МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЄКТУВАННЯ 3D-МОДЕЛЕЙ ЛОПАТОК ГАЗОТУРБІННОГО ДВИГУНА У СИСТЕМІ АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ SIEMENS NX»**

Виконав: студент 4 курсу, групи 6.1220
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
М.О. Слатвінський
(ініціали та прізвище)
Керівник старший викладач кафедри комп'ютерних наук Циммерман Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії , к.ф.-м.н., доцент Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., доцент

_____ Шило Г.М.
(підпис)

" 22 " грудня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Слатвінському Максиму Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка програмного забезпечення для автоматизації проєктування 3D-моделей лопаток газотурбінного двигуна у системі автоматизованого проєктування Siemens NX
керівник роботи Циммерман Геннадій Анатолійович
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 15.05.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Створення моделі лопатки газотурбінного двигуна

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 25.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	11.01.2024	
2.	Збір вихідних даних.	01.02.2024	
3.	Обробка методичних та теоретичних	06.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	15.04.2024	
5.	Розробка третього розділу.	13.05.2024	
6.	Оформлення та нормоконтроль	05.06.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	23.06.2024	

Студент _____ М.О. Слатвінський
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.А. Циммерман
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка програмного забезпечення для автоматизації проєктування 3D-моделей лопаток газотурбінного двигуна у системі автоматизованого проєктування Siemens NX»: 60 с., 13 рис., 8 джерел, 3 додатків.

ГАЗОТУРБІННИЙ ДВИГУН, ЛОПАТКА, МОДЕЛЮВАННЯ, МОДЕЛЬ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Об’єкт дослідження – система для тривимірного моделювання Siemens NX, у якій за допомогою додатків з’являється можливість для автоматизування процесів моделювання.

Мета роботи: ознайомитися з процесом моделювання, дізнатися про програмне забезпечення для 3D моделювання, отримати знання про його застосування, створити модель лопатки засобами автоматизованого проєктування Siemens NX.

Метод дослідження – описовий, аналіз.

Кваліфікаційна робота містить опис технологічного процесу використання системи Siemens NX для моделювання лопатки газотурбінного двигуна.Зазначений технологічний процес уточнюється за допомогою програмних засобів, розроблених додатково мовою C++. Практична частина містить етапи моделювання та отримання результатів.Внаслідок виконаної роботи отримано перевірену процедуру моделювання об'єкту, який можна використовувати у наступній ланці виробничого процесу.

SUMMARY

Bachelor’s Qualifying Theses «Development of software for automating the design of 3D models of gas turbine engine blades in the Siemens NX automated design system»: 60 pages, 13 figures, 8 references, 3 supplements.

GAS TURBINE ENGINE, BLADE, SIMULATION, MODEL, SOFTWARE

The object of the research is the Siemens NX three-dimensional modeling system, in which, with the help of applications, there is an opportunity to automate modeling processes.

The purpose of the work: to get acquainted with the modeling process, to learn about the software for 3D modeling, to gain knowledge about its application, to create a model of a blade using Siemens NX automated design.Method of research.

Methods of research: descriptive, analysis.

The qualification work contains a description of the technological process of using the Siemens NX system for modeling the blade of a gas turbine engine. The specified technological process is specified by using software tools developed additionally in the C++ language. The practical part contains modeling stages and obtaining results. As a result of the work performed, a proven object modeling procedure was obtained, which can be used in the next link of the production process.

ЗМІСТ

ВСТУП

Сучасний процес моделювання все більше починає розвиватися і створення різновидних моделей стає дедалі різноманітнішим. Тому процес створення моделей стає необхідним для багатьох галузей. Обрана тема розробки програмного забезпечення було обрана з метою спрощення процесу моделювання обраної деталі.

Вибір саме такої теми для розробки був зумовлений необхідністю автоматизації процесу моделювання сучасних лопаток газотурбінного двигуна, які є необхідною частиною даного механізму.

Вибір програмного забезпечення Siemens NX був зумовлений тим, що це спеціалізований засіб для створення моделей та механізмів. Наявність великого вибору інструментів для редагування дає можливість робити та змінювати модель як того потрібно людині яка цим займається. Завдяки такому різноманіттю можливостей в застосунку створення моделей стає набагато легше.

Основною метою кваліфікаційної роботи є ознайомлення з процесом моделюванням та його видами, а також подальший процес автоматизації проєктування, який полягає у використанні по сторонніх програмних застосунків, які допомагають скоротити час та спростити процес створення моделі.

1 МОДЕЛЮВАННЯ

1.1 Що таке моделювання

Моделювання – це спосіб дослідження будь-яких явищ, процесів або об'єктів шляхом побудови й аналізу їх моделей. У широкому розумінні моделювання є однією з основних категорій теорії пізнання і мало не єдиним науково обґрунтованим методом наукових досліджень систем і процесів будь-якої природи в багатьох сферах людської діяльності.[1] В нинішній час моделювання в багатьох випадках допомагає створити та вирішити різноманітні ситуації з якими стається людство, які за різних обставин не можуть бути реалізовані у реальному житті.

1.2 Види моделювання

У моделювання є достатньо багато видів моделювання (рис. 1.1) у яких присутньо багато відмінностей за характерними ознаками та між їх застосуванням в залежності від потреб конкретної задачі. В нашому випадку ми будемо використовувати один із цих видів – макетування, вчасності будемо використовувати 3D моделювання за допомогою програмного забезпечення, яке дозволяє створювати моделі на електронно-обчислювальній машині.

1.3 Переваги моделювання над реальністю

Моделювання та реалізація проектів в реальному житті являються невід’ємною складовою великих підприємств. Тож для більшої результативності та успіху в розробці вихідного продукту його моделюють та

проводять необхідні тестування за допомогою різноманітного програмного забезпечення, яке значно заощаджує зусилля, час та кошти, необхідні для виготовлення продукту.

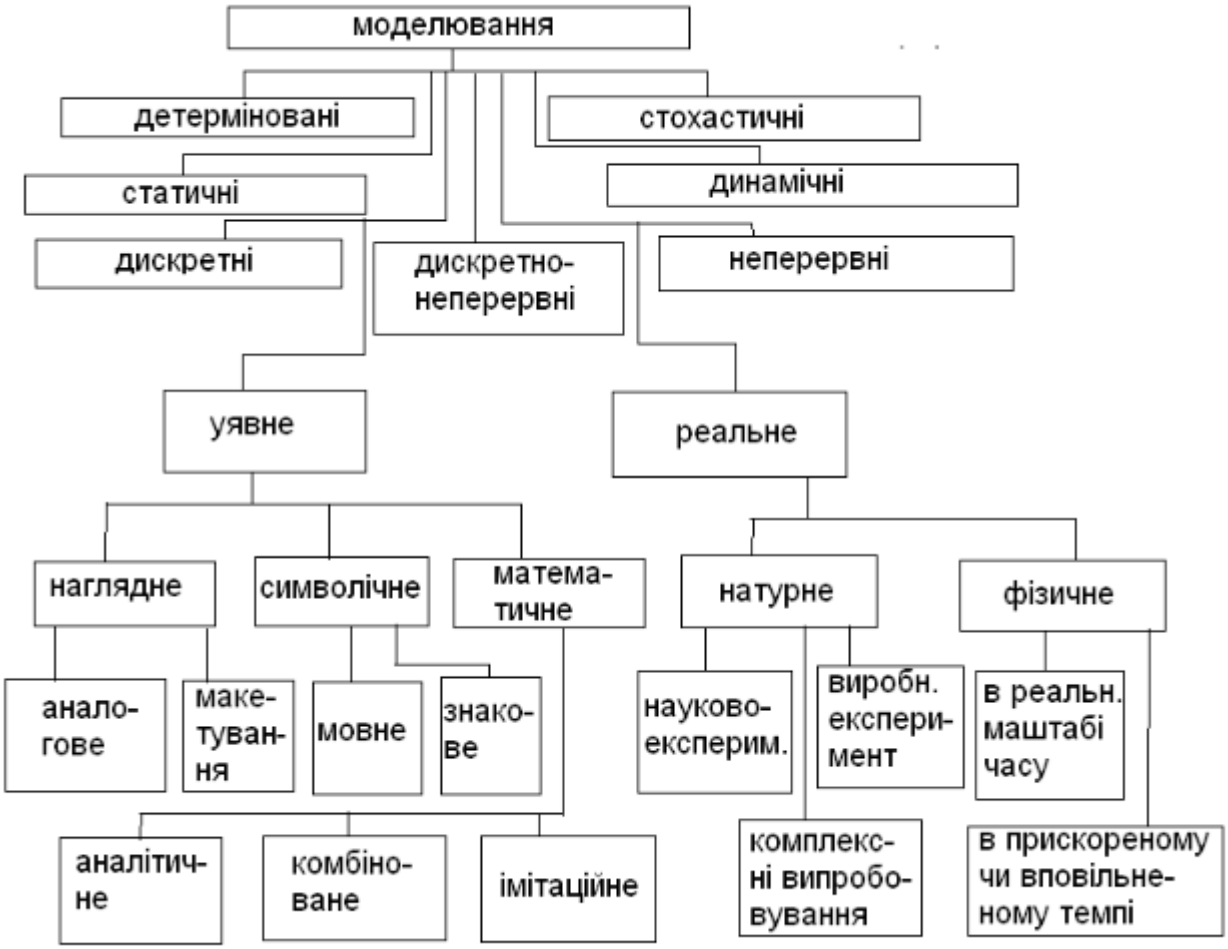

Рисунок 1.1 – Різновид моделювання

2 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ 3D МОДЕЛЮВАННЯ

2.1 Різновид програмного забезпечення

В нинішній час для 3D моделювання є багато доступного програмного забезпечення, яке може надати безліч можливостей у самих різноманітних сферах діяльності. В залежності від потреб які потрібно втілити користувачу, він може використовувати необхідне йому програмне забезпечення. Якщо в проекті нахил йде в графічний дизайн та анімацію, то краще буде використовувати програмне забезпечення таке як Adobe Substance 3D, 3ds Max, Blender (рис. 2.1). В випадку коли діяльність буде спеціалізована на розробку механізмів та окремих деталей, гарним вибором є Fusion 360, AutoCAD, Siemens NX (рис. 2.2), SolidWorks.

В залежності від специфікації роботи з якою необхідно буде мати справу є можливість підібрати програмне забезпечення яке доповнить та спростить процес виконання поставленої задачі.

Наприклад Siemens NX є системою тривимірного моделювання, в якій інженер може створювати вироби будь-якого ступеня складності. Для позначення систем такого класу використовується абревіатура CAD/CAM/CAE.

Підсистема CAD (Computer-Aided Design) – проектування за допомогою комп'ютера. Призначена для розробки проектно-конструкторської документації (моделювання деталей та складання, креслення, аналіз, оптимізація конструкції тощо). [8]

Підсистема CAM (Computer-aided manufacturing) – Виготовлення за допомогою комп'ютера. Вона призначена для автоматизованої підготовки програм для верстатів з ЧПУ на основі математичної моделі деталі, створеної в CAD-підсистемі. [8]

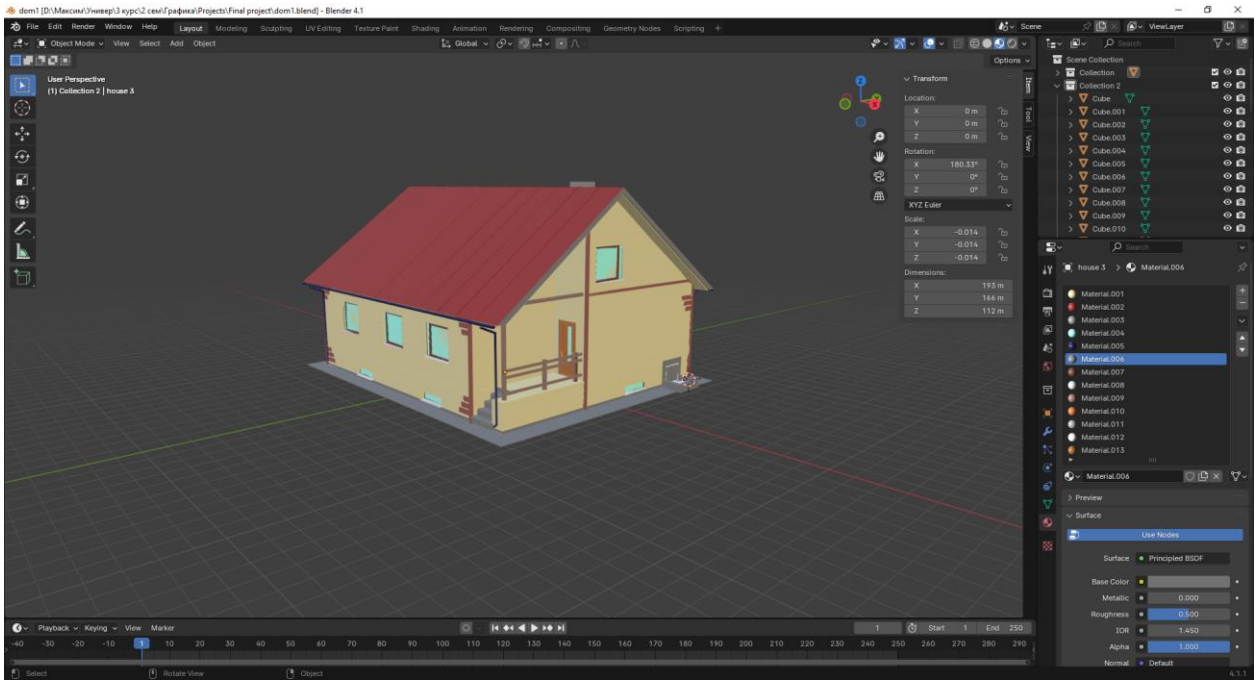

Рисунок 1.1 – Моделювання у Blender

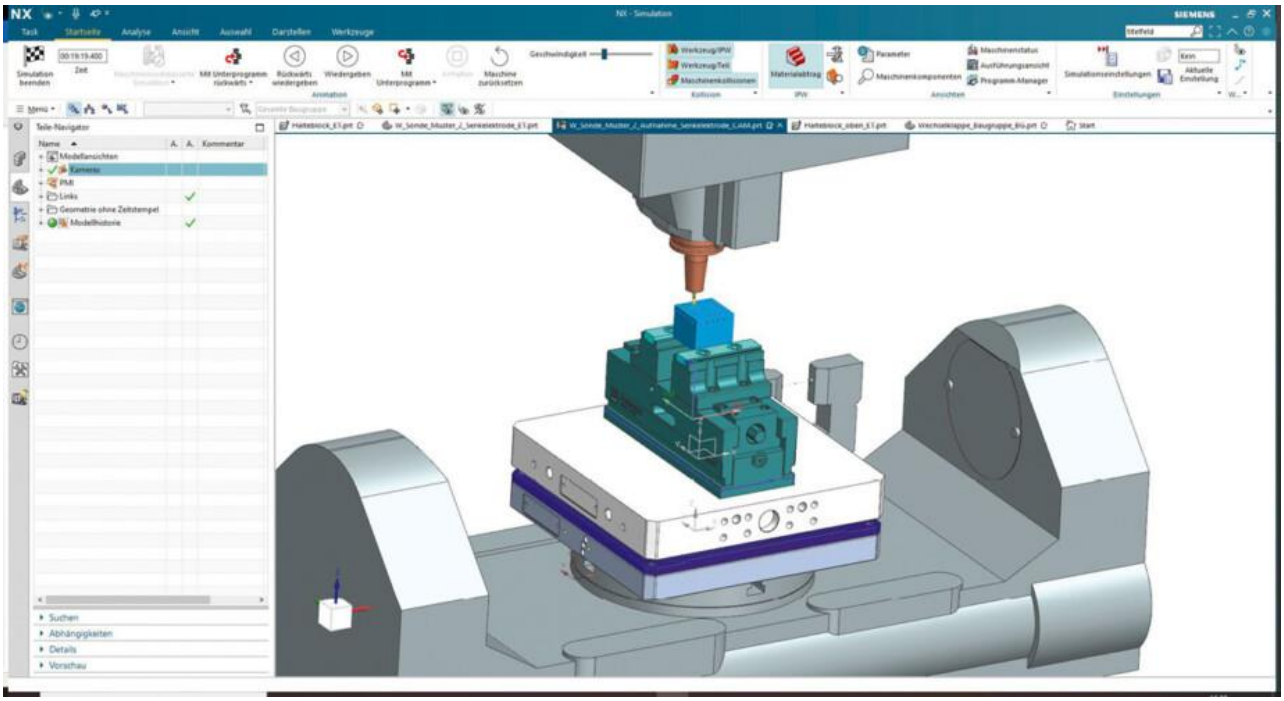

Рисунок 2.2 – Моделювання у Siemens NX

Підсистема CAE (Computer-aided engineering) – інженерний аналіз. Ця підсистема дозволяє за допомогою розрахункових методів (метод кінцевих елементів, метод кінцевих різниць, метод кінцевих обсягів) оцінити, як поведеться цифрова модель виробу в реальних умовах експлуатації. Вона

забезпечує симуляцію процесів та перевірку працездатності виробу без великих витрат часу та коштів.[8]

Ці системи дозволяють одразу створювати тривимірні об'єкти, а вже по них – плоскі креслення. Отже, процес розробки плоского креслення нині майже повністю автоматизовано. У деяких випадках стадія розробки креслення взагалі опускається, а за допомогою CAM-програм виробляється генерація машинних кодів для верстатів з ЧПУ, на яких виготовляється кінцевий виріб. [8]

2.2 Модернізація за допомогою 3D

Особливу роль 3D моделювання відіграє у конструкторській діяльності де необхідно створювати моделі з точними параметрами, які надалі будуть виготовлятися за допомогою спеціалізованого обладнання. Завдяки такому підходу до праці підприємство може розробляти нові моделі виробів або модернізувати вже існуючі продукти та робити їх ще більше прогресивними в експлуатації використовуючи новітні розробки та технології.

Також за допомогою цього програмного забезпечення можливо перевірити рухливі можливості механізмів, чи будуть деталі конфліктувати між собою, а також за допомогою додатків до програми з’являється можливість спостерігати тестовим процесом фрезерування деталей у симуляції. Не менш важливим є і симуляція реальних навантажень на деталь або механізм, що також можливо перевірити за допомогою спеціального програмного забезпечення, наприклад такого як Ansys Mechanical (рис. 2.3), Nastran, Dassault Systèmes Simulia Abaqus, що є світовими лідерами у цьому напрямку.[2]

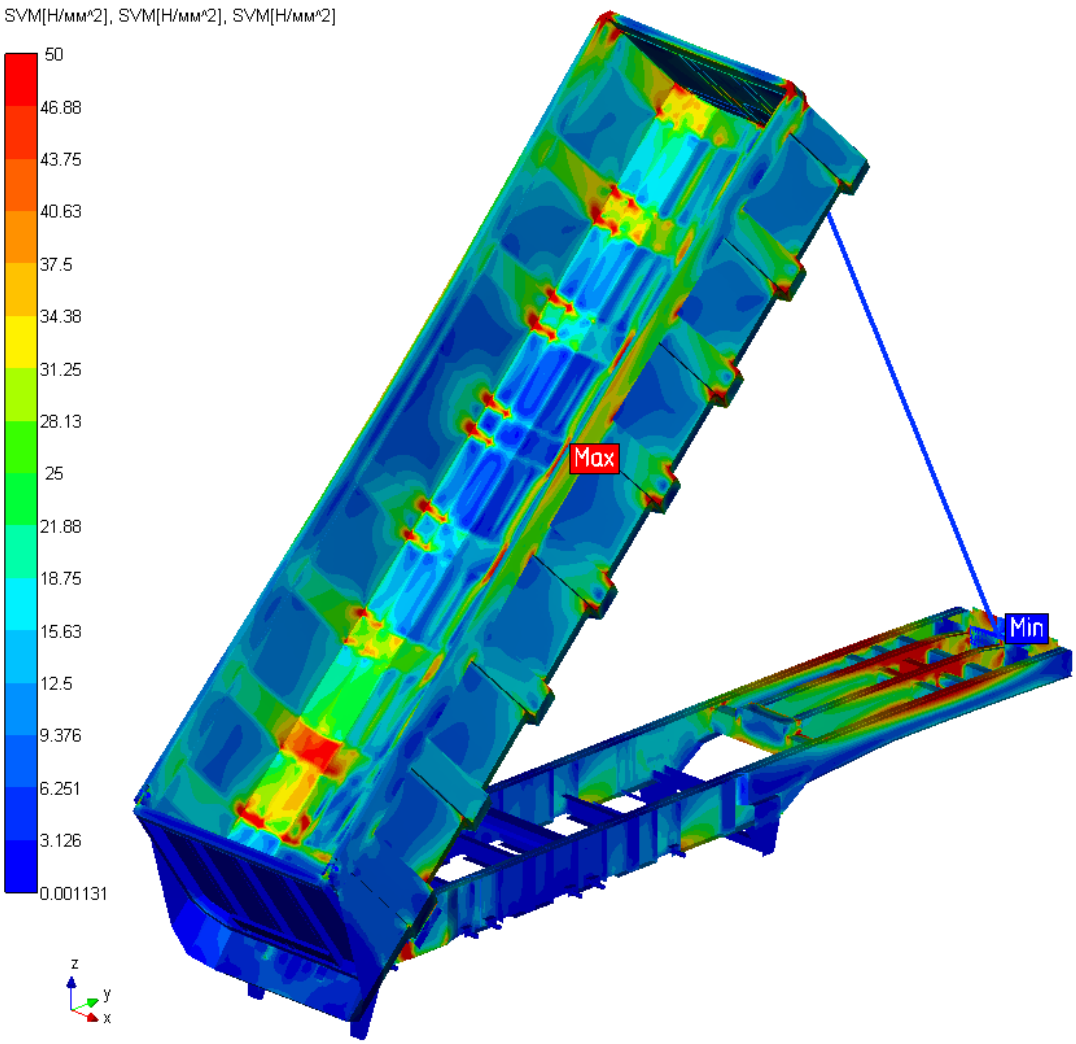

Рисунок 2.3 – Симуляція фізичних навантажень

3 ЗАСТОСУВАННЯ ТА ВИГОТОВЛЕННЯ ЛОПАТКИ

3.1 Роль лопатки в газотурбінному двигуні

Однією з основних компонуючих деталей газотурбінного двигуна (рис. 3.1) є лопатка. В залежності від їх застосування вони бувають двох видів – компресорна та турбінна. Лопатки схожі за структурою, але в залежності від ступені де вони знаходиться можуть бути різних розмірів та структур. Компресор стискає повітря до необхідних величин, після чого повітря надходить у камеру згоряння, де підігрівається до необхідної температури за рахунок згоряння палива і далі газ, що вже вийшов, надходить на турбіну, де віддає частину енергії обертаючи її, а інша частина при подальшому розгоні газу в реактивному соплі. перетворюється на імпульс тяги, що й штовхає літак вперед.[3]

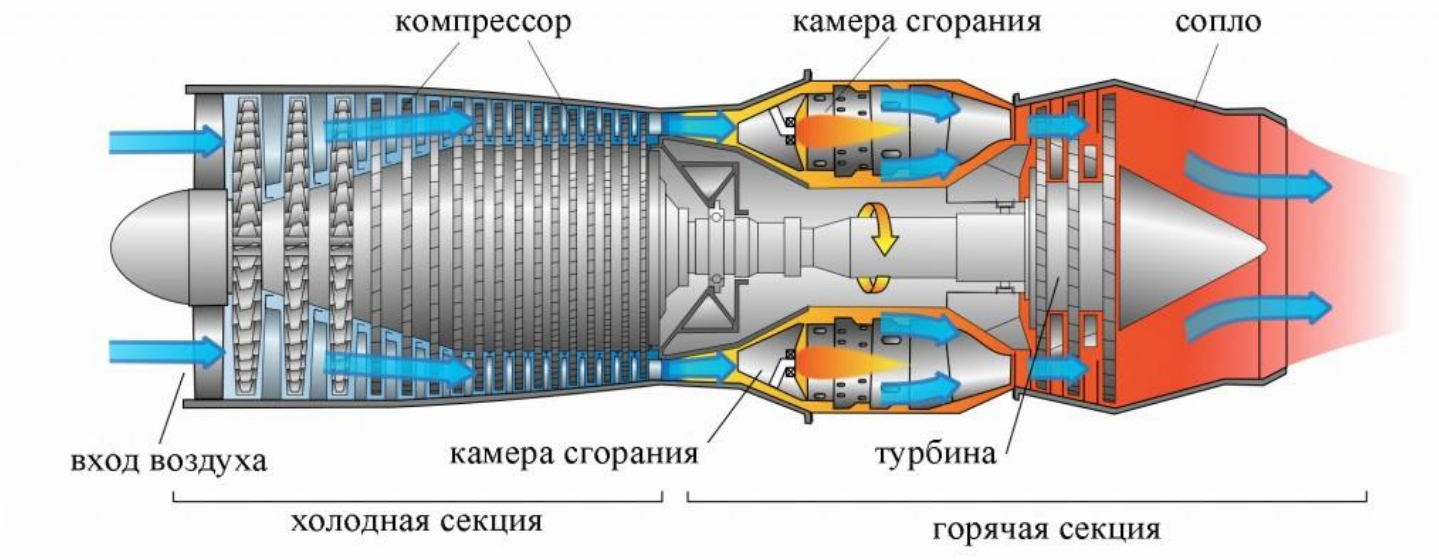

Рисунок 3.1 – Схема газотурбінного двигуна

Оскільки у секції турбінних лопаток йде робота з високими температурами, приблизно 1260°C, то їх структура відрізняється від звичайних лопаток та має систему для охолоджування. Повітря для охолодження

відбирається від вхідного пристрою, охолоджується шляхом змішування з водою або паливом і подається у внутрішні порожнини турбінних лопаток через вал двигуна та відцентрові канали компресора, вбудованого в робоче колесо турбіни.[4]

3.2 Матеріал для виготовлення

Лопатки компресора є одним з найбільш відповідальних і масових виробів двигуна який має ресурс роботи від кількох годин до кількох десятків тисяч годин, відчувають широкий спектр впливів від динамічних та статичних напруг, високотемпературного газового потоку, що містить абразивні частинки, а також окисні продукти довкілля та згоряння. палива. При цьому необхідно відзначити, що в залежності від географічного місця експлуатації та режиму роботи двигуна температура його тракту коливається від -50 ... -40 С ° до 700 ... 800 ° в компресорі. Як конструкційні матеріали для лопаток компресора сучасних газотурбінних двигунів застосовують титанові сплави (ВТ22, ВТ3-1, ВТ6, ВТ-8, ВТ-33), жароміцні сталі (ЕІ-961, ЕП-517Ш), а для лопаток турбіни ливарні сплави на нікелевій основі (ЖС6У, ЖС32).[5]

Для більш ефективного охолодження була розроблена конструкція лопаток з каналами складної конфігурації і вихідними отворами, що забезпечують виникнення тонкої повітряної плівки навколо лопатки, виключаючи зіткнення розжарених газів з матеріалом лопатки, що виключає нагрівання лопатки і охолодження відхідних газів, і відпрацьованих газів, внаслідок чого збільшується ККД двигуна.[6]

3.3 Процес виготовлення

Виготовлення лопаток здійснюють методом лиття за моделями, що виплавляються, існуючі на сьогоднішній день технології дозволяють отримувати литу заготовку, що не вимагає механічної обробки, передбачається тільки шліфування і нанесення захисного покриття. Це особливо важливо, через те що нікелеві та титанові сплави для лопаток практично не піддаються механічній обробці. Охолоджуючі канали формують керамічними стрижнями. Раніше для їх виробництва використовували оксид кремнію, зараз віддають перевагу оксиду алюмінію (Al2O3), що має високу температуру плавлення – 2050°C. [6]

Заготівлю переміщують в зону обробки і сканують геометричні параметри заготівлі в ортогональних координатах в поперечному, поздовжньому і вертикальному напрямках за допомогою вимірювального модуля. Отримані дані передають блок управління, в якому виробляють математичну обробку даних, порівняння з теоретичною формою деталі і визначення безлічі областей для обробки заготовки за допомогою робочого інструменту при його переміщенні по трьох ортогональних координат. Здійснюють зняття поточних параметрів обробки заготовки за допомогою силомоментного датчика, які передаються блок управління для вироблення команд управління. При цьому на основі пристрою встановлений портал з розміщеними на ньому заготовкою, робочим інструментом, вимірювальним блоком та приводами переміщення інструменту та заготівлі, пов'язаними з вимірювальним блоком та блоком керування. В результаті забезпечується високопродуктивна фінішна обробка поверхонь лопаток газотурбінного двигуна.[7]

Завдяки такій обробці вихідного виробу вдається виявити та мінімізувати кількість деталей які можуть бути не використані у виробництві подальшого механізму.

4 МОДЕЛЮВАННЯ ЛОПАТКИ

4.1 Програмний код забезпечення для автоматизації побудови

В додатку А реалізовано запуск процесу автоматозованної побудови координатних точок за якими в подальшому процесі буде побудована лопатка газотурбінного двигуна. Внаслідок запуску програмного коду починається процес запуску програмного коду з додатку Б у якому реалізовано процес читання данних з файлу, у якому знаходяться координати. Додаток В представляє собою програмний код, який надає необхідні координати, які будуть використовуватися при побудові координатних точок програмному забезпеченні Siemens NX.

4.2 Процес моделювання

Початок процесу моделювання розпочинається з вибору програмного забезпечення та підготовки робочого середовища в якому буде розроблюватися модель. В нашому випадку будемо використовувати програмне забезпечення для автоматизованого моделювання Siemens NX (попередня назва «Unigraphics»).

Процес розробки моделі розпочинається зі створення нового файлу, в якому буде створено майбутню модель.

Оскільки при побудові лопатки необхідна точність, то замість ручного створення моделі ми будемо використовувати програмний код (рис. 4.1). Використовуючи утиліту, яка читатиме файл з даними, та за допомогою бібліотеки NXOpen ми отримуємо точки (рис. 4.2), які знаходяться у тривимірній координатній площині та формують модель лопатки.

```
// Чтение из файла
        while ((fscanf( stream, "%s\n", XYZ )) != EOF)
        {
            // кол-во узлов
            sscanf (XYZ, "%d\n", &NNodes);
            i_size = (i_sech + 1) * NNodes;
            i_pnt0 = i_sech * NNodes;
// Выделение памяти
            PntSp = (struct pts_struct *)realloc((void *)PntSp,sizeof(PntSp[0])*i_size);
            PntKor = (struct pts_struct *)realloc((void *)PntKor,sizeof(PntKor[0])*i_size);
            // спинка
            for ( i = 0; i < NNodes; i++)
            {
                i_pnt = i_pnt0 + i;
                fscanf( stream, "%lf%lf%lf\n", &xx, &yy, &zz) ;
                PntSp[i_pnt].X = xx;
                PntSp[i_pnt].Y = yy;
                PntSp[i_pnt].Z = zz;
            }
            fscanf( stream, "%d\n", &NNodes );
            if (PntSp[0].X < PntSp[NNodes - 1].X)
                DirPositiv = 1;
            else
                DirPositiv = 0;
            i_pnt0 = i_sech * NNodes;
            // корыто
            for ( i = 0; i < NNodes; i++)
            {
                i_pnt = i_pnt0 + i;
                fscanf( stream, "%lf%lf%lf\n", &xx, &yy, &zz) ;
                PntKor[i_pnt].X = xx;
                PntKor[i_pnt].Y = yy;
                PntKor[i_pnt].Z = zz;
            }
```

Рисунок 4.1 – Частина коду з додатку

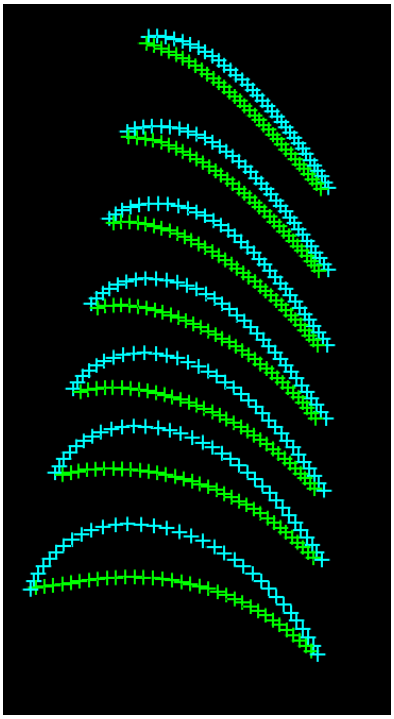

Рисунок 4.2 – Координатні точки побудовані кодом

Наступним кроком у даному моделюванні буде поєднання отриманих координатних точок за допомогою функції «Сплайн по точкам» (рис. 4.3). У результаті використання функції ми отримаємо вісь за необхідною нам формою (рис. 4.4). Повторюємо цю процедуру також і на інших рівнях, отримуючи необхідні осі для побудови лопатки (рис. 4.5).

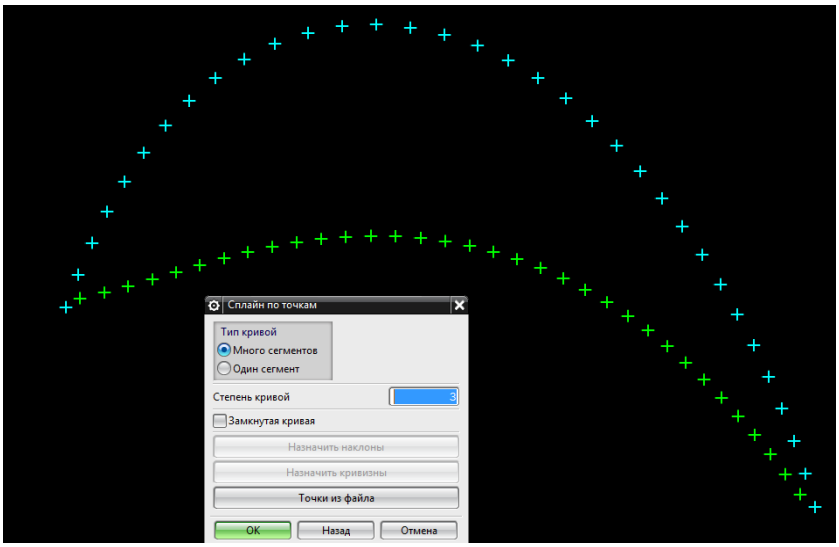

Рисунок 4.3 – Вибір точок для з’єднання

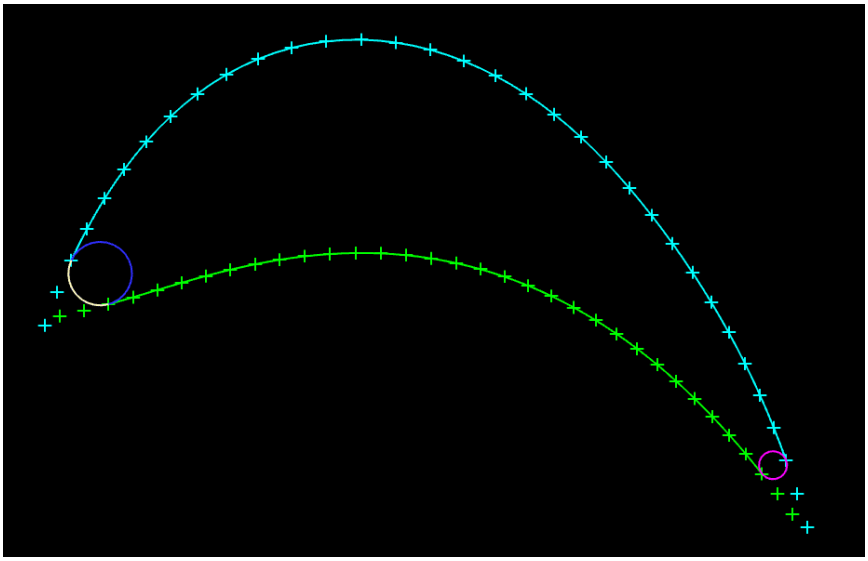

Рисунок 4.4 – З’єднані точки

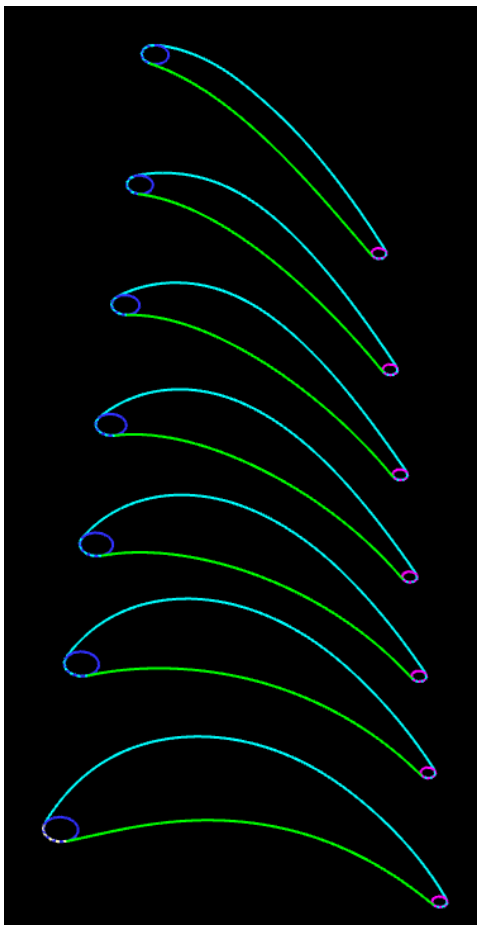

Рисунок 4.5 – Побудовані осі лопатки

Оскільки ми маємо необхідні осі, то наступним кроком буде використання функції «Зшивання» для отримання необхідних граней, після чого ці грані будуть повторно зшиті (рис. 4.6 – 4.7).

В результаті зроблених до цього операцій ми отримуємо тривимірну модель лопатки газотурбінного двигуна, з якою далі можна працювати та модифікувати (рис. 4.8).

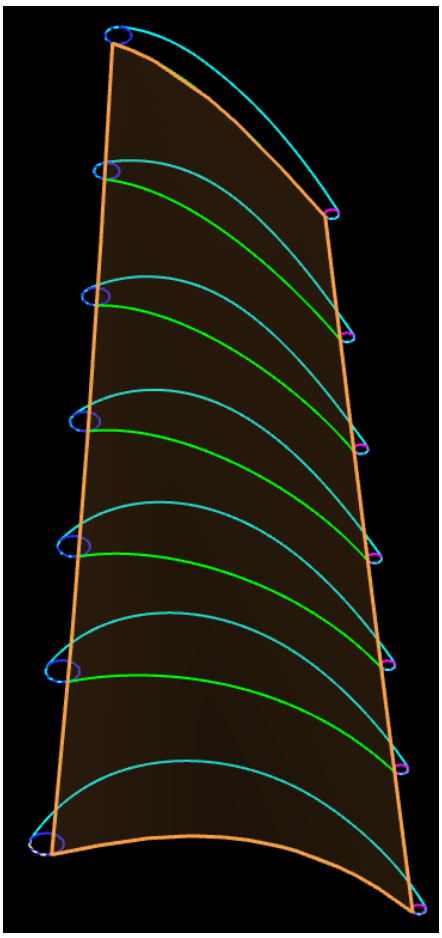

Рисунок 4.6 – Зшивання однієї грані

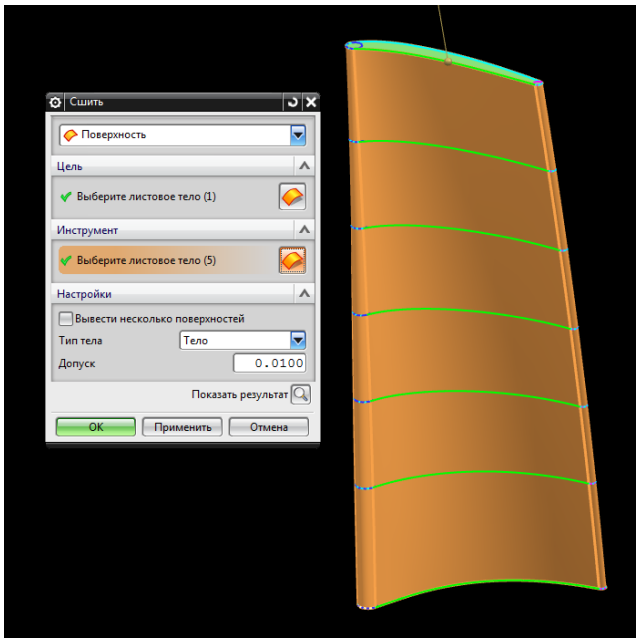

Рисунок 4.7 – Зшиті всі грані

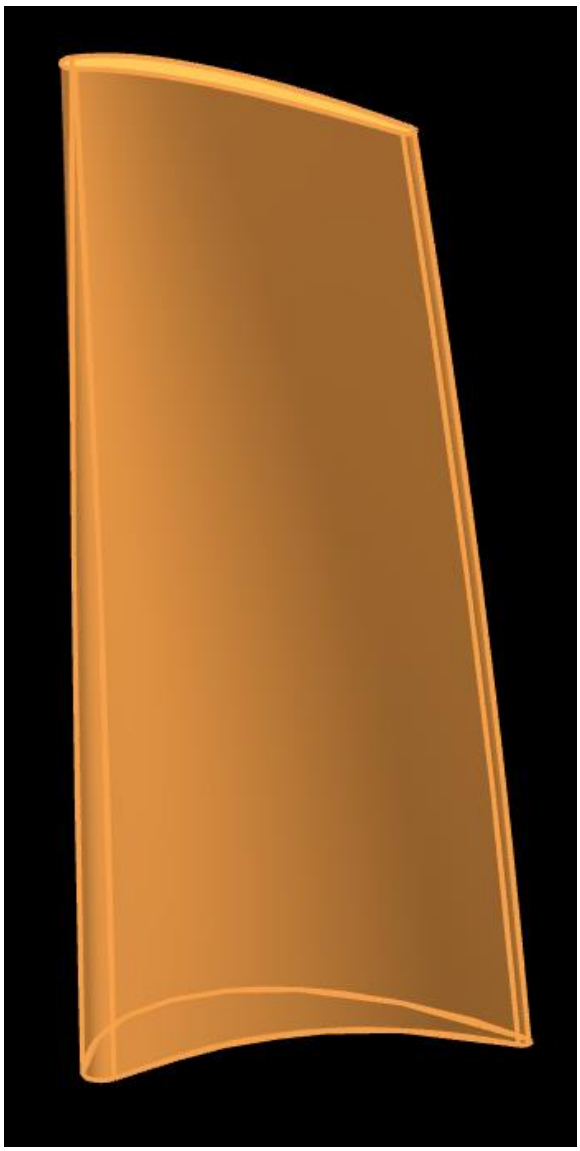

Рисунок 4.8 – Зшита цільна модель

ВИСНОВКИ

У кваліфікаційній роботі було реалізовано процес 3D моделювання лопатки газотурбінного двигуна у системі автоматизованого проектування з використанням сторонніх програмних файлів, які спростили процес та зменшили затрати часу на процес моделювання.

Система Siemens NX дає можливість для створення різноманітних моделей та механізмів, які після цього можна використовувати для модернізації чи виготовлення на виробництві.

У процесі виконання роботи були розглянуті різновиди моделювання та процес створення моделі за допомогою Siemens NX, у якій за допомогою утиліт було автоматизовано процес моделюванняю.

Тож, результатом виконання кваліфікаційної роботи є створена модель газотурбінної лопатки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке моделювання. URL : https://dimitroova.blogspot.com/p/blog-page_7.html, 2020 (дата звернення : 02.02.2024).

2. Компанія АСКОН. CAE для любопытствующих: инженерные расчеты – какие они бывают URL : https://habr.com/ru/companies/ascon/articles/792200/, (дата звернення : 10.02.2024).

3. Турбореактивный двигатель. Элементы конструкции URL : http://avia-simply.ru/tyrboreaktivnij-dvigatel-konstrukcija/, (дата звернення : 20.02.2024).

4. Письменный В. Л. Способ охлаждения турбинных лопаток газотурбинного двигателя URL : https://patents.google.com/patent/RU2409745C1/ru, (дата звернення : 05.03.2024).

5. Конструкция лопаток и технические условия на их изготовление URL : https://studfile.net/preview/2202901/page:11/, (дата звернення : 10.03.2024).

6. Изготовление турбинных лопаток URL : https://on-v.com.ua/novosti/texnologii-i-nauka/izgotovlenie-turbinnyx-lopatok/ (дата звернення : 13.04.2024).

7. Раков Д. Л., Кондратьев И. М., Печейкина М. А.. Способ финишной обработки заготовки лопатки газотурбинного двигателя и устройство для его осуществления URL : https://patents.google.com/patent/RU2674358C1/ru, (дата звернення : 25.04.2024).

8. Начало работы в NX URL : https://media.plm.automation.siemens.com/ru_ru/nx/book/chapter-1.pdf, (дата звернення : 16.05.2024).

ДОДАТОК А

Програмний код запуску програми

```
/*=================================================
файл, вызывающий диалог прорисовки пера лопатки.
=================================================*/

#include "ProfLop.h"

static void ECHO(char *format, ...)
{
   char msg[1025];
   va_list args;
   va_start(args, format);
   vsnprintf_s(msg, sizeof(msg), 1024, format, args);
   va_end(args);
   UF_UI_open_listing_window();
   UF_UI_write_listing_window(msg);
   UF_print_syslog(msg, FALSE);
}

static int report_error( char *file, int line, char *call, int irc)
{
   if (irc)
   {
      char err[133];

      UF_get_fail_message(irc, err);
      ECHO("*** ERROR code %d at line %d in %s:\n",
         irc, line, file);
      ECHO("+++ %s\n", err);
      ECHO("%s;\n", call);
   }

   return(irc);
}

extern DllExport void ufusr( char *parm, int *returnCode, int rlen )
{
   char  messg[1132] = "\0";
```

```
    int  response=-1, i;
    int  error=-1;
    FILE *stream;
    char *MyStr[MAX_PATH] = {"\0"};
    char FileDLG[MAX_PATH] = "\0";

    int errorCode = UF_initialize();
    dialog_id = -1;
// Определение начальных данных

    dlg_cancel = false;

    if ( 0 == errorCode )
    {

      response=UF_translate_variable("UGII_TURBINE_DIR", MyStr);
      strcpy(ProgDIR, MyStr[0]);
      strcpy(FileDLG,ProgDIR);
      strcat(FileDLG,"\\dialog\\UGProf.dlg");

// Проверка  на существовоние файла диалога UGProf.dlg

       if( (stream  = fopen( FileDLG, "r" )) == NULL )
       {
         sprintf((char *)messg, "Файл диалога %s не найден!", FileDLG);
         uc1601(messg, TRUE);
         return;
       }

/* Проверка на наличие открытой части */
       work_part = UF_ASSEM_ask_work_part();

       if ( work_part == NULL_TAG)
       {
          strcpy((char *)messg,"Нет активной рабочей части \n");
          strcat((char *)messg,"Откройте имеющуюся или создайте новую");
          uc1601(messg, TRUE);
          return;
       }
// Предварительные настройки
        UF_DRF_ask_preferences(mpi_array_old, mpr_array_old,
                        rad_symbol, dia_symbol);
// Сохранение старых настроек
        for(i=0;i <=99;i++)
        { mpi_array_new[i] = mpi_array_old[i];
```

```
        mpr_array_new[i] = mpr_array_old[i];}
// Создание диалога на базе файла UGProf.dlg
dlg:     dialog_id = UF_STYLER_create_dialog ( FileDLG,
         PROF_cbs,      /* Callbacks from dialog */
         PROF_CB_COUNT, /* number of callbacks*/
         NULL,        /* This is your client data */
         &response ) ;
      if (dialog_id!=0)
      {
         sprintf(messg, "Error create dialog number %d\n End programm", dialog_id);
         uc1601(messg, TRUE);
         errorCode = UF_terminate();
      }
      if (dlg_cancel == true) return;
      if( strlen(TrimSp(FileName)) == 0)
      {
          sprintf((char *)messg, "Введите имя файла!");
          uc1601(messg, TRUE);
          goto dlg ;
      }

// Чтение координат из файла
      error = ReadCoord();

// Построение профиля
      DRAW_Prof();

// Возврат старых настроек
       UF_DRF_set_preferences(mpi_array_old, mpr_array_old,
                    rad_symbol, dia_symbol);

//zoom_view(Box_point);
//      MessageBox(NULL,"Построение окончено","",MB_OK);
       UF_DISP_regenerate_display();
       errorCode = UF_terminate();
   }

   UF_CALL(UF_terminate());
}

extern int ufusr_ask_unload( void )
{
   return( UF_UNLOAD_IMMEDIATELY );
}
// Обрезка строки с двух сторон
```

```
static char *TrimSp (char *str)
{
   int i1 = 0, i2 = 0;
   char TrimStr[MAX_FSPEC_SIZE+1] = "";

   while ( str[i1] == ' ' ) i1++;
   i2 = strlen(str);
   while ( str[i2-1] == ' ') i2--;
   strncpy(TrimStr, (str+i1), i2-i1);
   return TrimStr;
}
```

ДОДАТОК Б

Код читання координат

```
#include "ProfLop.h"
// Чтение координат из файла
extern int ReadCoord(void)
{
   int i, i_sech, i_pnt0, i_pnt;
   FILE *stream;
   char  messg[132];
   char XYZ[80];
   double xx=0, yy=0, zz=0, rr=0;

   stream = fopen( FileName, "r+" );
   if( stream == NULL )
   {
      sprintf ( messg, "Could not open the file, %s\n", FileName);
      uc1601(messg,1);
      return 1;
   }
   else
   {
      i_sech = 0;
      PntSp = NULL;
      PntSp = (struct pts_struct *)malloc(sizeof(PntSp[0]));
      PntKor = NULL;
      PntKor = (struct pts_struct *)malloc(sizeof(PntKor[0]));
// Чтение из файла
      while ((fscanf( stream, "%s\n", XYZ )) != EOF)
      {
         // кол-во узлов
         sscanf (XYZ, "%d\n", &NNodes);
         i_size = (i_sech + 1) * NNodes;
         i_pnt0 = i_sech * NNodes;
// Выделение памяти
         PntSp = (struct pts_struct *)realloc((void *)PntSp,sizeof(PntSp[0])*i_size);
         PntKor = (struct pts_struct *)realloc((void
*)PntKor,sizeof(PntKor[0])*i_size);
         // спинка
         for ( i = 0; i < NNodes; i++)
```

```
        {
            i_pnt = i_pnt0 + i;
            fscanf( stream, "%lf%lf%lf\n", &xx, &yy, &zz) ;
            PntSp[i_pnt].X = xx;
            PntSp[i_pnt].Y = yy;
            PntSp[i_pnt].Z = zz;
        }
        fscanf( stream, "%d\n", &NNodes );
        if (PntSp[0].X < PntSp[NNodes – 1].X)
            DirPositiv = 1;
        else
            DirPositiv = 0;
        i_pnt0 = i_sech * NNodes;
        // корыто
        for ( i = 0; i < NNodes; i++)
        {
            i_pnt = i_pnt0 + i;
            fscanf( stream, "%lf%lf%lf\n", &xx, &yy, &zz) ;
            PntKor[i_pnt].X = xx;
            PntKor[i_pnt].Y = yy;
            PntKor[i_pnt].Z = zz;
        }
        // кромка входная
        fscanf( stream, "%lf%lf%lf%lf\n", &xx, &yy, &zz, &rr) ;
        Krom1[i_sech].X = xx;
        Krom1[i_sech].Y = yy;
        Krom1[i_sech].Z = zz;
        RR1[i_sech] = rr;
        // кромка выходная
        fscanf( stream, "%lf%lf%lf%lf\n", &xx, &yy, &zz, &rr) ;
        Krom2[i_sech].X = xx;
        Krom2[i_sech].Y = yy;
        Krom2[i_sech].Z = zz;
        RR2[i_sech] = rr;
        i_sech = i_sech + 1;
    }      // while
    NSech = i_sech;

    if( fclose( stream ) )
    {
        sprintf ( messg,"The file '%s' was not closed\n", FileName);
        uc1601(messg,1);
        return 3;
    }
// Определение расположения спинки (внизу или вверху по оси Y)
```

```
        i = NNodes/2;
        if (PntSp[i].Y <  PntKor[i].Y)
        {SpUp = 0;}
        else
        {SpUp = 1;}
    }          // else
    return 0;
}
//
// Построение профиля
//
 extern void DRAW_Prof(void)
{
    int degree = 3;
    int periodicity = 0;

   UF_CURVE_pt_slope_crvatr_t point_data[NUMBER_POINTS];
    int save_def_data = 1;

    tag_t *spin_tag, *kor_tag;
    tag_t *Rad1, *Rad2, Rad;
    int isp = 0;        // счетчик по точкам всех спинок
    int ikor = 0;   // счетчик по точкам всех корыт
    int i_pnt;      // счетчик по точкам для построения средней линии
    int i_s, im;
    int SpKor;      // Указатель ( спинка = 1, корыто = 2 )
    tag_t *MidLine_Tag; //Указатели на средние линии сечений
    tag_t arc1, arc2, LinWrk;
    double junk[3], MinDist=0.0;
    int in_sp, in_kor;      //кол-во точек на спинке и корыте после обрезки
    int i_beg_sp[MAX_SECH], i_end_sp[MAX_SECH]; // Начальные и конечные
точки после обрезки
    int i_beg_kor[MAX_SECH], i_end_kor[MAX_SECH];
    tag_t *Strings_sp, *Strings_kor;                    // Указатели на стрингера
    int n_str_sp, n_str_kor;                            // Количество стрингеров
    tag_t Surf_sp, Surf_kor, Surf_arc1, Surf_arc2;  // Указатели на поверхности
    tag_t Work_tag[6];
    tag_t Plane1, Plane2;       // Указатели на плоскости
    tag_t  Body_id;    // Для построения тела
    int SEW_GENERAL = 0,
       SHEET = 1;  // тело
    tag_t cur_pointer;
    tag_t *arc1_cut, *arc2_cut;     // Вписаные кромки
    double  Krom1_mid[3], Krom2_mid[3]; // Средние точки хорд посл. точек в
металле
```

```
    double IntersPt1[3], IntersPt2[3];          // Крайние точки лопатки
    double Pnt11[3], Pnt22[3];      // Точки для построения кромок
    UF_CURVE_line_t LinWrk_coords;
    tag_t arc_cut_work;
    tag_t *arc_p_t;
    int Im1, Im2;                       // Индексы узлов, между которыми нах. Cmax
    double Cmax;

       tag_t   target[6];
    int i, count;
    double  toler;
    uf_list_p_t  disjoint_list;

    logical is_asso = FALSE;

// Активировать все слои
    UF_LAYER_set_all_but_work (UF_LAYER_ACTIVE_LAYER);

    spin_tag = (tag_t *)malloc(NSech*sizeof(spin_tag[0]));
    kor_tag = (tag_t *)malloc(NSech*sizeof(kor_tag[0]));
    Rad1 = (tag_t *)malloc(NSech*sizeof(Rad1[0]));
    Rad2 = (tag_t *)malloc(NSech*sizeof(Rad2[0]));
    MidLine_Tag = (tag_t *)malloc(NSech*sizeof(MidLine_Tag[0]));
    arc1_cut = (tag_t *)malloc(NSech*sizeof(arc1_cut[0]));
    arc2_cut = (tag_t *)malloc(NSech*sizeof(arc2_cut[0]));

    for ( i_s=0; i_s < NSech; i_s++)
    {
       i_pnt = isp;        // индекс точки для построения средней линии
// Спинка
       SpKor = 1;
       Insert_SpinKor ( PntSp, NNodes, i_s, SpKor, &isp, start_sp[i_s], end_sp[i_s],
&cur_pointer);
       spin_tag[i_s] = cur_pointer;
// Корыто
       SpKor = 2;
       Insert_SpinKor ( PntKor, NNodes, i_s, SpKor, &ikor,  start_kor[i_s],
end_kor[i_s], &cur_pointer);
       kor_tag[i_s] = cur_pointer;
    }
    UF_VIEW_fit_view (NULL_TAG, 1.0 );
// Радиусы из файла
    for ( i_s=0; i_s < NSech; i_s++)
    {
       Insert_Rad (Krom1[i_s],  RR1[i_s],  i_s, &Rad1[i_s] );
```

```
      Insert_Rad (Krom2[i_s],  RR2[i_s],  i_s, &Rad2[i_s] );
   }

// Кромки
   for ( i_s=0; i_s < NSech; i_s++)
   {
   // Входная
      Rad = Rad1[i_s];
      arc_cut_work=NULL;
      if (Krom1[i_s].X < Krom2[i_s].X && Krom1[i_s].Y < Krom2[i_s].Y)        //
Рабочая лопатка
      {   TypeLop = 1;
         Insert_Krom (Krom1[i_s],  RR1[i_s], Rad,  i_s,  &i_pnt, 1, spin_tag[i_s],
kor_tag[i_s], &arc_cut_work,
            i_beg_sp[i_s], i_end_sp[i_s], Pnt11, Pnt22, Krom1_mid, IntersPt1,
&in_sp);
      }
      else if (Krom1[i_s].X > Krom2[i_s].X && Krom1[i_s].Y > Krom2[i_s].Y)    //
Сечение
      //Лопатка перевернута , поменять местами спинку и корыто
      {   TypeLop = 2;
         Insert_Krom (Krom1[i_s],  RR1[i_s], Rad,  i_s,  &i_pnt, 1, spin_tag[i_s],
kor_tag[i_s], &arc_cut_work,
            i_beg_sp[i_s], i_end_sp[i_s], Pnt11, Pnt22, Krom1_mid, IntersPt1,
&in_sp);
      }
      else if (Krom1[i_s].X < Krom2[i_s].X && Krom1[i_s].Y > Krom2[i_s].Y)    //
Сопловая
      {   TypeLop = 3;
         Insert_Krom (Krom1[i_s],  RR1[i_s], Rad,  i_s,  &i_pnt, 1, spin_tag[i_s],
kor_tag[i_s], &arc_cut_work,
            i_beg_sp[i_s], i_end_sp[i_s], Pnt11, Pnt22, Krom1_mid, IntersPt1,
&in_sp);
      }
      arc1_cut[i_s] = cur_pt;
      UF_OBJ_delete_object (Rad);
   // Выходная
      Rad = Rad2[i_s];
      switch (TypeLop)
      {
      case 1 :       // Рабочая лопатка
         Insert_Krom (Krom2[i_s], RR2[i_s], Rad, i_s,  &i_pnt, 2, spin_tag[i_s],
kor_tag[i_s], arc_cut_work,
            i_beg_sp[i_s], i_end_sp[i_s], Pnt11, Pnt22, Krom2_mid, IntersPt2,
&in_sp);
```

```
            break;
        case 2 :        // Сечение
            Insert_Krom (Krom2[i_s], RR2[i_s], Rad, i_s,  &i_pnt, 2, kor_tag[i_s],
spin_tag[i_s], arc_cut_work,
                i_beg_sp[i_s], i_end_sp[i_s], Pnt11, Pnt22, Krom2_mid, IntersPt2,
&in_sp);
            break;
        case 3 :        // Сопловая
            Insert_Krom (Krom2[i_s], RR2[i_s], Rad, i_s,  &i_pnt, 2, kor_tag[i_s],
spin_tag[i_s], arc_cut_work,
                i_beg_sp[i_s], i_end_sp[i_s], Pnt11, Pnt22, Krom2_mid, IntersPt2,
&in_sp);
            break;
        }
        arc2_cut[i_s] = cur_pt;
        UF_CALL(UF_OBJ_set_color (cur_pt, 75));
        UF_OBJ_delete_object (Rad);
        if ( TypeLevel == 0)
        {
            n_layer = i_s + 101;
            UF_OBJ_set_color (arc1_cut[i_s], MyColor[i_s]);
            UF_OBJ_set_color (arc2_cut[i_s], MyColor[i_s]);
        }
        else
        {
            n_layer = 101;
            UF_OBJ_set_color (arc1_cut[i_s], MyColor[4]);
            UF_OBJ_set_color (arc2_cut[i_s], MyColor[4]);
        }
        UF_OBJ_set_layer (arc1_cut[i_s], n_layer);
        UF_OBJ_set_layer (arc2_cut[i_s], n_layer);

//Конечные точки обрезаных сплайнов
// СПИНКА
        UF_CALL(UF_MODL_ask_curve_props(spin_tag[i_s], 0.0, start_sp[i_s], junk,
junk, junk, junk, junk));
        UF_CALL(UF_MODL_ask_curve_props(spin_tag[i_s], 1.0, end_sp[i_s], junk,
junk, junk, junk, junk));
        MinDist = MinDistSpl(start_sp[i_s], PntSp, 1, i_s, &in_sp, Pnt11);
        i_beg_sp[i_s] = in_sp + 1;      // первая узловая точка для стрингеров
        MinDist = MinDistSpl(end_sp[i_s], PntSp, 2, i_s, &in_sp, Pnt22);
        i_end_sp[i_s] = in_sp – 1;      // последняя точка
// Конечные точки обрезаных сплайнов
// КОРЫТО
```

```
      UF_CALL(UF_MODL_ask_curve_props(kor_tag[i_s], 0.0, start_kor[i_s], junk,
junk, junk, junk, junk));
      UF_CALL(UF_MODL_ask_curve_props(kor_tag[i_s], 1.0, end_kor[i_s], junk,
junk, junk, junk, junk));
      MinDist = MinDistSpl(start_kor[i_s], PntKor, 1, i_s, &in_kor, Pnt11);
      i_beg_kor[i_s] = in_kor + 1;        // первая узловая точка для стрингеров
      MinDist = MinDistSpl(end_kor[i_s], PntKor, 2, i_s, &in_kor, Pnt22);
      i_end_kor[i_s] = in_kor – 1;        // последняя точка в металле
// Средняя линия сечения
      if ( TypeDraw == 1 || TypeDraw == 2 )       // ср.поверхность или тело +
ср.поверхность
      {
         if ( TypeLevel == 0)    n_layer = NSech + 101;
         else        n_layer = 103;
         im = i_end_sp[i_s] – i_beg_sp[i_s] + 1;     // кол-во точек на средней линии
         Insert_Mid_Line ( i_s, &i_pnt,i_beg_sp[i_s], im, Krom1_mid, Krom2_mid,
&cur_pointer);
         MidLine_Tag[i_s] = cur_pointer;
         UF_OBJ_set_color (cur_pointer, 29);
         UF_OBJ_set_layer (cur_pointer, n_layer);
         // Линия от входной кромки до средины хорды
         XYZ_init (LinWrk_coords.start_point, Krom1_mid);
         XYZ_init (LinWrk_coords.end_point, IntersPt1);
         UF_CALL(UF_CURVE_create_line (&LinWrk_coords, &LinWrk));
         UF_OBJ_set_color (LinWrk, 29);
         UF_OBJ_set_layer (LinWrk, n_layer);
         // Линия от выходной кромки до средины хорды
         XYZ_init (LinWrk_coords.start_point, Krom2_mid);
         XYZ_init (LinWrk_coords.end_point, IntersPt2);
         UF_CALL(UF_CURVE_create_line (&LinWrk_coords, &LinWrk));
         UF_OBJ_set_color (LinWrk, 29);
         UF_OBJ_set_layer (LinWrk, n_layer);
      }
   }       // следующее сечение
//Построение средней поверхности
   if ( TypeDraw == 1 || TypeDraw == 2 )       // ср.поверхность или тело +
ср.поверхность
   {
      Insert_SurfSpl ( MidLine_Tag, NSech, &Surf_sp);
      if ( TypeLevel == 0)    n_layer = NSech + 101;
      else        n_layer = 103;
      UF_OBJ_set_color (Surf_sp, 69);
      UF_OBJ_set_layer (Surf_sp, n_layer);
   }
// Построение стрингеров для поверхности
```

```
// СПИНКА
   if ( TypeDraw == 0 || TypeDraw == 2 )       // тело или тело + ср.поверхность
   {
      if ( TypeLevel == 0)    n_layer = NSech + 102;
      else       n_layer = 104;
   //  if ( TypeSurf == 1 )       // по сечениям и стрингерам
      //{
         Strings_sp = (tag_t *)malloc(NNodes*sizeof(Strings_sp[0]));
         Insert_Strings ( PntSp, i_beg_sp, i_end_sp, (double *)start_sp, (double
*)end_sp, 1, Strings_sp, &n_str_sp);
      //}
// Построение поверхности
      if ( TypeSurf == 0 )       // по сечениям
         Insert_SurfSpl ( spin_tag, NSech, &Surf_sp);
      else                       // по сечениям и стрингерам
         Insert_SurfMesh ( spin_tag, NSech, Strings_sp, n_str_sp, &Surf_sp);
      UF_CALL(UF_OBJ_set_color (Surf_sp, 108));
// КОРЫТО

      //{
         Strings_kor = (tag_t *)malloc(NNodes*sizeof(Strings_kor[0]));
         Insert_Strings ( PntKor,i_beg_kor, i_end_kor, (double *)start_kor, (double
*)end_kor, 2, Strings_kor, &n_str_kor);
      //}
// Построение поверхности
      if ( TypeSurf == 0 )       // по сечениям
         Insert_SurfSpl ( kor_tag, NSech, &Surf_kor);
      else                       // по сечениям и стрингерам
         Insert_SurfMesh ( kor_tag, NSech, Strings_kor, n_str_kor, &Surf_kor);
      UF_CALL(UF_OBJ_set_color (Surf_kor, 120));
      if ( TypeLevel == 0)    n_layer = NSech + 103;
      else       n_layer = 105;
      UF_OBJ_set_layer (Surf_sp, n_layer);
      UF_OBJ_set_layer (Surf_kor, n_layer);
// Построение поверхностей по кромкам
      if (Krom1[i_s-1].X < Krom2[i_s-1].X)
      {
         Work_tag[0] = Strings_sp[0];
         Work_tag[1] = Strings_kor[0];
      }
      //Лопатка перевернута, поменять местами спинку и корыто
      else
      {
         Work_tag[0] = Strings_kor[0];
         Work_tag[1] = Strings_sp[0];
```

```
        }
        Insert_SurfMesh ( arc1_cut, NSech, Work_tag, 2,  &Surf_arc1);
        //Если не построилось по сетке – строим по сплайнам
        if ( Surf_arc1 == 0)   Insert_SurfSpl ( arc1_cut, NSech, &Surf_arc1);
        UF_CALL(UF_OBJ_set_color (Surf_arc1, 75));
        if (Krom1[i_s-1].X < Krom2[i_s-1].X)
        {
           Work_tag[0] = Strings_sp[n_str_sp-1];
           Work_tag[1] = Strings_kor[n_str_kor-1];
        }
        //Лопатка перевернута, поменять местами спинку и корыто
        else
        {
           Work_tag[0] = Strings_kor[n_str_kor-1];
           Work_tag[1] = Strings_sp[n_str_sp-1];
        }
        Insert_SurfMesh ( arc2_cut, NSech, Work_tag, 2,  &Surf_arc2);
        //Если не построилось по сетке – строим по сплайнам
        if ( Surf_arc2 == 0)   Insert_SurfSpl ( arc2_cut, NSech, &Surf_arc2);
        UF_CALL(UF_OBJ_set_color (Surf_arc2, 75));
        UF_OBJ_set_layer (Surf_arc1, n_layer);
        UF_OBJ_set_layer (Surf_arc2, n_layer);

// Построение ограничивающих плоскостей по крайним сечениям
        Work_tag[0] = spin_tag[0];
        Work_tag[1] = arc1_cut[0];
        Work_tag[2] = kor_tag[0];
        Work_tag[3] = arc2_cut[0];
        Insert_Bplane ( Work_tag, 4, &Plane1);
        UF_CALL(UF_OBJ_set_color (Plane1, 114));
        Work_tag[0] = spin_tag[NSech-1];
        Work_tag[1] = kor_tag[NSech-1];
        Work_tag[2] = arc1_cut[NSech-1];
        Work_tag[3] = arc2_cut[NSech-1];
        Insert_Bplane ( Work_tag, 4, &Plane2);
        UF_CALL(UF_OBJ_set_color (Plane2, 114));
        UF_OBJ_set_layer (Plane1, n_layer);
        UF_OBJ_set_layer (Plane2, n_layer);

// Сшить поверхности в тело
        Work_tag[0] = Plane1;
        Work_tag[1] = Plane2;
        Work_tag[2] = Surf_arc1;
        Work_tag[3] = Surf_arc2;
        Work_tag[4] = Surf_sp;
```

```
        Work_tag[5] = Surf_kor;

    int objType, subType, Body_Type;
    char  messg[1132] = "\0";
    UF_OBJ_ask_type_and_subtype(Body_id, &objType, &subType);
    sprintf((char *)messg, "objType = %d, subType = %d ", objType, subType);
    uc1601(messg, TRUE);

    if (subType == UF_solid_body_subtype)
    {
       // Тип тела запроса
       UF_MODL_ask_body_type(Body_id, &Body_Type);
       if (Body_Type == UF_MODL_SOLID_BODY) // сущность
       {
          UF_OBJ_set_color(Body_id, 186);
       }
    }

    UF_CALL(UF_OBJ_set_color (Body_id, 120));
       if ( TypeLevel == 0)    n_layer = NSech + 104;
       else        n_layer = 106;
       UF_OBJ_set_layer (Body_id, n_layer);
    }       // if
    free(PntSp);
    free(PntKor);
}

//  Построение  тела методом сшивки поверхностей

int make_sew(tag_t *sheet_list, int n_sheet, int body_type, tag_t *sew_tag)
{
 int option=0;
 double tolerance=0.01;
 //int type_body=0;
 uf_list_p_t disjoint_list;

 int num_parents=0;
 tag_t *parent_array=NULL;
 int num_children=0;
 tag_t *children_array=NULL;
 tag_t set_tag=NULL_TAG;
 tag_t *feat_array=NULL;
 int i=0;
```

```
 if(UF_CALL(UF_MODL_create_sew ( option,  1, &sheet_list[0], n_sheet-1,
&sheet_list[1],
  tolerance,  body_type, &disjoint_list, sew_tag ))) return -1;

 UF_CALL(UF_MODL_delete_list (&disjoint_list ));

 if(UF_CALL(UF_MODL_ask_feat_relatives ( *sew_tag, &num_parents,
&parent_array,
  &num_children, &children_array ))) return 1;

 UF_free(children_array);

 if(UF_CALL(allocate_memory(sizeof(tag_t)*(num_parents+1),
(void**)&feat_array)))
 {
  UF_free(parent_array);
  return 1;
 }

 for (i=0; i<num_parents; i++)
 {
  feat_array[i]=parent_array[i];
 }

 feat_array[num_parents]=*sew_tag;

 UF_free(parent_array);

 if(UF_CALL(UF_MODL_create_set_of_feature ( "IMPORT_FEATURE",
feat_array,
  num_parents+1, 1, &set_tag )))
 {
  UF_free(feat_array);
  return 1;
 }

 UF_free(feat_array);

 return 0;
}

int allocate_memory(unsigned int nbytes, void **where)
{
    int  resp;
```

```
    *where = UF_allocate_memory(nbytes, &resp);

    return resp;
}

// Построение спинки или корыта
void Insert_SpinKor ( struct pts_struct *Pnt_Dat, int NNodes, int i_s, int Spin, int
*ip,
                      double *start_pnt, double *end_pnt, tag_t *tag_cur)
{
    int i, i_p;
    int degree = 3;
    int periodicity = 0;
    UF_CURVE_pt_slope_crvatr_t point_data[NUMBER_POINTS];
    double parameters[NUMBER_POINTS];
    int save_def_data = 1;
    tag_t point_id;

    parameters[0] = 0.0;
    i_p = *ip;
    for ( i=0; i < NNodes; i++)
    {
        point_data[i].point[0] = Pnt_Dat[i_p].X ;
        point_data[i].point[1] = Pnt_Dat[i_p].Y;
        point_data[i].point[2] = Pnt_Dat[i_p].Z;
        if ( i == 0 )
        {
            start_pnt[0] = Pnt_Dat[i_p].X ;
            start_pnt[1] = Pnt_Dat[i_p].Y ;
            start_pnt[2] = Pnt_Dat[i_p].Z  ;
        }
        UF_CURVE_create_point (point_data[i].point, &point_id );
        if ( TypeLevel == 0)
        {
            n_layer = i_s + 101;
            UF_OBJ_set_color (point_id, MyColor[i_s]);
        }
        else
        {
            if ( Spin == 1)
            {
                UF_CALL(UF_OBJ_set_color (point_id, MyColor[4]));
                n_layer = 101;
            }
            else
```

```
            {
               UF_CALL(UF_OBJ_set_color (point_id, MyColor[5]));
               n_layer = 102;
            }
         }
         UF_OBJ_set_layer (point_id, n_layer);
// Вычисление параментра = длине хорды
         if ( i > 0 )
            parameters[i] = parameters[i – 1] + sqrt(Pnt_Dat[i_p].X * Pnt_Dat[i_p].X +
                                 Pnt_Dat[i_p].Y * Pnt_Dat[i_p].Y + Pnt_Dat[i_p].Z *
Pnt_Dat[i_p].Z);
         point_data[i].slope_type = UF_CURVE_SLOPE_AUTO;
         point_data[i].slope[0] = 0.0;
         point_data[i].slope[1] = 0.0;
         point_data[i].slope[2] = 0.0;
         point_data[i].crvatr_type = UF_CURVE_CRVATR_NONE;
         point_data[i].crvatr[0] = 0.0;
         point_data[i].crvatr[1] = 0.0;
         point_data[i].crvatr[2] = 0.0;
         i_p = i_p + 1;    // следующая точка сплайна*/
      }
      end_pnt[0] = Pnt_Dat[i_p-1].X ;
      end_pnt[1] = Pnt_Dat[i_p-1].Y ;
      end_pnt[2] = Pnt_Dat[i_p-1].Z  ;
// Create B-spline UFCurve
      UF_CALL(UF_CURVE_create_spline_thru_pts(degree,
                                 periodicity,
                                 NNodes,
                                 point_data,
                                 parameters,
                                 save_def_data, tag_cur));

      if ( TypeLevel == 0)
      {
         UF_CALL(UF_OBJ_set_layer (*tag_cur, i_s + 101));
         UF_CALL(UF_OBJ_set_color (*tag_cur, MyColor[i_s]));
      }
      else
      {
         if ( Spin == 1)
         {
            UF_CALL(UF_OBJ_set_color (*tag_cur, MyColor[4]));
            n_layer = 101;
         }
         else
```

```
        {
          UF_CALL(UF_OBJ_set_color (*tag_cur, MyColor[5]));
          n_layer = 102;
        }
        UF_CALL(UF_OBJ_set_layer (*tag_cur, n_layer));
    }
    *ip = i_p;
}

//Построение радиусов из файла
void Insert_Rad (struct pts_struct Krom, double RR,  int i_sech,  tag_t *Rad )
{
    tag_t wcs_id, arc_id;
    UF_CURVE_arc_t arc_coords;

  /* Fill out the data structure */
  arc_coords.start_angle = 0.0;
  arc_coords.end_angle = 360.0 * DEGRA;
  arc_coords.arc_center[0] = Krom.X;
  arc_coords.arc_center[1] = Krom.Y;
  arc_coords.arc_center[2] = Krom.Z;
  arc_coords.radius = RR;

  UF_CALL(UF_CSYS_ask_wcs(&wcs_id));
  UF_CALL(UF_CSYS_ask_matrix_of_object(wcs_id,
                                &arc_coords.matrix_tag));
  UF_CALL(UF_CURVE_create_arc(&arc_coords,&arc_id));
    if ( TypeLevel == 0)
    {
        UF_CALL(UF_OBJ_set_layer (arc_id, i_sech + 101));
        UF_CALL(UF_OBJ_set_color (arc_id, MyColor[i_sech]));
    }
    else
    {
        UF_CALL(UF_OBJ_set_layer (arc_id, 102));
        UF_CALL(UF_OBJ_set_color (arc_id, MyColor[5]));
    }
    *Rad = arc_id;
}
// Построение средней линии сечения по средним точкам хорд
void Insert_Mid_Line ( int i_s, int *ip, int i_beg, int Npt,  double *Krom1_mid,
double *Krom2_mid,
                       tag_t *MidLine_Tag)
{
    UF_CURVE_line_t LinWrk_coords;
```

```
tag_t LinWrk, point_id;
tag_t arc_cutl;
int i;
int i_p, i_n;
logical is_asso = FALSE;
UF_CURVE_arc_t arc_coords;
double pt1[3], pt2[3], mid_pnt[NUMBER_POINTS][3];
int degree =3;
int periodicity = 0;
UF_CURVE_pt_slope_crvatr_t point_data[NUMBER_POINTS];
double parameters[NUMBER_POINTS];
int save_def_data = 1;

parameters[0] = 0.0;
point_data[0].point[0]  =   Krom1_mid[0];
point_data[0].point[1]  =   Krom1_mid[1];
point_data[0].point[2]  =   Krom1_mid[2];
point_data[0].slope_type    = UF_CURVE_SLOPE_AUTO;
point_data[0].slope[0]  =   0.0;
point_data[0].slope[1]  =   0.0;
point_data[0].slope[2]  =   0.0;
point_data[0].crvatr_type   = UF_CURVE_CRVATR_NONE;
point_data[0].crvatr[0] =   0.0;
point_data[0].crvatr[1] =   0.0;
point_data[0].crvatr[2] =   0.0;

i_p = i_beg;
i_n =1;       // индекс точек на средней линии
for ( i=0; i < Npt; i++)
{
   pt1[0] = PntSp[i_p].X ;
   pt1[1] = PntSp[i_p].Y ;
   pt1[2] = PntSp[i_p].Z ;
   pt2[0] = PntKor[i_p].X ;
   pt2[1] = PntKor[i_p].Y ;
   pt2[2] = PntKor[i_p].Z ;
   UF_VEC3_midpt (pt1, pt2, mid_pnt[i_n]);
   if ( point_data[i_n-1].point[0] == mid_pnt[i_n][0] )
   {
      i_p =  i_p +1;    // следующая точка
      continue;
   }
   point_data[i_n].point[0] = mid_pnt[i_n][0];
   point_data[i_n].point[1] = mid_pnt[i_n][1];
   point_data[i_n].point[2] = mid_pnt[i_n][2];
```

```
// Вычисление параментра = длине хорды
      parameters[i_n] = parameters[i_n – 1] + sqrt(point_data[i_n].point[0] *
point_data[i_n].point[0] +
         point_data[i_n].point[1] * point_data[i_n].point[1] + point_data[i_n]. point[2]
* point_data[i_n].point[2]);
      point_data[i_n].slope_type = UF_CURVE_SLOPE_AUTO;
      point_data[i_n].slope[0] = 0.0;
      point_data[i_n].slope[1] = 0.0;
      point_data[i_n].slope[2] = 0.0;
      point_data[i_n].crvatr_type = UF_CURVE_CRVATR_NONE;
      point_data[i_n].crvatr[0] = 0.0;
      point_data[i_n].crvatr[1] = 0.0;
      point_data[i_n].crvatr[2] = 0.0;
      i_n = i_n + 1;
      i_p =  i_p +1;    // следующая точка
   }
   point_data[i_n].point[0] = Krom2_mid[0];
   point_data[i_n].point[1] = Krom2_mid[1];
   point_data[i_n].point[2] = Krom2_mid[2];
   parameters[i_n] = parameters[i_n – 1] + sqrt(point_data[i_n].point[0] * point_data
[i_n].point[0] +
      point_data[i_n].point[1] * point_data[i_n].point[1] + point_data[i_n]. point[2] *
point_data[i_n].point[2]);
   point_data[i_n].slope_type = UF_CURVE_SLOPE_AUTO;
   point_data[i_n].slope[0] = 0.0;
   point_data[i_n].slope[1] = 0.0;
   point_data[i_n].slope[2] = 0.0;
   point_data[i_n].crvatr_type = UF_CURVE_CRVATR_NONE;
   point_data[i_n].crvatr[0] = 0.0;
   point_data[i_n].crvatr[1] = 0.0;
   point_data[i_n].crvatr[2] = 0.0;
   // Create B-spline UFCurve
   UF_CALL(UF_CURVE_create_spline_thru_pts(degree,
                                           periodicity,
                                           i_n+1,
                                           point_data,
                                           parameters,
                                           save_def_data, MidLine_Tag));
}

// Построение кромок
void Insert_Krom (  struct pts_struct Krom, double RR, tag_t arc, int i_sech,  int *ip,
int Begin, tag_t sp_tag,
                    tag_t kr_tag, tag_t *arc_cut, int i_beg, int i_end, double Pnt1[3],
double Pnt2[3],
```

```
                    double *Krom_mid, double *IntersPtDat, int *in)
{
   double TmpPnt[] = {0,0,0};
   double MinDist=0.0, Pnt[3], resp;
   int im;
   UF_CURVE_line_t LinWrk_coords;
   tag_t LinWrk, LinWrk1, LinWrk2, LinWrk3;
   double VecPr[3], VecUnit[3], VecRad[3], tol, tol_work, mag;
   logical is_asso = FALSE;
   double tmpPoints[3];
   double LenLine;     // Длина касательной
   double VecTang[3], VecTang2[3];     // Векторы касательной
   double start[3], end[3], parm, arc_pnt[3], junk[3];
   tag_t start_pt,  end_pt;
   tag_t Tag_pt1, Tag_pt2, IntersPtFeat, IntersPtTag;
   tag_t *cur_pointer1;

   tag_t WrkArc, PtCentr;
      double MidPoint[ 3 ], tangent[ 3 ], p_norm[ 3 ], b_norm[ 3 ], torsion,
rad_of_cur;
   int i, i_int = 0, n_int = 0;

   char  grip_exe[MAX_PATH];
   int   status, grip_arg_count = 5;
   UF_args_t    grip_arg_list[5];
// for NX2
   double Pos[10][3];
   UF_MODL_intersect_info_p_t *intersections;
   char  messg[132];

   strcpy(grip_exe, ProgDIR);
   strcat(grip_exe,"\\Fillet_Trim.grx");
// Построение вписаной кромки
   UF_CALL(UF_MODL_ask_minimum_dist (sp_tag, arc, 0, TmpPnt, 0, TmpPnt,
&MinDist, Pnt1, Pnt));
   MinDist = MinDistSpl(Pnt1, PntSp, Begin, i_sech, &im, Pnt1);
   if ( Begin == 1)       // входная кромка
   {
      i_beg = im – 1;     // первая точка в металле
   }
   else                    // выходная кромка
   {
      i_end = im + 1;     // последняя точка в металле
      *in = im;           // кол-во точек после обрезки
   }
```

```
// Вторая Help-точка на вписываемой дуге – на спинке
   UF_CALL(UF_MODL_ask_minimum_dist (kr_tag, arc, 0, TmpPnt, 0, TmpPnt,
&MinDist, Pnt2, Pnt));
   MinDist = MinDistSpl(Pnt2, PntKor, Begin, i_sech, &im, Pnt2);
// Первая Help-точка на вписываемой дуге – на корыте
   XYZ_init (LinWrk_coords.start_point, Pnt1);
   XYZ_init (LinWrk_coords.end_point, Pnt2);
   UF_VEC3_midpt (Pnt1, Pnt2, Krom_mid );          // Средняя точка хорды
   UF_CALL(UF_CURVE_create_line (&LinWrk_coords, &LinWrk));        // Хорда
– последние точки в металле
   UF_CALL(UF_OBJ_set_color(LinWrk, 186));
   tmpPoints[0]=Krom.X;
   tmpPoints[1]=Krom.Y;
   tmpPoints[2]=Krom.Z;
// Центр кромки
   UF_CURVE_create_point (tmpPoints, &PtCentr );
   UF_CALL(UF_MODL_ask_curve_parm (LinWrk, tmpPoints, &parm, arc_pnt));
   XYZ_init (LinWrk_coords.start_point, tmpPoints);
   XYZ_init (LinWrk_coords.end_point, arc_pnt);
// НОВАЯ ВЕРСИЯ
   UF_OBJ_delete_object (LinWrk);
   UF_CALL(UF_CURVE_create_line (&LinWrk_coords, &LinWrk));        //
Перпендикуляр к хорде
   UF_CALL(UF_OBJ_set_color(LinWrk, 181));
   UF_VEC3_sub (LinWrk_coords.end_point, LinWrk_coords.start_point, VecPr);
   UF_MODL_ask_distance_tolerance (&tol);
   tol_work = tol/100;
   UF_VEC3_unitize (VecPr, tol_work, &mag, VecUnit);
   UF_VEC3_affine_comb (VecPr, RR, VecUnit, VecRad);
   tmpPoints[0] = arc_pnt[0] + VecRad[0];
   tmpPoints[1] = arc_pnt[1] + VecRad[1];
   tmpPoints[2] = arc_pnt[2] + VecRad[2];
   XYZ_init (LinWrk_coords.start_point, arc_pnt);
   XYZ_init (LinWrk_coords.end_point, tmpPoints);
   UF_OBJ_delete_object (LinWrk);
   UF_CALL(UF_CURVE_create_line(&LinWrk_coords, &LinWrk));     // Прямая
– радиус
   UF_CALL(UF_CURVE_create_point(Pnt, &Tag_pt1));
   UF_CALL(UF_CURVE_create_point(tmpPoints, &Tag_pt2));
// for NX2
   UF_MODL_ask_distance_tolerance(&tol);
   UF_CALL(UF_MODL_intersect_objects(arc,LinWrk, tol, &n_int,
&intersections));
   for (i = 0; i < n_int; i++)
   {
```

```
      Pos[i_int][0] = intersections[i]->intersect.point.coords[0];
      Pos[i_int][1] = intersections[i]->intersect.point.coords[1];
      Pos[i_int][2] = intersections[i]->intersect.point.coords[2];
      i_int++;
   }
   XYZ_init (IntersPtDat, Pos[0]);
   UF_CALL(UF_CURVE_create_point(IntersPtDat, &Tag_pt1));

//**********************************
   UF_VEC3_sub(arc_pnt, IntersPtDat, VecRad);
  UF_VEC3_ask_perpendicular(VecRad, VecPr);
   Pnt1[0] = IntersPtDat[0] + VecPr[0];
   Pnt1[1] = IntersPtDat[1] + VecPr[1];
   Pnt1[2] = IntersPtDat[2] + VecPr[2];
   LenLine = sqrt((IntersPtDat[0] – Pnt1[0]) * (IntersPtDat[0] – Pnt1[0]) +
                  (IntersPtDat[1] – Pnt1[1]) * (IntersPtDat[1] – Pnt1[1]) +
                  (IntersPtDat[2] – Pnt1[2]) * (IntersPtDat[2] – Pnt1[2]));
   LenLine = LenLine * 2;
   UF_VEC3_sub(IntersPtDat, Pnt1, VecTang);
   UF_CALL(UF_VEC3_unitize(VecPr, tol, &mag, VecUnit));
   UF_VEC3_affine_comb(VecTang, 2.0, VecTang, VecTang2);
   Pnt2[0] = Pnt1[0] + VecTang2[0];
   Pnt2[1] = Pnt1[1] + VecTang2[1];
   Pnt2[2] = Pnt1[2] + VecTang2[2];
   XYZ_init(LinWrk_coords.start_point, Pnt1);
   XYZ_init(LinWrk_coords.end_point, Pnt2);
   UF_OBJ_delete_object (LinWrk);
   UF_CALL(UF_CURVE_create_line(&LinWrk_coords, &LinWrk));     //
Касательная
// Вписать дугу в 3 объекта (спинка, корыто, касательная) – GRIP (FILLET\)

/* Define the argument list for UG/Open API calling GRIP */
   grip_arg_list[0].type    = UF_TYPE_TAG_T;
   grip_arg_list[0].length  = 0;
   if (Begin == 1 && TypeLop == 1) {          // входная кромка, спинка ниже
корыта по Y (рабочая)
      grip_arg_list[0].address = &kr_tag;
      grip_arg_list[2].address = &sp_tag;}
   else if (Begin == 2 && TypeLop == 1) {      // выходная кромка, спинка ниже
корыта по Y (рабочая)
      grip_arg_list[0].address = &sp_tag;
      grip_arg_list[2].address = &kr_tag;}
   else if (Begin == 1 && TypeLop == 2) {      // входная кромка, спинка выше
корыта по Y (сечение)
      grip_arg_list[0].address = &kr_tag;
```

```
        grip_arg_list[2].address = &sp_tag;}
    else if (Begin == 2 && TypeLop == 2) {      // выходная кромка, спинка выше
корыта по Y (сечение)
        grip_arg_list[0].address = &kr_tag;
        grip_arg_list[2].address = &sp_tag;}
    else if (Begin == 1 && TypeLop == 3) {      // входная кромка, спинка выше
корыта по Y (сопловая)
        grip_arg_list[0].address = &sp_tag;
        grip_arg_list[2].address = &kr_tag;}
    else if (Begin == 2 && TypeLop == 3) {      // выходная кромка, спинка выше
корыта по Y (сопловая)
        grip_arg_list[0].address = &sp_tag;
        grip_arg_list[2].address = &kr_tag;}
    grip_arg_list[1].type    = UF_TYPE_TAG_T;
    grip_arg_list[1].length  = 0;
    grip_arg_list[1].address = &LinWrk;
    grip_arg_list[2].type    = UF_TYPE_TAG_T;
    grip_arg_list[2].length  = 0;
    grip_arg_list[3].type    = UF_TYPE_TAG_T;
    grip_arg_list[3].length  = 0;
    grip_arg_list[3].address = &PtCentr;
    grip_arg_list[4].type    = UF_TYPE_TAG_T;
    grip_arg_list[4].length  = 0;
    grip_arg_list[4].address = &cur_pt;    //arc_cut; – ошибка

  /* Execute the GRIP program */
   status = UF_CALL(UF_call_grip (grip_exe, grip_arg_count,
                        grip_arg_list));
}
// Построение стрингеров
void Insert_Strings ( struct pts_struct *Pnt_Dat, int *ind_begin, int *ind_end,
                  double *start_pnt, double *end_pnt, int ind, tag_t *Strings_tag,   int
*n_Strings)
{
    int i, j, i_p, i_p0;

    int degree = 3;
    int periodicity = 0;
    UF_CURVE_pt_slope_crvatr_t point_data[NUMBER_POINTS];
    double parameters[NUMBER_POINTS];
    int save_def_data = 1;

    *n_Strings = ind_end[0] – ind_begin[0] + 1 + 2; // (+ 2) – стрингера по
конечным точкам неузловым
    i_p0 = ind_begin[0] – 1;
```

```
   for ( i = 0; i < *n_Strings; i++)
   {
      parameters[0] = 0.0;
      i_p = i_p0 + i;
      for ( j = 0; j < NSech; j++)
      {
         if ( i == 0 ) {
            point_data[j].point[0] = *(start_pnt + 3*j);
            point_data[j].point[1]  = *(start_pnt + 3*j + 1);
            point_data[j].point[2]  = *(start_pnt + 3*j + 2);
         }
         else if ( i == *n_Strings – 1) {
            point_data[j].point[0] = *(end_pnt + 3*j);
            point_data[j].point[1]  = *(end_pnt + 3*j + 1);
            point_data[j].point[2]  = *(end_pnt + 3*j + 2);
         }
         else {
            point_data[j].point[0] = Pnt_Dat[i_p].X;
            point_data[j].point[1]  = Pnt_Dat[i_p].Y;
            point_data[j].point[2]  = Pnt_Dat[i_p].Z;
         }
  // Вычисление параментра = длине хорды
         if ( j > 0 )
            parameters[j] = parameters[j – 1] + sqrt(point_data[j].point[0] *
point_data[j].point[0] +
                                                    point_data[j].point[1] *
point_data[j].point[1] +
                                                    point_data[j].point[2] *
point_data[j].point[2]);
         point_data[j].slope_type = UF_CURVE_SLOPE_AUTO;
         point_data[j].slope[0] = 0.0;
         point_data[j].slope[1] = 0.0;
         point_data[j].slope[2] = 0.0;
         point_data[j].crvatr_type = UF_CURVE_CRVATR_NONE;
         point_data[j].crvatr[0] = 0.0;
         point_data[j].crvatr[1] = 0.0;
         point_data[j].crvatr[2] = 0.0;
         i_p = i_p + NNodes;    // следующая точка */
      }
      if (TypeSurf == 1 || i == 0 || i ==  *n_Strings-1){     // по стрингерам или по
крайним точкам сечений
// Create B-spline UFCurve
         UF_CALL(UF_CURVE_create_spline_thru_pts(degree,
                                    periodicity,
                                    NSech,
```

```
                                point_data,
                                parameters,
                                save_def_data, &Strings_tag[i]));
        if ( TypeLevel == 0)
        {
           UF_CALL(UF_OBJ_set_layer (Strings_tag[i], NSech + 102));
           UF_CALL(UF_OBJ_set_color (Strings_tag[i], MyColor[NSech]));
        }
        else
        {
           UF_CALL(UF_OBJ_set_layer (Strings_tag[i], 104));
           UF_CALL(UF_OBJ_set_color (Strings_tag[i], MyColor[5]));
        }
      }
   }
}
// Построение поверхностей по сетке кривых
void Insert_SurfMesh ( tag_t  *Primary_tag, int n_Prim, tag_t *Cross_tag, int
n_Cross, tag_t *Surf_tag)
{
UF_STRING_t
      primary,
      cross,
      spine;
   int
      i,
      c_flag[4],
      end_point = 0,
      emphasis = 3,
      body_type = 0,
      spln_pt = 0;
   UF_FEATURE_SIGN
      boolean = UF_NULLSIGN;
   double
      tol[3];
   tag_t  c_face[4];

   UF_MODL_ask_distance_tolerance(&tol[0]);
   UF_MODL_ask_angle_tolerance(&tol[1]);
   tol[2] = 0.02;

   UF_MODL_init_string_list(&primary);
   UF_MODL_init_string_list(&cross);
   UF_MODL_init_string_list(&spine);
```

```
   UF_MODL_create_string_list(n_Prim,n_Prim,&primary); /* n_Prim strings with 1
curve each */
   UF_MODL_create_string_list(n_Cross,n_Cross,&cross);

   primary.num = n_Prim; /* number of strings for primary curve */
   for ( i = 0; i < n_Prim; i++)
   {
      primary.dir[i] = 1;
      primary.string[i] = 1;  /* number of entities in the prim string */
      primary.id[i] = Primary_tag[i];
   }

   cross.num = n_Cross; /* number of cross strings */
   for ( i = 0; i < n_Cross; i++)
   {
      cross.dir[i] = 1;
      cross.string[i] = 1;    /* number of entities in the cross string */
      cross.id[i] = Cross_tag[i];
   }

   for (i=0; i<4; i++)
   {
      c_face[i] = 0;
      c_flag[i] = 0;
   }

   UF_CALL(UF_MODL_create_curve_mesh(&primary, &cross, &spine,
&end_point,
      &emphasis,&body_type,&spln_pt,boolean,tol,c_face,c_flag,Surf_tag));

   UF_MODL_free_string_list(&primary);
   UF_MODL_free_string_list(&cross);
}

// Построение поверхности по сплайнам
void Insert_SurfSpl ( tag_t  *Primary_tag, int n_Prim, tag_t *Surf_tag)
{
   UF_STRING_t  primary, spine;
   int i,
      alignment = 1,
      body_type = 0,
      c_flag[2]  = { 0,0 }, c_num[2] = {0,0},
      patch = 2,
      vdegree = 3,
      vstatus = 0;
```

```
    double
       tol[3]={0.02,0.02,0.02}, value[6];
    tag_t
       c_face[2] = { NULL, NULL };

    UF_MODL_ask_distance_tolerance(&tol[0]);
    UF_MODL_ask_angle_tolerance(&tol[1]);
    //tol[2] = 0.02;
    for (i=0; i<6; i++)
    {
       value[i] = 0.0;
    }

    UF_MODL_init_string_list (&primary);
    UF_MODL_init_string_list (&spine);

    UF_MODL_create_string_list(n_Prim,n_Prim,&primary); /* n_Prim strings with 1
curve each */

    primary.num = n_Prim; /* number of strings for primary curve */
    for ( i = 0; i < n_Prim; i++)
    {
       primary.dir[i] = 1;
       primary.string[i] = 1;  /* number of entities in the prim string */
       primary.id[i] = Primary_tag[i];
    }

    UF_MODL_create_thru_curves ( &primary, &spine, &patch,
         &alignment, value, &vdegree, &vstatus, &body_type, UF_NULLSIGN,
         tol, c_face, c_flag, Surf_tag);

    UF_MODL_free_string_list(&primary);
    UF_MODL_free_string_list(&spine);
}

// Построение плоскости по граничным кривым
void Insert_Bplane ( tag_t  *Curve_tag, int n_Curve, tag_t *Bplane_tag)
{
 /* bounded plane edge tolerance */
    double tol[3];
    int i;
    UF_STRING_t ge ;

    UF_MODL_init_string_list(&ge);
    UF_MODL_create_string_list(1,n_Curve,&ge);
```

```
/* Store section loop data into string list structure. */
   ge.num = 1; /* number of strings for curve */
   ge.string[0] = n_Curve;
   ge.dir[0] = 1;
   for ( i = 0; i < n_Curve; i++)
   {
      ge.id[i] = Curve_tag[i];
   }

   UF_MODL_ask_distance_tolerance(&tol[0]);
   UF_MODL_ask_angle_tolerance(&tol[1]);
   tol[2] = 0;

/* Create a bounded plane */
  UF_CALL(UF_MODL_create_bplane(&ge,tol,Bplane_tag));
  UF_MODL_free_string_list(&ge);
}

// Построение  тела методом сшивки поверхностей
void Insert_Body ( tag_t  *Surf_tag, int n_surf, tag_t *Body_tag)
{
      tag_t   target[6];
   int i, count,
      SEW_GENERAL = 0,
      SHEET = 1;  // тело
   double  toler;
   uf_list_p_t  disjoint_list;

   UF_MODL_ask_distance_tolerance(&toler);
   for ( i = 0; i < n_surf; i++)
   {
      target[i] = Surf_tag[i];    // Указатель на первую поверхность
   }
   UF_CALL( UF_MODL_create_sew( SEW_GENERAL, 1, target, n_surf,
         Surf_tag, toler, SHEET, &disjoint_list, Body_tag));

   UF_CALL(UF_MODL_ask_list_count( disjoint_list, &count));
   if (count > 0)
   {
     UF_MODL_delete_list(&disjoint_list);
   }
}

// Вычисление минимального расстояния между точкой и узлами сплайна
```

```
double MinDistSpl ( double *Pnt, struct pts_struct *PntSPl, int Begin, int I_Sech, int
*Imin, double *Pmin)
{
   double MinDist, MinDist1, MinDist2;
   double Pnt1[3], Pnt2[3], PntWrk[3];
   int i, i0, iNN;

      i0 = I_Sech * NNodes;
      MinDist = 0.0;
      if ( Begin == 1)      // Входная кромка
      {
         PntWrk[0] = PntSPl[i0].X;
         PntWrk[1] = PntSPl[i0].Y;
         PntWrk[2] = PntSPl[i0].Z;
         UF_CALL(UF_MODL_ask_minimum_dist ( NULL_TAG, NULL_TAG, 1,
PntWrk, 1, Pnt, &MinDist1, Pnt1, Pnt2));
         for (i = i0; i < i0 + NNodes; i++)
         {
            PntWrk[0] = PntSPl[i].X;
            PntWrk[1] = PntSPl[i].Y;
            PntWrk[2] = PntSPl[i].Z;
            UF_CALL(UF_MODL_ask_minimum_dist ( NULL_TAG, NULL_TAG,
1, PntWrk, 1, Pnt, &MinDist2, Pnt1, Pnt2));
            if ( MinDist2 > MinDist1)
            {
               MinDist = MinDist1;           // Минимальное растояние между точкой
и узлами сплайна
               Pmin[0] = PntSPl[i – 1].X;      // Ближайшая узловая точка
               Pmin[1] = PntSPl[i – 1].Y;
               Pmin[2] = PntSPl[i – 1].Z;
               *Imin = i – 1;                  // Индекс ближайшей точки
               break;
            }
            else    MinDist1 = MinDist2;
         }    // for  i
      }
      else                // Выходная кромка
      {
         iNN = I_Sech * NNodes + NNodes – 1;
         PntWrk[0] = PntSPl[iNN].X;
         PntWrk[1] = PntSPl[iNN].Y;
         PntWrk[2] = PntSPl[iNN].Z;
         UF_CALL(UF_MODL_ask_minimum_dist ( NULL_TAG, NULL_TAG, 1,
PntWrk, 1, Pnt, &MinDist1, Pnt1, Pnt2));
         for( i = iNN – 1; i >= I_Sech * NNodes; i--)
```

```
        {
           PntWrk[0] = PntSPl[i].X;
           PntWrk[1] = PntSPl[i].Y;
           PntWrk[2] = PntSPl[i].Z;
           UF_CALL(UF_MODL_ask_minimum_dist ( NULL_TAG, NULL_TAG,
1, PntWrk, 1, Pnt, &MinDist2, Pnt1, Pnt2));
           if (MinDist2 > MinDist1)
           {
              MinDist = MinDist1;            // Минимальное растояние между точкой
и узлами сплайна
              Pmin[0] = PntSPl[i + 1].X;      //  Ближайшая узловая точка
              Pmin[1] = PntSPl[i + 1].Y;
              Pmin[2] = PntSPl[i + 1].Z;
              *Imin = i – 1;                  // Индекс ближайшей точки
              break;
           }
           else  MinDist1 = MinDist2;
        }
      }
     return MinDist;
}

// Вычисление минимального расстояния между точкой и узлами сплайна
double FunCmax ( int I_Sech, int *Imax1, int *Imax2)
{
   double MaxDist, Dist1, Dist2;
   double Pnt1[3], Pnt2[3], PntWrkSp[3], PntWrkKor[3];
   int i, i0;

   i0 = I_Sech * NNodes + 1;
   PntWrkSp[0] = PntSp[i0].X;
   PntWrkSp[1] = PntSp[i0].Y;
   PntWrkSp[2] = PntSp[i0].Z;
   PntWrkKor[0] = PntKor[i0].X;
   PntWrkKor[1] = PntKor[i0].Y;
   PntWrkKor[2] = PntKor[i0].Z;
   UF_CALL(UF_MODL_ask_minimum_dist ( NULL_TAG, NULL_TAG, 1,
PntWrkSp, 1, PntWrkKor, &Dist1, Pnt1, Pnt2));
   *Imax1 = i0;
   for (i = i0+1; i < i0 + NNodes; i++)
   {
      PntWrkSp[0] = PntSp[i].X;
      PntWrkSp[1] = PntSp[i].Y;
      PntWrkSp[2] = PntSp[i].Z;
      PntWrkKor[0] = PntKor[i].X;
```

```
        PntWrkKor[1] = PntKor[i].Y;
        PntWrkKor[2] = PntKor[i].Z;
        UF_CALL(UF_MODL_ask_minimum_dist ( NULL_TAG, NULL_TAG, 1,
PntWrkSp, 1, PntWrkKor, &Dist2, Pnt1, Pnt2));
        if ( Dist2 < Dist1)
        {
           MaxDist = Dist1;                  // Максимальное растояние между узловыми
точками спинки и корыта
           *Imax1 = i – 1;                   // Индекс ближайшей точки
            break;
        }
         else   Dist1 = Dist2;
    }    // for  i
    return MaxDist;
}
// Инициализация вектора XYZ
static void XYZ_init(double vec_out[3], double vec_in[3])
{
    vec_out[0] = vec_in[0];
    vec_out[1] = vec_in[1];
    vec_out[2] = vec_in[2];
}
```

ДОДАТОК В

Код з координатами

```
/*============================================== ======
координаты
==========================================================
================*/

#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include "dialog.h"

#include <uf.h>
#include <uf_defs.h>

#include <uf_styler.h>
#include <uf_exit.h>
#include <uf_ui.h>
#include <uf_mb.h>
#include <uf_attr.h>
#include <uf_part.h>
#include <uf_layer.h>
#include <uf_curve.h>
#include <uf_ugfont.h>
#include <uf_obj.h>
#include <uf_object_types.h>
#include <uf_drf_types.h>
#include <uf_drf.h>
#include <uf_modl.h>
#include <uf_modl_types.h>
#include <uf_modl_freeform.h>
#include <uf_csys.h>
#include <uf_view.h>
#include <uf_vec.h>
#include <uf_assem.h>
```

```
#define UF_CALL(X) (report( __FILE__, __LINE__, #X, (X)))
#define ug_err(THIS_FILE, __LINE__, irc)

static int report( char *file, int line, char *call, int irc)
{
 if (irc)
 {
    char    messg[133];
    printf("%s, line %d:  %s\n", file, line, call);
    (UF_get_fail_message(irc, messg)) ?
      printf("    returned a %d\n", irc) :
      printf("    returned error %d:  %s\n", irc, messg);
 }
 return(irc);
}

// Описание глобальных переменных и констант
#define DELTA 0.01
#define TOLER 0.001
#define  NUMBER_POINTS  300
#define MAX_SECH 50
char ProgDIR[MAX_PATH];   // DIR проекта
int dlg_cancel, count, more_direct, ind_font;
static tag_t work_part;
int dialog_id, n_layer;
int    mpi_array_old[100], mpi_array_new[100];
double mpr_array_old[70], mpr_array_new[70];
char   rad_symbol[27], dia_symbol[27];
short TypeLevel;   // 1 – N слоев, 2 – 2слоя
short TypeDraw;    // Тип построения (тело,ср.пов,тело+ср.пов,сечения)
short TypeSurf;    // Тип построения поверхности (по сечениям,по сечениям и
стрингерам)
double MyDelta, MyToler;
char FileName[MAX_FSPEC_SIZE+1];  // Файл данных
int NNodes, NSech;      // Кол-во узлов и сечений
struct pts_struct {        // Структура 3D точки
  double X;
  double Y;
  double Z;
};
struct pts_struct *PntSp;       // Координаты спинок по сечениям
struct pts_struct *PntKor;       // Координаты корыт по сечениям
int i_size;               // Кол-во точек во всех спинках
struct pts_struct Krom1[MAX_SECH];    // Центры кромок
struct pts_struct Krom2[MAX_SECH];
```

```
double RR1[MAX_SECH], RR2[MAX_SECH];    // Радиусы кромок
static int MyColor[] = { 186, 6, 72, 67, 211, 181, 216, 128, 160, 137, 167, 152, 161,
131, 141, 105, 117, 155, 83,
                        101, 134, 69, 75, 112, 47, 29, 97, 55, 130, 153, 186, 6, 72, 67,
211, 181, 216, 128, 160, 137,
                        67, 152, 161, 131, 141, 105, 117, 155, 83, 101 };

static int DirPositiv = 1;    // Положительное направление лопатки
static int SpUp;      // Расположение спинки относительно корыта ( = 0 – спика
ниже корыта по оси Y)
static int TypeLop;     // Тип лопатки ( = 1 – рабочая, 2 – сечение, 3 – сопловая
double start_sp[MAX_SECH][3], start_kor[MAX_SECH][3],
end_sp[MAX_SECH][3], end_kor[MAX_SECH][3];

static tag_t MyLinWrk[NUMBER_POINTS];

// Описание функций
extern int ReadCoord(void);
extern void DRAW_Prof(void);
void Insert_SpinKor ( struct pts_struct *Pnt_Dat, int NNodes, int i_s, int Spin, int
*ip,
        double *start_pnt, double *end_pnt, tag_t *curv_tag);
void Insert_Mid_Line ( int i_s, int *ip, int i_beg, int i_end, double *Krom1_mid,
double *Krom2_mid,
        tag_t *MidLine_Tag);
void Insert_SurfSpl ( tag_t  *Primary_tag, int n_Prim, tag_t *Surf_tag);
void Insert_Rad (struct pts_struct Krom, double RR,  int i_sech,  tag_t Rad );
void Insert_Krom (  struct pts_struct Krom, double RR, tag_t arc, int i_sech,  int *ip,
int Begin, tag_t sp_tag,
        tag_t kr_tag, tag_t arc_cut, int i_beg, int i_end, double Pnt1[3], double
Pnt2[3],
        double *Krom_mid, double *IntersPtDat, int *in) ;
void Insert_Strings ( struct pts_struct *Pnt_Dat, int *ind_begin, int *ind_end,
      double *start_pnt, double *end_pnt, int ind, tag_t *Strings_tag, int
*n_Strings);
void Insert_SurfMesh ( tag_t  *Primary_tag, int n_Prim, tag_t *Cross_tag, int
n_Cross, tag_t *Surf_tag);
void Insert_Bplane ( tag_t  *Curve_tag, int n_Curve, tag_t *Bplane_tag);
void Insert_Body ( tag_t  *Surf_tag, int n_surf, tag_t *Body_tag);
double MinDistSpl ( double *Pnt,  struct pts_struct *PntSPl, int Begin, int I_Sech,
double *Pmin);
double FunCmax ( int I_Sech, int *Imax1, int *Imax2);
static void XYZ_init(double vec_out[3], double vec_in[3]);
static char *TrimSp (char *str);
```

```
int make_sew(tag_t *sheet_list, int n_sheet, int body_type, tag_t *sew_tag);
int allocate_memory(unsigned int nbytes, void **where);

static UF_STYLER_callback_info_t PROF_cbs[PROF_CB_COUNT] =
{
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_CONSTRUCTOR_CB  , 0,
PROF_constructor_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_DESTRUCTOR_CB   , 0,
PROF_destructor_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_OK_CB           , 0,
PROF_OK_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_CANCEL_CB       , 0,
PROF_cancel_cb},
 {PROF_FIND_FILE        , UF_STYLER_ACTIVATE_CB     , 0,
PROF_action_find_file_cb},
 {PROF_VIEW_FILE        , UF_STYLER_ACTIVATE_CB     , 0,
PROF_action_view_file_cb},
 {UF_STYLER_NULL_OBJECT, UF_STYLER_NO_CB, 0, 0 }
};

static UF_MB_styler_actions_t actions[] = {
   { "UGProf.dlg",  NULL,   PROF_cbs,  UF_MB_STYLER_IS_NOT_TOP },
   { NULL,  NULL,  NULL,  0 } /* This is a NULL terminated list */
};

char *WorkStr;

  tag_t cur_pt;
```