СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При выполнении инженерных расчётов, связанных с анализом прочности конструкций, на практике используют численные методы, так как применение аналитических методов требует высокого уровня математического аппарата. Кроме того, как правило, аналитические расчёты позволяют получить решение задач для простых тел и для простой схемы нагруженности. В то же время применение численных методов, к которым относятся методы конечных разностей, конечных элементов, граничных элементов, не ограничено ни сложностью геометрии тела, ни способами приложения нагрузок.

Пакеты для математического моделирования – это гибкие, надёжные средства проектирования и анализа. Они работают в среде операционных систем самых распространённых компьютеров – от персональных компьютеров до рабочих станций и суперкомпьютеров, однако обладают серьёзным недостатком. Это дорогостоящие и многогранные, сложные продукты. Для внедрения их на узкоспециализированном малом производстве необходимы немалые средства.

Целью курсовой работы является моделирование и анализ напряжённо-деформированного состояния пространственного объекта – сплошного цилиндра и написание программного приложения на языке высокого уровня, выполняющего аналогичный расчёт конструкции методом конечных элементов. На сегодняшний день данная тема является важной, так как сплошные цилиндры являются часто встречающимися элементами во многих конструкциях. Различным инженерам, проектирующим конструкции, в которых есть цилиндры, необходимо знать оптимальные размеры цилиндра, какую нагрузку прилагать или какой материал использовать, чтобы конструкции не разрушались.

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

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 [9].

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

1.2 Метод конечных разностей

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 [8].

Для решения эллиптической задачи методом конечных разностей на расчётной области строится сетка, затем выбирается разностная схема и для каждого узла сетки записывается разностное уравнение (аналог исходного уравнения, но с использованием разностной схемы), затем производится учёт краевых условий (для краевых условий второго и третьего рода так же строится некоторая разностная схема). Получается система линейных алгебраических уравнений, решая которую в ответе получают приближенные значения решения в узлах.

Например, пусть дана эллиптическая задача:

$$\frac{d^2u}{dx^2} = f(x), x \in [0, 1] \quad (1.1)$$

Граничные условия:

$$\mathrm{u}(0) = \mathrm{u}(1) = 0 \quad (1.2)$$

Тогда строится сетка с постоянным шагом h. В данном случае шаг h равен 0,25. Для аппроксимации выбирается трёхточечный шаблон, то есть для аппроксимации исходной точки используются точки слева и справа от неё. Разностное уравнение будет выглядеть следующим образом:

$$\frac{-u_{i-1}+2u_i-u_{i+1}}{h^2} = f_i, i \in 1,2,3 \quad (1.3)$$

Учитывая граничные условия, компоненты системы линейных уравнений вида $Au=b$ принимают вид:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -16 & 32 & -16 & 0 & 0 \\ 0 & -16 & 32 & -16 & 0 \\ 0 & 0 & -16 & 32 & -16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ f(0,25) \\ f(0,5) \\ f(0,75) \\ 0 \end{pmatrix} \quad (1.4)$$

Решением краевой задачи будет решение этого СЛАУ.

Главной проблемой метода является построение правильной разностной схемы, которая будет сходиться к решению. Построение схемы выполняется исходя из свойств исходного дифференциального оператора [1].

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.

1.3 Метод конечных элементов

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 [5].

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 [7].

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, что бы верно вычислить необходимые для анализа конструкций параметры [2].

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

1.4 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* [4].

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.

2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ

2.1 Постановка задачи

Задачей является разработка приложения на языке высокого уровня, которое производит анализ напряжённо-деформированного состояния сплошного цилиндра. 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 сплошного цилиндра;

– наложение сил и закреплений к узлам созданной сетки;

– выбор материала сплошного цилиндра;

– построение математической модели сплошного цилиндра, представляющей собой матрицу жёсткости системы и вектор узловых нагрузок системы;

– решение полученной СЛАУ, в результате которой вычисляется вектор узловых перемещений системы;

– вычисление деформаций и напряжений системы;

– графическое отображение результатов вычислений.

Исходными данными для проекта являются:

– материал сплошного цилиндра – бетон;

– диаметр цилиндра – 0,4 метра;

– высота цилиндра – 0,5 метра;

– закрепление цилиндра – снизу;

– давление на верхнюю поверхность цилиндра – 5 тонн.

2.2 Математическая модель сплошного цилиндра

Моделируемая конструкция представляет собой цилиндр. Диаметр цилиндра равен 0,4 метра. Высота равна 0,5 метра.

Распределённая нагрузка сверху равна 5 тонн. Закрепление применяется к нижней поверхности цилиндра.

Материал, из которого сделан цилиндр – бетон. Модуль Юнга для данного материала равен $1{,}7 \cdot 10^{10}$ Па, а коэффициент Пуассона равен 0,2.

Для того, что бы построить математическую модель сплошного цилиндра, необходимо разбить его на конечно-элементную сетку.

Применяя конечные элементы второго порядка возможно сделать цилиндр с круглыми основаниями, однако это сильно усложнит систему. В данном случае цилиндр заменяется на 20-угольную призму, которая сначала разбивается на равные слои по высоте, а затем от центра основания призмы к её углам проводятся прямые, разбивая её на треугольные призмы.

Эти треугольные призмы не являются конечными элементами. В качестве исходных конечных элементов выбраны тетраэдры, которые получаются в результате разбиения треугольной призмы. К их вершинам приложены узловые усилия. Вектор узловых усилий тетраэдра представлен в виде:

$$\{\mathrm{R}\}^{\mathrm{T}} = \{\mathrm{X}_1, \mathrm{Y}_1, \mathrm{Z}_1, \mathrm{X}_2, \mathrm{Y}_2, \mathrm{Z}_2, \mathrm{X}_3, \mathrm{Y}_3, \mathrm{Z}_3, \mathrm{X}_4, \mathrm{Y}_4, \mathrm{Z}_4\} \quad (2.1)$$

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:

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

Перемещения узла тетраэдра также можно представить в таком виде:

$$\{F\} = [A^*]\{\alpha\} \quad (2.4)$$

где
$$[A^*] = \begin{bmatrix} 1 & x & y & z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x & y & z & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x & y & z \end{bmatrix}; \quad (2.5)$$

$$\{\alpha\}^{\mathrm{T}} = \{\alpha_1 \quad \alpha_2 \quad \ldots \quad \alpha_{12}\}. \quad (2.6)$$

Так как формула (2.4) применяется для любого узла тетраэдра, то для всех узлов тетраэдра вектор узловых перемещений принимает вид:

$$\{F^e\} = [A]\{\alpha\} \quad (2.7)$$

где $$[A] = \begin{bmatrix} 1 & x_1 & y_1 & z_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_1 & y_1 & z_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_2 & y_2 & z_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_3 & y_3 & z_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_4 & y_4 & z_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_4 & y_4 & z_4 \end{bmatrix}; \quad (2.8)$$

x_i, y_i, z_i $(i = \overline{1,4})$ – координаты узлов тетраэдра.

Из формулы (2.7) следует, что:

$$\{\alpha\} = [A]^{-1}\{F^e\} \quad (2.9)$$

где $$[A]^{-1} = \frac{1}{6V}\begin{bmatrix} a_1 & 0 & 0 & a_2 & 0 & 0 & a_3 & 0 & 0 & a_4 & 0 & 0 \\ b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 & 0 \\ c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 & 0 & 0 \\ d_1 & 0 & 0 & d_2 & 0 & 0 & d_3 & 0 & 0 & d_4 & 0 & 0 \\ 0 & a_1 & 0 & 0 & a_2 & 0 & 0 & a_3 & 0 & 0 & a_4 & 0 \\ 0 & b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 \\ 0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 & 0 \\ 0 & d_1 & 0 & 0 & d_2 & 0 & 0 & d_3 & 0 & 0 & d_4 & 0 \\ 0 & 0 & a_1 & 0 & 0 & a_2 & 0 & 0 & a_3 & 0 & 0 & a_4 \\ 0 & 0 & b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 \\ 0 & 0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 \\ 0 & 0 & d_1 & 0 & 0 & d_2 & 0 & 0 & d_3 & 0 & 0 & d_4 \end{bmatrix}; \quad (2.10)$$

V – объём элементарного тетраэдра;

$$a_i = (-1)^{i+1}\begin{vmatrix} x_j & y_j & z_j \\ x_k & y_k & z_k \\ x_n & y_n & z_n \end{vmatrix};$$

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

$$c_i = (-1)^{i+1}\begin{vmatrix} 1 & x_j & z_j \\ 1 & x_k & z_k \\ 1 & x_n & z_n \end{vmatrix};$$

$$d_i = (-1)^i \begin{vmatrix} 1 & x_j & y_j \\ 1 & x_k & y_k \\ 1 & x_n & y_n \end{vmatrix};$$

i, j, k, n – номера вершин элементарного тетраэдра.

Остальные значения *a, b, c, d* получаются круговой перестановкой индексов. Используя уравнения Коши и физические уравнения теории упругости, вычисляются деформации. Формула деформаций:

$$\{\varepsilon\} = [Q]\{\alpha\} \tag{2.11}$$

Также вычисляются и напряжения. Формула для напряжений:

$$\{\sigma\} = [E_0]\{\varepsilon\} = [E_0][Q]\{\alpha\}, \tag{2.12}$$

где
$$[Q] = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}; \tag{2.13}$$

$$[E_0] = \begin{bmatrix} 2G+\lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2G+\lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2G+\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{bmatrix}; \tag{2.14}$$

G, λ - модуль сдвига и коэффициент Ламе соответственно.

Подставляя (2.9) в (2.11) и (2.12), получаются выражения для деформаций и напряжений соответственно:

$$\{\varepsilon\} = [Q][A]^{-1}\{F^e\} \tag{2.15}$$

$$\{\sigma\} = [E_0][Q][A]^{-1}\{F^e\} \tag{2.16}$$

На основании принципа возможных перемещений составляется уравнение системы:

$$\{\delta F^e\}^T\{R\} = \iiint_V \{\delta\varepsilon\}^T\{\sigma\}dxdydz \tag{2.17}$$

Подставив в (2.17) выражения (2.15) и (2.16), учитывая, что в рассматриваемом случае все матрицы и $\{F^e\}$ не зависят от координат, получается, что узловые усилия через перемещения можно выразить следующим образом:

$$\{R\} = V[D]^T[E_0][D]\{F^e\} \tag{2.18}$$

где $[D] = [Q][A]^{-1}$.

Тогда матрица жёсткости тетраэдра принимает вид:

$$[k] = V[D]^T[E_0][D] \tag{2.19}$$

В результате матричных операций в (2.19) матрица жёсткости тетраэдра принимает вид:

$$[k] = \frac{1}{36V}\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \tag{2.20}$$

где $\left[k_{ij}\right] =$

$$= \begin{bmatrix} b_ib_j\rho + \left(c_ic_j + d_id_j\right)G & b_ic_j\lambda + c_ib_jG & b_id_j\lambda + d_ib_jG \\ c_ib_j\lambda + b_ic_jG & c_ic_j\rho + \left(b_ib_j + d_id_j\right)G & c_id_i\lambda + d_ic_jG \\ d_ib_j\lambda + b_id_jG & d_ic_j\lambda + c_id_jG & d_id_j\rho + \left(c_ic_j + b_ib_j\right)G \end{bmatrix}; \tag{2.21}$$

$\rho = 2G + \lambda$.

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

$$K_{ij} = \sum_{r=1}^{n} k_{ij}^r \tag{2.22}$$

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:

$$[K]\left\{F^{\text{узл}}\right\} = \{R\} \tag{2.23}$$

где $[K]$ – глобальная матрица жёсткости;

$\left\{F^{\text{узл}}\right\}$ – 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 [10].

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

2.3 Метод Гаусса для решения основного уравнения МКЭ

Для решения системы линейных алгебраических уравнений в этом курсовом проекте используется метод Гаусса.

Процесс решения по методу Гаусса состоит из двух этапов: прямой ход и обратный ход.

На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме. А именно, среди элементов первого столбца матрицы выбирают ненулевой, перемещают его на крайнее верхнее положение перестановкой строк и вычитают получившуюся после перестановки первую строку из остальных строк, умножив её на величину, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, обнуляя тем самым столбец под ним.

Обратный ход: осуществляется определение значений неизвестных. Из последнего уравнения преобразованной системы вычисляется значение переменной x_n, после этого из предпоследнего уравнения становится возможным определение переменной x_{n-1} и так далее [3].

Преимущества метода:

– менее сложный для реализации по сравнению с другими методами;

– позволяет однозначно установить, совместна система или нет, и если совместна, найти её решение;

– позволяет найти максимальное число линейно независимых уравнений – ранг матрицы системы.

Существенным недостатком этого метода является скорость выполнения. Существует огромный ряд методов, вычисляющих СЛАУ намного быстрее. Однако из-за своей простоты в реализации был выбран именно метод Гаусса.

2.4 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 [6].

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*.

Для разработки приложения была выбрана трёхслойная архитектура. Она представляет собой разбиение приложения на три логических части. Каждая логическая часть представляет собой отдельный проект и имеет ссылку на предыдущую.

Первая часть – слой доступа к данным *DAL*. Этот слой необходим для определения сущностей, которые используются в программе и реализации слоя доступа к данным. Он разбит на два пространства имён – *Data* и *Loaders*.

Data хранит сущности, используемые в слое *DAL*. Они представлены представлены в таблице 3.1.

Таблица 3.1 – Описание классов сущностей *DAL* слоя

Имя класса сущности	**Описание**
Node	Класс, описывающий узел конечно-элементной сетки. Хранит в себе уникальный идентификатор узла и расположение его в пространстве.
Element	Абстрактный класс конечного элемента. Хранит идентификатор и список индексов узлов, ему принадлежащих.
LinearTetrahedron	Класс линейного тетраэдрального конечного элемента. Наследуется от абстрактного класса конечного элемента. Хранит 4 идентификатора узла конечно-элементной сетки, которые ему принадлежат.
Mesh	Класс конечно-элементной сетки детали. Хранит списки узлов и конечных элементов сетки.
NodeLoad	Класс условия, наложенного на узел. Хранит в себе состояние – является ли узел закреплённым или находится под действием силы. Хранит идентификатор узла, к которому прилагается и силы, применяемые к узлу по каждой координатной оси.
Material	Класс, описывающий материал детали. Хранит в себе название материала, его цвет, уникальный идентификатор и характеристики – модуль Юнга и коэффициент Пуассона.

Loaders представляет собой набор классов, реализующих доступ к данным из любых источников. Для этого используется модернизированный шаблон программирования репозиторий, позволяющий получать сущности слоя *DAL* из любого источника данных. Примерами таких источников данных являются текстовые файлы, файлы формата *csv*, базы данных и любые другие. Они все имеют одну абстрактную реализацию. Благодаря этому источник данных для приложения возможно менять динамически в ходе выполнения программы.

Описание классов, реализующих доступ к данным при помощи шаблона программирования репозиторий, находится в таблице 3.2.

Таблица 3.2 – Описание классов, реализующих доступ к данным

Имя класса	**Описание**
ILoader<T>	Параметризованный интерфейс, содержащий описание сигнатуры метода загрузки для любого класса сущности *T*.
MaterialsCsvLoader	Класс, реализующий функционал интерфейса *ILoader<IList<Material>>*. Предназначен для загрузки списка материалов детали из *csv* файла.
MeshTxtLoader	Класс, реализующий функционал интерфейса *ILoader<Mesh>*. Предназначен для загрузки конечно-элеметной сетки детали из файла формата *txt*.
NodeLoadsCsvLoader	Класс, реализующий функционал интерфейса *ILoader<IList<NodeLoad>>*. Предназначен для загрузки списка условий, наложенных на узлы конечно-элементной сетки, из *csv* файла.

Вторая часть – слой бизнес логики *BLL*. Это самая объёмная часть программы. Именно здесь происходят все построения и вычисления. Она разбита на множество пространств имён, каждое из которых отвечает за свою область.

Первое пространство имён *Data*. Оно отвечает за отражения сущностей *DAL* слоя в *BLL* слой. При этом это не просто копии классов *DAL* слоя. Это полноценные классы, которые содержат собственные методы и свойства, которых нет в классах *DAL* слоя. Эти классы называются объектами передачи данных *DTO*. Описание этих классов находится в таблице 3.3.

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

Имя класса сущности	**Описание**
NodeDTO	Класс, описывающий узел конечно-элементной сетки. Хранит в себе уникальный идентификатор узла и расположение его в пространстве.

Продолжение таблицы 3.3

Имя класса сущности	Описание
ElementDTO	Абстрактный класс конечного элемента. Хранит массив узлов, ему принадлежащих. Также содержит свойства центральной точки и объёма этого конечного элемента. Кроме того определяет сигнатуры методов и свойств нахождения расстояния до другого конечного элемента, получения матриц, используемых в вычислениях, в том числе матрицы жёсткости данного конечного элемента.
LinearTetrahedronDTO	Класс линейного тетраэдрального конечного элемента. Наследуется от абстрактного класса конечного элемента. Хранит четыре узла конечного элемента, являющихся его вершинами. Реализует абстрактные методы и свойства класса конечного элемента.
MeshDTO	Класс конечно-элементной сетки детали. Хранит списки узлов и конечных элементов сетки. Также определяет метод получения глобальной матрицы жёсткости конечно-элеметной сетки.
NodeForceVector	Класс, представляющий собой вектор сил, действующих на узел.
NodeLoadDTO	Класс условия, наложенного на узел. Хранит в себе состояние – является ли узел закреплённым или находится под действием силы. Хранит идентификатор узла, к которому прилагается и вектор сил, применяемый к этому узлу.
MaterialDTO	Класс, описывающий материал детали. Хранит в себе название материала, его цвет, уникальный идентификатор и характеристики – модуль Юнга и коэффициент Пуассона. Также хранит методы вычисления модуля сдвига и коэффициента Ламе для данного материала.

Следующее пространство имён – *Services*. В нём находятся классы, которые позволяют преобразовывать сущности слоя *DAL* в слой *BLL*. Все классы основаны на двух ключевых особенностях.

Первая особенность состоит в том, что в конструктор класса передаётся абстрактный загрузчик, представляющий собой интерфейс *ILoader<T>*, в результате чего сервис не зависит от конкретного файла, из которого загружаются данные.

Вторая особенность в том что сервисы преобразуют сущности одного слоя в другой только тогда, когда в этом есть необходимость.

Более детальное описание сервисов находится в таблице 3.4.

Таблица 3.4 – Описание классов сервисов

Название сервиса	**Описание**
MaterialService	Класс сервиса, выполняющее преобразование списка сущностей материалов слоя *DAL* в список сущностей материалов слоя *BLL*.
MeshService	Класс сервиса, выполняющее преобразование сущности конечно-элементной сетки слоя *DAL* в сущность конечно-элементной сетки слоя *BLL*.
NodeLoadService	Класс сервиса, выполняющее преобразование списка условий, наложенных на узлы слоя *DAL* в список условий, наложенных на узлы слоя *BLL*.

Далее идёт одно из ключевых пространств имён – *Generators.* Это пространство имён отвечает за программную генерацию сущностей, в отличие от сервисов, которые берут сущности извне.

Это пространство содержит также пространство имён геометрических фигур. Оно отвечает за определение геометрии фигур и их внутреннего разбиения на необходимые примитивы.

Подробное описание классов пространства имён *Generators* находится в таблице 3.5.

Таблица 3.5 – Описание классов генераторов

Название класса	**Описание**
Prism	Класс *n*-угольной призмы. Класс хранит свойства призмы, такие как количество сторон, длина сторон, высота призмы и радиус описанной вокруг призмы цилиндра. Также определяет метод разбиения призмы на узлы и треугольные призмы, которые из этих узлов состоят.
TrianglePrism	Класс треугольной призмы. Хранит идентификатор этой треугольной призмы во внешней призме и реализует метод разбиения треугольной призмы на линейные тетраэдральные конечные элементы
MeshGenerator	Класс, генерирующий сетки для различных геометрических тел. Также класс определяет методы генерации сеток на основе уже существующих, получающихся в результате перемещения или масштабирования исходных.
NodeLoadsGenerator	Класс, генерирующий нагрузки на сетки. Определяет методы генерации нагрузок пресса.

Четвёртое пространство имён – *ModelMakers*. Это пространство имён отвечает за построение *3D* модели сетки. Описание классов этого пространства имён содержится в таблице 3.6.

Таблица 3.6 – Описание классов построения *3D* модели сетки

Название класса	**Описание**
IMeshModelMaker	Интерфейс генератора модели конечно-элементной сетки. Определяет методы создания геометрии сетки с наложенными нагрузками и создания сетки с распределением характеристик конечных элементов(перемещений, деформаций, напряжений).
SimpleModelMaker	Класс, реализующий интерфейс генератора модели конечно-элементной сетки. Переопределяет все методы интерфейса.

Пятое пространство имён – *Solvers*. В этом пространстве имён содержатся классы для вычисления узловых перемещений сетки и характеристик, которые от них зависят: деформации и напряжения. Описание этих классов представлено в таблице 3.7.

Таблица 3.7 – Описание классов пространства имён *Solvers*

Название класса	**Описание**
ISolver	Интерфейс, определяющий описание методов для вычисления узловых перемещения, деформаций и напряжений конечно-элементной сетки.
SimpleMeshSolver	Класс, реализующий интерфейс, вычисляющий узловые перемещения, деформации и напряжения конечно-элементной сетки.

Далее следует пространство имён *Tools*. Тут содержатся классы, представляющие собой инструменты, используемые в ходе выполнения программы. Подробное описание классов находится в таблице 3.8.

Таблица 3.8 – Описание классов пространства имён *Tools*

Название класса	**Описание**
GradientMaker	Класс, содержащий метод, возвращающий значение цвета в зависимости от входящего вещественного значения.
SLAESolver	Класс, содержащий методы для решения СЛАУ. Реализованный метод, использованный в программе – метод Гаусса.

Седьмое пространство имён – *UIElements*. Здесь содержатся классы, представляющие собой элементы графического интерфейса. Описание классов этого пространства имён содержится в таблице 3.9.

Таблица 3.9 – Описание классов пространства имён *UIElements*

Название класса	**Описание**
GradientScale	Класс, представляющий собой градиентную шкалу, изображаемую в области рисования сетки, значения которой сопоставляются с характеристикой детали, отображаемой на данный момент(перемещение, деформация или напряжение).

Заключительное пространство имён – *DI*. Для того что бы сделать приложение более гибким, используется механизм внедрения зависимостей. Этот механизм позволяет сделать взаимодействующие в приложении объекты слабосвязанными путём использования специальных контейнеров, которые устанавливают зависимости между абстракциями и конкретными объектами.

Для реализации внедрения зависимостей использовалась внешняя библиотека *Ninject*. Пространство имён *DI* представляет собой реализации этих модулей. Детальное описание этих классов представлено в таблице 3.10.

Таблица 3.10 – Описание классов пространства имён *DI*

Название класса	**Описание**
MaterialsLoaderModule	Класс, представляющий собой модуль для внедрения зависимостей, соединяющий интерфейс для загрузки списка материалов с его реализаций, получающей этот список из *csv* файла.
MeshTxtLoaderModule	Класс, представляющий собой модуль для внедрения зависимостей, соединяющий интерфейс для загрузки конечно-элементной сетки с его реализаций, получающей эту сетку из *txt* файла.
NodeLoadsCsvLoaderModule	Класс, представляющий собой модуль для внедрения зависимостей, соединяющий интерфейс для загрузки списка наложенных на узел условий с его реализаций, получающей этот список из *csv* файла.
SimpleModelMakerModule	Класс, являющийся модулем для внедрения зависимостей, соединяющий интерфейс создания геометрии сетки с его реализацией.

Продолжение таблицы 3.10

SimpleSolverModule	Класс, являющийся модулем для внедрения зависимостей, соединяющий интерфейс вычисления узловых перемещений детали и её дополнительных характеристик с его реализацией.

Третья часть – слой графического интерфейса *WPFGUI*. В этом слое содержатся окна графического интерфейса и логика взаимодействия графического интерфейса и внутренних классов программы. Описание классов графического интерфейса слоя *WPFGUI* находится в таблице 3.11.

Таблица 3.11 – Описание классов графического интерфейса

Название класса	**Описание**
App	Класс, унаследовавший функционал класса *Application*. Является точкой входа в приложение. В этом классе происходит чтение файла конфигурации и создаётся контейнер внедрения зависимостей.
MainWindow	Класс, представляющий собой окно приложения. Это окно является главным, именно его вызывает класс *App*. В этом окне происходит взаимодействие оконного интерфейса и главной модели представления программы.
MainViewModel	Класс, представляющий собой главную модель представления программы. Является связующим звеном между окном приложения и классами слоя бизнес логики, где происходит вся обработка информации.

Кроме того на этом слое также находится файл конфигурации приложения *config.json* и файлы с разметкой графического интерфейса, представленные файлами формата *xaml*.

3.2 Функционал разработанного приложения

При запуске программы открывается главное окно пользовательского интерфейса. При этом программа загружает материалы из файла с материалами, путь к которому прописан в файле конфигурации.

Внешний вид главного окна приложения изображён на рисунке 3.1.

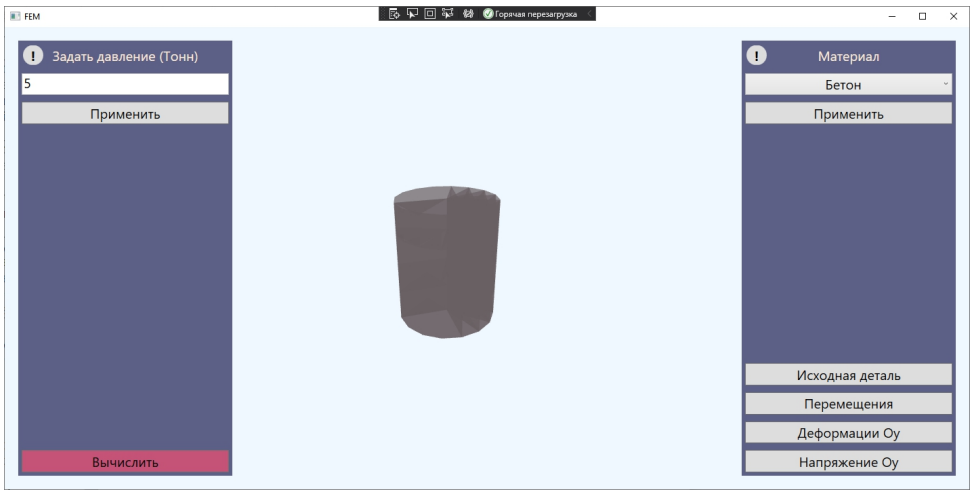

Рисунок 3.1 – Внешний вид главного окна приложения

Также во время запуска программа строит конечно-элементную сетку для сплошного цилиндра заданных размеров.

В программе имеется функционал выбора различных материалов. Для выбора материала необходимо выбрать в выплывающем списке нужный материал и нажать на кнопку “Применить”.

Что бы посмотреть информацию о выбранном материале необходимо навести мышкой на круглую кнопку 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.

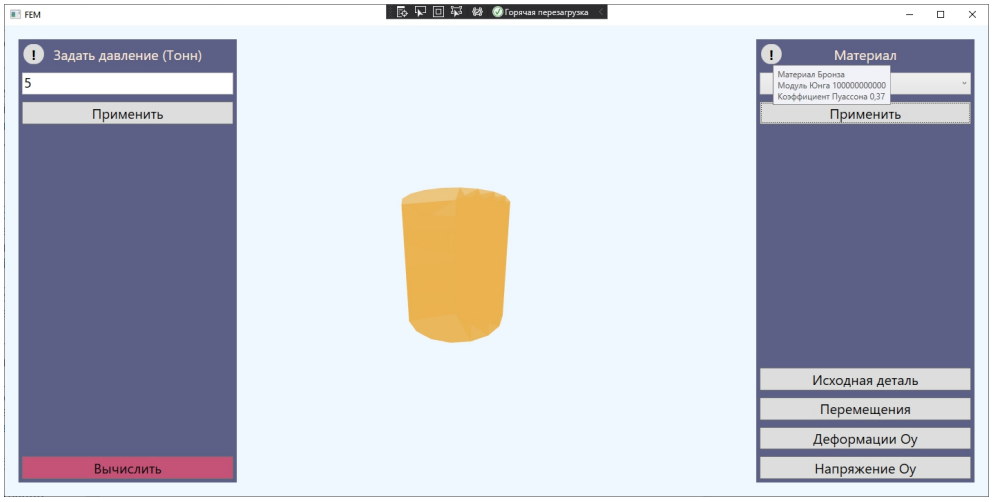

Рисунок 3.2 – Внешний вид окна после выбора материала “Бронза” и наведения на кнопку с информацией о материале

В левом верхнем углу программы располагается поле для введения давления на деталь. Давление задаётся в тоннах на верхнюю поверхность детали. Автоматически происходит закрепление детали снизу.

Для установки давления на верхнюю поверхность определённой величины в поле ввода необходимо ввести значение давления в тоннах и нажать на кнопку “Применить”. Также можно посмотреть информацию о конечно-элементной сетке детали, наведя мышку на круглую кнопку с изображением восклицательного знака. Результат этих действий изображён на рисунке 3.3.

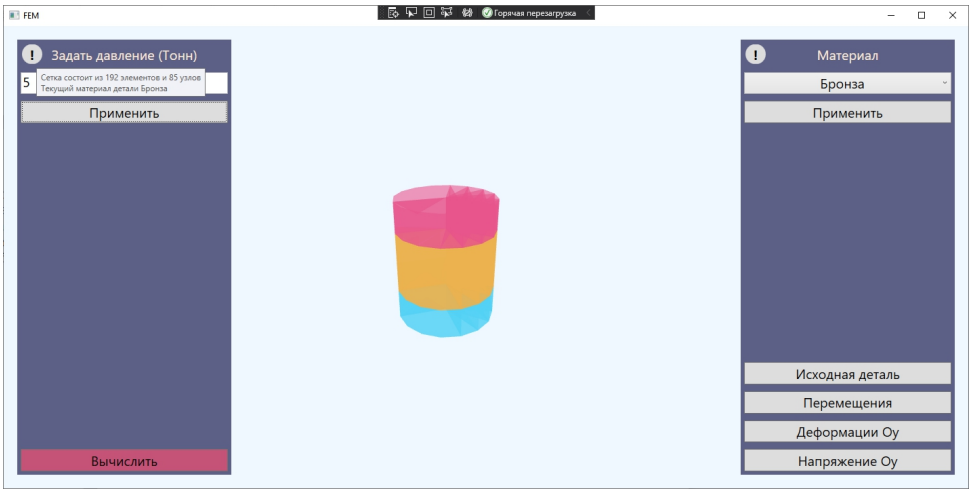

Рисунок 3.3 – Результат применения давления и закрепления на деталь и наведение мышки на кнопку с информацией о детали

После выбора материала и применения давления необходимо нажать на кнопку “Вычислить”. Во время вычисления графический интерфейс становится заблокированным, пока вычисление не завершится. После завершения вычисления произойдёт уведомление пользователя об этом и на экран будет выведено время, затраченное на вычисление.

Результат нажатия на кнопку “Вычислить” изображён на рисунке 3.4.

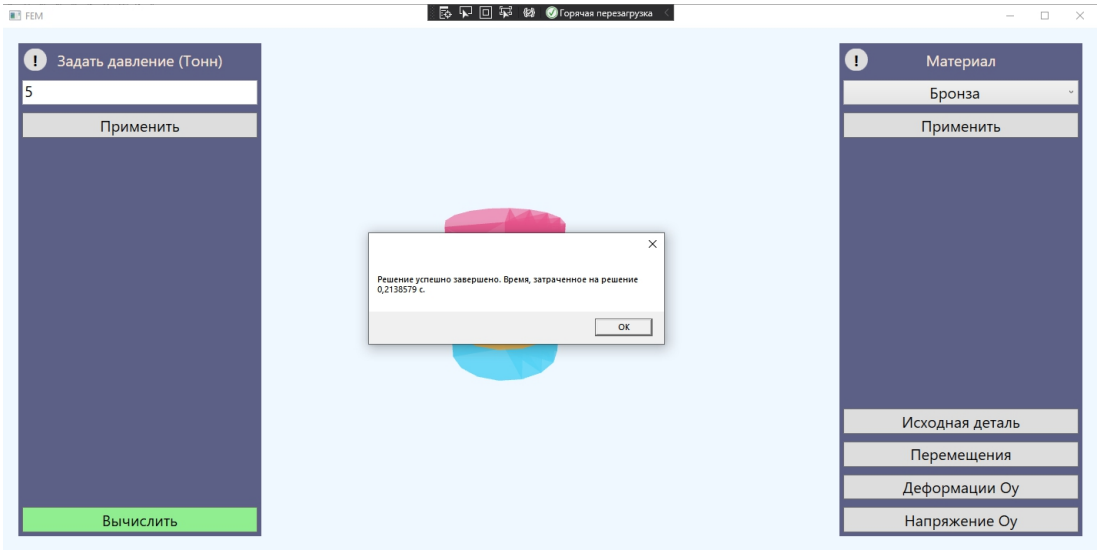

Рисунок 3.4 – Результат нажатия на кнопку “Вычислить”

После вычисления появляется возможность посмотреть на вычисленные характеристики детали после применения введённого давления. В правом нижнем углу расположены четыре кнопки, имеющие следующую функциональность:

– кнопка “Исходная деталь” позволяет посмотреть на деталь, перед применением давления;

– кнопка “Перемещения” позволяет отобразить информацию об узловых перемещений детали в результате применённого давления;

– кнопка “Деформации *Oy*” позволяет отобразить информацию об узловых деформациях детали в результате применённого давления;

– кнопка “Деформации *Ox*” позволяет отобразить информацию об узловых напряжениях детали в результате применённого давления.

Результат нажатия на кнопку “Перемещения” после выбора материала, задания давления и проведения вычисления изображён на рисунке 3.5.

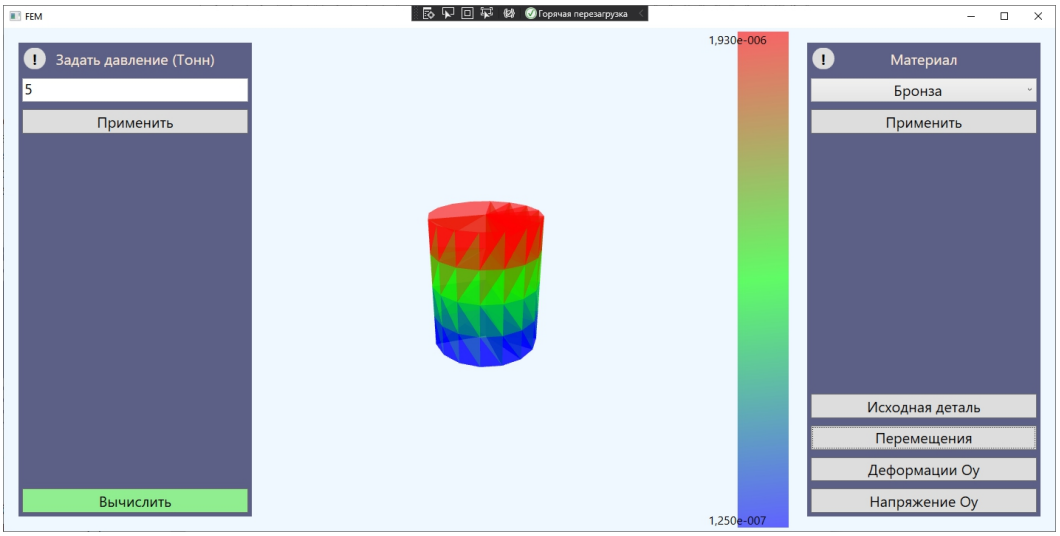

Рисунок 3.5 – Результат нажатия на кнопку “Перемещения”

В результате нажатия на любую из кнопок, отображающих характеристику узлов сетки после приложения условий, деталь окрашивается градиентом, который представляет собой диапазон изменений значений этой характеристики в детали для каждого конечного элемента.

Также, после нажатия на любую из кнопок характеристики, выводится шкала, на которой происходит сопоставление цвета градиента конечного элемента детали и значения вычисленной характеристики для этого конечного элемента.

Проверка результатов вычислений производится в следующей главе.

4 ВЕРИФИКАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Пример решения задачи разработанным приложением

Во время запуска приложения произошло построение детали, соответствующей заданию, а именно сплошного цилиндра с высотой 500 миллиметров и радиусом 200 миллиметров.

После запуска приложения был выбран материал цилиндра. Исходный материал детали – бетон.

После этого было задано давление на верхнюю поверхность детали. Значение давления 5 тонн. Снизу детали автоматически было задано закрепление.

Далее было произведено вычисление узловых перемещений детали посредством нажатия на кнопку “Вычислить”. После вычисления было выведено время выполнения вычисления, равное приблизительно трёмстам миллисекундам.

Результат вычисления изображён на рисунке 4.1.

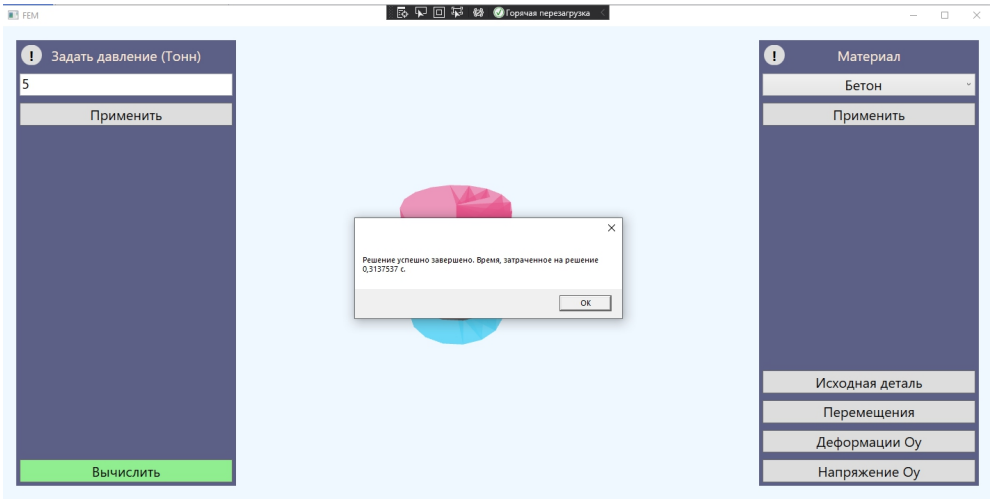

Рисунок 4.1 – Результат выбора параметров вычисления и нажатия на кнопку “Вычислить”

После вычисления были графически и численно выведены распределения параметров узлов сетки. Значения параметров узлов сетки содержатся в таблице 4.1.

Таблица 4.1 – Значения вычисленных параметров узлов

Имя параметра	**Максимальное значение**	**Минимальное значение**
Перемещение	$1{,}214 \cdot 10^{-5}$ м.	$8{,}247 \cdot 10^{-7}$ м.
Деформация	$6{,}693 \cdot 10^{-6}$	$-7{,}485 \cdot 10^{-6}$
Напряжение	$4{,}728 \cdot 10^{4}$ Па	$-4{,}555 \cdot 10^{4}$ Па

Для каждой характеристики происходит графический вывод распределения. Внешний вид распределения узловых перемещений изображён на рисунке 4.2.

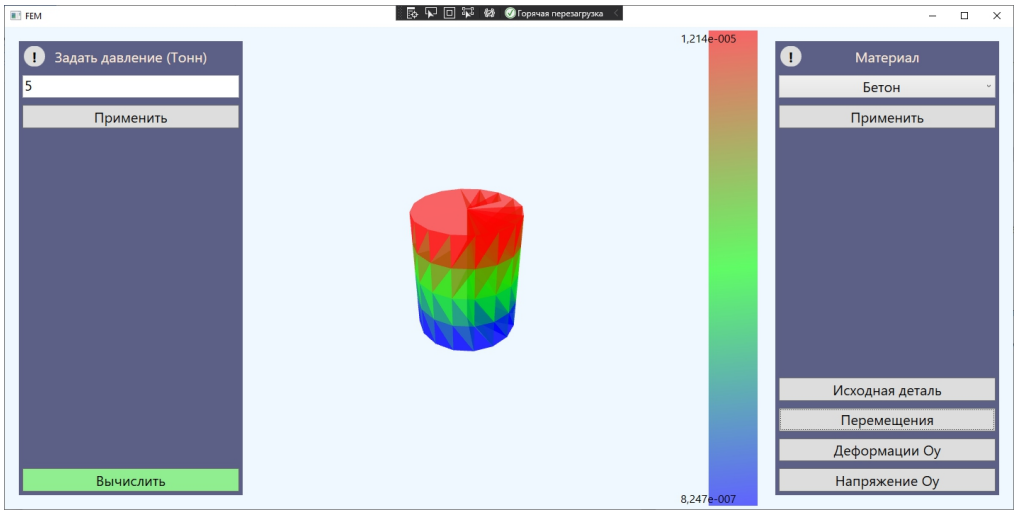

Рисунок 4.2 – Внешний вид распределения узловых перемещений

4.2 Верификация полученных результатов в пакете *SOLIDWORKS*

В первую очередь была построена деталь по размерам задания. Был построен эскиз окружности радиусом 200 миллиметров. Далее была применена команда “Вытянутая бобышка/основание”, где в поле параметра длины было задано значение 500 миллиметров. Внешний вид построенной детали изображён на рисунке 4.3.

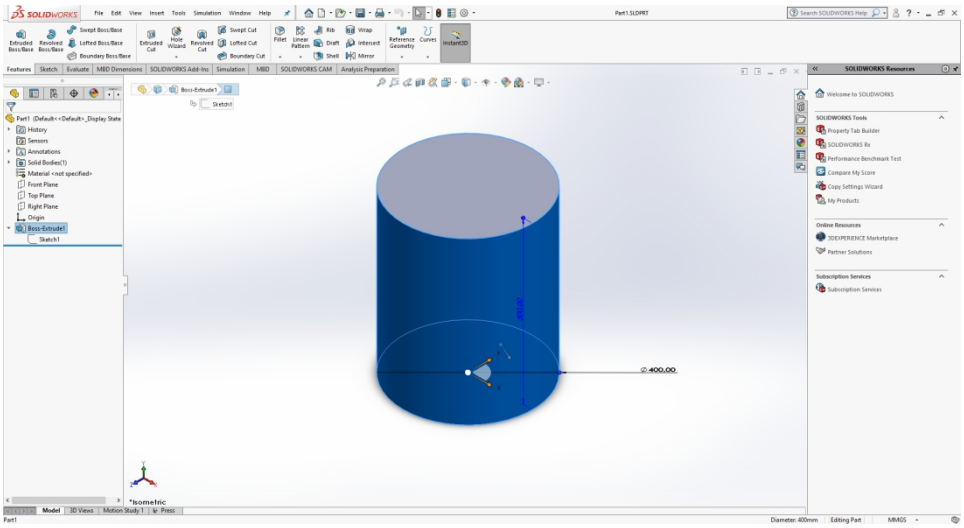

Рисунок 4.3 – Исходная деталь, построенная в *SOLIDWORKS*

Далее при помощи плагина *SOLIDWORKS Simulation* был задан материал – бетон.

После этого было применено закрепление детали на нижней поверхности и давление на верхней. Давление было вычислено путём деления силы, применённой на поверхность детали на площадь поверхности детали.

Результат задания материала и приложения нагрузок изображён на рисунке 4.4.

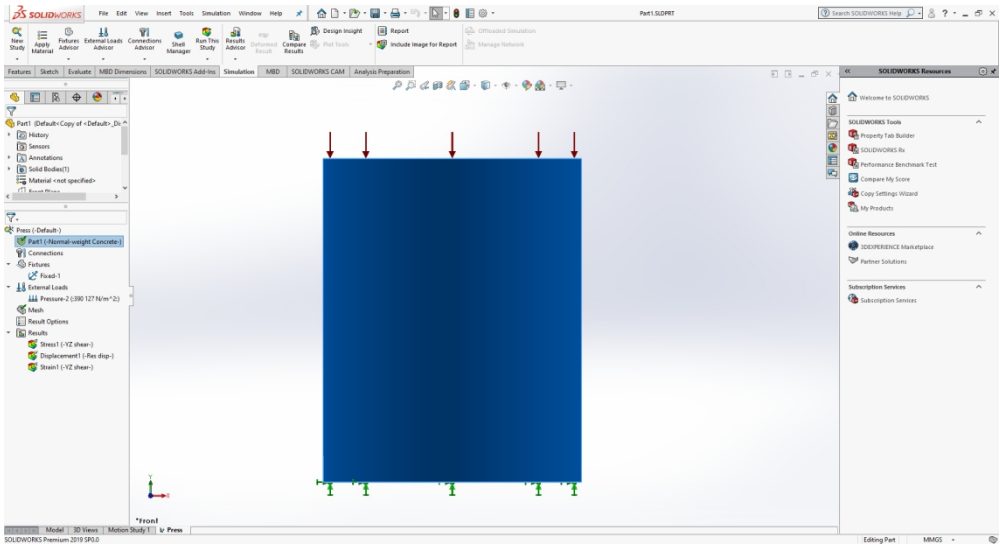

Рисунок 4.4 – Результат задания материала цилиндра и приложения к нему закрепления и давления

Далее было произведено вычисление характеристик узлов детали. Внешний вид распределения характеристики детали на рисунке 4.5.

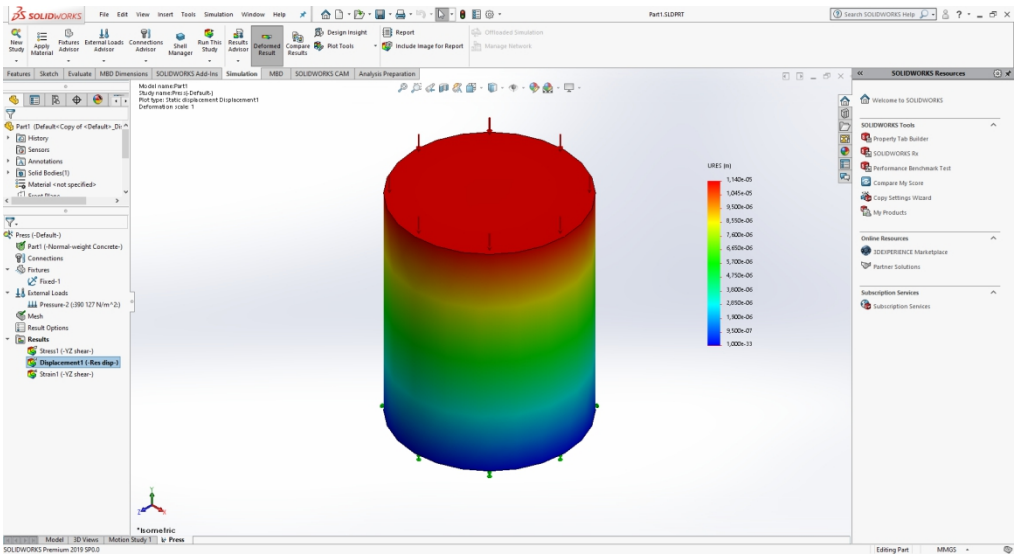

Рисунок 4.5 – Распределения узловых перемещений детали

Сравнительные значения характеристик, вычисленных в приложении и в *SOLIDWORKS Simulation* расположены в таблице 4.2.

Таблица 4.2 – Сравнительные значения вычисленных параметров

Параметр	**Значение приложения**	**Значение *SOLIDWORKS***	**Погрешность**
Максимальное перемещение	$1{,}214 \cdot 10^{-5}$ м.	$1{,}140 \cdot 10^{-5}$ м.	6%
Минимальное перемещение	$8{,}247 \cdot 10^{-7}$ м.	$9{,}5 \cdot 10^{-7}$ м.	13%
Максимальная деформация	$6{,}693 \cdot 10^{-6}$	$6{,}102 \cdot 10^{-6}$	9%
Минимальная деформация	$-7{,}485 \cdot 10^{-6}$	$-6{,}162 \cdot 10^{-6}$	17%
Максимальное напряжение	$4{,}728 \cdot 10^{4}$ Па	$4{,}322 \cdot 10^{4}$ Па	9%
Минимальное напряжение	$-4{,}555 \cdot 10^{4}$ Па	$-4{,}364 \cdot 10^{4}$ Па	4%

Максимальная погрешность вычислений относительно *SOLIDWORKS* не превышает 20%. Погрешность складывается из того, что в приложении давление применяется на всю поверхность тела равномерно, при этом тело по бокам немного сплющивается, а в *SOLIDWORKS* на деталь накладывается дополнительное условие что верхняя поверхность остаётся плоской.

Также погрешность исходит из разных конечно-элементных сеток и разных алгоритмов вычисления характеристик узлов сетки.

Верификация показала, что приложение работает правильно и верно вычисляет узловые характеристики детали.

ЗАКЛЮЧЕНИЕ

Для решения поставленной задачи было разработано приложение моделирования напряжённо-деформированного состояния сплошного цилиндра при наличии нагрузки в виде давления, применяемого на поверхность детали и закрепления нижней поверхности детали.

Приложение разработано на языке программирования высокого уровня C#. Использована трёхслойная архитектура приложения и механизм внедрения зависимостей. 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 пакетом *SOLIDWORKS*. Результаты вычисления имеют небольшую погрешность относительно этого пакета, что говорит о правильности реализации алгоритма решения поставленной задачи.

При сравнении разработанного программного комплекса с такими профессиональными пакетами как *SOLIDWORKS*, заметен 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 *SOLIDWORKS*, а смогут использовать достаточно простое и удобное в использовании разработанное приложение, способное решать поставленную задачу.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Метод конечных разностей [Электронный ресурс]. – Режим доступа: http://www.simumath.net/library/book.html?code=Ur_Mat_Ph_method_net. – Дата доступа: 10.04.2021.
2. Метод конечных элементов [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Метод_конечных_элементов. – Дата доступа: 10.04.2021.
3. Метод гаусса [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Метод_Гаусса. – Дата доступа: 10.04.2021.
4. Алямовский, А. А. Инженерные расчеты и SolidWorks Simulation / А.А. Алямовский. – М.: ДМК Пресс, 2015. – 464 с.
5. Шимкович, Д.Г. Femap & Nastran. Инженерный анализ методом конечных элементов / Д.Г. Шимкович. – М.: ДМК Пресс, 2018. – 738 с.
6. Агальцов, В.П. Математические методы в программировании: Учебник / В.П. Агальцов, И.В. Волдайская. – М.: ИД Форум, 2013. – 240 с.
7. 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.
8. Рассел, Джесси Метод дискретного элемента: моногр. / Джесси Рассел. – М.: VSD, 2013. – 716 с.
9. 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.
10. Курочка, К.С. Конечно-элементное моделирование физических систем. / Курочка К.С., Токочаков В.И., Комраков В.В. – Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2012, – 178 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг основных классов приложения

```
namespace FEM.BLL.Data
{
    /// <summary>
    /// Узел конечно-элементной сетки
    /// </summary>
    public class NodeDTO
    {
        /// <summary>
        /// Создание узла
        /// </summary>
        /// <param name="id">Идентификатор узла</param>
        /// <param name="x">Координата x узла</param>
        /// <param name="y">Координата y узла</param>
        /// <param name="z">Координата z узла</param>
        public NodeDTO(int id, double x, double y, double z)
        {
            Id = id;
            X = x;
            Y = y;
            Z = z;
        }
        /// <summary>
        /// Идентификатор узла
        /// </summary>
        public int Id { get; }
        /// <summary>
        /// Координата x узла
        /// </summary>
        public double X { get; private set; }
        /// <summary>
        /// Координата y узла
        /// </summary>
        public double Y { get; private set; }
        /// <summary>
        /// Координата z узла
        /// </summary>
        public double Z { get; private set; }

        /// <summary>
        /// Создаёт копию текущего узла
        /// </summary>
        /// <returns>Копия текущего узла</returns>
        public NodeDTO Copy()
        {
            return new NodeDTO(Id, X, Y, Z);
        }

        /// <summary>
        /// Перемещает узел по трём координатам
        /// </summary>
        /// <param name="x">Смещение по координате x</param>
        /// <param name="y">Смещение по координате y</param>
        /// <param name="z">Смещение по координате z</param>
        public void Move(double x, double y, double z)
        {
            X += x;
            Y += y;
            Z += z;
```

```
        }
    }
}


using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Media.Media3D;

namespace FEM.BLL.Data
{
    /// <summary>
    /// Конечный элемент
    /// </summary>
    public abstract class ElementDTO
    {
        /// <summary>
        /// Создание конечного элемента
        /// </summary>
        /// <param name="id">Идентификатор элемента</param>
        /// <param name="nodes">Узлы конечного элемента</param>
        /// <param name="color">Цвет конечного элемента</param>
        public ElementDTO(int id, NodeDTO[] nodes, Color color)
        {
            Id = id;
            Nodes = nodes;
            Color = color;
        }
        /// <summary>
        /// Идентификатор конечного элемента
        /// </summary>
        public int Id { get; }
        /// <summary>
        /// Узлы конечного элемента
        /// </summary>
        public NodeDTO[] Nodes { get; }
        /// <summary>
        /// Цвет конечного элемента
        /// </summary>
        public Color Color { get; set; }
        /// <summary>
        /// Получает локальную матрицу жесткости конечного элемента
        /// </summary>
        /// <param name="material">Материал конечного элемента</param>
        /// <returns>Локальная матрица жесткости конечного элемента</returns>
        public abstract double[,] GetStiffnessMatrix(MaterialDTO material);
        /// <summary>
        /// Создаёт копию экземпляра конечного элемента
        /// </summary>
        /// <param name="nodes">Узлы конечно-элементной сетки</param>
        /// <returns>Копия экземпляра конечного элемента</returns>
        public abstract ElementDTO Copy(IList<NodeDTO> nodes);

        /// <summary>
        /// Объём конечного элемента
        /// </summary>
        public abstract double Volume { get; }
        /// <summary>
        /// Центральная точка конечного элемента
        /// </summary>
        public abstract Point3D CenterPoint { get; }

        public double GetDistanceTo(ElementDTO other)
```

```
            {
                return Math.Sqrt(Math.Pow(this.CenterPoint.X - other.CenterPoint.X, 2)
+ Math.Pow(this.CenterPoint.Y - other.CenterPoint.Y, 2) + Math.Pow(this.CenterPoint.Z -
other.CenterPoint.Z, 2));
            }

            /// <summary>
            /// Вычисляет эластичную матрицу конечного элемента
            /// </summary>
            /// <param name="material">Материал линейного тетраэдра</param>
            /// <returns>Эластичная матрица линейного тетраэдра</returns>
            public abstract double[,] GetElasticMatrix(MaterialDTO material);
            /// <summary>
            /// Вычисляет обратную матрицу A для вычисления геометрической матрицы
            /// </summary>
            /// <returns>Обратная матрица A</returns>
            public abstract double[,] GetInversedA();
            /// <summary>
            /// Геометрическая матрица конечного элемента [D]
            /// </summary>
            public abstract double[,] GeometricMatrix { get; }
        }
    }

    using System.Windows.Media;

    namespace FEM.BLL.Data
    {
        /// <summary>
        /// Материал конечного элемента
        /// </summary>
        public class MaterialDTO
        {
            public MaterialDTO()
            {

            }
            /// <summary>
            /// Создание материала
            /// </summary>
            /// <param name="id">Идентификатор материала</param>
            /// <param name="name">Имя материала</param>
            /// <param name="color">Цвет материала</param>
            /// <param name="modulus">Модуль Юнга материала при 18 градусах цельсия
(Н/м^2)</param>
            /// <param name="coefficient">Коэффициент Пуассона материала</param>
            public MaterialDTO(int id, string name, Color color, double modulus,
double coefficient)
            {
                Id = id;
                Name = name;
                Color = color;
                YoungModulus = modulus;
                PuassonsCoefficient = coefficient;
            }
            /// <summary>
            /// Идентификатор материала
            /// </summary>
            public int Id { get; }
            /// <summary>
            /// Имя материала
            /// </summary>
            public string Name { get; }
            /// <summary>
```

```
            /// Цвет материала
            /// </summary>
            public Color Color { get; }
            /// <summary>
            /// Модуль Юнга материала при 18 градусах цельсия (Н/м^2)
            /// </summary>
            public double YoungModulus { get; }
            /// <summary>
            /// Коэффициент Пуассона материала
            /// </summary>
            public double PuassonsCoefficient { get; }

            private double? _shearModulus;
            /// <summary>
            /// Модуль сдвига материала
            /// </summary>
            public double ShearModulus
            {
                get
                {
                    if (_shearModulus == null)
                    {
                        _shearModulus = YoungModulus / (2 * (1 + PuassonsCoefficient));
                    }
                    return (double)_shearModulus;
                }
            }

            private double? _lameCoefficient;
            /// <summary>
            /// Коэффициент Ламе материала
            /// </summary>
            public double LameCoefficient
            {
                get
                {
                    if (_lameCoefficient == null)
                    {
                        _lameCoefficient = PuassonsCoefficient * YoungModulus / ((1 +
PuassonsCoefficient) * (1 - 2 * PuassonsCoefficient));
                    }
                    return (double)_lameCoefficient;
                }
            }
        }
    }


    using System.Collections.Generic;

    namespace FEM.BLL.Data
    {
        /// <summary>
        /// Конечно-элементная сетка детали
        /// </summary>
        public class MeshDTO
        {
            /// <summary>
            /// Создание конечно-элементной сетки
            /// </summary>
            /// <param name="nodes">Узлы сетки</param>
            /// <param name="elements">Конечные элементы сетки</param>
            public MeshDTO(IList<NodeDTO> nodes, IList<ElementDTO> elements)
```

```
        {
            Nodes = nodes;
            Elements = elements;
        }
        /// <summary>
        /// Узлы сетки
        /// </summary>
        public IList<NodeDTO> Nodes { get; }
        /// <summary>
        /// Конечные элементы сетки
        /// </summary>
        public IList<ElementDTO> Elements { get; }
        /// <summary>
        /// Создаёт копию сетки
        /// </summary>
        /// <returns>Копия текущей сетки</returns>
        public MeshDTO Copy()
        {
            IList<NodeDTO> nodes = new List<NodeDTO>();
            IList<ElementDTO> elements = new List<ElementDTO>();

            foreach (var node in Nodes)
            {
                nodes.Add(node.Copy());
            }

            foreach (var element in Elements)
            {
                elements.Add(element.Copy(nodes));
            }

            return new MeshDTO(nodes, elements);
        }

        /// <summary>
        /// Получает глобальную матрицу жесткости сетки
        /// </summary>
        /// <param name="material">Материал детали сетки</param>
        /// <returns>Глобальная матрица жесткости</returns>
        public double[,] GetStiffnessMatrix(MaterialDTO material)
        {
            double[,] stiffnessMatrix = new double[Nodes.Count * 3, Nodes.Count *
3];
            foreach (var el in Elements)
            {
                int nodesCount = el.Nodes.Length;
                double[,] elStiffness = el.GetStiffnessMatrix(material);
                //Цикл для прохода по узлам матрицы жесткости конечного элемента
                for (int i = 0; i < nodesCount; i++)
                {
                    for (int j = 0; j < nodesCount; j++)
                    {
                        //Цикл для прохода по всем степеням свободы узла конечного
элемента
                        for (int k = 0; k < 3; k++)
                        {
                            for (int m = 0; m < 3; m++)
                            {
                                stiffnessMatrix[3 * el.Nodes[i].Id + k, 3 *
el.Nodes[j].Id + m] += elStiffness[3 * i + k, 3 * j + m];
                            }
                        }
                    }
                }
```

```
                }
                return stiffnessMatrix;
            }


        }
    }


    namespace FEM.BLL.Data
    {
        /// <summary>
        /// Перечисление, хранящее тип узла
        /// </summary>
        public enum NodeLoadType
        {
            Force,
            Fixed
        }
        /// <summary>
        /// Класс, отвечающий за нагрузку узла
        /// </summary>
        public class NodeLoadDTO
        {
            public NodeLoadDTO()
            {

            }
            /// <summary>
            /// Создание класса нагрузки узла
            /// </summary>
            /// <param name="id">Идентификатор узла, к которому прилагается
нагрузка</param>
            /// <param name="type">Тип нагрузки узла</param>
            /// <param name="force">Вектор нагрузки на узел</param>
            public NodeLoadDTO(int id, NodeLoadType type, NodeForceVector force)
            {
                Id = id;
                NodeLoadType = type;
                ForceVector = force;
            }
            /// <summary>
            /// Идентификатор узла
            /// </summary>
            public int Id { get; set; }
            /// <summary>
            /// Тип нагрузки на узле
            /// </summary>
            public NodeLoadType NodeLoadType { get; set; }
            public NodeForceVector ForceVector { get; set; }
        }
    }


    namespace FEM.BLL.Data
    {
        /// <summary>
        /// Вектор узлового усилия
        /// </summary>
        public class NodeForceVector
        {
            /// <summary>
```

```
            /// Создание вектора узлового усилия
            /// </summary>
            /// <param name="x">Усилие по координате X</param>
            /// <param name="y">Усилие по координате Z</param>
            /// <param name="z">Усилие по координате Y</param>
            public NodeForceVector(double x, double y, double z)
            {
                X = x;
                Y = y;
                Z = z;
            }
            /// <summary>
            /// Усилие по координате X
            /// </summary>
            public double X { get; set; }
            /// <summary>
            /// Усилие по координате Y
            /// </summary>
            public double Y { get; set; }
            /// <summary>
            /// Усилие по координате Z
            /// </summary>
            public double Z { get; set; }
        }
    }
```

```
    using Accord.Math;
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Linq;
    using System.Windows.Media.Media3D;

    namespace FEM.BLL.Data.Elements
    {
        /// <summary>
        /// Линейный тетраэдральный конечный элемент
        /// </summary>
        public class LinearTetrahedronDTO : ElementDTO
        {
            /// <summary>
            /// Создание линейного тетраэдрального конечного элемента
            /// </summary>
            /// <param name="id">Идентификатор конечного элемента</param>
            /// <param name="nodes">Узлы конечного элемента</param>
            /// <param name="color">Цвет конечного элемента</param>
            public LinearTetrahedronDTO(int id, NodeDTO[] nodes, Color color) :
base(id, nodes, color)
            {
                if (nodes.Length != 4)
                {
                    throw new Exception("У линейного тетраэдра должно быть 4 различных
узла");
                }
            }

            /// <summary>
            /// Объём конечного элемента
            /// </summary>
            public override double Volume
            {
                get
```

```
            {
                return Math.Abs(Matrix.Determinant(CoordinateMatrix) / 6);
            }
        }

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

        /// <summary>
        /// Координатная матрица конечного элемента
        /// </summary>
        private double[,] CoordinateMatrix
        {
            get
            {
                return new double[,]
                {
                    {1, Nodes[0].X, Nodes[0].Y, Nodes[0].Z },
                    {1, Nodes[1].X, Nodes[1].Y, Nodes[1].Z },
                    {1, Nodes[2].X, Nodes[2].Y, Nodes[2].Z },
                    {1, Nodes[3].X, Nodes[3].Y, Nodes[3].Z }
                };
            }
        }

        /// <summary>
        /// Геометрическая матрица конечного элемента [D]
        /// </summary>
        public override 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());
            }
        }

        public override Point3D CenterPoint
        {
            get
            {
                Point3D center = new Point3D(0, 0, 0);
                foreach (var node in Nodes)
                {
                    center.Offset(node.X, node.Y, node.Z);
```

```
        }
        double coef = 1.0 / (Nodes.Length + 1);
        center.X *= coef;
        center.Y *= coef;
        center.Z *= coef;
        return center;
    }
}

/// <summary>
/// Вычисляет обратную матрицу A для вычисления геометрической матрицы
/// </summary>
/// <returns>Обратная матрица A</returns>
public override 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].X, Nodes[i].Y, Nodes[i].Z },
        { Nodes[j].X, Nodes[j].Y, Nodes[j].Z },
        { Nodes[m].X, Nodes[m].Y, Nodes[m].Z }
    });
    double B = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        {1, Nodes[i].Y, Nodes[i].Z },
        {1, Nodes[j].Y, Nodes[j].Z },
        {1, Nodes[m].Y, Nodes[m].Z }
    });
    double C = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        { Nodes[i].X, 1,Nodes[i].Z },
        { Nodes[j].X, 1,Nodes[j].Z },
        { Nodes[m].X, 1,Nodes[m].Z }
    });
    double D = Math.Pow(-1, index) * Matrix.Determinant(new double[,]
    {
        { Nodes[i].X, Nodes[i].Y, 1 },
        { Nodes[j].X, Nodes[j].Y, 1 },
        { Nodes[m].X, Nodes[m].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 override double[,] GetElasticMatrix(MaterialDTO material)
            {
                double ro = material.ShearModulus * 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.ShearModulus, 0, 0 },
                    {0, 0, 0, 0, material.ShearModulus, 0 },
                    {0, 0, 0, 0, 0, material.ShearModulus }
                };
            }

            /// <summary>
            /// Метод копирует конечный элемент
            /// </summary>
            /// <param name="nodes">Узлы конечно-элементной сетки, в котором находится
этот конечный элемент</param>
            /// <returns>Копия текущего конечного элемента</returns>
            public override ElementDTO Copy(IList<NodeDTO> nodes)
            {
                var copyNodes = Nodes.Select(node => nodes.First(n => n.Id ==
node.Id));
                return new LinearTetrahedronDTO(Id, copyNodes.ToArray(), Color);
            }
        }
    }
```

```
    using FEM.BLL.Data;
    using System;
    using System.Collections.Generic;

    namespace FEM.BLL.MeshGenerators.Figures
    {
        public class Prism
        {
            /// <summary>
            /// Количество сторон призмы
            /// </summary>
            public int SidesCount { get; set; }
            /// <summary>
            /// Длина стороны призмы
            /// </summary>
            public double SidesLength { get; set; }
            /// <summary>
```

```
        /// Высота призмы
        /// </summary>
        public double Height { get; set; }
        /// <summary>
        /// Радиус описанной окружности
        /// </summary>
        public double Radius { get; set; }

        /// <summary>
        /// Создание n-угольной призмы
        /// </summary>
        /// <param name="sidesCount">Количество сторон призмы</param>
        /// <param name="sidesLength">Длина стороны призмы</param>
        /// <param name="height">Высота призмы</param>
        public Prism(int sidesCount, double sidesLength, double height)
        {
            SidesCount = sidesCount;
            SidesLength = sidesLength;
            Height = height;

            double angle = 360.0 / sidesCount;
            Radius = SidesLength / (2 * Math.Sin(Math.PI * angle / 180.0 / 2));
        }

        /// <summary>
        /// Создание 16 угольной призмы, заменяющей собой цилиндр
        /// </summary>
        /// <param name="radius">Радиус описанной вокруг призмы окружности</param>
        /// <param name="height">Высота призмы</param>
        public Prism(double radius, double height)
        {
            SidesCount = 16;

            //Вычисление радиуса призмы
            Radius = Math.Sqrt(2 * Math.PI * (Math.Pow(radius, 2) / (SidesCount *
Math.Sin(Math.PI * (360.0 / SidesCount) / 180.0))));

            SidesLength = Math.Sqrt(2 * Math.Pow(Radius, 2) - 4 * Radius *
Math.Cos(360.0 / SidesCount));
            Height = height;
        }

        /// <summary>
        /// Разбивает призму на узлы и треугольные призмы
        /// </summary>
        /// <returns>Узлы и треугольные призмы</returns>
        public (IList<TrianglePrism>, IList<NodeDTO>) GetTrianglePrismsWithNodes()
        {
            IList<TrianglePrism> prisms = new List<TrianglePrism>();
            IList<NodeDTO> nodes = new List<NodeDTO>();

            double angle = 360.0 / SidesCount;

            // Центральные узлы призмы
            nodes.Add(new NodeDTO(0, 0, 0, 0));
            nodes.Add(new NodeDTO(1, 0, Height / 4, 0));
            nodes.Add(new NodeDTO(2, 0, Height / 2, 0));
            nodes.Add(new NodeDTO(3, 0, -Height / 4, 0));
            nodes.Add(new NodeDTO(4, 0, -Height / 2, 0));

            // Узлы начала отсчёта призмы
            nodes.Add(new NodeDTO(5, Radius, 0, 0));
            nodes.Add(new NodeDTO(6, Radius, Height / 4, 0));
            nodes.Add(new NodeDTO(7, Radius, Height / 2, 0));
```

```
                    nodes.Add(new NodeDTO(8, Radius, -Height / 4, 0));
                    nodes.Add(new NodeDTO(9, Radius, -Height / 2, 0));

                    double currentAngle = angle;
                    int currentNodeId = 10;
                    int currentPrismId = 0;

                    while (currentAngle < 360)
                    {
                        double x = Radius * Math.Cos(Math.PI * currentAngle / 180.0);
                        double z = Radius * Math.Sin(Math.PI * currentAngle / 180.0);

                        nodes.Add(new NodeDTO(currentNodeId, x, 0, z));
                        nodes.Add(new NodeDTO(currentNodeId + 1, x, Height / 4, z));
                        nodes.Add(new NodeDTO(currentNodeId + 2, x, Height / 2, z));
                        nodes.Add(new NodeDTO(currentNodeId + 3, x, -Height / 4, z));
                        nodes.Add(new NodeDTO(currentNodeId + 4, x, -Height / 2, z));

                        prisms.Add(new TrianglePrism(currentPrismId, new NodeDTO[]
{ nodes[0], nodes[1], nodes[currentNodeId - 5], nodes[currentNodeId - 4],
nodes[currentNodeId], nodes[currentNodeId + 1] }));
                        prisms.Add(new TrianglePrism(currentPrismId + 1, new NodeDTO[]
{ nodes[0], nodes[3], nodes[currentNodeId - 5], nodes[currentNodeId - 2],
nodes[currentNodeId], nodes[currentNodeId + 3] }));
                        prisms.Add(new TrianglePrism(currentPrismId + 2, new NodeDTO[]
{ nodes[1], nodes[2], nodes[currentNodeId - 4], nodes[currentNodeId - 3],
nodes[currentNodeId + 1], nodes[currentNodeId + 2] }));
                        prisms.Add(new TrianglePrism(currentPrismId + 3, new NodeDTO[]
{ nodes[3], nodes[4], nodes[currentNodeId - 2], nodes[currentNodeId - 1],
nodes[currentNodeId + 3], nodes[currentNodeId + 4] }));

                        currentNodeId += 5;
                        currentPrismId += 4;
                        currentAngle += angle;
                    }

                    prisms.Add(new TrianglePrism(currentPrismId, new NodeDTO[] { nodes[0],
nodes[1], nodes[currentNodeId - 5], nodes[currentNodeId - 4], nodes[5], nodes[6] }));
                    prisms.Add(new TrianglePrism(currentPrismId + 1, new NodeDTO[]
{ nodes[0], nodes[3], nodes[currentNodeId - 5], nodes[currentNodeId - 2], nodes[5],
nodes[8] }));
                    prisms.Add(new TrianglePrism(currentPrismId + 2, new NodeDTO[]
{ nodes[1], nodes[2], nodes[currentNodeId - 4], nodes[currentNodeId - 3], nodes[6],
nodes[7] }));
                    prisms.Add(new TrianglePrism(currentPrismId + 3, new NodeDTO[]
{ nodes[3], nodes[4], nodes[currentNodeId - 2], nodes[currentNodeId - 1], nodes[8],
nodes[9] }));

                    return (prisms, nodes);
                }
            }
        }



        using FEM.BLL.Data;
        using FEM.BLL.Data.Elements;
        using System.Collections.Generic;
        using System.Drawing;

        namespace FEM.BLL.MeshGenerators.Figures
        {
            /// <summary>
            /// Класс треугольной призмы
```

```
        /// </summary>
        public class TrianglePrism
        {
            int _id;
            NodeDTO[] _nodes;
            /// <summary>
            /// Создание класса треугольной призмы
            /// </summary>
            /// <param name="id">Идентификатор призмы</param>
            /// <param name="nodes">Узлы треугольной призмы</param>
            public TrianglePrism(int id, NodeDTO[] nodes)
            {
                _id = id;
                _nodes = nodes;
            }

            /// <summary>
            /// Разбиение призмы на линейные тетраэдры
            /// </summary>
            /// <returns>Список линейных тетраэдров из которых состоит
призма</returns>
            public IList<LinearTetrahedronDTO> GetLinearTetrahedrons()
            {
                IList<LinearTetrahedronDTO> tetras = new List<LinearTetrahedronDTO>();
                tetras.Add(new LinearTetrahedronDTO(_id * 3, new NodeDTO[] { _nodes[0],
_nodes[2], _nodes[3], _nodes[4] }, Color.Gray));
                tetras.Add(new LinearTetrahedronDTO(_id * 3, new NodeDTO[] { _nodes[1],
_nodes[4], _nodes[3], _nodes[5] }, Color.Gray));
                tetras.Add(new LinearTetrahedronDTO(_id * 3, new NodeDTO[] { _nodes[0],
_nodes[4], _nodes[3], _nodes[1] }, Color.Gray));
                return tetras;
            }
        }
    }



    using FEM.BLL.Data;
    using FEM.BLL.Data.Elements;
    using FEM.BLL.MeshGenerators.Figures;
    using System.Collections.Generic;

    namespace FEM.BLL.Generators
    {
        /// <summary>
        /// Класс, генерирующий различные сетки
        /// </summary>
        public class MeshGenerator
        {
            /// <summary>
            /// Получает копию текущей сетки, в которой узлы перемещены
            /// </summary>
            /// <param name="nodesDisplacement">Вектор узловых перемещений</param>
            /// <returns>Копия текущей сетки, в которой узлы перемещены</returns>
            public MeshDTO GenerateMovedMesh(MeshDTO mesh, double[] nodesDisplacement)
            {
                MeshDTO copy = mesh.Copy();
                foreach (var node in copy.Nodes)
                {
                    node.Move(nodesDisplacement[node.Id * 3],
nodesDisplacement[node.Id * 3 + 1], nodesDisplacement[node.Id * 3 + 2]);
                }
                return copy;
            }
```

```
            /// <summary>
            /// Генерация сетки для цилиндра
            /// </summary>
            /// <param name="radius">Радиус цилиндра</param>
            /// <param name="height">Высота цилиндра</param>
            /// <returns>Сетка для цилиндра</returns>
            public MeshDTO GetCylinderMesh(double radius, double height)
            {
                Prism cylinder = new Prism(radius, height);
                (IList<TrianglePrism> prisms, IList<NodeDTO> nodes) =
cylinder.GetTrianglePrismsWithNodes();
                IList<LinearTetrahedronDTO> tetrahedrons = new
List<LinearTetrahedronDTO>();
                foreach (var prism in prisms)
                {
                    foreach (var tetra in prism.GetLinearTetrahedrons())
                    {
                        tetrahedrons.Add(tetra);
                    }
                }
                return new MeshDTO(nodes, new List<ElementDTO>(tetrahedrons));
            }

            /// <summary>
            /// Генерация сетки для многоугольной призмы
            /// </summary>
            /// <param name="sidesCount">Количество сторон призмы</param>
            /// <param name="sidesLength">Длина стороны призмы</param>
            /// <param name="height">Высота призмы</param>
            /// <returns></returns>
            public MeshDTO GetPolygonMesh(int sidesCount, double sidesLength, double
height)
            {
                Prism poli = new Prism(sidesCount, sidesLength, height);
                (IList<TrianglePrism> prisms, IList<NodeDTO> nodes) =
poli.GetTrianglePrismsWithNodes();
                IList<LinearTetrahedronDTO> tetrahedrons = new
List<LinearTetrahedronDTO>();
                foreach (var prism in prisms)
                {
                    foreach (var tetra in prism.GetLinearTetrahedrons())
                    {
                        tetrahedrons.Add(tetra);
                    }
                }
                return new MeshDTO(nodes, new List<ElementDTO>(tetrahedrons));
            }
        }
    }



    using FEM.BLL.Data;
    using System;
    using System.Collections.Generic;
    using System.Linq;

    namespace FEM.BLL.Generators
    {
        /// <summary>
        /// Генератор узловых нагрузок
        /// </summary>
        public class NodeLoadsGenerator
```

```
        {
            public enum Face
            {
                Top,
                Bottom,
                Left,
                Right,
                Front,
                Back
            }

            /// <summary>
            /// Генерация списка узловых нагрузок, как если бы на деталь давил пресс
            /// </summary>
            /// <param name="mesh">Сетка для которой необходимо сформировать вектор
узловых нагрузок</param>
            /// <param name="pressureValue">Давление пресса в тоннах</param>
            /// <param name="area">Площадь поверхности детали</param>
            /// <returns>Список узловых нагрузок детали</returns>
            public IList<NodeLoadDTO> GeneratePressLoads(MeshDTO mesh, double
pressureValue, double area)
            {
                double pascales = pressureValue * 1000 * 9.8 / area;

                var topNodes = mesh.Nodes.Where(node => node.Y == mesh.Nodes.Select(n
=> n.Y).Max());
                var bottomNodes = mesh.Nodes.Where(node => node.Y ==
mesh.Nodes.Select(n => n.Y).Min());

                var oneNodePascal = pascales / topNodes.Count();

                IList<NodeLoadDTO> loads = new List<NodeLoadDTO>();

                foreach (var node in topNodes)
                {
                    loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force, new
NodeForceVector(0, -oneNodePascal, 0)));
                }

                foreach (var node in bottomNodes)
                {
                    loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed, new
NodeForceVector(0, 0, 0)));
                }

                return loads;
            }

            /// <summary>
            /// Генерирует силовые нагрузки для определенной поверхности сетки
            /// </summary>
            /// <param name="face">Поверхность сетки</param>
            /// <param name="mesh">Сетка</param>
            /// <param name="forceVector">Вектор узловых нагрузок</param>
            /// <param name="isSurfaceForce">Является ли сила поверхностной или
действует на каждый узел</param>
            /// <returns>Список узловых нагрузок детали</returns>
            public IList<NodeLoadDTO> GenerateForceLoads(Face face, MeshDTO mesh,
NodeForceVector forceVector, bool isSurfaceForce = true)
            {
                IList<NodeLoadDTO> loads = new List<NodeLoadDTO>();
                int nodesCount;
                switch (face)
                {
```

```
                        case Face.Top:
                            var topNodes = mesh.Nodes.Where(node => node.Y ==
mesh.Nodes.Select(n => n.Y).Max());
                            nodesCount = topNodes.Count();
                            if (isSurfaceForce)
                            {
                                forceVector = new NodeForceVector(forceVector.X /
nodesCount, forceVector.Y / nodesCount, forceVector.Z / nodesCount);
                            }
                            foreach (var node in topNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force,
forceVector));
                            }
                            return loads;
                        case Face.Bottom:
                            var bottomNodes = mesh.Nodes.Where(node => node.Y ==
mesh.Nodes.Select(n => n.Y).Min());
                            nodesCount = bottomNodes.Count();
                            if (isSurfaceForce)
                            {
                                forceVector = new NodeForceVector(forceVector.X /
nodesCount, forceVector.Y / nodesCount, forceVector.Z / nodesCount);
                            }
                            foreach (var node in bottomNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force,
forceVector));
                            }
                            return loads;
                        case Face.Left:
                            var leftNodes = mesh.Nodes.Where(node => node.X ==
mesh.Nodes.Select(n => n.X).Min());
                            nodesCount = leftNodes.Count();
                            if (isSurfaceForce)
                            {
                                forceVector = new NodeForceVector(forceVector.X /
nodesCount, forceVector.Y / nodesCount, forceVector.Z / nodesCount);
                            }
                            foreach (var node in leftNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force,
forceVector));
                            }
                            return loads;
                        case Face.Right:
                            var rightNodes = mesh.Nodes.Where(node => node.X ==
mesh.Nodes.Select(n => n.X).Max());
                            nodesCount = rightNodes.Count();
                            if (isSurfaceForce)
                            {
                                forceVector = new NodeForceVector(forceVector.X /
nodesCount, forceVector.Y / nodesCount, forceVector.Z / nodesCount);
                            }
                            foreach (var node in rightNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force,
forceVector));
                            }
                            return loads;
                        case Face.Front:
                            var frontNodes = mesh.Nodes.Where(node => node.Z ==
mesh.Nodes.Select(n => n.Z).Max());
                            nodesCount = frontNodes.Count();
```

```
                            if (isSurfaceForce)
                            {
                                forceVector = new NodeForceVector(forceVector.X /
nodesCount, forceVector.Y / nodesCount, forceVector.Z / nodesCount);
                            }
                            foreach (var node in frontNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force,
forceVector));
                            }
                            return loads;
                        case Face.Back:
                            var backNodes = mesh.Nodes.Where(node => node.Z ==
mesh.Nodes.Select(n => n.Z).Min());
                            nodesCount = backNodes.Count();
                            if (isSurfaceForce)
                            {
                                forceVector = new NodeForceVector(forceVector.X /
nodesCount, forceVector.Y / nodesCount, forceVector.Z / nodesCount);
                            }
                            foreach (var node in backNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Force,
forceVector));
                            }
                            return loads;
                        default:
                            throw new Exception("Данная позиция не реализована");
                    }
                }

                /// <summary>
                /// Генерирует закрепления для определенной поверхности сетки
                /// </summary>
                /// <param name="face">Поверхность сетки</param>
                /// <param name="mesh">Сетка</param>
                /// <returns>Список закреплений детали</returns>
                public IList<NodeLoadDTO> GenerateFixationLoads(Face face, MeshDTO mesh)
                {
                    IList<NodeLoadDTO> loads = new List<NodeLoadDTO>();
                    switch (face)
                    {
                        case Face.Top:
                            var topNodes = mesh.Nodes.Where(node => node.Y ==
mesh.Nodes.Select(n => n.Y).Max());
                            foreach (var node in topNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed,
null));
                            }
                            return loads;
                        case Face.Bottom:
                            var bottomNodes = mesh.Nodes.Where(node => node.Y ==
mesh.Nodes.Select(n => n.Y).Min());
                            foreach (var node in bottomNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed,
null));
                            }
                            return loads;
                        case Face.Left:
                            var leftNodes = mesh.Nodes.Where(node => node.X ==
mesh.Nodes.Select(n => n.X).Min());
                            foreach (var node in leftNodes)
```

```
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed,
null));
                            }
                            return loads;
                        case Face.Right:
                            var rightNodes = mesh.Nodes.Where(node => node.X ==
mesh.Nodes.Select(n => n.X).Max());
                            foreach (var node in rightNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed,
null));
                            }
                            return loads;
                        case Face.Front:
                            var frontNodes = mesh.Nodes.Where(node => node.Z ==
mesh.Nodes.Select(n => n.Z).Max());
                            foreach (var node in frontNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed,
null));
                            }
                            return loads;
                        case Face.Back:
                            var backNodes = mesh.Nodes.Where(node => node.Z ==
mesh.Nodes.Select(n => n.Z).Min());
                            foreach (var node in backNodes)
                            {
                                loads.Add(new NodeLoadDTO(node.Id, NodeLoadType.Fixed,
null));
                            }
                            return loads;
                        default:
                            throw new Exception("Данная позиция не реализована");
                    }
                }
            }
        }



        using FEM.BLL.Data;
        using FEM.BLL.Data.Elements;
        using FEM.BLL.Tools;
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Windows.Media;
        using System.Windows.Media.Media3D;

        namespace FEM.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(MeshDTO mesh, MaterialGroup material,
IList<NodeLoadDTO> loads)
        {
            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 =
GetElementMaterial(mesh.Elements[i], material, loads);
                geometryModel3D.Material = elementMaterial;
                geometryModel3D.BackMaterial = elementMaterial;
                modelGeometries[i] = geometryModel3D;
            }

            model.Children = new Model3DCollection(modelGeometries);

            return model;
        }

        /// <summary>
        /// Вычисляет модель с градиентным окрасом. Градиент показывает изменения
характеристики элементов 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(MeshDTO 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 GetElementMaterial(ElementDTO element, MaterialGroup
defaultMaterial, IList<NodeLoadDTO> loads)
            {
                foreach (var node in element.Nodes)
                {
                    if (loads.Select(load => load.Id).Contains(node.Id))
                    {
                        var load = loads.First(load => load.Id == node.Id);
                        MaterialGroup material = new MaterialGroup();
                        System.Windows.Media.Media3D.Material[] materials = new
System.Windows.Media.Media3D.Material[1];
                        if (load.NodeLoadType == Data.NodeLoadType.Force)
                        {
                            materials[0] = new DiffuseMaterial(new
SolidColorBrush(System.Windows.Media.Color.FromArgb(0x99, 0xe8, 0x54, 0x8c)));
                        }
                        else
                        {
                            materials[0] = new DiffuseMaterial(new
SolidColorBrush(System.Windows.Media.Color.FromArgb(0x99, 0x53, 0xd2, 0xf5)));
                        }
                        material.Children = new MaterialCollection(materials);
                        return material;
                    }
                }
                return defaultMaterial;
            }

            private MeshGeometry3D GetGeometryFromElement(ElementDTO element)
            {
                MeshGeometry3D triangleGeometry3D = new MeshGeometry3D();
                if (element is LinearTetrahedronDTO)
                {
                    Point3D[] positionsArray = new Point3D[element.Nodes.Length];
                    for (int i = 0; i < element.Nodes.Length; i++)
                    {
```

```
                        positionsArray[i] = new Point3D(element.Nodes[i].X,
element.Nodes[i].Y, element.Nodes[i].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;
                }
                else
                {
                    throw new Exception("Геометрия такого типа конечного элемента не
реализована");
                }
                return triangleGeometry3D;
            }
        }
    }



    using Accord.Math;
    using FEM.BLL.Data;
    using FEM.BLL.Tools;
    using System;
    using System.Collections.Generic;
    using System.Linq;

    namespace FEM.BLL.Solvers
    {
        /// <summary>
        /// Класс, решающий задачи упругости методом конечных элементов
        /// </summary>
        public class SimpleMeshSolver : ISolver
        {
            /// <summary>
            /// Вычисляет перемещения узлов в сетке
            /// </summary>
            /// <param name="mesh">Конечно-элементная сетка, перемещения узлов которых
вычисляются</param>
            /// <param name="material">Материал, из которого состоит деталь, на
которую наложена эта сетка</param>
            /// <param name="nodeLoads">Узловые нагрузки конечно-элементной
сетки</param>
            /// <returns></returns>
            public double[] GetNodesDisplacement(MeshDTO mesh, MaterialDTO material,
IList<NodeLoadDTO> nodeLoads)
            {
                var globalStiffnessWithCondisplacements =
GetGlobalStiffnessWithConstrains(mesh, material, nodeLoads);

                var nodeForces = GetNodeForces(mesh, nodeLoads);

                double[] displacement = new
SLAESolver().Gauss(globalStiffnessWithCondisplacements, nodeForces);
```

```
                return displacement;
            }

            /// <summary>
            /// Метод, возвращающий перемещения конечных элементов после решения
относительно исходной сетки
            /// </summary>
            /// <param name="originalMesh">Исходная сетка</param>
            /// <param name="nodesDisplacement">Перемещения всех узлов сетки</param>
            /// <returns>Вектор перемещений элементов</returns>
            public double[] GetElementsDisplacement(MeshDTO originalMesh, double[]
nodesDisplacement)
            {
                IList<double> displacements = new List<double>();
                foreach (var element in originalMesh.Elements)
                {
                    double xDisplacement = 0, yDisplacement = 0, zDisplacement = 0;
                    foreach (var node in element.Nodes)
                    {
                        xDisplacement += nodesDisplacement[node.Id * 3];
                        yDisplacement += nodesDisplacement[node.Id * 3 + 1];
                        zDisplacement += nodesDisplacement[node.Id * 3 + 2];
                    }

                    xDisplacement /= 4;
                    yDisplacement /= 4;
                    zDisplacement /= 4;

                    displacements.Add(Math.Sqrt(Math.Pow(xDisplacement, 2) +
Math.Pow(yDisplacement, 2) + Math.Pow(zDisplacement, 2)) / 7);
                }
                return displacements.ToArray();
            }

            /// <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(MeshDTO 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[] currentStrain =
element.GeometricMatrix.Dot(element.GetInversedA()).Dot(currentDisplacement);

                    strain.Add(currentStrain[1] / 100);
```

```
                }
                return strain.ToArray();
            }
            /// <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(MeshDTO originalMesh, double[]
nodesDisplacement, MaterialDTO 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[] currentStresses =
element.GetElasticMatrix(material).Dot(element.GeometricMatrix).Dot(element.GetInversedA()
).Dot(currentDisplacement);

                    stresses.Add(currentStresses[1] / 500);
                }
                return stresses.ToArray();

            }

            /// <summary>
            /// Вычисляет вектор узловых усилий конечно-элементной сетки
            /// </summary>
            /// <param name="mesh">Конечно-элементная сетка, вектор узловых усилий
которой необходимо вычислить</param>
            /// <param name="nodeLoads">Узловые нагрузки для сетки</param>
            /// <returns>Вектор узловых усилий конечно-элементной сетки</returns>
            private double[] GetNodeForces(MeshDTO mesh, IList<NodeLoadDTO> nodeLoads)
            {
                double[] forces = new double[mesh.Nodes.Count * 3];
                foreach (var load in nodeLoads)
                {
                    if (load.NodeLoadType == NodeLoadType.Force)
                    {
                        forces[load.Id * 3] = load.ForceVector.X;
                        forces[load.Id * 3 + 1] = load.ForceVector.Y;
                        forces[load.Id * 3 + 2] = load.ForceVector.Z;
                    }
                }
                return forces;
            }

            /// <summary>
            /// Вычисляет матрицу жесткости детали с наложенными ограничениями в виде
закреплений
```

```
            /// </summary>
            /// <param name="mesh">Конечно-элементная сетка детали</param>
            /// <param name="material">Материал детали</param>
            /// <param name="nodeLoads">Узловые нагрузки детали</param>
            /// <returns>Матрица жесткости детали с наложенными ограничениями в виде
закреплений</returns>
            private double[,] GetGlobalStiffnessWithConstrains(MeshDTO mesh,
MaterialDTO material, IList<NodeLoadDTO> nodeLoads)
            {
                var nodesCount = mesh.Nodes.Count;
                var globalStiffnessMatrix = mesh.GetStiffnessMatrix(material);
                foreach (var load in nodeLoads)
                {
                    if (load.NodeLoadType == NodeLoadType.Fixed)
                    {
                        for (int i = 0; i < nodesCount * 3; i++)
                        {
                            globalStiffnessMatrix[3 * load.Id, i] = 0;
                            globalStiffnessMatrix[3 * load.Id + 1, i] = 0;
                            globalStiffnessMatrix[3 * load.Id + 2, i] = 0;
                            globalStiffnessMatrix[i, 3 * load.Id] = 0;
                            globalStiffnessMatrix[i, 3 * load.Id + 1] = 0;
                            globalStiffnessMatrix[i, 3 * load.Id + 2] = 0;
                        }
                        globalStiffnessMatrix[3 * load.Id, 3 * load.Id] = 1;
                        globalStiffnessMatrix[3 * load.Id + 1, 3 * load.Id + 1] = 1;
                        globalStiffnessMatrix[3 * load.Id + 2, 3 * load.Id + 2] = 1;
                    }
                }
                return globalStiffnessMatrix;
            }
        }
    }
```

```
    using System;

    namespace FEM.BLL.Tools
    {
        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 исходной 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);
            }
        }
    }



    using System;

    namespace FEM.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);
            }
        }
    }
```

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

        namespace FEM.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, int stepCount)
                {
                    MinValue = minValue;
                    MaxValue = maxValue;

                    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 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)}";
                }
            }
        }
    }


    using FEM.BLL.DI;
    using Microsoft.Extensions.Configuration;
    using Ninject;
    using Ninject.Modules;
    using System.IO;
    using System.Windows;

    namespace FEM.WPFGUI
    {
        /// <summary>
        /// Interaction logic for App.xaml
        /// </summary>
        public partial class App : Application
        {
            private IKernel _container;

            protected override void OnStartup(StartupEventArgs e)
            {
                base.OnStartup(e);
                ConfigureContainer();
                ComposeObjects();
                Current.MainWindow.Show();
            }

            private void ConfigureContainer()
            {
                IConfigurationRoot config = GetConfig();
                //BLL dependencies
                NinjectModule geometryMakerModule = new SimpleModelMakerModule();
                NinjectModule meshLoaderModule = new
MeshTxtLoaderModule(config["NodesTxt"], config["ElementsTxt"]);
                NinjectModule nodeLoadsLoaderModule = new
NodeLoadsCsvLoaderModule(config["NodeLoadsCsv"]);
                NinjectModule materialsLoaderModule = new
MaterialsLoaderModule(config["MaterialsCsv"]);
                NinjectModule solver = new SimpleSolverModule();

                _container = new StandardKernel(geometryMakerModule, meshLoaderModule,
nodeLoadsLoaderModule, materialsLoaderModule, solver);
            }

            private IConfigurationRoot GetConfig()
            {
                var curDir = Directory.GetCurrentDirectory();
                var baseDir = Directory.GetParent(curDir).Parent.Parent;

                var config = new ConfigurationBuilder()
                    .AddJsonFile(@$"{baseDir}\config.json")
                    .Build();

                return config;
            }
```

```
            private void ComposeObjects()
            {
                Current.MainWindow = _container.Get<MainWindow>();
                Current.MainWindow.Title = "FEM";
            }
        }
    }



    using FEM.BLL.Data;
    using FEM.BLL.UIElements;
    using System;
    using System.Windows;
    using System.Windows.Controls;
    using System.Windows.Input;
    using System.Windows.Media;

    namespace FEM.WPFGUI
    {
        /// <summary>
        /// Interaction logic for MainWindow.xaml
        /// </summary>
        public partial class MainWindow : Window
        {
            private readonly MainViewModel _model;
            public MainWindow(MainViewModel model)
            {
                InitializeComponent();
                _model = model;
                InitializeMaterials();
                InitializeGeometryModel();
            }

            private void InitializeMaterials()
            {
                try
                {
                    foreach (var material in _model.Materials)
                    {
                        MaterialsBox.Items.Add(material);
                    }
                    MaterialsBox.SelectedItem = _model.SelectedMaterial;
                    MaterialsBox.DisplayMemberPath = "Name";
                }
                catch (Exception exception)
                {
                    MessageBox.Show($"Ошибка загрузки материалов.
{exception.Message}");
                }
            }

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

```
            }
        }

        private void PressureApplyButton_Click(object sender, RoutedEventArgs e)
        {

            try
            {
                _model.GeneratePressLoads(double.Parse(PressureBox.Text));
                MeshModel.Content = _model.OriginalMeshModel;
                HideGradientScale();
                RefreshSolutionButton();
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка применения сил к детали.
{exception.Message}");
            }
        }

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

        private void MaterialApplyButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                _model.CurrentMaterial = _model.SelectedMaterial;
                MeshModel.Content = _model.OriginalMeshModel;
                RefreshSolutionButton();
                HideGradientScale();
                MessageBox.Show($"Материал успешно изменён");
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка применения материала.
{exception.Message}");
            }
        }

        private void ShowOriginalDetailButton_Click(object sender, RoutedEventArgs
e)
        {
            try
            {
                MeshModel.Content = _model.OriginalMeshModel;
                HideGradientScale();
            }
            catch (Exception exception)
            {
```

```
                    MessageBox.Show($"Ошибка показа исходной детали.
{exception.Message}");
                }
            }

            private void ShowDisplacementButton_Click(object sender, RoutedEventArgs e)
            {
                try
                {
                    MeshModel.Content = _model.GetDisplacementMeshModel();
                    ShowGradientScale(_model.DisplacementGradientScale);
                }
                catch (Exception exception)
                {
                    MessageBox.Show($"Ошибка показа перемещений детали. Возможно вы
забыли произвести решение. {exception.Message}");
                }
            }

            private void ShowStressButton_Click(object sender, RoutedEventArgs e)
            {
                try
                {
                    MeshModel.Content = _model.GetStressMeshModel();
                    ShowGradientScale(_model.StressesGradientScale);
                }
                catch (Exception exception)
                {
                    MessageBox.Show($"Ошибка показа перемещения детали. Возможно вы
забыли произвести решение. {exception.Message}");
                }

            }

            private void ShowStrainButton_Click(object sender, RoutedEventArgs e)
            {
                try
                {
                    MeshModel.Content = _model.GetStrainMeshModel();
                    ShowGradientScale(_model.StrainGradientScale);
                }
                catch (Exception exception)
                {
                    MessageBox.Show($"Ошибка показа деформаций детали. Возможно вы
забыли произвести решение. {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)
```

```
        {
            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}");
            }
        }

        private void MaterialsBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            try
            {
                _model.SelectedMaterial = (MaterialDTO)MaterialsBox.SelectedItem;
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка изменения выбранного материала.
{exception.Message}");
            }
        }

        private void SolutionInfoButton_MouseEnter(object sender, MouseEventArgs e)
        {
            try
            {
                ((ToolTip)SolutionInfoButton.ToolTip).Content =
_model.GetSolutionInfo();
            }
            catch (Exception exception)
            {
                MessageBox.Show($"Ошибка просмотра информации о рещении.
{exception.Message}");
            }
        }

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

        private void RefreshSolutionButton(bool isSolved = false)
        {
```

```
                MakeSolutionButton.Background = isSolved ? Brushes.LightGreen : new
SolidColorBrush(Color.FromArgb(0xFF, 0xC5, 0x53, 0x78));
            }

            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;
            }
        }
    }



    using FEM.BLL.Data;
    using FEM.BLL.Generators;
    using FEM.BLL.ModelMakers;
    using FEM.BLL.Services;
    using FEM.BLL.Solvers;
    using FEM.BLL.UIElements;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Windows.Media;
    using System.Windows.Media.Media3D;

    namespace FEM.WPFGUI
    {
        public class MainViewModel
        {
            MeshService _meshService;
            MaterialService _materialService;
            NodeLoadService _nodeLoadService;
            IMeshModelMaker _geometryMaker;
            ISolver _solver;
            NodeLoadsGenerator _nodeLoadsGenerator;
            MeshGenerator _meshGenerator;
            public MainViewModel(MeshService meshService, MaterialService
materialService, NodeLoadService nodeLoadService,
                IMeshModelMaker geometryMaker, ISolver solver, NodeLoadsGenerator
nodeLoadsGenerator, MeshGenerator meshGenerator)
            {

                _meshService = meshService;
                _materialService = materialService;
                _nodeLoadService = nodeLoadService;
                _geometryMaker = geometryMaker;
                _solver = solver;
                _nodeLoadsGenerator = nodeLoadsGenerator;
                _meshGenerator = meshGenerator;
            }

            private MeshDTO _originalMesh;
            public MeshDTO OriginalMesh
            {
                get
```

```
                {
                    if (_originalMesh == null)
                    {
                        _originalMesh = _meshGenerator.GetCylinderMesh(0.2, 0.5);
                    }
                    return _originalMesh;
                }
            }

            public Model3DGroup OriginalMeshModel
            {
                get
                {
                    return _geometryMaker.GenerateGeometry(OriginalMesh, MaterialGroup,
NodeLoads);
                }
            }

            private Transform3DGroup _transforms;
            public Transform3DGroup Transforms
            {
                get
                {
                    if (_transforms == null)
                    {
                        _transforms = new Transform3DGroup();
                        _transforms.Children.Add(new RotateTransform3D(new
AxisAngleRotation3D(new Vector3D(0, 1, 0), 30)));
                        _transforms.Children.Add(new RotateTransform3D(new
AxisAngleRotation3D(new Vector3D(1, 0, 0), 30)));
                        _transforms.Children.Add(new ScaleTransform3D(7, 7, 7, 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;
            }

            private IList<MaterialDTO> _materials;
```

```
        public IList<MaterialDTO> Materials
        {
            get
            {
                if (_materials == null)
                {
                    _materials = _materialService.Materials;
                }
                return _materials;
            }
        }

        private MaterialDTO _selectedMaterial;
        public MaterialDTO SelectedMaterial
        {
            get
            {
                if (_selectedMaterial == null)
                {
                    _selectedMaterial = Materials.LastOrDefault();
                }
                return _selectedMaterial;
            }
            set
            {
                _selectedMaterial = value;
            }
        }

        private MaterialDTO _currentMaterial;
        public MaterialDTO CurrentMaterial
        {
            get
            {
                if (_currentMaterial == null)
                {
                    _currentMaterial = Materials.LastOrDefault();
                }
                return _currentMaterial;
            }
            set
            {
                _currentMaterial = value;
            }
        }

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

        public string GetSelectedMaterialInfo()
        {
            StringBuilder builder = new StringBuilder();
            builder.Append($"Материал {SelectedMaterial.Name}\n" +
                $"Модуль Юнга {SelectedMaterial.YoungModulus}\n" +
                $"Коэффициент Пуассона {SelectedMaterial.PuassonsCoefficient}");
```

```
                return builder.ToString();
            }

            public string GetSolutionInfo()
            {
                StringBuilder builder = new StringBuilder();
                builder.Append($"Сетка состоит из {OriginalMesh.Elements.Count}
элементов и {OriginalMesh.Nodes.Count} узлов\n" +
                    $"Текущий материал детали {CurrentMaterial.Name}");
                return builder.ToString();
            }

            private IList<NodeLoadDTO> _nodeLoads;
            public IList<NodeLoadDTO> NodeLoads
            {
                get
                {
                    if (_nodeLoads == null)
                    {
                        _nodeLoads = _nodeLoadService.NodeLoads;
                    }
                    return _nodeLoads;
                }
            }

            public void GeneratePressLoads(double value)
            {
                NodeLoads.Clear();
                foreach (var load in
_nodeLoadsGenerator.GeneratePressLoads(OriginalMesh, value, Math.PI * Math.Pow(0.2, 2)))
                {
                    NodeLoads.Add(load);
                }
            }

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

            public void Solve()
            {
                SolutionNodesDisplacement = _solver.GetNodesDisplacement(OriginalMesh,
CurrentMaterial, NodeLoads);
                SolutionMesh = _meshGenerator.GenerateMovedMesh(OriginalMesh,
SolutionNodesDisplacement);
            }

            public double[] SolutionElementsDisplacement { get; set; }

            public Model3DGroup GetDisplacementMeshModel()
            {
                SolutionElementsDisplacement =
_solver.GetElementsDisplacement(OriginalMesh, SolutionNodesDisplacement);
                return _geometryMaker.GenerateCharacteristicModel(SolutionMesh,
SolutionElementsDisplacement);
            }
```

```
                public GradientScale DisplacementGradientScale
                {
                    get
                    {
                        return new GradientScale(new BLL.Tools.GradientMaker(),
SolutionElementsDisplacement.Min(), SolutionElementsDisplacement.Max(), 10);
                    }
                }

                public double[] SolutionElementsStrain { get; set; }

                public Model3DGroup GetStrainMeshModel()
                {
                    SolutionElementsStrain = _solver.GetElementsStrain(OriginalMesh,
SolutionNodesDisplacement);
                    return _geometryMaker.GenerateCharacteristicModel(SolutionMesh,
SolutionElementsStrain);
                }

                public GradientScale StrainGradientScale
                {
                    get
                    {
                        return new GradientScale(new BLL.Tools.GradientMaker(),
SolutionElementsStrain.Min(), SolutionElementsStrain.Max(), 10);
                    }
                }

                public double[] SolutionElementsStresses { get; set; }

                public Model3DGroup GetStressMeshModel()
                {
                    SolutionElementsStresses = _solver.GetElementsStresses(OriginalMesh,
SolutionNodesDisplacement, CurrentMaterial);
                    return _geometryMaker.GenerateCharacteristicModel(SolutionMesh,
SolutionElementsStresses);
                }

                public GradientScale StressesGradientScale
                {
                    get
                    {
                        return new GradientScale(new BLL.Tools.GradientMaker(),
SolutionElementsStresses.Min(), SolutionElementsStresses.Max(), 10);
                    }
                }
            }
        }


        using FEM.DAL.Data;
        using System;
        using System.Collections.Generic;
        using System.IO;
        using System.Windows.Media;

        namespace FEM.DAL.Loaders
        {
            /// <summary>
            /// Класс для загрузки списка материалов из csv файла
            /// </summary>
            public class MaterialsCsvLoader : ILoader<IList<Material>>
            {
                readonly string _pathToCsv;
```

```
        readonly bool _hasHeader;
        readonly char _delimiter;
        /// <summary>
        /// Создание класса загрузки
        /// </summary>
        /// <param name="pathToCsv">Путь к csv файлу</param>
        /// <param name="hasHeader">Есть ли у данных в csv файле шапка</param>
        /// <param name="delimiter">Разделитель ячеек csv файла</param>
        public MaterialsCsvLoader(string pathToCsv, bool hasHeader = true, char
delimiter = ';')
        {
            _pathToCsv = pathToCsv;
            _hasHeader = hasHeader;
            _delimiter = delimiter;
        }
        /// <summary>
        /// Загрузка списка материалов из csv файла
        /// </summary>
        /// <returns>Список материалов</returns>
        public IList<Material> Load()
        {
            try
            {
                IList<Material> materials = new List<Material>();
                using (StreamReader sr = new StreamReader(_pathToCsv))
                {
                    if (_hasHeader)
                    {
                        sr.ReadLine();
                    }
                    string line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        string row = ProcessRow(line);
                        string[] values = row.Split(_delimiter);
                        (byte a, byte r, byte g, byte b) =
GetARGBFromString(values[2]);
                        materials.Add(new Material(int.Parse(values[0]),
                            values[1],
                            Color.FromArgb(a, r, g, b),
                            double.Parse(values[3]),
                            double.Parse(values[4]))
                            );
                    }
                }
                return materials;
            }
            catch (Exception ex)
            {
                throw new Exception($"Не удалось загрузить материалы.
{ex.Message}");
            }
        }

        private (byte, byte, byte, byte) GetARGBFromString(string color)
        {
            color = color.Trim();
            string a = color.Substring(0, 2);
            string r = color.Substring(2, 2);
            string g = color.Substring(4, 2);
            string b = color.Substring(6, 2);
            return (Convert.ToByte(a, 16), Convert.ToByte(r, 16), Convert.ToByte(g,
16), Convert.ToByte(b, 16));
        }
```

```
        /// <summary>
        /// Убирает лишние символы из строки csv файла
        /// </summary>
        /// <param name="line">Строка csv файла</param>
        /// <returns>Обработанная строка csv файла</returns>
        private string ProcessRow(string line)
        {
            return line.Trim(_delimiter).Replace('"', ' ');
        }
    }
}
```

```
using FEM.DAL.Data;
using FEM.DAL.Data.Elements;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

namespace FEM.DAL.Loaders
{
    /// <summary>
    /// Класс для загрузки сетки из txt файла
    /// </summary>
    public class MeshTxtLoader : ILoader<Mesh>
    {
        readonly string[] _elementTypes;
        readonly string _pathToNodes;
        readonly string _pathToElements;
        /// <summary>
        /// Создание загрузчика сетки
        /// </summary>
        /// <param name="pathToNodes">Путь к файлу с узлами</param>
        /// <param name="pathToElements">Путь к файлу с элементами</param>
        public MeshTxtLoader(string pathToNodes, string pathToElements)
        {
            _pathToNodes = pathToNodes;
            _pathToElements = pathToElements;
            _elementTypes = new string[]
            {
                "TETRA_4"
            };
        }
        /// <summary>
        /// Загрузка сетки из txt файла
        /// </summary>
        /// <returns>Конечно-элементная сетка</returns>
        public Mesh Load()
        {
            try
            {
                IList<Node> nodes = GetNodes();
                IList<Element> elements = GetElements();
                return new Mesh(nodes, elements);
            }
            catch (Exception ex)
            {
                throw new Exception($"Не удалось загрузить сетку {ex.Message}");
            }
        }
```

```
        /// <summary>
        /// Получает узлы конечно-элементной сетки из файла с узлами
        /// </summary>
        /// <returns>Узлы конечно-элементной сетки</returns>
        private IList<Node> GetNodes()
        {
            IList<Node> nodes = new List<Node>();
            using (StreamReader sr = new StreamReader(_pathToNodes))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    string rowType = line.Split(' ')[0];
                    if (rowType == "node")
                    {
                        nodes.Add(GetNodeFromLine(line));
                    }
                }
            }
            return nodes;
        }

        /// <summary>
        /// Получает элементы конечно-элементной сетки из файла с узлами
        /// </summary>
        /// <returns>Элементы конечно-элементной сетки</returns>
        public IList<Element> GetElements()
        {
            IList<Element> elements = new List<Element>();
            using (StreamReader sr = new StreamReader(_pathToElements))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    string rowType = line.Split(' ')[0];
                    if (_elementTypes.Contains(rowType))
                    {
                        elements.Add(GetElementFromLine(line));
                    }
                }
            }
            return elements;
        }

        /// <summary>
        /// Преобразует строку из файла с узлами в класс, содержащий информацию об
узле
        /// </summary>
        /// <param name="line">Строка из файла с узлами</param>
        /// <returns>Класс узла конечно-элементной сетки</returns>
        private Node GetNodeFromLine(string line)
        {
            Regex regex = new Regex(@"internal\s(?<Id>\d+).*\s(?<X>-
?\d+(\.\d+)?(e-\d+)?)\s(?<Y>-?\d+(\.\d+)?(e-\d+)?)\s(?<Z>-?\d+(\.\d+)?(e-\d+)?)");
            Match match = regex.Match(line);
            int id = int.Parse(match.Groups["Id"].Value);
            double x = double.Parse(match.Groups["X"].Value.Replace(".", ","));
            double y = double.Parse(match.Groups["Y"].Value.Replace(".", ","));
            double z = double.Parse(match.Groups["Z"].Value.Replace(".", ","));
            return new Node(id, x, y, z);
        }

        /// <summary>
```

```
            /// Преобразует строку из файла с элементами в класс, содержащий
информацию об элементе, в зависимости от вида
            /// конечного элемента
            /// </summary>
            /// <param name="line">Строка из файла с элементами</param>
            /// <returns>Класс элемента конечно-элементной сетки</returns>
            private Element GetElementFromLine(string line)
            {
                string elementType = line.Split(' ')[0];
                Regex regex;
                int id;
                int[] nodeIds;
                switch (elementType)
                {
                    case "TETRA_4":
                        regex = new
Regex(@"internal\s(?<Id>\d+).*\s(?<Node1>\d+)\s(?<Node2>\d+)\s(?<Node3>\d+)\s(?<Node4>\d+)
");
                        Match match = regex.Match(line);
                        id = int.Parse(match.Groups["Id"].Value);
                        nodeIds = new int[]
                        {
                            int.Parse(match.Groups["Node1"].Value),
                            int.Parse(match.Groups["Node2"].Value),
                            int.Parse(match.Groups["Node3"].Value),
                            int.Parse(match.Groups["Node4"].Value)
                        };
                        return new LinearTetrahedron(id, nodeIds);
                    default:
                        throw new Exception("Нет такого типа конечного элемента");
                }
            }
        }
    }


    using FEM.DAL.Data;
    using System;
    using System.Collections.Generic;
    using System.IO;

    namespace FEM.DAL.Loaders
    {
        /// <summary>
        /// Загрузчик информации о нагрузках в узле из csv файла
        /// </summary>
        public class NodeLoadsCsvLoader : ILoader<IList<NodeLoad>>
        {
            readonly string _pathToCsv;
            readonly bool _hasHeader;
            readonly char _delimiter;
            /// <summary>
            /// Создание класса загрузки информации о нагрузках в узле
            /// </summary>
            /// <param name="pathToCsv">Путь к csv файлу</param>
            /// <param name="hasHeader">Есть ли у данных в csv файле шапка</param>
            /// <param name="delimiter">Разделитель ячеек csv файла</param>
            public NodeLoadsCsvLoader(string pathToCsv, bool hasHeader = true, char
delimiter = ';')
            {
                _pathToCsv = pathToCsv;
                _hasHeader = hasHeader;
                _delimiter = delimiter;
            }
```

```
        /// <summary>
        /// Возвращает список нагрузок узлов из csv файла
        /// </summary>
        /// <returns>Список нагрузок узлов</returns>
        public IList<NodeLoad> Load()
        {
            try
            {
                IList<NodeLoad> loads = new List<NodeLoad>();
                using (StreamReader sr = new StreamReader(_pathToCsv))
                {
                    if (_hasHeader)
                    {
                        sr.ReadLine();
                    }
                    string line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        string row = ProcessRow(line);
                        string[] values = row.Split(_delimiter);
                        loads.Add(new NodeLoad(int.Parse(values[0]),
                            (NodeLoadType)Enum.Parse(typeof(NodeLoadType),
values[1]),
                            double.Parse(values[2]),
                            double.Parse(values[3]),
                            double.Parse(values[4])
                            ));
                    }
                }
                return loads;
            }
            catch (Exception ex)
            {
                throw new Exception($"Не удалось загрузить узловые нагрузки.
{ex.Message}");
            }
        }

        /// <summary>
        /// Убирает лишние символы из строки csv файла
        /// </summary>
        /// <param name="line">Строка csv файла</param>
        /// <returns>Обработанная строка csv файла</returns>
        private string ProcessRow(string line)
        {
            return line.Trim(_delimiter).Replace('"', ' ');
        }
    }
}
```

ПРИЛОЖЕНИЕ Б
(Обязательное)
Руководство пользователя

Введение. Разработанное приложение предназначено для исследования напряжённо-деформированного состояния сплошного цилиндра.

Назначение и условия применения. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор *Intel Core i3* с тактовой частотой 2.30 МГц или любой эквивалент, равный по производительности или превосходящий;

– наличие клавиатуры, мыши и цветного монитора *SVGA*;

– операционная система *Windows* 7 и выше;

– более 1 Гб оперативной памяти.

Кроме аппаратного обеспечения необходим установленный стек технологий *.NET*.

Подготовка к работе. Запуск приложения осуществляется путём открытия файла *FEM.WPFGUI.exe*, который находится в директории *FEM\FEM.WPFGUI\bin\Debug\net5.0-windows*.

Подробное описание операций работы с приложением находится в подразделе 3.2.

ПРИЛОЖЕНИЕ В
(Обязательное)
Руководство программиста

Введение. Разработанное приложение предназначено для исследования напряжённо-деформированного состояния сплошного цилиндра.

Назначение и условия применения. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор *Intel Core i3* с тактовой частотой 2.30 МГц или любой эквивалент, равный по производительности или превосходящий;

– наличие клавиатуры, мыши и цветного монитора *SVGA*;

– операционная система *Windows* 7 и выше;

– среда разработки VisualStudio 2019;

– установленные плагины для разработки настольных приложений при помощи технологии *WPF*;

– более 1 Гб оперативной памяти.

Характеристики приложения. Разработанное приложение написано на языке программирования *C#.* Графический интерфейс написан при помощи *WPF*. При запуске открывается окно программы с кнопками для управления приложением.

Обращение к приложению. Запуск приложения осуществляется путём открытия файла *FEM.WPFGUI.exe*, который находится в директории *FEM\FEM.WPFGUI\bin\Debug\net5.0-windows*.

ПРИЛОЖЕНИЕ Г
(Обязательное)
Чертёж детали

ПРИЛОЖЕНИЕ Д

(Обязательное)

Блок-схемы алгоритмов основных методов решения задачи