \\ 0 & d_i & c_i & 0 & c_j & b_j & 0 & d_m & c_m & 0 & d_n & c_n \\ d_i & 0 & b_i & d_j & 0 & b_j & d_m & 0 & b_m & d_n & 0 & b_n \end{bmatrix}, \quad (2.4)$$

где V – объём тетраэдра, который представлен в виде:

$$V = \frac{1}{6} \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}; \quad (2.5)$$

$$b_i = \begin{vmatrix} 1 & y_j & z_j \\ 1 & y_k & z_k \\ 1 & y_n & z_n \end{vmatrix}; \quad (2.6)$$

$$c_i = \begin{vmatrix} x_j & 1 & z_j \\ x_k & 1 & z_k \\ x_n & 1 & z_n \end{vmatrix}; \quad (2.7)$$

$$d_i = \begin{vmatrix} x_j & y_j & 1 \\ x_k & y_k & 1 \\ x_n & y_n & 1 \end{vmatrix}, \tag{2.8}$$

где *i, j, k, n* – номера вершин линейного конечного элемента, в виде тетраэдра.

Остальные значения *b, c, d* получаются круговой перестановкой индексов.

Используя уравнения Коши и физические уравнения теории упругости, вычисляются деформации. Формула деформаций:

$$\{\varepsilon\} = [B]\{u\}; \tag{2.9}$$

Также вычисляются и напряжения. Формула для напряжений:

$$\{\sigma\} = [\mathrm{E}]\{\varepsilon\}; \tag{2.10}$$

$$[\mathrm{E}] = p_1 \begin{bmatrix} p_2 & p_3 & p_3 & 0 & 0 & 0 \\ p_3 & p_2 & p_3 & 0 & 0 & 0 \\ p_3 & p_3 & p_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}, \tag{2.11}$$

$$\mathrm{p}_1 = \frac{\mathrm{E}}{1 + \nu}; \tag{2.12}$$

$$\mathrm{p}_2 = \frac{1 - \nu}{1 - 2\,\nu}; \tag{2.13}$$

$$\mathrm{p}_3 = \frac{\nu}{1 - 2\,\nu}; \tag{2.14}$$

где *E* – модуль Юнга,

ν – коэффициент Пуассона

Тогда матрица жёсткости тетраэдра принимает вид:

$$[k] = V[B]^T[\mathrm{E}][B]; \tag{2.15}$$

В результате матричных операций в (2.15) матрица жёсткости тетраэдра принимает вид:

$$[k] = \frac{1}{36}\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix}; \quad (2.16)$$

Далее происходит объединение локальных матриц жёсткости, то есть матриц жёсткости тетраэдров в глобальную, то есть в матрицу жёсткости детали. Формирование глобальной матрицы жёсткости происходит по формуле:

$$K_{ij} = \sum_{r=1}^{n} k_{ij}^{r}, \quad (2.17)$$

где n – количество конечных элементов, которыми дискретизирована рассматриваемая система;

K_{ij} – элемент глобальной матрицы жёсткости $[K]$, характеризующий вклад j-го единичного перемещения в i-й компонент узловых сил всей системы в целом;

k_{ij}^{r}– элемент локальной матрицы жёсткости $[k^r]$ r-го конечного элемента, характеризующий вклад j-го единичного перемещения в i-й компонент узловых сил.

Под знаком суммы ненулевой вклад дадут лишь элементы, примыкающие к узлу, в котором приложен i-й компонент сил. Таким образом, для всей рассматриваемой системы будет получена следующая СЛАУ, описывающая её:

$$[K]\{F^{\text{узл}}\} = \{R\}, \quad (2.18)$$

где $[K]$ – глобальная матрица жёсткости;

$\{F^{\text{узл}}\}$ – вектор узловых перемещений всей системы;

$\{R\}$ – вектор узловых усилий системы.

Однако это уравнение не учитывает закреплений в системе. Для того что бы их учесть необходимо для каждого закреплённого в матрице жёсткости узла занулить все элементы в строках и столбцах, которые к нему относятся и только на главной диагонали этого узла оставить единицы. А в векторе узловых усилий установить нули. Тогда перемещения этого узла при любом раскладе будут равны нулю.

После построения математической модели системы, представляющей собой СЛАУ, необходимо её решить и вычислить узловые перемещения системы. А после этого деформации и перемещения.

2.3 Стадии решения прочностной задачи методом конечных элементов

Первая стадия решения – геометрическое моделирование. Оно включает в себя создание геометрии модели конструкции, пригодной для МКЭ, с учётом всех параметров, которые могут оказать существенное влияние на результаты расчётов.

На первой стадии помимо ввода геометрических параметров конструкции задаются физические свойства материалов, из которых она изготовлена.

Следующая стадия – создание сетки конечных элементов. На этой стадии выясняется целесообразность использования различных видов конечных элементов (оболочечных, балочных, пластин, объёмных и т. д.) в рассматриваемой модели. На этой стадии выполняются мероприятия по созданию максимально возможного количества областей с регулярной сеткой конечных элементов. В местах, где предполагаются большие градиенты напряжений, необходима более мелкая сетка.

Далее происходит построение глобальной матрицы жёсткости системы для созданной конечно-элементной сетки.

После этого идёт стадия наложения граничных условий. На стадии наложения граничных условий учитывается, как действие активных сил, так и наложенных на систему связей. Приложение силовых факторов должно учитывать особенности реальной работы конструкции при рассматриваемых режимах эксплуатации. Количество связей должно быть достаточным, чтобы обеспечить построение кинематически неизменяемой модели.

Затем, в результате наложения граничных условий, формируется основное уравнение МКЭ. Основное уравнение МКЭ представляет собой систему линейных уравнений, которая решается одним из методов решения СЛАУ. В данном случае используется метод Гаусса.

На заключительном этапе происходит анализ полученных результатов путём получения полей законов распределения напряжений и деформаций, а также построения необходимых графических зависимостей либо табличных форм вывода результатов.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

3.1 Структура разработанного программного комплекса

Программный комплекс, решающий поставленную задачу, реализован на языке высокого уровня *C#*. Для создания графического интерфейса использована технология *WPF* [10].

Для разработки приложения была выбрана двухслойная архитектура. Она представляет собой разбиение приложения на две логических части. Каждая логическая часть представляет собой отдельный проект и имеет ссылку на предыдущий проект.

Первая часть – слой бизнес логики *BLL*. Именно здесь происходят все построения и вычисления. Она разбита на множество пространств имён, каждое из которых отвечает за свою задачу.

Interfaces представляет из себя набор интерфейсов для создания слабосвязанного кода. Они представлены в таблице 3.1.

Таблица 3.1 – Описание интерфейсов *BLL* слоя

Имя класса сущности	**Описание**
IElementsProvider	Интерфейс предоставляющей контракт по чтению элементов и их узлов из различных источников данных.
IGridSerializer	Интерфейс предоставляющий контракт по сериализации и десериализации данных о детали.
ILoadsProvider	Интерфейс предоставляющий контракт по загрузке нагрузок на узлы и фиксации узлов.
IMaterialsProvider	Интерфейс предоставляющий контракт по загрузке материалов.
ISolutionMaker	Интерфейс предоставляющий контракт по вычислению перемещений, деформаций и напряжений.

ModelMakers представляет из себя набор, состоящий из интерфейса и его реализации для отрисовки конечно-элементной сетки.

Описание класса и интерфейса, реализующих отрисовку конечно-элементной сетки находится в таблице 3.2.

Таблица 3.2 – Описание класса и интерфейса, реализующих отрисовку конечно-элементной сетки

Имя класса	**Описание**
IMeshModelMaker	Интерфейс предоставляющий контракт по отрисовке конечно-элементной сетке.
SimpleModelMaker	Класс предоставляющий функционал по отрисовке конечно-элементной сетке.

Model представляет из себя набор классов, отражающих предметную область.

Описание классов отражающих предметную область показано в таблице 3.3.

Таблица 3.3 – Описание классов *Models* слоя *BLL*

Имя класса сущности	**Описание**
Cube	Класс описывающий первоначальное разбиение сетки.
DetailMaterial	Класс описывающий материал.
Element	Класс содержащий информацию о конечном элементе и логику вычисления локальной матрицы жесткости элемента.
ElementType	Перечисление описывающие тип конечного элемента.
FiniteElementModel	Класс реализующий функции вычисления глобальной матрицы жесткости и описывающий конечно-элементную сетку.
LoadType	Перечисление описывающее тип нагрузки.
Node	Класс описывающий узел конечно-элементной сетки.
NodeLoad	Класс хранящий информацию о нагрузке на узел.
Circle	Класс представляющий составную часть детали в виде окружности.
GridSettings	Класс хранящий информацию о составных частях детали.
IFigure	Интерфейс описывающий составную часть детали.
Rectangle	Класс представляющий составную часть детали в виде прямоугольника

Следующее пространство имён – *Services*. Предоставляет различные сервисы, которые необходимы для загрузки, выгрузки данных, а также реализующих основные вычисления.

Более детальное описание сервисов находится в таблице 3.4.

Таблица 3.4 – Описание классов сервисов

Название сервиса	**Описание**
SLAESolver	Класс сервиса, реализующий метод Гаусса для решения СЛАУ.
SolutionMaker	Класс сервиса, реализующий вычисления деформации, напряжения и перемещения.
GridSerializer	Класс сервиса, предоставляющий функционал по сериализации и десериализации данных о детали.

Продолжение таблицы 3.4

JsonElementsProvider	Класс реализующий выгрузку конечно-элементной сетки из *JSON* файлов.
BaseTxtProvider	Абстрактный класс, предоставляющий базовый функционал по выгрузке данных из текстовых файлов.
TxtElementsProvider	Класс реализующий выгрузку конечно-элементной сетки из текстовых файлов.
TxtLoadsProvider	Класс реализующий выгрузку нагрузок из текстовых файлов.
TxtMaterialsProvider	Класс реализующий выгрузку материалов из текстовых файлов.

Вторая часть – слой графического интерфейса *UI*. В этом слое содержатся окна графического интерфейса и логика взаимодействия графического интерфейса и внутренних классов программы. Описание классов графического интерфейса слоя *UI* находится в таблице 3.5.

Таблица 3.5 – Описание классов графического интерфейса

Название класса	**Описание**
MainWindow	Класс, представляющий собой окно приложения. Это окно является главным. В этом окне происходит взаимодействие оконного интерфейса и главной модели представления программы.
MainViewModel	Класс, хранящий текущее состояние приложения и предоставляющий функционал по его изменению.

3.2 Функционал разработанного приложения

При запуске программы открывается главное окно пользовательского интерфейса. При этом программа загружает данные о материалах из файла по умолчанию, путь к которому прописан в файле конфигурации.

По умолчанию загружаются три файла связанные с деталью:

– файл с конечными элементами;
– файл с узлами сетки;
– файл с приложенными нагрузками и фиксацией.

Интерфейс приложения представляет из себя окно, визуально разделенное на три части:

– часть для загрузки данных;
– часть отображения детали;
– часть для отображения результатов после вычислений.

Внешний вид главного окна приложения изображён на рисунке 3.1.

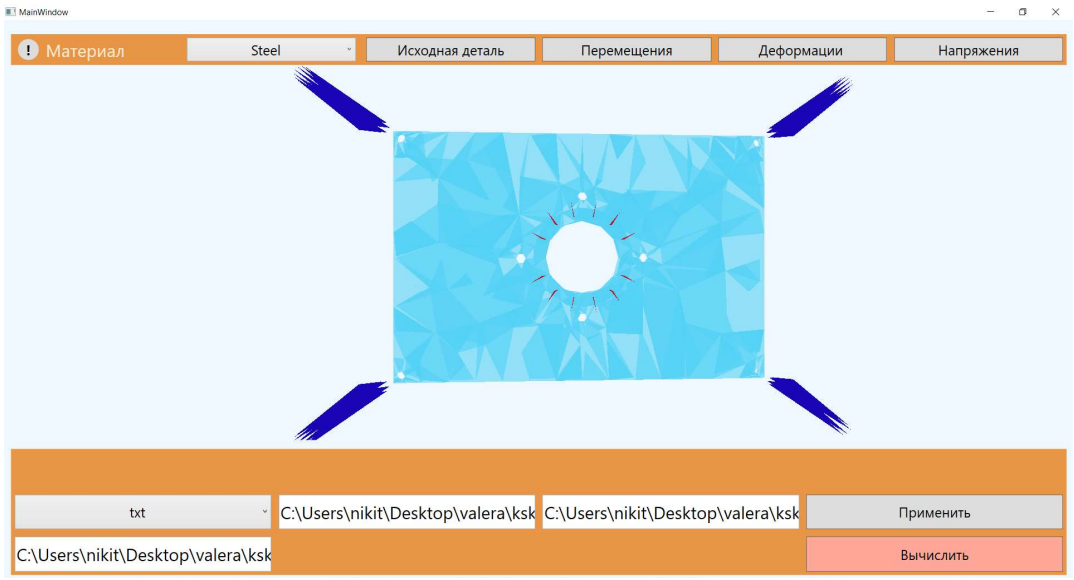

Рисунок 3.1 – Внешний вид главного окна приложения

Во время запуска программа строит конечно-элементную сетку для детали робота-манипулятора из файла по умолчанию, с заданными размерами.

Красным цветом обозначена нагрузка, приложенная к детали, а синим закрепления.

В программе имеется функционал работы с различными материалами. Для выбора материала необходимо открыть выпадающий список и выбрать нужный материал.

Чтобы посмотреть информацию о материале необходимо навести курсор на круглый значок с изображением восклицательного знака. После выбора материала деталь поменяет цвет в соответствии с цветом материала.

Внешний вид окна приложения после выбора другого материала и наведения на кнопку с подсказкой изображён на рисунке 3.2.

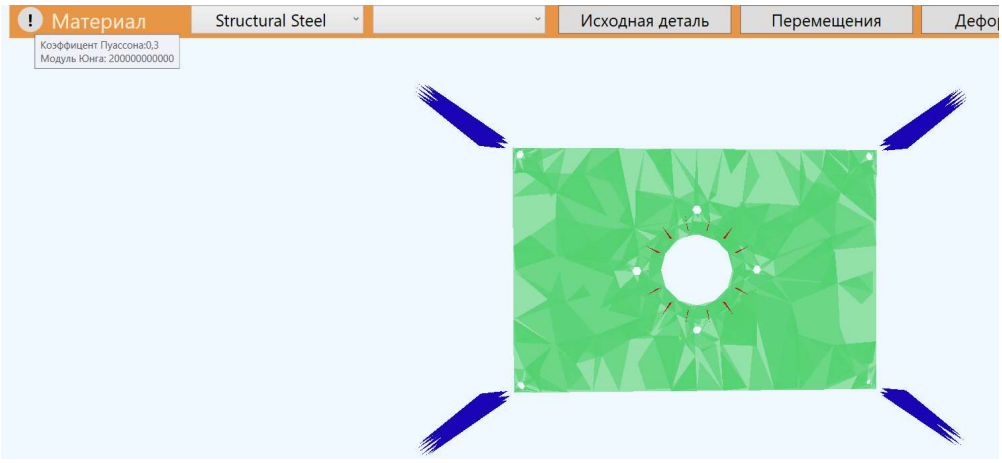

Рисунок 3.2 – Внешний вид детали после выбора материала

В программе предусмотрена загрузка данных двумя способами, в текстовом формате и формате *JSON*.

При загрузке данных с помощью текстового файла необходимо выбрать пути к трем файлам:

– к файлу с конечными элементами;
– к файлу с узлами сетки;
– к файлу нагрузок и закреплений.

Структура файла с конечными элементами для загрузки детали с помощью текстового формата показана на рисунке 3.3.

Element Number	Element Type	Nodes
1	Tet4	352 388 435 429
2	Tet4	352 388 387 435
3	Tet4	270 348 435 429
4	Tet4	115 270 348 435
5	Tet4	64 270 435 429

Рисунок 3.3 – Структура файла с конечными элементами

Файл содержит в себе три столбца:

– *Element Number*(номер конечного элемента)*;*
– *Element Type*(тип конечного элемента)*;*
– *Nodes*(номера узлы);

Структура файла с узлами сетки показана на рисунке 3.4.

Node Number	X Location (m)	Y Location (m)	Z Location (m)
1	-0,15001	9,999e-002	0,
2	-0,15001	-0,10001	0,
3	-0,15001	5,999e-002	0,
4	-0,15001	1,999e-002	0,

Рисунок 3.4 – Структура файла с узлами сетки

Структура файла с нагрузками и фиксациями показана на рисунке 3.5.

NodeId	Type	X	Y	Z
168	Force	0	0	-7.142857
169	Force	0	0	-7.142857
170	Force	0	0	-7.142857
120	Fixed	0	0	0
121	Fixed	0	0	0
122	Fixed	0	0	0

Рисунок 3.5 – Структура файла с нагрузками и фиксациями

Файл содержит в себе пять столбцов:

– *NodeId*(номер узла)*;*
– *Type*(тип нагрузка/фиксация)*;*
– *X*(значение по координате *X*);
– *Y*(значение по координате *Y*);
– *Z*(значение по координате *Z*).

При выборе типа «*Force*», то есть нагрузка, необходимо указать значение силы по одной или нескольким координатам в столбцах *X,Y,Z*. Знаком «-» задается направление силы.

При выборе типа «*Fixed*», то есть фиксация, чтобы зафиксировать узел во всех направлениях, необходимо в столбцах *X,Y,Z* указать нули. Если к узлу не прикладывается сила или фиксация, то указывать его в файле не нужно.

После нажатия на кнопку «Вычислить», если все данные введены корректно, произойдут вычисления и появится соответствующее сообщение (рисунок 3.6)

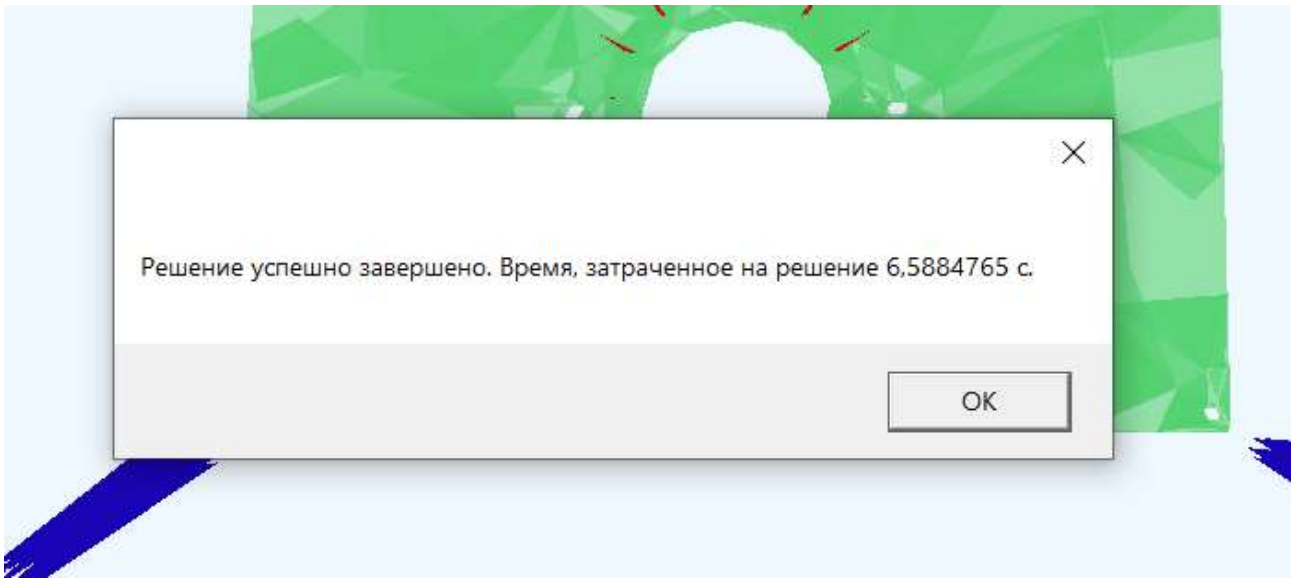

Рисунок 3.6 – Сообщение об успешных результатах вычислений

При необходимости и желании вернуться к исходной детали, необходимо нажать кнопку «Исходная деталь», которая загрузит файлы данных детали по умолчанию и отобразит ее.

После выполнения вычислений, у пользователя есть выбор, какие результаты вычислений отобразить:

– отобразить деформации;
– отобразить перемещения;
– отобразить напряжения.

Проверка результатов вычислений производится в следующей главе.

4 ВЕРИФИКАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Пример решения задачи разработанным приложением

После запуска приложения необходимо выбрать файлы с конечными элементами, узлами, нагрузками и фиксациями, выбрать материал и нажать кнопку «Применить». Когда появится деталь с приложенными нагрузками и фиксацией, необходимо нажать кнопку «Вычислить».

После вычисления было выведено время выполнения программы в всплывающем окне, и кнопка «Вычислить» окрасить в зеленый цвет.

Результат вычисления изображён на рисунке 4.1.

Рисунок 4.1 – Результат нажатия на кнопку «Вычислить»

После выполненных вычислений, пользователь может выбрать отображение одного из трех вариантов результатов:

– перемещения,
– деформации,
– напряжения.

После выбора отображения, в окне появится градиентная шкала с полученными значениями в результате вычисления.

4.2 Верификация полученных напряжений в пакете *ANSYS*

Значения максимальных напряжений содержатся в таблице 4.1.

Таблица 4.1 – Сравнение максимальных напряжений

№ п\п	Значение, полученное в приложении (Па)	Значение, полученное в системе ANSYS (Па)	Погрешность (%)
1	2525574	2525900	0,012
2	2188160	2188600	0,020
3	2059422	2060200	0,037
4	1939891	1940200	0,015
5	1575806	1576000	0,012
6	1370604	1375300	0,342
7	1338727	1341700	0,221
8	1332350	1332700	0,026
9	1148775	1147900	0,076
10	1138253	1139000	0,065

Средняя погрешность максимальных напряжений составила 0,082 процента. В следствии чего, можно сделать вывод о том, что программа производит довольно корректные подсчеты напряжений.

Внешний вид распределения напряжений изображён на рисунке 4.2.

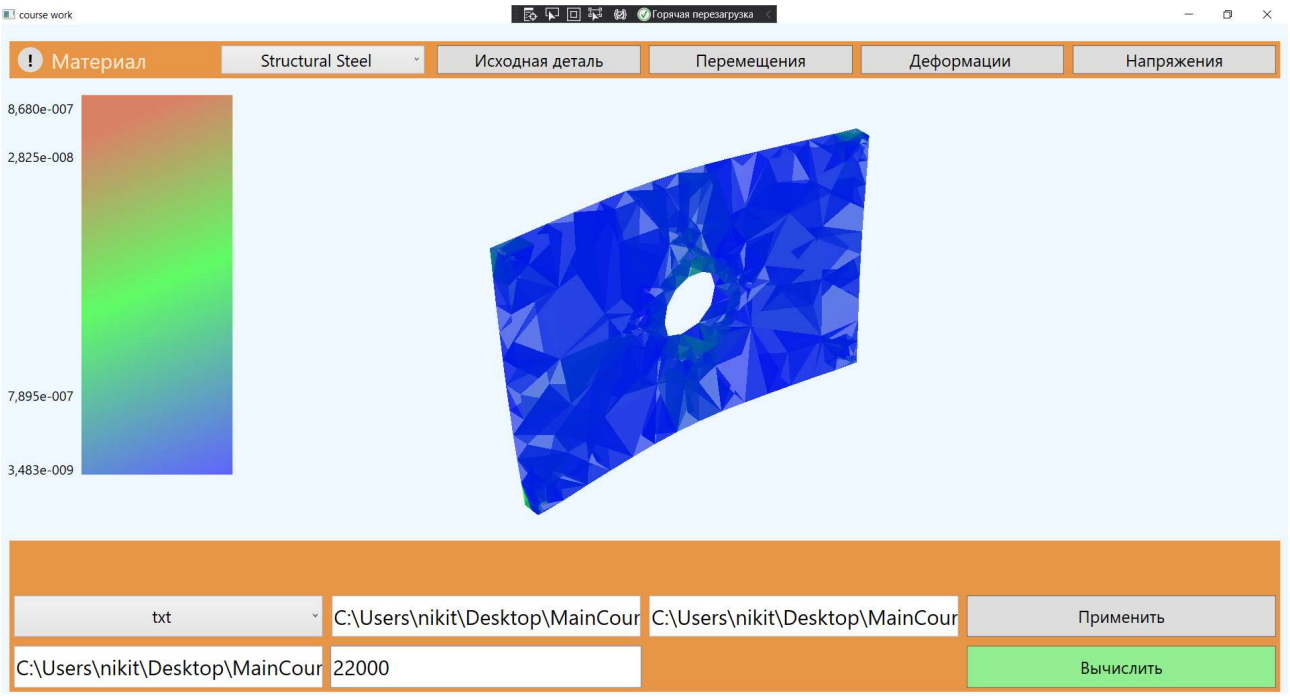

Рисунок 4.2 – Внешний вид распределения напряжений

На рисунке 4.3 изображены напряжения, полученные в пакете *ANSYS*.

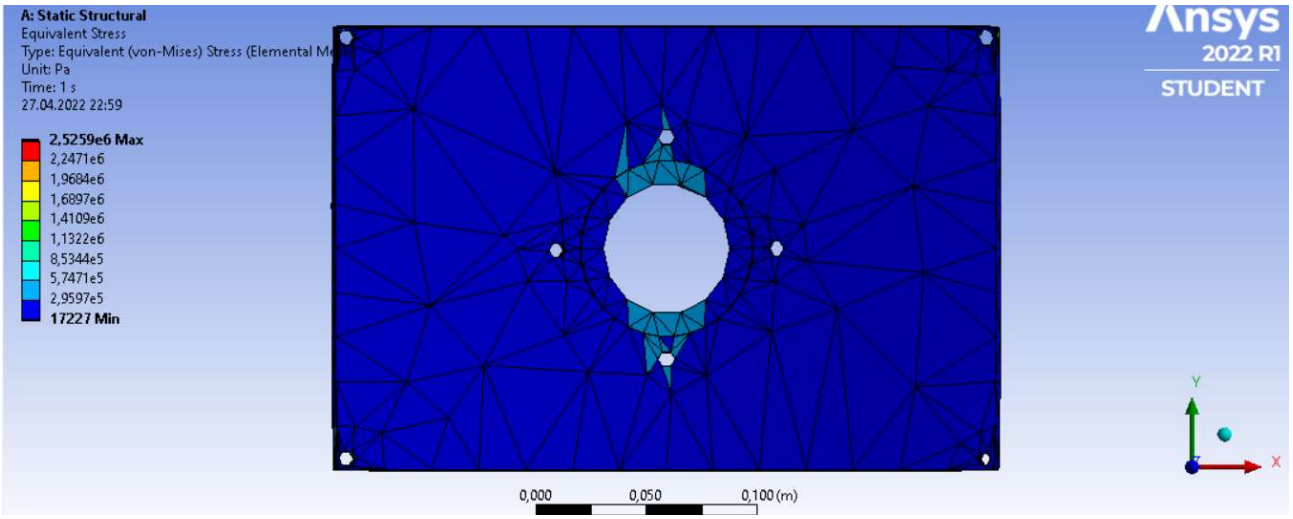

Рисунок 4.3 – Внешний вид распределения напряжений в пакете *ANSYS*

При сравнении изображения напряжений в приложении и в пакете *ANSYS* можно сделать вывод о том, что приложение довольно точно отображает узловые напряжения равномерно по всей детали.

При анализе вычислений напряжений стало понятно, что в углах у отверстий высокая концентрация напряжений. Поэтому было принято решение сместить отверстия дальше от краев детали и провести новые вычисления.

Значения максимальных напряжений после новых вычислений содержатся в таблице 4.2.

Таблица 4.2 – Сравнение максимальных напряжений после новых вычислений

№ п\п	Значение, полученное в приложении (Па)	Значение, полученное в системе ANSYS (Па)	Погрешность (%)
1	1048468	1048300	0,016
2	686112	686120	0,001
3	682551	682910	0,052
4	666430	666470	0,006
5	661009	661090	0,012
6	659634	659780	0,022
7	658159	658090	0,010
8	650391	641930	1,318
9	641955	632650	1,470
10	632564	630420	0,340

Средняя погрешность максимальных напряжений составила 0,3247 процента. В следствии чего, можно сделать вывод о том, что программа провела довольно корректные подсчеты напряжений. Также удалось уменьшить концентрацию напряжений почти в два с половиной раза, тем самым оптимизировав размеры отверстий детали робота-манипулятора.

Внешний вид распределения новых напряжений изображён на рисунке 4.4.

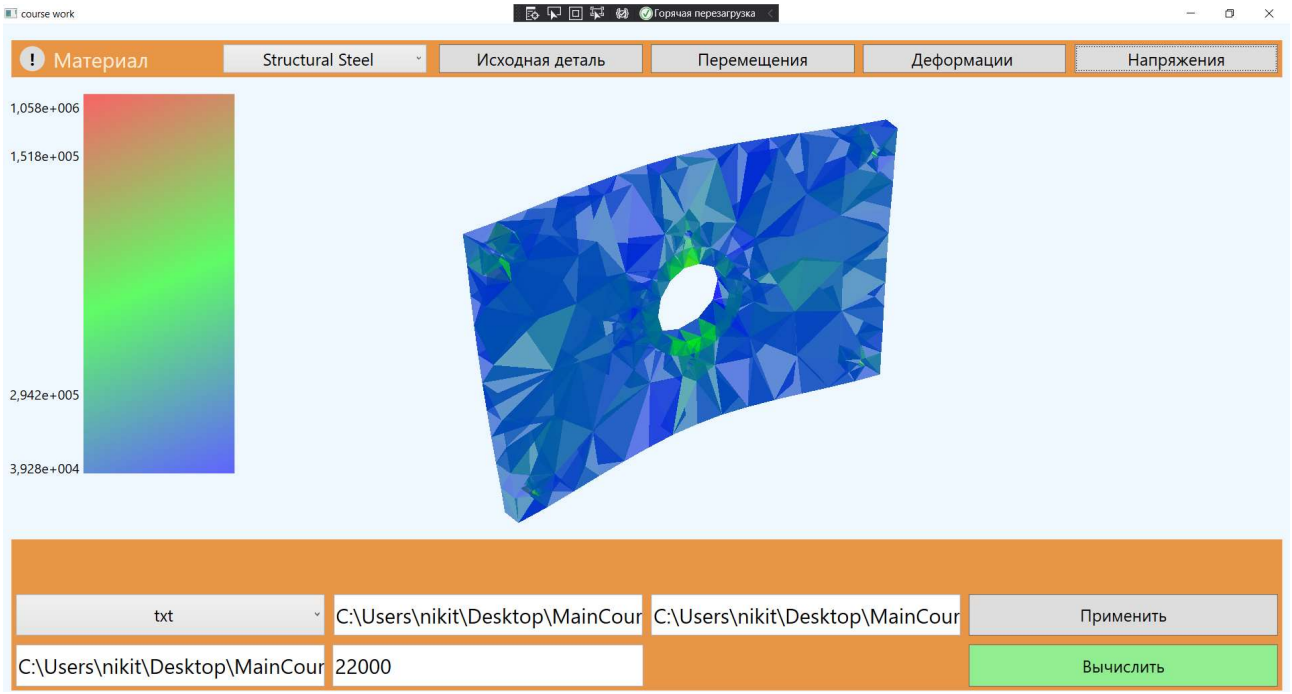

Рисунок 4.4 – Внешний вид распределения новых напряжений

На рисунке 4.5 изображены напряжения, полученные в пакете *ANSYS*.

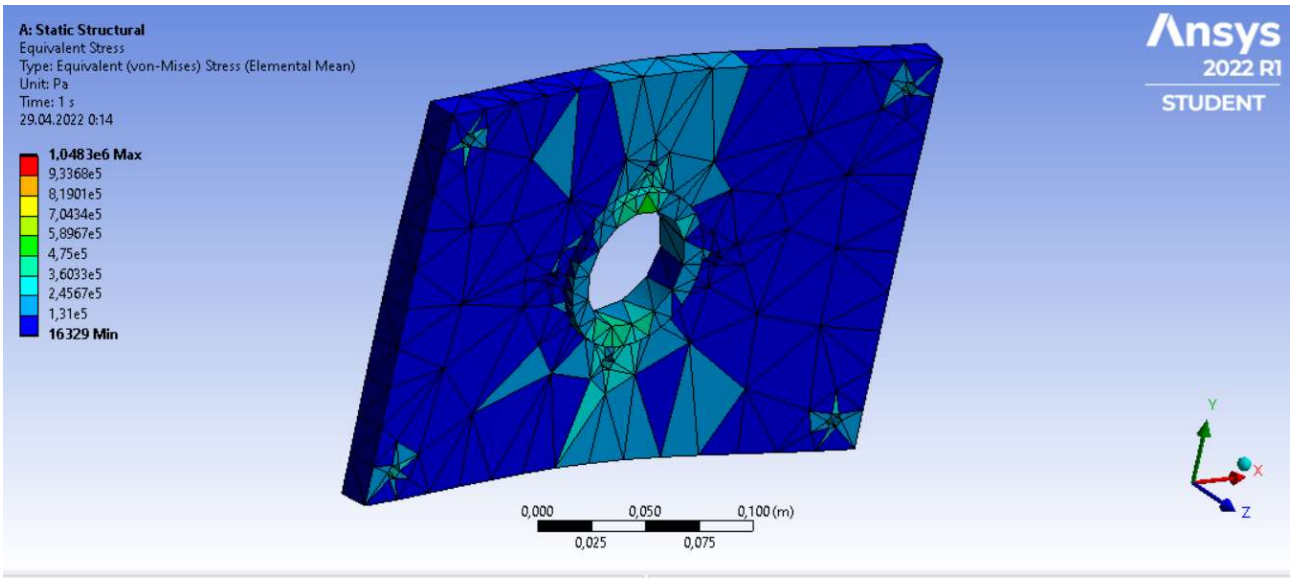

Рисунок 4.5 – Внешний вид распределения напряжений в пакете *ANSYS*

4.3 Верификация полученных перемещений в пакете *ANSYS*

Значения максимальных перемещений содержатся в таблице 4.3.

Таблица 4.3 – Сравнение максимальны напряжений

№ п\п	Значение, полученное в приложении $\times 10^{-7}$ (м)	Значение, полученное в системе ANSYS $\times 10^{-7}$ (м)	Погрешность (%)
1	6,3945	6,3216	1,153
2	6,3742	6,2980	1,209
3	6,3696	6,2881	1,279
4	6,3247	6,2802	0,703
5	6,2810	6,2737	0,116
6	6,2669	6,2726	0,090
7	6,2511	6,2706	0,311
8	6,2221	6,2675	0,724
9	6,2098	6,2288	0,305
10	6,2081	6,2261	0,289

Средняя погрешность максимальных перемещений составила 0,6341 процента. В следствии чего, можно сделать вывод о том, что программа производит довольно корректные подсчеты перемещений.

Внешний вид распределения перемещений изображён на рисунке 4.4.

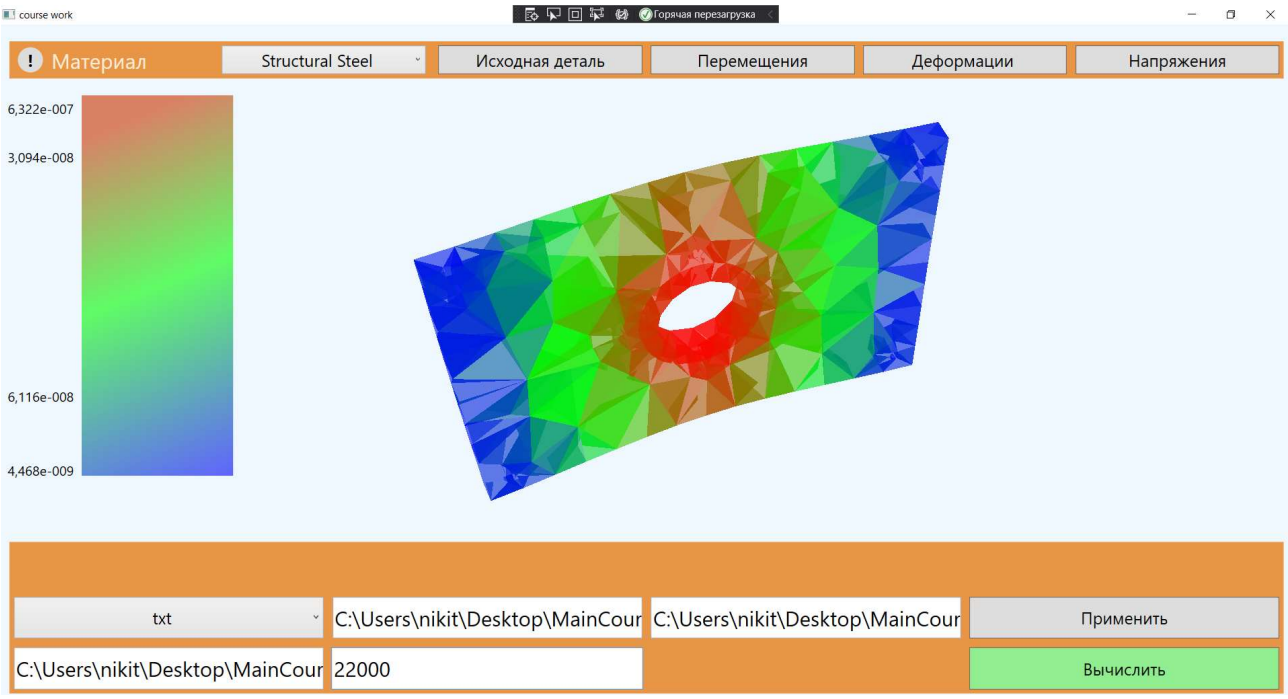

Рисунок 4.4 – Внешний вид распределения узловых перемещений

На рисунке 4.5 изображены перемещения, полученные в пакете *ANSYS*.

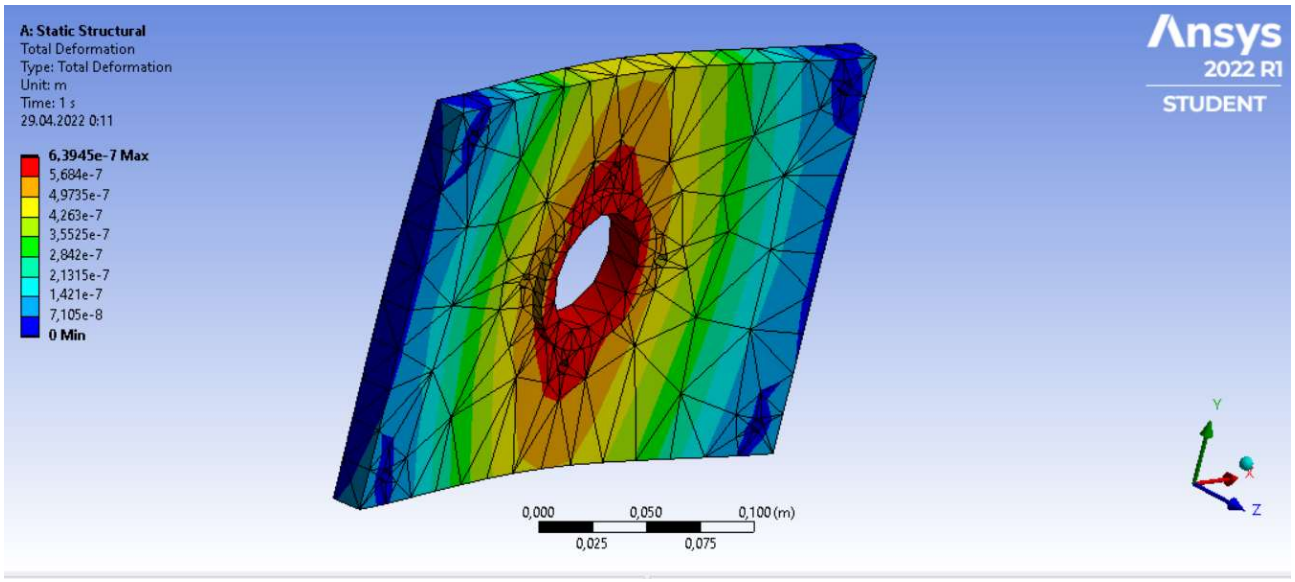

Рисунок 4.5 – Внешний вид распределения узловых перемещений

При сравнении изображения перемещений в приложении и в пакете *ANSYS* можно сделать вывод о том, что приложение довольно точно отображает узловые перемещения равномерно по всей детали.

ЗАКЛЮЧЕНИЕ

Для решения поставленной задачи было разработано приложение моделирования напряжённо-деформированного состояния детали робота-манипулятора при наличии нагрузки в виде силы, действующей на выступающую часть детали в виде окружности и закрепления в отверстиях по бокам в углах детали. Также удалось оптимизировать размеры отверстий детали робота-манипулятора.

Приложение разработано на языке программирования высокого уровня *C#*. Использована двухслойная архитектура приложения. Также в приложении реализован графический интерфейс пользователя при помощи технологии *WPF*.

Приложение позволяет быстро и эффективно решить поставленную задачу, используя метод конечных элементов. Полученные результаты перемещений, деформаций и напряжений представлены как в графическом, так и численном виде.

Разработанное приложение верифицировано с помощью пакета *ANSYS*. Результаты вычисления имеют весьма малую погрешность относительно этого пакета, что говорит о правильности реализации алгоритма решения поставленной задачи.

При сравнении разработанного программного комплекса с такими профессиональными пакетами как *ANSYS*, замечен недостаток в виде ограниченности в функционале и меньшей точности. Однако разработанное приложение имеет очень серьёзное преимущество, а именно стоимость. Так, некоторые предприятия, специализирующиеся в некоторой узкой области, не будут покупать такие большие пакеты как *ANSYS*, а смогут использовать достаточно простое и удобное в использовании разработанное приложение, способное решать поставленную задачу.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бугаенко, Г. А. Механика: учебник для вузов / Г. А. Бугаенко, В. В. Маланин, В. И. Яковлев. – 2-е изд., испр. и доп. – Москва: Издательство Юрайт, 2019. – 368 с.
2. Рассел, Джесси Метод дискретного элемента: моногр. / Джесси Рассел. – М.: VSD, 2013. – 716 с.
3. Метод гаусса [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Метод_Гаусса. – Дата доступа: 10.04.2021.
4. Шимкович, Д.Г. *Femap* & *Nastran*. Инженерный анализ методом конечных элементов / Д.Г. Шимкович. – М.: ДМК Пресс, 2018. – 738 с.
5. Ибрагимов, Н.Х. Практический курс дифференциальных уравнений и математического моделирования. Классические и новые методы. Нелинейные математические модели. Симметрия и принципы инвариантности / Н.Х. Ибрагимов. – М.: Физматлит, 2012. – 332 с.
6. Агальцов, В.П. Математические методы в программировании: Учебник / В.П. Агальцов, И.В. Волдайская. – М.: ИД Форум, 2013. – 240 с.
7. Алямовский, А. А. Инженерные расчеты и *SolidWorks Simulation* / А.А. Алямовский. – М.: ДМК Пресс, 2015. – 464 с.
8. Рихтер Дж. *CLR VIA C#.* Программирование на платформе *Microsoft.NET Framework* 4.5 на языке *C#* / Рихтер Дж. 4-е изд., перераб. и доп. – Питер: 2019 – 896 с.
9. Шарп Джон *Microsoft Visual C#*: учебное пособие / Шарп Джон – 8-е изд., – Питер, 2017. – 848
10. Особенности платформы *WPF*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: https://metanit.com/sharp/wpf/1.php – Дата доступа: 12.04.2021

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг основных классов приложения

Листинг интерфейса *IElementsProvider*
```
using CourseWork.BLL.Models;
namespace CourseWork.BLL.Interfaces
{
   public interface IElementsProvider
   {
      public FiniteElementModel GetModel();
   }
}
```

Листинг класса *GridSettings*

```
using CourseWork.BLL.Models.GridSettings;

namespace CourseWork.BLL.Interfaces
{
   public interface IGridSerializer
   {
      void Serialize(GridSettings grid, string path);

      GridSettings Deserialize(string path);
   }
}
```

Листинг интерфейса *ILoadsProvider*

```
using CourseWork.BLL.Models;
using System.Collections.Generic;

namespace CourseWork.BLL.Interfaces
{
   public interface ILoadsProvider
   {
      List<NodeLoad> GetNodeLoads();
   }
}
```

Листинг интерфейса *IMaterialsProvider*
```
using CourseWork.BLL.Models;
using System.Collections.Generic;

namespace CourseWork.BLL.Interfaces
{
   public interface IMaterialsProvider
   {
      List<DetailMaterial> GetMaterials();
   }
}
```

Листинг интерфейса *ISolutionMaker*

```
using CourseWork.BLL.Models;

namespace CourseWork.BLL.Interfaces
{
```

```
    public interface ISolutionMaker
    {
        double[] GetNodeDisplacements(FiniteElementModel model, DetailMaterial material);

        double[] GetElementsDisplacement(FiniteElementModel model, double[] nodeDisplacements);
        public double[] GetElementsStrain(FiniteElementModel originalMesh, double[] nodesDisplacement);
        public double[] GetElementsStresses(FiniteElementModel originalMesh, double[] nodesDisplacement, Detail-
Material material);
    }
}
```

Листинг интерфейса *IMeshModelMaker*

```
using CourseWork.BLL.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Media.Media3D;

namespace CourseWork.BLL.ModelMakers
{
    /// <summary>
    /// Интерфейс создателя геометрии сетки
    /// </summary>
    public interface IMeshModelMaker
    {
        /// <summary>
        /// Метод создания геометрии сетки
        /// </summary>
        /// <param name="mesh">Конечно-элементная сетка</param>
        /// <param name="material">Материал сетки</param>
        /// <returns>Геометрия сетки</returns>
        /// <param name="loads"></param>
        public Model3DGroup GenerateGeometry(FiniteElementModel mesh, MaterialGroup material);

        /// <summary>
        /// Вычисляет модель с градиентным окрасом. Градиент показывает изменения характеристики элемен-
тов сетки
        /// </summary>
        /// <param name="mesh">Сетка, на основе который строится модель</param>
        /// <param name="elementsCharacteristic">Характеристика (Напряжение, деформации и т.д.) конечных эле-
ментов в этой сетке</param>
        /// <returns>Модель с градиентным окрасом в зависимости от характеристики</returns>
        public Model3DGroup GenerateCharacteristicModel(FiniteElementModel mesh, double[] elementsCharacteristic);
    }
}
```

Листинг класса *SimpleModelMaker*

```
using CourseWork.BLL.Models;
using CourseWork.BLL.Tools;
using CourseWork.Common.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Media;
using System.Windows.Media.Media3D;

namespace CourseWork.BLL.ModelMakers
```

```
{
    /// <summary>
    /// Стандартный создатель геометрии сетки
    /// </summary>
    public class SimpleModelMaker : IMeshModelMaker
    {
        private readonly GradientMaker _gradientMaker;
        public SimpleModelMaker(GradientMaker gradientMaker)
        {
            _gradientMaker = gradientMaker;
        }
        /// <summary>
        /// Метод для создания стандартной геометрии сетки
        /// </summary>
        /// <param name="mesh">Конечно-элементная сетка</param>
        /// <param name="material">Материал детали</param>
        /// <param name="loads">Нагрузки детали</param>
        /// <returns>Геометрия сетки</returns>=
        public Model3DGroup GenerateGeometry(FiniteElementModel model, MaterialGroup material = default)
        {
            Model3DGroup mesh = new Model3DGroup();
            var modelGeometries = new List<Model3D>();

            foreach (var element in model.Elements)
            {
                GeometryModel3D geometryModel3D = new GeometryModel3D();
                geometryModel3D.Geometry = GetGeometryFromElement(element);

                MaterialGroup elementMaterial = material ?? GetMaterial();
                geometryModel3D.Material = elementMaterial;
                geometryModel3D.BackMaterial = elementMaterial;
                modelGeometries.Add(geometryModel3D);
            }

            foreach (var load in model.NodeLoads)
            {
                GeometryModel3D geometryModel3D = new GeometryModel3D();
                var node = model.Nodes.First(n => n.Id == load.NodeId);
                geometryModel3D.Geometry = GetPointGeometry(node.Position);

                var color = load.LoadType switch
                {
                    LoadType.Fixed => Color.FromRgb(24, 4, 180),
                    LoadType.Force => Color.FromRgb(203, 11, 32),
                    _ => throw new InvalidOperationException($"Тип нагрузки '{load.LoadType}' не поддерживается.")
                };
                geometryModel3D.Material = GetMaterial(color);
                geometryModel3D.BackMaterial = GetMaterial(color);
                modelGeometries.Add(geometryModel3D);
            }

            mesh.Children = new Model3DCollection(modelGeometries);

            return mesh;
        }

        /// <summary>
        /// Вычисляет модель с градиентным окрасом. Градиент показывает изменения характеристики элемен-
тов сетки
        /// </summary>
        /// <param name="mesh">Сетка, на основе который строится модель</param>
```

```
        /// <param name="elementsCharacteristic">Характеристика (Напряжение, деформации и т.д.) конечных элементов в этой сетке</param>
        /// <returns>Модель с градиентным окрасом в зависимости от характеристики</returns>
        public Model3DGroup GenerateCharacteristicModel(FiniteElementModel mesh, double[] elementsCharacteristic)
        {
            double minCharacteristic = elementsCharacteristic.Min();
            double maxCharacteristic = elementsCharacteristic.Max();

            Model3DGroup model = new Model3DGroup();
            var modelGeometries = new Model3D[mesh.Elements.Count];

            for (int i = 0; i < mesh.Elements.Count; i++)
            {
                GeometryModel3D geometryModel3D = new GeometryModel3D();
                geometryModel3D.Geometry = GetGeometryFromElement(mesh.Elements[i]);

                MaterialGroup elementMaterial = GetMaterialRGBGradientValue(minCharacteristic, maxCharacteristic, elementsCharacteristic[i]);

                geometryModel3D.Material = elementMaterial;
                geometryModel3D.BackMaterial = elementMaterial;
                modelGeometries[i] = geometryModel3D;
            }

            model.Children = new Model3DCollection(modelGeometries);

            return model;
        }

        /// <summary>
        /// Метод возвращает материал на основе градиента
        /// </summary>
        /// <param name="minValue">Минимальное значение параметра для градиента</param>
        /// <param name="maxValue">Максимальное значение параметра для градиента</param>
        /// <param name="value">Текущее значение параметра для градиента</param>
        /// <returns></returns>
        private MaterialGroup GetMaterialRGBGradientValue(double minValue, double maxValue, double value)
        {
            (byte R, byte G, byte B) = _gradientMaker.GetRGBGradientValue(minValue, maxValue, value);
            MaterialGroup material = new MaterialGroup();
            System.Windows.Media.Media3D.Material[] materials = new System.Windows.Media.Media3D.Material[1];
            materials[0] = new DiffuseMaterial(new SolidColorBrush(System.Windows.Media.Color.FromArgb(0x99, R, G, B)));
            material.Children = new MaterialCollection(materials);

            return material;
        }


        private MaterialGroup GetMaterial(Color? color = default)
        {
            var material = new MaterialGroup();
            var materials = new Material[] { new DiffuseMaterial(new SolidColorBrush( color ?? Color.FromArgb(0x99, 0x53, 0xd2, 0xf5))) };
            material.Children = new MaterialCollection(materials);
            return material;
        }

        private MeshGeometry3D GetGeometryFromElement(Element element)
        {
            MeshGeometry3D triangleGeometry3D = new MeshGeometry3D();
```

```
            Point3D[] positionsArray = new Point3D[element.Nodes.Count];
            for (int i = 0; i < element.Nodes.Count; i++)
            {
                positionsArray[i] = new Point3D(element.Nodes[i].Position.X, element.Nodes[i].Position.Y, ele-
ment.Nodes[i].Position.Z);
            }
            Point3DCollection positions = new Point3DCollection(positionsArray);

            int[] triangleIndeces = new int[]
            {
                0, 3, 1,
                0, 1, 2,
                0, 2, 3,
                1, 2, 3
            };
            Int32Collection indeces = new Int32Collection(triangleIndeces);

            triangleGeometry3D.Positions = positions;
            triangleGeometry3D.TriangleIndices = indeces;

            return triangleGeometry3D;
        }

        private MeshGeometry3D GetPointGeometry(Point position)
        {
            MeshGeometry3D triangleGeometry3D = new MeshGeometry3D();

            Point3D[] positionsArray = new Point3D[3];
            positionsArray[0] = new Point3D(position.X + 0.1 * position.X, position.Y + 0.1 * position.Y, posi-
tion.Z + 0.1 * position.Z);
            positionsArray[1] = new Point3D(position.X + 0.55 * position.X, position.Y + 0.55 * position.Y, posi-
tion.Z + 0.25 * position.Z);
            positionsArray[2] = new Point3D(position.X + 0.22 * position.X, position.Y + 0.12 * position.Y, posi-
tion.Z + 0.12 * position.Z);
            Point3DCollection positions = new Point3DCollection(positionsArray);

            int[] triangleIndeces = new int[]
            {
                0, 2, 1,
            };
            Int32Collection indeces = new Int32Collection(triangleIndeces);

            triangleGeometry3D.Positions = positions;
            triangleGeometry3D.TriangleIndices = indeces;

            return triangleGeometry3D;
        }
    }
}
```

Листинг класса *IFigure*

```
using CourseWork.Common.Models;
using System;

namespace CourseWork.BLL.Models.GridSettings
{
    public class Circle : IFigure
    {
        public int Radius { get; set; }

        public bool IsFill { get; set; }
```

```
        public Point StartPoint { get; set; }

        public int Height { get; set; }

        public void Draw(int[,,] matrix, double step = 0.1)
        {
            var filler = IsFill ? 1 : 0;
            for (var z = (int)(StartPoint.Z/step); z < StartPoint.Z/step + Height/step; z++)
            {
                for(var x = 0; x < matrix.GetLength(0); x++)
                {
                    for (var y = 0; y < matrix.GetLength(1); y++)
                    {
                        if(Math.Sqrt(Math.Pow(StartPoint.X/step - x, 2) + Math.Pow(StartPoint.Y / step - y, 2)) <= Ra-
dius / step )
                        {
                            matrix[x, y, z] = filler;
                        }
                    }
                }
            }
        }
    }
}
```

Листинг класса *GridSettings*

```
using System.Collections.Generic;

namespace CourseWork.BLL.Models.GridSettings
{
    public class GridSettings
    {
        public List<IFigure> Figures { get; set; }

        public int Lenght { get; set; }

        public int Width { get; set; }

        public int Height { get; set; }
    }
}
```

Листинг интерфейса *IFigure*

```
using CourseWork.Common.Models;

namespace CourseWork.BLL.Models.GridSettings
{
    public interface IFigure
    {

        Point StartPoint { get; set; }

        int Height { get; set; }

        bool IsFill { get; set; }

        void Draw(int[,,] matrix, double step = 0.1);
    }
}
```

Листинг класса *Rectangle*

```
using CourseWork.Common.Models;

namespace CourseWork.BLL.Models.GridSettings
{
   public class Rectangle : IFigure
   {
      public Point StartPoint { get; set; }
      public bool IsFill { get; set; }

      public int Length { get; set; }

      public int Width { get; set; }

      public int Height { get; set; }

      public void Draw(int[,,] matrix, double step = 0.1)
      {
         var filler = IsFill ? 1 : 0;

         for (var z = (int)(StartPoint.Z / step); z < StartPoint.Z / step + Height / step; z++)
         {
            for (var x = (int)(StartPoint.X / step); x < StartPoint.X / step + Height / step; x++)
            {
               for (var y = (int)(StartPoint.Y / step); y < StartPoint.Y / step + Height / step; y++)
               {
                  matrix[x, y, z] = filler;
               }
            }
         }
      }
   }
}
```

Листинг класса *Cube*

```
using System.Collections.Generic;

namespace CourseWork.BLL.Models
{
   public class Cube
   {
      public List<Node> Nodes { get; set; }

      public List<Element> ToElements()
      {
         return new List<Element> {
         new Element
         {
            Id = 1,
            Nodes = new List<Node> { Nodes[0], Nodes[1], Nodes[4], Nodes[7] }
         },
         new Element
         {
            Id = 2,
            Nodes = new List<Node> { Nodes[1], Nodes[4], Nodes[5], Nodes[7] }
         },
         new Element
         {
```

```
            Id = 3,
            Nodes = new List<Node> { Nodes[0], Nodes[1], Nodes[3], Nodes[7] }
        },
        new Element
        {
            Id = 4,
            Nodes = new List<Node> { Nodes[1], Nodes[5], Nodes[6], Nodes[7] }
        },
        new Element
        {
            Id = 5,
            Nodes = new List<Node> { Nodes[1], Nodes[2], Nodes[6], Nodes[7] }
        },
        new Element
        {
            Id = 6,
            Nodes = new List<Node> { Nodes[1], Nodes[2], Nodes[3], Nodes[7] }
        }
        };
    }
  }
}
```

Листинг класса *DetailMaterial*

```
using System.Windows.Media;

namespace CourseWork.BLL.Models
{
  public class DetailMaterial
  {
    public DetailMaterial(string name, double youngModulus, double poissonCoefficient, Color color)
    {
      Name = name;
      YoungModulus = youngModulus;
      PoissonCoefficient = poissonCoefficient;
      LameCoefficient = PoissonCoefficient * YoungModulus / ((1 + PoissonCoefficient) * (1 - 2 * PoissonCoeffi-
cient));
      ShiftModulus = YoungModulus / (2 * (1 + PoissonCoefficient));
      Color = color;
    }

    public string Name { get; }

    public double YoungModulus { get; }

    public double PoissonCoefficient { get; }

    public double LameCoefficient { get; }

    public double ShiftModulus { get; }

    public Color Color { get; }

    public string GetInfo()
    {
      return $"Коэффицент Пуассона:{PoissonCoefficient}\nМодуль Юнга: {YoungModulus}";
    }

    public override string ToString()
    {
      return Name;
```

```
        }
    }
}
```

Листинг класса *Element*

```
using System;
using System.Collections.Generic;
using System.Linq;
using Accord.Math;

namespace CourseWork.BLL.Models
{
    public class Element : ICloneable
    {
        public ElementType ElementType { get; set; } = ElementType.Tet4;

        public int Id { get; set; }

        public List<Node> Nodes { get; set; }

        /// <summary>
        /// Объём конечного элемента
        /// </summary>
        public double Volume
        {
            get
            {
                return Math.Abs(Matrix.Determinant(
                new double[,]
                {
                    {1,Nodes[0].Position.X, Nodes[0].Position.Y, Nodes[0].Position.Z},
                    {1,Nodes[1].Position.X, Nodes[1].Position.Y, Nodes[1].Position.Z},
                    {1,Nodes[2].Position.X, Nodes[2].Position.Y, Nodes[2].Position.Z},
                    {1,Nodes[3].Position.X, Nodes[3].Position.Y, Nodes[3].Position.Z}
                }) / 6);
            }
        }

        public (int i, int j, int m) getBias(int index)
        {
            var i = (index + 1) < 4 ? index + 1 : index + 1 - 4;
            var j = (index + 2) < 4 ? index + 2 : index + 2 - 4;
            var m = (index + 3) < 4 ? index + 3 : index + 3 - 4;
            return (i, j, m);
        }

        public double get_b(int index) {
            var (i, j, m) = getBias(index);
            var b = new double[,]
                {
                    {1, Nodes[i].Position.Y, Nodes[i].Position.Z },
                    {1, Nodes[j].Position.Y, Nodes[j].Position.Z },
                    {1, Nodes[m].Position.Y, Nodes[m].Position.Z },
                };
        return Matrix.Determinant(b);
        }

        public double get_c(int index)
        {
            var (i, j, m) = getBias(index);
            var c = new double[,]
```

```
            {
                {Nodes[i].Position.X, 1, Nodes[i].Position.Z },
                {Nodes[j].Position.X, 1, Nodes[j].Position.Z },
                {Nodes[m].Position.X, 1, Nodes[m].Position.Z },
            };
        return Matrix.Determinant(c);
    }

    public double get_d(int index)
    {
        var (i, j, m) = getBias(index);
        var d = new double[,]
            {
                {Nodes[i].Position.X, Nodes[i].Position.Y, 1 },
                {Nodes[j].Position.X, Nodes[j].Position.Y, 1 },
                {Nodes[m].Position.X, Nodes[m].Position.Y, 1 },
            };
        return Matrix.Determinant(d);
    }

    public double[,] GeometryMatrix()
    {
        var bi = get_b(0);
        var ci = get_c(0);
        var di = get_d(0);
        var bj = get_b(1);
        var cj = get_c(1);
        var dj = get_d(1);
        var bm = get_b(2);
        var cm = get_c(2);
        var dm  = get_d(2);
        var bn = get_b(3);
        var cn = get_c(3);
        var dn = get_d(3);
        var b = new double[,]
        {
                {bi, 0, 0, bj, 0, 0, bm, 0, 0, bn, 0, 0},
                {0, ci, 0, 0, cj, 0, 0, cm, 0, 0, cn, 0},
                {0, 0, di, 0, 0, dj, 0, 0, dm, 0, 0, dn},
                {ci, bi, 0, cj, bj, 0, cm, bm, 0, cn, bn, 0},
                {0, di, ci, 0, dj, cj, 0, dm, cm, 0, dn, cn},
                {di, 0, bi, dj, 0, bj, dm, 0, bm, dn, 0, bn},
        };
        return b.Divide(6 * Volume);
    }

    //public double[,] getElastic(DetailMaterial material)
    //{
    //    var k1 = material.PoissonCoefficient / (1 - material.PoissonCoefficient);
    //    var k2 = (1 - 2 * material.PoissonCoefficient) / (2 * (1 - material.PoissonCoefficient));

    //    var D = new double[,]
    //    {
    //            {1, k1, k1, 0, 0, 0},
    //            {k1, 1, k1, 0, 0, 0},
    //            {k1, k1, 1, 0, 0, 0},
    //            {0, 0, 0, k2, 0, 0},
    //            {0, 0, 0, 0, k2, 0},
    //            {0, 0, 0, 0, 0, k2},
    //    };
    //    return D.Multiply(material.YoungModulus * (1 - material.PoissonCoefficient) / ((1 + material.PoissonCoeffi-
cient) * (1 - 2 * material.PoissonCoefficient)));
```

```
//}
public double[,] getElastic(DetailMaterial material)
{
   var k1 = material.YoungModulus / (1 + material.PoissonCoefficient);
   var k2 = (1 - material.PoissonCoefficient) / (1 - 2 * material.PoissonCoefficient);
   var k3 = material.PoissonCoefficient / (1 - 2 * material.PoissonCoefficient);
   var D = new double[,]
   {
         {k2, k3, k3, 0, 0, 0},
         {k3, k2, k3, 0, 0, 0},
         {k3, k3, k2, 0, 0, 0},
         {0, 0, 0, 0.5, 0, 0},
         {0, 0, 0, 0, 0.5, 0},
         {0, 0, 0, 0, 0, 0.5},
   };
   return D.Multiply(k1);
}

public double[,] GetLocalStiffnessMatrix(DetailMaterial material)
{
   return GeometricMatrix.Transpose().Dot(getElastic(material)).Dot(GeometricMatrix).Multiply(Volume);
}

///// <summary>
///// Вычисление матрицы жесткости линейного тетраэдра
///// </summary>
///// <param name="material">Материал тетраэдра</param>
///// <returns>Матрица жесткости линейного тетраэдра</returns>
//public double[,] GetLocalStiffnessMatrix(DetailMaterial material)
//{
//    return GeometricMatrix.Transpose().Dot(getElastic(material)).Dot(GeometricMatrix).Multiply(Volume);
//}

/// <summary>
/// Геометрическая матрица конечного элемента [D]
/// </summary>
public double[,] GeometricMatrix
{
   get
   {
      double[,] Q = new double[6, 12];
      Q[0, 1] = 1;
      Q[1, 6] = 1;
      Q[2, 11] = 1;
      Q[3, 2] = 1;
      Q[3, 5] = 1;
      Q[4, 7] = 1;
      Q[4, 10] = 1;
      Q[5, 3] = 1;
      Q[5, 9] = 1;

      return Q.Dot(GetInversedA());
   }
}

/// <summary>
/// Вычисляет обратную матрицу A для вычисления геометрической матрицы
/// </summary>
/// <returns>Обратная матрица A</returns>
public double[,] GetInversedA()
{
```

```
    double[,] inversedA = new double[12, 12];
    for (int i = 0; i < 4; i++)
    {
        (double A, double B, double C, double D) = GetABCD(i);
        for (int j = 0; j < 3; j++)
        {
            inversedA[4 * j, i * 3 + j] = A;
            inversedA[4 * j + 1, i * 3 + j] = B;
            inversedA[4 * j + 2, i * 3 + j] = C;
            inversedA[4 * j + 3, i * 3 + j] = D;
        }
    }
    return inversedA.Multiply(1 / (6 * Volume));
}

/// <summary>
/// Метод, получающий коэффициенты геометрических характеристик
/// </summary>
/// <param name="index">Индекс узла</param>
/// <returns>Коэффициенты геометрических характеристик</returns>
private (double, double, double, double) GetABCD(int index)
{
    (int i, int j, int m) = ShiftIndexes(index);
    double A = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        { Nodes[i].Position.X, Nodes[i].Position.Y, Nodes[i].Position.Z },
        { Nodes[j].Position.X, Nodes[j].Position.Y, Nodes[j].Position.Z },
        { Nodes[m].Position.X, Nodes[m].Position.Y, Nodes[m].Position.Z }
    });
    double B = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        {1, Nodes[i].Position.Y, Nodes[i].Position.Z },
        {1, Nodes[j].Position.Y, Nodes[j].Position.Z },
        {1, Nodes[m].Position.Y, Nodes[m].Position.Z }
    });
    double C = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        { Nodes[i].Position.X, 1,Nodes[i].Position.Z },
        { Nodes[j].Position.X, 1,Nodes[j].Position.Z },
        { Nodes[m].Position.X, 1,Nodes[m].Position.Z }
    });
    double D = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        { Nodes[i].Position.X, Nodes[i].Position.Y, 1 },
        { Nodes[j].Position.X, Nodes[j].Position.Y, 1 },
        { Nodes[m].Position.X, Nodes[m].Position.Y, 1 }
    });
    return (A, B, C, D);
}

/// <summary>
/// Выполняет круговую перестановку индексов
/// </summary>
/// <param name="index">Текущий индекс</param>
/// <returns>Кортеж индексов после круговой перестановки</returns>
private (int, int, int) ShiftIndexes(int index)
{
    return ((index + 1) % 4, (index + 2) % 4, (index + 3) % 4);
}

/// <summary>
/// Вычисляет эластичную матрицу линейного тетраэдра
```

```
        /// </summary>
        /// <param name="material">Материал линейного тетраэдра</param>
        /// <returns>Эластичная матрица линейного тетраэдра</returns>
        public double[,] GetElasticMatrix(DetailMaterial material)
        {
            double ro = material.ShiftModulus * 2 + material.LameCoefficient;
            return new double[,]
            {
                {ro, material.LameCoefficient, material.LameCoefficient, 0, 0, 0 },
                {material.LameCoefficient, ro, material.LameCoefficient, 0, 0, 0 },
                {material.LameCoefficient, material.LameCoefficient, ro, 0, 0, 0 },
                {0, 0, 0, material.ShiftModulus, 0, 0 },
                {0, 0, 0, 0, material.ShiftModulus, 0 },
                {0, 0, 0, 0, 0, material.ShiftModulus }
            };
        }

        public override string ToString()
        {
            return string.Join("\n", Nodes);
        }

        public object Clone()
        {
            return new Element
            {
                Id = Id,
                ElementType = ElementType,
                Nodes = Nodes.Select(n => (Node)n.Clone()).ToList(),
            };
        }
    }
}
```

Листинг перечисления *ElementType*

```
using System;

namespace CourseWork.BLL.Models
{
    public enum ElementType
    {
        Tet4
    }

    public static class ElementTypeExtensions
    {
        public static int GetNodeCount(this ElementType elementType)
        {
            return elementType switch
            {
                ElementType.Tet4 => 4,
                _ => throw new NotSupportedException($"Element type {elementType} is not supported."),
            };
        }
    }
}
```

Листинг класса *FiniteElementModel*

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;

namespace CourseWork.BLL.Models
{
   public class FiniteElementModel : ICloneable
   {
      public FiniteElementModel(List<Element> elements, List<Node> nodes, List<NodeLoad> nodeLoads = default)
      {
         Elements = elements;
         Nodes = nodes;
         if (nodeLoads != null)
         {
            NodeLoads = nodeLoads;
         }
      }

      public List<Element> Elements { get; set; } = new List<Element>();

      public List<Node> Nodes { get; set; } = new List<Node>();

      public List<NodeLoad> NodeLoads { get; set; } = new List<NodeLoad>();

      /// <summary>
      /// Gets global stiffness matrix.
      /// </summary>
      /// <param name="material">Material.</param>
      /// <returns>Global stiffness matrix.</returns>
      public double[,] GetGlobalStiffnessMatrix(DetailMaterial material)
      {
         var globalMatrix = new double[Nodes.Count * 3, Nodes.Count * 3];
         foreach (var element in Elements)
         {
            var localMatrix = element.GetLocalStiffnessMatrix(material);
            for (int i = 0; i < element.Nodes.Count; i++)
            {
               for (int j = 0; j < element.Nodes.Count; j++)
               {
                  for (int k = 0; k < 3; k++)
                  {
                     for (int n = 0; n < 3; n++)
                     {
                        globalMatrix[3 * element.Nodes[i].Id + k, 3 * element.Nodes[j].Id + n] += localMa-
trix[3 * i + k, 3 * j + n];
                     }
                  }
               }
            }
         }

         return ApplyFixedSupport(globalMatrix);
      }

      public double[] GetNodeForces()
      {
         double[] forces = new double[Nodes.Count * 3];
         foreach (var load in NodeLoads)
         {
            if (load.LoadType == LoadType.Force)
            {
               forces[load.NodeId * 3] = load.X;
               forces[load.NodeId * 3 + 1] = load.Y;
               forces[load.NodeId * 3 + 2] = load.Z;
```

```
            }
        }

        return forces;
    }

    private double[,] ApplyFixedSupport(double[,] globalMatrix)
    {
        foreach (var load in NodeLoads)
        {
            if (load.LoadType == LoadType.Fixed)
            {
                for (int i = 0; i < Nodes.Count * 3; i++)
                {
                    globalMatrix[3 * load.NodeId, i] = 0;
                    globalMatrix[3 * load.NodeId + 1, i] = 0;
                    globalMatrix[3 * load.NodeId + 2, i] = 0;
                    globalMatrix[i, 3 * load.NodeId] = 0;
                    globalMatrix[i, 3 * load.NodeId + 1] = 0;
                    globalMatrix[i, 3 * load.NodeId + 2] = 0;
                }
                globalMatrix[3 * load.NodeId, 3 * load.NodeId] = 1;
                globalMatrix[3 * load.NodeId + 1, 3 * load.NodeId + 1] = 1;
                globalMatrix[3 * load.NodeId + 2, 3 * load.NodeId + 2] = 1;
            }
        }

        return globalMatrix;
    }

    public object Clone()
    {
        return new FiniteElementModel(
            Elements.Select(e => (Element)e.Clone()).ToList(),
            Nodes.Select(n => (Node)n.Clone()).ToList(),
            NodeLoads.Select(l => (NodeLoad)l.Clone()).ToList());
    }
  }
}

namespace CourseWork.BLL.Models
{
  public enum LoadType
  {
    Fixed,
    Force
  }
}
```

Листинг класса *Node*

```
using CourseWork.Common.Models;
using System;

namespace CourseWork.BLL.Models
{
  public class Node : ICloneable
  {
    public int Id { get; set; }

    public Point Position { get; set; }
```

```
        public object Clone()
        {
            return new Node
            {
                Id = Id,
                Position = (Point)Position.Clone(),
            };
        }

        public override string ToString()
        {
            return Position.ToString();
        }
    }
}
```

Листинг класса *NodeLoad*

```
using System;

namespace CourseWork.BLL.Models
{
    public class NodeLoad : ICloneable
    {
        public LoadType LoadType { get; set; }

        public int NodeId { get; set; }

        public double X { get; set; }

        public double Y { get; set; }

        public double Z { get; set; }

        public object Clone()
        {
            return new NodeLoad
            {
                LoadType = LoadType,
                NodeId = NodeId,
                X = X,
                Y = Y,
                Z = Z,
            };
        }
    }
}
```

Листинг класса *GridSerializer*

```
using CourseWork.BLL.Interfaces;
using Newtonsoft.Json;
using System.IO;
using CourseWork.BLL.Models.GridSettings;

namespace CourseWork.BLL.Services.ElementLoaders.Json
{
    public class GridSerializer : IGridSerializer
    {
        public void Serialize(GridSettings grid, string path)
        {
            var json = JsonConvert.SerializeObject(
```

```
            grid,
            new JsonSerializerSettings { TypeNameHandling = TypeNameHandling.Objects, Serializa-
tionBinder = new GridSettingsBinder() });
        using var writer = new StreamWriter(path);
        writer.Write(json);
    }

    public GridSettings Deserialize(string path)
    {
        using var reader = new StreamReader(path);
        var json = reader.ReadToEnd();
        return JsonConvert.DeserializeObject<GridSettings>(
            json,
            new JsonSerializerSettings {
                TypeNameHandling = TypeNameHandling.Objects,
                SerializationBinder = new GridSettingsBinder()
            });
    }
  }
}
```

Листинг класса *GridSettingsBinder*

```
using CourseWork.Common.Models;
using CourseWork.BLL.Models.GridSettings;
using Newtonsoft.Json.Serialization;
using System;
using System.Collections.Generic;
using System.Linq;

namespace CourseWork.BLL.Services.ElementLoaders.Json
{
  public class GridSettingsBinder : ISerializationBinder
  {
    public List<Type> KnownTypes { get; set; }

    public GridSettingsBinder()
    {
        KnownTypes = GetKnownTypes();
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        return KnownTypes.SingleOrDefault(t => t.Name == typeName);
    }

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        assemblyName = null;
        typeName = serializedType.Name;
    }

    private List<Type> GetKnownTypes()
    {
        var figureType = typeof(IFigure);
        var types = figureType.Assembly
            .GetTypes()
            .Where(t => figureType.IsAssignableFrom(t) && t.IsClass && !t.IsAbstract)
            .ToList();
        types.Add(typeof(GridSettings));
        types.Add(typeof(Point));
```

```
            return types;
        }
    }
}
```

Листинг класса *JsonElementsProvider*

```
using CourseWork.Common.Models;
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.Models;
using System.Collections.Generic;
using System.Linq;

namespace CourseWork.BLL.Services.ElementLoaders.Json
{
    /// <summary>
    /// Gets grid settings and creates elements by using them.
    /// </summary>
    public class JsonElementsProvider : IElementsProvider
    {
        private readonly string _settingsPath;
        private readonly IGridSerializer _gridSerializer;
        private readonly ILoadsProvider _loadsProvider;
        private readonly double _step;

        public JsonElementsProvider(string settingsPath, IGridSerializer figureSerializer, ILoadsProvider loadsPro-
vider = default,double step = 3)
        {
            _settingsPath = settingsPath;
            _gridSerializer = figureSerializer;
            _loadsProvider = loadsProvider;
            _step = step;
        }

        public FiniteElementModel GetModel()
        {
            var gridSettings = _gridSerializer.Deserialize(_settingsPath);
            var matrix = new int[gridSettings.Lenght*10, gridSettings.Width*10, gridSettings.Height*10];
            foreach (var figure in gridSettings.Figures)
            {
                figure.Draw(matrix, _step);
            }
            var (cubes, nodes) = GetCubes(matrix);
            for (var i = 0; i < nodes.Count; i++)
            {
                nodes[i].Id = i;
            }

            var elements = cubes.SelectMany(c => c.ToElements()).ToList();
            var nodeLoads = _loadsProvider != null ? _loadsProvider.GetNodeLoads() : new List<NodeLoad>();
            nodes.ForEach(n => n.Position = new Point(n.Position.X/100, n.Position.Y/100, n.Position.Z/100));
            return new FiniteElementModel(elements, nodes, nodeLoads);
        }

        /// <summary>
        /// Gets cubes for the specified matrix.
        /// </summary>
        /// <param name="matrix">Matrix filled by 0 and 1.</param>
        /// <returns>List of cubes.</returns>
        private (List<Cube>, List<Node> nodes) GetCubes(int[,,] matrix)
        {
```

```
            var cubes = new List<Cube>();
            var nodes = new List<Node>();
            for (var z = 0; z < matrix.GetLength(2)-1; z++)
            {
               for (var y = 0; y < matrix.GetLength(1)-1; y++)
               {
                  for (var x = 0; x < matrix.GetLength(0)-1; x++)
                  {
                     var nodePositions = GetNodePositions(x, y, z);
                     var cubeNodes = nodePositions.Select(p => matrix[(int)p.X, (int)p.Y, (int)p.Z]);
                     if (cubeNodes.All(n => n == 1))
                     {
                        cubes.Add(new Cube
                        {
                           Nodes = nodePositions.Select(p =>
                           {
                              var existingNode = nodes.FirstOrDefault(n => n.Position.X == p.X && n.Posi-
tion.Y == p.Y && n.Position.Z == p.Z);
                              if(existingNode == null)
                              {
                                 existingNode = new Node { Position = p };
                                 nodes.Add(existingNode);
                              }

                              return existingNode;
                           }).ToList()
                        });
                     };
                  }
               }
            }

            return (cubes, nodes);
         }

         /// <summary>
         /// Gets node positions for the Cube.
         /// </summary>
         /// <param name="x">X.</param>
         /// <param name="y">Y.</param>
         /// <param name="z">Z.</param>
         /// <returns>List of node positions.</returns>
         private List<Point> GetNodePositions(int x,int y,int z)
         {
            return new List<Point>
            {
               new Point(x, y, z),
               new Point(x+1, y, z),
               new Point(x + 1, y, z+1),
               new Point(x, y, z+1),
               new Point(x, y+1, z),
               new Point(x+1, y+1, z),
               new Point(x + 1, y+1, z+1),
               new Point(x, y+1, z+1),
            };
         }
      }
   }
```

Листинг абстрактного класса *BaseTxtProvider*

```
using System;
```

```
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

namespace CourseWork.BLL.Services.ElementLoaders.Txt
{
    public abstract class BaseTxtProvider
    {
        /// <summary>
        /// Loads values from txt file with ability to skip first info row.
        /// </summary>
        /// <param name="path">File path.</param>
        /// <param name="skipRows">Rows to skip.</param>
        /// <returns>List of rows values.</returns>
        protected List<string[]> GetValuesFromTextFile(string path, string separator = "\t", int skipRows = 1)
        {
            using var reader = new StreamReader(path);
            var text = reader.ReadToEnd();
            var currentSeparator = CultureInfo.CurrentCulture.NumberFormat.NumberDecimalSeparator;
            var regex = new Regex(@"\.|,");
            text = regex.Replace(text, currentSeparator);
            text = text.Replace("\r\n", "\n");
            var rows = text.Split("\n").Skip(skipRows).Where(row => !string.IsNullOrEmpty(row));
            return rows.Select(r => r.Split(separator, StringSplitOptions.RemoveEmptyEntries)).ToList();
        }
    }
}
```

Листинг класса *TxtElementsProvider*

```
using CourseWork.Common.Models;
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace CourseWork.BLL.Services.ElementLoaders.Txt
{
    /// <summary>
    /// Gets elements from txt(ANSYS format) file.
    /// </summary>
    public class TxtElementsProvider : BaseTxtProvider, IElementsProvider
    {
        private const string Separator = "\t";
        private readonly string _elementsPath;
        private readonly string _nodesPath;
        private readonly ILoadsProvider _loadsProvider;

        public TxtElementsProvider(string elementsPath, string nodesPath, ILoadsProvider loadsProvider = default)
        {
            _elementsPath = elementsPath;
            _nodesPath = nodesPath;
            _loadsProvider = loadsProvider;
        }

        public FiniteElementModel GetModel()
        {
            var elements = new List<Element>();
            var nodes = GetNodes();
```

```
        var nodeLoads = _loadsProvider != null ? _loadsProvider.GetNodeLoads() : new List<NodeLoad>();
        var elementRows = GetValuesFromTextFile(_elementsPath, Separator, 1).ToList();
        if (elementRows.Any())
        {
            if (elementRows[0].Length < 3)
            {
                throw new InvalidOperationException($"Element row "{string.Join(" ", elemen-
tRows[0])}' has no node ids.");
            }

            if (Enum.TryParse<ElementType>(elementRows[0][1], true, out var elementType))
            {
                var rowValuesCount = 2 + elementType.GetNodeCount();
                if (elementRows.Any(r => r.Length != rowValuesCount))
                {
                    throw new InvalidOperationException($"Each element row of type '{ele-
mentType}' should have '{rowValuesCount}' values.");
                }
                foreach (var row in elementRows)
                {
                    if (!int.TryParse(row[0], out var id))
                    {
                        throw new InvalidOperationException($"Couldn't parse element id '{row[0]}'.");
                    }
                    var elementNodes = GetElementNodes(row.Skip(2), nodes).ToList();
                    elements.Add(new Element
                    {
                        Id = id-1,
                        ElementType = elementType,
                        Nodes = elementNodes,
                    });
                }
            }
            else
            {
                throw new InvalidOperationException($"Found unknown element type {elementRows[0][1]}.");
            }
        }

        return new FiniteElementModel(elements, nodes, nodeLoads);
    }

    /// <summary>
    /// Loads all nodes from path.
    /// </summary>
    /// <returns>Nodes list.</returns>
    private List<Node> GetNodes()
    {
        var nodes = new List<Node>();
        var rows = GetValuesFromTextFile(_nodesPath, Separator, 1);
        foreach (var row in rows)
        {
            if (row.Length != 4)
            {
                throw new InvalidOperationException($"Invalid format of node '{string.Join(" ",row)}");
            }
            if (int.TryParse(row[0], out var id)
                && double.TryParse(row[1], out var x)
                && double.TryParse(row[2], out var y)
                && double.TryParse(row[3], out var z))
            {
                nodes.Add(new Node
```

```
                {
                    Id = id-1,
                    Position = new Point(x, y, z),
                });
            }
            else
            {
                throw new InvalidOperationException($"Couldn't parse node values '{string.Join(" ", row)}'.");
            }
        }

        return nodes;
    }

    /// <summary>
    /// Gets nodes for element by string ids.
    /// </summary>
    /// <param name="nodeIds">List of node ids.</param>
    /// <param name="nodes">All nodes.</param>
    /// <returns>Element nodes.</returns>
    private List<Node> GetElementNodes(IEnumerable<string> nodeIds, List<Node> nodes)
    {
        return nodeIds.Select(stringNodeId =>
            {
                if (!int.TryParse(stringNodeId, out var nodeId))
                {
                    throw new InvalidOperationException($"Couldn't parse node id '{stringNodeId}'.");
                }

                return nodeId;
            })
            .Select(nodeId =>
            {
                var node = nodes.FirstOrDefault(n => n.Id == nodeId-1);
                if (node is null)
                {
                    throw new InvalidOperationException($"Node with id '{nodeId}' was not found.");
                }

                return node;
            }).ToList();
    }
  }
}
```

Листинг класса *TxtLoadsProvider*

```
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.Models;
using System;
using System.Collections.Generic;

namespace CourseWork.BLL.Services.ElementLoaders.Txt
{
  public class TxtLoadsProvider : BaseTxtProvider, ILoadsProvider
  {
    private const string Separator = "\t";
    private readonly string _nodeLoadsPath;

    public TxtLoadsProvider(string nodeLoadsPath)
    {
      _nodeLoadsPath = nodeLoadsPath;
    }
```

```
        public List<NodeLoad> GetNodeLoads()
        {
            var loads = new List<NodeLoad>();
            var rows = GetValuesFromTextFile(_nodeLoadsPath, Separator, 1);
            foreach (var row in rows)
            {
                if (row.Length != 5)
                {
                    throw new InvalidOperationException($"Invalid format of node load '{string.Join(" ", row)}");
                }
                if (int.TryParse(row[0], out var nodeId)
                    && Enum.TryParse<LoadType>(row[1], true, out var loadType)
                    && double.TryParse(row[2], out var x)
                    && double.TryParse(row[3], out var y)
                    && double.TryParse(row[4], out var z))
                {
                    loads.Add(new NodeLoad
                    {
                        NodeId = nodeId-1,
                        LoadType = loadType,
                        X = x,
                        Y = y,
                        Z = z,
                    });
                }
                else
                {
                    throw new InvalidOperationException($"Couldn't parse node load values '{string.Join(" ", row)}'.");
                }
            }

            return loads;
        }
    }
}
```

Листинг класса *TxtMaterialsProvider*

```
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Media;

namespace CourseWork.BLL.Services.ElementLoaders.Txt
{
    public class TxtMaterialsProvider : BaseTxtProvider, IMaterialsProvider
    {
        private const string Separator = "\t";
        private readonly string _nodeLoadsPath;

        public TxtMaterialsProvider(string nodeLoadsPath)
        {
            _nodeLoadsPath = nodeLoadsPath;
        }

        public List<DetailMaterial> GetMaterials()
        {
            var materials = new List<DetailMaterial>();
            var rows = GetValuesFromTextFile(_nodeLoadsPath, Separator, 1);
```

```
            foreach (var row in rows)
            {
                if (row.Length < 6)
                {
                    throw new InvalidOperationException($"Некорректный формат материала '{string.Join(" ", row)}.");
                }
                if (!string.IsNullOrEmpty(row[0])
                    && double.TryParse(row[1], out var youngModulus)
                    && double.TryParse(row[2], out var poissonCoefficient))
                {
                    materials.Add(new DetailMaterial(row[0], youngModulus, poissonCoeffi-
cient, GetColor(row.Skip(3).ToArray())));
                }
                else
                {
                    throw new InvalidOperationException($"Не удалось загрузить материал из '{string.Join(" ", row)}'.");
                }
            }

            return materials;
        }

        private Color GetColor(params string[] rowValues)
        {
            var components = new List<byte>();
            foreach(var value in rowValues)
            {
                if(!byte.TryParse(value, out var component))
                {
                    throw new InvalidOperationException($"Не удалось преобразовать компонетнт цвета '{value}'.");
                }

                components.Add(component);
            }

            return components.Count switch
            {
                3 => Color.FromRgb(components[0], components[1], components[2]),
                4 => Color.FromArgb(components[0], components[1], components[2], components[3]),
                _ => throw new InvalidOperationException($"Цвет не может быть преобразован из '{compo-
nents.Count}' компонетнов")
            };
        }
    }
}
```

Листинг класса *SLAESolver*

```
using System;

namespace CourseWork.BLL.Services
{
    public class SLAESolver
    {
        /// <summary>
        /// Метод, решающий систему линейных уравнений методом Гаусса
        /// </summary>
        /// <param name="matrix">Матрица коэффициентов</param>
        /// <param name="answers">Матрица ответов></param>
        /// <returns>Решение СЛАУ</returns>
        public double[] Gauss(double[,] matrix, double[] answers)
        {
```

```
    (int n, int m) = (matrix.GetLength(0), matrix.GetLength(1));

    //Создание расширенной матрицы
    double[,] extMatrix = GetExtendedMatrix(matrix, answers);

    //Превращение расширенной матрицы в треугольную
    double M = 0;
    for (int j = 0; j < m + 1; j++)
    {
        for (int i = j + 1; i < n; i++)
        {
            M = extMatrix[i, j] / extMatrix[j, j];
            for (int k = 0; k < m + 1; k++)
            {
                extMatrix[i, k] -= M * extMatrix[j, k];
            }
        }
    }

    //Обратное разбиение матрицы
    (double[,] triangleMatrix, double[] correctedAnswers) = ExpandMatrix(extMatrix);

    //Проверка на ранг матрицы
    if (triangleMatrix[n - 1, m - 1] == 0)
    {
        throw new Exception("Нет решения");
    }

    double[] solution = new double[n];

    solution[n - 1] = correctedAnswers[n - 1] / triangleMatrix[n - 1, m - 1];

    //Решение треугольной матрицы
    double dif;
    for (int i = 1; i < n; i++)
    {
        dif = 0;
        for (int j = 0; j < i; j++)
        {
            dif += triangleMatrix[n - i - 1, n - j - 1] * solution[n - 1 - j];
        }
        solution[n - 1 - i] = (correctedAnswers[n - i - 1] - dif) / triangleMatrix[n - i - 1, n - i - 1];
    }

    return solution;
}

/// <summary>
/// Делает расширенную матрицу из матрицы и вектора той же размерности
/// </summary>
/// <param name="matrix">Матрица</param>
/// <param name="vector">Вектор той же размерности</param>
/// <returns>Расширенную матрицу</returns>
public double[,] GetExtendedMatrix(double[,] matrix, double[] vector)
{
    if (matrix.GetLength(1) != vector.GetLength(0))
    {
        throw new Exception("Количество строк матрицы должно быть равно количество элементов вектора.");
    }

    (int n, int m) = (matrix.GetLength(0), matrix.GetLength(1));
    double[,] extMatrix = new double[n, m + 1];
```

```
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    extMatrix[i, j] = matrix[i, j];
                }
                extMatrix[i, m] = vector[i];
            }

            return extMatrix;
        }

        /// <summary>
        /// Метод раскладывает матрицу на исходную матрицу без правого столбца и правый столбец исход-
ной матрицы
        /// </summary>
        /// <param name="extMatrix">Расширенная матрица</param>
        /// <returns>Исходная матрица без правого столбца и правый столбец исходной матрицы</returns>
        public (double[,], double[]) ExpandMatrix(double[,] extMatrix)
        {
            (int n, int m) = (extMatrix.GetLength(0), extMatrix.GetLength(1));

            double[,] matrix = new double[n, m - 1];
            double[] vector = new double[n];

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m - 1; j++)
                {
                    matrix[i, j] = extMatrix[i, j];
                }
                vector[i] = extMatrix[i, m - 1];
            }

            return (matrix, vector);
        }
    }
}
```

Листинг класса *SolutionMaker*

```
using Accord.Math;
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.Models;
using MathNet.Numerics.LinearAlgebra.Double;
using MathNet.Numerics.LinearAlgebra.Double.Solvers;
using System;
using System.Collections.Generic;
using System.Linq;

namespace CourseWork.BLL.Services
{
    public class SolutionMaker : ISolutionMaker
    {
        public double[] GetNodeDisplacements(FiniteElementModel model, DetailMaterial material)
        {
            var globalStiffnessWithCondisplacements = model.GetGlobalStiffnessMatrix(material);

            var nodeForces = model.GetNodeForces();
            //var a = SparseMatrix.OfArray(globalStiffnessWithCondisplacements);
            //var b = DenseVector.OfArray(nodeForces);
```

```
            //var displacement = a.SolveIterative(b, new MlkBiCgStab()).ToArray();
            var displacement = new SLAESolver().Gauss(globalStiffnessWithCondisplacements, nodeForces);
            var zDisp = new List<double>();
            var dict = new Dictionary<int, double>();
            for(var i = 0; i< displacement.Length; i+=3)
            {
                zDisp.Add((displacement[i] + displacement[i + 1] + displacement[i + 2]) / 3);
            }
            zDisp.Sort();
            return displacement;
        }

        public double[] GetElementsDisplacement(FiniteElementModel model, double[] nodeDisplacements)
        {
            var displacements = new List<double>();
            foreach (var element in model.Elements)
            {
                double xDisplacement = 0, yDisplacement = 0, zDisplacement = 0;
                foreach (var node in element.Nodes)
                {
                    xDisplacement += nodeDisplacements[node.Id * 3];
                    yDisplacement += nodeDisplacements[node.Id * 3 + 1];
                    zDisplacement += nodeDisplacements[node.Id * 3 + 2];
                }
                xDisplacement /= 4;
                yDisplacement /= 4;
                zDisplacement /= 4;

                displacements.Add(Math.Sqrt(Math.Pow(xDisplacement, 2) + Math.Pow(yDisplace-
ment, 2) + Math.Pow(zDisplacement, 2)));
            }

            foreach(var node in model.Nodes)
            {
                node.Position.X += nodeDisplacements[node.Id * 3];
                node.Position.Y += nodeDisplacements[node.Id * 3 + 1];
                node.Position.Z += nodeDisplacements[node.Id * 3 + 2];
            }
            var s = displacements.ToArray();
            s.Sort();
            return s;
        }


        /// <summary>
        /// Вычисляет деформации конечных элементов?
        /// </summary>
        /// <param name="originalMesh">Исходная сетка</param>
        /// <param name="nodesDisplacement">Перемещения всех узлов сетки</param>
        /// <returns></returns>
        public double[] GetElementsStrain(FiniteElementModel originalMesh, double[] nodesDisplacement)
        {
            IList<double> strain = new List<double>();

            foreach (var element in originalMesh.Elements)
            {
                IList<double> curElementNodeDisplacement = new List<double>();
                foreach (var node in element.Nodes)
                {
                    curElementNodeDisplacement.Add(nodesDisplacement[node.Id * 3]);
                    curElementNodeDisplacement.Add(nodesDisplacement[node.Id * 3 + 1]);
                    curElementNodeDisplacement.Add(nodesDisplacement[node.Id * 3 + 2]);
```

```
            }
            double[] currentDisplacement = curElementNodeDisplacement.ToArray();

            double[] elementStrains = element.GeometricMatrix.Dot(currentDisplacement);

            strain.Add(Math.Sqrt(2) / 3 * Math.Sqrt(Math.Pow(elementStrains[0] - elementStrains[1], 2) + Math.Pow(elementStrains[1] - elementStrains[2], 2) + Math.Pow(elementStrains[0] - elementStrains[2], 2) + 3/2 * (Math.Pow(elementStrains[3], 2) + Math.Pow(elementStrains[4], 2) + Math.Pow(elementStrains[5], 2))));
        }
        var s = strain.ToArray();
        s.Sort();
        return s;
    }

    /// <summary>
    /// Вычисляет напряжения конечных элементов
    /// </summary>
    /// <param name="originalMesh">Исходная сетка</param>
    /// <param name="nodesDisplacement">Перемещения всех узлов сетки</param>
    /// <param name="material">Материал детали</param>
    /// <returns></returns>
    public double[] GetElementsStresses(FiniteElementModel originalMesh, double[] nodesDisplacement, DetailMaterial material)
    {
        IList<double> stresses = new List<double>();

        foreach (var element in originalMesh.Elements)
        {
            IList<double> curElementNodeDisplacement = new List<double>();
            foreach (var node in element.Nodes)
            {
                curElementNodeDisplacement.Add(nodesDisplacement[node.Id * 3]);
                curElementNodeDisplacement.Add(nodesDisplacement[node.Id * 3 + 1]);
                curElementNodeDisplacement.Add(nodesDisplacement[node.Id * 3 + 2]);
            }
            double[] currentDisplacement = curElementNodeDisplacement.ToArray();

            double[] elementStresses = element.getElastic(material).Dot(element.GeometricMatrix.Dot(currentDisplacement));

            stresses.Add(1/Math.Sqrt(2) * Math.Sqrt(Math.Pow(elementStresses[0] - elementStresses[1],2) + Math.Pow(elementStresses[1] - elementStresses[2], 2) + Math.Pow(elementStresses[0] - elementStresses[2], 2) + 6* (Math.Pow(elementStresses[3],2) + Math.Pow(elementStresses[4], 2) + Math.Pow(elementStresses[5], 2))));
        }

        var s = stresses.ToArray();
        s.Sort();
        return s;
    }
  }
}

using System;


namespace CourseWork.BLL.Tools
{
  /// <summary>
  /// Класс для создания градиента
  /// </summary>
  public class GradientMaker
```

```
    {
        /// <summary>
        /// Возвращает RGB значения определенного значения в диапазоне значений градиента
        /// </summary>
        /// <param name="minValue">Минимальное значение градиента</param>
        /// <param name="maxValue">Максимальное значение градиента</param>
        /// <param name="value">Текущее значение градиента</param>
        /// <returns>Цвет RGB, отражающий собой цвет данного значения в диапазоне значений градиента</returns>
        public (byte, byte, byte) GetRGBGradientValue(double minValue, double maxValue, double value)
        {
            double ratio = 2 * (value - minValue) / (maxValue - minValue);
            byte B = (byte)Math.Max(0, 255 * (1 - ratio));
            byte R = (byte)Math.Max(0, 255 * (ratio - 1));
            byte G = (byte)(255 - B - R);
            return (R, G, B);
        }
    }
}
```

Листинг класса *GradientScale*

```
using System;
using System.Collections.Generic;
using System.Windows.Media;
using CourseWork.BLL.Tools;

namespace CourseWork.BLL.UIElements
{
    /// <summary>
    /// Градиентная шкала
    /// </summary>
    public class GradientScale
    {
        /// <summary>
        /// Создание градиентной шкалы
        /// </summary>
        /// <param name="gradientMaker">Класс, создающий градиент</param>
        /// <param name="minValue">Минимальное значение шкалы</param>
        /// <param name="maxValue">Максимальное значение шкалы</param>
        /// <param name="stepCount">Количество шагов градиента</param>
        public GradientScale(GradientMaker gradientMaker, double minValue, double maxValue, double minValueUnder, double maxValueUnder, int stepCount)
        {
            MinValue = minValue;
            MaxValue = maxValue;
            MinValueUnder = minValueUnder;
            MaxValueUnder = maxValueUnder;

            double step = (maxValue - minValue) / stepCount;
            IList<GradientStop> gradientStops = new List<GradientStop>();
            double currentValue = minValue;
            while (currentValue < maxValue)
            {
                double currentValueNormalized = (currentValue - minValue) / (maxValue - minValue);
                (byte R, byte G, byte B) = gradientMaker.GetRGBGradientValue(minValue, maxValue, currentValue);
                var stop = new GradientStop(Color.FromArgb(0x99, R, G, B), 1 - currentValueNormalized);
                gradientStops.Add(stop);
                currentValue += step;
            }

            Brush = new LinearGradientBrush();
```

```
            Brush.GradientStops = new GradientStopCollection(gradientStops);
        }

        public LinearGradientBrush Brush { get; set; }
        public double MaxValue { get; set; }
        public double MinValue { get; set; }
        public double MaxValueUnder { get; set; }
        public double MinValueUnder { get; set; }

        public string MaxValueText
        {
            get
            {
                if (MaxValue > 10000 || MaxValue < 1e-4)
                {
                    return MaxValue.ToString("e3");
                }
                return $"{Math.Round(MaxValue, 4)}";
            }
        }
        public string MinValueText
        {
            get
            {
                if (MinValue > 10000 || MinValue < 1e-4)
                {
                    return MinValue.ToString("e3");
                }
                return $"{Math.Round(MinValue, 4)}";
            }
        }

        public string MaxValueTextUnder
        {
            get
            {
                if (MaxValueUnder > 10000 || MaxValueUnder < 1e-4)
                {
                    return MaxValueUnder.ToString("e3");
                }
                return $"{Math.Round(MaxValueUnder, 4)}";
            }
        }
        public string MinValueTextUnder
        {
            get
            {
                if (MinValueUnder > 10000 || MinValueUnder < 1e-4)
                {
                    return MinValueUnder.ToString("e3");
                }
                return $"{Math.Round(MinValueUnder, 4)}";
            }
        }
    }
}
```

Листинг класса *Point*

```
using System;

namespace CourseWork.Common.Models
{
```

```
    public class Point : ICloneable
    {
        public Point(double x,double y,double z)
        {
            X = x;
            Y = y;
            Z = z;
        }

        public Point()
        {
        }
        public double X { get; set; }

        public double Y { get; set; }

        public double Z { get; set; }

        public object Clone()
        {
            return new Point(X, Y, Z);
        }

        public override string ToString()
        {
            return $" {X} {Y} {Z}; ";
        }
    }
}
```

Листинг класса *MainViewModel*

```
using Accord.Math;
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.ModelMakers;
using CourseWork.BLL.Models;
using CourseWork.BLL.UIElements;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Media;
using System.Windows.Media.Media3D;

namespace CourseWork.UI
{
    public class MainViewModel
    {
        private readonly ISolutionMaker _solutionMaker;
        private readonly IMeshModelMaker _geometryMaker;
        public MainViewModel(IElementsProvider elementsProvider, IMaterialsProvider materialsPro-
vider, ISolutionMaker solutionMaker, IMeshModelMaker geometryMaker, DetailMaterial material = default)
        {
            OriginalModel = elementsProvider.GetModel();
            Materials = materialsProvider.GetMaterials();
            _solutionMaker = solutionMaker;
            _geometryMaker = geometryMaker;
            if(Materials.Count == 0)
            {
                Materials.Add(new DetailMaterial("Steel", 200000000000, 0.3, Color.FromArgb(0x99, 0x53, 0xd2, 0xf5)));
            }
            CurrentMaterial = Materials.First();
        }
```

```
        public string CurrentFileType { get; set; }

        public FiniteElementModel OriginalModel { get; set; }

        public FiniteElementModel CurrentModel { get; set; }

        public Model3DGroup OriginalModelGeometry
        {
            get
            {
                return _geometryMaker.GenerateGeometry(OriginalModel, CurrentMaterialGroup);
            }
        }

        public Model3DGroup DisplacedModelGeometry
        {
            get
            {
                CurrentModel = (FiniteElementModel)OriginalModel.Clone();
                for(var i = 0; i < CurrentModel.Nodes.Count; i++)
                {
                    CurrentModel.Nodes[i].Position.X += SolutionNodesDisplacement[i * 3];
                    CurrentModel.Nodes[i].Position.Y += SolutionNodesDisplacement[i * 3 + 1];
                    CurrentModel.Nodes[i].Position.Z += SolutionNodesDisplacement[i * 3 + 2];
                }

                SolutionElementsDisplacement = _solutionMaker.GetElementsDisplacement(CurrentModel, Solution-
NodesDisplacement);
                return _geometryMaker.GenerateCharacteristicModel(CurrentModel, SolutionElementsDisplacement);
            }
        }

        private Transform3DGroup _transforms;
        public Transform3DGroup Transforms
        {
            get
            {
                if (_transforms == null)
                {
                    const double scale = 40;
                    _transforms = new Transform3DGroup();
                    _transforms.Children.Add(new RotateTransform3D(new AxisAngleRotation3D(new Vec-
tor3D(0, 1, 0), 30)));
                    _transforms.Children.Add(new RotateTransform3D(new AxisAngleRotation3D(new Vec-
tor3D(1, 0, 0), 30)));
                    _transforms.Children.Add(new ScaleTransform3D(scale, scale, scale, 0, 0, 0));
                }
                return _transforms;
            }
        }

        public void Scale(bool upscale = true)
        {
            if (upscale)
            {
                ((ScaleTransform3D)(Transforms).Children[2]).ScaleX *= 1.2;
                ((ScaleTransform3D)(Transforms).Children[2]).ScaleY *= 1.2;
                ((ScaleTransform3D)(Transforms).Children[2]).ScaleZ *= 1.2;
            }
            else
            {
```

```
        ((ScaleTransform3D)(Transforms).Children[2]).ScaleX *= 0.8;
        ((ScaleTransform3D)(Transforms).Children[2]).ScaleY *= 0.8;
        ((ScaleTransform3D)(Transforms).Children[2]).ScaleZ *= 0.8;
    }
}

public void Rotate(double xRotate, double yRotate)
{
    ((AxisAngleRotation3D)((RotateTransform3D)(Transforms).Children[0]).Rotation).Angle += xRotate;
    ((AxisAngleRotation3D)((RotateTransform3D)(Transforms).Children[1]).Rotation).Angle += yRotate;
}

public IList<DetailMaterial> Materials { get; }

public DetailMaterial CurrentMaterial { get; set; }

public MaterialGroup CurrentMaterialGroup
{
    get
    {
        MaterialGroup geometryMaterial = new MaterialGroup();
        geometryMaterial.Children = new MaterialCollection();
        geometryMaterial.Children.Add(new DiffuseMaterial(new SolidColorBrush(CurrentMaterial.Color)));

        return geometryMaterial;
    }
}


public FiniteElementModel SolutionMesh { get; set; }
public Model3DGroup SolutionMeshModel
{
    get
    {
        return _geometryMaker.GenerateGeometry(SolutionMesh, null);
    }
}
public double[] SolutionNodesDisplacement { get; set; }

public void Solve()
{
    SolutionNodesDisplacement = _solutionMaker.GetNodeDisplacements(OriginalModel, CurrentMaterial);
    CurrentModel = (FiniteElementModel)OriginalModel.Clone();

}

public double[] SolutionElementsDisplacement { get; set; }



public GradientScale DisplacementGradientScale
{
    get
    {
        return new GradientScale
            (
            new BLL.Tools.GradientMaker(),
            SolutionElementsDisplacement.Min(),
            SolutionElementsDisplacement.Max(),
            SolutionElementsDisplacement[2],
            SolutionElementsDisplacement[SolutionElementsDisplacement.Length - 3],
            10
```

```
                );
            }
        }

        public double[] SolutionElementsStrain { get; set; }

        public Model3DGroup GetStrainMeshModel()
        {
            CurrentModel = (FiniteElementModel)OriginalModel.Clone();
            SolutionElementsStrain = _solutionMaker.GetElementsStrain(OriginalModel, SolutionNodesDisplacement);
            SolutionElementsStrain.Sort();
            return _geometryMaker.GenerateCharacteristicModel(CurrentModel, SolutionElementsStrain);
        }

        public GradientScale StrainGradientScale
        {
            get
            {
                return new GradientScale
                    (
                    new BLL.Tools.GradientMaker(),
                    SolutionElementsStrain.Min(),
                    SolutionElementsStrain.Max(),
                    SolutionElementsStrain[1],
                    SolutionElementsStrain[SolutionElementsStrain.Length - 2],
                    10
                    );
            }
        }

        public double[] SolutionElementsStresses { get; set; }

        public Model3DGroup GetStressMeshModel()
        {
            CurrentModel = (FiniteElementModel)OriginalModel.Clone();
            SolutionElementsStresses = _solutionMaker.GetElementsStresses(OriginalModel, SolutionNodesDisplace-
ment, CurrentMaterial);
            SolutionElementsStresses.Sort();
            return _geometryMaker.GenerateCharacteristicModel(CurrentModel, SolutionElementsStresses);
        }

        public GradientScale StressesGradientScale
        {
            get
            {
                return new GradientScale
                    (
                    new BLL.Tools.GradientMaker(),
                    SolutionElementsStresses.Min(),
                    SolutionElementsStresses.Max(),
                    SolutionElementsStresses[1],
                    SolutionElementsStresses[SolutionElementsStresses.Length-2],
                    10
                    );
            }
        }
    }
}
```

Листинг класса *MainWindow*

```
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Media3D;
using CourseWork.BLL.Interfaces;
using CourseWork.BLL.ModelMakers;
using CourseWork.BLL.Models;
using CourseWork.BLL.Models.GridSettings;
using CourseWork.BLL.Services;
using CourseWork.BLL.Services.ElementLoaders.Json;
using CourseWork.BLL.Services.ElementLoaders.Txt;
using CourseWork.BLL.Tools;
using CourseWork.BLL.UIElements;
using Microsoft.Win32;
using Point = CourseWork.Common.Models.Point;

namespace CourseWork.UI
{
    public partial class MainWindow : Window
    {
        private readonly MainViewModel _model;
        public MainWindow()
        {
            var serializer = new GridSerializer();
            var grid = new GridSettings
            {
                Lenght = 10,
                Width = 10,
                Height = 10,
                Figures = new List<IFigure>
                {
                    new Circle
                    {
                        Height = 5,
                        IsFill = true,
                        Radius = 32,
                        StartPoint = new Point { X = 40, Y = 40, Z = 0 }
                    },
                    new Circle
                    {
                        Height = 4,
                        IsFill = false,
                        Radius = 8,
                        StartPoint = new Point { X = 40, Y = 40, Z = 0 }
                    },
                    new Circle
                    {
                        Height = 3,
                        IsFill = false,
                        Radius = 10,
                        StartPoint = new Point { X = 40, Y = 40, Z = 2 }
                    },
                    new Circle
                    {
                        Height = 3,
                        IsFill = false,
                        Radius = 1,
                        StartPoint = new Point { X = 26, Y = 26, Z = 0 }
                    }
```

```
            }
        };
        serializer.Serialize(grid, @".\..\..\..\..\Data\elements.json");
        var loadsProvider = new TxtLoadsProvider(@".\..\..\..\..\Data\nodeLoads.txt");
        var elementsProvider = new TxtElementsProvider(@".\..\..\..\..\Data\ele-
mentsMetric.txt", @".\..\..\..\..\Data\nodesMetric.txt", loadsProvider);

        var materialProvider = new TxtMaterialsProvider(@".\..\..\..\..\Data\materials.txt");
        var solutionMaker = new SolutionMaker();
        _model = new MainViewModel(elementsProvider, materialProvider, solutionMaker, new Simple-
ModelMaker(new GradientMaker()));

        var model = elementsProvider.GetModel();
        var minY = model.Nodes.Min(n => n.Position.Y);
        var minNodes = model.Nodes.Where(n => n.Position.Y == minY).ToList();
        var color = (Color)ColorConverter.ConvertFromString("Red");

        InitializeComponent();
        InitializeGeometryModel();
        MaterialsBox.SelectionChanged -= MaterialsBox_SelectionChanged;
        MaterialsBox.ItemsSource = _model.Materials;
        MaterialsBox.SelectedIndex = 0;
        MaterialsBox.SelectionChanged += MaterialsBox_SelectionChanged;
    }


    private void InitializeGeometryModel()
    {
        try
        {
            MeshModel.Content = _model.OriginalModelGeometry;
            MeshModel.Transform = _model.Transforms;
            HideGradientScale();
        }
        catch (Exception exception)
        {
            MessageBox.Show($"Ошибка инициализации геометрии. {exception.Message}");
        }
    }


    private void Grid_PreviewMouseWheel(object sender, MouseWheelEventArgs e)
    {

        try
        {
            _model.Scale(e.Delta > 0);
        }
        catch (Exception exception)
        {
            MessageBox.Show($"Ошибка масштабирования. {exception.Message}");
        }
    }

    private double startXPos;
    private double startYPos;
    private void Viewport_PreviewMouseMove(object sender, MouseEventArgs e)
    {

    }
```

```
        private void MaterialsBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            try
            {
                _model.CurrentMaterial = (DetailMaterial)MaterialsBox.SelectedItem;
                MeshModel.Content = _model.OriginalModelGeometry;
                RefreshSolution();
                HideGradientScale();
                MessageBox.Show($"Материал успешно изменён");
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка изменения выбранного материала. {exception.Message}");
            }
        }

        private void MaterialInfoButton_MouseEnter(object sender, MouseEventArgs e)
        {
            try
            {
                ((ToolTip)MaterialInfoButton.ToolTip).Content = _model.CurrentMaterial.GetInfo();
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка просмотра информации о материале. {exception.Message}");
            }
        }

        private void RefreshSolution(bool isSolved = false)
        {
            if (!isSolved)
            {
                _model.CurrentModel = null;
            }
            MakeSolutionButton.Background = isSolved ? Brushes.LightGreen : (SolidColorBrush)new BrushCon-
verter().ConvertFrom("#ffa899");
        }

        private void HideGradientScale()
        {
            GradientScaleGrid.Visibility = Visibility.Hidden;
        }

        private void ShowGradientScale(GradientScale gradientScale)
        {
            GradientScaleGrid.Visibility = Visibility.Visible;
            GradientScaleRectangle.Fill = gradientScale.Brush;
            GradientScaleMaxValue.Text = gradientScale.MaxValueText;
            GradientScaleMinValue.Text = gradientScale.MinValueText;
            GradientScaleMaxValueUnder.Text = gradientScale.MaxValueTextUnder;
            GradientScaleMinValueUnder.Text = gradientScale.MinValueTextUnder;
        }

        private void ShowStressButton_Click(object sender, RoutedEventArgs e)
        {
            if (_model.CurrentModel == null)
            {
                MessageBox.Show("Небходимо произвести решение.");
                return;
            }
            try
            {
```

```
            MeshModel.Content = _model.GetStressMeshModel();
            ShowGradientScale(_model.StressesGradientScale);
        }
        catch (Exception exception)
        {
            MessageBox.Show($"Ошибка показа перемещения детали. Возможно вы забыли произвести реше-
ние. {exception.Message}");
        }

    }

    private void MakeSolutionButton_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            var startTime = DateTime.Now;
            _model.Solve();
            RefreshSolution(true);
            Message-
Box.Show($"Решение успешно завершено. Время, затраченное на решение {(DateTime.Now - startTime).Total-
Seconds} с.");
        }
        catch (Exception exception)
        {
            MessageBox.Show($"Ошибка решения. {exception.Message}");
        }
    }

    private void PressureApplyButton_Click(object sender, RoutedEventArgs e)
    {

        try
        {

            HideGradientScale();
            RefreshSolution();
        }
        catch (Exception exception)
        {
            MessageBox.Show($"Ошибка применения сил к детали. {exception.Message}");
        }
    }

    private void ShowOriginalDetailButton_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            MeshModel.Content = _model.OriginalModelGeometry;
            HideGradientScale();
        }
        catch (Exception exception)
        {
            MessageBox.Show($"Ошибка показа исходной детали. {exception.Message}");
        }
    }

    private void ShowDisplacementButton_Click(object sender, RoutedEventArgs e)
    {
        if (_model.CurrentModel == null)
        {
            MessageBox.Show("Небходимо произвести решение.");
            return;
```

```
            }
            try
            {
                MeshModel.Content = _model.DisplacedModelGeometry;
                MeshModel.Transform = _model.Transforms;
                ShowGradientScale(_model.DisplacementGradientScale);
            }
            catch (Exception exception)
            {
                Message-
Box.Show($"Ошибка показа перемещений детали. Возможно вы забыли произвести решение. {exception.Mes-
sage}");
            }
        }

        private void ShowStrainButton_Click(object sender, RoutedEventArgs e)
        {
            if (_model.CurrentModel == null)
            {
                MessageBox.Show("Небходимо произвести решение.");
                return;
            }
            try
            {
                MeshModel.Content = _model.GetStrainMeshModel();
                ShowGradientScale(_model.StrainGradientScale);
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка показа деформаций детали. Возможно вы забыли произвести реше-
ние. {exception.Message}");
            }

        }

        private void FileTypeBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            _model.CurrentFileType = (FileTypeBox.SelectedItem as ComboBoxItem)?.Content as string;
            switch (_model.CurrentFileType)
            {
                case "json":
                    NodesBox.Visibility = Visibility.Hidden;
                    StepBox.Visibility = Visibility.Visible;


                    break;
                case "txt":
                    NodesBox.Visibility = Visibility.Visible;
                    StepBox.Visibility = Visibility.Hidden;

                    break;
            }
        }

        private void ElementsBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
        {
            try
            {
                ElementsBox.Text = "";
                OpenFileDialog openFileDialog = new OpenFileDialog();
                openFileDialog.Filter = GetCurrentFileFilter();
                if (openFileDialog.ShowDialog() == true)
```

```
            {
                ElementsBox.Text = openFileDialog.FileName;
            }
        }
        catch (Exception)
        {

            MessageBox.Show("Ошибка. Выберите корректный файл\nс элементами!","Ошибка", MessageBoxBut-
ton.OK, MessageBoxImage.Error);
        }
    }

    private void NodesBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
    {
        try
        {
            NodesBox.Text = "";
            OpenFileDialog openFileDialog = new OpenFileDialog();

            openFileDialog.Filter = GetCurrentFileFilter();
            if (openFileDialog.ShowDialog() == true)
            {
                NodesBox.Text = openFileDialog.FileName;
            }
        }
        catch (Exception)
        {

            MessageBox.Show("Ошибка. Выберите корректный файл\nс узлами!", "Ошибка", MessageBoxBut-
ton.OK, MessageBoxImage.Error);
        }
    }

    private void LoadsBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
    {
        try
        {
            LoadsBox.Text = "";
            OpenFileDialog openFileDialog = new OpenFileDialog();

            openFileDialog.Filter = "Text files (*.txt)|*.txt";
            if (openFileDialog.ShowDialog() == true)
            {
                LoadsBox.Text = openFileDialog.FileName;
            }
            var loadsProvider = new TxtLoadsProvider(LoadsBox.Text);
            var loads = loadsProvider.GetNodeLoads();
            _model.OriginalModel.NodeLoads = loads;
        }
        catch (Exception)
        {

            MessageBox.Show("Ошибка. Выберите корректный файл\nс нагрузками!", "Ошибка", MessageBoxBut-
ton.OK, MessageBoxImage.Error);
        }
    }

    private string GetCurrentFileFilter()
    {
        return _model.CurrentFileType == "json" ? "Json files (*.json)|*.json" : "Text files (*.txt)|*.txt";
    }
```

```
        private void ModelApplyButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                IElementsProvider elementsProvider = _model.CurrentFileType switch
                {
                    "json" => new JsonElementsProvider(ElementsBox.Text, new GridSerializer(),default,dou-
ble.Parse(NodesBox.Text)),
                    "txt" => new TxtElementsProvider(ElementsBox.Text, NodesBox.Text),
                    _ => throw new NotSupportedException($"File type '{_model.CurrentFileType}' is not supported."),
                };
                var loads = _model.OriginalModel.NodeLoads;
                _model.OriginalModel = elementsProvider.GetModel();
                _model.OriginalModel.NodeLoads = loads;
                _model.CurrentModel = null;
                InitializeGeometryModel();
            }
            catch (Exception)
            {
                MessageBox.Show("Поверьте корректность введенных данных","Ошибка",MessageBoxButton.OK,Mes-
sageBoxImage.Error);
            }
        }

        private void Border_MouseMove(object sender, MouseEventArgs e)
        {
            try
            {

                if (e.LeftButton == MouseButtonState.Pressed)
                {
                    var xRotate = (e.GetPosition(Canvas).X - startXPos);
                    var yRotate = (e.GetPosition(Canvas).Y - startYPos);

                    _model.Rotate(xRotate, yRotate);
                }
                startXPos = e.GetPosition(Canvas).X;
                startYPos = e.GetPosition(Canvas).Y;
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка вращения. {exception.Message}");
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Б
(Обязательное)
Чертёж детали