БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

КУРСОВОЙ ПРОЕКТ

по дисциплине «Методы и алгоритмы компьютерной графики»

Тема: «Система автоматизации конструирования с неравномерным масштабированием трехгранных полостей, вписанных в область детали»

|  |  |
| --- | --- |
| Исполнитель: | студент группы 10702119 Микитич Б.В. |
| Руководитель проекта: | доц. Полозков Ю.В. |

Минск 2022

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Методы и алгоритмы компьютерной графики»

Тема: «Система автоматизации конструирования с неравномерным масштабированием трехгранных полостей, вписанных в область детали»

|  |  |
| --- | --- |
| Исполнитель: | студент группы 10702119 Микитич Б.В. |
| Руководитель проекта: | доц. Полозков Ю.В. |

Минск 2022

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc121946817)

[1. Проектирование приложения для автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы 6](#_Toc121946818)

[1.1. Обзор используемого программного обеспечения 6](#_Toc121946819)

[1.2. Описание геометрической конфигурации проектируемой модели призмы 9](#_Toc121946820)

[2. Разработка приложения для автоматизации конструирования с неравномерным масштабированием трехгранных полостей, вписанных в область детали. 11](#_Toc121946821)

[2.1. Описание процесса интерактивного конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы 11](#_Toc121946822)

[2.2. Программная реализация автоматизированного конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы 14](#_Toc121946823)

[2.3. Руководство пользователю 24](#_Toc121946824)

[ЗАКЛЮЧЕНИЕ 27](#_Toc121946825)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc121946826)

[ПРИЛОЖЕНИЕ А 29](#_Toc121946827)

# ВВЕДЕНИЕ

Трехмерное моделирование - это целый раздел компьютерной графики, а также совокупность инструментов (аппаратных и ПО) и приемов, которые используются для создания объемных объектов.

Основная цель 3D моделирования состоит в создании модели какого-то объекта для дальнейшего использования, например, для изготовления прототипа, мастер-модели, восковки и прочего. Готовая трехмерная модель наглядно изображает, как в перспективе будет выглядеть готовый объект со всеми деталями и особенностями. Однако для построения одной такой модели необходимо приложить большое количество усилий и времени. В связи с этим возник вопрос, как можно автоматизировать данный процесс. Ведь основная цель создания автоматизированных производственных процессов — повышение эффективности труда инженеров, которая включает:

* сокращения трудоёмкости проектирования и планирования;
* сокращения сроков проектирования;
* сокращения себестоимости проектирования и изготовления, уменьшение затрат на эксплуатацию;
* повышения качества и технико-экономического уровня результатов проектирования;
* сокращения затрат на натурное моделирование и испытания.

Таким образом, создание приложения для автоматизации конструирования для автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы обеспечит достижение применения методов вариантного проектирования и оптимизации, повторного использования проектных решений, данных и наработок и посодействует в создании натурных испытаний и макетирования математическим моделированием.

Результатами автоматизированных производственных процессов служат законченные проекты или части его. Они могут быть использованы на различных производствах.

# Проектирование приложения для автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы

# Обзор используемого программного обеспечения

Для проектирования приложения по автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы будет целесообразно выбрать простые, удобные, но в то же время профессиональные, обладающие достаточными средствами для разработки сложных трехгранных проектов средства разработки. Язык программирования C# и интегрированную среду разработки Microsoft Visual Studio 2022 как раз являются таковыми. В связке с программным комплексом САПР SolidWorks 2018, который обеспечивает разработку изделий любой степени сложности и назначения, а также легко интегрируется с языком программирования C#, мы сможем быстро, что важно для разработчика, и квалифицированно, что важно для заказчика, создать приложение для автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы.

**Язык программирования C#.** На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. На нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. C# уже не молодой язык и, как и вся платформа .NET уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 11, которая вышла 8 ноября 2022 года вместе с релизом .NET 7. C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче. C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей.

**Microsoft Visual Studio 2022**. Microsoft Visual Studio – это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию. Эти инструменты предназначены для максимально эффективной совместной работы; все они доступны в интегрированной среде разработки (IDE) Visual Studio.

Visual Studio можно использовать для создания различных типов приложений, от простых консольных приложений и игр для мобильных клиентов до больших и сложных систем, обслуживающих предприятия и центры обработки данных.

По умолчанию Visual Studio обеспечивает поддержку C#, C и C++, JavaScript, F# и Visual Basic. Также можно самостоятельно расширить Visual Studio, создав собственные инструменты для выполнения специализированных задач.

**SolidWorks 2018.** Анализируя мировые тенденции развития САПР и проводя аналогии с серединой 90-х годов, когда CAD-системы среднего уровня одержали уверенную победу над 2D-системами, можно сделать вывод, что сейчас наблюдается очередной качественный переход в развитии 3D-моделирования — в сторону единых интегрированных решений. Появилась новая градация в позиционировании САПР на рынке — комплексные системы. Безусловно, ярчайшим представителем этого направления является SolidWorks 2003, степень интеграции и функциональные возможности которой уже давно ни у кого не вызывают сомнений. В настоящий момент структуру пакета SolidWorks можно представить следующим образом:

* базовое решение, куда входят возможности 3D-моделирования деталей и сборок, экспресс-анализ прочности и кинематики, оформление чертежей, импорт/экспорт геометрии из других систем, API-интерфейс;
* помимо базового решения разработано более 300 специализированных модулей, решающих различные прикладные задачи, такие как управление данными, технологическая подготовка производства и т.д.

В SolidWorks можно одинаково удачно работать как с твердыми телами, так и с поверхностями. Как правило, деталь представляет собой твердое тело, поверхность либо сочетание твердого тела и набора поверхностей. Процесс построения 3D-модели основывается на создании элементарных геометрических примитивов и выполнении различных операций между ними. Подобно конструктору LEGO модель набирается из стандартных элементов (блоков) и может быть отредактирована путем либо добавления (удаления) этих элементов, либо изменения характерных параметров блоков.

3D-модель содержит наиболее полное описание физических свойств объекта (объем, масса, моменты инерции) и дает проектанту возможность работы в виртуальном 3D-пространстве, что позволяет на самом высоком уровне приблизить компьютерную модель к облику будущего изделия, исключая этап макетирования.

Таким образом приложение для автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы получится кросс-платформенным, адаптивным и переиспользуемым, что важно в современных реалиях.

# Описание геометрической конфигурации проектируемой модели призмы

Треугольная призма — это тело, представляющее собой геометрическую фигуру призмы с двумя треугольными параллельными основаниями и тремя боковыми гранями в форме прямоугольников, которые соединяют соответствующие стороны оснований и являются перпендикулярными к ним.

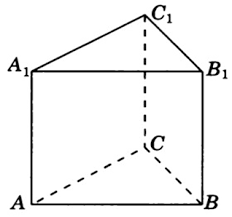


Рисунок 1.2.1 – Пример треугольной призмы

Основанием треугольной призмы является треугольник. Треугольник — геометрическая фигура, образованная тремя отрезками, которые соединяют три точки, не лежащие на одной прямой.

Для построения треугольника любой формы и размеров не обязательно знать все его три стороны и три угла между ними, достаточно лишь будет информации о двух смежных сторонах и угла между ними.

Пусть треугольник будет обозначен как ABC, известные стороны будут обозначены как AB и AC соответственно, а угол между ними – α. Пусть высота треугольника будет обозначена как h, а точка, в которую она падает – точка О.

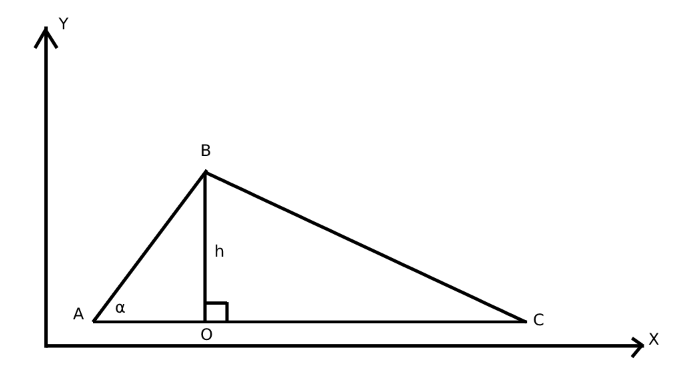


Рисунок 1.2.2 – Пример треугольника (основания треугольной призмы)

Таким образом высота треугольника h высчитывается по формуле h = sin(α) \* AB, сторона AO будет равна AO = cos(α) \* AB. С помощью найденной высоты h и стороны AB теперь легко переместиться из первоначальной точки A к точке B в координатной плоскости ХУ. Пусть точка A находится в координатах (0;0), тогда координаты точки В = (AO; h). Из точки В переместиться в точку C будет так же несложно, координаты точки C в координатной плоскости ХУ: A = (AC; 0).

Если же угол будет отрицательным, то треугольник получится перевернутым, однако все расчеты все равно будут произведены корректно с учетом знаков.

Длина треугольной призмы – это расстояние от одного основания до другого. Длина треугольной призмы определяет объемность 3D тела и его объем.

Для автоматизации конструирования нескольких выпуклых трехгранных полостей в виде треугольной призмы таким образом, чтобы они не пересекались задается шаг между ними, при построении.

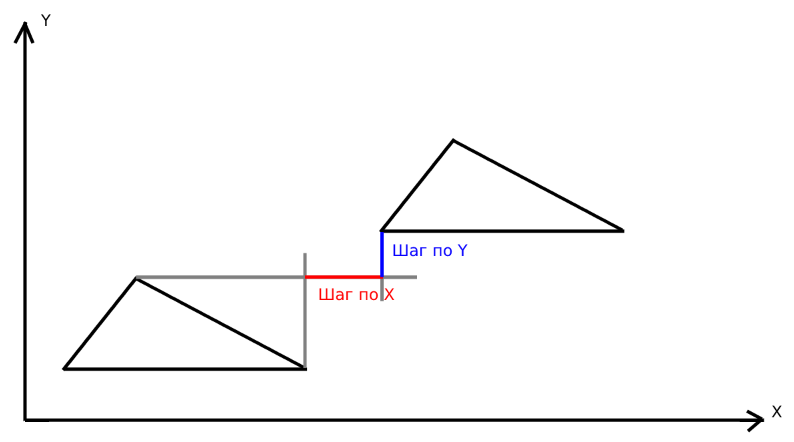


Рисунок 1.2.5 – Шаг при построении нескольких трехгранных полостей в виде треугольной призмы

# Разработка приложения для автоматизации конструирования с неравномерным масштабированием трехгранных полостей, вписанных в область детали.

# Описание процесса интерактивного конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы

Сперва необходимо запустить программу SOLIDWORKS.



Рисунок 2.1.1 – Запуск приложения SOLIDWORKS

Так как суть проекта заключается в том, чтобы он был адаптивным и мог выполнять автоматизацию конструирования выпуклых трехгранных полостей в виде треугольной призмы в любой детали, открывается заранее подготовленная 3D модель произвольной формы.

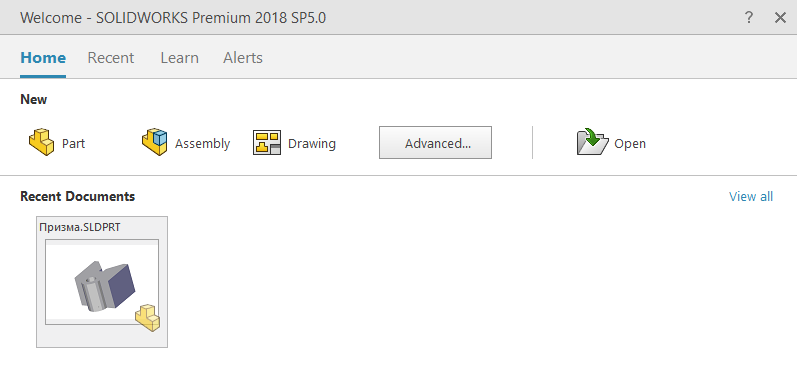


Рисунок 2.1.2 – Выбор файла «Призма.SLDPRT»

Теперь в выбранной части детали необходимо создать трехгранную полость в виде треугольной призмы. Для этого создается «Трехмерный эскиз».

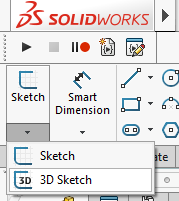


Рисунок 2.1.3 – Создание «Трехмерного эскиза»

Для описания треугольной призмы нужно создать ее основание в виде произвольного треугольника, что осуществляется путем последовательного создания трех замкнутых прямых линий инструментом «Линия».

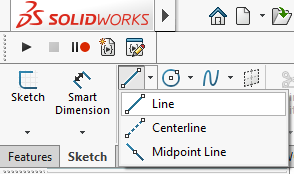


Рисунок 2.1.4 ­– Инструмент «Линия».

Производится построение основания треугольной призмы в виде «Трехмерного эскиза».

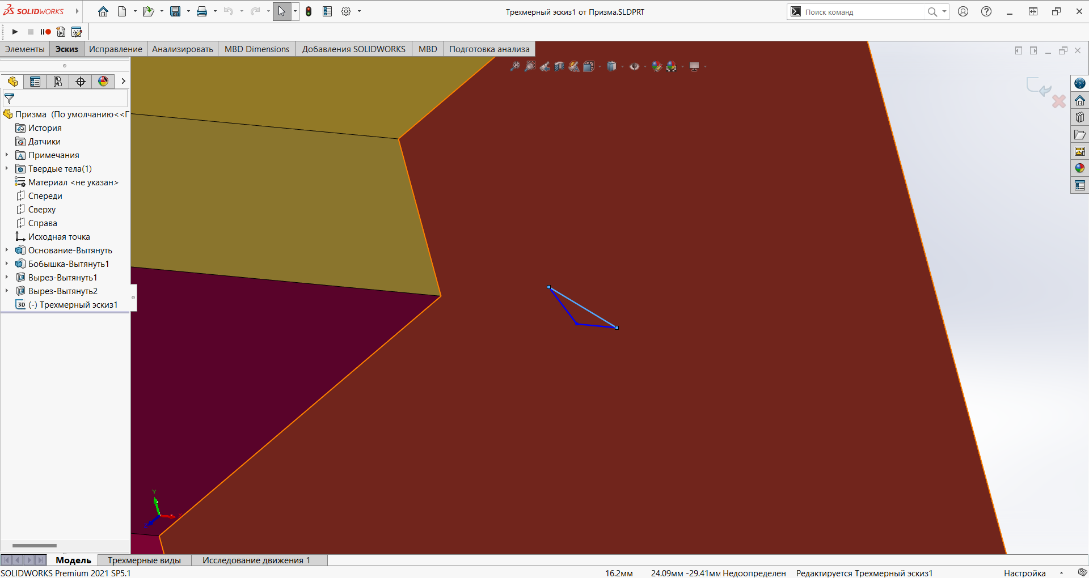


Рисунок 2.1.5 ­– Основание треугольной призмы

Далее выбирается полученный «Трехмерный эскиз» и с помощью инструмента «Вытянутый вырез» создается полость заданной длины.

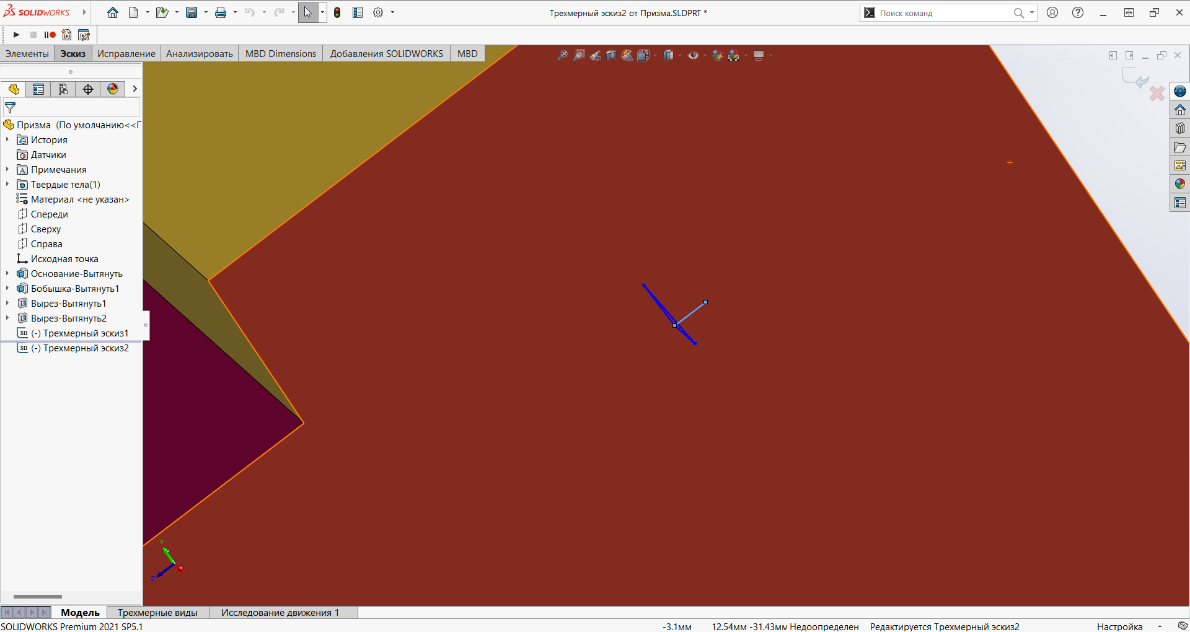


Рисунок 2.1.7 – Применение инструмента «Вытянутый вырез»

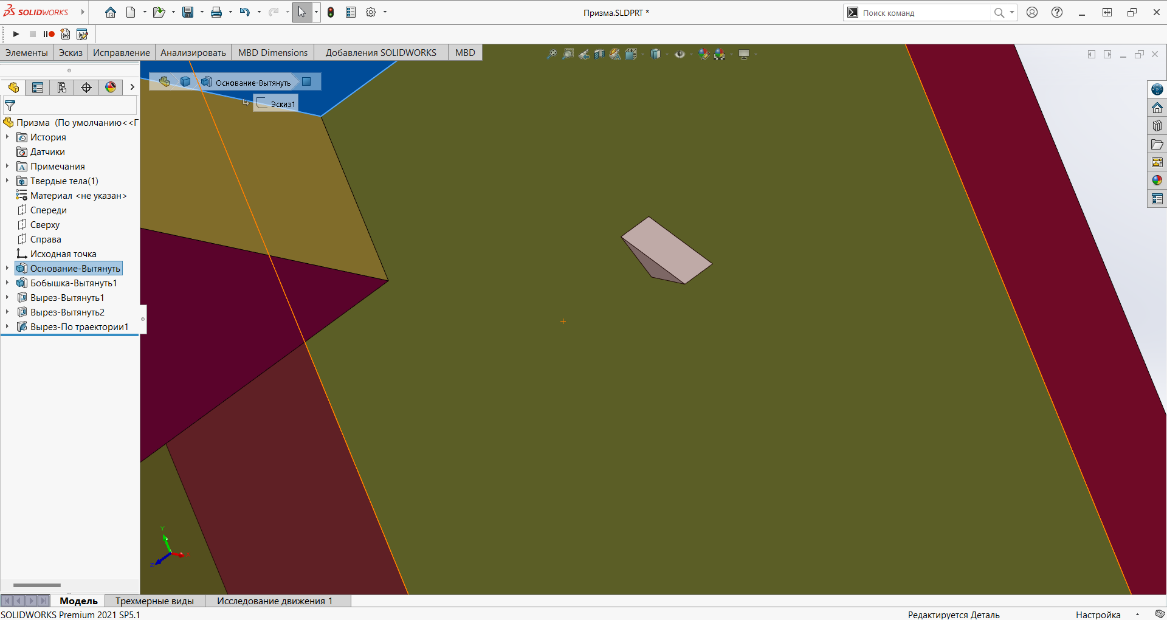


Рисунок 2.1.8 – Результат применение инструмента «Вытянутый вырез»

Таким образом с помощью программного средства SOLDIWORKS в интерактивном режиме конструируется 3D модель с выпуклой трехгранной полостью в виде треугольной призмы. Однако для создания множества подобных полостей, экономии времени и средств разработчика реализуется система автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы.

## Программная реализация автоматизированного конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы

Для взаимодействия с приложением SOLIDOWRKS из системы, использовался SOLDIWORKS API. SOLIDWORKS API – это интерфейс программирования COM для программного обеспечения SOLIDWORKS. API содержит сотни функций, которые можно вызывать из Visual Basic (VB), Visual Basic for Applications (VBA), VB.NET, C++, C# или файлов макросов SOLIDWORKS. Эти функции предоставляют программисту прямой доступ к функциональности SOLIDWORKS.

Использовались следующие функции SOLIDWORKS API:

* Insert3DSketch ­– вставляет новый 3D-эскиз в модель или закрывает активный эскиз;
* CreateLine ­– cоздает линию эскиза в активном 2D или 3D эскизе;
* SelectById2 ­– выбирает указанную сущность;
* FeatureCut4 – вырезает из тела вытянутое отверстие;
* ClearSelection - Отменяет выбор всех выделенных в данный момент ячеек.

Для автоматизации конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы необходимо создать новый проект на языке С# в Visual Studio.

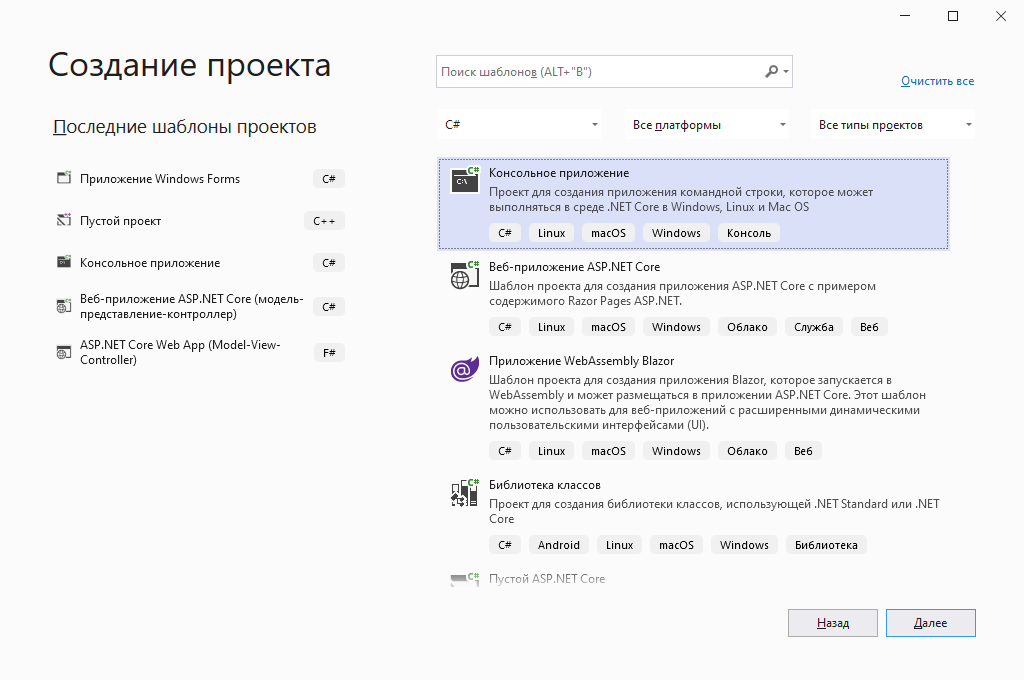


Рисунок 2.2.1 – Создание нового проекта на языке С# в Visual Studio

Теперь нужно прописать основную логику работы программы и создать пользовательский интерфейс, для удобного взаимодействия с программой.

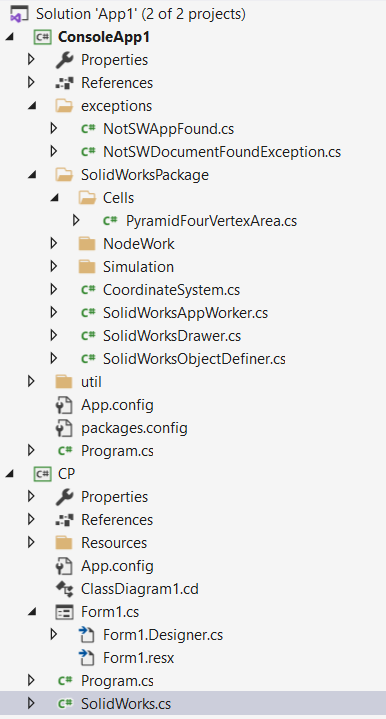


Рисунок 2.2.2 – Структура проекта

Основные компоненты, реализующие главную логику автоматизации конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы:

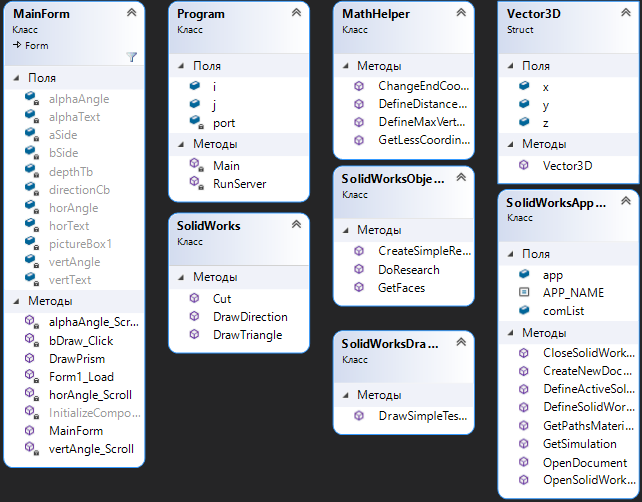


Рисунок 2.2.3 – Диаграмма классов приложения

1. Класс Form1

Класс Form1 служит для отображения, обработки, передачи и хранения элементов пользовательского интерфейса, таких как ComboBox, TrackBar, TextBox, Label, Button, PictureBox.

Свойства класса:

* public static float a1 – значение стороны а основания треугольной призмы;
* public static float b1 – значение стороны b основания треугольной призмы;
* public static int alpha1 – значение угла α основания треугольной призмы;

– public static float depth1 – значение длины полости в виде треугольной призмы;

– public static float choice – значение, отвечающее за выбор паттерна рисовки призм;

– public static float scale – значение, отвечающее за коэффициент увеличивающий\уменьшающий размера призм.

Методы класса:

* public static void InitializeComponent – служит для инициализации всех компонентов формы;
* private void bDraw\_Click – предназначен для обработки события при нажатии на кнопку «bDraw», преобразования введённых пользователем данных в параметры, обработки неправильно введенных значений и передачи данных в другие классы программы;

private void bDraw\_Click(object sender, EventArgs e)

{

// Prism parameters

double width = 0;

double height = 0;

double thickness = 0;

// Triangle parameters

double a = 0;

double b = 0;

double alphaSign = 1;

// Cut parameters

double depth = 0;

double betaSign = 1;

SolidWorks sw;

try

{

a1 = float.Parse(aSide.Text);

b1 = float.Parse(bSide.Text);

alpha1 = alphaAngle.Value;

depth1 = float.Parse(tbDepth.Text);

scale = float.Parse(textBox1.Text);

}

catch (Exception)

{

MessageBox.Show("Введены неверные значения!", "Ошибка!");

return;

}

try

{

sw = new SolidWorks();

}

catch (Exception)

{

MessageBox.Show("Solid Works не открыт!", "Ошибка!");

return;

}

if (signAlpha.SelectedIndex == 1)

{

alphaSign = -1;

}

if (radioButton1.Checked)

{

choice = 0;

}

else if (radioButton2.Checked)

{

choice = 1;

}

else if (radioButton3.Checked)

{

choice = 2;

}

else

{

choice = 0;

}

}

* private void alphaAngle\_Scroll – служит для конвертации полученного значения угла в Text

1. Класс SolidWorks

Данный класс предназначен для конструирования полостей в виде треугольной призмы.

Методы класса:

* public static void DrawTriangle – служит для создания основания треугольной призмы;
* public void Cut – служит для осуществления выреза полости треугольной призмы;
* private void CreatePart – предназначен для получения детали в SolidWorks

1. Класс Program

Данный класс служит для запуска консоли и обработки запросов.

Методы класса:

* static void Main(string[] args) – служит для запуска консоли и всех компонентов программы;

1. Класс SolidWorksObjectDefiner

Данный класс служит для определения объектов SolidWorks.

Методы класса:

* public static HashSet<Face> GetFaces – служит для получения граней фигуры;
* public static void DoResearch – служит для запуска исследования фигуры и вывод значений на консоль;

public static void DoResearch()

{

try

{

SolidWorksAppWorker.DefineSolidWorksApp();

var doc = SolidWorksAppWorker.DefineActiveSolidWorksDocument();

Console.WriteLine("Приложение SW и документ определены!\n");

var studyManager = new StudyManager();

var study = studyManager.GetExistingCompletedStudy();

if (study == null)

{

Console.WriteLine("Создание исследования запущено ...");

study = studyManager.CreateStudy(CreateSimpleRecord());

Console.WriteLine("Создание исследования завершено. Проведение исследования начато ...");

study.RunStudy();

Console.WriteLine("Проведение исследования завершено успешно!");

}

else

{

Console.WriteLine("Загружено активное исследование!");

}

var studyResults = study.GetResult();

Console.WriteLine($" Результаты исследования: кол-во элементов: {studyResults.meshElements.Count()}, узлов: {studyResults.nodes.Count()}");

string param = "VON";

var stressValues = studyResults.DefineMinMaxStressValues(param);

double minvalue = stressValues["min"],

maxvalue= 2 \* MaterialManager.GetMaterials()[study.MaterialName].physicalProperties["SIGYLD"];

Console.WriteLine($" минимальное напряжение " +

$"VON = {minvalue} // " +

$"максимальное напряжение по VON {stressValues["max"]}" +

$" // предел текущести = {maxvalue}");

var cutElements = studyResults.DetermineCutElements(param, minvalue, maxvalue) as HashSet<Element>;

Console.WriteLine("Начало поиска областей");

var areas = ElementAreaWorker.DefineElementAreas(cutElements);

Console.WriteLine($"Окончание поиска областей. Их общее количество - {areas.Count}");

Console.Write($"Введите номер области для выреза от 0 до {areas.Count-1}: ");

int number = Convert.ToInt32(Console.ReadLine());

Form1 form = new Form1();

Application.Run(form);

Program.a = CW.Form1.a1 / 1000f;

Program.b = CW.Form1.b1 / 1000f;

if (number >= 0 && number < areas.Count)

{

foreach (var element in areas.ElementAt(number).elements)

{

var elementPyramid = new PyramidFourVertexArea(element.GetDrawingVertexes(1));

SolidWorksDrawer.DrawPyramid(doc, elementPyramid);

}

}

}

catch (Exception ex)

{

Console.WriteLine("" + ex);

}

}

* public static StaticStudyRecord CreateSimpleRecord – задает сетку и материал для исследования фигуры;

1. Класс NotSWAppFound

Данный класс предназначен для вывода сообщения о ошибки

Методы класса:

* public override string ToString – конвертирует ошибку в сообщения "SolidWorks instanse was not found!";

1. Класс NotSWDocumentFoundException

Данный класс предназначен для вывода сообщения о ошибки

Методы класса:

* public override string ToString – конвертирует ошибку в сообщения " "SolidWorks document was not found!";

1. Класс PrismFourVertexArea

Данный класс служит для генерации точек, по которым формируются отверстия в виде треугольной призмы.

Свойства класса:

* public readonly Point3D vertex1– хранит в себе значения координат 1 вектора;
* public readonly Point3D vertex2– хранит в себе значения координат 2 вектора;
* public readonly Point3D vertex3– хранит в себе значения координат 3 вектора;
* public readonly Point3D vertex4– хранит в себе значения координат 4 вектора;

Методы класса:

* public PrismFourVertexArea– заполняет значения векторов;

1. Класс FeatureFace

Данный класс служит для определения характеристик граней в исследовании.

Свойства класса:

* public string name – хранит данные о названии ребра;
* public double force – хранит данные о усилии;
* public readonly Face face– хранит данные о ребре;
* public Color color – хранит информацию о цвете;

Методы класса:

* public Point3D GetCenter – предназначен для поиска центра у грани;
* public void SetColor – устанавливает цвета для граней;
* public Color GetColor – предоставляет значения цвета у грани

1. Класс FeatureFaceManager

Данный класс служит для получения значений исследования граней.

Свойства класса:

* public HashSet<FeatureFace> – хранит в себе значения о свободных ребрах;

Методы класса:

* public FeatureFace GetFeatureFacePerName – получает имя свободного ребра;
* public void DefineFace – получает параметры определенного ребра;

1. Класс Material

Данный класс служит для определения характеристик материала.

Свойства класса:

* public readonly string category – хранит в себе значения о категории материала;
* public readonly string name – хранит в себе название материала;
* public readonly Dictionary<string, double> physicalProperties – хранит в себе значения физических свойств материала;

1. Класс CoordinateSystem

Данный класс служит координатной системой.

Свойства класса:

* MathUtility swMathUtils – хранит в себе значения математической утилиты;

Методы класса:

* public double GetDefaultCoordinateSystem – получает стандартные координаты системы координат;

1. Класс SolidWorksAppWorker

Данный класс служит для управления приложением SolidWorks.

Свойства класса:

* public static SldWorks app – хранит в данные о приложении SolidWorks;
* public const string APP\_NAME – хранит имя приложения;

Методы класса:

* public static void DefineSolidWorksApp() – определяет приложения SolidWorks;
* public static ModelDoc2 DefineActiveSolidWorksDocument – определяет документ в приложенииSolidWorks;
* public static dynamic GetSimulation – получает данные симуляции для фигуры;
* public static void DefineSolidWorksApp() – определяет приложения SolidWorks;

1. Класс SolidWorksDrawer

Данный класс служит для конструирования трехгранных полостей в виде треугольной призмы по векторам.

Методы класса:

* public static void DrawPrism – конструирует на выбранной области полости в виде треугольной призмы по координатам векторов;

Для удобства проектирования и поддержки работы приложения была спроектирована диаграмма классов.

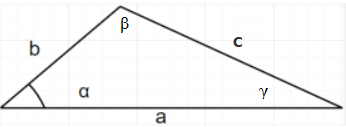
В итоге, при запуске системы автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы главная форма приложения передает основные данные в консоль. В результате чего в SolidWorks начинается генерация массива векторов, по координатам которых происходит построение полостей в виде треугольной призмы по заданным параметрам.

Для построения одной полости в виде треугольной призмы создается «Трехмерный эскиз» с помощью метода «Insert3DSketch» объекта «SketchManager». В нем рисуется треугольное основание с помощью метода «CreateLine» объекта «SketchManager». Далее создается еще один «Трехмерный эскиз» и в нем рисуется прямая направления выреза полости треугольной призмы по двум точкам, где первая точка – центр треугольного основания, а вторая находится путем математических вычислений. Чтобы найти вторую точку линии направления, необходимо по горизонтальной координате умножить длину выреза на тангенс горизонтального угла, а по вертикальной координате умножить длину выреза на тангенс вертикального угла.

После определения основных элементов треугольной призмы с помощью метода «SelectById2» объекта «Extension» выделяются необходимые эскизы методом «FeatureCut4» объекта «FeatureManager». Данный эскиз вытягивается по заданной траектории. Далее в зависимости от выбранных областей и параметров будет построено множество полостей в виде треугольной призмы.

## Описание алгоритма построения выпуклых трехгранных полостей в виде треугольной призмы

При построения каждой призмы используется основание в виде треугольника, которое строится от **основной точки**, которая лежит в основании угла α. Длины сторон **a** и **b** для треугольника в текущей итерации, соответствуют промежуточным значениям **A** и **B**, которые итерационно изменяются между построениями каждой призмы, в зависимости от выбранного метода.



Зная координаты основной точки, и длины прилегающих сторон, мы можем полностью построить треугольник, определяя координаты оставшихся точек, по формулам:

βx = α x  + b \* cos(αугл)

βy = α y  \* (b \* sin(αугл))

γx = αx + a

γy = αy;

Далее для построения призмы, используется значение **глубины выреза**, определяющее высоту призмы, и именной на это значение происходит вытягивание треугольника в треугольную призму.

Все вышеуказанные значения вводит оператор, кроме основных точек, которые определяются в момент исследования детали внутренними средствами SolidWorks, и по итогу для каждой основной точки строится призма с учётом условий, введёнными оператором.

Переходя непосредственно к алгоритму, сначала определяется выбранный метод построения: уменьшение, увеличение, чередование. Значение можно получить из свойства CW.Form1.Choice, которое может иметь значение 0, 1, 2, соответствующее методу построения.

Метод «**Увеличение**» приводит к увеличению **промежуточных значений** **A** и **B** в полях Program.a и Program.b соотвтетвенно, каждую итерацию метода, на значение масштаба, хранящимся в свойстве CW.Form1.ScaleValue, а так же деление на значение unit, для приведения значений к масштабам в документе SolidWorks.

Метод «**Уменьшение**» приводит к уменьшению промежуточных значений каждую итерацию, на тоже значение масштаба, как и в случае с предыдущим методом, и так же деление на значение unit, для приведения значений к масштабам в документе SolidWorks.

В случае с методом «**Чередование**», происходит чередование уменьшения и увеличения промежуточных значений на масштаб, в зависимости от номера итераций, хранящийся в поле Program.k, которое после каждой итерации инкрементируется на единицу. Если номер итерации является чётным числом, что определяется остатком от деления на 2, которое должно быть нулём, то происходит уменьшение, в противном случае происходит увеличение.

Значение **угла α** между сторонами треугольника **a** и **b** вводится оператором, однако его нужно получить в радианах. Для этого мы берём введённое оператором значение, из свойства CW.Form1.AngleValue, делим на число Пи и умножаем на 180, стандартный метод перевода углов в радианы, и записываем значение в переменную alphaRadians.

Далее начинаем рисовать ребра треугольника, на основе которого и будет построена **треугольная** **призма**. Для этого берётся **основная точка** пирамиды конечного элемента area, значение всех осей точки делится на unit, для приведения значений к масштабам в документе SolidWorks, и записываются в переменные x, y и z, соответствующее осям X Y Z.

Для рисования линий, вызывается метод CreateLine у объекта SketchManager, активного документа, который позволяет рисовать линию. Данному методу передаётся 6 параметров: 3 координаты каждой из 2х точек, между которыми строится линия. Для рисования треугольника, метод CreateLine вызывается 3 раза.

В первый раз передаются координаты основной точки два раза, но в параметрах первой точки, значение x увеличивается на промежуточное значение A.

Во второй раз, методу передаётся первыми 3-мя параметрами координаты основной точки, однако координаты второй точки вычисляются следующими действиями:

* Координата Х вычисляется через увеличение координаты Х основной точки на промежуточное значение B, умноженное на косинус угла между сторонами треугольника.
* Координата Y вычисляется через умножение единицы на значение координаты Y увеличенной на промежуточное значение B, которое в свою очередь умножено на синус угла между сторонами треугольника.
* Координата Z берётся без изменений у основной точки.

В третий раз, методу передаются значения точек, вычисленные в прошлые вызовы: параметры первой точки, соответствуют координатам второй точки из второго вызова, а параметры второй точки соответствуют координатам точки из первого вызова метода.

После того как мы получили 3 линии, являющимся рёбрами треугольника, мы объединяем их в треугольник уже в SolidWorks. Для этого перед созданиями точек был вызван метод Insert3DSketch объекта SketchManager, активного документа, с параметром true, что позволило начать редактирование 3D эскиза и связать все линии в одну фигуру, после чего мы ещё раз вызываем этот же метод, но с параметром false, что позволяет завершить редактирования 3D эскиза. Так же перед вызовами метода рекомендуется использовать метод ClearSelection, активного документа, для отмены ненужных выделений.

Однако для создания призмы, нужна ещё одна линия, высота призмы. Для её построения мы снова активируем редактирования 3D эскиза, через вызов метода Insert3DSketch, с передачей ему в качестве параметра значения true. Вызываем метод CreateLine и в качестве параметров передаём ему координаты основной точки два раза, однако в параметрах второй точки координату Z увеличиваем на значение **глубины выреза**, получаемого из свойства CW.Form1.CutDeep, так же не забываем делить на значение unit, для приведения значений к масштабам в документе SolidWorks.

После вышеописанных действий, мы получаем, два трёхмерных эскиза, описывающий нашу призму, и для выреза фигуры по призме, вызывается метод InsertCutSwept4 объекта FeatureManager, активного документа, предварительно выделив наши эскизы, и после чего данный метод построит треугольную призму и сделает вырез по ней основную в основной фигуре.

## Руководство пользователю

Перед запуском программы по автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы необходимо предварительно запустить SolidWorks, открыть любую 3D модель и включить SolidWorks Simulation. Если этого не сделать, то пользователю будет выведено диалоговое окно с подсказкой.

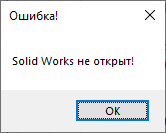


Рисунок 2.3.1 – Диалоговое окно ошибки

После запуска приложения открывается консоль. В данном окне необходимо указать номер области детали, непосредственно в которой будут вырезаться полости в виде треугольной призмы.

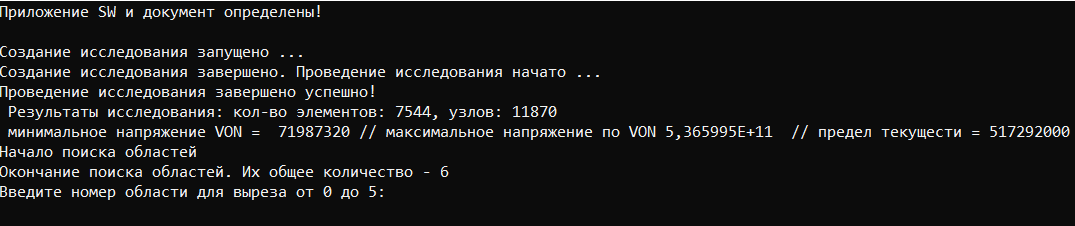


Рисунок 2.3.2 – Консоль приложения

После выбора номер области необходимо нажать кнопку «Ввода» после чего откроется форма для настройки параметров выреза полостей в виде.

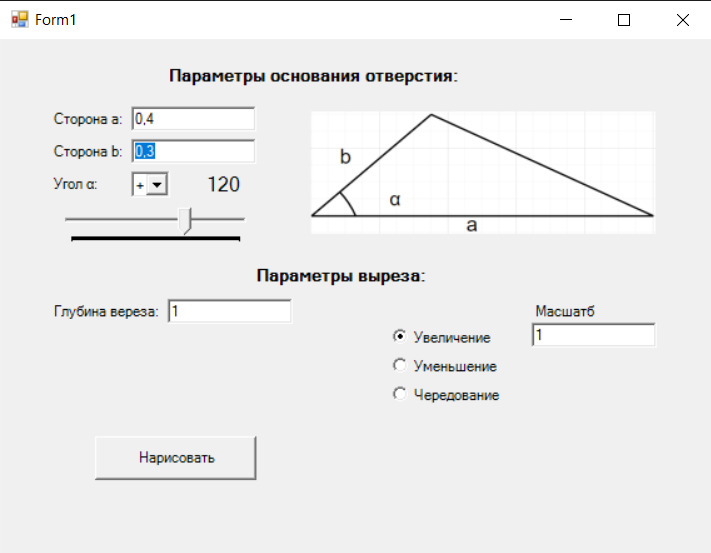


Рисунок 2.3.3 – Окно настройки параметров выреза полостей в виде треугольной призмы.

На данной форме присутствуют следующие настраиваемые поля:

- Сторона a: нижняя сторона треугольного основания треугольной призмы;

- Сторона b: смежная к стороне a сторона треугольного основания треугольной призмы;

- Угол alpha: угол от -180 до 180 градусов между смежными сторонами a и b треугольного основания треугольной призмы;

- Глубина выреза: длина выреза полостей в виде треугольных призм;

- Параметры выреза: паттерн отрисовки вырезов полостей в виде треугольных призм;

- Масштаб: коэффициент размера вырезов полостей в виде треугольных призм;

При неправильном вводе каких-либо параметров пользователю будет выведено диалоговое окно с подсказкой.

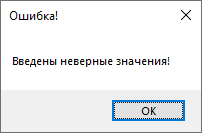


Рисунок 2.3.4 – Диалоговое окно ошибки

После правильного ввода всех необходимых параметров необходимо нажать кнопку «Нарисовать», после чего в SolidWorks начнется автоматизированное построение полостей в виде треугольных призм.

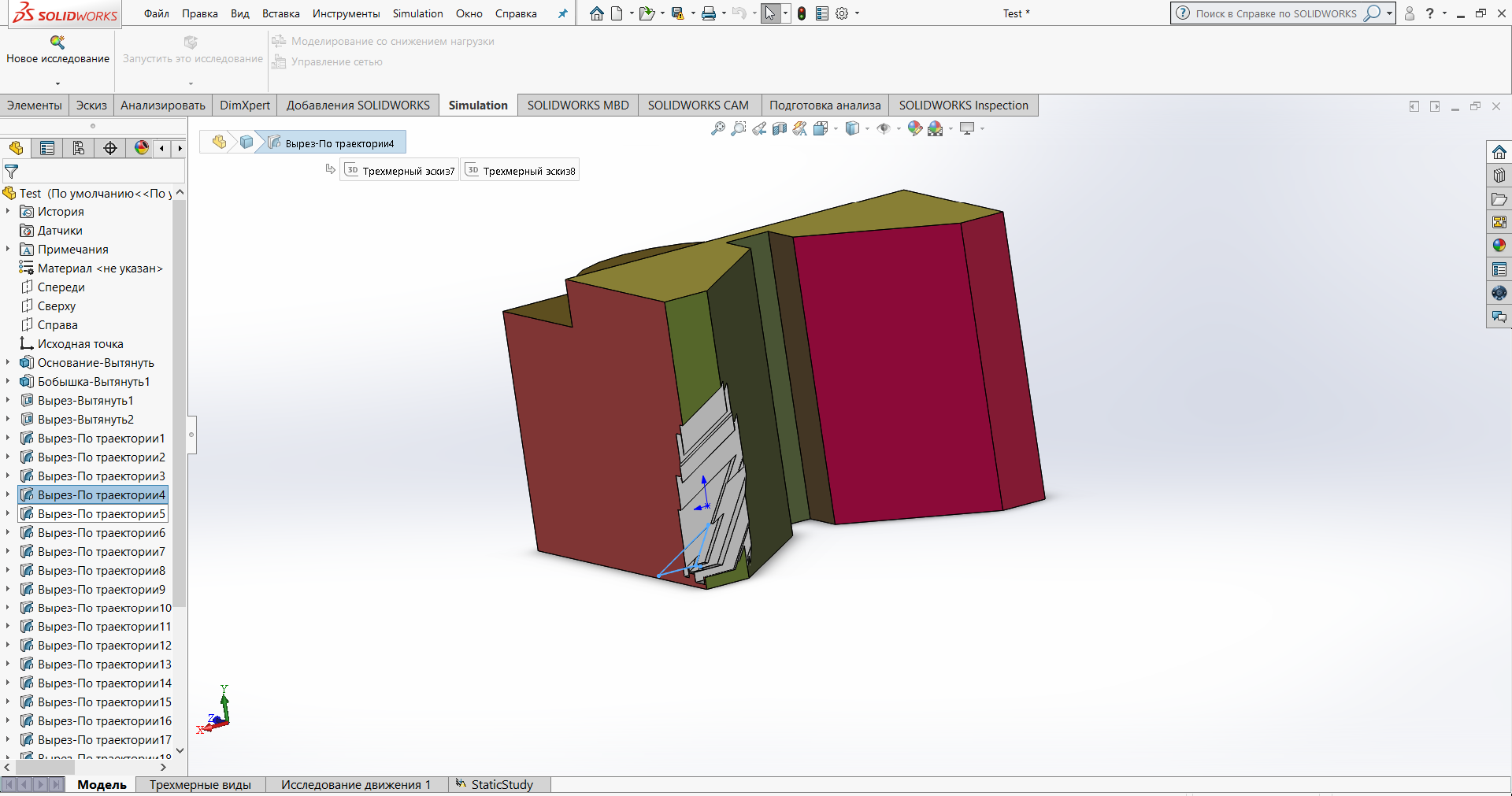


Рисунок 2.3.5 – Результат работы системы автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы

В конечном итоге, пользователю системы автоматизации конструирования 3D модели с выпуклыми трехгранными полостями в виде треугольной призмы был предоставлен доступный и интуитивно понятный интерфейс. Следовательно, программой смогут пользоваться специалисты с разным уровнем квалификации и знаний.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта был выполнен обзор на несколько видов программного обеспечения таких, как Microsoft Visual Studio 2022 и SolidWorks, которые в дальнейшем были использованы для создания системы автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы.

Также была исследована и описана геометрическая конфигурация проектируемой модели (треугольной призмы) и её применение.

Было спроектировано и реализовано приложение для автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы использующее для взаимодействия с приложением SolidWokrs следующие SOLIDWORKS API функции: NewPart, Insert3DSketch, InsertSketch, CreateLine, SelectById, InsertCutSwept. Программа предоставляет возможность выбора различных вариантов построения трехгранных полостей в виде треугольной призмы. Приложение имеет интуитивно понятный интерфейс, разработанный посредством использования языка программирования C#, в среде разработки Microsoft Visual Studio 2022, а также с интеграцией с SolidWokrs. Для приложения было разработано руководство пользователя. Конечный продукт представляет собой полноценную систему автоматизации конструирования выпуклых трехгранных полостей в виде треугольной призмы с продвинутым функционалом.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. SOLIDWORKS API [Электронный ресурс] – Режим доступа: https://help.solidworks.com/2017/english/SolidWorks/sldworks/c\_solidworks\_api.htm?verRedirect=1, свободный. – Загл. с экрана.

2. Алгоритм конструирования ячеек в 3D модели детали с использованием API универсальной CAD системы/ Ю. В. Полозков [и др.] // Наука – образованию, производству, экономике: материалы 15-й Международной научно-технической конференции. - Минск: БНТУ, 2017. - Т. 1. - С. 306.

3. Полозков, Ю. В. Подход к классификации ячеек и ячеистых структур для использования в проектировании легковесных деталей / Ю. В. Полозков, И. П. Климович // Наука – образованию, производству, экономике: материалы 15-й Международной научно-технической конференции. - Минск: БНТУ, 2017. - Т. 1. - С. 325.

# ПРИЛОЖЕНИЕ А

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using App2;

using App2.SolidWorksPackage;

using App2.SolidWorksPackage.Cells;

using ConsoleApp1.SolidWorksPackage.NodeWork;

using SolidWorks.Interop.sldworks;

namespace ConsoleApp1

{

public partial class MainForm : Form

{

int number;

List<ElementArea> areas;

ModelDoc2 doc;

public MainForm(int number, List<ElementArea> areas, ModelDoc2 doc)

{

InitializeComponent();

this.number = number;

this.areas = areas;

this.doc = doc;

}

private void bDraw\_Click(object sender, EventArgs e)

{

if (number >= 0 && number < areas.Count)

{

foreach (var element in areas.ElementAt(number).elements)

{

var elementPyramid = new PyramidFourVertexArea(element.GetDrawingVertexes(1d));

DrawPrism(doc, elementPyramid);

}

}

}

public void DrawPrism(ModelDoc2 modelDoc2, PyramidFourVertexArea area)

{

// Triangle parameters

double a = 0;

double b = 0;

int alpha = alphaAngle.Value;

int vertical = vertAngle.Value;

int horizontal = horAngle.Value;

// Cut parameters

double depth = 0;

int beta = vertAngle.Value;

try

{

a = double.Parse(aSide.Text) / 10000;

b = double.Parse(bSide.Text) / 10000;

depth = double.Parse(depthTb.Text) / 10000;

if (a <= 0 || b <= 0 || depth <= 0)

{

throw new Exception();

}

}

catch (Exception)

{

MessageBox.Show("Введены неверные значения!", "Ошибка!");

return;

}

double unit = 1000;

double vertRadians = (double)vertical \* Math.PI / 180;

double horRadians = (double)horizontal \* Math.PI / 180;

double alphaRadians = (double)alpha \* Math.PI / 180;

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

double x = area.vertex1.x / unit;

double y = area.vertex1.y / unit;

double z = area.vertex1.z / unit;

SketchSegment sk1 = modelDoc2.SketchManager.CreateLine(x + a, y, z, x, y, z);

SketchSegment sk2 = modelDoc2.SketchManager.CreateLine(x, y, z, x + (b \* Math.Cos(alphaRadians)), (y + (b \* Math.Sin(alphaRadians))), z);

SketchSegment sk3 = modelDoc2.SketchManager.CreateLine(x + (b \* Math.Cos(alphaRadians)), (y + (b \* Math.Sin(alphaRadians))), z, x + a, y, z);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

SketchSegment sk;

if (directionCb.SelectedIndex == 0)

{

sk = modelDoc2.SketchManager.CreateLine(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + (depth \* Math.Tan(horRadians)), area.vertex1.y / unit + (depth \* Math.Tan(vertRadians)), area.vertex1.z / unit - depth);

}

else if (directionCb.SelectedIndex == 1)

{

sk = modelDoc2.SketchManager.CreateLine(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + ((depth \* -1d) \* Math.Tan(horRadians)), area.vertex1.y / unit + ((depth \* -1d) \* Math.Tan(vertRadians)), area.vertex1.z / unit - (depth \* -1d));

}

else

{

modelDoc2.SketchManager.CreateLine(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + ((depth / 2) \* Math.Tan(horRadians)), area.vertex1.y / unit + ((depth / 2) \* Math.Tan(vertRadians)), area.vertex1.z / unit - (depth / 2));

}

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

Program.i += 2;

Program.j += 2;

modelDoc2.Extension.SelectByID2("3DSketch" + Program.i, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.Extension.SelectByID2("3DSketch" + Program.j, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.ClearSelection();

//modelDoc2.Extension.SelectByID2("3DSketch" + Program.i, "SKETCH", 0, 0, 0, false, 1, null, 0);

//modelDoc2.Extension.SelectByID2("3DSketch" + Program.j, "SKETCH", 0, 0, 0, true, 4, null, 0);

//modelDoc2.FeatureManager.InsertCutSwept4(false, true, 0, false, false, 0, 0, false, 0, 0, 0, 0, true, true, 0, true, true, true, false);

modelDoc2.Extension.SelectByID2($"3DSketch{Program.i}", "SKETCH", 0, 0, 0, false, 0, null, 0);

//modelDoc2.Extension.SelectByID2($"Line1@3DSketch{4 + id}", "EXTSKETCHSEGMENT", origin.x + (cutLength / 4 \* Math.Tan(alphaRadians)), origin.y + (cutLength / 4 \* Math.Tan(betaRadians)), origin.z + cutLength / 4, true, 16, null, 0);

//modelDoc2.Extension.SelectByID2("3DSketch" + Program.j, "SKETCH", 0, 0, 0, true, 4, null, 0);

if (directionCb.SelectedIndex == 1)

{

modelDoc2.Extension.SelectByID2($"Line1@3DSketch{Program.j}", "EXTSKETCHSEGMENT", area.vertex1.x / unit - (depth / 4 \* Math.Tan(horRadians)), area.vertex1.y / unit - (depth \* Math.Tan(vertRadians)), area.vertex1.z / unit + depth / 4, true, 16, null, 0);

}

else

{

modelDoc2.Extension.SelectByID2($"Line1@3DSketch{Program.j}", "EXTSKETCHSEGMENT", area.vertex1.x / unit + (depth / 4 \* Math.Tan(horRadians)), area.vertex1.y / unit + (depth \* Math.Tan(vertRadians)), area.vertex1.z / unit - depth / 4, true, 16, null, 0);

}

if (directionCb.SelectedIndex == 2)

{

modelDoc2.FeatureManager.FeatureCut4(true, false, false, 6, 0, depth, 0.01, false, false, false, false, 1.74532925199433E-02, 1.74532925199433E-02, false, false, false, false, false, true, true, true, true, false, 0, 0, false, false);

}

else

{

modelDoc2.FeatureManager.FeatureCut4(true, false, false, 0, 0, depth, 0.01, false, false, false, false, 1.74532925199433E-02, 1.74532925199433E-02, false, false, false, false, false, true, true, true, true, false, 0, 0, false, false);

}

//modelDoc2.ClearSelection();

}

private void vertAngle\_Scroll(object sender, EventArgs e)

{

vertText.Text = vertAngle.Value.ToString();

}

private void horAngle\_Scroll(object sender, EventArgs e)

{

horText.Text = horAngle.Value.ToString();

}

private void alphaAngle\_Scroll(object sender, EventArgs e)

{

alphaText.Text = alphaAngle.Value.ToString();

}

private void Form1\_Load(object sender, EventArgs e)

{

directionCb.SelectedIndex = 2;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using SolidWorks.Interop.sldworks;

using SolidWorks.Interop.swconst;

using SolidWorks.Interop.swcommands;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace CW

{

public class SolidWorks

{

private SldWorks sldWorks = null;

private ModelDoc2 modelDoc2;

public SolidWorks()

{

CreatePart();

}

public void DrawTriangle(double x, double y, double z,double r)

{

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Спереди", "PLANE", 0, 0, 0, false, 0, null, 0);

modelDoc2.SketchManager.Insert3DSketch(true);

SketchSegment sk1 = modelDoc2.SketchManager.CreateLine(x + a, y, z, x, y, z);

SketchSegment sk2 = modelDoc2.SketchManager.CreateLine(x, y, z, x + (b \* Math.Cos(alphaRadians)), (y + (b \* Math.Sin(alphaRadians))), z);

SketchSegment sk3 = modelDoc2.SketchManager.CreateLine(x + (b \* Math.Cos(alphaRadians)), (y + (b \* Math.Sin(alphaRadians))), z, x + a, y, z);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.InsertSketch(true);

}

public void DrawDirection(double width, double height, double thickness)

{

width /= 2;

height /= 2;

modelDoc2.SketchManager.InsertSketch(true);

modelDoc2.Extension.SelectByID2("Спереди", "PLANE", 0, 0, 0, false, 0, null, 0);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.CreateLine(-width, height, 0, width, height, 0);

modelDoc2.SketchManager.CreateLine(width, height, 0, width, -height, 0);

if (directionCb.SelectedIndex == 1)

{

modelDoc2.Extension.SelectByID2($"Line1@3DSketch{Program.j}", "EXTSKETCHSEGMENT", area.vertex1.x / unit - (depth / 4 \* Math.Tan(horRadians)), area.vertex1.y / unit - (depth \* Math.Tan(vertRadians)), area.vertex1.z / unit + depth / 4, true, 16, null, 0);

}

else

{

modelDoc2.Extension.SelectByID2($"Line1@3DSketch{Program.j}", "EXTSKETCHSEGMENT", area.vertex1.x / unit + (depth / 4 \* Math.Tan(horRadians)), area.vertex1.y / unit + (depth \* Math.Tan(vertRadians)), area.vertex1.z / unit - depth / 4, true, 16, null, 0);

}

if (directionCb.SelectedIndex == 2)

{

modelDoc2.FeatureManager.FeatureCut4(true, false, false, 6, 0, depth, 0.01, false, false, false, false, 1.74532925199433E-02, 1.74532925199433E-02, false, false, false, false, false, true, true, true, true, false, 0, 0, false, false);

}

else

{

modelDoc2.FeatureManager.FeatureCut4(true, false, false, 0, 0, depth, 0.01, false, false, false, false, 1.74532925199433E-02, 1.74532925199433E-02, false, false, false, false, false, true, true, true, true, false, 0, 0, false, false);

}

modelDoc2.ClearSelection();

modelDoc2.SketchManager.InsertSketch(true);

modelDoc2.FeatureManager.FeatureExtrusion2(true, false, true, 0, 0, thickness, 0.01, false, false, false, false, 0, 0, false, false, false, false, true, true, true, 0, 0, false);

modelDoc2.ClearSelection();

}

public void Cut(double x, double y, double z, int beta,int alpha, double depth)

{

double alphaRadians = (double)alpha \* Math.PI / 100;

double betaRadians = (double)beta \* Math.PI / 180;

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Справа", "PLANE", 0, 0, 0, false, 0, null, 0);

modelDoc2.SketchManager.Insert3DSketch(true);

SketchSegment sk = modelDoc2.SketchManager.CreateLine(x, y, z, x + (depth \* Math.Tan(betaRadians)), y + (depth \* Math.Tan(betaRadians)), z - depth);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз1", "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз2", "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Трехмерный эскиз1", "SKETCH", 0, 0, 0, false, 1, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз2", "SKETCH", 0, 0, 0, true, 4, null, 0);

modelDoc2.FeatureManager.InsertCutSwept4(false, true, 0, false, false, 0, 0, false, 0, 0, 0, 0, true, true, 0, true, true, true, false);

}

private void CreatePart()

{

sldWorks = (SldWorks)Marshal.GetActiveObject("SldWorks.Application");

if (sldWorks.ActiveDoc == null)

{

modelDoc2 = (ModelDoc2)sldWorks.INewPart();

modelDoc2.SetUnits((short)swLengthUnit\_e.swMM, (short)swFractionDisplay\_e.swDECIMAL, 0, 0, false);

}

modelDoc2 = sldWorks.IActiveDoc2;

}

}

}

using SolidWorks.Interop.sldworks;

using System.Collections.Generic;

using System.Linq;

using App2.Simulation.Study;

using App2.SolidWorksPackage.Simulation.FeatureFace;

using App2.SolidWorksPackage.Simulation.MaterialWorker;

using App2.SolidWorksPackage.Simulation.MeshWorker;

using App2.SolidWorksPackage.Simulation.Study;

using System;

using App2.SolidWorksPackage.Cells;

using ConsoleApp1.SolidWorksPackage.NodeWork;

using App2.SolidWorksPackage.NodeWork;

using CW;

using System.Windows.Forms;

namespace App2.SolidWorksPackage

{

internal class SolidWorksObjectDefiner

{

public static HashSet<Face> GetFaces(ModelDoc2 swDoc)

{

HashSet<object> result = new HashSet<object>();

object[] features = swDoc.FeatureManager.GetFeatures(true) as object[];

foreach (Feature feature in features)

{

object[] faces = (object[])feature.GetFaces();

if (faces != null)

{

foreach (Face face in faces)

{

result.Add(face);

}

}

}

return new HashSet<Face>(result.Cast<Face>());

}

// Before start open SolidWorks and <3d Part document>

// with "Simulation" AddIn (NOT "Flow Simulation")

public static void DoResearch()

{

try

{

SolidWorksAppWorker.DefineSolidWorksApp();

var doc = SolidWorksAppWorker.DefineActiveSolidWorksDocument();

Console.WriteLine("Приложение SW и документ определены!\n");

var studyManager = new StudyManager();

var study = studyManager.GetExistingCompletedStudy();

if (study == null)

{

Console.WriteLine("Создание исследования запущено ...");

study = studyManager.CreateStudy(CreateSimpleRecord());

Console.WriteLine("Создание исследования завершено. Проведение исследования начато ...");

study.RunStudy();

Console.WriteLine("Проведение исследования завершено успешно!");

}

else

{

Console.WriteLine("Загружено активное исследование!");

}

var studyResults = study.GetResult();

Console.WriteLine($" Результаты исследования: кол-во элементов: {studyResults.meshElements.Count()}, узлов: {studyResults.nodes.Count()}");

string param = "VON";

//var strainValues = studyResults.DefineMinMaxStrainValues("ESTRN");

var stressValues = studyResults.DefineMinMaxStressValues(param);

double minvalue = stressValues["min"],

maxvalue= 2 \* MaterialManager.GetMaterials()[study.MaterialName].physicalProperties["SIGYLD"];

Console.WriteLine($" минимальное напряжение " +

$"VON = {minvalue} // " +

$"максимальное напряжение по VON {stressValues["max"]}" +

$" // предел текущести = {maxvalue}");

var cutElements = studyResults.DetermineCutElements(param, minvalue, maxvalue) as HashSet<Element>;

//while (cutElements.Count() != 0)

//{

// Console.WriteLine($"Вырез элементов {cutElements.Count()}");

// int i = 0;

// foreach (var element in cutElements)

// {

// if (i > 5) break;

// var elementPyramid = new PyramidFourVertexArea(element.GetDrawingVertexes(0.2));

// SolidWorksDrawer.DrawPyramid(doc, elementPyramid);

// i++;

// }

// Console.WriteLine("Повторное исследование ");

// study.CreateDefaultMesh();

// study.RunStudy();

// Console.WriteLine("Повторные результаты и поиск элементов!");

// studyResults = study.GetResult();

// cutElements = studyResults.DetermineCutElements(param, minvalue, maxvalue);

//}

Console.WriteLine("Начало поиска областей");

var areas = ElementAreaWorker.DefineElementAreas(cutElements);

Console.WriteLine($"Окончание поиска областей. Их общее количество - {areas.Count}");

Console.Write($"Введите номер области для выреза от 0 до {areas.Count-1}: ");

int number = Convert.ToInt32(Console.ReadLine());

Form1 form = new Form1();

Application.Run(form);

if (CW.Form1.flag)

{

Program.i += -1;

Program.j += -1;

}

if (number >= 0 && number < areas.Count)

{

foreach (var element in areas.ElementAt(number).elements)

{

var elementPyramid = new PyramidFourVertexArea(element.GetDrawingVertexes(CW.Form1.interval));

SolidWorksDrawer.DrawPyramid(doc, elementPyramid);

}

SolidWorksDrawer.ClearAllLine(doc);

}

}

catch (Exception ex)

{

Console.WriteLine("" + ex);

}

}

public static StaticStudyRecord CreateSimpleRecord()

{

// Задание сетки и материала

Material material = MaterialManager.GetMaterials()["Медь"];

var mesh = new Mesh();

FeatureFaceManager faceManager = new FeatureFaceManager(

SolidWorksAppWorker.DefineActiveSolidWorksDocument());

// Определение фиксированных граней

faceManager.DefineFace("Грань 1", FaceType.Fixed);

var fixFaces = faceManager.GetFacesPerType(FaceType.Fixed);

// Определение нагруженных граней с силой в 10000000 Н

faceManager.DefineFace("Грань 2", FaceType.ForceLoad, 9900000);

var loadFaces = faceManager.GetFacesPerType(FaceType.ForceLoad);

return new StaticStudyRecord(0, material, fixFaces, loadFaces, mesh);

}

}

}

using App2.SolidWorksPackage.Cells;

using MathNet.Numerics.Distributions;

using SolidWorks.Interop.sldworks;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using CW;

using System.Windows.Forms;

namespace App2.SolidWorksPackage

{

internal class SolidWorksDrawer

{

public static void DrawSimpleTestModel(ModelDoc2 doc)

{

if (doc is null)

{

return;

}

double x0 = 0.05, y0 = 0.05, z0 = 0.05;

// рисуем параллелепипед

// выбор передней плоскости

doc.Extension.SelectByID2("Front Plane", "PLANE", 0, 0, 0, false, 0, null, 0);

doc.SketchManager.InsertSketch(true);

// рисование на плоскости

doc.SketchManager.CreateCornerRectangle(x0, y0, z0, x0 + 0.135, y0 + 0.09, z0);

// вытягивание

doc.FeatureManager.FeatureExtrusion2(true, false, false, 0, 0, 0.082, 0.082, false, false, false, false, z0, z0, false, false, false, false, true, true, true, 0, 0, false);

// выбор передней плоскости

doc.Extension.SelectByID2("Front Plane", "PLANE", 0, 0, 0, false, 0, null, 0);

doc.SketchManager.InsertSketch(true);

doc.SketchManager.CreateArc(x0 + 0.135 / 2 - 0.076 / 2 + 0.038, y0 + 0.082 - 0.05, z0, x0 + (0.135 - 0.076) / 2, y0 + 0.082 - 0.05, z0, x0 + (0.135 + 0.076) / 2, y0 + 0.082 - 0.05, z0, -1);

doc.SketchManager.CreateLine(x0 + (0.135 - 0.076) / 2, y0 + 0.082 - 0.05, z0, x0 + (0.135 - 0.076) / 2, y0 + 0.025, z0);

doc.SketchManager.CreateLine(x0 + (0.135 + 0.076) / 2, y0 + 0.082 - 0.05, z0, x0 + (0.135 + 0.076) / 2, y0 + 0.025, z0);

doc.SketchManager.CreateLine(x0 + (0.135 - 0.076) / 2, y0 + 0.025, z0, x0 + (0.135 + 0.076) / 2, y0 + 0.025, z0);

doc.SketchManager.CreateCornerRectangle(x0, y0 + 0.018, z0, x0 + (0.135 - 0.104) / 2, y0 + 0.09, z0);

doc.SketchManager.CreateCornerRectangle(x0 + (0.135 - 0.104) / 2 + 0.104, y0 + 0.018, z0, x0 + 0.135, y0 + 0.09, z0);

doc.FeatureManager.FeatureCut3(false, false, false, 1, 0, 0.09, 0.09, false, false, false, false, 1.74532925199433E-02, 1.74532925199433E-02,

false, false, false, false, false, true, true, true, true, false, 0, 0, false);

doc.ClearSelection2(true);

}

public static void DrawPyramid(ModelDoc2 modelDoc2, PyramidFourVertexArea area)

{

int alpha = CW.Form1.alpha1;

int beta = CW.Form1.beta1;

double unit = 1000;

double r = CW.Form1.r1 / 1000f;

double depth = CW.Form1.depth1 / 1000f;

double depth2 = (-1)\*CW.Form1.depth2 / 1000f;

double alphaRadians = (double)alpha \* Math.PI / 180;

double betaRadians = (double)beta \* Math.PI / 180;

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

SketchSegment sk1 = modelDoc2.SketchManager.CreateCircle(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + r, area.vertex1.y / unit, area.vertex1.z / unit);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(false);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

if (CW.Form1.flag)

{

SketchSegment sk = modelDoc2.SketchManager.CreateLine(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + (depth \* Math.Tan(betaRadians)), area.vertex1.y / unit + (depth \* Math.Tan(alphaRadians)), area.vertex1.z / unit - depth);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(false);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

SketchSegment sk2 = modelDoc2.SketchManager.CreateLine(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + (depth2 \* Math.Tan(betaRadians)), area.vertex1.y / unit + (depth2 \* Math.Tan(alphaRadians)), area.vertex1.z / unit - depth2);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

Program.i += 3;

Program.j += 3;

Program.k += 3;

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.i, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.j, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.i, "SKETCH", 0, 0, 0, false, 1, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.j, "SKETCH", 0, 0, 0, true, 4, null, 0);

modelDoc2.FeatureManager.InsertCutSwept4(false, true, 0, false, false, 0, 0, false, 0, 0, 0, 0, true, true, 0, true, true, true, false);

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.i, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.k, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.i, "SKETCH", 0, 0, 0, false, 1, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.k, "SKETCH", 0, 0, 0, true, 4, null, 0);

modelDoc2.FeatureManager.InsertCutSwept4(false, true, 0, false, false, 0, 0, false, 0, 0, 0, 0, true, true, 0, true, true, true, false);

}

else

{

SketchSegment sk = modelDoc2.SketchManager.CreateLine(area.vertex1.x / unit, area.vertex1.y / unit, area.vertex1.z / unit, area.vertex1.x / unit + (depth \* Math.Tan(betaRadians)), area.vertex1.y / unit + (depth \* Math.Tan(alphaRadians)), area.vertex1.z / unit - depth);

modelDoc2.ClearSelection();

modelDoc2.SketchManager.Insert3DSketch(true);

Program.i += 2;

Program.j += 2;

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.i, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.j, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.ClearSelection();

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.i, "SKETCH", 0, 0, 0, false, 1, null, 0);

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + Program.j, "SKETCH", 0, 0, 0, true, 4, null, 0);

modelDoc2.FeatureManager.InsertCutSwept4(false, true, 0, false, false, 0, 0, false, 0, 0, 0, 0, true, true, 0, true, true, true, false);

}

//modelDoc2.Extension.SelectByID2("Спереди", "PLANE", 0, 0, 0, false, 0, null, 0);

//;

//SketchSegment sk1 = modelDoc2.SketchManager.CreateCircle(0, 0.028921, 0, 0.010565, 0.016508, 0);

//SketchSegment sk1 = modelDoc2.SketchManager.CreateLine(x + a, y, z, x, y, z);

//SketchSegment sk2 = modelDoc2.SketchManager.CreateLine(x, y, z, x + (b \* Math.Cos(alphaRadians)), sign \* (y + (b \* Math.Sin(alphaRadians))), z);

//SketchSegment sk3 = modelDoc2.SketchManager.CreateLine(x + (b \* Math.Cos(alphaRadians)), sign \* (y + (b \* Math.Sin(alphaRadians))), z, x + a, y, z);

//modelDoc2.SketchManager.InsertSketch(true);

//double betaRadians = (double)30 \* Math.PI / 180;

//modelDoc2.ClearSelection();

//modelDoc2.Extension.SelectByID2("Справа", "PLANE", 0, 0, 0, false, 0, null, 0);

//modelDoc2.SketchManager.InsertSketch(true);

//SketchSegment sk = modelDoc2.SketchManager.CreateLine(sketchPoint.X, sketchPoint.Y, sketchPoint.Z, sketchPoint.X + 10, sketchPoint.Y + (10 \* Math.Tan(betaRadians)), sketchPoint.Z);

//modelDoc2.ClearSelection();

//modelDoc2.SketchManager.InsertSketch(true);

//modelDoc2.Extension.SelectByID2("Эскиз5", "SKETCH", 0, 0, 0, true, 0, null, 0);

//modelDoc2.Extension.SelectByID2("Эскиз6", "SKETCH", 0, 0, 0, true, 0, null, 0);

//modelDoc2.ClearSelection();

//modelDoc2.Extension.SelectByID2("Эскиз5", "SKETCH", 0, 0, 0, false, 1, null, 0);

//modelDoc2.Extension.SelectByID2("Эскиз6", "SKETCH", 0, 0, 0, true, 4, null, 0);

//modelDoc2.FeatureManager.InsertCutSwept4(false, true, 0, false, false, 0, 0, false, 0, 0, 0, 0, true, true, 0, true, true, true, false);

}

public static void ClearAllLine(ModelDoc2 modelDoc2)

{

for (int i = 0; i < 100; i++)

{

modelDoc2.Extension.SelectByID2("Трехмерный эскиз" + i, "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.EditDelete();

}

modelDoc2.Extension.SelectByID2("Трехмерный эскиз162" , "SKETCH", 0, 0, 0, true, 0, null, 0);

modelDoc2.EditDelete();

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

using App2.exceptions;

using App2.SolidWorksPackage;

using App2.util;

using SolidWorks.Interop.sldworks;

using SolidWorks.Interop.swconst;

namespace App2

{

internal class SolidWorksAppWorker

{

public static SldWorks app;

public static List<object> comList;

public const string APP\_NAME = "SldWorks.Application";

public static void DefineSolidWorksApp()

{

try

{

app = Marshal.GetActiveObject("SldWorks.Application") as SldWorks;

} catch (COMException)

{

throw new NotSWAppFoundException();

}

}

public static void OpenSolidWorksApp()

{

comList = RotManager.GetRunningInstances(APP\_NAME);

if (comList.Count == 0)

{

app = Activator.CreateInstance(Type.GetTypeFromProgID(APP\_NAME)) as SldWorks;

app.Visible = true;

app.LoadAddIn(app.GetExecutablePath() + @"\Simulation\cosworks.dll");

var coordinateSystem = new CoordinateSystem(app.GetMathUtility() as MathUtility);

}

}

public static void CloseSolidWorksApp()

{

foreach (Process proc in Process.GetProcessesByName("sldProcMon"))

{

proc.Kill();

}

foreach (Process proc in Process.GetProcessesByName("SLDWORKS"))

{

proc.Kill();

}

}

public static ModelDoc2 DefineActiveSolidWorksDocument()

{

if (app == null)

{

throw new NotSWAppFoundException();

}

ModelDoc2 swDoc = (ModelDoc2)app.ActiveDoc;

if (swDoc == null)

{

throw new NotSWDocumentFoundException();

}

int pref\_toggle = (int)swUserPreferenceToggle\_e.swInputDimValOnCreate;

app.SetUserPreferenceToggle(pref\_toggle, false);

return swDoc;

}

public static void OpenDocument(string path)

{

int fileError = default(int);

int fileWarning = default(int);

//Open doc

ModelDoc2 swDoc = app.OpenDoc6(

path,

(int)swDocumentTypes\_e.swDocPART,

(int)swOpenDocOptions\_e.swOpenDocOptions\_Silent,

null,

ref fileError,

ref fileWarning

);

if (swDoc != null)

{

//Set the working directory to the document directory

string pathName = swDoc.GetPathName();

app.SetCurrentWorkingDirectory(pathName.Substring(0, pathName.LastIndexOf("\\")));

}

}

public static void CreateNewDocument()

{

if (app != null)

app.NewPart();

}

//Получить путь к файлам с материалами

public static string[] GetPathsMaterialDataBase()

{

if (app == null)

{

DefineSolidWorksApp();

}

object[] result = app.GetMaterialDatabases() as object[];

return result.Cast<string>().ToArray();

}

public static dynamic GetSimulation()

{

if (app == null)

{

DefineSolidWorksApp();

}

int swVersion = Convert.ToInt32(app.RevisionNumber().Substring(0, 2));

dynamic COSMOSObject = app.GetAddInObject($"SldWorks.Simulation.{swVersion - 15}");

return COSMOSObject == null ? null : COSMOSObject.CosmosWorks;

}

}

}

using SolidWorks.Interop.sldworks;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using MathNet.Numerics.LinearAlgebra;

using App2.util.mathutils;

namespace App2.SolidWorksPackage

{

public class CoordinateSystem

{

//public double[,] matrix;

MathUtility swMathUtils;

public CoordinateSystem(MathUtility swMathUtils)

{

this.swMathUtils = swMathUtils;

//this.matrix = GetDefaultCoordinateSystem();

}

public CoordinateSystem()

{

}

public double[,] Multiply(double[,] matrix1, double[,] matrix2)

{

Matrix<double> m1 = Matrix<double>.Build.DenseOfArray(matrix1);

Matrix<double> m2 = Matrix<double>.Build.DenseOfArray(matrix2);

Matrix<double> result = m1 \* m2;

return result.ToArray();

}

public double[,] CreateCoordinateSystem(Vector3D vectorX, Vector3D vectorY, Vector3D vectorZ)

{

double[,] matrix = new double[,] {

{ vectorX.x, vectorX.y, vectorX.z, 0 },

{ vectorY.x, vectorY.y, vectorY.z, 0 },

{ vectorZ.x, vectorZ.y, vectorZ.z, 0 },

{ 0, 0, 0, 1 }

};

return matrix;

}

public double[,] GetDefaultCoordinateSystem()

{

return new double[,] {

{ 1, 0, 0, 0 },

{ 0, 1, 0, 0 },

{ 0, 0, 1, 0 },

{ 0, 0, 0, 1 }

};

}

public double[,] MoveTo(double[,] matrix, Point3D point)

{

double[,] DCS = GetDefaultCoordinateSystem();

DCS[0, 3] = point.x;

DCS[1, 3] = point.y;

DCS[2, 3] = point.z;

Matrix<double> mDCS = Matrix<double>.Build.DenseOfArray(DCS);

Matrix<double> mCDCS = Matrix<double>.Build.DenseOfArray(matrix);

Matrix<double> result = mDCS \* mCDCS;

return result.ToArray();

}

public double[,] RotateAroundX(double[,] matrix, double radians)

{

MathTransform mathTransform = swMathUtils.CreateTransformRotateAxis(

swMathUtils.CreatePoint(new double[] { matrix[0, 3], matrix[1, 3], matrix[2, 3] }) ,

swMathUtils.CreateVector(new double[] { matrix[0, 0], matrix[0, 1], matrix[0, 2] }),

radians) as MathTransform;

return GetMatrix(mathTransform.ArrayData as double[]);

}

public double[,] RotateAroundY(double[,] matrix, double radians)

{

MathTransform mathTransform = swMathUtils.CreateTransformRotateAxis(

swMathUtils.CreatePoint(new double[] { matrix[0, 3], matrix[1, 3], matrix[2, 3] }),

swMathUtils.CreateVector(new double[] { matrix[1, 0], matrix[1, 1], matrix[1, 2] }),

radians) as MathTransform;

return GetMatrix(mathTransform.ArrayData as double[]);

}

public double[,] RotateAroundZ(double[,] matrix, double radians)

{

MathTransform mathTransform = swMathUtils.CreateTransformRotateAxis(

swMathUtils.CreatePoint(new double[] { matrix[0, 3], matrix[1, 3], matrix[2, 3] }),

swMathUtils.CreateVector(new double[] { matrix[2, 0], matrix[2, 1], matrix[2, 2] }),

radians) as MathTransform;

return GetMatrix(mathTransform.ArrayData as double[]);

}

public Vector3D TransformVector(double[,] matrix, Vector3D vector)

{

Vector3D result = new Vector3D();

result.x = vector.x \* matrix[0, 0] + vector.y \* matrix[0, 1] + vector.z \* matrix[0, 2] + matrix[0, 3];

result.y = vector.x \* matrix[1, 0] + vector.y \* matrix[1, 1] + vector.z \* matrix[1, 2] + matrix[1, 3];

result.z = vector.x \* matrix[2, 0] + vector.y \* matrix[2, 1] + vector.z \* matrix[2, 2] + matrix[2, 3];

return result;

}

public Point3D TransformPoint(double[,] matrix, Point3D point)

{

Point3D result = new Point3D();

result.x = point.x \* matrix[0, 0] + point.y \* matrix[0, 1] + point.z \* matrix[0, 2] + matrix[0, 3];

result.y = point.x \* matrix[1, 0] + point.y \* matrix[1, 1] + point.z \* matrix[1, 2] + matrix[1, 3];

result.z = point.x \* matrix[2, 0] + point.y \* matrix[2, 1] + point.z \* matrix[2, 2] + matrix[2, 3];

return result;

}

public double[,] GetMatrix(double[] transformMatrix)

{

double[,] matrix = new double[,] {

{ transformMatrix[0], transformMatrix[1], transformMatrix[2], transformMatrix[9] },

{ transformMatrix[3], transformMatrix[4], transformMatrix[5], transformMatrix[10] },

{ transformMatrix[6], transformMatrix[7], transformMatrix[8], transformMatrix[11] },

{ 0, 0, 0, 1 }

};

return matrix;

}

public double[] GetTransformMatrix(double[,] matrix)

{

double[] transformMatrix = new double[] {

matrix[0,0], matrix[0,1], matrix[0,2],

matrix[1,0], matrix[1,1], matrix[1,2],

matrix[2,0], matrix[2,1], matrix[2,2],

matrix[0,3], matrix[1,3], matrix[2,3],

0, 0, 0,

1

};

return transformMatrix;

}

}

}

using App2.SolidWorksPackage.NodeWork;

using App2.util.mathutils;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace App2.SolidWorksPackage.Cells

{

public class PyramidFourVertexArea

{

public readonly Point3D vertex1;

public readonly Point3D vertex2;

public readonly Point3D vertex3;

public readonly Point3D vertex4;

public PyramidFourVertexArea() {}

public PyramidFourVertexArea(

Point3D vertex1,

Point3D vertex2,

Point3D vertex3,

Point3D vertex4)

{

this.vertex1 = vertex1;

this.vertex2 = vertex2;

this.vertex3 = vertex3;

this.vertex4 = vertex4;

}

public PyramidFourVertexArea(IEnumerable<Point3D> vertexes)

{

if (vertexes.Count() != 4)

{

throw new ArgumentException("PyramidFourVertexArea must have 4 vertexes," +

$"given {vertexes.Count()}");

}

this.vertex1 = vertexes.ElementAt(0);

this.vertex2 = vertexes.ElementAt(1);

this.vertex3 = vertexes.ElementAt(2);

this.vertex4 = vertexes.ElementAt(3);

}

public PyramidFourVertexArea(IEnumerable<Node> vertexes)

{

if (vertexes.Count() != 4)

{

throw new ArgumentException("PyramidFourVertexArea must have 4 vertexes," +

$"given {vertexes.Count()}");

}

this.vertex1 = vertexes.ElementAt(0).point;

this.vertex2 = vertexes.ElementAt(1).point;

this.vertex3 = vertexes.ElementAt(2).point;

this.vertex4 = vertexes.ElementAt(3).point;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace App2.exceptions

{

internal class NotSWAppFoundException : Exception

{

public override string ToString()

{

return "SolidWorks instanse was not found!";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace App2.exceptions

{

internal class NotSWDocumentFoundException:Exception

{

public override string ToString()

{

return "SolidWorks document was not found!";

}

}

}

using App2.util.mathutils;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace App2.SolidWorksPackage.NodeWork

{

public class Element

{

public readonly int number;

//Первые 4 нода это вершины, остальные 6 это центры ребер

public readonly IEnumerable<Node> nodes;

//Верщины

public readonly IEnumerable<Node> vertexNodes;

//Центр области

public readonly Point3D center;

public Element() { }

public Element(int number, IEnumerable<Node> nodes, Point3D center)

{

this.number = number;

this.nodes = nodes;

this.vertexNodes = new List<Node>() {

nodes.ElementAt(0) ,

nodes.ElementAt(1) ,

nodes.ElementAt(2) ,

nodes.ElementAt(3) };

this.center = center;

}

public bool Contains(Node node)

{

return Contains(nodes, node);

}

private static bool Contains(IEnumerable<Node> nodes, Node node)

{

return nodes.Contains(node);

}

public bool isAdjacent(Element element)

{

IEnumerable<Node> result = vertexNodes.Intersect(element.vertexNodes);

return result?.Count() >= 3;

}

public IEnumerable<Point3D> GetDrawingVertexes(double coefficient=0.03)

{

return MathHelper.GetLessCoordinatesOfPyramid

(

new List<Point3D>()

{

vertexNodes.ElementAt(0).point,

vertexNodes.ElementAt(1).point,

vertexNodes.ElementAt(2).point,

vertexNodes.ElementAt(3).point

},

center,

coefficient

);

}

public HashSet<Node> DefineInsideNodes(IEnumerable<Node> nodes)

{

var insideNodes = new HashSet<Node>();

foreach(var item in nodes)

{

if (this.IsNodeInside(item))

insideNodes.Add(item);

}

return insideNodes;

}

public bool IsNodeInside(Node node)

{

double maxDistance = MathHelper.DefineMaxVertexDistanceFromPyramidCenter(

GetNodexCoords(), center);

double realDistance = MathHelper.DefineDistanceBetweenPoints(node.point, center);

return realDistance < maxDistance;

}

public List<Point3D> GetNodexCoords()

{

var coords = new List<Point3D>();

foreach (var node in vertexNodes)

{

coords.Add(node.point);

}

return coords;

}

public override string ToString()

{

var sb = new StringBuilder();

sb.AppendLine($"Element {number} \n[");

foreach (var node in vertexNodes)

{

sb.AppendLine(node.ToString());

}

sb.AppendLine($"]");

return sb.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using SolidWorks.Interop.sldworks;

using SolidWorks.Interop.swconst;

using App2.util.mathutils;

namespace App2.SolidWorksPackage.Simulation.FeatureFace

{

public class FeatureFace

{

public string name;

public double force;

public readonly Face face;

public FaceType type;

private Entity entity;

public Color color;

public Vector3D GetNormal()

{

double[] normal = (double[]) face.Normal;

return new Vector3D(normal[0], normal[1], normal[2]);

}

public Point3D GetCenter()

{

double[] UVBounds = (double[])face.GetUVBounds();

double centerU = (UVBounds[0] + UVBounds[1]) / 2;

double centerV = (UVBounds[2] + UVBounds[3]) / 2;

Surface surface = face.GetSurface() as Surface;

double[] param = (double[]) surface.Evaluate(centerU, centerV, 0, 0);

return new Point3D(param[0] \* 1000, param[1] \* 1000, param[2] \* 1000);

}

public FeatureFace()

{

this.entity = face as Entity;

type = FaceType.NoneType;

}

public FeatureFace(Face face, string name) :base()

{

this.name = name;

this.face = face;

this.color = GetColor();

}

public FeatureFace(Face face, string name, Color color) : this(face, name)

{

this.color = color;

SetColor(color);

}

public void SetColor(Color color)

{

double[] param = face.MaterialPropertyValues as double[];

if (param == null)

{

param = new double[9] {

0, 0, 0,

1, 1, 0.5,

0.4, 0, 0

};

}

param[0] = color.R / 255f;

param[1] = color.G / 255f;

param[2] = color.B / 255f;

face.MaterialPropertyValues = param;

}

public Color GetColor()

{

double[] param = face.MaterialPropertyValues as double[];

if (param == null)

{

param = new double[3];

}

Color color = Color.FromArgb(

Convert.ToInt32(param[0] \* 255),

Convert.ToInt32(param[1] \* 255),

Convert.ToInt32(param[2] \* 255)

);

return color;

}

public Point3D[] GetVertixs()

{

object[] edges = face.GetEdges() as object[];

HashSet<Point3D> vertices = new HashSet<Point3D>();

if (edges != null)

{

foreach (Edge edge in edges)

{

double[] start = edge.IGetStartVertex().GetPoint() as double[];

double[] end = edge.IGetEndVertex().GetPoint() as double[];

vertices.Add(new Point3D(start[0] \* 1000, start[1] \* 1000, start[2] \* 1000));

vertices.Add(new Point3D(end[0] \* 1000, end[1] \* 1000, end[2] \* 1000));

}

}

return vertices.ToArray();

}

public void Select(bool append = false)

{

entity.Select(append);

}

public void DeSelect()

{

entity.DeSelect();

}

public override string ToString()

{

return name;

}

}

} using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using SolidWorks.Interop.sldworks;

using SolidWorks.Interop.swconst;

using App2.util.mathutils;

namespace App2.SolidWorksPackage.Simulation.FeatureFace

{

public class FeatureFace

{

public string name;

public double force;

public readonly Face face;

public FaceType type;

private Entity entity;

public Color color;

public Vector3D GetNormal()

{

double[] normal = (double[]) face.Normal;

return new Vector3D(normal[0], normal[1], normal[2]);

}

public Point3D GetCenter()

{

double[] UVBounds = (double[])face.GetUVBounds();

double centerU = (UVBounds[0] + UVBounds[1]) / 2;

double centerV = (UVBounds[2] + UVBounds[3]) / 2;

Surface surface = face.GetSurface() as Surface;

double[] param = (double[]) surface.Evaluate(centerU, centerV, 0, 0);

return new Point3D(param[0] \* 1000, param[1] \* 1000, param[2] \* 1000);

}

public FeatureFace()

{

this.entity = face as Entity;

type = FaceType.NoneType;

}

public FeatureFace(Face face, string name) :base()

{

this.name = name;

this.face = face;

this.color = GetColor();

}

public FeatureFace(Face face, string name, Color color) : this(face, name)

{

this.color = color;

SetColor(color);

}

public void SetColor(Color color)

{

double[] param = face.MaterialPropertyValues as double[];

if (param == null)

{

param = new double[9] {

0, 0, 0,

1, 1, 0.5,

0.4, 0, 0

};

}

param[0] = color.R / 255f;

param[1] = color.G / 255f;

param[2] = color.B / 255f;

face.MaterialPropertyValues = param;

}

public Color GetColor()

{

double[] param = face.MaterialPropertyValues as double[];

if (param == null)

{

param = new double[3];

}

Color color = Color.FromArgb(

Convert.ToInt32(param[0] \* 255),

Convert.ToInt32(param[1] \* 255),

Convert.ToInt32(param[2] \* 255)

);

return color;

}

public Point3D[] GetVertixs()

{

object[] edges = face.GetEdges() as object[];

HashSet<Point3D> vertices = new HashSet<Point3D>();

if (edges != null)

{

foreach (Edge edge in edges)

{

double[] start = edge.IGetStartVertex().GetPoint() as double[];

double[] end = edge.IGetEndVertex().GetPoint() as double[];

vertices.Add(new Point3D(start[0] \* 1000, start[1] \* 1000, start[2] \* 1000));

vertices.Add(new Point3D(end[0] \* 1000, end[1] \* 1000, end[2] \* 1000));

}

}

return vertices.ToArray();

}

public void Select(bool append = false)

{

entity.Select(append);

}

public void DeSelect()

{

entity.DeSelect();

}

public override string ToString()

{

return name;

}

}

} using System;

using System.Collections.Generic;

using System.Linq;

using System.Drawing;

using SolidWorks.Interop.sldworks;

namespace App2.SolidWorksPackage.Simulation.FeatureFace

{

public class FeatureFaceManager

{

public HashSet<FeatureFace> freeFaces;

private static Random rand = new Random(unchecked((int)(DateTime.Now.Ticks)));

public FeatureFaceManager()

{

freeFaces = new HashSet<FeatureFace>();

}

public FeatureFaceManager(ModelDoc2 swDoc)

{

freeFaces = GetFeatureFaces(SolidWorksObjectDefiner.GetFaces(swDoc));

swDoc.GraphicsRedraw2();

}

public FeatureFaceManager(IEnumerable<Face> faces)

{

freeFaces = GetFeatureFaces(faces);

}

public FeatureFaceManager(IEnumerable<FeatureFace> faces)

{

freeFaces = faces.ToHashSet();

}

public FeatureFace GetFeatureFacePerName(string name)

{

return freeFaces.Where(x => x.name.Equals(name)).FirstOrDefault();

}

public void DefineFace(string name, FaceType type, double loadValue = 0)

{

var face = GetFeatureFacePerName(name);

face.type = type;

face.force = loadValue;

}

public IEnumerable<FeatureFace> GetFacesPerType(FaceType type)

{

return from face in freeFaces

where face.type.Equals(type)

select face;

}

private static string GetName(int index)

{

return "Грань " + index;

}

private static Color GetColor()

{

int r = rand.Next(100, 255);

int g = rand.Next(0, 255);

int b = rand.Next(55, 100);

return Color.FromArgb(r, g, b);

}

private static HashSet<FeatureFace> GetFeatureFaces(IEnumerable<Face> faces)

{

HashSet<FeatureFace> result = new HashSet<FeatureFace>();

int index = 1;

foreach (Face face in faces)

{

string faceName = GetName(index);

Color faceColor = GetColor();

FeatureFace featureFace = new FeatureFace(face, faceName, faceColor);

result.Add(featureFace);

index++;

}

return result;

}

}

} using System.Collections.Generic;

namespace App2.SolidWorksPackage.Simulation.MaterialWorker

{

public class Material

{

public readonly string category;

public readonly string name;

public readonly Dictionary<string, double> physicalProperties;

//// Модуль упругости

//public readonly double EX;

//// Коэффициент Пуассона

//public readonly double NUXY;

//// Модуль сдвига

//public readonly double GXY;

//// Коэффициент теплового расширения

//public readonly double ALPX;

//// Массовая плотность

//public readonly double DENS;

//// Теплопроводность

//public readonly double KX;

//// Удельная теплоемкость

//public readonly double C;

//// Предел прочности при растяжении

//public readonly double SIGXT;

//// Предел текучести

//public readonly double SIGYLD;

public Material() { }

public Material(string category, string name, double[] physicalProperties)

{

this.category = category;

this.name = name;

this.physicalProperties = new Dictionary<string, double>();

this.physicalProperties.Add("EX", physicalProperties[0]);

this.physicalProperties.Add("NUXY", physicalProperties[1]);

this.physicalProperties.Add("GXY", physicalProperties[2]);

this.physicalProperties.Add("ALPX", physicalProperties[3]);

this.physicalProperties.Add("DENS", physicalProperties[4]);

this.physicalProperties.Add("KX", physicalProperties[5]);

this.physicalProperties.Add("C", physicalProperties[6]);

this.physicalProperties.Add("SIGXT", physicalProperties[7]);

this.physicalProperties.Add("SIGYLD", physicalProperties[8]);

}

public Material(

string category,

string name,

double EX,

double NUXY,

double GXY,

double ALPX,

double DENS,

double KX,

double C,

double SIGXT,

double SIGYLD)

{

this.category = category;

this.name = name;

physicalProperties = new Dictionary<string, double>();

physicalProperties.Add("EX", EX);

physicalProperties.Add("NUXY", NUXY);

physicalProperties.Add("GXY", GXY);

physicalProperties.Add("ALPX", ALPX);

physicalProperties.Add("DENS", DENS);

physicalProperties.Add("KX", KX);

physicalProperties.Add("C", C);

physicalProperties.Add("SIGXT", SIGXT);

physicalProperties.Add("SIGYLD", SIGYLD);

}

public override string ToString()

{

return this.name;

}

}

}

using System;

using System.Collections.Generic;

using System.Xml;

namespace App2.SolidWorksPackage.Simulation.MaterialWorker

{

public class MaterialManager

{

private const string XML\_NODE\_MATERIAL\_CATEGORY = "classification";

private const string XML\_NODE\_MATERIAL\_CATEGORY\_ATTR\_NAME = "name";

private const string XML\_NODE\_MATERIAL = "material";

private const string XML\_NODE\_MATERIAL\_ATTR\_NAME = "name";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE = "physicalproperties";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_EX = "EX";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_NUXY = "NUXY";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_GXY = "GXY";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_ALPX = "ALPX";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_DENS = "DENS";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_KX = "KX";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_C = "C";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_SIGXT = "SIGXT";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_SIGYLD = "SIGYLD";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_ATTR\_NAME = "displayname";

private const string XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_ATTR\_VALUE = "value";

public static Dictionary<string, Material> GetMaterials()

{

var materialDictionary = new Dictionary<string, Material>();

var paths = SolidWorksAppWorker.GetPathsMaterialDataBase();

foreach (var path in paths)

{

foreach (Material material in GetMaterialsFromPath( path))

{

materialDictionary.Add(material.name, material);

}

}

return materialDictionary;

}

private static List<Material> GetMaterialsFromPath(string path)

{

var xDoc = new XmlDocument();

xDoc.Load(path);

var xRoot = xDoc.DocumentElement;

List<Material> materials = new List<Material>();

foreach (XmlNode xNode in xRoot)

{

if (xNode.Name == XML\_NODE\_MATERIAL\_CATEGORY)

{

string category = xNode.Attributes.GetNamedItem(XML\_NODE\_MATERIAL\_CATEGORY\_ATTR\_NAME).Value;

materials.AddRange(GetMaterialsFromXmlChildNodes(xNode.ChildNodes, category));

}

}

return materials;

}

private static HashSet<Material> GetMaterialsFromXmlChildNodes(XmlNodeList xNodeList, string categoryName)

{

HashSet<Material> result = new HashSet<Material>();

foreach (XmlNode xNode in xNodeList)

{

if (xNode.Name == XML\_NODE\_MATERIAL)

{

string materialName = xNode.Attributes.GetNamedItem(XML\_NODE\_MATERIAL\_ATTR\_NAME).Value;

double[] physicalProperties = GetMaterialPhysicalProperties(xNode.ChildNodes);

Material material = new Material(categoryName, materialName, physicalProperties);

result.Add(material);

}

}

return result;

}

private static double[] GetMaterialPhysicalProperties(XmlNodeList xNodeList)

{

double[] result = new double[9];

foreach (XmlNode xNode in xNodeList)

{

if (xNode.Name == XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE)

{

XmlNodeList xNodeListPhysicalProperties = xNode.ChildNodes;

result[0] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_EX);

result[1] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_NUXY);

result[2] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_GXY);

result[3] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_ALPX);

result[4] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_DENS);

result[5] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_KX);

result[6] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_C);

result[7] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_SIGXT);

result[8] = GetMaterialPhysicalPropertieValue(xNodeListPhysicalProperties,

XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_SIGYLD);

break;

}

}

return result;

}

private static double GetMaterialPhysicalPropertieValue(XmlNodeList xNodeList, string propertie)

{

double result = 0;

foreach (XmlNode xNode in xNodeList)

{

if (xNode.Name == propertie)

{

string value = xNode.Attributes.GetNamedItem(XML\_NODE\_MATERIAL\_PHYSICAL\_PROPERTIE\_ATTR\_VALUE).Value;

value = value.Replace('.', ',');

result = Convert.ToDouble(value);

break;

}

}

return result;

}

//public TreeNode[] GetTreeNodeView()

//{

// List<TreeNode> result = new List<TreeNode>();

// Dictionary<string, TreeNode> categories = new Dictionary<string, TreeNode>();

// foreach (string name in materials.Keys)

// {

// Material material = materials[name];

// string category = material.category;

// TreeNode categoryNode;

// if (!categories.TryGetValue(category, out categoryNode))

// {

// categoryNode = new TreeNode(category);

// categories.Add(category, categoryNode);

// result.Add(categoryNode);

// }

// categoryNode.Nodes.Add(name);

// }

// return result.ToArray();

//}

}

}

using App2.util.mathutils;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace App2.SolidWorksPackage.NodeWork

{

public class Node

{

public readonly int number;

public readonly Point3D point;

public readonly StressNode stress;

public readonly StrainNode strain;

public Node(int number , Point3D point, StressNode stress, StrainNode strain) {

this.number = number;

this.point = point;

this.stress = stress;

this.strain = strain;

}

public override string ToString()

{

string text = String.Format("Node:{0} X:{1:f5} Y:{2:f5} Z:{3:f5}",

number,

point.x,

point.y,

point.z);

//text += $"\n-Stress:\n{stress}\n-Strain:\n{strain}";

return text;

}

}

}