Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждения высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**РАЗРАБОТКА ПЛАГИНА «ЛЮСТРА» ДЛЯ «КОМПАС-3D»**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

по дисциплине

«Основы разработки САПР» (ОРСАПР)

Выполнил:

студент гр. 580-3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горюнов А. С.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Руководитель:

к.т.н., доцент каф. КСУП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калентьев А.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Томск, 2023

**1 ВВЕДЕНИЕ**

Выполняется разработка библиотеки "Люстра" для САПР "Компас-3D" заказчиком, кандидатом технических наук, доцентом Калентьевым Алексеем Анатольевичем. Работа должна соответствовать нормативам, таким как ГОСТ 34.602-2020, ОС ТУСУР 01-2021, ОК 012-93, ГОСТ 19.103-77.

Цель проекта – автоматизация построения люстр. Назначение разрабатываемого плагина заключается в быстром моделировании люстр разных типов. Расширение позволяет мастерам по люстрам наглядно рассматривать и, при необходимости, перестраивать спроектированные модели.

**2 ПОСТАНОВКА И АНАЛИЗ ЗАДАЧИ**

В процессе анализа задач по разработке библиотеки "Люстра" для САПР "Компас-3D", первоначальной задачей было определение функциональных требований к плагину. Это включало в себя анализ потребностей пользователей, понимание особенностей работы с программой "Компас-3D" и определение ключевых параметров, подлежащих автоматизации.

Сроки для этапа анализа функциональных требований были установлены с 23 сентября по 10 октября 2023 года. Подводными камнями на этом этапе было недостаточное понимание как это все реализовать, что могло привести к дополнительным коррекциям в ходе разработки начальной версии приложения.

Для анализа использовались ресурсы в виде документации по API "Компас-3D" [1], стандартов ГОСТ, а также образцов проектирования люстр. Положительным результатом было четкое определение базовых параметров люстр для плагина.

Плановые сроки начала и завершения работ – с 23 сентября 2023 года по 29 декабря 2023 года.

**3 ОПИСАНИЕ ПРЕДМЕТА ПРОЕКТИРОВАНИЯ**

Люстра — подвесной потолочный светильник для создания бестеневого, в отличие от точечного источника, освещения помещений. Размеры люстры (расстояние между отдельными точечными источниками) и количество точечных источников света в ней определяют равномерность освещения. Изначально в качестве источника света использовались свечи, сегодня чаще всего лампы накаливания, люминесцентные лампы и светодиодные лампы [3].

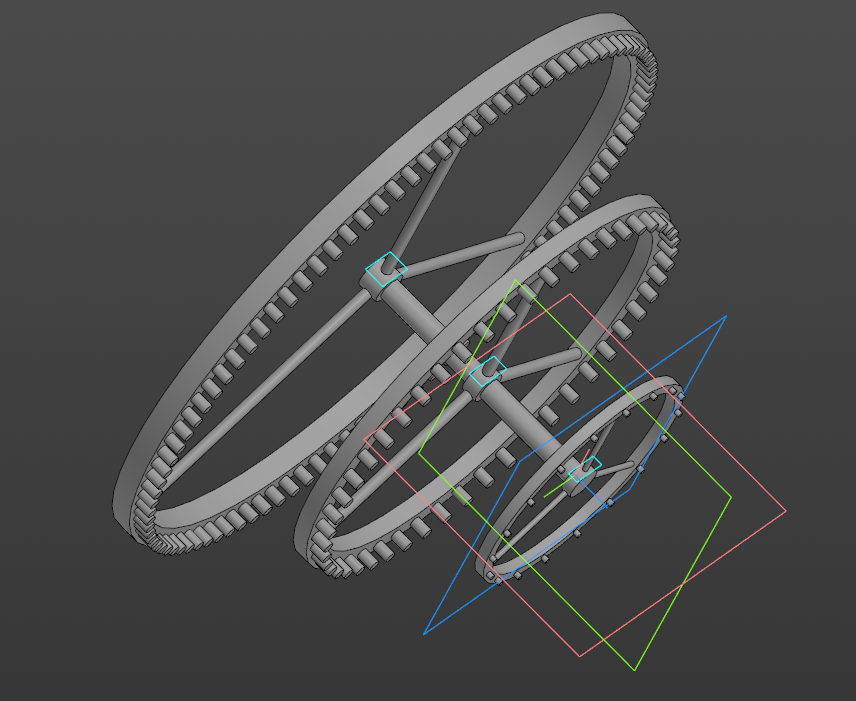


Рисунок 3.1 – Модель люстры.

**4 ВЫБОР ИНСТРУМЕНТОВ И СРЕДСТВ РЕАЛИЗАЦИИ**

При разработке плагина "Люстра" для САПР "Компас-3D" был выбран ряд инструментов и технологий, обеспечивающих эффективную и качественную разработку.

Технология: Плагин был реализован с использованием .NET и WinForms.

Библиотеки и Инструменты:

ReSharper: интегрированное средство для повышения качества кода. Автоматическое обнаружение ошибок и анализ кода.

Fine Code Coverage: инструмент для оценки покрытия кода тестами.

XAML Styler: предназначен для форматирования и стилизации кода XAML.

Editor Guidelines: инструмент, поддерживающий визуальное выделение и поддержание структуры кода, что содействует его пониманию и редактированию.

Spell Checker: обеспечивает автоматическую проверку орфографии в тексте документации.

NUnit и NUnit3TestAdapter: использовались для написания и выполнения модульных тестов.

Также, использовалась документация к API компаса [1].

**5 НАЗНАЧЕНИЕ ПЛАГИНА**

Назначение разрабатываемого плагина заключается в быстром моделировании люстр разных типов. Расширение позволяет мастерам по люстрам наглядно рассматривать и, при необходимости, перестраивать спроектированные модели.

**6 ОБЗОР АНАЛОГОВ**

Косвенным аналогом для плагина “Люстра” является плагин GearTeq для SolidWorks.

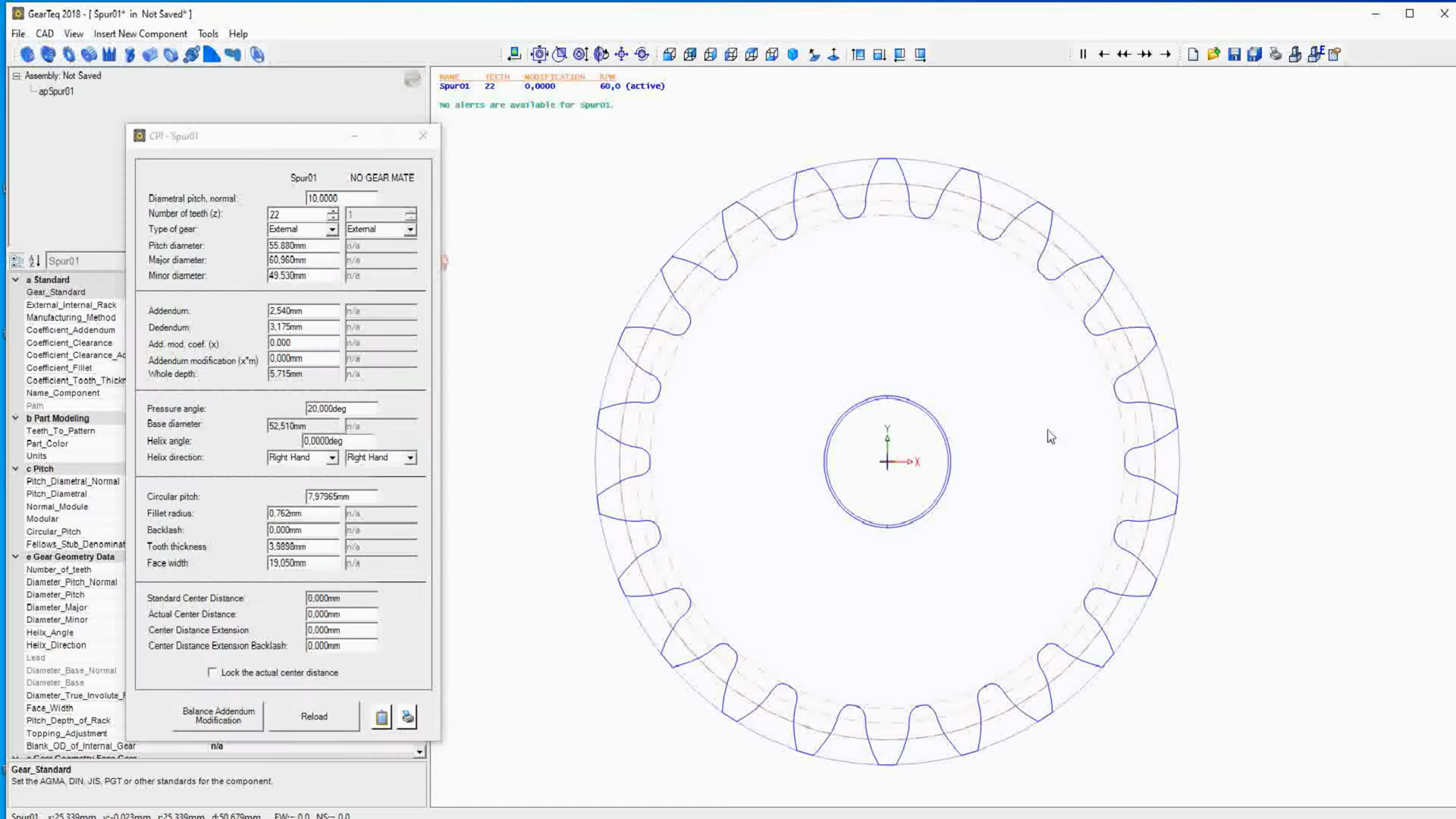


Рисунок 6.1 – Пользовательский интерфейс GearTeq.

Этот плагин можно считать аналогом так как в нем есть множество настраиваемых свойств, и можно добиться создания основы под люстры. Затем подредактировать ее, и получить что-то похожее на люстру.

Однако, этот плагин все-таки косвенный, так как он предназначен совершенно под другие задачи, а именно для проектирования различных шестеренок. [2]

**7 ОПИСАНИЕ РЕАЛИЗАЦИИ**

* 1. **Диаграмма классов после проектирования**

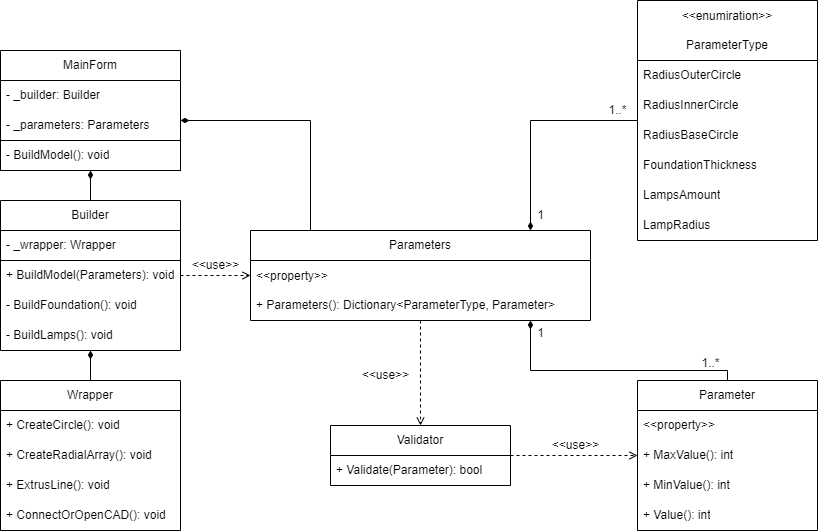


Рисунок 7.1 — Диаграмма классов после проектирования.

Разберем основные классы проекта:

− MainForm – является главным окном приложения. Хранит в себе параметры (Parameters) и объект класса строителя модели (Builder);

− Parameters – класс, хранящий в себе все параметры модели;

− Buider – класс строитель модели;

− Wrapper – класс обертка API САПР. В нем находятся все нужные методы создания примитивов и документов, которые пригодятся для построения модели;

− ParameterType – перечисление параметров детали;

− Parameter – Значения свойств параметра;

− Validator – Класс для проверки значений.

* 1. **Диаграмма классов после реализации**

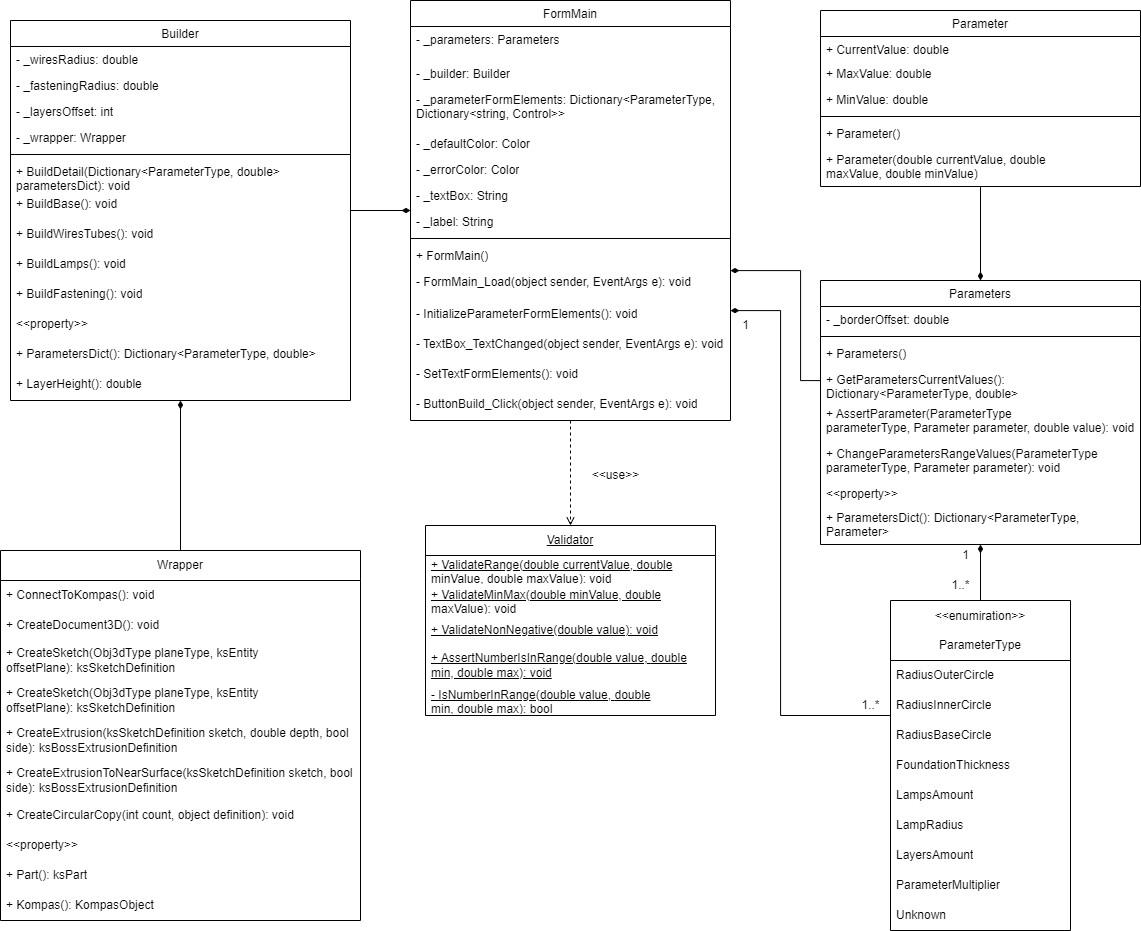
****

Рисунок 7.2 — Диаграмма классов после реализации.

Таблица 7.1 – Поля класса FormMain.

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| \_parameters | Parameters | Экземпляр класса параметров. |
| \_builder | Builder | Экземпляр класса строителя. |
| \_parameterFormElements | Dictionary<ParameterType, Dictionary<string, Control>> | Словарь, содержащий элементы управления формы для каждого типа параметра. |
| \_defaultColor | Color | Цвет по умолчанию для элементов формы. |
| \_errorColor | Color | Цвет для обозначения ошибок ввода. |
| \_textBox | string | Строка обозначающая textBox. |
| \_label | string | Строка обозначающая label. |

Таблица 7.2 – Методы класса FormMain.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Название | Аргументы | | | | | | | | | Тип данных | Описание |
| Название | Тип | | | | Описание | | | |
| FormMain() | - | | | | | | | | |  | Инициализирует новый экземпляр класса FormMain. |
| FormMain\_Load() | sender | | object | | | | Объект, который вызвал событие | | |  | Обработчик события загрузки формы. |
| e | | EventArgs | | | | Представляет информацию о событии | | |
| InitializeParameterFormElements() | - | | | | | | | | | void | Инициализирует элементы управления формы для каждого типа параметра. |
| TextBox\_TextChanged() | sender | | | object | | | | Объект, который вызвал событие | | void | Обработчик события изменения текста в текстовом поле. |
| e | | | EventArgs | | | | Представляет информацию о событии | |
| SetTextFormElements() | - | | | | | | | | | void | Устанавливает текст и значения элементов управления формы на основе текущих параметров. |
| ButtonBuild\_Click() | sender | | | | object | | | | Объект, который вызвал событие | void | Обработчик события нажатия кнопки "Построить". |
| e | | | | EventArgs | | | | Представляет информацию о событии |

Различия со старой версией:

1. Добавились дополнительные поля:

Поля с цветом нужны для удобной работы с элементами формы, которые реагируют на ошибки (текстбоксы);

Поля \_textBox и \_label необходимы для удобного взаимодействия элементов формы с словарями параметров;

1. Добавились дополнительные методы:

В целом, все эти методы необходимы для корректной работы View с Model.

Таблица 7.3 – Поля класса Builder.

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| \_wiresRadius | double | Радиус трубы. |
| \_fasteningRadius | double | Расстояние между слоями. |
| \_layersOffset | int | Расстояние между слоями. |
| \_wrapper | Wrapper | Экземпляр класса обертки. |

Таблица 7.4 – Методы класса Builder.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Аргументы | | | Тип данных | Описание |
| Название | Тип | Описание |
| ParametersDict() | - | | | Dictionary<ParameterType, double> | Словарь текущих значений всех параметров. |
| LayerHeight() | - | | | double | Расстояние между слоями. |
| BuildDetail() | parametersDict | Dictionary<ParameterType, double> | Словарь текущих значений всех параметров. | void | Строит деталь на основе заданных параметров. |
| BuildBase() | - | | | void | Строит основание детали. |
| BuildWiresTubes() | - | | | void | Строит трубы под провода. |
| BuildLamps() | - | | | void | Строит лампы детали. |
| BuildFastening() | - | | | void | Строит крепление между слоями. |

Различия со старой версией:

1. Добавились дополнительные поля:

Для более удобной работы внутри класса были добавлены поля значений параметров и нескольких свойств из класса обертки;

1. Добавились дополнительные методы:

Можно было оставить все в одном методе, но он бы получился огромным, поэтому этот метод был декомпозирован на несколько методов, отвечающие за построения отдельных частей детали.

Таблица 7.5 – Методы класса Wrapper.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Аргументы | | | Тип данных | Описание |
| Название | Тип | Описание |
| Part | - | | | ksPart | Компонент исполнения. |
| Kompas | - | | | KompasObject | Получает объект KOMPAS-3D. |
| ConnectToKompas() | - | | | bool | Подключается к активной сессии KOMPAS-3D. |
| CreateDocument3D() | - | | | ksDocument3D | Создает 3D-документ KOMPAS-3D. |
| CreateOffsetPlane(Obj3dType plane, double offset) | plane | Obj3dType | Плоскость построения | ksEntity | Создает смещенную плоскость относительно другой плоскости. |
| offset | double | Отступ плоскости |
| CreateSketch(Obj3dType planeType, ksEntity offsetPlane) | planeType | Obj3dType | Тип плоскости | ksSketchDefinition | Создает эскиз на заданной плоскости. |
| offsetPlane | ksEntity | Отступ-плоскость |
| CreateExtrusion(ksSketchDefinition sketch, double depth, bool side) | sketch | ksSketchDefinition | Эскиз | ksBossExtrusionDefinition | Создает выдавливание на основе эскиза. |
| depth | double | Глубина выдавливания |
| side | bool | Направление |
| CreateExtrusionToNearSurface( ksSketchDefinition sketch, bool side) | sketch | ksSketchDefinition | Эскиз | ksBossExtrusionDefinition | Создает выдавливание до ближайшей поверхности на основе эскиза. |
| side | bool | Направление |
| CreateCircularCopy(int count, object definition) | count | int | Количество копий | void | Создает кольцевую копию объекта. |
| definition | object | Объект копирования |

Различия со старой версией:

1. Добавлены необходимые поля для работы с Компасом;
2. Добавлены методы для создания минимальных операций в среде моделирования (создание эскиза, выдавливание и т.д.).

Таблица 7.6 – Поля класса Parameter.

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| CurrentValue | double | Получает или задает текущее значение параметра. |
| MaxValue | double | Получает или задает максимальное значение параметра. |
| MinValue | double | Получает или задает минимальное значение параметра. |

Таблица 7.7 – Методы класса Parameter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Аргументы | | | Тип данных | Описание |
| Название | Тип | Описание |
| Parameter() | - | | |  | Инициализирует новый экземпляр класса Parameter с дефолтными значениями. |
| Parameter(double currentValue, double maxValue, double minValue) | currentValue | double | Текущее значение |  | Инициализирует новый экземпляр класса Parameter с указанными значениями. |
| maxValue | double | Максимальное значение |
| minValue | double | Минимальное значение |

Различия со старой версией: добавлены конструкторы.

Таблица 7.8 – Поля класса Parameters.

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| \_borderOffset | double | Отступ от возможных граничных параметров детали. |

Таблица 7.9 – Методы класса Parameters.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Аргументы | | | Тип данных | Описание |
| Название | Тип | Описание |
| Parameters() | - | | |  | Инициализирует новый экземпляр класса Parameters со значениями параметров по умолчанию. |
| ParametersDict() | - | | | Dictionary<ParameterType, Parameter> | Словарь, содержащий параметры с их типами. |
| GetParametersCurrentValues() | - | | | Dictionary<ParameterType, double> | Получает словарь текущих значений параметров. |
| AssertParameter(  ParameterType parameterType, Parameter parameter, double value) | parameterType | ParameterType | Тип параметра | void | Проверяет и задает значение для указанного параметра, а затем обновляет диапазоны других параметров. |
| parameter | Parameter | Параметр |
| value | double | Значение параметра |
| ChangeParametersRangeValues(  ParameterType parameterType,  Parameter parameter) | parameterType | ParameterType | Тип параметра | void | Изменяет значения диапазона других параметров в зависимости от изменения указанного параметра. |
| parameter | Parameter | Параметр |

Различия со старой версией:

1. Добавлено поле для обозначения расстояние между граничными параметрами;
2. Добавлены методы для проверки и задания параметров.

Таблица 7.10 – Элементы перечисления ParametersType.

|  |  |
| --- | --- |
| Название | Описание |
| RadiusOuterCircle | Радиус внешнего круга. |
| RadiusInnerCircle | Радиус внутреннего круга. |
| RadiusBaseCircle | Радиус основания. |
| FoundationThickness | Толщина фундамента. |
| LampsAmount | Количество ламп. |
| LampRadius | Радиус лампы. |
| LayersAmount | Количество слоев. |
| ParameterMultiplier | Множитель параметров. |
| Unknown | Неизвестный тип параметра. |

Различия со старой версией: добавлено 2 параметра для дополнительной функциональности, для неизвестного типа параметров.

Таблица 7.11 – Методы класса Validator.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Аргументы | | | Тип данных | Описание |
| Название | Тип | Описание |
| ValidateRange(double currentValue, double minValue, double maxValue) | currentValue | double | Текущее значение | void | Проверяет, что текущее значение находится в пределах от минимального до максимального. |
| maxValue | double | Максимальное значение |
| minValue | double | Минимальное значение |
| ValidateMinMax(double minValue, double maxValue) | maxValue | double | Максимальное значение | void | Проверяет, что минимальное значение не больше максимального. |
| minValue | double | Минимальное значение |
| ValidateNonNegative(double value) | value | double | Текущее значение | void | Проверяет, что число не меньше нуля. |
| AssertNumberIsInRange(double value, double min, double max) | value | double | Текущее значение | void | Проверяет, что значение находится в заданном числовом диапазоне. |
| max | double | Максимальное значение |
| min | double | Минимальное значение |
| IsNumberInRange (double value, double min, double max) | value | double | Текущее значение | bool | Проверяет, что значение находится в заданном числовом диапазоне. |
| max | double | Максимальное значение |
| min | double | Минимальное значение |

Различия со старой версией: декомпозирован метод Validate().

В целом, все изменения были внесены из-за отсутствия опыта работы с подобного рода приложениями.

1. **ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПОЛЬЗОВАТЕЛЯ**

Разобраться в программе пользователю не составит труда. Интерфейс максимально прост. Нужно подставить необходимые значения параметров и нажать кнопку “Построить”;

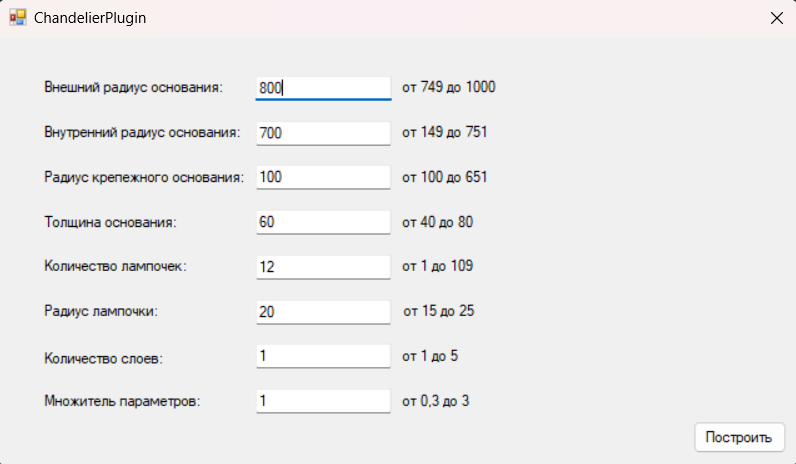


Рисунок 8.1 – Пользовательский интерфейс программы.

При возникновении каких-либо ошибок, кнопка “Построить” будет заблокирована.



Рисунок 8.2 – Ошибка: неверный формат строки.



Рисунок 8.3 – Ошибка: число не попадает в диапазон возможных значений.

**9 ТЕСТИРОВАНИЕ ПЛАГИНА**

**9.1 Функциональное тестирование**

Стандартные параметры, заданы по умолчанию.

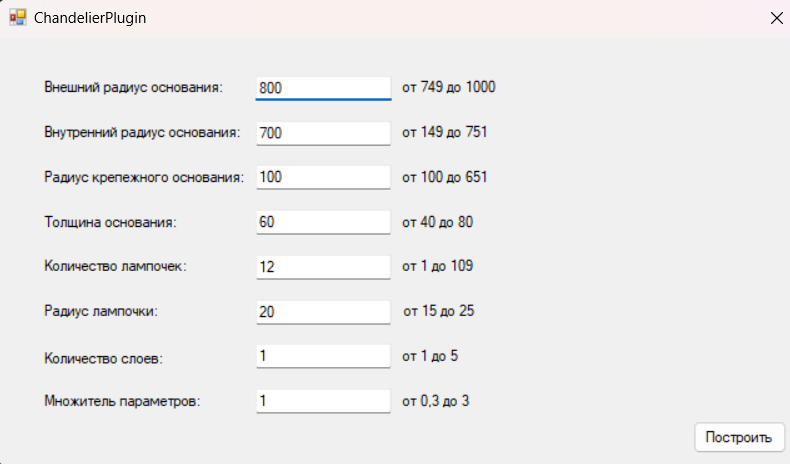


Рисунок 9.1 – Стандартные параметры.

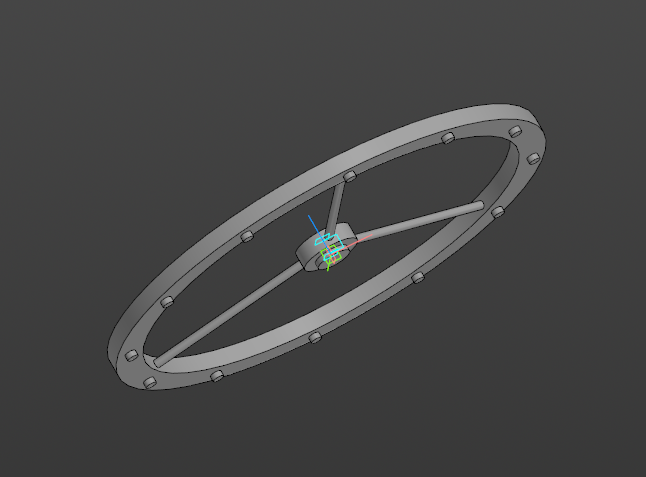


Рисунок 9.2 – Деталь с заданными стандартными параметрами.

Введем максимальные параметры.

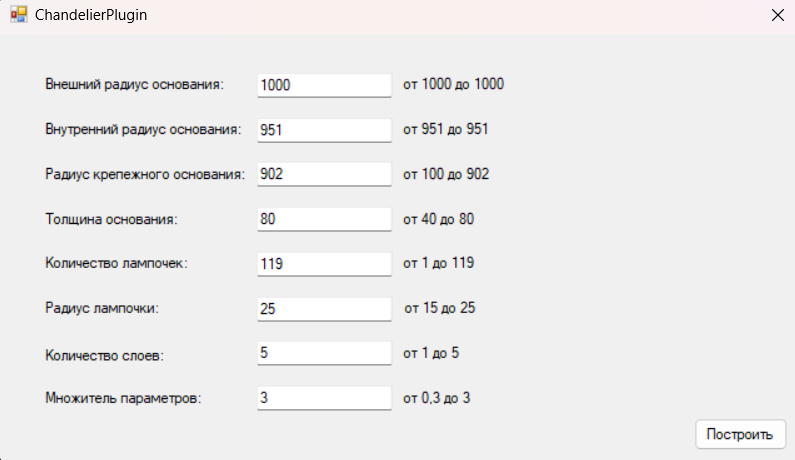


Рисунок 9.3 – Максимальные параметры.

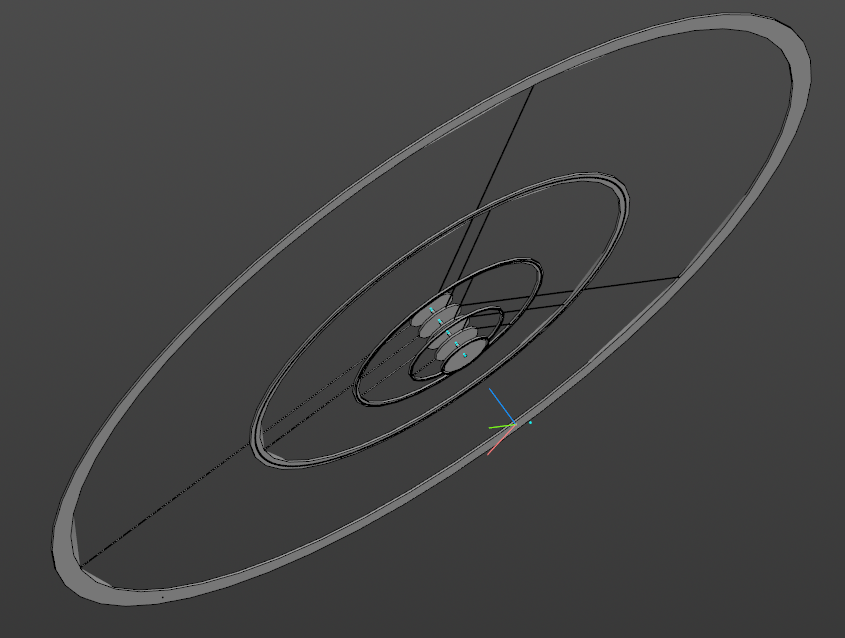


Рисунок 9.4 – Деталь с заданными максимальными параметрами.

Введем минимальные параметры.

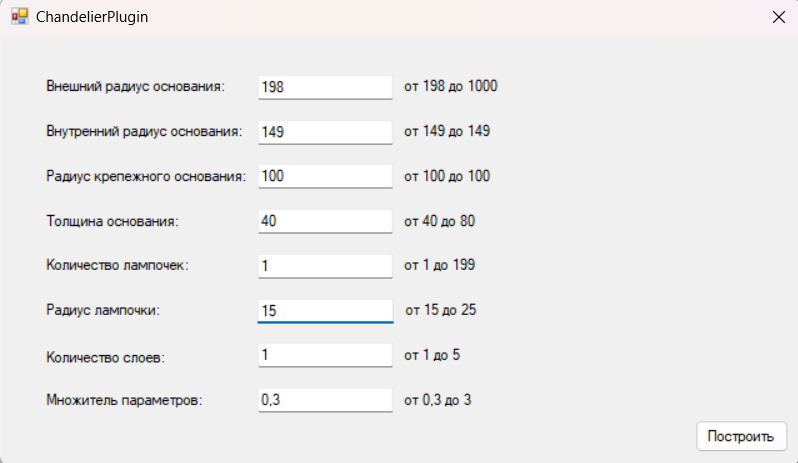


Рисунок 9.5 – Минимальные параметры.

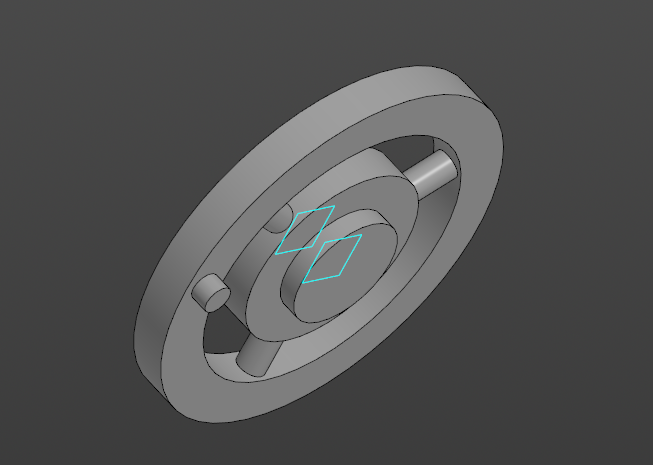


Рисунок 9.6 – Деталь с заданными минимальными параметрами.

**9.2 Модульное тестирование**

Всего написано 46 тестов.

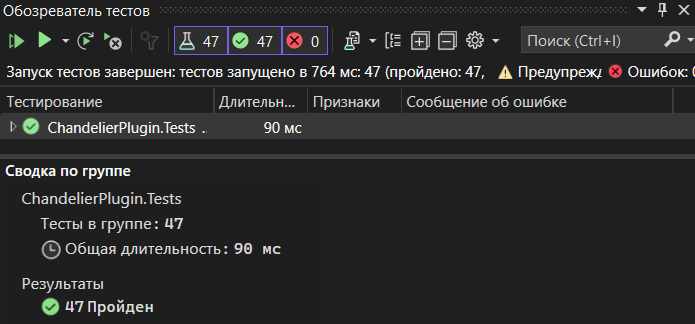


Рисунок 9.7 – Количество тестов.

Model покрыта на 100%.

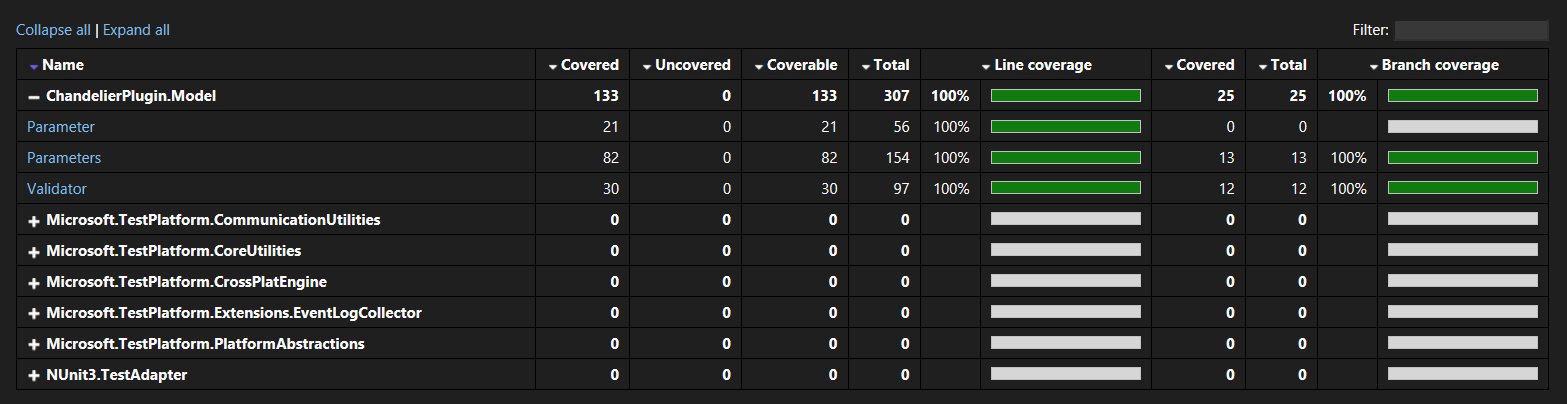


Рисунок 9.8 – Покрытие кода тестами.

Таблицы с описанием тестов.

Таблица 9.1 – Элементы класса ParametersTests.

|  |  |
| --- | --- |
| Название | Описание |
| AssertParameter\_Parameter\_UpdateParameter( ParameterType parameterType, double currentValue, double maxValue, double minValue, double value) | Этот метод предназначен для проверки и обновления параметра в заданных границах. |
| AssertParameter\_WrongParameterValue\_ThrowException( ParameterType parameterType, double currentValue, double maxValue, double minValue, double value) | Этот метод предназначен для проверки параметра и генерации исключения, если предоставленное значение неверно. |

Продолжение таблицы 9.1 – Элементы класса ParametersTests.

|  |  |
| --- | --- |
| ChangeParametersRangeValues\_Parameter\_UpdateRangeValues( ParameterType parameterType, double currentValue, double maxValue, double minValue, ParameterType[] parametersTypes, double[] expectedParameters) | Этот метод отвечает за обновление значений диапазона для конкретного типа параметра и проверку обновления по ожидаемым параметрам. |
| public void GetParametersCurrentValues\_CurrentValuesAreEqual( ParameterType[] parametersTypes,double[] currentValues) | Этот метод предназначен для проверки текущих значений параметров. |
| ChangeParametersRangeValues\_Parameter\_NothingHappens( ParameterType parameterType, double currentValue, double maxValue, double minValue) | Этот метод предназначен для отсутствия действий при изменении значений диапазона параметра. |
| ChangeParametersRangeValues\_Parameter\_ThrowException( ParameterType parameterType, double currentValue, double maxValue, double minValue) | Этот метод предназначен для генерации исключения при попытке изменения значений диапазона параметра. |

Таблица 9.2 – Элементы класса ParameterTests.

|  |  |
| --- | --- |
| Название | Описание |
| Parameter\_Initialization\_  SetPropertiesCorrectly(double currentValue, double maxValue, double minValue) | Метод предназначен для инициализации параметра с корректными значениями свойств. |
| Parameter\_SetProperties\_UpdateValues(double currentValue, double maxValue, double minValue) | Метод выполняет установку свойств параметра с обновлением значений. |
| Parameter\_Initialization\_  CurrentValueOutOfRange\_ThrowArgumentException( double currentValue, double maxValue, double minValue) | Метод инициализирует параметр, но выбрасывает исключение, если текущее значение находится вне заданных границ. |
| Parameter\_Initialization\_  MinValueGreaterThanMaxValue\_ThrowArgumentException( double currentValue, double maxValue, double minValue) | Метод выполняет инициализацию параметра, однако генерирует исключение, если минимальное значение больше максимального. |
| Parameter\_Initialization\_  WithNegativeValues\_ShouldThrowArgumentException(  double currentValue, double maxValue, double minValue) | Метод предназначен для инициализации параметра, при этом выбрасывается исключение, если указаны отрицательные значения параметра. |

Таблица 9.3 – Элементы класса ValidatorTests.

|  |  |
| --- | --- |
| Название | Описание |
| ValidateRange\_ShouldNotThrowException(double value, double minValue, double maxValue) | Метод предназначен для валидации значения в заданном диапазоне, не генерируя исключений при успешной проверке. |
| ValidateRange\_ShouldThrowArgumentException(double value, double minValue, double maxValue) | Метод выполняет валидацию значения в заданном диапазоне и генерирует исключение типа ArgumentException в случае неудачной проверки. |
| ValidateMinMax\_ShouldNotThrowException(double minValue, double maxValue) | Метод предназначен для проверки корректности заданных минимального и максимального значений, не вызывая исключений при успешной проверке. |
| ValidateMinMax\_ShouldThrowArgumentException(double minValue, double maxValue) | Метод выполняет проверку корректности заданных минимального и максимального значений и генерирует исключение типа ArgumentException в случае неудачной проверки. |
| ValidateNonNegative\_ShouldNotThrowException(double value) | Метод предназначен для валидации, что значение является неотрицательным числом, не генерируя исключений при успешной проверке. |
| ValidateNonNegative\_ShouldThrowArgumentException(double value) | Метод выполняет валидацию, что значение является неотрицательным числом, и генерирует исключение типа ArgumentException в случае неудачной проверки. |
| AssertNumberIsInRange\_ShouldNotThrowException(double value, double minValue, double maxValue) | Метод предназначен для утверждения, что число находится в заданном диапазоне, не генерируя исключений при успешной проверке. |
| AssertNumberIsInRange\_ShouldThrowException(double value, double minValue, double maxValue) | Метод выполняет утверждение, что число находится в заданном диапазоне, и генерирует исключение типа Exception в случае неудачной проверки. |

**9.3 Нагрузочное тестирование**

Тестирование проводилось на системе со следующими характеристиками:

* Оперативная память – 8,00 ГБ (доступно: 7,71 ГБ);
* Процессор – 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz;
* Тип системы – 64-разрядная операционная система, процессор x64;
* Видеокарта – NVIDEA GeForce GTX 1650.

Проведем нагрузочное тестирование приложения с обычными параметрами (100 итераций в цикле). На 78 итерации программа сломалась, закончилась память.

Рисунок 9.9 – График зависимости памяти ОЗУ (ГБ) от построения модели.

Из графика 1 видно, что память изменяется линейно, но с некоторыми скачками вниз. Это связано с тем, что для регенерации памяти происходит остановка обмена данными, что снижает нагрузку на оперативную память. В конце график не сильно идет вверх, это связано с тем, что, когда на оперативную память идет сильная нагрузка, ОС начинает использовать файл подкачки для разгрузки оперативной памяти.

Рисунок 9.10 – График зависимость времени от построения модели.

Из графика 2 видно, что время построения моделей изменяется линейно, но ближе к концу графика идет как экспонента. Это связано с увеличением нагрузки на оперативную память и центральный процессор с каждым построением модели. За счет этого время построения модели увеличивается.

**10 Заключение**

Был реализован плагин по построению люстр для САПР “КОМПАС-3Д”. Плагин позволяет быстро создавать модели различных типов люстр, что облегчит работу мастеров по люстрам. Он также обеспечивает возможность наглядного рассмотрения спроектированных моделей и их перестройки в случае необходимости.

Программа реализована на C#. Также, были использованы сторооние библиотеки: ReSharper, Fine Code Coverage, XAML Styler, Editor Guidelines, Spell Checker, NUnit и NUnit3TestAdapter.

Были рассмотрены прямые и косвенные аналоги приложения.

Model программы покрыта тестами на 100%. Всего 47 тестов.

Также, проведено нагрузочное тестирование. После 78 попытки построения закончилась память.

Получен навык анализа документации и работы с внешними библиотеками для языка программирования C#.

**11 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

* + - 1. Документация к API KOMPAS-3D [Электронный ресурс] // ASCON [Официальный сайт] URL: <https://help.ascon.ru/KOMPAS_SDK/22/ru-RU/index.html> (дата обращения: 25.10.2023);
      2. Gearteq для SolidWorks [Электронный ресурс] // Страница партнера SolidWorks [Официальный сайт] URL: <https://www.solidworks.com/ru/partner-product/gearteq> (дата обращения: 17.10.2023);
      3. Люстра [Электронный ресурс] // Страница на Википедии URL: <https://ru.wikipedia.org/wiki/Люстра> (дата обращения: 17.10.2023).