МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут
комп'ютерних наук та управління проектами
Програмного забезпечення автоматизованих систем

Пояснювальна записка

до кваліфікаційної роботи

за темою ____Вдосконалення програмного забезпечення побудови і аналізу тривимірних полігональних моделей об'єктів_____

Виконав: студент 6 курсу, групи 6151м
спеціальності
121 «Інженерія програмного
забезпечення»
(шифр і назва спеціальності)

Філін Д.С.
(підпис, прізвище та ініціали)
Керівник МР Суслов С.В.
(підпис, прізвище та ініціали)
Рецензент МР _____
(підпис, прізвище та ініціали)
Завідувач кафедри Приходько С.Б.
(підпис, прізвище та ініціали)

м. Миколаїв – 2020 р.

Міністерство освіти і науки України
Національний університет кораблебудування
імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра програмного забезпечення автоматизованих систем
Освітній ступень Магістр
Галузь 12 «Інформаційні технології»
(шифр і назва)
Спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

" 26 " 10 2020 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ (МАГІСТЕРСЬКУ) РОБОТУ СТУДЕНТУ

Філін Дмитро Сергійович
(прізвище, ім'я, по батькові)

1.Тема магістерської роботи Вдосконалення програмного забезпечення побудови і аналізу тривимірних полігональних моделей об'єктів
керівник роботи Суслов Сергій Віталійович, канд. техн.. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 26 " жовтня 2020 року №1037-уч
2. Строк подання студентом роботи 01.12.2020 року
3. Вихідні дані до роботи Документація з програмного забезпечення побудови і аналізу тривимірних полігональних моделей об'єктів
4.Зміст магістерської роботи (МР):
• Титульний аркуш, завдання на кваліфікаційну (магістерську) роботу, реферат (українською, англійською), зміст, перелік умовних позначень, символів, одиниць та термінів (за необхідності).
• Вступ (Актуальність теми. Зв'язок роботи з науковими програмами, планами, темами. Мета і завдання дослідження. Об'єкт дослідження. Предмет дослідження. Методи дослідження. Наукова новизна одержаних результатів. Практичне значення одержаних результатів. Особистий внесок здобувача. Апробація результатів досліджень**.** Публікації.)
• Огляд літератури за темою, обґрунтування необхідності проведення досліджень за обраною темою, вибір напрямків досліджень, мета дослідження, основні задачі дослідження
• Викладення результатів власних досліджень з висвітленням того нового, що пропонується
• Проект програмного забезпечення
• Результати досліджень та розробки проекту програмного забезпечення
• Організаційно-економічний розділ
• Розділи з охорони праці та охорони навколишнього середовища
• Висновки
• Список використаних джерел
• Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма і методика випробувань програмного забезпечення)

5. Перелік графічного матеріалу

5.1 Архітектура системи
5.2 Діаграми класів
5.3 Діаграми станів

6. Консультанти розділів магістерської роботи

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

7. Дата видачі завдання 20.10.2020

КАЛЕНДАРНИЙ ПЛАН

Назва етапів кваліфікаційної (магістерської) роботи (МР)	Термін виконання	Примітка
1. Підготовка вступної частини МР	18.10.2020	
2. Підготовка розділу (ів) МР з огляду літератури за темою, обґрунтування необхідності проведення досліджень за обраною темою, вибір напрямів досліджень	19.10.2020	
3. Підготовка розділу (ів) МР з результатів власних досліджень	22.10.2020	
4. Підготовка розділу МР з проекту програмного забезпечення	16.11.2020	
5. Підготовка організаційно-економічного розділу	18.11.2020	
6. Підготовка розділу з охорони праці	20.11.2020	
7. Підготовка розділу з охорони навколишнього середовища	23.11.2020	
8. Підготовка розділу МР – Висновки	25.11.2020	
9. Оформлення списку використаних джерел	27.11.2020	
10. Оформлення додатків	30.11.2020	
11. Підготовка презентації МР та доповіді	01.12.2020	
12. Подання МР на попередній захист	01.12.2020	
13. Подання МР рецензенту	11.12.2020	
14. Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, разом з заявою щодо самостійності виконання роботи та ідентичності друкованої та електронної версії роботи	14.12.2020	
15. Подання на кафедру ПЗАС електронних версії наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК від 19.05.2020 р. за №287-уч); презентації доповіді	18.12.2020	
16. Подання на кафедру ПЗАС письмового відгуку та рецензії на кваліфікаційну роботу	18.12.2020	

Студент _____ Філін Д.С
(підпис) (прізвище та ініціали)

Керівник роботи _____ Суслов С.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Філін Дмитро Сергійович

«Вдосконалення програмного забезпечення побудови і аналізу тривимірних полігональних моделей об’єктів»

Магістерська робота на здобуття освітнього рівня магістра зіспеціальності 121 – «Інженерія програмного забезпечення».

Національний університет кораблебудування імені адміралаМакарова. Миколаїв, 2020 р.

Обсяг роботи: 138 стор., 5 табл., 41 рис., 11 використаних джерел, 5 додаток.

Актуальність теми роботи: Програмне забезпечення побудови і аналізу тривимірних полігональних моделей об’єктів використовується для розрахунків при проектуванні гвинтів суден. Існує потреба у покращенні характеристик його ефективності.

Мета дослідження: підвищення швидкості виконання програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів

Об’єкт дослідження: процес розроблення програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

Предмет дослідження: розроблення програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

Методи дослідження: аналіз, синтез, порівняння, індукція, математичне моделювання.

Інновації: шляхом вдосконалення алгоритмів, оптимізації інформаційної моделі стосовно пошуку окремих елементів та використання багатопоточного оброблення даних підвищено швидкість виконання програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Практичне значення одержаних результатів: полягає у покращенні характеристик ефективності програмного забезпечення побудови і аналізу тривимірних полігональних моделей об’єктів.

Ключові слова: МОДЕЛЮВАННЯ, ТРИВИМІРНІСТЬ, ГВИНТ, ЛОПАСТІ, ПОЛІГОНАЛЬНІ МОДЕЛІ, РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ.

ABSTRACT

Filin Dmytro Serhijovych

"Improvement of software for construction and analysis of three-dimensional polygonal models of objects"

Master's thesis for obtaining the educational level of master's degree in specialty 121 - "Software Engineering".

Admiral Makarov National University of Shipbuilding. Mykolaiv, 2020

Volume of work: 138 pages, 5 tables, 41 figures, 11 used sources, 5 appendix.

Relevance of the topic: The program developed within the framework of diploma design is designed to automate these calculations and, most importantly, to build a three-dimensional model of the resulting screw.

The purpose of the study: to increase the execution speed of software for creating, modifying and analyzing three-dimensional polygonal models of objects

Object of research: the process of developing software for creating, modifying and analyzing three-dimensional polygonal models of objects.

Subject of research: development of software for creation, modification and analysis of three-dimensional polygonal models of objects.

Research methods: analysis, synthesis, comparison, induction, mathematical modeling.

Innovations: by improving the algorithms, optimizing the information model for finding individual elements and using multithreaded data processing, the speed of software for building and calculating the geometric characteristics of three-dimensional polygonal models of objects is increased.

Practical significance of the obtained results: calculation of the necessary characteristics and geometry of the propeller for the calculation of watercraft structures.

Keywords: SIMULATION, THREE-DIMENSIONALITY, SCREW, BLADES, POLYGONAL MODELS, DISTRIBUTED CALCULATIONS.

ЗМІСТ

ВСТУП

Актуальність теми. Задачі обрахунку елементів суднобудування набули нового змісту з появою технологій, що дозволяють автоматизувати оброку складних обчислень з великими масивами структурованих даних. Такі задачі вимагають нових підходів до обробки даних включаючи паралельні та розподілені обчислення. Особлива увага при математичному моделюванні приділяється графічній візуалізації результатів моделювання, особливо з використанням тривимірних побудов. Наявність даних технологій актуалізувало вдосконалення програми розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Головною задачею оптимізації є використання багатопоточного оброблення даних, підвищено швидкість виконання програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Розроблювана програма покликана автоматизувати дані розрахунки та, найголовніше, побудувати тривимірну модель отриманого гвинта.

Мета дослідження: підвищення швидкості виконання програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

Для реалізації поставленої мети ставляться наступні завдання:

- Проаналізувати та обрати структуру даних для вдосконалення програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів;
- Проаналізувати та обрати спосіб обробки даних для вдосконалення програмного забезпечення;
- Проаналізувати та обрати спосіб тривимірної візуалізації для вдосконалення програмного забезпечення;

- Провести вдосконалення програмного забезпечення в розрізі структур даних, методів їх обробки та візуалізації;
- Оцінити ефективність застосування обраних технології для вдосконалення програмного забезпечення.

Об’єкт дослідження: процес розроблення програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

Предмет дослідження - підвищення ефективності вдосконаленого програмного забезпечення внаслідок використання ефективніших структур даних, методів їх обробки та візуалізації.

Методи дослідження: аналіз, синтез, порівняння, математичне моделювання, статистичний аналіз.

Втілена інновація: шляхом вдосконалення алгоритмів, оптимізації інформаційної моделі стосовно пошуку окремих елементів та використання багатопоточного оброблення даних підвищено швидкість виконання програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Практичне значення одержаних результатів: розроблене програмне забезпечення має універсальний характер і може бути використане для інших поверхонь, в тому числі корпусу плавзасобу. Використані полігональні моделі та розподілена обробка даних сприяє отриманню максимально можливої продуктивності роботи прикладних програм.

РОЗДІЛ 1. СТРУКТУРИ ДАНИХ ДЛЯ ПОБУДОВИ І АНАЛІЗУ ТРИВИМІРНИХ ПОЛІГОНАЛЬНИХ МОДЕЛЕЙ ОБЄКТІВ

1.1 Аналіз та обґрунтування вибору структур даних з погляду вдосконалення ПЗ

На даний час відоме досить велике кількість різних методів подання тривимірних об'єктів і пов'язаних з ними методів візуалізації, у тому числі багатомасштабних.

Для вибору оптимальної моделі представлення даних проаналізуємо найвідповідніші з них.

Всі подання можна розділити на кілька класів, що володіють характерними властивостями:

- Поверхневі / об'ємні.
- Зв'язані / дискретні.
- Явні / параметричні.

Поверхневі моделі описують тільки поверхню об’єкта в тривимірному просторі. На противагу їм, об'ємні (воксельні) структури дозволяють задавати моделі, як частину тривимірного простору, розбитого деяким чином на осередки, які вважаються заповненими, якщо вони містять частину об’єкта, і порожніми – у противному випадку.

Зв'язані моделі явно або неявно містять інформацію про безперервні ділянки поверхонь моделей, тоді як дискретні подання описують тільки наближення поверхні об’єкта.

Явне завдання моделей припускає, що опис моделі об’єкта в даному поданні доступно в явній формі, а параметричне – що для його одержання необхідно додатково обчислювати деяку функцію, яка залежить від параметру.

Методи візуалізації можна умовно розділити на проекційні методи й методи трасування променів.

Проекційні методи - це методи, у яких синтез зображення виконується за допомогою афінних перетворень і перетворень проекції. Тривимірна сцена як набір примітивів візуалізації (звичайно, багатокутників, точок, ліній тощо) трансформується у двомірний масив, що і відображається на екрані монітора.

Методи трасування променів працюють на рівні пікселів вихідного зображення, розраховуючи їхні кольори на основі даних про геометрію сцени і положення віртуальної камери.

Сьогодні інтерактивну швидкість синтезу зображень надають тільки проекційні методи, часто за підтримкою апаратного забезпечення. Надалі в роботі розглядаються тільки такі методи, які в свою чергу накладають деякі обмеження на можливі подання об'єктів. Характерною рисою поставленого завдання є робота з реальними даним, тобто даними, введеними в комп'ютер за допомогою пристроїв дистанційного сканування. Такі дані, в більшості випадків, дискретні й задані явно, звичайно у вигляді набору точок або набору карт глибини. Крім того, за винятком томографів (цей випадок у роботі не розглядається), що одержують внутрішню структуру об’єкта, всі сканери працюють тільки з поверхнею об’єкта. Таким чином, подання повинне гарно описувати явно задані дискретні поверхні.

Проведемо аналіз різних подань із метою виявлення придатності їхнього використання для рішення поставленого завдання. У тексті не розглядаються різні параметричні й процедурні подання тому, що їх особливості (складність обчислень, відсутність апаратної підтримки) роблять складним використання цих подань для моделювання реальних об'єктів.

Воксельні моделі

Класичні воксельні (voxel) моделі являють собою тривимірний масив, кожному елементу якого зіставлений колір й коефіцієнт прозорості. Такий масив задає наближення об’єкта з точністю, обумовленої обмеженням масиву.

Воксельні (або об'ємними) методами візуалізації називаються методи візуалізації тривимірних функцій, у дискретному випадку заданих, наприклад, за допомогою описаного вище масиву.

Типові методи воксельної візуалізації обробляють масив, і формують проекцію кожного його елемента на видову площину. Вихідний масив являє собою регулярну структуру даних, що істотно використовується в методах візуалізації. Звичайно елемент масиву з’являється на екрані у вигляді деякого примітиву, так званого відбитку (footprint) або сплату (splat). Різні методи відрізняються способами обчислень форми й розмірів зображення[3].

Обсяги даних у воксельних поданнях значні, навіть для невеликих моделей. Єдиною реальною можливістю працювати зі складними об'єктами є використання деревоподібних ієрархій. У роботі Лаур Д. та Ханрахана П. на основі вихідного масиву будується багатомасштабне подання у вигляді восьмеричного дерева. Кожен вузол дерева містить усереднене значення кольорів і прозорості всіх своїх нащадків. Крім того, кожен вузол дерева містить змінну, що показує середню помилку, асоційовану з даним вузлом. Ця змінна показує помилку, що виникає при заміні оригінального набору вокселів в даній області простору на константну функцію, рівну середньому значенню кольорів всіх нащадків даного вузла. Надалі це значення використовується для керування якістю візуалізації й рівнем деталей.

Незважаючи на те, що воксельні методи орієнтовані в першу чергу на наукову візуалізацію, багато ідей, що використовується в цих методах, знаходять своє застосування в інших областях. Наприклад, ідея решітки використається при роботі із точковими поданнями, а також з поданнями, заснованими на зображеннях.

Переваги даного алгоритму: простота регулярної структури; апаратна підтримка.

Недоліки даного алгоритму: великий обсяг даних, тому необхідно використовувати спеціальні багатомасштабні структури для роботи зі складними об'єктами; використовувані структури даних зберігають внутрішньої, невидимі, частини об’єкта, тоді як для поставленого завдання достатній опис поверхні.

Моделі, засновані на зображеннях

Моделювання й візуалізація, засновані на зображеннях (Image-Based Modeling and Rendering, далі IBMR) являють собою альтернативний підхід до рішення завдань синтезу зображення [4].

Такі методи не використовують проміжні структури даних, і синтезують підсумкову картинку, ґрунтуючись на вихідних даних - як правило, зображеннях або зображеннях з глибиною. Більш формально метод візуалізації, заснований на зображеннях, можна визначити як алгоритм, що визначає, як по кінцевому наборі вихідних (reference) зображень сцени одержати нове, результуюче (resulting) зображення для заданої точки спостереження й заданих параметрів віртуальної камери.

Структури даних, використовувані для такого алгоритму візуалізації, можуть сильно відрізнятися, незмінним залишається орієнтація методів на безпосередню роботу з вихідними даними, що робить методи IBMR концептуально близькими до поставленого завдання.

Зображення з картами глибини

Однієї з найпростіших структур даних, використовуваних в IBMR є набори зображень із картами глибини. Визначимо пари зображення плюс карта глибини як кольорове зображення, якій зіставлене напівтонові зображення відповідного розміру, інтенсивність у кожній точці якого відповідає відстані від камери до поверхні об’єкта.

Примітною властивістю подання є те, що сучасні дистанційні сканери дозволяють прямо одержувати дані у вигляді карт глибини, а найбільш дорогі моделі одержують і колірну інформацію про об'єкт. Отже, таке подання максимально підходить для роботи зі складними реальними даними, а завдання полягає в розробці методу візуалізації.

Варто помітити, що пари зображення плюс карта глибини однозначно визначає дискретне наближення поверхні в тривимірному просторі, при цьому якість наближення залежить від роздільної здатності зображення й обраного положення камери.

Одна карта глибини зберігає тільки видиму частину об’єкта, тому для відновлення повного об’єкта необхідно використати набір з декількох карт глибини, залежно від складності сцени (Рисунок 1.1).

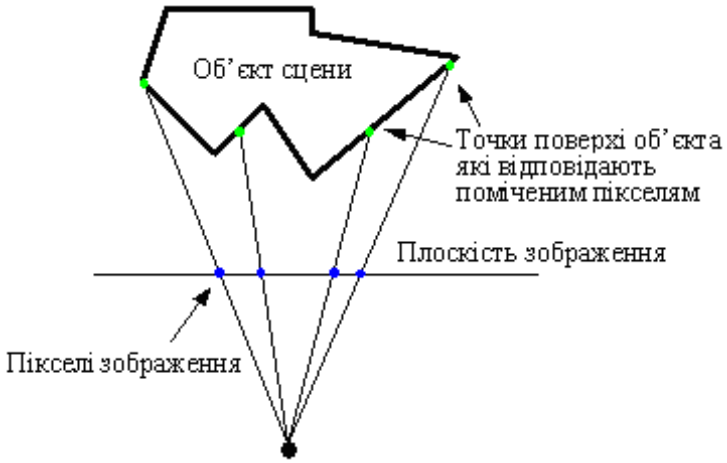


Рисунок 1.1 – Створення карти глибин по пікселям [5]

Було запропоновано досить багато методів візуалізації й використання подібних структур даних. Наприклад, Леонардо-Макмиллан використовує систему обробки зображень для деформації (warping) вихідного зображення з обліком вихідної й результуючої (поточної) камер таким чином, щоб результат, відображений на екрані, створював ілюзію тривимірності [5].

У роботі Мартіна Олів’єрі також використовується деформація зображень, однак результатом роботи алгоритму є текстури створені з карт глибини для поточного положення віртуальної камери й накладені на просту (плоску) полігональну сітку - так називані рельєфні текстури (relief textures) [6].

Однак ці методи мають серйозні недоліки. З одного боку, в умовах недостатньої точності вихідних даних й або великому відхиленні віртуальної камери від вихідної, у результуючому зображенні можлива поява дірок (holes), тобто погіршення якості візуалізації. З іншого боку, результатом роботи дистанційних сканерів часто є набори даних з 50-70 карт глибини, які в описаних вище алгоритмах будуть оброблятися сепаратно, створюючи додаткові погрішності візуалізації. Крім того, час візуалізації однієї карти

глибини розміром 512x512 по методу Олів’єрі на комп'ютері із процесором Pentium III 866 і відео картою NVidia GeForce2 Pro становить близько 70 мс. Обробка 50-ти зображень займе біля 4-х секунд.

Іншим можливим варіантом є пряме відновлення тривимірних координат семплів (sample) і їхня візуалізація прямо за допомогою проекції на видову площину віртуальної камери. Такий підхід дозволяє використати апаратне прискорення тому, що пікселі вихідних зображень у просторі можуть бути представлені крапками або багатокутниками. Однак, на практиці такий метод працює тільки для досить невеликих наборів даних.

Головною перешкодою для створення багатошарових методів візуалізації карт глибини є відсутність чіткої просторової структури пари зображення плюс карта глибини.

Багатошарові зображення із глибиною

Останнім часом було почато кілька спроб використання багато масштабних методів разом із заснованими на зображеннях поданнями. Одна з них описана в роботі Чанга й Бішопа й як базове подання використовує багатошарові зображення із глибиною (Layered Depth Images - LDI), у перше описані в статі Гортлера С. Солена М. (Візуалізація багатошарових глибин зображення

Багатошарові зображення із глибиною зберігають для кожного пікселя карти кольорів всі перетинання відповідного променя з моделлю. Одного багатошарового зображення досить для опису повного об’єкта (Рисунок 1.2).

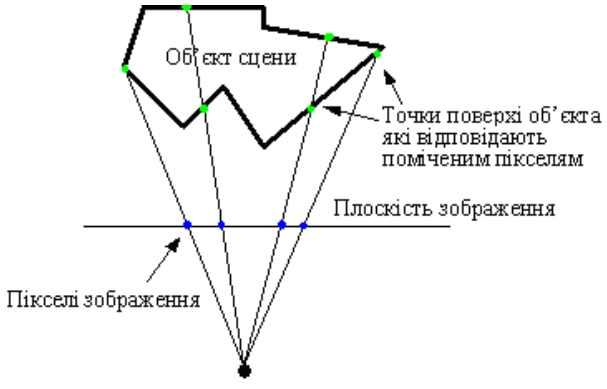


Рисунок 1.2 – Багатошарове зображення [5]

Відмінність багатошарових зображень із глибиною від простих полягає в тім, що одне зображення дозволяє зберігати інформацію не тільки про видиму з даної вихідної камери частини поверхні об’єкта, а повну інформацію про об'єкт. По суті, LDI – це тривимірна структура даних, що представляє собою прямокутну матрицю, кожним елементом якої є список крапок. Кожна крапка містить глибину (відстань до опорної площини) і атрибути, у найпростішому випадку – кольори. Для подання всього об’єкта можна використати єдине багатошарове зображення, що використовує шість перспективних LDI з єдиним центром проекції (3).

Така структура дозволяє проводити візуалізацію як описаними вище методами Макмілана й Олів’єрі, так і просто використати збережену інформацію як скупчення точок і відображати його прямо за допомогою одного із графічних API (наприклад, OpenCL).

З використанням LDI-подібних структур зв'язані деякі обмеження на візуалізацію, обумовлені тим, що всі крапки в зображенні орієнтовані на одну базову площину. Крім того, LDI не можуть бути прямо отримані із пристроїв введення й для створення такої структури необхідне використання додаткових алгоритмів, наприклад, деформуючи зображення із глибиною по методу Макмілана таким чином, щоб площина результуючого зображення збігалася з базовою площиною LDI. Відзначимо, що процес формування LDI відбувається до безпосередньої візуалізації, і тому його ефективність не відбивається на швидкості візуалізації.

Однак LDI не дозволяє прямо відображати об'єкт із різними ступенями деталізації. Але була почата спроба створити багатомасштабне подання на основі LDI з використанням так називаного дерева LDI (LDI tree).

Сутність методу полягає в наступному: замість одного LDI формується восьмеричне дерево, у кожному вузлі якого перебуває свій LDI і посилання на інші вузли, у яких перебуває LDI меншого розміру (в одиницях сцени), але того ж дозволу. Також для кожного вузла є обмежуючий паралелепіпед.

Всі LDI у дереві мають однаковий дозвіл. Висота дерева залежить від дозволу LDI. Чим менше дозвіл LDI, тим більше висота дерева. Кожен LDI у дереві містить інформацію тільки про ту частину сцени, що втримується в його обмежуючому паралелепіпеді. Обмежуючі паралелепіпеди вузлів наступного рівня дерева виходять дробленням обмежуючого паралелепіпеда поточного рівня на вісім рівних частин (Рисунок 1.3).

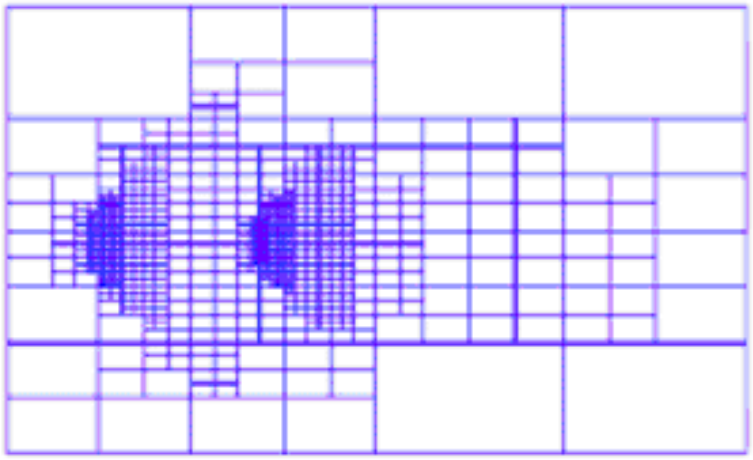


Рисунок 1.3 – Дерево із LDI [6]

Дерево LDI дозволяє вирішувати проблему візуалізації дуже великих структур даних, використовуючи наступну ідею: при візуалізації немає необхідності обробляти нащадків вузла, якщо сам вузол забезпечує достатній ступінь деталізації. Автори використають наступний критерій ступеня деталізації: вважається, що LDI забезпечує достатній рівень деталізації, якщо "відбиток" (footprint, splat) його пікселя на результуючому зображенні покриває не більше одного пікселя екрана.

З іншого боку, використання того ж підходу дозволяє доповнити дані низької роздільної здатності штучно відновленими додатковими рівнями дерева, створюючи ефект фільтрації одержуваного зображення.

Візуалізація виробляється за допомогою обходу дерева від кореня до листів і малювання LDI методом Макмілана. При цьому обробка всіх вузлів

дерева не потрібно, і обхід вітки дерева завершується на першому LDI, що забезпечує достатню точність.

Алгоритм має високу якість візуалізації, можливість прогресивної передачі даних. Однак його ефективність, як за часом, так і по пам'яті, досить низька. Час одержання зображення в дозволі 512x512 для LDI середньої складності на графічній станції SGI Onyx2 (16Гбайт оперативної пам'яті, 32 процесора MIPS R1000 250Mhz) зайняло більше трьох секунд.

Переваги даних алгоритмів: орієнтація на проблемну область; легкість одержання й моделювання.

Недоліки даних алгоритмів: складні, не завжди якісні методи візуалізації; труднощі з підтримкою багатомасштабності; робота тільки з дифузійними поверхнями.

Точкове подання

Класичні подання, засновані на зображеннях, спрямовані на використання зображень як примітиви візуалізації. Підвищення ефективності візуалізації досягається за рахунок того, що час візуалізації в них пропорціональне не складності геометрії, як у традиційних системах, заснованих на полігональних сітках, а числу пікселів у вихідних зображеннях. Хоча такі підходи досить добре працюють для візуалізації складних об'єктів, вони, як правило, вимагають значних обсягів пам'яті, візуалізація страждає від появи артефактів у результуючому зображенні, а також від неможливості роботи з динамічним освітленням. Крім того, на сьогодні не розроблено ефективних багатомасштабних алгоритмів.

Клас точкових (point sample) алгоритмів компенсують недоліки IBM при частковому збереженні описаних вище переваг [7].

Об'єкти моделюються, як щільний набір точок поверхні, які відновлюються з вихідних зображень і зберігаються разом з кольорами, глибиною й інформацією про нормалі, уможливлюючи використання Z-буфера для видалення невидимих поверхонь, затінення по Фонгу, і інші ефекти, наприклад, тіні (Рисунок 1.4).

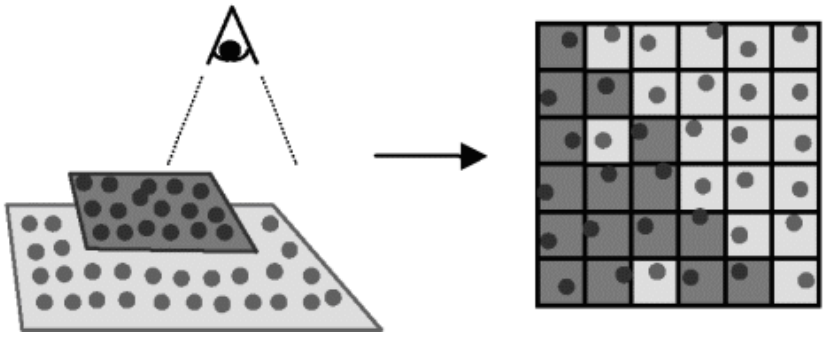


Рисунок 1.4 – Дірки при візуалізації крапкових моделей [6]

Метод візуалізації таких даних нагадує класичні методи деформацій (warping), але з тим розходженням, що точки містять додаткову інформацію про геометрію, і вони видо незалежні (view independent), тобто кольори точки не залежить від напрямку, з якого вона була відновлена. Подібний підхід використався для реалістичної візуалізації дерев [2].

Основною проблемою візуалізації моделей у таких поданнях є відновлення безперервних поверхонь, тобто гарантія відсутності дірок після того, як положення кожної крапки буде наведено до віконних координат. Одним з можливих практичних рішень є решітка (splatting), тобто обчислення форми «відбитка» точки на площині екрана. Решітка також часто використається в методах, заснованих на зображеннях. Можливі також інші підходи, наприклад комбінація ресемплінгу й ієрархічного z-буфера, недоліками яких є недостатнє використання сучасних апаратних прискорювачів, що виражається в часі візуалізації близько 3-5 секунд на кадр для нескладних сцен.

Головним недоліком точкових подань, як і багатьох інших, є складність із поданням великих обсягів даних. Використання неструктурованого набору крапок дозволяє досягти певної гнучкості при візуалізації, але при збільшенні обсягу на перший план виходять методи відео залежного спрощення, використати які в реальному часі не представляється можливим без введення додаткових структур даних.

Переваги даного алгоритму: орієнтація на проблемну область; легкість одержання й моделювання; значна гнучкість у методах візуалізації.

Недоліки даного алгоритму: труднощі з підтримкою багатомасштабності; неякісна або повільна візуалізація.

Ієрархічні подання

Як правило, ієрархічні подання будуються на базі одного з подань, розглянутих вище. У переважній більшості випадків для побудови ієрархій використовуються деревоподібні структури, наприклад, восьмеричні дерева, або kd-дерева. Також використаються загальні види тривимірних дерев на основі ієрархій що описують сфери та куби. Властивості ієрархій можуть сильно розрізнятися залежно від базового подання, однак загальні принципи побудови й візуалізації ієрархічних структур подібні між собою.

Нижній рівень дерева, тобто сукупність листових вершин, являє собою максимально деталізовану модель. Далі, піднімаючись до кореня, атрибути усередняться, структура спрощується до виродження в один примітив на вершині дерева. Візуалізація, навпаки, виробляється за допомогою рекурсивного обходу дерева від вершини до листів.

Ієрархічні структури на базі полігональних сіток використаються для динамічного контролю за рівнем деталізації й складністю моделі. У методі злиття вершин з вузлом дерева асоційована вершина сітки, і перехід до вище поставленого вузла дерева здійснюється за допомогою злиття вершин.

Точкові семпли - це точки, що володіють кінцевим розміром і положенням у просторі, тобто вони є аналогом нерівномірно розподілених у просторі вокселів. На їхній основі будується ієрархія сфер, таких що, сфера, асоційована із внутрішнім вузлом дерева, містить безліч своїх нащадків.

Переваги даного алгоритму: підтримка багатомасштабності; можливість прогресивної обробки й передачі по мережі; можливість динамічного контролю над рівнем деталізації.

Недоліки даного алгоритму: необхідність у попередній обробці, або перетворенні з іншого подання; у загальному випадку більша вартість обробки

примітива, чим у лінійних поданнях; часте збільшення обсягу через необхідність підтримки багатомасштабності (для зберігання показників тощо).

Для реалізації автоматизованого розрахунку гвинта корабля доцільно використати полігональні сітки.

Полігональні сітки є на даний момент найпоширенішим поданням, для якого створене велика кількість програмного забезпечення, що дозволяє редагувати, передавати по мережі й відображати моделі з використанням апаратної підтримки.

Характерною перевагою полігональних сіток є підтримка зв’язності моделі. У силу цього полігональні подання добре пристосовані для опису великої кількості синтетичних поверхонь.

В нашому випадку кожен гвинт описується структурою данних, що містить понад 40 елементів.

Однак, відскановані дані, споконвічно не містять інформації про зв’язність й безперервність поверхонь, а являють собою набір близько розташованих часток (sample). Такі обмеження випливають із пристрою скануючого механізму, що має дискретний крок кінцевого розширення.

Отже, для використання полігональних моделей зв’язність повинна бути введена штучно на етапі препроцесування. Таким чином, полігональні моделі не призначені для прямої роботи з відсканованими даними, тому що вимагають відновлення поверхні, що в загальному випадку є нетривіальним завданням і сильно залежить від класу оброблюваних об'єктів. При цьому відновлена поверхня не обов'язково буде використовуватися на етапі візуалізації (наприклад, якщо модель такої складності, що проекція трикутника на екран при типовій проекції перегляду близька по площі з одним пікселем).

З іншого боку, створена велика кількість методів, що дозволяють досить ефективно перетворювати дискретні відскановані дані в полігональні сітки. Більше того, сучасне устаткування високого класу дозволяє виконувати перетворення в сітку апаратно [1].

Набагато складніша ситуація з поданням великих обсягів даних і підтримкою різних рівнів деталізації. Структура полігональних сіток лінійна й вони не забезпечують «природної» підтримки багатомасштабності. Тому робота з великими сітками ускладнена і потребує різних, найчастіше обчислювально складних методів спрощення. Було створено безліч алгоритмів для створення багатомасштабних подань на основі сіток, що мають безпосереднє відношення до поставленого завдання.

Практично всі технології спрощення сіток використовують деякі варіації або комбінації наступних механізмів: семплюваня (sampling), проріджування (decimation), адаптивної розбивки (adaptive subdivision) і злиття вершин (vertex merging) [2].

Алгоритми семплюваня спрощують первісну геометрію моделі, використовуючи або підмножину вихідних точок, або перетинання вокселів з моделлю на тривимірній сітці. Такі алгоритми найкраще працюють на гладких поверхнях без гострих кутів.

Алгоритми, що використовують адаптивну розбивку, знаходять просту базову (base) сітку, що потім рекурсивно розбивається для апроксимації первісної моделі. Такий підхід працює добре, коли знайти базову модель відносно просто. Наприклад, базова модель для ділянки ландшафту звичайно прямокутник. Для досягнення гарних результатів на довільних моделях потрібне створення базової моделі, що відбиває важливі властивості вихідної, що може бути нетривіально.

Проріджуючи алгоритм, ітеративно видаляє вершини або грані з полігональної сітки, роблячи тріангуляцію після кожного кроку. Більшість із них використають тільки локальні зміни, що дозволяє виконувати спрощення досить швидко (Рисунок 1.5).

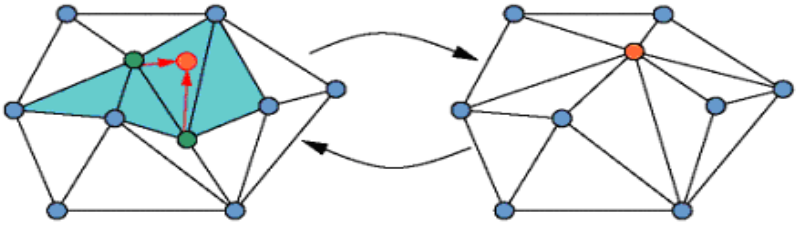


Рисунок 1.5 – Об’єднання ребер (процес тріангуляції) [4]

Схеми зі злиттям вершин працюють за допомогою об’єднання двох або більше вершин деталізованої моделі в одну, котра у свою чергу може бути сполучена з іншими вершинами. Злиття вершин трикутника знищує його, зменшуючи загальне число трикутників моделі. Звичайно алгоритми використовують складні методи визначення, які вершини потрібно об’єднати разом і у якому порядку. Методи, що використовують злиття ребер (edge collapse), завжди зливають вершини, що розділяють одну грань. Такі методи зберігають локальну топологію і, крім того, при деяких умовах можуть працювати в реальному часі.

Складність обчислень у цих методах висока і їхнє використання не завжди виправдане при роботі з дискретними даними, оскільки складність з'являється насамперед через необхідність підтримувати зв’язаність моделі. Іншою причиною високої складності методів є лінійна структура сітки, яку необхідно відновлювати для візуалізації за допомогою графічних API.

Переваги даного алгоритму: розповсюджене представлення; апаратна підтримка.

Недоліки алгоритму: неефективні для роботи з дискретними даними через штучну підтримку зв’язності, складного препроцесінга; неефективні для більших моделей через труднощі з організацією багатомасштабності.

1.2Розробка структур даних

Вище розглянуто аналіз та обґрунтування вибору структур даних, необхідних для оптимізації програми. Головною задачею оптимізації є використання багатопоточного оброблення даних підвищено швидкість виконання програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Розглянемо задачу реалізації полігональної структури даних методами багатопоточного оброблення даних.

Для програмної реалізації розрахунку гвинта в полігональній моделі використаємо структуру класу Model: class Model. Даний клас є бібліотечним.

В програмі використано 4 таких класи:

- для збереження вхідних характеристик,
- для проміжних обчислень гвинта,
- для вихідної інформації про гвинт,
- для побудови тривимірного зображення.

Клас для збереження вхідних характеристик є полігональною структурою (рис. 1.6) та містить, зокрема, наступне:

- геометричний вектор TDirection, що містить характеристики граней гвинта;

- TFaceEdgeDescriptionзберігає індекси опорних вузлів вектора в контексті межі;

- TDirectionDescription містить список граней та покажчик на список, за яким можна перевіряти знаходження об'єкта в великих списках;

- конструктори, деструктори класу;

- процедури та функції для роботи з представниками класу.

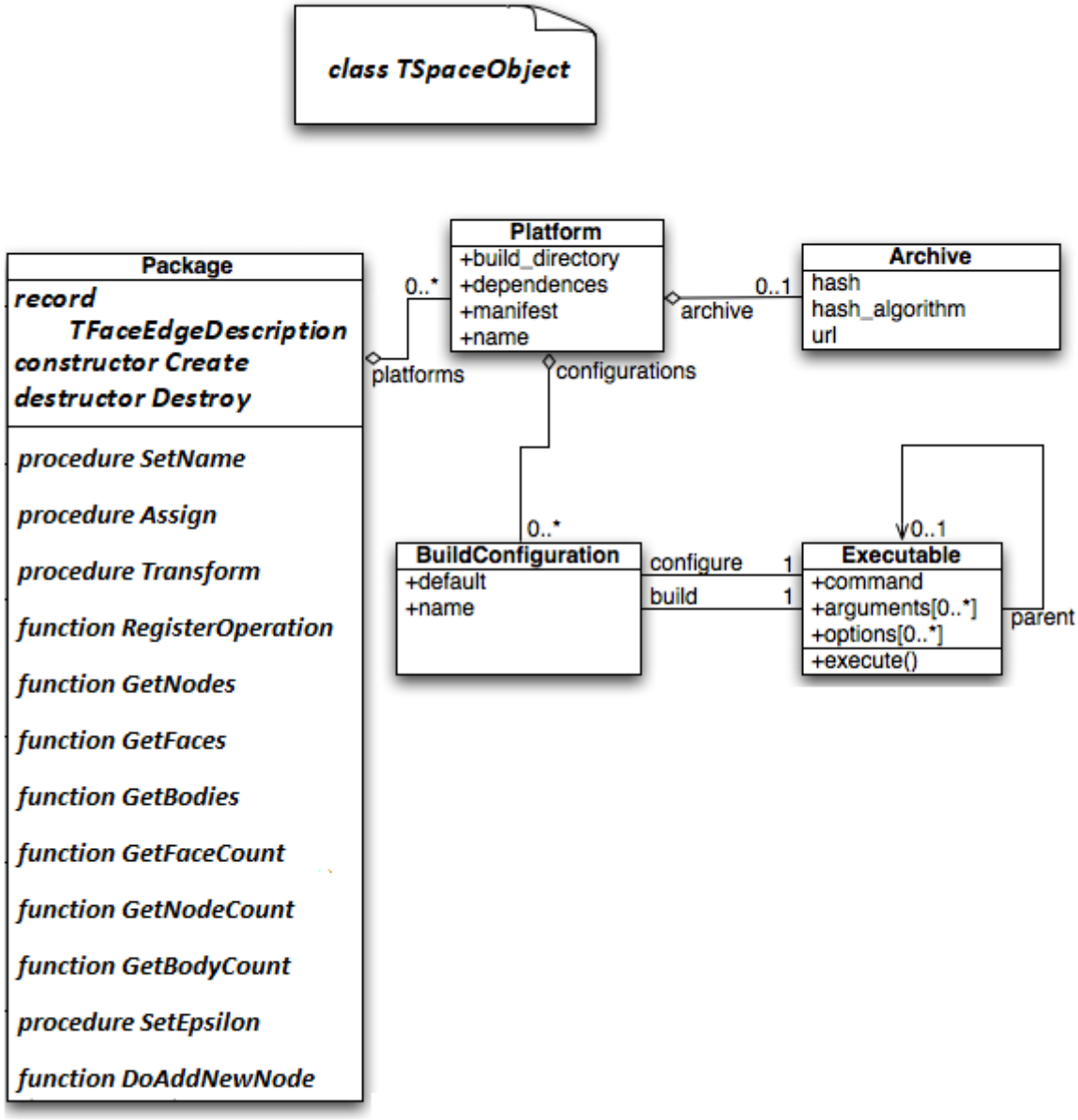


Рисунок 1.6 – Клас для збереження вхідних характеристик

Клас для збереження проміжних обчислень характеристик гвинта є полігональною структурою (рис. 1.7) та містить, зокрема, наступне:

- function TriangulateBody, функція тріангуляції;
- function NodeIndexзберігає індекси розрахованих вузлів;
- конструктори, деструктори класу;
- процедури та функції для роботи з представниками класу.

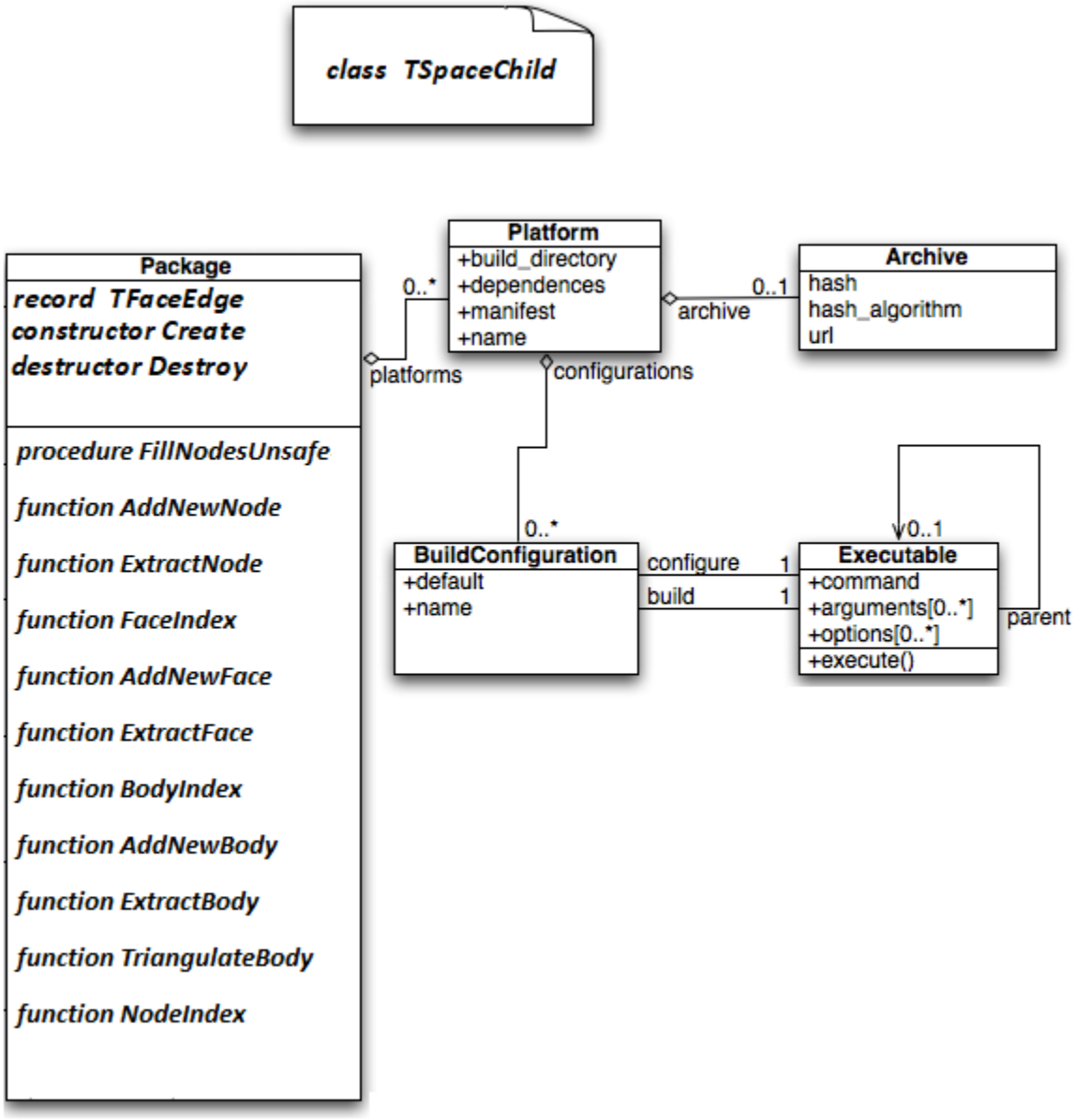


Рисунок 1.7 – Клас для збереження проміжних обчислень

Клас для збереження вихідної інформації про гвинт е полігональною структурою (рис. 1.8) та містить, зокрема, наступне:

- function GetBodies – інформацію про топологію гвинта;
- function GetIsTriangle – результати триангуляції;
- function GetIndexInSpace - співвідношення з площиною виводу графіки.

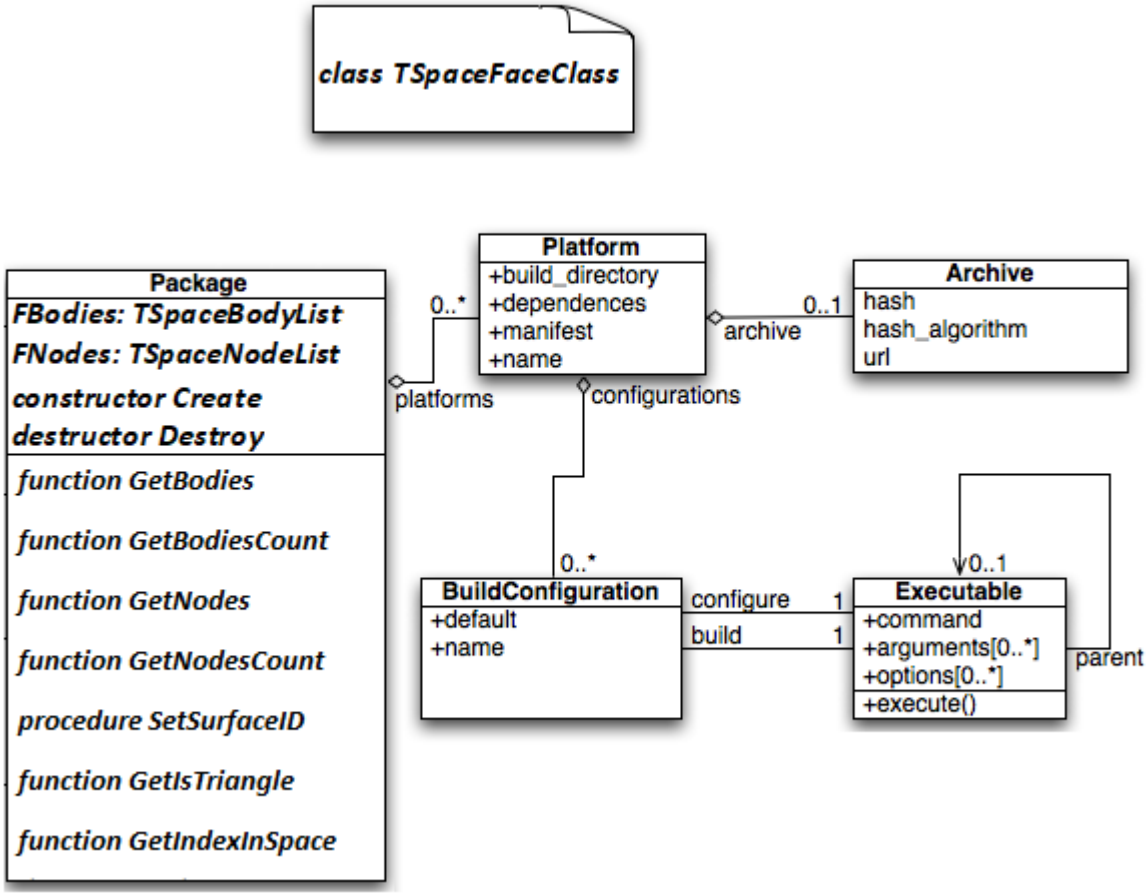


Рисунок 1.8 – Клас для збереження вихідної інформації про гвинт

Клас для збереження інформації про гвинт для побудови тривимірного зображення є полігональною структурою (рис. 1.9) та є стандартним об’єктом OpenCL, що не потребує змін та використовується в первісному вигляді.

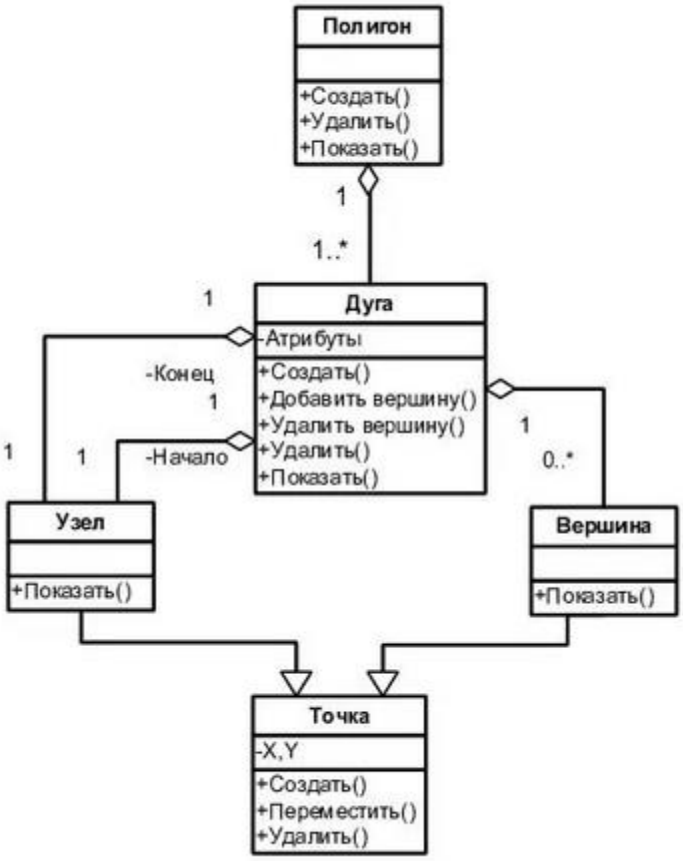


Рисунок 1.9 – Клас для збереження інформації про гвинт для побудови тривимірного зображення

В контексті OpenCL існує деяка кількість точок прив'язки, що визначають куди прив'язати юніформ-буфер. Створивши юніформ-буфер з'єднуємо його з однією з точок прив'язки, а так само з'єднуємо юніформ-блок з цієї ж точкою, по суті пов'язуючи їх разом. Діаграма нижче наочно показує це:

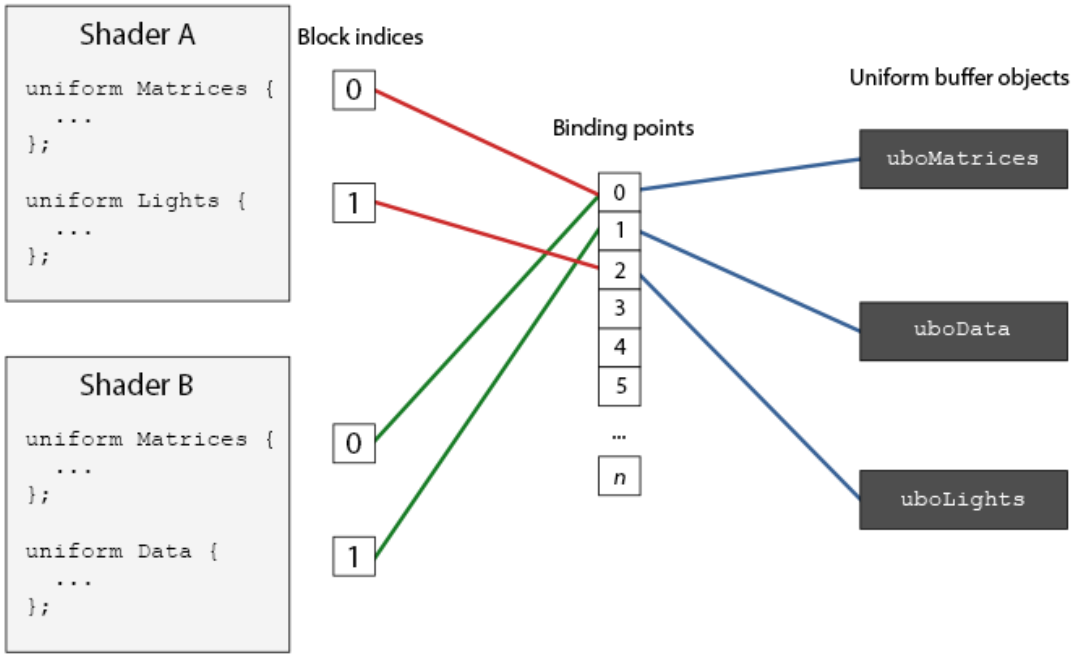


Рисунок 1.10 – Клас Model. Обмін даними

Закритий (private) розділ класу Model містить вектор, що складається з об'єктів типу Mesh. Далі, при створенні об'єкта класу, конструктор запитує розташування файлу моделі. Потім він відразу ж завантажує файл за допомогою функції loadModel (), яка викликається в ньому ж.

Функція Draw () по суті є лише обгорткою для циклу for, за допомогою якого проходимся по всьому мешам, викликаючи у кожного його власну функцію Draw ():

```
void Draw(Shader &shader)
{
for(unsigned int i = 0; i < meshes.size(); i++)
    meshes[i].Draw(shader);
}
void Draw(Shader &shader)
{
for(unsigned int i = 0; i < meshes.size(); i++)
    meshes[i].Draw(shader);
}
```

Імпорт 3D-моделі в OpenCL

Щоб імпортувати модель (рис. 1.11) і перевести її в нашу власну структуру даних, необхідно підключити відповідні заголовки бібліотеки Assimp:

#include <assimp/Importer.hpp>

#include <assimp/scene.h>

#include <assimp/postprocess.h>

#include <assimp/Importer.hpp>

#include <assimp/scene.h>

#include <assimp/postprocess.h>

Перша функція, що використана, - це loadModel (), вона викликається безпосередньо з конструктора. Усередині loadModel ()задіємо бібліотеку Assimp для завантаження моделі в структуру даних Assimp, звану об'єктом сцени.

Бібліотека може акуратно створити шар абстракції, відокремлюючи від купи технічних деталей завантаження даних з різних форматів файлів, за допомогою одного рядка коду:

Assimp::Importer importer;

const aiScene *scene = importer.ReadFile(path, aiProcess_Triangulate | aiProcess_FlipUVs);

Assimp::Importer importer;

const aiScene *scene = importer.ReadFile(path, aiProcess_Triangulate | aiProcess_FlipUVs);

Спочатку оголошуємо об'єкт importer типу Importer з простору імен Assimp, а потім викликаємо у створеного об'єкту функцію ReadFile (). В якості аргументів функція очікує отримати шлях до файлу моделі і кілька параметрів постобробки. Assimp дозволяє вказати кілька параметрів, які змушують бібліотеку виконувати додаткові обчислення / операції з імпортованими даними. Ставлячи параметр aiProcess_Triangulate, повідомляємо Assimp, що, якщо модель не тільки повністю з трикутників, то необхідно спочатку перетворити все примітивні форми моделі в трикутники. Параметр

aiProcess_FlipUVsперевертає під час обробки координати текстури на осі Y, де це необхідно. Але це не все опції, є ще кілька:

aiProcess_GenNormals - створює нормальні вектори для кожної вершини, якщо модель не містить нормальних векторів.

aiProcess_SplitLargeMeshes - розбиває великі меші на кілька мешів меншого розміру, що корисно, коли ваш рендеринг вже містить максимально дозволену кількість вершин або може обробляти тільки меші невеликих розмірів.

aiProcess_OptimizeMeshes - робить протилежне, намагаючись об'єднати кілька мешів в один великий меш, тим самим покращуючи оптимізацію програми за рахунок зменшення кількості викликів функції відтворення об'єкта.

Assimp надає великий набір опцій постобробки, з повним списком яких ви можете ознайомитися тут . Завантаження моделі через Assimp (як ви можете бачити) дивно проста. Найскладніша частина роботи полягає в тому, щоб, використовуючи повертається об'єкт сцени, перетворити завантажені даних в масив об'єктів типу Mesh.

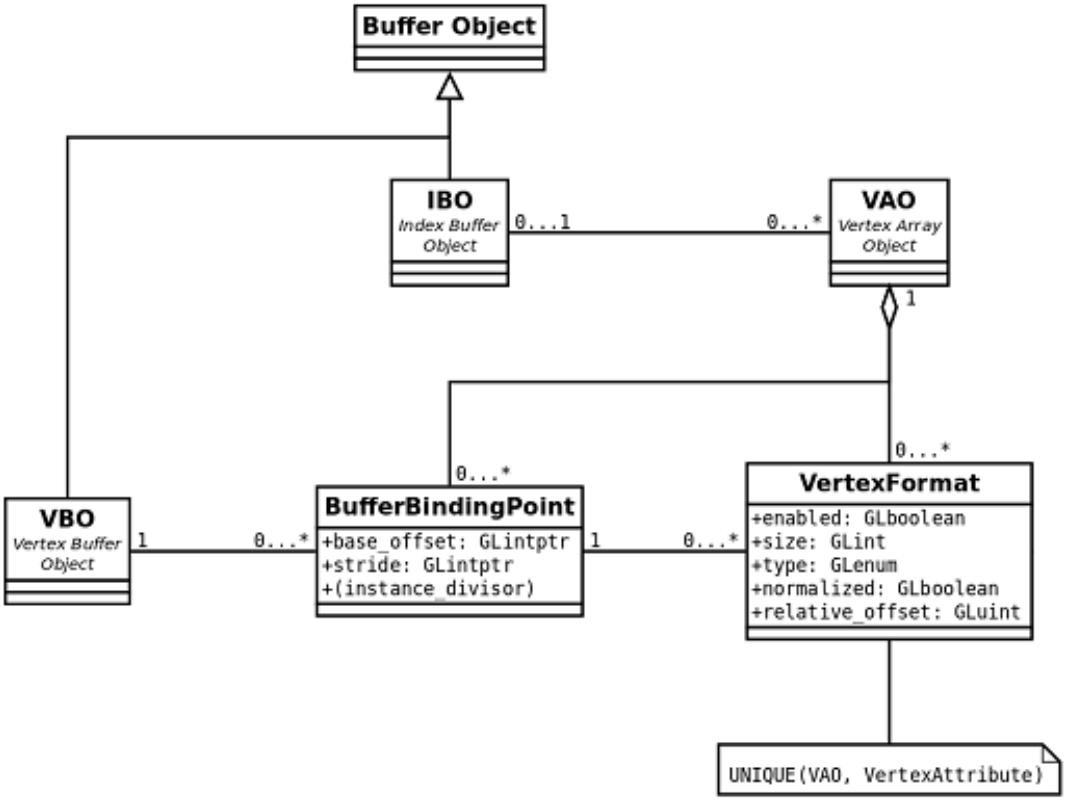


Рисунок 1.11. Модель функції loadModel ()

Після завантаження моделі повинні упевнитися, що змінні сцени і кореневого вузла сцени не є нульовими, а також за допомогою перевірки одного з прапорів сцени переконуємося, що повертаються дані є повними. Якщо будь-яка з цих умов не виконується, то повідомляємо про помилку, повертаючи її опис за допомогою функції GetErrorString ().

Якщо процес завантаження моделі пройшов успішно, то починаємо обробляти всі вузли сцени. Передаємо перший вузол (кореневої) рекурсивної функції processNode (). Оскільки кожен вузол (можливо) містить набір дочірніх елементів, то необхідно спочатку обробити обраний вузол, а потім продовжити обробку всіх його дочірніх елементів і так далі. У нашому випадку умова виходу з рекурсії виконається тоді, коли всі вузли будуть оброблені.

Кожен вузол містить набір індексів мешів, що вказують на певний меш в об'єкті сцени. Таким чином, спочатку необхідно отримати меш-індекси, потім отримати кожен окремий меш, обробити його, і, під кінець, знову виконати всі ці дії для кожного з дочірніх вузлів. Зміст функції processNode () показано нижче.

Спочатку перевіряємо кожен меш-індекс обраного вузла і витягаємо відповідний меш, індексуючи масив mMeshesсцени. Потім отриманий меш передається в функцію processMesh (), яка повертає об'єкт типу Mesh, який тепер можемо зберегти у змінній meshes типу список / вектор.

Після того, як всі меші будуть оброблені, почнемо перебирати дочірні елементи обраного вузла і викликати одну й ту ж функцію processNode () для кожного дочірнього елемента. Як тільки переберемо всі дочірні елементи вузла, рекурсія припиняється.

Переклад об'єкта aiMeshт в наш власний меш-об'єкт не надто складний. Все, що потрібно зробити, - це отримати доступ до кожного з відповідних властивостей меша і зберегти їх в нашому власному об'єкті. Обробка - це процес, що складається з 3-х частин:

- вилучення всіх вершинних даних;
- витяг індексів;

– витяг відповідних даних про матеріал.

Оброблені дані зберігаються в 3-х векторах, з яких, згодом, створюється об'єкт типу Mesh, і повертається зухвалому об'єкту функції.

В рамках ітерації заповнюємо структуру усіма відповідними даними. Для вершинних координат це робиться в такий спосіб:

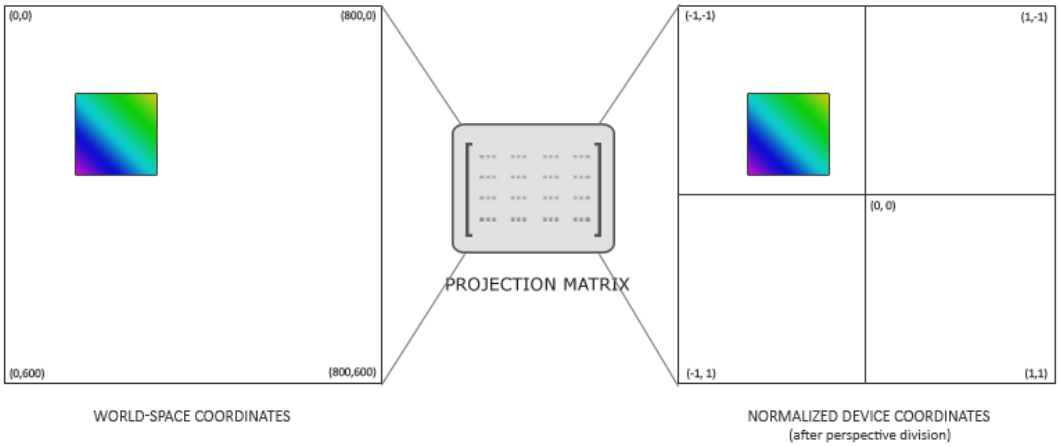
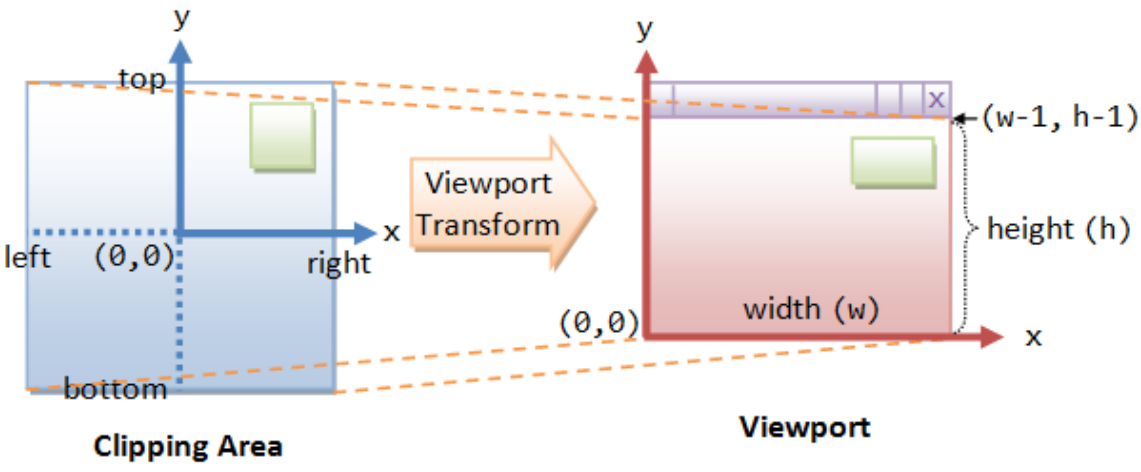


Рисунок 1.12 – Модель функції Vertex

З текстурними координатами відбувається приблизно те ж саме, але при цьому варто зауважити, що Assimp дозволяє моделі мати до 8 різних координат текстури на вершину. Також необхідно перевірити, чи дійсно меш містить текстурні координати (їх може і не бути зовсім):



Clipping Area and Viewport: Objects will be distorted if the aspect ratios of the clipping area and viewport are different.

Рисунок 1.13 – Модель функції Vertex

Інтерфейс Assimp визначає кожен меш як об'єкт, що містить масив граней, при цьому кожна грань являє собою окремий примітив, який, в нашому випадку (через опції aiProcess_Triangulate), завжди є трикутником. Грань містить індекси вершин примітиву в тому порядку, в якому повинні їх малювати. Так що, якщо переберемо всі грані і збережемо індекси в векторі indices, то все буде готово:

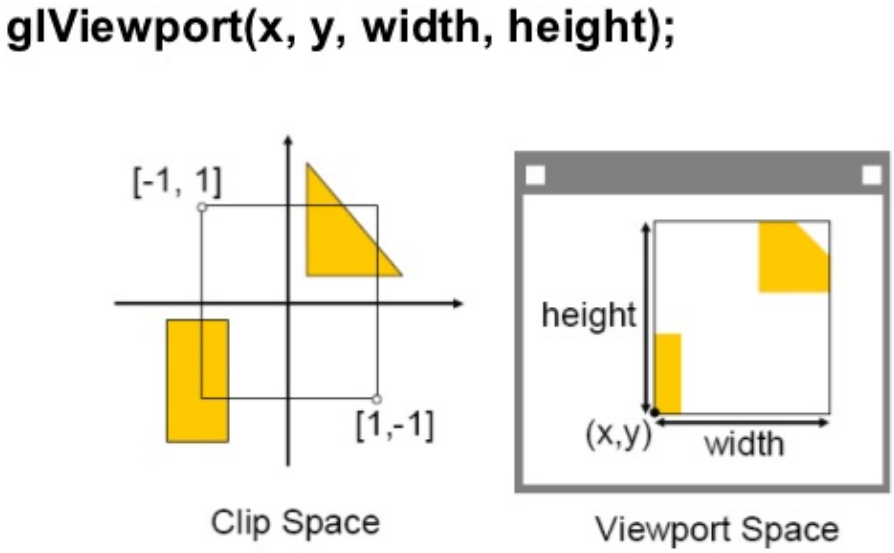


Рисунок 1.14 – Інтерфейс Assimp

Тепер, після завершення зовнішнього циклу for, будемо мати повний набір вершин і індексів, необхідний для відтворення за допомогою функції glDrawElements ().

Подібно вузлів, меш містить тільки індекс об'єкта матеріалу. Щоб отримати сам матеріал, потрібно проіндексовати масив mMaterials сцени.

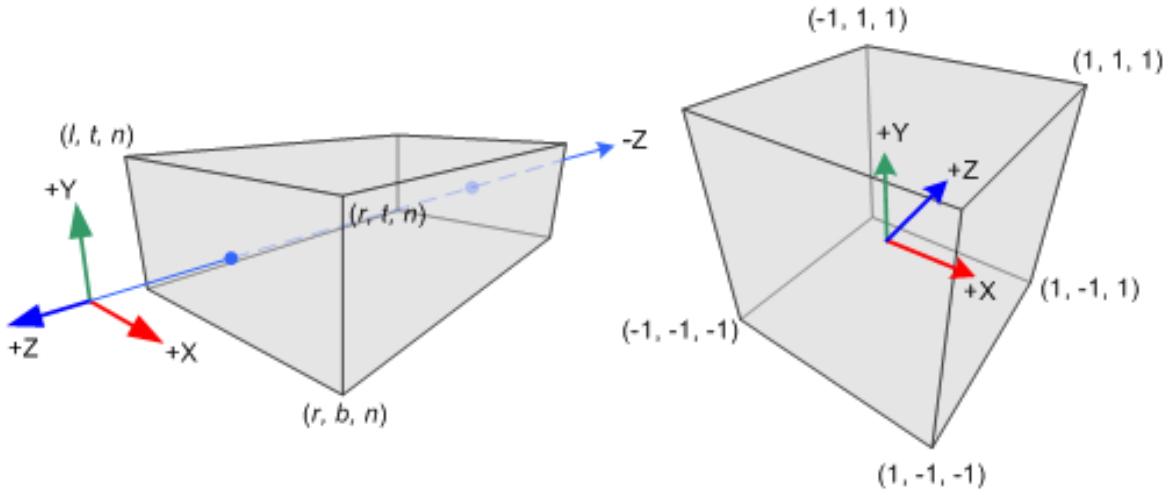


Рисунок 1.15 – mMaterials сцени

Спочатку витягуємо об'єкт aiMaterial з масиву mMaterials сцени. Потім завантажуємо дифузні і / або дзеркальні текстури. Об'єкт матеріалу зберігає в собі масив розташування текстури для кожного типу текстури. Всі різні типи текстур мають префікс aiTextureType_. Для вилучення, завантаження і ініціалізації текстури з матеріалу використовуємо допоміжну функцію loadMaterialTextures (). Вона повертає вектор, що містить елементи у вигляді структур Textures, які будуть збережені в кінці вектора моделі textures.

Функція loadMaterialTextures () виконує ітерацію по всіх локаціях текстур заданого типу, витягує розташування файлу текстури, а потім завантажує і генерує текстуру і зберігає інформацію в структурі Vertex. Це виглядає наступним чином:

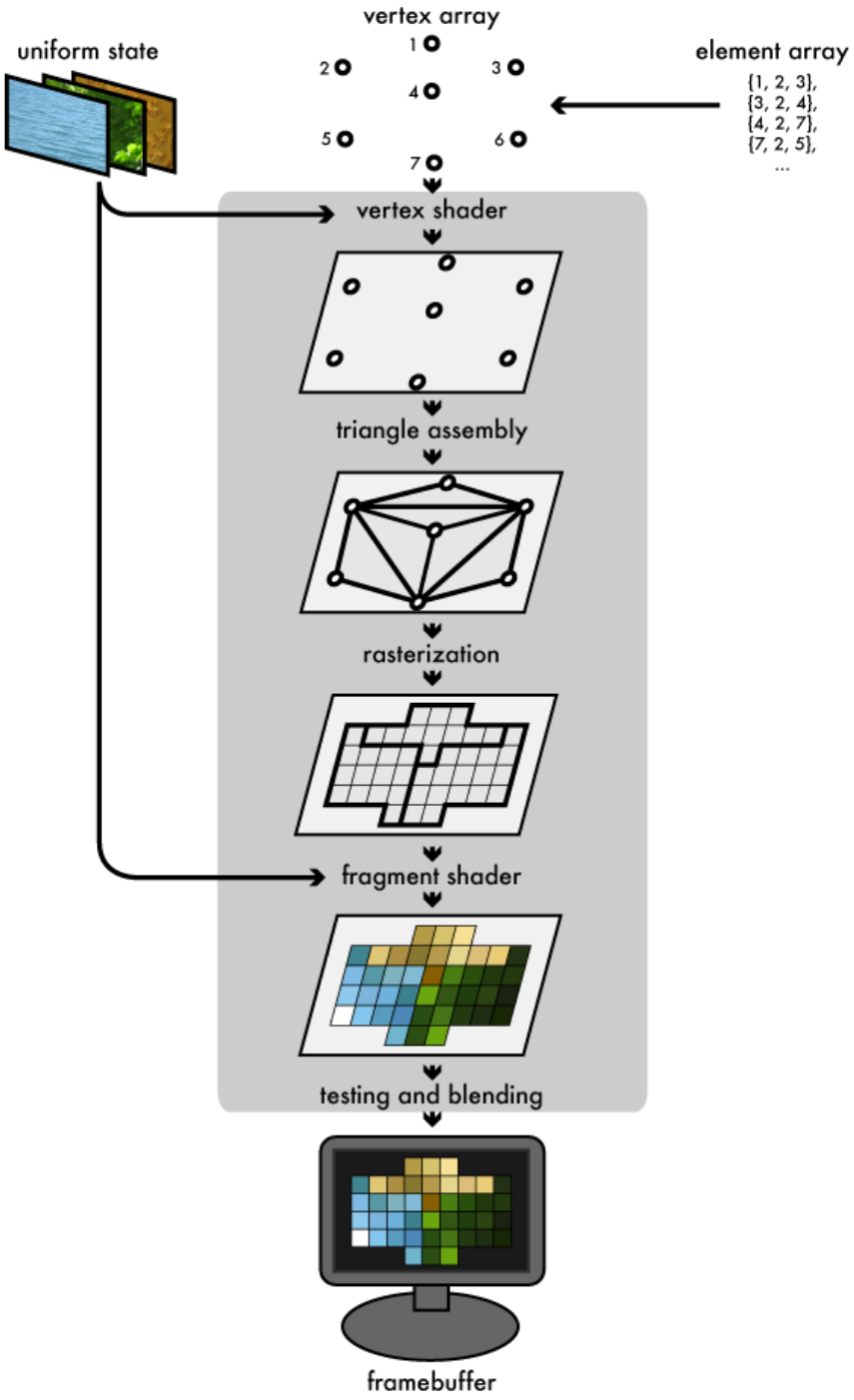


Рисунок 1.16 – Функція loadMaterialTextures ()

Спочатку перевіряємо кількість текстур, що зберігаються в матеріалі, за допомогою функції GetTextureCount (), яка очікує отримати в якості параметра один з переданих типів текстур. Далі витягуємо кожне розташування файлу текстури за допомогою функції GetTexture (), яка зберігає результат у змінній типу aiString. Потім використовуємо іншу допоміжну функцію TextureFromFile

(), яка завантажує текстуру (за допомогою заголовки stb_image.h) і повертає її ідентифікатор.

Потім необхідно зберегти всі завантажені текстури в векторі, оголошеному у верхній частині файлу класу моделі в якості private-змінної:

vector<Texture> textures_loaded;

vector<Texture> textures_loaded;

У функції loadMaterialTextures ()порівнюємо шлях завантажується текстури з усіма текстурами з вектора textures_loaded. Якщо є збіг, то пропускаємо частина коду завантаження / генерації текстури і просто використовуємо знайдену текстурну структуру в якості текстури заважав.

Повний вихідний код класу представлено в Додатку А.

1.3 Результати використання розроблених структур даних

На основі розробленої структури даних реалізовано наступну функціональність.

Процедура вводу даних для побудови відбувається трьома можливими способами:

1. Заповнення панелі вхідних геометричних даних (Рисунок 1.17) на базі яких будуть побудовані гвинтові перетини. Усі введені геометричні данні будуть збережені у вигляді record типу конфігурації та комплексно перевірені на відповідність до їх можливих мінімальних та максимальних значень. У випадку коли усі данні є коректними вони будуть збережені у базі даних (class Propeller) та використані для подальших розрахунків. Отриманні після розрахунків перетини зберігаються у базі даних підкласу TBladeSectionTable у класі Propeller. За допомогою цих перетинів буде відбуватися сама побудова тривимірної полігональної моделі гвинта.

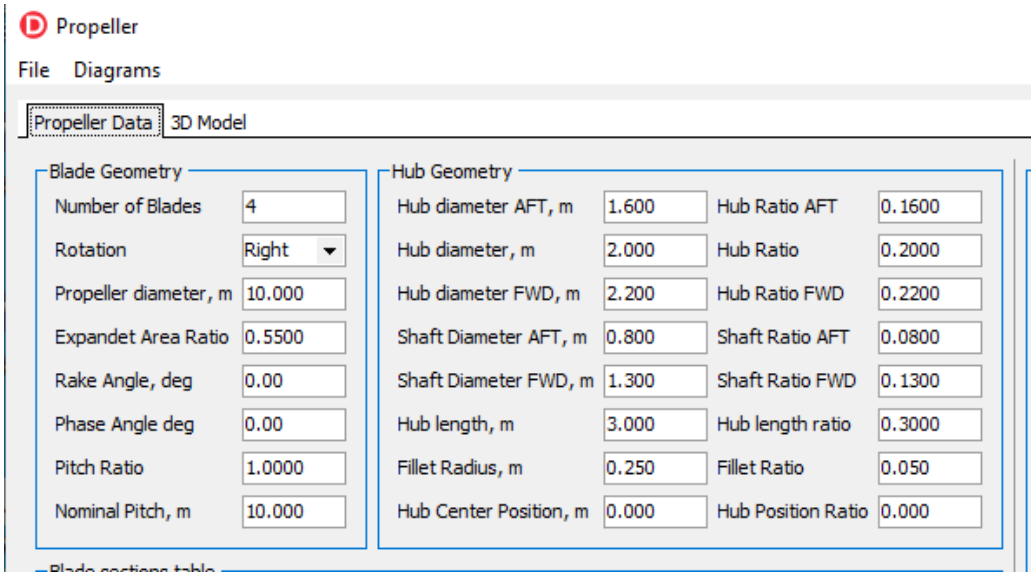
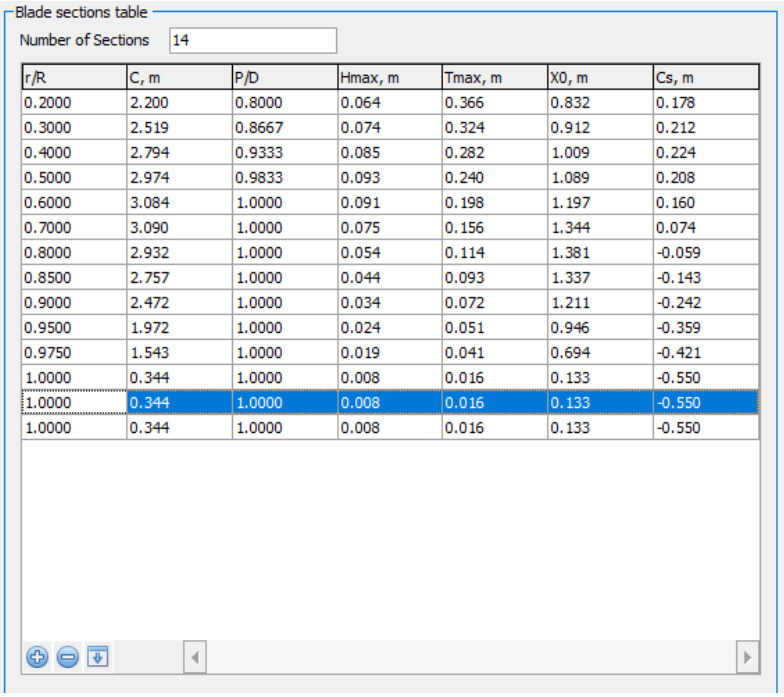


Рисунок 1.17 – Панель вхідних даних геометрії гвинта

2. Пряме перенесення даних з креслення гвинта у таблицю перетинів яка знаходиться нижче панелі вхідних даних геометрії гвинта (Рисунок 1.18).

3. Імпорт збережених раніше даних геометричних характеристик гвинта у форматі XML які одразу заповнять необхідні поля та таблицю та після перевірки на коректність цих даних будуть збережені у класі Propeller для подальших розрахунків.



r/R	C, m	P/D	Hmax, m	Tmax, m	X0, m	Cs, m
0.2000	2.200	0.8000	0.064	0.366	0.832	0.178
0.3000	2.519	0.8667	0.074	0.324	0.912	0.212
0.4000	2.794	0.9333	0.085	0.282	1.009	0.224
0.5000	2.974	0.9833	0.093	0.240	1.089	0.208
0.6000	3.084	1.0000	0.091	0.198	1.197	0.160
0.7000	3.090	1.0000	0.075	0.156	1.344	0.074
0.8000	2.932	1.0000	0.054	0.114	1.381	-0.059
0.8500	2.757	1.0000	0.044	0.093	1.337	-0.143
0.9000	2.472	1.0000	0.034	0.072	1.211	-0.242
0.9500	1.972	1.0000	0.024	0.051	0.946	-0.359
0.9750	1.543	1.0000	0.019	0.041	0.694	-0.421
1.0000	0.344	1.0000	0.008	0.016	0.133	-0.550
1.0000	0.344	1.0000	0.008	0.016	0.133	-0.550
1.0000	0.344	1.0000	0.008	0.016	0.133	-0.550

Рисунок 1.18 – Панель табличних характеристик геометрії гвинта

Для створення нової моделі гвинту використано процедуру задання характеристик.

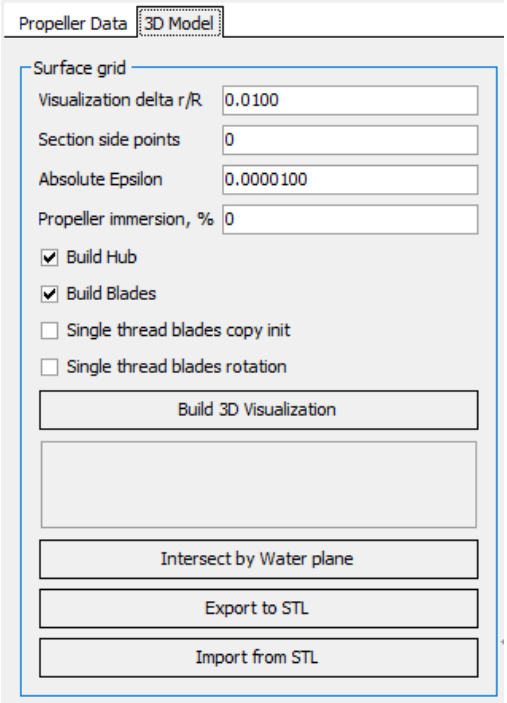


Рисунок 1.19 – Характеристики моделі об’єкта

Обираємо характеристики виводу тривимірної візуалізації на екран, є можливість задати деталізованість моделі (за рахунок введення необхідної кількості полігонів на половину перетину). Була додана функція відключення побудови маточини якщо потрібно розглянути кріплення лопаті до неї, та функція відключення побудови лопатей якщо потрібно детально звернути увагу на маточину.

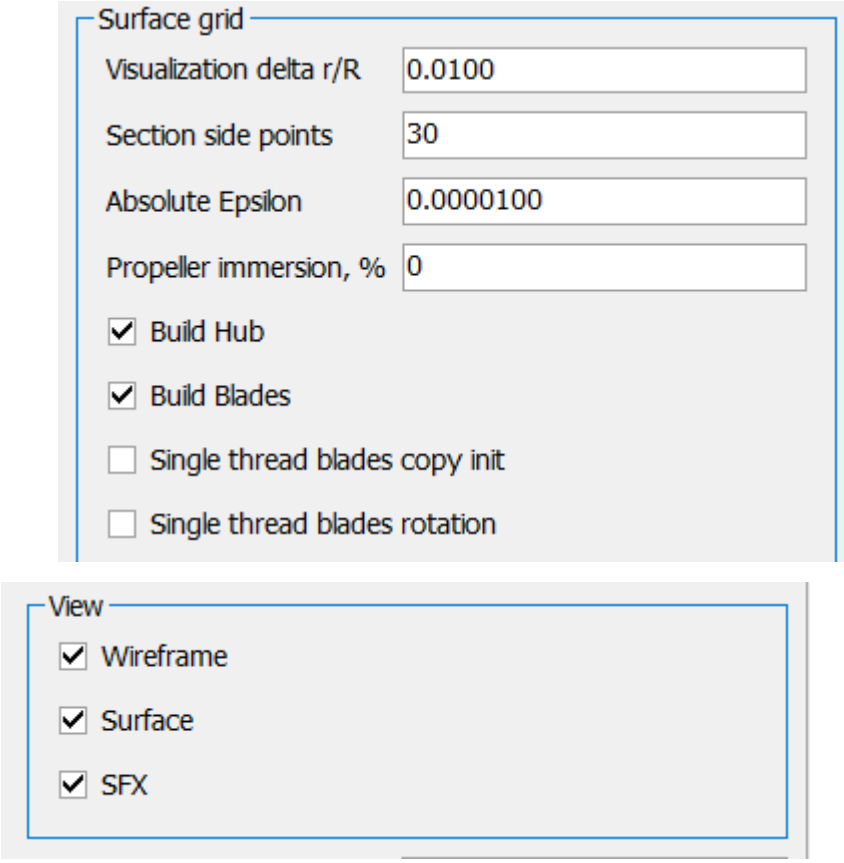


Рисунок 1.20 – Характеристики відображення

Після побудови, можна приховати полігональну сітку, полігони та візуальні ефекти якщо це потрібно для нагального розгляду (Рисунок 1.21 та Рисунок 1.22).

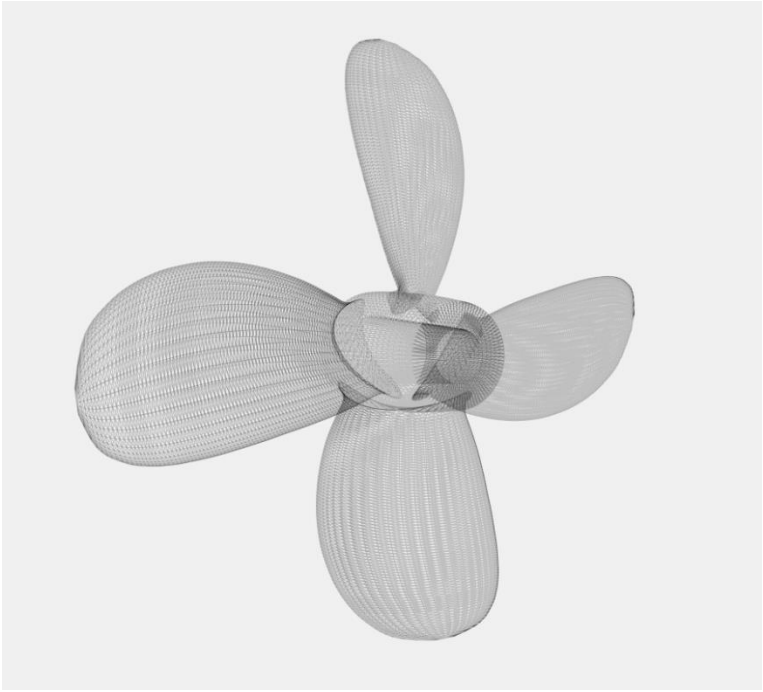


Рисунок 1.21 – Вид полігональної полігональної сітки моделі

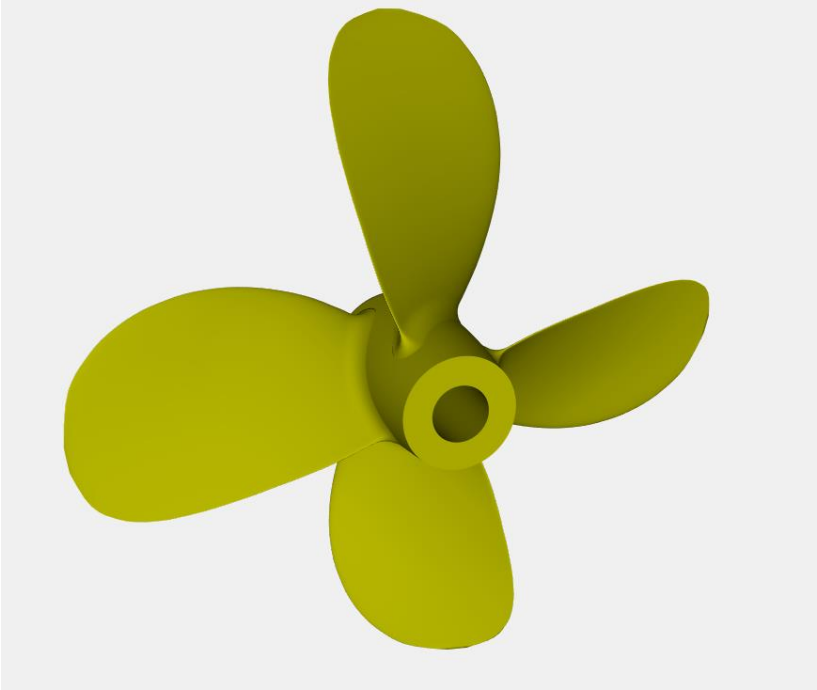


Рисунок 1.22 –Вид моделі без сітки та ефектів

2. Обчислюємо характеристики геометрії гвинта за допомогою формул аналізу побудованої моделі та виводимо їх до користувача (Рисунок 1.23).

Calculated Geometry Characteristics	
Volume, m3	12.4842
Centroid X, m	0.198
Centroid Y, m	0.000
Centroid Z, m	0.000
Mass, kg	12.4842
Moment Inert YZ, kg m2	5.6283
Moment Inert XZ, kg m2	21.7374
Moment Inert XY, kg m2	21.7374
Moment Inert X, kg m2	43.4749
Moment Inert Y, kg m2	27.3658
Moment Inert Z, kg m2	27.3658
Statistics	

Рисунок 1.23 – Обчислені характеристики гвинта

Модель гвинта була побудована коректно и виведені характеристики відповідають табличним значенням.

РОЗДІЛ 2 ЗАСТОСУВАННЯ БАГАТОПОТОЧНОЇ ОБРОБКИ ДАНИХ

2.1Сучасні методи багатопоточної обробки даних

Паралельні обчислення – спосіб комп’ютерних обчислень, при організації якого програми розробляються як набір взаємодіючих обчислювальних процесів, що працюють паралельно (одночасно). Термін охоплює сукупність питань паралелізму в програмуванні, а також створення ефективно діючих апаратних реалізацій.

А розподілені обчислення – це спосіб розв’язання трудомістких обчислювальних задач із використанням декількох комп’ютерів, найчастіше об’єднаних у паралельну обчислювальну систему, у тому числі й за допомогою звичайної локальної обчислювальної мережі. Розподілені обчислення застосовані також у розподілених системах керування.

Іншими словами, розподілені обчислення обов’язково передбачають виконання окремих частин загальної задачі на окремих процесорах або комп’ютерах, тим чи іншим чином з’єднаних між собою. Паралельні ж обчислення можуть бути виконані як на багатопроцесорних системах, так і на комп’ютерах з одиночним процесором, виконуючись у режимі розподілу часу для окремих потоків одного процесу.

Але, не дивлячись на відмінності, ці два види обчислень поєднує та обставина, що в обох випадках загальна задача повинна бути розпаралелена для виконання в окремих потоках на одному процесорі або на ансамблі паралельно працюючих процесорів.

У цей час існують два основних підходи до розпаралелювання обчислень.

Це паралелізм даних і паралелізм задач.

При здійсненні паралелізму даних одна операція виконується відразу над всіма елементами цілого масиву даних. При цьому різні фрагменти

такого масиву обробляються на різних процесорах паралельної машини однаковою послідовністю операторів програми. У такій парадигмі програмування бажано забезпечити автоматичний розподіл даних між процесорами, тому ця задача покладається на програму. У цьому випадку за звичаєм один з процесорів виконує роботу з розподілу даних рівними фрагментами між іншими процесорами системи. Векторизація або розпаралелювання – розподіл того самого виконуваного коду по процесорах у цьому випадку найчастіше виконуються вже під час трансляції, тобто перекладу вихідного тексту програми в машинні команди. Роль програміста при створенні таких програм за звичаєм мінімальна й зводиться до завдання опцій паралельної оптимізації компілятора, директив паралельної компіляції, використання спеціалізованих мов для паралельних обчислень. Навіть при програмуванні складних обчислювальних алгоритмів можна використовувати бібліотеки підпрограм, спеціально розроблених і оптимізованих для конкретної архітектури комп’ютера. В останньому випадку програміст навіть може нічого й не знати про те, що програма виконується паралельно, прикладом таких бібліотек програм може служити добре відомий пакет математичного моделювання MATLAB.

Мова реляційних баз даних SQL (Structured Query Language) є мовою з неявною паралельністю, яка може використовуватися відповідним компілятором у розподіленій системі баз даних, не потребуючи для цього спеціальних мовних конструктивів. Крім того, більшість транзакцій SQL-програм обробляється різними користувачами паралельно або з рознесенням часових інтервалів їхнього виконання. SQL-інструкції можуть використовуватись безпосередньо (інтерактивним чином) у базі даних або інтегруватися в одну з паралельних мов програмування у вигляді вбудованого блоку SQL.

Основними особливостями розглянутого підходу є те, що обробкою даних керує одна програма:

– простір імен є глобальним, тобто для програміста існує одна єдина пам’ять, деталі структури даних, доступу до пам’яті й межпроцесорного обміну даними від нього сховані;

– слабка синхронізація обчислень на паралельних процесорах, тобто виконання команд на різних процесорах відбувається, як правило, незалежно й тільки іноді виробляється узгодження виконання циклів або інших програмних конструкцій – їхня синхронізація. Кожен процесор виконує той же фрагмент програми, але немає гарантії, що в заданий момент часу на всіх процесорах виконується та сама машинна команда;

– паралельні операції над елементами масиву виконуються одночасно на всіх доступних даній програмі процесорах.

Підхід, заснований на паралелізмі даних, базується на використанні в програмах базового набору операцій:

– операції керування даними – у певних ситуаціях виникає необхідність у керуванні розподілом даних між процесорами, це може знадобитися, наприклад, для забезпечення рівномірного завантаження процесорів – чим більш рівномірно завантажені роботою процесори, тим ефективнішою буде робота комп’ютера;

– операції над масивами в цілому, а також над їхніми фрагментами – аргументами цих операцій є масиви в цілому або їхні фрагменти (перетини), при цьому одна операція застосовується одночасно до всіх елементів масиву або до його частини, наприклад, операціями такого типу є операції множення елементів масиву на скалярний або векторний множник, можливі здійснення й складніших дій – обчислення функції від масиву;

– умовні операції – вони застосовуються лише до тих елементів масиву, які задовольняють певну умову, наприклад, у сіткових методах це можуть бути елементи, що відповідають парним/непарним номерам рядків або стовпців сітки;

– операції приведення – застосовуються до всіх елементів масиву

(або до його частки), а результатом є одне-єдине значення, прикладами операції приведення є операції обчислення суми елементів масиву або максимального значення його елементів;

− операції зсуву – операції зсуву масивів потрібні для ефективної реалізації деяких паралельних алгоритмів, наприклад, зсуви часто використовуються в алгоритмах обробки зображень, у кінцево-різницевих алгоритмах іт.ін.;

− операції сканування – такі операції також називаються префіксними/суфіксними операціями, наприклад, префіксна операція підсумовування виконується в такий спосіб – елементи масиву підсумовуються послідовно, а результат чергового підсумовування заноситься в чергову комірку нового, результуючого масиву, причому номер цієї комірки збігається із числом підсумованих елементів вихідного масиву;

− операції, пов’язані з пересиланням даних – операції можуть здійснюватися, наприклад, між масивами різної форми, які мають різну розмірність, довжину по кожному виміру та ін.

Реалізація моделі паралелізму даних потребує підтримки паралелізму на рівні транслятора. Таку підтримку можуть забезпечувати:

− препроцесори, що використовують існуючі послідовні транслятори й спеціалізовані бібліотеки, з реалізаціями паралельних алгоритмічних конструкцій;

− перед транслятори, які виконують попередній аналіз логічної структури програми, перевірку залежностей і обмежену паралельну оптимізацію;

− транслятори з розпаралелюванням, які виявляють паралелізм у вихідному коді програми й виконують його перетворення в паралельні конструкції. Для спрощення перетворення у вихідний текст програми можуть додаватися спеціальні директиви трансляції.

Навпаки, стиль програмування, оснований на паралелізмі задач,

передбачає, що обчислювальна задача розбивається на декілька незалежних одна від одної під задач, і кожен процесор завантажується своєю власною під задачею. При цьому одночасно виконується множина різних операцій над множиною, загалом кажучи, різних і різнотипних даних. Для кожної під задачі пишеться своя власна програма звичайною мовою програмування. Важливо те, що всі ці

програми обмінюються результатами своєї роботи й такий обмін здійснюється викликом процедур спеціалізованої бібліотеки. Програміст контролює розподіл даних між процесорами і обмін даними. У порівнянні з підходом, основаним на паралелізмі даних, такий підхід більш трудомісткий, з ним зв’язана низка таких проблем, як: підвищена трудомісткість налагодження програми, необхідність забезпечення рівномірного завантаження процесорів, мінімізація обміну даними між задачами, можливість виникнення тупикових ситуацій, коли відправлена одною програмою посилка з даними не доходить до місця призначення.

Особливостями цього підходу є більша гнучкість і більша свобода, яка надається програмістові в розробці програми, яка ефективно використовує ресурси паралельного комп’ютера й, як наслідок, є можливість досягнення максимальної швидкодії.

В обох підходах, при складанні паралельних алгоритмів роботи, необхідно вирішити дві основні проблеми:

- забезпечення рівномірного завантаження процесорів;
- забезпечення деякої мінімально необхідної швидкості обміну інформацією.

Перша проблема пов’язана з тим, що чим більш нерівномірно виконано розподіл задачі по процесорах/потоках, тим далі відходимо від паралельних обчислень. Так, якщо з десяти процесорів завантажений тільки один, то утворюється ефект звичайних послідовних обчислень. Таким чином, передбачається, що всі процесори повинні виконувати приблизно однакову

кількість обчислень над однаковими обсягами даних. Неоднорідність процесорів в обчислювальній системі додає труднощів при розпаралелюванні задачі, тому що процесори працюють із різною швидкістю й швидкісні процесори будуть простоювати, очікуючи поки найповільніший з процесорів виконає свою частку обчислень.

Друга не менш важлива проблема, полягає в тому, що при низькій швидкості обміну між процесорами основна частина часу буде витрачатися на очікування процесорами необхідної інформації. Звідси витікає той висновок, що в розподілених системах дуже велике значення має пропускна здатність мережі міжз'єднань і швидкодія окремих процесорів у таких системах зумовлюється швидкістю передачі даних.

Паралельна обробка даних, втілюючи ідею одночасного виконання декількох дій, має два різновиди: конвеєрність і власне паралельність. Обидва види паралельної обробки, загалом кажучи, не потребують яких-небудь особливих пояснень, але коротко про них можна сказати таке.

Паралельна обробка. Якщо якийсь пристрій виконує одну операцію за одиницю часу, то тисячу операцій він виконає за тисячу одиниць. Якщо припустити, що є п'ять саме таких незалежних пристроїв, здатних працювати одночасно й незалежно, то ту ж тисячу операцій система з п'яти пристроїв може виконати вже не за тисячу, а за двісті одиниць часу. У загальному випадку, якщо однопроцесорнийпристрійвиконуєдеякуроботузачасtоп,тосистемазnпристроїв ,найімовірніше,тужроботувиконаєприблизнозат$_{мп}$=t$_{оп}$/nодиницьчасу, тобто буде отримане прискорення роботи в n раз. Пізніше буде показано, що це співвідношення найчастіше не виконується, але в цілому ряді випадків цілком є справедливим для систем паралельної обробки даних.

Конвеєрна обробка. Для додавання двох чисел дійсного типу, представлених у формі із плаваючою комою, необхідно виконати велику кількість дрібних операцій, таких як порівняння порядків, вирівнювання порядків, додавання мантис, нормалізація й та інше. Процесори перших

комп’ютерів виконували всі ці «мікрооперації» для кожної пари аргументів послідовно одна за одною доти, поки не доходили до остаточного результату, і лише після цього переходили до обробки такої пари доданків.

Ідея конвеєрної обробки полягає у виділенні окремих етапів виконання загальної операції, причому кожен етап, виконавши свою частку роботи, передає результат такому й одночасно приймає нову порцію вхідних даних. Таким чином одержується очевидний виграш у швидкості обробки за рахунок суміщення раніше рознесених у часі операцій. Припустимо, що в операції можна виділити п’ять мікрооперацій, кожна з яких виконується за одиницю часу. Якщо є один неподільний послідовний пристрій, то 100 пар аргументів він обробить за 500 одиниць часу. Якщо ж кожну мікрооперацію виділити в окремий етап (або інакше кажучи – ступінь) конвеєрного пристрою, то на п’ятій одиниці часу на різній стадії обробки такого пристрою будуть перебувати перші п’ять пар аргументів. Перший результат, таким чином, визначиться через 5 одиниць часу, кожен такий – через одну одиницю після попереднього. Весь же набір зі ста пар буде оброблений за 5 + 99 = 104 одиниці часу, тобто буде отримане прискорення в порівнянні з послідовним пристроєм майже в п’ять разів (за кількістю ступенів конвеєра). Те ж саме можна сказати і у загальному випадку.

Якщо деяка операція містить n незалежних ступенів, які виконуються послідовно один за одним, кожен ступінь спрацьовує за одиницю часу й потрібно виконати m (m набагато більше за n) таких складених операцій, то час роботи одно процесорного пристрою складе:

$$t_{оп} = m \times n.$$

Тобто у результаті маємо прискорення майже в n раз за рахунок використання конвеєрної обробки даних.

Здавалося б, конвеєрну обробку можна з успіхом замінити звичайним паралелізмом, для чого достатньо просто скопіювати основний пристрій

стільки разів, скільки ступенів конвеєра передбачається виділити. Однак вартість і складність отриманої системи буде непорівнянна з вартістю й складністю конвеєрного варіанта, а продуктивність буде майже такою.

Зі сказаного вище можна зробити висновок, що розпаралелювання програм є доволі складним процесом. Тому перш ніж приступити безпосередньо до складання й опису деяких паралельних обчислювальних алгоритмів, необхідно розглянути загальні принципи розв'язання задачі розпаралелювання.

2.2 Організація паралельних обчислень з використанням наявних технологій (PVM,MPI)

Розробка алгоритмів (а особливо методів паралельних обчислень) для розв'язання складних науково-технічних задач часто є доволі значною проблемою. Дії для визначення ефективних способів організації паралельних обчислень можуть полягати в такому (Рисунок 2.1):

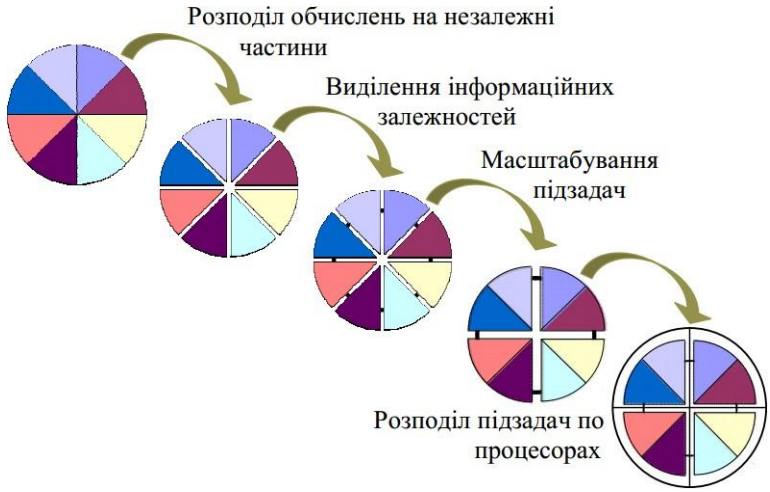


Рисунок 2.1 − Загальна схема розробки паралельних алгоритмів

− виконати аналіз наявних обчислювальних схем і здійснити їхній поділ (декомпозицію) на частини (підзадачі), які можна реалізувати значною мірою

незалежно одна від одної;

− виділити для сформованого набору підзадач інформаційні взаємодії, які повинні здійснюватися в ході розв’язання вихідної поставленої задачі;

− визначити необхідну (або доступну) для розв’язання задачі обчислювальну систему й виконати розподіл отриманого набору підзадач між процесорами системи.

При найбільш загальному розгляді є цілком очевидним, що обсяг обчислень для кожного використаного в системі процесора повинен бути приблизно однаковим – це дозволить забезпечити рівномірне обчислювальне завантаження (балансування) процесорів. Також зрозуміло те, що розподіл підзадач між процесорами повинен виконуватись таким чином, щоб кількість інформаційних зв’язків (комунікаційних взаємодій) між підзадачами була мінімальною.

Після виконання всіх перерахованих етапів проектування можна оцінити ефективність розроблювальних паралельних методів: для цього за звичаєм обумовлюються значення показників якості породжуваних паралельних обчислень (прискорення, ефективність, масштабованість). За результатами проведеного аналізу може виявитися необхідним повторення окремих (у крайньому випадку всіх) етапів розробки. Слід зауважити, що повернення до попередніх кроків розробки може відбуватися на будь-якій стадії проектування паралельних обчислювальних схем.

Тому часто додатковою дією, яка виконується в наведеній вище схемі проектування, є коректування складу сформованої множини підзадач після визначення наявної кількості процесорів – підзадачі можуть бути укрупнені (агреговані) при наявності малого числа процесорів або навпаки деталізовані у протилежному випадку. В цілому, ці дії можуть визначатись як масштабування розроблювального алгоритму й виділятись як окремий етап проектування паралельних обчислень.

Щоб використати одержуваний в остаточному підсумку паралельний метод, необхідно виконати розробку програм для розв’язання сформованого

набору підзадач і розмістити розроблені програми по процесорах відповідно до обраної схеми розподілу підзадач. Для проведення обчислень програми запускаються на виконання (програми на стадії виконання зазвичай йменуються процесами). Для реалізації інформаційних взаємодій програми повинні мати у своєму розпорядженні засоби обміну даними – канали передачі повідомлень.

Слід зазначити, що найчастіше кожен процесор виділяється для вирішення єдиної підзадачі, однак, при наявності великої кількості підзадача бо використанні обмеженого числа процесорів це правило може не дотримуватися й, у наслідку, на процесорах може виконуватись одночасно кілька програм (процесів). Зокрема, при розробці й початковій перевірці паралельної програми для виконання всіх процесів може використовуватися лише один процесор (при розташуванні на одному процесорі процеси виконуються в режимі поділу часу).Уважно розглядаючи розроблену схему проектування й реалізації паралельних обчислень, можна відзначити, що запропонований підхід значній мірі орієнтований на обчислювальні системи з розподіленою пам’яттю, коли необхідні інформаційні взаємодії реалізуються за допомогою передачі повідомлень по каналах зв’язку між процесорами. Проте, ця схема може застосовуватись без втрати ефективності паралельних обчислень і для розробки паралельних методів для систем з поділюваною пам’яттю – у цьому випадку механізми передачі повідомлень для забезпечення інформаційних взаємодій повинні замінюватись операціями доступу до спільних (поділюваних) змінних.

Станом на сьогодні розроблена безліч мов і систем програмування, як для паралельних, так і для розподілених обчислювальних систем. Прикладами таких систем є OpenMP, [p]threads, shmem (для систем з поділюваною пам’яттю), Occam, Concurrent C, HPF (високопродуктивний ФОРТРАН), HPC++, Distributed Java (для систем з розподіленою пам’яттю), Parlog, Concurrent Prolog (паралельні версії мови ПРОЛОГ), Multilisp (паралельна версія мови ЛІСП).

Дуже часто для паралельного програмування використовуються дві такі

системи бібліотечних функцій: Message Passing Interface (MPI) – Інтерфейс Передачі Повідомлень і Parallel Virtual Machine (PVM) – Паралельна Віртуальна Машина. Ці системи програмування призначені для роботи на розподілених системах, але вони можуть використовуватися й для систем з розподілюваною пам’яттю.

Обидві системи реалізують модель передачі повідомлень і містять бібліотеки функцій і підпрограм для стандартних мов програмування C, C++, Fortran забезпечують взаємодії типу «точка-точка» і групові. У той же час системи мають і істотні відмінності.

Центральним елементом системи розробки програм у проекті PVM є «віртуальна машина» – набір різнорідних обчислювальних вузлів, який з точки зору користувача є одним великим паралельним комп’ютером. Одним з нюансів реалізації цієї паралельної машини є збереження простоти способу обміну даними між задачами й універсальність інтерфейсу у подальшому розвитку проекту.

На відміну від PVM, який був зароджений й по сьогодні розвивається в рамках дослідницького середовища, API MPI-1 був створений провідними експертами в індустрії комп’ютерів і програмного забезпечення в 1993-1994 роках. Необхідність розробки MPI обумовлена потребою стандартизувати уніфікувати API передачі повідомлень тому, що раніш кожен виробник багатопроцесорних обчислювальних комплексів пропонував свій API для обміну повідомленнями. Таким чином, MPI був прийнятий як стандарт API для обміну повідомленнями.

І PVM, і MPI надають можливість логічного з’єднання машин у єдину обчислювальну систему. І PVM, і MPI від самого народження містять велику кількість бібліотек для розробки паралельних програм мовами Fortran і C. На сьогодні ж розроблені бібліотеки класів для мов Java (кілька різних бібліотек від різних розроблювачів), Python, C# й для багатьох інших. Обидві системи реалізують модель передачі повідомлень.

Якщо застосування, яке розробляється для багатопроцесорного

обчислювального комплексу, буде запускатися лише на одній архітектурі й не потребує значної переносимості, то використання MPI може принести значний виграш у швидкодії. Система містить значно багатіші засоби комунікації, тому її застосування є кращим в обчислювальних системах, де використовуються спеціальні режими комунікації, недоступні в PVM. Але з метою забезпечення високої швидкості взаємодії на MPI були накладені обмеження. Найбільш значимими з них є відсутність сумісності – різні реалізації MPI не можуть взаємодіяти між собою і відсутність можливості писати стійкі до відмови застосування за допомогою MPI. Правда стандарт MPI-2 і всі пізніші дозволяють створювати стійкі до відмови застосування, але, на жаль, поки не всі розроблювачі підтримують цю властивість другого стандарту MPI.

Через те, що PVM реалізує поняття «віртуальної машини» , ця система має переваги при виконанні обчислень у системі, яка складається з неоднорідних вузлів. PVM містить системи керування ресурсами й процесами, які важливі для написання застосувань, що запускаються як на кластерах робочих станцій, так і на багатопроцесорних обчислювальних комплексах. Чим більше вузлів у кластері, тим більш важливим стає стійкість PVM до відмови. Можливість написання програм, які успішно працюють навіть під час відмови обчислювальних вузлів, дуже важлива для розподілених обчислень у неоднорідному середовищі.

Вибір найбільш зручного API (MPI або PVM) залежить від структури обчислювальної системи й задач, які потрібно вирішити.

Паралельна Віртуальна Машина – PVM просто є пакетом програм, який дозволяє використовувати зв’язаний у локальну мережу набір різнорідних комп’ютерів, працюючих під керуванням Unix-сумісної операційної системи (існують системи й для Windows, але вони мають дуже обмежене застосування), як один великий паралельний комп’ютер. Таким чином, проблема великих обчислень досить ефективно вирішується за рахунок використання сукупної потужності й пам’яті великої кількості комп’ютерів. Пакет програм PVM легко переноситься на будь-яку Unix-платформу.

Проект PVM був створений в 1989 році Вайді Сандерманом (VaidySunderman) – професором університету Еморі м. Атланта і лабораторією Oak Ridge National Laboratory. Перший реліз був запущений у березні 1991. Комплекс PVM забезпечує роботу паралельних програм у гетерогенних обчислювальних середовищах. Машина PVM поєднує безліч різнорідних обчислювальних вузлів в один величезний ресурс. У системі можливе створення застосувань як типу SPMD (одна паралельна програма, багато потоків даних), так і типу MPMD (багато паралельних програм, багато потоків даних). Вона сумісна із застосуваннями, написаними на різних мовах програмування (наприклад, C, C++, Fortran) і коректно перетворює дані при передачі їх між вузлами різної архітектури. PVM від створення є відкритою й вільно розповсюджуваною системою.

Паралельну віртуальну машину можна визначити як частину засобів реального обчислювального комплексу (процесори, пам’ять, периферійні пристрої й т.ін.), призначену для виконання множини задач, які беруть участь в одержанні загального результату обчислень. У загальному випадку число задач може перевершувати число процесорів, включених в PVM. Крім того, до складу PVM можна включати досить різнорідні обчислювальні машини, несумісні по системах команд і форматах даних. Інакше кажучи, Паралельною Віртуальною Машиною може стати як окремо взятий ПК, так і локальна мережа, що включає в себе суперкомп’ютери з паралельною архітектурою, універсальні ЕОМ, графічні робочі станції й ті ж самі малопотужні ПК. Важливо лише, щоб про обчислювальні засоби, які включаються в PVM, була інформація у використовуваному програмному забезпеченні PVM. Завдяки цьому програмному забезпеченню користувач може вважати, що він спілкується з одною обчислювальною машиною, у якій можливо паралельне виконання множини задач.

PVM дозволяє користувачам використовувати існуючі апаратні засоби, для розв’язання досить складних задач при мінімальних витратах. Сотні дослідницьких груп в усьому світі використовують PVM, щоб вирішити

важливі наукові, технічні, і медичні проблеми, а так само використовують PVM як освітній інструмент, для викладання паралельного програмування.

Задачі PVM можуть містити структури для забезпечення необхідних рівнів контролю й залежності. Інакше кажучи, у будь-якій «точці» виконання взаємозалежних застосувань будь-яка існуюча задача може запускати або зупиняти інші задачі, додавати або видаляти комп'ютери з віртуальної машини. Кожен процес може взаємодіяти та/або синхронізуватися з будь-яким іншим. Кожна специфічна структура для контролю і залежності може бути реалізована в системі PVM адекватним використанням конструкцій PVM і керуючих конструкцій головної (хост) мови системи.

Маючи таку всеосяжну природу (специфічно для концепції віртуальної машини), а також через свій простий, але функціонально повний програмний інтерфейс, система PVM набула широкого розповсюдження, у тому числі й у науковому співтоваристві, пов'язаному з високошвидкісними обчисленнями.

Система PVM складається із двох частин. Перша частина – це демон, що поміщається на всі комп'ютери, які входять до складу віртуальної машини. (Прикладом програми-демона може бути поштова програма, що виконується у фоновому режимі й обробляє всю вхідну й вихідну електронну пошту комп'ютера). Демон розроблюється таким чином, щоб будь-який користувач із достовірним логіном міг інсталювати його на машину. Коли користувач бажає запустити розподілене застосування PVM, він, насамперед, створює віртуальну машину, запускаючи PVM. Після цього застосування PVM можна запустити з будь-якого Unix-термінала на кожному з хостів. Декілька користувачів можуть конфігурувати віртуальні машини, які перекриваються, кожен користувач може послідовно запустити кілька застосувань PVM.

Друга частина системи – це бібліотека підпрограм інтерфейсу PVM. Вона містить функціонально повний набір примітивів, необхідних для взаємодії між задачами застосування. Ця бібліотека містить викликувані користувачем підпрограми для обміну повідомленнями, породження процесів, координування задач і модифікації віртуальної машини.

Функціонування PVM ґрунтується на механізмах обміну інформацією між задачами, виконуваними в її середовищі. Щодо цього, найбільше зручно реалізовувати PVM у рамках багатопроцесорного обчислювального комплексу, виділивши віртуальній машині кілька процесорів і індивідуальні або загальне ОЗП (залежно від умов). Використання PVM припустиме як на багатопроцесорних комп'ютерах (SMP) так і на обчислювальних комплексах, побудованих за кластерною технологією. При використанні PVM, як правило, значно спрощуються проблеми швидкого інформаційного обміну між задачами, а також проблеми узгодження форматів представлення даних між задачами, виконуваними на різних процесорах.

Ефективне програмування для PVM починається з того, що алгоритм обчислень доцільно адаптувати до складу PVM і до її характеристик. Це досить творча задача, яка у багатьох випадках повинна вирішуватися програмістом. Крім задачі розпаралелювання обчислень виникає необхідність вирішення задачі керування обчислювальним процесом, координації дій задач-учасників цього процесу. Іноді для керування доводиться створювати спеціальну задачу, яка сама не бере участь в обчисленнях, але забезпечує погоджену роботу інших задач-обчислювачів.

В PVM існує два варіанти організації обчислень.

При першому варіанті всі задачі запускаються одною командою, у якій вказується ім'я виконуваного файла, кількість задач, що запускаються, а також кількість й тип використовуваних процесорів. За цією командою на зазначених процесорах запускається необхідна кількість копій зазначеного виконуваного файла. Отже, програмні коди всіх запущених в PVM задач у цьому випадку однакові. Для того щоб ці задачі могли виконувати різні дії, їм повинен бути відомий деякий атрибут, який відрізняє кожну задачу від інших. Тоді, використовуючи цей атрибут в умовних операторах, легко запрограмувати виконання задачами різних дій. Наявність такої ознаки передбачено в будь-який багатозадачній операційній системі. Це так званий ідентифікатор задачі (TID) – ціле число, яке привласнюється задачі при її запуску. Слід зазначити,

що при запуску задача одержує ідентифікатор, відмінний від ідентифікаторів, виконуваних у цей час задач. Це гарантує, що ідентифікатори всіх запущених задач в PVM будуть різними. Якщо тепер забезпечити задачі можливістю визначати власний ідентифікатор і обмінюватися інформацією з іншими задачами, то зрозуміло, що їм легко розподілити між собою обчислювальну роботу, у залежності, наприклад, від займаного місця в упорядкованому наборі ідентифікаторів задач.

Другий варіант на практиці використовується набагато частіше. У ньому спочатку запускається одна задача (master), що у множині задач буде відігравати функції координатора робіт. Ця задача виконує деякі підготовчі дії, після чого запускає інші задачі (slaves), яким може відповідати той самий виконуваний файл або різні виконувані файли. Такому варіанту організації паралельних обчислень віддають перевагу при ускладненні логіки керування обчислювальним процесом, а також в тому випадку, коли алгоритми, реалізовані в різних задачах, істотно різняться або є великий обсяг операцій, які обслуговують обчислювальний процес у цілому (наприклад, операції введення/виведення).

Як вже відзначалось, у системі PVM кожна задача (якій відповідає виконуваний файл), будучи запущеною на деякому процесорі, ідентифікується цілим числом – ідентифікатором задачі (TID). За змістом TID схожий на ідентифікатор процесу в операційній системі Unix. Конкретні значення TID несуттєві, важливо лише, щоб всі задачі, запущені в PVM, мали різні TID. Окремо слід зазначити, що навіть копії того самого виконуваного файла, будучи запущені паралельно на N процесорах PVM, створюють N задач із різними TID.

У моделі програмування, яка підтримується MPI, програма породжує кілька процесів, взаємодіючих між собою за допомогою звертань до підпрограм передачі й прийому повідомлень. За звичаєм, при ініціалізації MPI-програми створюється фіксований набір процесів, причому кожен процес виконується на

своєму процесорі. У цих процесах можуть виконуватись різні програми, тому модель програмування MPI іноді називають MPMD-моделлю (Multiple Program Multiple Data – багато програм, багато даних), на відміну від SPMD-моделі, коли на кожному процесорі виконуються тільки однакові задачі.

Для організації локальних і неструктурованих комунікацій використовуються, так звані, двоточкові обміни даними. При виконанні глобальних операцій застосовуються колективні обміни. Асинхронні комунікації реалізуються за допомогою запитів про одержання повідомлень. Механізм, який має назву комунікатор, приховує від програміста внутрішні комунікаційні структури.

Алгоритми, у яких є фіксована кількість підзадач, допускають пряму реалізацію за допомогою двоточкових і групових обмінів. Алгоритми, у яких кількість процесів змінюється в процесі виконання програми, не можуть бути реалізовані в MPI безпосередньо (але починаючи з специфікації MPI-2 така можливість вже реалізована). У цьому випадку доводиться відразу, у момент запуску застосування, створювати множину процесів зі структурою, у яку вписуються всі можливі конфігурації підзадач, що виникають у процесі виконання програми. Динаміка буде в цьому випадку підтримуватися змінюваною структурою комунікацій. Часто це є можливим.

Специфікація MPI забезпечує переносимість програм на рівні вихідних кодів і велику функціональність. Підтримується робота на гетерогенних кластерах і симетричних багатопроцесорних системах: Не підтримується, як вже відзначалось, запуск процесів під час виконання MPI-програми. У специфікації відсутні описи паралельного введення/виведення й налагодження паралельних програм (починаючи з стандарту MPI-2 уже передбачені й такі операції). Інколи такі можливості включаються до складу конкретної реалізації MPI у вигляді додаткових пакетів і утиліт. Стандартом не гарантується сумісність різних реалізацій MPI.

Важливою властивістю паралельної програми, так само як і послідовної, є її детермінізм – це означає, що програма повинна завжди давати той самий

результат для того самого набору вхідних даних. Модель паралельного програмування з передачею повідомлень, загалом кажучи, цієї властивості не має, оскільки порядок одержання повідомлень від двох процесів третім не визначений. Якщо ж один процес послідовно посилає кілька повідомлень іншому процесу, MPI гарантує, що другий процес одержить їх саме в тім порядку, у якому вони були відправлені. Відповідальність за забезпечення детермінованого виконання програми повністю лягає на програміста.

Однією з найпоширеніших реалізацій специфікації MPI є бібліотека процедур MPICH (MPI CHameleon), яка підтримує роботу на великій кількості платформ і з різними комунікаційними інтерфейсами, у тому числі TCP/IP, і є вільно розповсюджуваним програмним забезпеченням.

Різні версії MPICH (наприклад, така версія як MPICH 1.2.2) характеризуються такими основними особливостями:

- повна сумісність зі специфікаціюMPI-1;
- наявність інтерфейсу в стилі MPI-2 з функціями для мови C++ зі специфікаціїMPI-1;
- наявність інтерфейсу із процедурами мовиFortran-77/90;
- існує реалізація для Windows NT, яка розповсюджується у вихідних текстах. Її встановлення й використання відрізняються від відповідних дій у системах стандарту POSIX;
- підтримка великої кількості архітектур, у тому числі кластерів робочих станцій, симетричних багатопроцесорних систем йт.ін.;
- часткова підтримка специфікації MPI-2;
- часткова підтримка паралельних введення/виведення(ROMIO);
- наявність засобів трасування й протоколювання (на основі масштабованого формату log-файлів SLOG);
- наявність засобів візуалізації продуктивності паралельних програм (up-shot ijumpshot);
- наявність у складі MPICH тестів продуктивності й перевірки функціонування системи.

До числа недоліків MPICH можна віднести неможливість запуску процесів під час виконання програми й відсутність засобів моніторингу за поточним станом обчислювальної системи. У результаті цього, якщо відбувається, наприклад, апаратний збій на одному із процесорів, які беруть участь у виконанні паралельної MPI-програми, її робота завершується аварійно. Неможливість динамічної зміни набору процесів не дозволяє використовувати ресурси обчислювальної системи з максимальною ефективністю.

Якщо при установленні бібліотека MPICH була сконфігурована для роботи на кластері, то навіть при пересиланнях повідомлень на одному комп'ютері буде використовуватися мережний протокол TCP/IP. Це не найефективніше рішення, але воно працює. Якщо ж зборка виконувалась для системи з поділюваною пам'яттю, MPICH-програма не зможе працювати на кластері.

До складу MPICH входять бібліотечні й заголовні файли. MPICH містить більше сотні підпрограм. З пакетом MPICH поставляються засоби візуального налагодження й профілювання паралельних програм. Це jumpshot або її більш стара версія upshot. Ці засоби написані мовою Java і працюють із файлами-протоколами подій (tracefiles) CLOG (jumpshot 2) і SLOG (jumpshot 3). Розмір файла-протоколу в третій версії jumpshot може бути дуже великим, до гібібайта. Існує документація до MPICH у форматах HTML і PostScript.

До складу MPICH включені також приклади програм, які (наприклад, для системи Linux) розташовуються в каталогах:

- mpich/examples/basic/ – демонстрація основних можливостей MPICH;
- mpich/examples/test/ – тестові програми;
- mpich/examples/perftest/ – тестові програми для визначення продуктивності.

Набір прикладів може бути різним у різних версіях MPICH.

Є й комерційні варіанти MPI, які випускаються, як правило, з оптимізацією для конкретної платформи й можуть містити додаткові функції й утиліти. Є варіанти для інших операційних систем. Як приклад можна навести

WMPI – реалізацію MPI для Microsoft Windows.

2.3 Застосування методу багатопоточної обробки даних для обрахування необхідних характеристик та геометрії гвинта

Програма складається із п’яти програмних модулів (рис. 2.2).

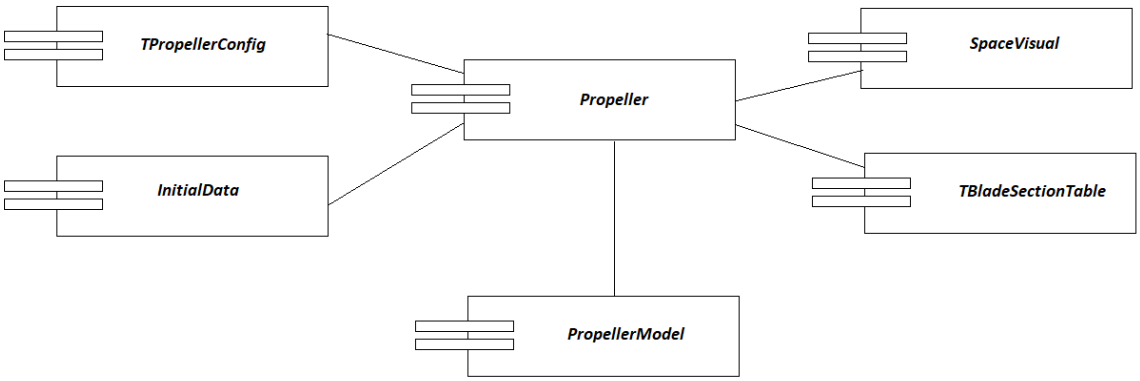


Рисунок 2.2 – Структура модулів програми

Модуль SpaceVisual – модуль зберігання та обробки даних.

Модуль TPropellerConfig – модуль, що відповідає за дані користувача.

Модуль Propeller – головний модуль, база даних геометричних характеристик гвинта.

Модуль PropellerModel – модуль розрахунку геометрії гвинта.

SpeceVisual – модуль тривимірної візуалізації.

TBladeSectionTable – модуль розрахованих характеристик перетинів гвинта.

Технологія багатопоточності використана для модуля SpaceVisual. Проаналізуємо його детальніше.

Модуль SpaceVisual. Цей модуль містить в собі кілька класів, які відповідають за зберігання і обробку даних, необхідних для побудови гвинта, а саме:

- TSpace (TSpaceObject) клас зберігання і обробки всіх об'єктів типу TSpace. Це модуль розрахунків і зберігання всіх даних про характеристики гвинта, його полігонів і точки. Нижче важливі функції цього класу:

- procedure Assign (копіює всі дані в об'єкт такого ж класу);

- procedure CreateTwoTriangles (Прискорене створення двох трикутників-полігонів по 4 точкам);

- procedure CreateTwoTriangles_FastSingleThread (версія попередньої процедури, яка оптимізована під використання багатопоточності);

- procedure MoveFaces (переміщення всіх полігонів з одного об'єкта типу TSpaceBody в другий, оптимізований під використання багатопоточності);

- function CombineTwoBodies (об'єднує два тіла в одне, оптимізований під використання багатопоточності);

- function FaceIndex (прискорене отримання індексу полігону в Space);

- function BodyIndex (прискорене отримання індексу тіла в Space);

- TSpaceBody клас для зберігання даних про тіло, зберігає в собі список всіх полігонів моделі;

- function CloneStructure (розділення на потоки);

- TSpaceFace - клас для зберігання даних про полігон;

- TSpaceNode - клас для зберігання даних про точку.

Алгоритм роботи програми наступний.

Заповнюються поля введення даних геометрії гвинта на інтерфейсі. При натисканні кнопки Build всі ці дані заповнюють record типу TPropellerConfig і відправляються на комплексну перевірку функцією CheckPropellerDimentionlessConsistency на коректність введених даних в модуль введення Configure який прописаний в основному модулі Propeller.

При успішному проході перевірки дані в модулі Configuration, який відповідає за збереження геометричних даних, замінюються на введені і запускається наступний етап алгоритму.

Використовується структура з модуля TSpaceOfNodes. Створюється новий об'єкт типу TSpace. Побудова починається з полігональної структури, для цього використовуються введені дані і здійснюється розрахунок функцією CreateBodyHub з модуля TPropeller. На основі розрахунків ми отримуємо величезну кількість точок (об'єкти типу TSpaceNode), які повинні розподілиться

по правильним полігонів (об'єктів типу TSpaceFace). На виході, результатом функції буде об'єкт типу TSpaceBody в якому буде зберігається все дані по моделі.

Далі запускається етап алгоритму, починається побудова лопатей функцією CreateBodyWithAllBlades. Перша лопать будується функцією CreateBasicBlade алгоритм якої:

- Створюємо нову таблицю TBladeSectionTable (модуль зберігання даних перетинів які вводяться користувачем або є дефолтними).

- Переносимо всіх дані перетинів з TBladeSectionTable в нову таблицю.

- Передаємо в процедуру інтерполяції перетинів нову таблицю, отримуємо велику кількість нових перетинів за якими і буде будуватися модель. Чим менше вказаний в інтерфейсі проміжок між перетинами тим більше буде інтерполювань нових перетинів і тим більше буде деталізація отриманої моделі.

- Створюємо об'єкти типу TSpaceNode для всіх точок перерізів таблиці.

- Заповнюємо наше тіло лопаті точками з уже повністю готовою таблиці перетинів.

- Застосовуємо багатопоточність. Основна проблема яку вона вирішує це те, що нам треба створити ще n кількість лопатей ідентичних першої, і повторювати наведену вище функцію кожен раз, це дуже затратно і довго адже точок в моделі може бути і 10 тисяч і мільйон, все залежить від деталізації моделі. Для цього багатопоточністю буде створено необхідну кількість клонів існуючої лопаті за допомогою функції TSpaceBody CloneStructure.

- Паралельно виконуємо побудову списку вузлів і списку граней всередині анонімної процедури Self отримуємо список унікальних вузлів поточного тіла.

- Клонуємо список вузлів, отримавши список нових вузлів, який відповідає списку вихідних вузлів.

- Заповнюємо тіло гранями.

- Створення клону завершено.

Технологія багатопоточності застосована наступним чином:

Етап 1. Багатопотокове створення клонів:

```
TParallelFor (0, NewFaces.Count - 1,
procedure (FaceIndex: Integer; ThreadCacheData: Pointer)
var
_Face, _FaceOriginal: TSpaceFace;
_NodeIndexInFace, _NewNodeIndex: Integer;
begin
_FaceOriginal: = FL.Objects [FaceIndex];
_Face: = NewFaces [FaceIndex];
for _NodeIndexInFace: = 0 to _FaceOriginal.NodesCount - 1 do
begin
_NewNodeIndex: = _FaceOriginal.Nodes [_NodeIndexInFace] .UnsafeObjectIndex;
_Face.AddNode (NewNodes [_NewNodeIndex]);
end;
```

Етап 2. Багатопотоковий поворот клонів

```
// Робимо клони, а потім багатопоточно повертаємо їх кожен на свій кут
ClonedBodyList: = TSpaceBodyList.Create (False);
try
// Робимо клони
T0: = GetTimeTPC ();
Result.CloneStructure (aSpace, ClonedBodyList, NumOfBlades - 1, SingleThreadBladesCopyInit);
FDuration_ms_CreateBladeCopies: = MeasureTime ();
// Повертаємо клони багатопоточно, тому що вони незалежні
if ClonedBodyList.Count> 0 then
TParallelFor (1, NumOfBlades - 1,
function (_NewBladeNum: Integer; ThreadCacheData: Pointer): Boolean
```

```
        var
          _Body: TSpaceBody;
          _Mat: TTransformationMatrix;
          _BladeAngle: Double;

        begin
          Result: = True;
          // Клон
          _Body: = ClonedBodyList [_NewBladeNum - 1];
          if not Assigned (_Body) then
            exit;
          // Кут, на який його потрібно повернути
          _BladeAngle: = _NewBladeNum * 2 * PI / NumOfBlades;
          // Застосувати перетворення
          _Mat.InitRotate (1, 0, 0, _BladeAngle);
          // _Body.Transform (_Mat);
          _Body.TransrormNodes (_Mat, _Body.CachedNodes);
          // Використовуємо список, який був заповнений при клонуванні
        end, nil, SingleThreadBladesRotation);
      FDuration_ms_RotateBladeCopies: = MeasureTime ();
    finally
```

Етап 3. Видалення сховища клонів

```
      ClonedBodyList.Free;
    end;
```

Вцілому діаграма багатопоточності матиме вигляд рис.2.3.

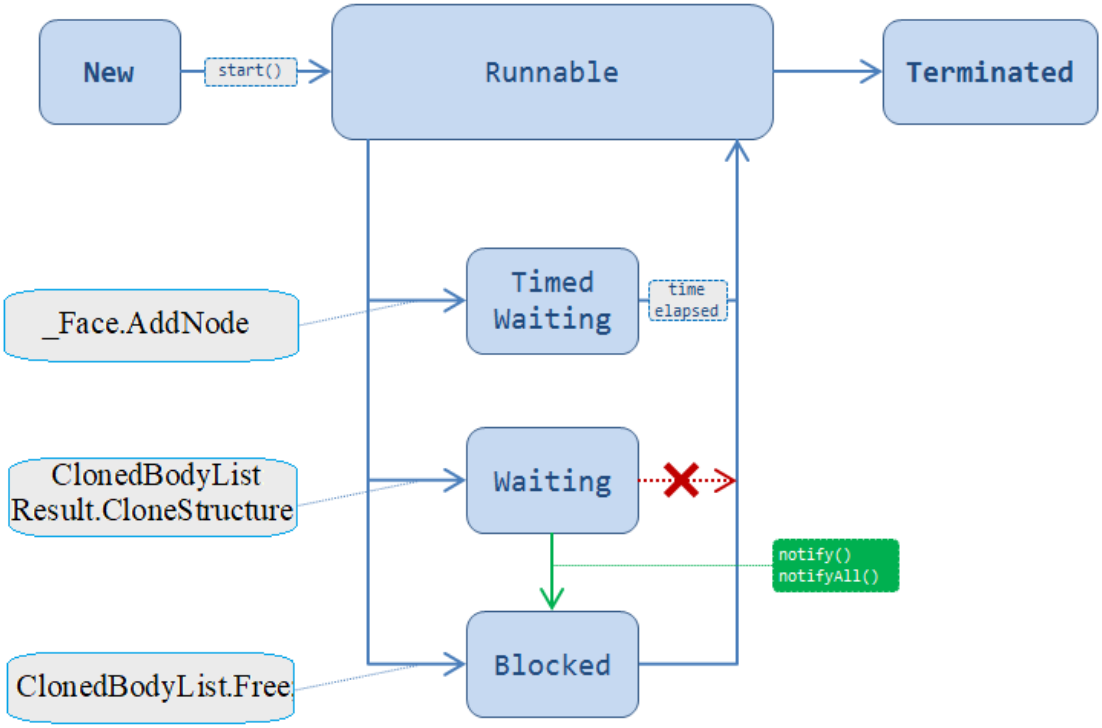


Рисунок 2.3 – Діаграма багатопоточності

Розгорнута діаграма станів показує фазові переходи багатопоточності (Рис. 2.4).

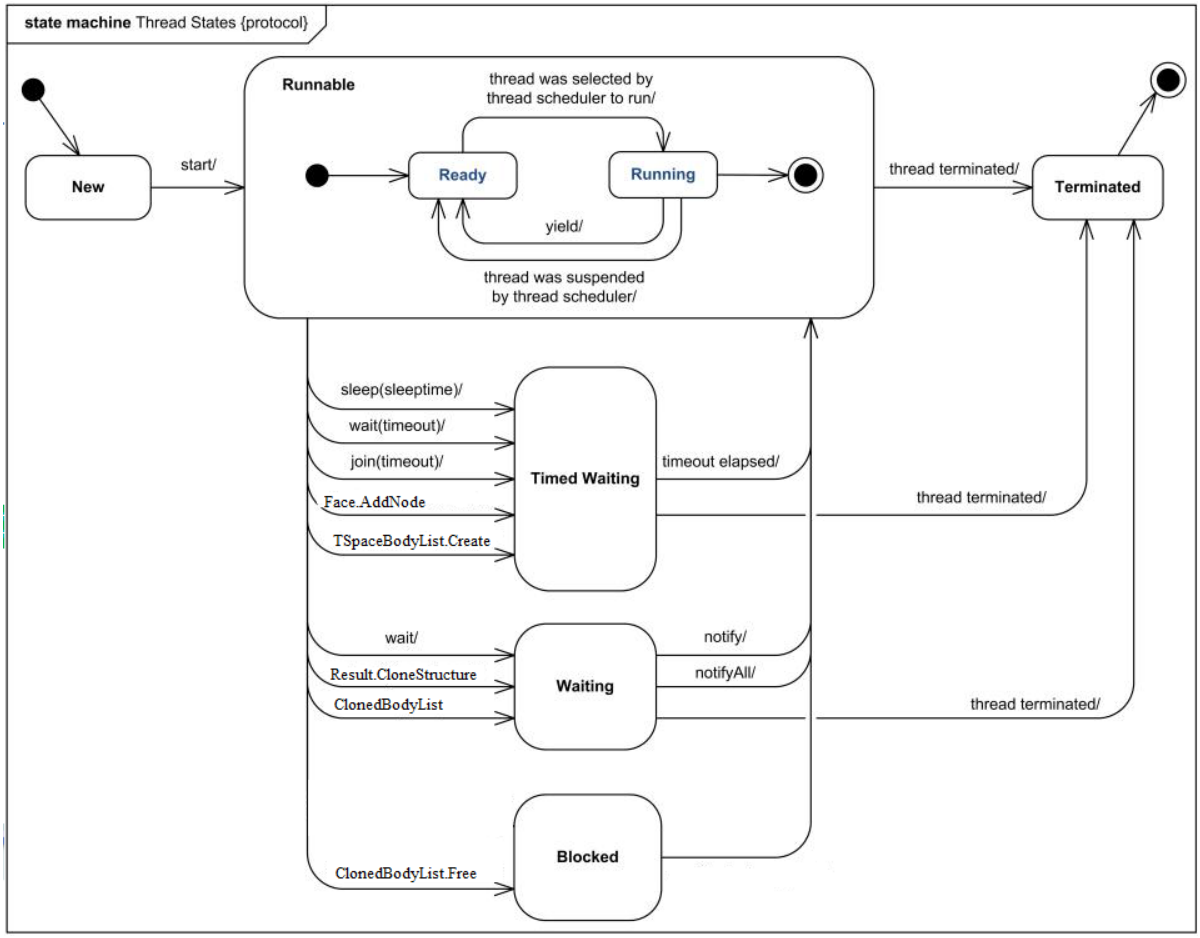


Рисунок 2.4 – Розгорнута діаграма багатопоточності

Розглянемо практичні результати застосування багатопоточності до обробки даних.

Функції перегляду та оцінки перетину у вигляді прямого (Рисунок 2.5, 2.6) на нахиленого перерізу (Рисунок 2.7) реалізовані за допомогою багатопоточного клонування фрагменту лопаті з подальшою її обробкою.

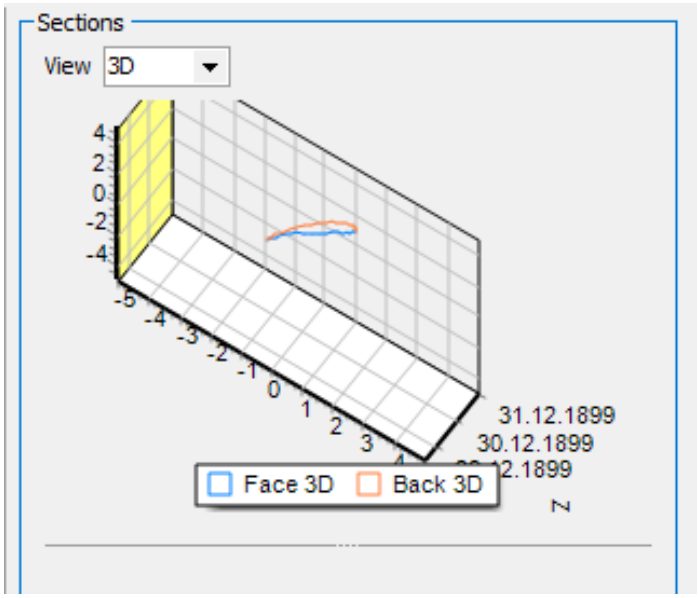


Рисунок 2.5 – Візуалізація обраного у таблиці перетину 3D

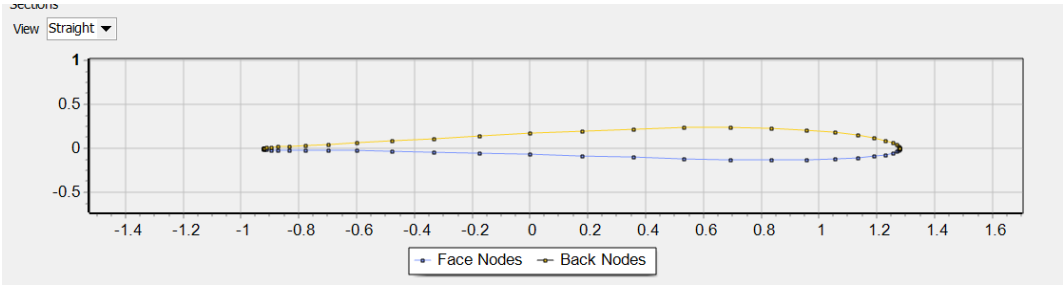


Рисунок 2.6 – Візуалізація обраного у таблиці перетину «Прямий» вид

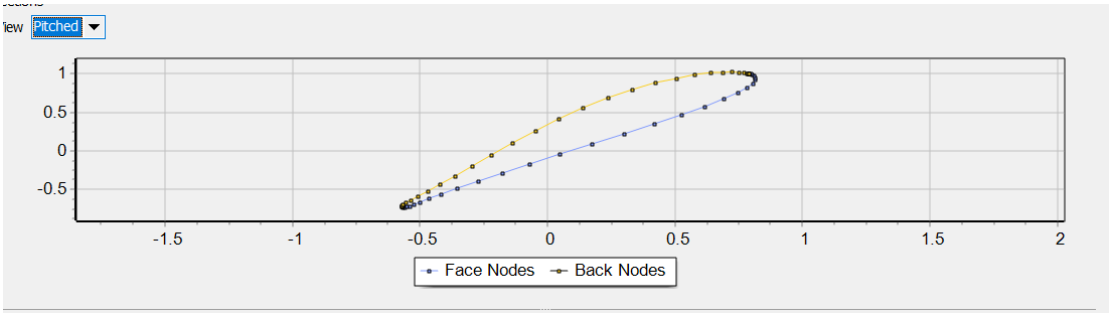


Рисунок 2.7 – Візуалізація обраного у таблиці перетину «Нахилений» вид

Виконується побудова лопаті гвинта (Рисунок 2.8).

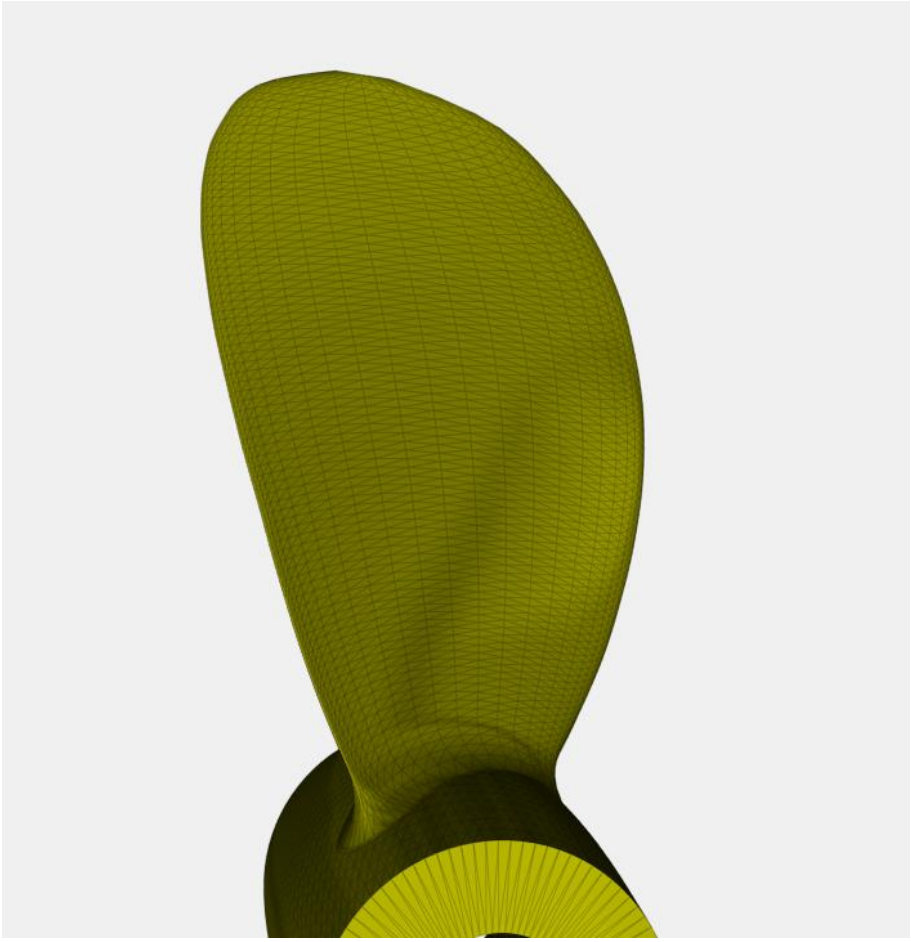


Рисунок 2.8 – Лопать гвинта

За рахунок структурного зберігання даних побудуємо першу «еталонну» лопать та за необхідністю (якщо зазначено більша кількість лопатей) скопіювати та повернути їх усі на необхідний кут одночасно.

Побудова відбувається за рахунок клонування структури Tspace, котра містить у собі наступні об'єкти:

1. SpaceNode, яка включає данні по координатам точки. Та знає про «фейси» у яких вона знаходиться.
2. SpaceFace, яка містить у собі інформацію про те які точки зберігаються у ній, та до який тіл вона належить.
3. SpaceBody, яка містить у собі інформацію про те, які «фейси» зберігаються у ній.

При цьому зберігаємо данні про цю структуру у базі даних Propeller, тому можемо при необхідності використовувати їх при подальших розрахунках, або

при перетині водною гладдю, без необхідності знову будувати нове тіло моделі (гвинта у нашому випадку) [17].

Як зазначено на Рисунку 2.9, можна бачити що на те, щоб побудувати одну лопать затрачено 13 мс, по розрахункам на створення усіх її копій буде потрачено 50 мс, менше однієї мс потратили на те щоб повернути їх усі та ще 16 мс щоб скомбінувати їх. При необхідності створювати та кріпити їх почергово це займає майже в 3 рази більше часу.

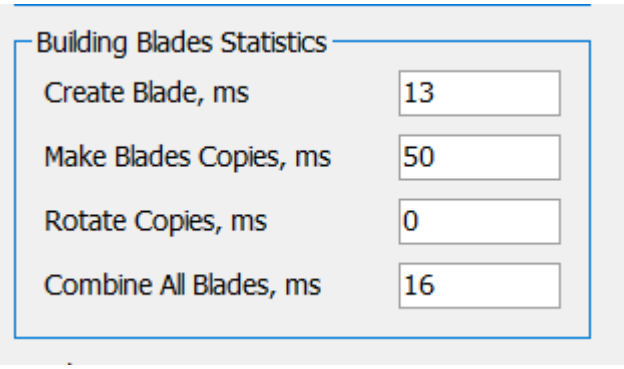


Рисунок 2.9 – Статистика затраченого часу на побудову

Отримано наступні тестові випадки візуалізації гвинта.

При наступних вхідних характеристиках

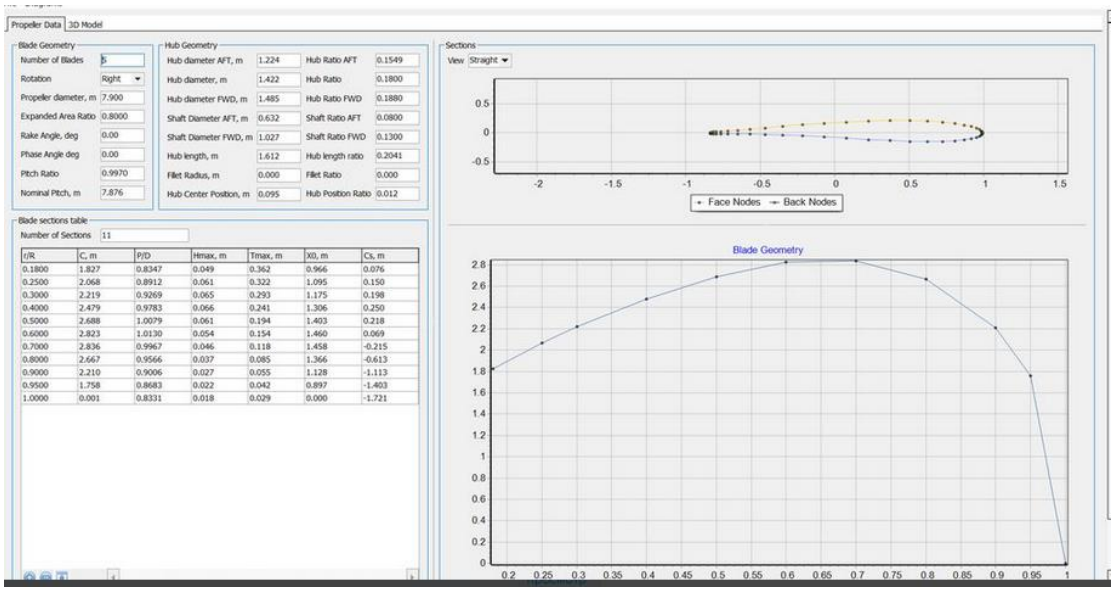


Рисунок 2.10 – Вхідні характеристики гвинта

Отримуємо наступну тривимірну модель:

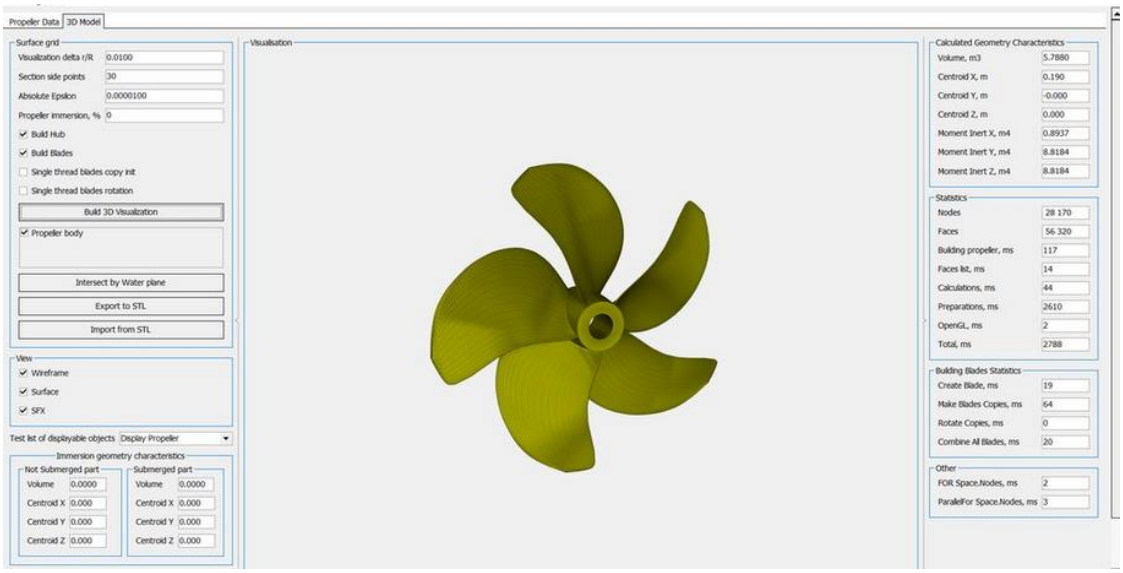


Рисунок 2.11 – Тривимірна візуалізація гвинта

При наступних вхідних характеристиках

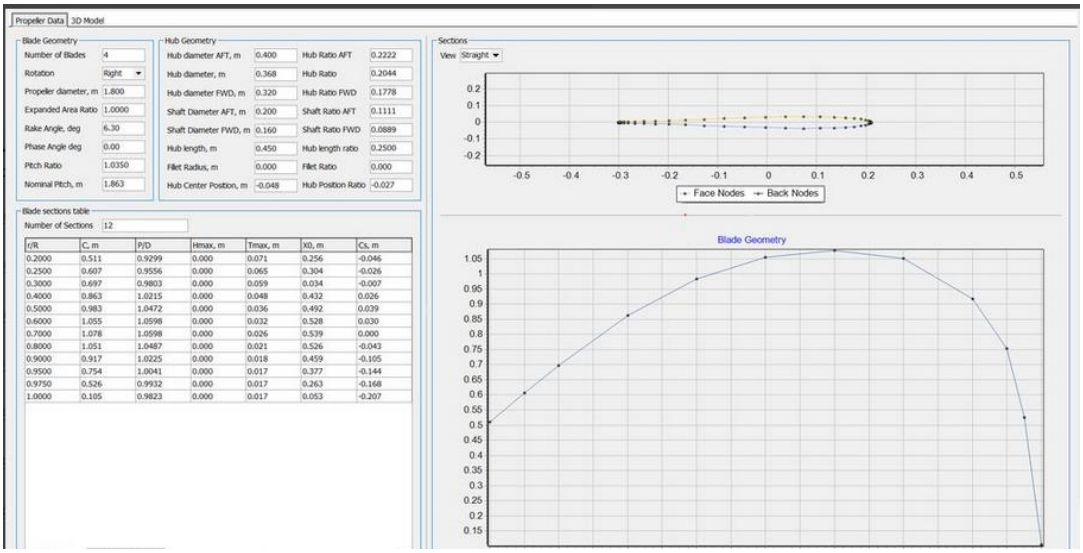


Рисунок 2.12 – Вхідні характеристики гвинта

Отримуємо наступну тривимірну модель:

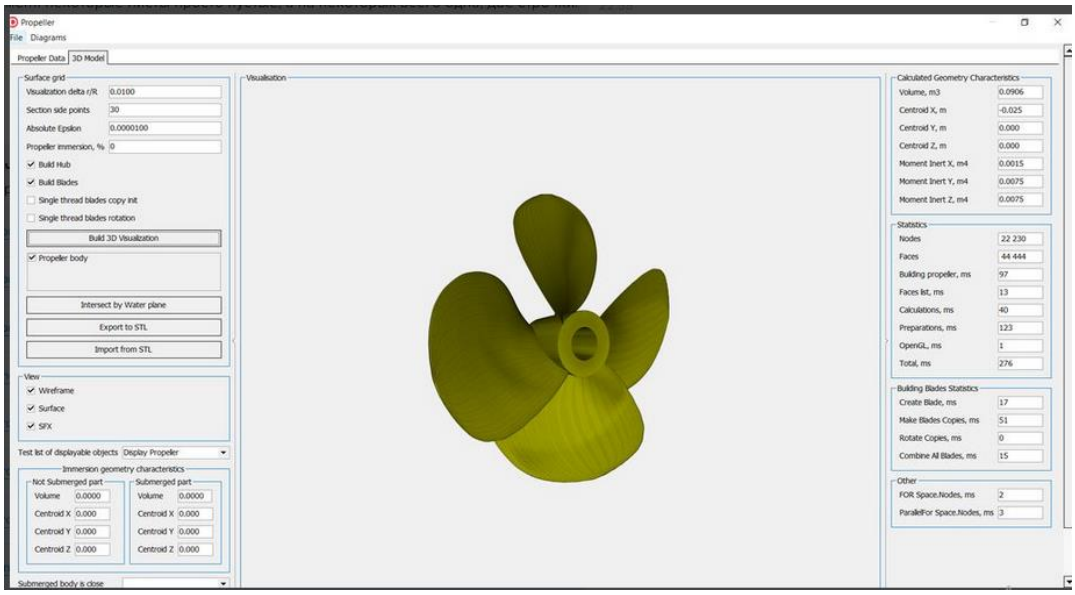


Рисунок 2.13 – Тривимірна візуалізація гвинта

При додаванні рівня води модель буде перерізана на тому рівні, який вказано користувачем при побудові:

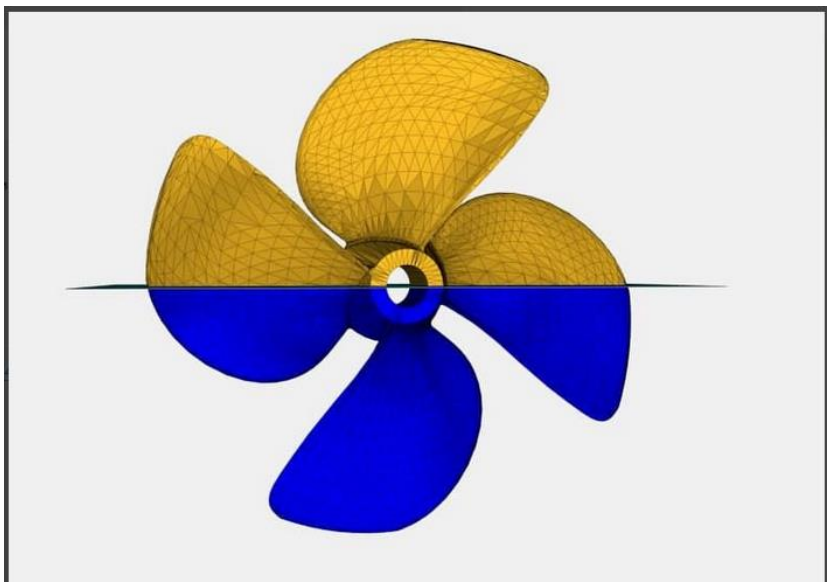


Рисунок 2.14 – Тривимірна візуалізація гвинта при перерізі водною гладдю

Була додана можливість розглянути переріз водною гладдю з різних боків, навіть приховати самі частини перерізаного гвинта (Рисунок 2.15) та отримати геометричні характеристики кожної с частин гвинта (Рисунок 2.16).

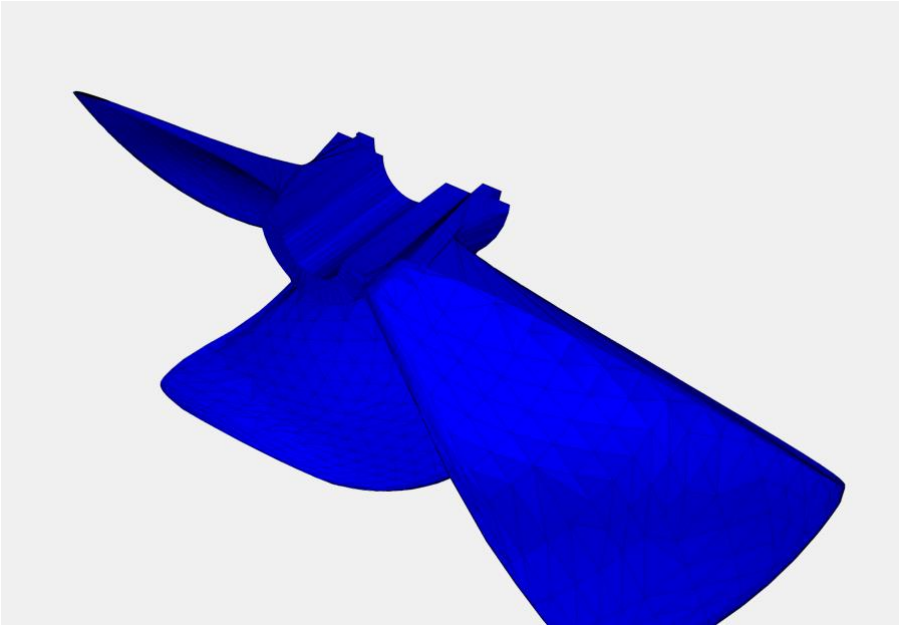


Рисунок 2.15 – Перерізана «занурена» частина гвинта

Dry part	
Volume, m3	0.0339
Centroid X	0.0343
Centroid Y	0.0040
Centroid Z	0.2038
Mass, kg	0.0339
Submerged part	
Volume, m3	0.0339
Centroid X	0.0342
Centroid Y	-0.0040
Centroid Z	-0.2038
Mass, kg	0.0339

Рисунок 2.16 – Геометричні характеристики частин розрізаного гвинта

Таким чином, отримано вдосконалення та оптимізація процесу побудови тривимірної моделі за рахунок використання багатопоточності.

РОЗДІЛ 3 ЗАГАЛЬНІ РЕЗУЛЬТАТИ ВДОСКОНАЛЕННЯ ПЗ

Прототипом вдосконалюваної системи була система розрахунку та побудови гвинта Propeller. Дана система володіла аналогічними функціями окрім можливості візуалізації тривимірної моделі.

Дана система володіла рядом недоліків:

1. Низька швидкість обробки даних.
2. Відсутність тривимірної візуалізації.

В результаті аналізу сучасних структур даних та методів їх обробки обрано нові рішення для усунення вказаних недоліків системи. А саме:

- використання полігональних моделей та розподіленої обробки даних для підвищення швидкості обробки даних,

- використання бібліотеки OpenCL для тривимірної візуалізації.

3.1 Аналіз ефективності застосування полігональних моделей

Проаналізуємо ефективність застосування полігональних моделей.

Проаналізувавши швидкість доступу програми до структури даних отримали:

Таблиця 3.1. Ефективність застосування полігональних моделей

Програми	FPS читання / запис	FPS читання / запис	FPS читання / запис	FPS оптимум
Propeller	150 / 43.5	90 / 43.47	45 /45.5	Up to 40
Propeller 2.0	250 / 90.9	150 / 90.9	90 / 90.9	Up to 90

З таблиці видно, що перша, не вдосконалена програма використовує в якості структури даних багатовимірні масиви поступається за швидкістю

доступу до даних вдосконаленій програмі, що використовує полігональні моделі для зберігання даних.

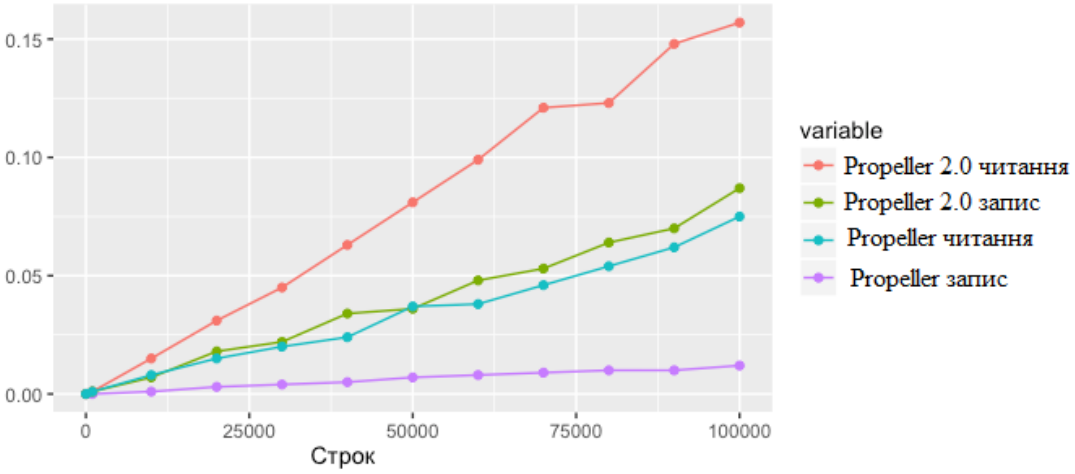


Рисунок 3.1 – Ефективність застосування полігональних моделей

На відміну від масивної, в полігональній моделі даних в вузлах ієрархічного механізму пошуку досить розташувати тільки атрибут, пов'язаний лише з одним адресним покажчиком на місце розташування даних (у вигляді величини зсуву в байтах від початку файлу), записаних останніми по порядку в первинну структуру зберігання інформації. В результаті виходить компактний ієрархічний механізм пошуку передбачуваного розміру, який може бути цілком розміщений в оперативній пам'яті комп'ютера.

Останні і попередні по порядку запису дані в первинній структурі зберігання інформації навігаційної моделі пов'язані між собою адресними покажчиками в список, аналогічний списку ідентифікаторів даних листа B $^{+}$ -дерева полігональної моделі. Додатковий пошук даних (адреса яких вже знайдена в вузлі дерева) в складі первинної структури зберігання інформації не проводиться (на етапі запису) або виробляється з використанням списку даних подібно пошуку з використанням списку ідентифікаторів в листі У $^{+}$ -дерева (на етапі читання).

У разі застосування полігональної моделі даних забезпечується тільки два звернення до енергонезалежності носію інформації на етапі запису (збереження

атрибута в складі вузла і збереження даних в складі первинної структури зберігання) і кілька звернень на етапі читання (в складі первинної структури зберігання).

З огляду на наведені вище оцінки, можна бачити, що розрив в продуктивності масивної та полігональної моделей на етапах запису і читання даних може скласти від 12 до 15 разів.

Додатковою перевагою полігональних моделей є забезпечення незалежної модифікації даних, пов'язаних в списки, в кожній окремій первинній структурі зберігання інформації навігаційної бази.

3.2 Аналіз ефективності використання розподіленої обробки даних

Наступним етапом вдосконалення є використання розподіленої обробки даних. Введемо кілька означень.

MPI-програма - це безліч паралельних взаємодіючих процесів. Всі процеси породжуються один раз, утворюючи паралельну частина програми. Кожен процес працює у своєму адресному просторі, ніяких загальних змінних або даних в MPI немає.

WCF засіб розробки програм для систем з розподіленою пам'яттю.

Програма-прототип використовує технологію обробки даних на основі MPI, а вдосконалена програма – WCF.

Проведемо оцінку застосовності WCF з наступних позицій:

− Можливість і складність розробки розподілених додатків для вирішення обчислювальних завдань.

− Ефективність обміну даними між компонентами розподіленого додатка.

− Загальна ефективність виконання розподілених обчислень з використанням WCF.

При такому підході оцінка ефективності зведеться до порівняльного аналізу часу виконання аналогічних програм на платформах WCF і MPI в аналогічному оточенні. Під аналогічними програмами маються на увазі програми, що використовують один і той же обчислювальний алгоритм аж до використовуваних оптимізацій і ідентичні типи даних. Під аналогічним оточенням розуміється ідентичність обчислювальних і мережних апаратних ресурсів, використовуваних для проведення експерименту.

В якості ілюстрації можливості, а так само оцінки складності розробки додатків для обчислювальних задач з використанням WCF пропонується реалізувати за її допомогою деякі демонстраційні алгоритми, описані на офіційному ресурсі MPI [6].

Для тестування продуктивності рішень використовуються дві ЕОМ з чотирма і двома процесорними ядрами рівній продуктивності відповідно. Комунікація здійснюється за допомогою мережі Ethernet 100Mbit. Тестування WCF проводиться на ОС Microsoft Windows 10, для побудови кластера MPI застосовується Pelican HPC [7] на ядрі ОС Linux 2.6.30 з використанням Open MPI 1.3.3.

Таким чином, можна стверджувати, що застосування WCF істотно полегшує завдання написання розподілених додатків. З результатами тестування продуктивності можна ознайомитися в таблиці 3.2.

Таблиця 3.2 Ефективність використання розподіленої обробки даних

Розмір масиву даних	Час обробки даних, мс		Перевага WCF, %
	MPI	WCF	
2	0,18	0,66	266,67
4	0,18	0,68	277,78
32	0,2	0,81	305,00
128	0,38	1,06	178,95
256	0,506	1,31	158,89
512	0,91	1,66	82,42
1024	1,528	2,58	68,85
2048	2,964	4,34	46,42
4096	5,718	7,84	37,11
8192	11,66	14,54	24,70
16384	22,706	26,18	15,30
32768	45,136	49,56	9,80
65536	89,71	103,18	15,02
131072	178,882	194,27	8,60
262144	357,234	384	7,49

З даної таблиці випливає, що, на жаль, технологія MPI не придатна для розробки програм, що вимагають частого межпроцессного обміну даними малого обсягу.

За даними результатами можна судити, що застосування технології WCF середовища .NET Framework для побудови розподілених обчислювальних додатків з малим числом межпроцессной комунікацій показує хороші результати.

За результатами представленого короткого тестування можна зробити наступні висновки:

- Розробка додатків на платформі .NET Framework для вирішення обчислювальних задач на системах з розподіленою пам'яттю є можливою.
- Технологія WCF, призначена для побудови додатків такого роду, забезпечує набагато більш простий спосіб межпроцессной комунікації, ніж це реалізовано в MPI.
- Таким чином, WCF більше підходить для вирішення завдань, що вимагають інтенсивного обміну малими групами даних між обчислювальними вузлами.

3.3 Аналіз ефективності застосування тривимірної графіки на основі OpenCL

Наступним етапом тестування ефективності вдосконалення програми є застосування тривимірної графіки на основі OpenCL.

Для тестування використано: NVidia Quadro FX5600, Версія драйвера 197.45.

Таблиця 3.3. Ефективність вдосконалення програми на основі OpenCL

Число частинок	OpenCL		MS VC++ 2008		Intel® Parallel Composer	
	FPS	GFLOPS	FPS	GFLOPS	FPS	GFLOPS
2048	15.6	1.30	15.5	1.30	110.2	9.24
4096	3.8	1.27	4	1.34	27.6	9.26
8192	1.0	1.34	1.0	1.34	7.0	9.39
16384	0.25	1.34	0.25	1.34	1.75	9.39

Код на OpenMP, скомпільований за допомогою MS VC ++ має нижчу продуктивність в порівнянні з OpenCL.

Програми на OpenCL показують гідну продуктивність у порівнянні з конкурентами і можуть успішно використовуватися для паралельних обчислень загального призначення.

Наступним етапом тестування ефективності вдосконалення програми стало проведення імпорту детальної моделі гвинта за допомогою програм – конкурентів, та порівняння часу, що був затрачений на цю операцію. Результати продемонстровано на рис. 3.2.

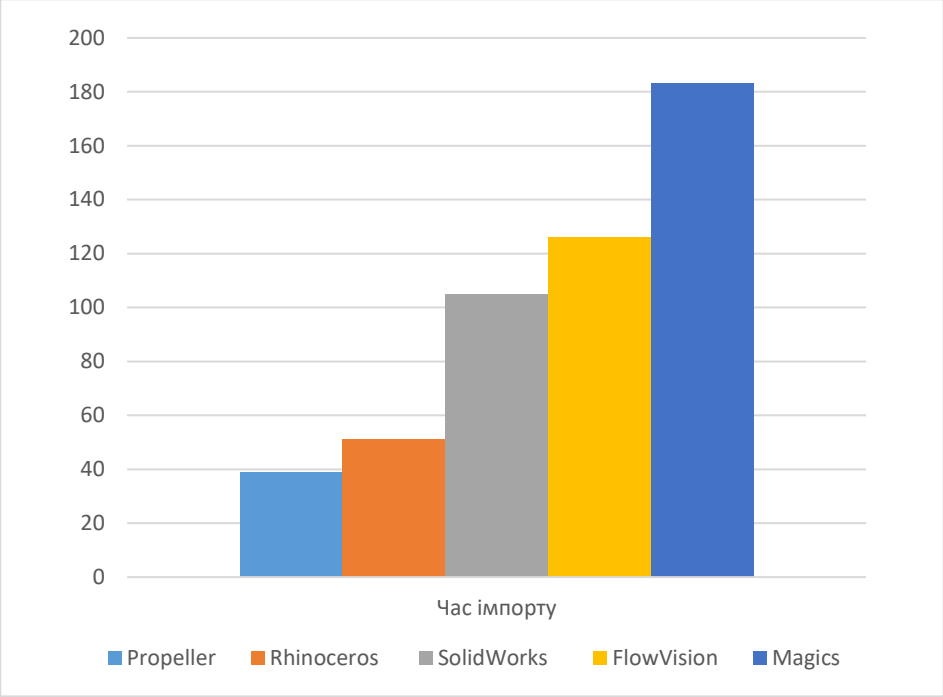


Рисунок 3.2 – Час імпорту моделі у різних програмах

Таким чином, використання полігональної моделі даних, розподілених обчислень та тривимірної графіки для вдосконалення системи себе повністю виправдали.

РОЗДІЛ 4 ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Загальні положення економічної ефективності

Необхідність вдосконалення методів і прийомів визначення економічної ефективності і розрахунків пояснюється тим, що упровадження обчислювальної техніки дорогий процес, і тому доцільність витрат на створення і упровадження інформаційної системи повинна бути серйозно обґрунтована. На створення інформаційної системи потрібні одноразові витрати на її розробку і придбання необхідного комплексу технічних засобів. Економічна ефективність системи визначається з урахуванням одноразових витрат і поточних витрат.

Для розрахунку економічної ефективності складається план аналітичної роботи, визначаючий цілі аналізу, перелік питань, що підлягають вивченню, підбір даних, використовуваних при аналізі, перевірку якості зібраних матеріалів, збіг планових і звітних показників. При розрахунку економічної ефективності інформаційної системи виробляли порівняння показників, визначення впливу окремих чинників обліку на показники, оцінка роботи і розкриття резервів, що висловлене в подальших підрозділах.

4.2 Основні показники економічної ефективності

Економічна ефективність витрат на створення інформаційної системи визначається відношенням економії, одержаної в результаті її функціонування, до капітальних витрат на розробку і упровадження системи. Економія від упровадження і експлуатації будь-якої інформаційної системи складається з економії у сфері управління. Це витікає з того, що основна роль системи полягає у тому, що вона покликана замінити управляючі функції людини

(економія у сфері управління) з метою підвищення, продуктивності суспільної праці (економія у сфері виробництва).

До основних показників, що визначають ефективність упровадження системи відносяться:

– $Є_г$- річна економія, взята в результаті упровадження системи;

– К - капітальні витрати на розробку, упровадження, придбання необхідних технічних засобів;

– С - річна собівартість системи;

– Е - річний економічний ефект;

– $E_р$ - розрахунковий коефіцієнт ефективності;

– $E_н$ - нормативний коефіцієнт ефективності.

Трудомісткість розробки програм найсильніше залежить від витрат праці на безпосередню розробку програм $З_Р$, які включають зарплату програміста, кількість розробників і годину на розробку програми. Витрати $З_Р$ розраховуються по формулі 4.1:

$$(З_Р = t_р \times C_з \times n + H_з)\,, \qquad (4.1)$$

де $t_р$ - година, що витрачається на розробку ;

$C_з$ - розмір заробітної платні програміста;

n - кількість розробників;

$H_з$ - нарахування на зарплату (26-70%).

Для розробки програмних засобів інформаційної системи необхідно орієнтовно 4 місяці.

Середня заробітна платня програміста складає 8 000 Грн. Для вирішення задачі необхідно 1 розробник. Підставляючи дані у формулу (4.1) одержимо:

$З_р$ = 4 * 8000 * 1+8000*0.0,3719*4 = 43900,8 грн.

$З_р$ = 43900,8 грн.

Розрахунок витрат на експлуатацію

Поточні витрати на експлуатацію системи $З_{ЕП}$ складаються з витрат на електроенергію $З_{ЕЛ}$, експлуатаційних витрат на витратні матеріали $З_Е$ і витрат на заробітну платню обслуговуючого персоналу $З_{ЗП}$.

Для того, щоб визначити витрати на електроенергію необхідно визначити яка кількість ПК буде підключено до джерела живлення, які периферійні пристрої, яку потужність споживають ті чи інші споживачі, яку кількість годин буде підключений ПК до джерел електроживлення і т.д.

Вартість витрат на електроенергію $З_{ЕЛ}$ на рік розраховується по формулі 4.2:

$$(З_{ЕЛ} = Р * Т_{НОМ} * С_{кч}), \quad (4.2)$$

де Р - споживана потужність;

$Т_{ном}$ - час роботи ПК в рік;

$С_{кч}$- вартість 1 кВт/год.

Середня споживана потужність ПК складає 300 Вт/год=0,3 кВт/год. Час роботи ПК складає 7 годин в день. Вважаємо, що в році 265 робочих днів. Вартість споживаної електроенергії - 1,51грн. за 1 кВт/год. Розрахуємо річні витрати на електроенергію:

$З_{ЕЛ1}$=1,51*7*265*0,3=840,36 грн.

Разом з використовуванням ПК необхідно використовувати місцеве освітлення, витрати на яку також потрібно враховувати при розрахунку. Річні витрати на місцеве освітлення розраховуються аналогічно і складають :

$З_{ЕЛ2}$=1,51*7*265*0,1=280,11 грн.

Сумарні витрати на споживання електроенергії за рік складають:

$З_{ЕЛ}$=840,36+280,11=1120,47 грн.

Зарплата одного працівника складає 3000 грн. в місяць. Для роботи з інформаційною системою досить 1 користувача. Підставивши дані у формулу розрахуємо річний фонд заробітної платні працівників :

$З_{ЗП}$=3000*12*1=36000 грн.

Визначимо сумарні поточні витрати на експлуатацію системи:

$З_{ЕП}$=1120,47+36000=37120,47 грн.

Річна собівартість

Річна собівартість проекту складається з поточних витрат на експлуатацію системи і капітальних вкладень з урахуванням нормативного коефіцієнта економічної ефективності капітальних вкладень $Е_н$. Значення коефіцієнта $Е_н$=0,15.

Для того, щоб визначити річну собівартість проекту, підставимо розраховані вище капітальні витрати і сумарні експлуатаційні витрати в формулу 4.3:

$$(С = З_{ЕП} + Е_н * К) \quad (4.3)$$

Підставляючи у формулу (4.3) розраховані вище капітальні і поточні витрати, розрахуємо річну собівартість системи:

С = 37120,47+ 0,15*43900,8=43705,59 грн.

4.3 Розрахунок економічної ефективності

Основним показником визначаючим економічну доцільність витрат на створення системи, є річний економічний ефект.

Економічний ефект від упровадження інформаційної системи можна виразити доходом, підвищенням продуктивності праці, скороченням чисельності персоналу, енергоспоживання і іншими економічними показниками.

Вибір найкращих економічних і функціональних критеріїв для узагальнення опису ефективності створення і використовування підсистеми залежить від їх призначення, області застосування і інших чинників.

Річну економію, одержану в результаті упровадження системи, $Є_г$ найпростіше і узагальнено можна описати сумарним доходом від використовування пропонованої системи. Цей прибуток можна представити як різницю витрат між існуючою технологією рішення задачі і пропонованим варіантом її рішення з урахуванням витрат, що знижують граничний дохід по формулі 4.4:

$$(Є_г = З_О - С), \qquad (4.4)$$

де $З_О$ - витрати по існуючому варіанту;

С - витрати по пропонованому варіанту.

Для розрахунку витрат по існуючому варіанту враховується загальний об'єм робіт в рік при експлуатації інформаційної системи і вартість виконаних робіт (заробітна платня на одну людину). Розрахуємо витрати на заробітну платню одного робітника $З_{роб}$:

$З_{роб}$ = 3000*12= 36000 грн.

Для роботи з існуючою інформаційною системою достатньо 2 користувача. Розрахуємо річні витрати по існуючому варіанту:

$З_о$ = 36000*2=72000 грн.

Підставляючи дані у формулу (4.4) можна розрахувати річну економію :

$Є_г$= 72000-43900,59=28099,41 грн.

Узагальнюючим показником розрахунку економічної ефективності витрат на створення інформаційної системи є розрахунковий коефіцієнт ефективності витрат E_p, який визначається по формулі 4.5:

$$(E_p = \frac{Є_Г}{K}) \qquad (4.5)$$

Підставивши у формулу (4.5) розраховані вище показники, визначимо значення коефіцієнта ефективності витрат:

$$E_p = \frac{28099.41}{43900} \approx 0{,}64$$

Ефективність системи визначається рівнем коефіцієнта ефективності витрат, якщо беруть коефіцієнт буде менше нормативного, то упровадження системи не виправдовує необхідних витрат, тобто система нерентабельна .

Якщо розрахунковий коефіцієнт ефективності капітальних вкладень на створення системи більше нормативного, то пропоновану систему можна вважати ефективною.

Іншим показником, що визначає економічну ефективність розробленої системи, є термін окупності капітальних витрат Т - показник, що характеризує період години, протягом якого загальні витрати відшкодовуються за рахунок економії поточних витрат і додаткового прибутку від реалізації продукції. Його величина є зворотній розрахунковому коефіцієнту ефективності і розраховується по формулі 4.6:

$$(T = \frac{K}{Є_{Г}} \quad) \tag{4.6}$$

Підставивши відповідні значення маємо:

$$T = \frac{43900}{28099.41} = 1{,}56 \text{ р.}$$

Річний економічний ефект пропонованої інформаційної системи розраховується по методиці визначення економічної ефективності використовування в народному господарстві нової техніки і винаходів. У розрахунку використовується єдиний нормативний коефіцієнт капітальних вкладень рівний 0,15. Розрахунок річного економічного ефекту від упровадження пропонованої системи виробляється по формулі 4.7:

$$(Є = Є_{Г} - 0{,}15 \times K) \tag{4.7}$$

Є = 28099,41 – 0,15*43900 = 21514,41 грн.

4.4 Висновки по ефективності інформаційної системи

Розрахунковим шляхом встановлено, що коефіцієнт ефективності капітальних вкладень на створення системи складає 0,64 при терміні окупності капітальних витрат, рівному близько 1,56 років. Отже, пропоновану систему можна вважати ефективною.

РОЗДІЛ 5 ОХОРОНА ПРАЦІ

5.1 Загальні положення про охорону праці та навколишнього середовища

Охорона праці є системою правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних и лікувально-профілактичних заходів, а також засобів, що забезпечують безпеку життя, здоров'я, працездатності людини в трудовій діяльності .

Закон України «Про охорону праці» визначає, основні положення по реалізації конституційного права громадян на охорону їх життя и здоров'я в процесі трудової діяльності.

У даного розділі розглядаються питання охорони праці и навколишнього середовища. Стосовно приміщення офісу бакалаврського проекту на тему: «Розробка програмного модулю авторизації для сайту на основі доступу до служби каталогу».

Все устаткування, что використовується у офісному приміщенні є споживачем однофазної мережі змінного струму з частотою 50 Гц, напругою 380/220 В. Потужність споживання від мережі змінного струму не перевищує 100 КВт.

Комп'ютер - прилад закритого типу, степень захисту 1, тобто мас Робочий ізоляцію, елемент заземлення и дріт без заземлюючої жили для Приєднання до джерела живлення, согласно ПУЕ-87 и ГОСТ 12.2.007.0-75 * . Приміщення знаходиться на іншому поверсі триповерхової Будівлі, її площа складає 5 * 3,5 = 17,5 $м^{2.}$ Офіс, де знаходиться багато енергоємного устаткування, відноситься до класу приміщень з підвищеною небезпеки поразка людини електричний ПУЕ-87, оскільки є можливість одночасного дотику до металоконструкцій будівель, что мають з'єднання із землею, з одного боку, і металева корпусами електроприладів з іншого.

Будівля, де знаходится офіс, по пожежній и вибухово-пожежній небезпеці відноситься до категорії В (в ній знаходяться тверді матеріали и Речовини, что згоряють), згідно НАПБ Б.03.002-2007 . Конструкція Будівлі за ступенем вогнестійкості відноситься до II (конструкції віконані з матеріалів, что НЕ згоряють) відповідно до ДБН В.1.1.-7-2002 .

Згідно НПАОП 40.1-1.32-01 в зоні класу П-II, П-II а допускається електроустаткування закритого виконанання.

Приміщення офісу відносяться до класу нормальних, оскільки в них відсутні ознаки, які властиві жарким, запиленим приміщенням, й приміщенням з хімічним активним середовищем.

5.2 Небезпечні та шкідливі виробничі чинники

Продуктивність праці багато в чому залежить від умов на виробництві.

Офісний робітник піддається дії шкідливих и небезпечних чинників виробничого середовища: неякісне освітлення, склад повітря, підвищений рівень шуму, шкідливі випромінювання (ультрафіолетове, інфрачервоне, електромагнітне), пожежна небезпека. Ці чинники по окремості й в комплексі впливають на організм людини, визначаючи его самопочуття.

У табліці 4.1 приведено перелік шкідливих и небезпечних чинників, что вплівають на людину при роботі з електронно-обчислювальних технікою, согласно ГОСТ 12.0.003-74 * .

Таблиця 5.1 - Небезпечні й шкідливі чинники при роботі на ПК

Найменування Чинника	Джерело Виникнення	Нормовані Параметри	Нормативний документ
Фізична небезпечна поразка електричного ланцюга	Живляча електрична мережа	$U_{пр}$ = 36 В I = 0,6 μA	ДСТУ ГОСТ 12.1.038: 2008.
Підвищений потенціал статичної електрики	Монітор комп'ютера	E = 20 кВ / м Напруженість електричного поля	ГОСТ 12.1.045-84 ДСанПіН 3.3.2-007-98 .
Коефіцієнт пульсації газорозрядними лампами	Неповна розфазірівка світильників	До $_{п}$ = 5%	ДБН В. 2.5-28-2006 .
Наявність відблісків	Монітор комп'ютера	L = 40 кд / м ^{2}	НПАОП 0.00-1.28-2010 . ДСанПіН 3.3.2-007-98 .
Підвіщений рівень шуму	Принтер, вентилятор, комп'ютер	L_A = 50дБА	ДСН 3.3.6.037-99 ГОСТ 12.1.003-83 * .
Інфрачервоне випромінювання	Монітор комп'ютера	50% поверхні тіла - 35 Вт / м ^{2} 25-50% поверхні тіла - 70Вт / м ^{2} Менше 25% поверхні тела - 100Вт / м ^{2}	ГОСТ 12.1.005-88 ДСанПіН 3.3.2-007-98 .

Найменування Чинника	Джерело Виникнення	Нормовані Параметри	Нормативний документ
Наявність пилу, озону, оксидів азоту и аероіонізації	Електронна променева трубка монітора	Озон = 0,1 мг / м ^{3} Оксид азоту = 5 мг / м ^{3} Пив = 4 мг / м ^{3}	ГОСТ 12.1.005-88 ДНАОП 0.03-3.06-80 .
Ергономічна яскравість світильників	Екран монітора ПЕОМ	L = 200 кд / м ^{2}	ВДОП 2.08-5.04-86 ДСанПіН 3.3.2-007-98 .
Нервово-психологічна напруженість праці	Відповідальність, трудність завдання	Режим роботи, розумово напряжение	ДСанПіН 3.3.2-007-98 .
Вібрація (технологічна З «в» комфорт)	принтер	$_L$ V = 75 дБ	ДСН 3.3.6.039-99 ДСТУ ГОСТ 12.1.012: 2008 .
Електромагнітне випромінювання	Електронна променева трубка монітора	кГц - 400 кГц 2,5 В / м 25нТл	НПАОП 0.00-1.28 ГОСТ12.1.006: 2008.
Ультрафіолетове випромінювання	Монітор комп'ютера	УФ - С УФ - В УФ - А	ДНАОП 0.03-3.17-88 ДСанПіН 3.3.2-007-98

5.3 Промислова санітарія

Метеорологічні умови при работі

Внутрішнє середовище приміщення визначається діючім на організм людини поєднанням температури, відносної вологості и швідкості руху повітря.

Виконувана робота відноситься до легкої категорії I а, енерговітраті організму-90-120 ккал / год. Враховуючи специфіку роботи за комп'ютером, оптімальні Параметри вибрані по ГОСТ 12.1.005-88 , ДСН 3.3.6.042-99 , які створюють відчуття теплового комфорту і є передумови збереження високого рівня працездатності, дані приведені в таблиці 5.2 .

Для забезпечення вищезгаданих метеорологічних умов в приміщенні згідно ДБН В.2.5; 2013 передбачає система опалення в холодний період року, природна вентиляція, яка здійснюється у наслідок перепадів тиску зовнішнього и внутрішнього повітря при відкритті кватирки и кондиціювання (кондиціонер БК).

Таблиця 5.2 - Оптимальні параметри мікроклімату у робочій зоні

Період року	Температура, ° С	Відносна *вологість,%*	ШВИДКІСТЬ руху Повітря, м / с
Холодний	22-24	40-60	0,1
Теплий	23-25	40-60	0,1

Освітлення виробничого приміщення

Працездатність багато в чому залежить від освітлення. Незадовільне освітлення кількісно або якісно стомлює не тільки зір, але викликає стомлення організму и цілому, робить вплив на продуктивність праці бухгалтера.

Для забезпечення нормального освітлення залежних від точності виконуваних робіт, офіс винен оснащуватися суміщеним и штучним

освітленням, враховуючи напруженість, очного аналізатора, згідно ДБН В.2.5.-28-2006 .

Згідно ДБН В.2.5.-28-2006 виконувана робота відноситься до розряду високої точності, мінімальній об'єкт розрізнення 0,3-0,5 мм, розряд зорової роботи III, фон - середній, контраст об'єкту з фоном - середній, під розряд - в.

Всі виробничі приміщення з постійнім знаходженням в них людей, відповідно до санітарних норм и правил, мають природнє освітлення.

У даному випадку використовується суміщене освітлення (природньо бокове двостороннє освітлення через віконні отвори в зовнішніх стінах, орієнтованих на південь, та штучне). Суміщене освітлення нормується коефіцієнтом природної освітленості (КПО) за ДБН В.2.5.-28-2006.

Нормовані значення КПО (e,%) для будівель розташованих в III світло кліматичному районі визначаються по формулі 5.1:

$$({}_e N = {}_e H * {}_m N), \quad (5.1)$$

де ${}_e N$ - значення КПО за таблицею1 ДБН В.2.5.-28-2006 , дорівнює 1,2%;

m - коефіцієнт світлового клімату, дорівнює 1,06 (вікна на північ);

N- номер світло кліматичного району ДБН

$e_3 = 1,2 * 1,04 = 1,25\%$

Згідно ДСанПіН 3.3.2-007-98 значення КПО винне бути не менше 1,5%.

У темний час доби використовується штучне освітлення. Освітлення загальне. Нормоване значення освітленості E_{min} = 300 лк для загально освітлення. Враховуючи можливості і вимоги до економії електроенергії, оберемо для загально освітлення відділу - люмінесцентні лампи ЛБ 40-4, люмінесцентні світильники ДСП 01 2 * 40. Обраний світильник прямого світла з дифузним віддзеркаленням, із захисним пристроєм, что оберігає від засліплення и відображеного блиску та оснащено захисним пристроєм для регулювання яскравості.

Шум і вібрація

У офісному приміщенні найсильнішими джерелами шуму та вібрації є друкуюча техніка, а також вентилятори процесора і блоку живлення, рівень

шуму, яких в робочих місцях не перевищує L_a=50дБА, що відповідає нормам ГОСТ12.1.003-83ССБТ,ДСН 3.3.6.037-99 . Вібрація в приміщеннях технологічна, категорія 36 «комфорт», рівень віброшвидкості не повинен перевищувати L_v= 75 дБ згідно ДСТУ ГОСТ 12.1.012:2008 , ДСН 3.3.6.039-99 . Оскільки для джерел шуму і вібрації передбачені конструктивні заходи захисту(виконання в шумовібростроковому варіанті), то додаткові засоби захисту від шуму і вібрації не використовуються. Для зменшення рівня шуму застосовуються демпфуючі матеріали (під принтер підкладається гумова ковдра та інш.).

5.4 Пожежна безпека

Пожежна безпека — стан об’єкта, при якому з регламентованою імовірністю виключається можливість виникнення і розвитку пожежі, дія на людину небезпечних факторів пожежі, а також забезпечується захист матеріальних цінностей. Причини, що можуть викликати пожежу в розглянутому приміщенні, є:

- несправність електропроводки і приладів;
- коротке замикання електричних ланцюгів;
- перегрів апаратури;
- блискавка.

Класифікація:

1) категорія приміщення за вибуховопожежній та пожежній безпеці НАПБ Б.03.002-2007 ;
2) клас приміщень по пожежонебезпеці П-II, П-IIа згідно НПАОП 40.1.-1.32.-01.

Ступінь вогнестійкості будинку - II згідно ДБН В. 1.1-7-2002 , (будинок з несучими конструкціями, що обгороджують, із природних, штучних, кам’яних матеріалів. У покриттях будинків допускається застосовувати незахищені

сталеві конструкції).

Пожежна безпека відповідно до ГОСТ 12.1.004-91* забезпечується системами запобігання пожежі, протипожежного захисту, організаційно-технічними заходами.

Міри систем і запобігання пожежі:

1) контроль і профілактика ізоляції;
2) наявність плавких вставок і запобіжників в устаткуванні;
3) для захисту від статичної електрики використовується захисне заземлення згідно НПАОП 0.00-1.29-97 ;
4) захист від блискавки будинку згідно ДСТУ Б В.2.5-28-2008 .

Для даного класу будинків з урахуванням кількості грозових годин у році установлюється II рівень захисту приміщення (клас зон: П-IIа, П-II; ступінь захисту: для устаткування IP 44, для світильників IP2X).

Система протипожежного захисту:

- система електричної пожежної сигналізації оснащена димовими сповіщувачами;
- приміщення оснащене вуглекислотними вогнегасниками ВВК-5 у розрахунку 2 шт. на 20 м^2.

Організаційні міри пожежної профілактики:

- навчання персоналу правилам пожежної безпеки;
- видання необхідних інструкцій і плакатів;
- план евакуації персоналу у випадку пожежі.

Для успішної евакуації персоналу, двері приміщення мають наступні розміри:

- ширина не менш 1,5 м;
- висота не менш 2 м;
- довжина не менш 1,8 м.

Робоче приміщення повинне мати два виходи. Відстань від найбільш віддаленого робочого місця не повинне перевищувати 100 м.

5.5Вимоги до організації робочого місця

Організація робочого місця по обслуговуванню, ремонту і налагодженню ПК повинна забезпечувати відповідність всіх елементів робочого місця і їхнє розташування ергономічним вимогам ГОСТ 12.2.032-78. ССБТ. , характеру і особливостям виконуваної роботи.

Площа робочого місця повинна бути не менше 6 $м^2$/чол. і об'ємом не менш 20 $м^3$ /чол.. та повинно знаходитися на відстані не менш 1 м від приладів опалення.

Ширина і глибина сидіння оператора не повинна бути менш 400 мм. Висота поверхні сидіння повинна регулюватися в межах 400-500 мм, а кут нахилу поверхні від 150° до 50° назад. Поверхня сидіння повинна бути плоскою, передній екран - закругленим.

Екран відео термінала і клавіатура повинні розташовуватися на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків і символів. Відстань від екрана до очей оператора повинна складати при розмірі екрана по діагоналі:

- 14"/15"- 600-700 мм;
- 17" - 700 - 800 мм;
- 19"- 800-900 мм;
- 21"- 900-1000 мм.

Розташування екрана відео термінала повинно забезпечувати зручність зорового спостереження у вертикальній площини під кутом ±300° від напрямку погляду оператора.

Розміщення принтера чи іншого пристрою для уведення висновку інформації на робочому місці повинне забезпечувати гарну видимість екрана відео термінала, зручність ручного керування пристроєм для уведення висновку інформації в зоні досяжності моторного поля чи:

- висота 900 - 1300 мм;

- глибина 400 - 500 мм.

5.6 Охорона навколишнього середовища

Закон України «Про охорону навколишнього природного середовища» визначає правові, економічні, соціальні основи охорони навколишнього середовища. Задача Закону полягає в регулюванні відносин в області охорони природи, використовуванні і відтворюванні природних ресурсів, забезпеченні екологічної безпеки, попередженні і ліквідації наслідків негативної дії на навколишнє середовище господарської і іншої діяльності людини, збереження природних ресурсів, генетичного фонду нації, ландшафтів і інших природних об’єктів.

На підприємстві упроваджують спеціальні дні захисту навколишнього середовища. В такі дні робітники прибирають навколишні території, роблять нові насадження.

Посилювання вимог до виробництва і матеріалів, а також розробка нових виробничих і утилізаційних технологій дозволяє зменшити антропогенне навантаження на навколишнє середовище. Необхідно упроваджувати на підприємствах.

РОЗДІЛ 6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Інтенсивний розвиток електроніки та радіотехніки викликав забруднення природного середовища електромагнітними випромінюваннями (полями). Головними їхніми джерелами є радіо-, телевізійні і радіолокаційні станції, високовольтні лінії електропередач, електротранспорт. Поблизу кожного обласного центру, багатьох районних центрів, великих міст розташовані телевізійні центри або ретранслятори, радіоцентри, засоби радіозв'язку різного призначення.

Рівень електромагнітних випромінювань у таких районах (діапазон радіочастот об'єктів може змінюватися від 50— 100 Гц до 100 ГГц) часто перевищує допустимі гігієнічні норми й дуже шкодить здоров'ю людей, що мешкають поруч.

Мірою забруднення електромагнітними полями є напруженість поля (В/м). Ці поля завдають шкоди перш за все нервовій системі. Так, напруженість поля 1000 В/м спричинює головний біль і сильну втому, більші значення зумовлюють розвиток неврозів, безсоння, важкі захворювання.

Існують розроблені на основі медико-біологічних досліджень санітарні норми та правила щодо радіотехнічних і електротехнічних об'єктів. Вони регламентують умови їхньої експлуатації з метою охорони населення від шкідливого впливу електромагнітних випромінювань.

Зростання енергетичних потужностей становить небезпеку для довкілля — розширюється мережа та зростає напруга повітряних ліній електропередач. Вони негативно впливають на нормальний розвиток тваринного та рослинного світу.

Спеціальні дослідження показали, що технічно найперспективнішими є лінії надвисокої та ультрависокої напруги (760— 1150 кВ), котрі становлять небезпеку. Навколо них утворюються потужні електромагнітні поля, які

негативно впливають на людину, порушують природну міграцію тварин, процеси росту рослин тощо.

Основні методи визначення забруднень

Турботу про стан навколишнього середовища стимулювала, започаткована в 1972 році, міжнародна програма UNEP (United Nation Environment Protection — Охорона навколишнього середовища ООН), яка передбачає глобальний моніторинг навколишнього середовища. Під моніторингом розуміється система спостереження, контролю прогнозу та управління екологічними процесами. Моніторинг дозволяє виявляти критичні та екстремальні ситуації, фактори антропогенного впливу на довкілля, здійснювати оцінку та прогноз стану об'єктів спостереження, керувати процесами взаємовпливу об'єктів гідросфери, літосфери, атмосфери, біосфери та техносфери.

Таким чином, суть моніторингу зводиться до таких функцій:

— контролю за станом об'єктів екосистеми;

— контролю за джерелами поширення екологічної рівноваги;

— моделювання та прогнозу екологічного стану екосистеми;

— керування екологічними процесами.

Важливими елементами моніторингу є визначення гранично допустимих концентрацій (ГДК) шкідливих хімічних домішок у повітрі, воді, ґрунті, продуктах харчування.

Гранично допустима концентрація — максимальна кількість шкідливих речовин в одиниці об'єму або маси середовища води, повітря або ґрунту, яка практично не впливає на стан здоров'я людини. ГДК встановлюється компетентними установами, комісіями як норматив. Останнім часом при нормуванні ГДК враховують не лише вплив забруднювачів на стан здоров'я людини, але й їхній вплив на диких тварин, рослин, гриби і мікроорганізми, природні угруповання, а також клімат, прозорість атмосфери і санітарно-побутові умови життя. Зараз у більшості країн встановлено значення ГДК більш ніж для 700 шкідливих газів, парів і пилу в повітрі. Гранично допустиме

навантаження (ГДН) — граничне значення господарського або рекреаційного навантаження на природне середовище, яке встановлюється з врахуванням ємності природного середовища або ресурсного потенціалу, здатності до саморегуляції і відтворення з метою охорони навколишнього середовища від забруднення, виснаження і руйнування.

Ці нормативи мають законодавчу силу і є юридичною основою для санітарного контролю.

Для всіх об'єктів, які забруднюють атмосферу, розраховують і встановлюють норми гранично допустимих викидів (ГДВ). Гранично допустимі викиди — це кількість шкідливих речовин, що не має перевищуватися під час викиду в повітря за одиницю часу, і концентрація забруднювачів повітря, яка на межі санітарної зони не повинна перевищувати ГДК. Виконується інвентаризація джерел забруднення атмосфери для кожного підприємства, а також екологічна паспортизація всіх об'єктів, які забруднюють довкілля.

У зв'язку з тим, що в реальних умовах людина відчуває на собі комбіновану, комплексну і сумісну дію хімічних, фізичних та біологічних факторів навколишнього середовища і це реальне навантаження визначає можливі зміни у стані здоров'я, введено поняття максимально допустимого навантаження (МДН). Під МДН слід розуміти таку максимальну інтенсивність дії всієї сукупності факторів навколишнього середовища, яка не справляє прямого чи опосередкованого шкідливого впливу на організм людини та її нащадків і не погіршує санітарних умов життя. В Україні стан довкілля нині контролюється кількома відомствами і міністерствами. Держкомгідромет України здійснює спостереження за станом атмосферного повітря на стаціонарних пунктах державної системи спостережень, він же організовує спостереження за станом атмосферних опадів, за метеорологічними умовами, за станом поверхневих, підземних вод суші та морських вод на пунктах спостереження, за станом озонового шару у верхній частині атмосфери.

Мінекобезпеки України контролює джерела промислових викидів у атмосферу, дотримання норм ГДВ, норм скидів стічних вод, тимчасово

погоджених скидів (ТПС) і гранично допустимих скидів (ГДС), контролює якість поверхневих вод суші, стан ґрунтів.

Важлива роль в питаннях контролю за станом довкілля належить Міністерству охорони здоров'я, лісового господарства, сільського господарства України, Держкомгеології, держводгоспу, держкомзему України та їхнім відділам в областях та районах.

ВИСНОВКИ

В рамках кваліфікаційного проектування реалізоване та досліджене підвищення швидкості виконання програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

Головною задачею оптимізації є використання багатопоточного оброблення даних підвищено швидкість виконання програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Для реалізації поставленої мети вирішені наступні задачі.

1 Проаналізована та обрана структуру даних для вдосконалення програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

2 Проаналізовано та обрано спосіб обробки даних для вдосконалення програмного забезпечення.

3 Проаналізований та обраний спосіб тривимірної візуалізації для вдосконалення програмного забезпечення.

4 Проведено вдосконалення програмного забезпечення в розрізі структур даних, методів їх обробки та візуалізації.

5 В результаті аналізу сучасних структур даних та методів їх обробки обрано нові рішення для усунення вказаних недоліків системи. А саме:

- використання полігональних моделей та розподіленої обробки даних для підвищення швидкості обробки даних,

- використання бібліотеки OpenCL для тривимірної візуалізації.

6 Проаналізувавши швидкість доступу приграми до структури даних отримали, що перша, не вдосконалена програма використовує в якості структури даних багатовимірні масиви поступається за швидкістю доступу до даних вдосконаленій програмі, що використовує полігональні моделі для зберігання даних. Це зумовлено тим, що у разі застосування полігональної

моделі даних забезпечується тільки два звернення до носія інформації на етапі запису (збереження атрибута в складі вузла і збереження даних в складі первинної структури зберігання) і кілька звернень на етапі читання (в складі первинної структури зберігання). З огляду на наведені вище оцінки, можна бачити, що розрив в продуктивності масивної та полігональної моделей на етапах запису і читання даних може скласти від 12 до 15 разів. Додатковою перевагою полігональних моделей є забезпечення незалежної модифікації даних, пов'язаних в списки, в кожній окремій первинній структурі зберігання інформації навігаційної бази.

7 Використання розподіленої обробки даних показує, що технологія MPI не придатна для розробки програм, що вимагають частого межпроцессного обміну даними малого обсягу. За даними результатами можна судити, що застосування технології WCF середовища .NET Framework для побудови розподілених обчислювальних додатків з малим числом межпроцесної комунікацій показує хороші результати. За результатами реалізації розподіленої обробки даних можна зробити наступні висновки:

- Розробка додатків на платформі .NET Framework для вирішення обчислювальних задач на системах з розподіленою пам'яттю є можливою.
- Технологія WCF, призначена для побудови додатків такого роду, забезпечує набагато більш простий спосіб межпроцессной комунікації, ніж це реалізовано в MPI.
- Таким чином, WCF більше підходить для вирішення завдань, що вимагають інтенсивного обміну малими групами даних між обчислювальними вузлами.

8 Вдосконалення програми застосуванням тривимірної графіки на основі OpenCL. Показало, що код на OpenMP, скомпільований за допомогою MS VC ++ має нижчу продуктивність в порівнянні з OpenCL.

9 Проведення імпорту детальної моделі гвинта за допомогою програм – конкурентів, та порівняння часу, що був затрачений на цю операцію в розробленої програми майже вдвічі менший.

Таким чином, використання полігональної моделі даних, розподілених обчислень та тривимірної графіки для вдосконалення системи себе повністю виправдали.

Розроблене програмне забезпечення має універсальний характер і може бути використане для інших поверхонь, в т.ч корпусу плавзасобу. Використані полігональні моделі та розподілена обробка даних сприяє отриманню максимально можливої продуктивності роботи прикладних програм.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вергунов С. В. К вопросу о проблемах визуализации. Понятийный аспект // Вісник Харківської державної академії дизайну і мистецтв : Зб. наук. пр. / За ред. Даниленка В. Я. — Харків : ХДАДМ, 2006. — №1. — С. 36 – 42.

2. Вергунов С. В. К вопросу о классификации видов компьютерной графики // Вісник Харківської державної академії дизайну і мистецтв : Зб. наук. пр. / За ред. Даниленка В. Я. — Харків : ХДАДМ, 2006. — №2. — С. 3 – 14.

3. Вергунов С. В. К вопросу о проблемах визуализации. Инструментальный аспект // Вісник Харківської державної академії дизайну і мистецтв : Зб. наук. пр. / За ред. Даниленка В. Я. — Харків : ХДАДМ, 2006. — №4. — С. 11 – 17.

4. Вергунов С. В. К вопросу о классификации компьютерных графических систем // Вісник Харківської державної академії дизайну і мистецтв : Зб. наук. пр. / За ред. Даниленка В. Я. — Харків : ХДАДМ, 2006. — №5. — С. 3 – 16.

5. Вергунов С. В. Классификация средств 3D-моделирования промышленного дизайнера // Вісник Харківської державної академії дизайну і мистецтв : Зб. наук. пр. / За ред. Даниленка В. Я. — Харків, 2009. — №14. — С. 18 – 23.

6. Вергунов С.В. Трехмерная модель в промышленном дизайне, ее определение и сущность // Культура народов Причерноморья : Научный журнал. — Симферополь, 2009. — №167. — С. 145 – 148.

7. Вергунов С.В. 3D-модель и 3D–моделирование в промышленном дизайне // Педагогіка вищої та середньої школи [Текст] : Зб. наук. пр. — Мистецька освіта в Україні (теорія, методи, технології) / редкол.: В. К. Буряк (гол. ред.) та ін. — Кривий Ріг : КДПУ, 2009. — С. 84 – 89.

8. Вергунов С. В. Роль и место дизайнера в процессе создания промышленных изделий на основе трехмерной модели. Часть 1. Этапы

процесса // Культура народов Причерноморья : Научный журнал. — Симферополь, 2009. — №169. — С. 10 – 13.

9. Вергунов С. В. Статистические исследования по применению трехмерного моделирования в курсовом проектировании студентов специализации «Промышленный дизайн» на кафедре «Дизайн» ХГАДИ в 2008-09 учебном году // Вісник Харківської державної академії дизайну і мистецтв [Текст] : Зб. наук. пр. / За ред. Даниленка В. Я. — Харків, 2009. — №15. — С. 11 – 13.

10. Вергунов С. В. Роль и место дизайнера в процессе создания промышленных изделий на основе трехмерной модели. Часть 2. Уровни этапов / Під загальною редакцією Трегуб Н. С. — Харків : ХДАДМ, 2010. — №1. — С. 260 – 262.

11. Чорний Є. М., Батрак Ю.А. Обчислення геометричних характеристик полігональних й поліедральних клітинних комплексів, Миколаїв, 1994, 19с.

ДОДАТОК А. ТЕХНІЧНЕ ЗАВДАННЯ

1 Загальні положення

Задачі обрахунку елементів судобудування набули нового змісту з появою технологій, що дозволяють автоматизувати оброку складних обчислень з великими масивами структурованих даних. Такі задачі вимагають нових підходів до обробки даних включаючи паралельні та розподілені обчислення. Особлива увага при математичному моделюванні приділяється графічній візуалізації результатів моделювання, особливо з використанням тривимірних побудов. Наявність даних технологій актуалізувало вдосконалення програми розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Головною задачею оптимізації є використання багатопоточного оброблення даних, підвищено швидкість виконання програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів.

Розроблювана програма покликана автоматизувати дані розрахунки та, найголовніше, побудувати тривимірну модель отриманого гвинта.

Мета дослідження: підвищення швидкості виконання програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

1.1 Змістовний опис і аналіз предметної області

Розробка алгоритмів (а особливо методів паралельних обчислень) для розв’язання складних науково-технічних задач часто є доволі значною проблемою. Дії для визначення ефективних способів організації паралельних

обчислень можуть полягати в такому:

− виконати аналіз наявних обчислювальних схем і здійснити їхній поділ (декомпозицію) на частини (підзадачі), які можна реалізувати значною мірою незалежно одна відодної;

− виділити для сформованого набору підзадач інформаційні взаємодії, які повинні здійснюватися в ході розв’язання вихідної поставленої задачі;

− визначити необхідну (або доступну) для розв’язання задачі обчислювальну систему й виконати розподіл отриманого набору підзадач між процесорами системи.

При найбільш загальному розгляді є цілком очевидним, що обсяг обчислень для кожного використаного в системі процесора повинен бути приблизно однаковим – це дозволить забезпечити рівномірне обчислювальне завантаження (балансування) процесорів. Також зрозуміло те, що розподіл підзадач між процесорами повинен виконуватись таким чином, щоб кількість інформаційних зв’язків (комунікаційних взаємодій) між підзадачами була мінімальною.

Після виконання всіх перерахованих етапів проектування можна оцінити ефективність розроблювальних паралельних методів: для цього за звичаєм обумовлюються значення показників якості породжуваних паралельних обчислень (прискорення, ефективність, масштабованість). За результатами проведеного аналізу може виявитися необхідним повторення окремих (у крайньому випадку всіх) етапів розробки. Слід зауважити, що повернення до попередніх кроків розробки може відбуватися на будь-якій стадії проектування паралельних обчислювальних схем.

Тому часто додатковою дією, яка виконується в наведеній вище схемі проектування, є коректування складу сформованої множини підзадач після визначення наявної кількості процесорів – підзадачі можуть бути укрупнені (агреговані) при наявності малого числа процесорів або навпаки деталізовані у протилежному випадку. В цілому, ці дії можуть визначатись як масштабування розроблювального алгоритму й виділятись як окремий етап проектування

паралельних обчислень.

Щоб використати одержуваний в остаточному підсумку паралельний метод, необхідно виконати розробку програм для розв’язання сформованого набору підзадач і розмістити розроблені програми по процесорах відповідно до обраної схеми розподілу підзадач. Для проведення обчислень програми запускаються на виконання (програми на стадії виконання зазвичай йменуються процесами). Для реалізації інформаційних взаємодій програми повинні мати у своєму розпорядженні засоби обміну даними – канали передачі повідомлень.

1.2 Математичне забезпечення

Для реалізації задачі застосувати полігональну структуру даних та методи багатопоточного оброблення даних.

Для програмної реалізації розрахунку гвинта в полігональній моделі використати структуру класу Model:

- для збереження вхідних характеристик,
- для проміжних обчислень гвинта,
- для вихідної інформації про гвинт,
- для побудови тривимірного зображення.

1.3 Опис процесу діяльності

Діаграма багатопоточності повинна мати вигляд рис.А.1.

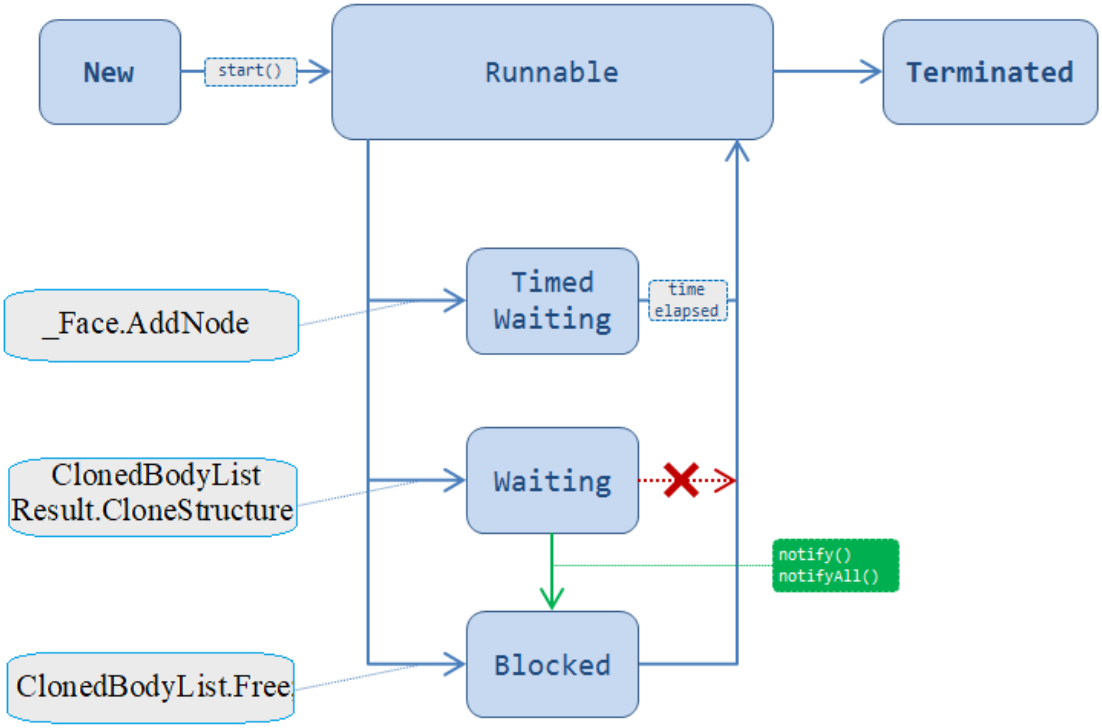


Рисунок А.1 – Діаграма багатопоточності

Розгорнута діаграма станів повинна показувати фазові переходи багатопоточності (Рис. А.2).

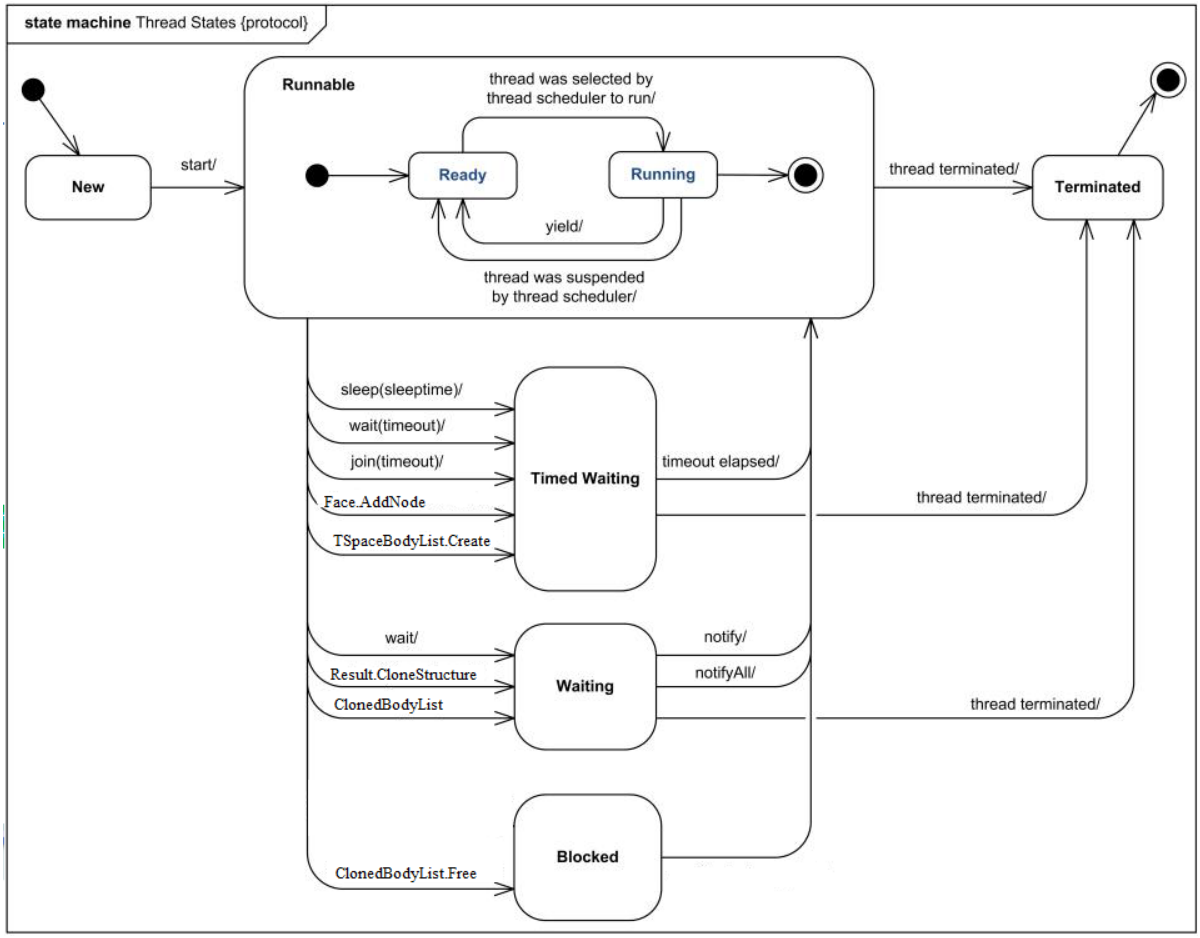


Рисунок А.2 – Розгорнута діаграма багатопоточності

1.4 Аналіз вимог до програмного забезпечення

1.4.1 Розроблення функціональних вимог

Побудова повинна відбувається за рахунок клонування структури Tspace, котра містить у собі наступні об'єкти:

4. SpaceNode, яка включає данні по координатам точки. Та знає про «фейси» у яких вона знаходиться.

5. SpaceFace, яка містить у собі інформацію про те які точки зберігаються у ній, та до який тіл вона належить.

6. SpaceBody, яка містить у собі інформацію про те, які «фейси» зберігаються у ній.

При цьому необхідно зберігати данні про цю структуру у базі даних Propeller, та, використовувати їх при подальших розрахунках.

1.4.2 Розроблення нефункціональних вимог

Були виділені наступні нефункціональні вимоги:

- продукт повинен працювати на операційній системі Windows та при наявності Microsoft Word версії 2007 та вище;
- має буди передбачений захист від некоректних дій користувача;
- інтерфейс має бути простим для використання.

1.4.3 Постановка комплексу завдань модулю

Мета дослідження: підвищення швидкості виконання програмного забезпечення створення, модифікації та аналізу тривимірних полігональних моделей об’єктів.

Для реалізації поставленої мети ставляться наступні завдання:

- Проаналізувати та обрати структуру даних для вдосконалення програмного забезпечення побудови та розрахунку геометричних характеристик тривимірних полігональних моделей об’єктів;
- Проаналізувати та обрати спосіб обробки даних для вдосконалення програмного забезпечення;
- Проаналізувати та обрати спосіб тривимірної візуалізації для вдосконалення програмного забезпечення;
- Провести вдосконалення програмного забезпечення в розрізі структур даних, методів їх обробки та візуалізації;
- Оцінити ефективність застосування обраних технології для вдосконалення програмного забезпечення.

ДОДАТОК Б. ТЕКСТ ПРОГРАМИ

```
    procedure   TPropertiesOfPropeller.BuildPropellerModel(aSpace:
TSpace;
      BuildHub,      BuildBlades,      SingleThreadBladesCopyInit,
SingleThreadBladesRotation: Boolean);
    begin
      // вызываем процедуру построения ступицы
      if BuildHub then
        FObjectPropeller.CreateBodyHub(aSpace);

      // вызываем процедуру пропеллера по созданию лопастей
    if BuildBlades then
        FObjectPropeller.CreateBodyWithAllBlades(aSpace,
SingleThreadBladesCopyInit,
    SingleThreadBladesRotation);
    end;

    function       TPropeller.CreateBodyHub(aSpace:       TSpace):
TSpaceBody;
    const
      N_fi = 90;
      // количество узлов в сечении
      N_x = 21; // количество сечений

      DeltaFi = 2 * PI / N_fi;
    var
      iAngl, iSect, i: Integer;
      SectArr: array of TSpaceNodeList;
      Fi, SectX, SectRad, HubRadius, X_fwd, X_aft: Double;
      DeltaX, aSin, aCos, Rad_fwd, Rad_aft: Double;
      aNode,  HubNode1,  HubNode2,  HubNode3,  HubNode4,  aNode_fwd,
aNode_aft: TSpaceNode;
      HubFace1,   HubFace2,   InternalHubFace1,   InternalHubFace2,
AFTHubFaceUpPart1, AFTHubFaceUpPart2,
        AFTHubFaceDownPart1, AFTHubFaceDownPart2: TSpaceFace;
      FWDHubFaceUpPart1,  FWDHubFaceUpPart2,  FWDHubFaceDownPart1,
FWDHubFaceDownPart2: TSpaceFace;
      LocalNodesData, FWDNodesData, AFTNodesData: TSpaceNodeList;
    begin
      Result := TSpaceBody.Create(aSpace);
    try
        //       СОЗДАНИЕ       ВНЕШНЕЙ       ЧАСТИ       СТУПИЦЫ
================================
        // N_L := 10; // шаг по длине по умолчанию - нужен ли,
пока нигде не используем
        // базовые значения и переменные
        //
================================================================
========
```

```
        DeltaX := HubLen / (N_x - 1);
        HubRadius := HubDiam * 0.5;
        // начинаем расчеты
        SetLength(SectArr, N_x);
    try
          for iSect := 0 to N_x - 1 do
          begin
            SectX := HubCenterPosition + (-HubLen * 0.5) + iSect
* DeltaX;
            SectRad := GetHubRadius(SectX);
            // Списокузлов
            LocalNodesData := TSpaceNodeList.Create(False);
            for iAngl := 0 to N_fi - 1 do
            begin
              aNode := TSpaceNode.Create(aSpace);
              Fi := iAngl * DeltaFi;
              SinCos(Fi, aSin, aCos);
              aNode.Coordinates.SetCoords(SectX, SectRad * aSin,
SectRad * aCos);
              LocalNodesData.Add(aNode);
            end;
            SectArr[iSect] := LocalNodesData;
    end;
          // Теперь нужно построить поверхность, распределив
полученные ноды по Фейсам.
    for iSect := 0 to N_x - 2 do
          begin
            for iAngl := 0 to N_fi - 2 do
            begin
              HubFace1 := TSpaceFace.Create(aSpace);
              HubFace2 := TSpaceFace.Create(aSpace);
              HubFace1.SurfaceID := 0;
              HubFace2.SurfaceID := 0;
              HubNode1 := SectArr[iSect].Items[iAngl + 1];
              HubNode2 := SectArr[iSect].Items[iAngl];
              HubNode3 := SectArr[iSect + 1].Items[iAngl];
              HubNode4 := SectArr[iSect + 1].Items[iAngl + 1];
              aSpace.CreateTwoTriangles(HubNode1,        HubNode2,
HubNode3, HubNode4, HubFace1, HubFace2);
              Result.AddFace(HubFace1);
              Result.AddFace(HubFace2);
            end;
            HubFace1 := TSpaceFace.Create(aSpace);
            HubFace2 := TSpaceFace.Create(aSpace);
            HubFace1.SurfaceID := 0;
            HubFace2.SurfaceID := 0;
            HubNode1 := SectArr[iSect].Items[0];
            HubNode2 := SectArr[iSect].Items[N_fi - 1];
            HubNode3 := SectArr[iSect + 1].Items[N_fi - 1];
            HubNode4 := SectArr[iSect + 1].Items[0];
            aSpace.CreateTwoTriangles(HubNode1,          HubNode2,
HubNode3, HubNode4, HubFace1, HubFace2);
```

```
            Result.AddFace(HubFace1);
            Result.AddFace(HubFace2);
    end;
    //
================================================================
========
          Rad_fwd := ShDiamFWD * 0.5;
          Rad_aft := ShDiamAFT * 0.5;
          X_fwd := HubLen * 0.5 + HubCenterPosition;
          X_aft := -HubLen * 0.5 + HubCenterPosition;

          FWDNodesData := TSpaceNodeList.Create(False);
          AFTNodesData := TSpaceNodeList.Create(False);
          try
            for iAngl := 0 to N_fi - 1 do
            begin
              Fi := iAngl * DeltaFi;
              SinCos(Fi, aSin, aCos);
    {$MESSAGE 'Можно так построить логику, чтобы SinCos
выполнялся только 1 раз, потому что второй раз результаты точно
такие же'}
    aNode_fwd := TSpaceNode.Create(aSpace);
              aNode_fwd.Coordinates.SetCoords(X_fwd,  Rad_fwd  *
aSin, Rad_fwd * aCos);
              FWDNodesData.Add(aNode_fwd);

              aNode_aft := TSpaceNode.Create(aSpace);
              aNode_aft.Coordinates.SetCoords(X_aft,  Rad_aft  *
aSin, Rad_aft * aCos);
              AFTNodesData.Add(aNode_aft);
            end;
            for iAngl := 0 to N_fi - 2 do
            begin
              InternalHubFace1 := TSpaceFace.Create(aSpace);
              InternalHubFace2 := TSpaceFace.Create(aSpace);
              InternalHubFace1.SurfaceID := 0;
              InternalHubFace2.SurfaceID := 0;
              aSpace.CreateTwoTriangles(FWDNodesData[iAngl  +  1],
FWDNodesData[iAngl],
                AFTNodesData[iAngl],  AFTNodesData[iAngl  +  1],
InternalHubFace1, InternalHubFace2);
              Result.AddFace(InternalHubFace1);
              Result.AddFace(InternalHubFace2);
            end;
            InternalHubFace1 := TSpaceFace.Create(aSpace);
            InternalHubFace2 := TSpaceFace.Create(aSpace);
            InternalHubFace1.SurfaceID := 0;
            InternalHubFace2.SurfaceID := 0;
            aSpace.CreateTwoTriangles(FWDNodesData[0],
FWDNodesData[N_fi - 1], AFTNodesData[N_fi - 1],
              AFTNodesData[0],                InternalHubFace1,
InternalHubFace2);
```

```
            Result.AddFace(InternalHubFace1);
            Result.AddFace(InternalHubFace2);
            //
============================================================
========
            for iAngl := 0 to N_fi - 2 do
            begin
              FWDHubFaceUpPart1 := TSpaceFace.Create(aSpace);
              FWDHubFaceUpPart2 := TSpaceFace.Create(aSpace);
              FWDHubFaceUpPart1.SurfaceID := 1;
              FWDHubFaceUpPart2.SurfaceID := 1;
              aSpace.CreateTwoTriangles(SectArr[0].Items[iAngl],
SectArr[0].Items[iAngl + 1],
                AFTNodesData[iAngl + 1], AFTNodesData[iAngl],
FWDHubFaceUpPart1, FWDHubFaceUpPart2);
              Result.AddFace(FWDHubFaceUpPart1);
              Result.AddFace(FWDHubFaceUpPart2);
            end;
            FWDHubFaceDownPart1 := TSpaceFace.Create(aSpace);
            FWDHubFaceDownPart2 := TSpaceFace.Create(aSpace);
            FWDHubFaceDownPart1.SurfaceID := 1;
            FWDHubFaceDownPart2.SurfaceID := 1;
            aSpace.CreateTwoTriangles(SectArr[0].Items[N_fi - 1],
SectArr[0].Items[0], AFTNodesData[0],
              AFTNodesData[N_fi - 1], FWDHubFaceDownPart1,
FWDHubFaceDownPart2);
            Result.AddFace(FWDHubFaceDownPart1);
            Result.AddFace(FWDHubFaceDownPart2);
            //
============================================================
========
            for iAngl := 0 to N_fi - 2 do
     begin
              AFTHubFaceUpPart1 := TSpaceFace.Create(aSpace);
              AFTHubFaceUpPart2 := TSpaceFace.Create(aSpace);
              AFTHubFaceUpPart1.SurfaceID := 1;
              AFTHubFaceUpPart2.SurfaceID := 1;
              aSpace.CreateTwoTriangles(SectArr[N_x -
1].Items[iAngl + 1],
                SectArr[N_x - 1].Items[iAngl],
FWDNodesData[iAngl], FWDNodesData[iAngl + 1],
                AFTHubFaceUpPart1, AFTHubFaceUpPart2);
              Result.AddFace(AFTHubFaceUpPart1);
              Result.AddFace(AFTHubFaceUpPart2);
            end;
            AFTHubFaceDownPart1 := TSpaceFace.Create(aSpace);
            AFTHubFaceDownPart2 := TSpaceFace.Create(aSpace);
            AFTHubFaceDownPart1.SurfaceID := 1;
            AFTHubFaceDownPart2.SurfaceID := 1;
            aSpace.CreateTwoTriangles(SectArr[N_x - 1].Items[0],
SectArr[N_x - 1].Items[N_fi - 1],
```

```
              FWDNodesData[N_fi     -     1],     FWDNodesData[0],
AFTHubFaceDownPart1, AFTHubFaceDownPart2);
            Result.AddFace(AFTHubFaceDownPart1);
            Result.AddFace(AFTHubFaceDownPart2);
          finally
            FWDNodesData.Free;
            AFTNodesData.Free;
          end;
        finally
          // Interpol.Free;
          for i := 0 to Length(SectArr) - 1 do
            SectArr[i].Free;
          Finalize(SectArr);
        end;
      except
        Result.Free;
        Raise;
    end;
    end;

    Функция построения базовой лопасти
    procedure TPropeller.CreateBasicBlade(out aBody: TSpaceBody);
    var
      iSect: Integer;
      SectionData: TBladeSectionTable;
      Sect: TBladeSection;
      Mat: TTransformationMatrix;
      LocalSpace: TSpace;
      // i: Integer;
      NeedOffset: Boolean;
      // TestFile: TextFile;
      // ===
      // BodyGr: TBodyGraph;
      // Closed, Normal: Boolean;
    begin
      LocalSpace := aBody.Space;

      SectionData    :=    TBladeSectionTable.Create(True);    //
Создаемновуютаблицу
    try
        // Перенос всех данных сечений из TBladeSectionTable в
новую таблицу
    for iSect := 0 to FBladeSectionTable.Count - 1 do
        begin
          Sect := TBladeSection.Create;
          Sect.Assign(FBladeSectionTable[iSect]);
          SectionData.Add(Sect);
        end;

        NeedOffset := True;
    // передаем в процедуру интерполяции сечений новую таблицу
    InterpolateSections(SectionData, NeedOffset);
```

```
        // просчитываем OutlineOffset длявсехсечений
        if NeedOffset then
          for iSect := 0 to SectionData.Count - 1 do
            SectionData[iSect].FOutlineOffset                    :=
OffsetByBladeFillet(SectionData[iSect].FrR)
        else
          for iSect := 0 to SectionData.Count - 1 do
            SectionData[iSect].FOutlineOffset := 0;

        // Считаемкоординатыдлявсехсечений
        SectionData.CalculateXY(PropDi,     RakAng,     FRotation,
FSegmentCnt);

        CheckBladeFitHub(SectionData);
        // Считаемточки(Space) длявсехсеченийтаблицы
        SpaceAdaptationOfSectionTable(LocalSpace, SectionData);
        //
Заполняемнашетелолопаститочкамиизужеполностьюготовойтаблицысечений
        FillBodyFromSectionTable(LocalSpace, SectionData, aBody);
      finally
        SectionData.Free;
    end;
      // Базовая лопасть готова можем начинать работу с новыми
      {
        //Проверка на замкнутость и нормальность
        BodyGr := TBodyGraph.Create;
    BodyGr := aBody.CreateBodyGraph;
        Closed := aBody.IsClosed(BodyGr); = TRUE
        Normal := aBody.IsNormal(BodyGr); = TRUE
      }
      if PhaseAngle <> 0 then
      begin
        Mat.InitRotate(1, 0, 0, PhaseAngle { * pi / 180 } );
    // Матрица для поворота
        aBody.Transform(Mat);  //  Поворачиваем  первую  и  базовую
лопасть
      end;
    end;
    Функция построения всех остальных лопастей
    function TPropeller.CreateBodyWithAllBlades(aSpace: TSpace;
      SingleThreadBladesCopyInit:      Boolean      =      False;
SingleThreadBladesRotation: Boolean = False)
      : TSpaceBody;
    var
      T0: TAMTime;

      function MeasureTime: TAMTime;
      var
        T: TAMTime;
      begin
        T := GetTimeTPC();
```

```
        Result := CalcDeltaTimeTPCms(T, T0);
        T0 := T;
      end;

    var
      ClonedBodyList: TSpaceBodyList;

    begin
      if FBladeSectionTable.Count < 2 then
    begin
        // Если сечений у нас меньше двух, то визуализируем
параллелепипед
    // Result := TParallelepiped.Create(1, 1, 1, 2, 3, 4,
aSpace);
        exit;
    // Ступица все равно есть
      end;

    {$MESSAGE 'Конкретное указание TSpaceBody может быть
ограничением в дальнейшем. Практические задачи не должны решаться
на TSpaceBody'}
      // создаем базовую лопасть
      Result := TSpaceBody.Create(aSpace);
      try
        T0 := GetTimeTPC();

        // инициализировать базовую лопасть
    CreateBasicBlade(Result);
        FDuration_ms_CreateBlade := MeasureTime();
    if NumOfBlades > 1 then
        begin
          // Делаем клоны, а потом многопоточно поворачиваем их
каждый на свой угол
    ClonedBodyList := TSpaceBodyList.Create(False);
          try
            // Делаемклоны
            T0 := GetTimeTPC();
            Result.CloneStructure(aSpace,          ClonedBodyList,
NumOfBlades - 1, SingleThreadBladesCopyInit);
    FDuration_ms_CreateBladeCopies := MeasureTime();

            // Поворачиваем клоны многопоточно, потому что они
независимые
            // По факту многопоточность не приносит ощутимого
ускорения,
            // но теоретически для многолопастных винтов с
большой подробностью она должна сработать
    if ClonedBodyList.Count > 0 then
              TParallelFor(1, NumOfBlades - 1,
                function(_NewBladeNum: Integer; ThreadCacheData:
Pointer): Boolean
                var
```

```
                    _Body: TSpaceBody;
                    _Mat: TTransformationMatrix;
                    _BladeAngle: Double;

                  begin
                    Result := True;

                    // Клон
                    _Body := ClonedBodyList[_NewBladeNum - 1];
                    if not Assigned(_Body) then
      exit;

                    // Угол, на который его нужно повернуть
                    _BladeAngle := _NewBladeNum * 2 * PI /
NumOfBlades;
                    // Применить преобразование
                    _Mat.InitRotate(1, 0, 0, _BladeAngle);

                    //  _Body.Transform(_Mat); // Внутри создается
список и убивается, а это долго
                    _Body.TransrormNodes(_Mat, _Body.CachedNodes);
                    // Используем список, который был заполнен при
клонировании
                  end, nil, SingleThreadBladesRotation);

              FDuration_ms_RotateBladeCopies := MeasureTime();
            finally
              // Хранилище клонов больше не нужно
              ClonedBodyList.Free;
            end;
          end;
          // Все тела присоединяем к первому, возможно, к ступице
          T0 := GetTimeTPC();
          while aSpace.BodyCount > 1 do
            Result   :=   aSpace.CombineTwoBodies(aSpace.Bodies[0],
aSpace.Bodies[1]);
          FDuration_ms_CombineBlades := MeasureTime();

        except
          Result.Free;
          Raise;
        end;
      end;
```

ДОДАТОК В. ОПИС ПРОГРАМИ

При наступних вхідних характеристиках

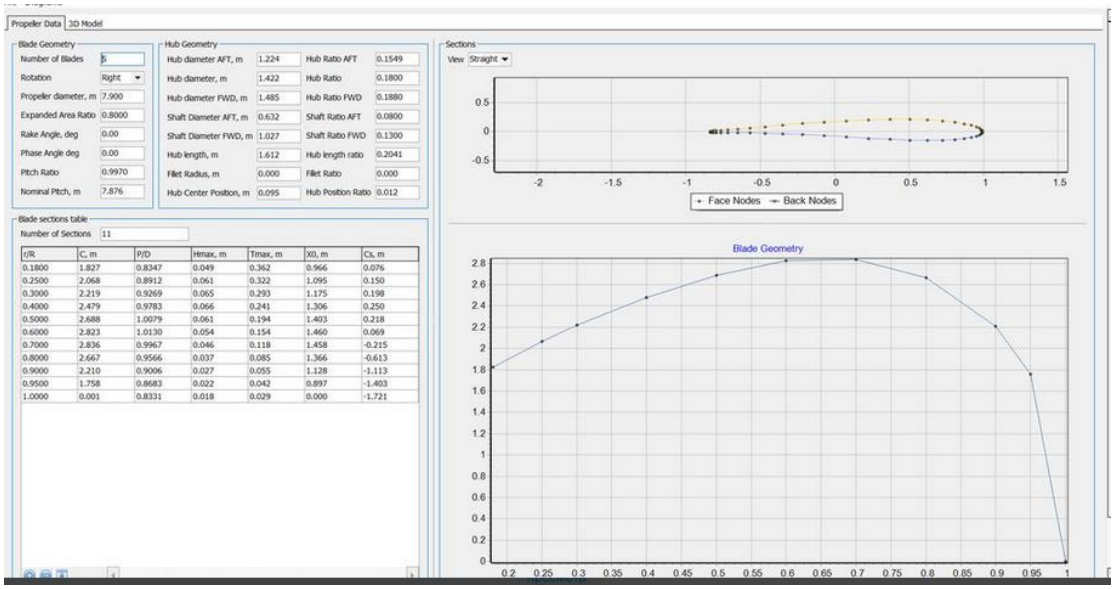


Рисунок В.1 – Вхідні характеристики гвинта

Отримуємо наступну тривимірну модель:

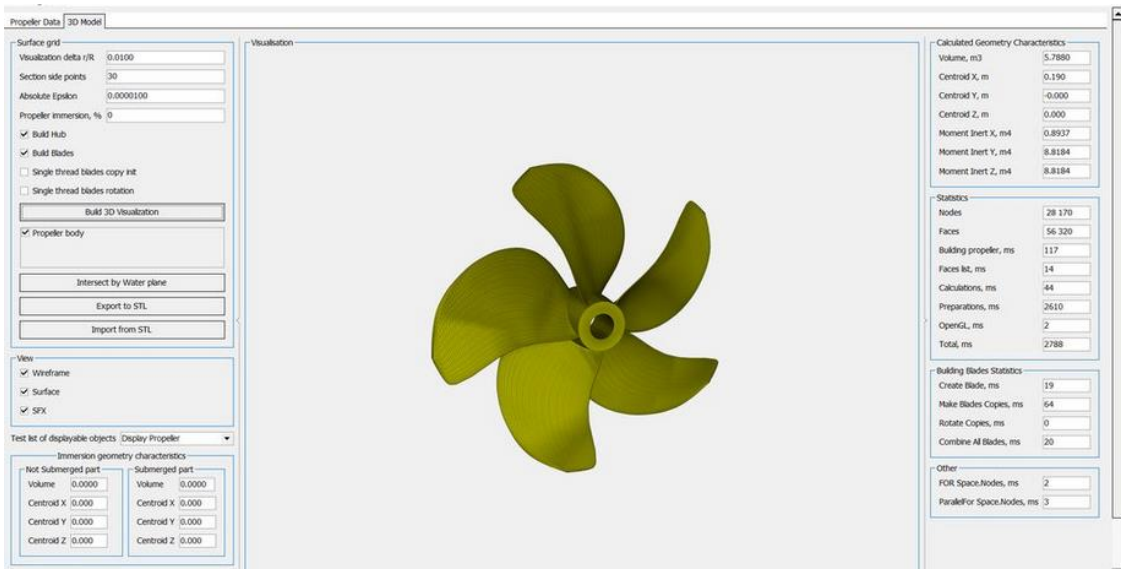


Рисунок В.2 – Тривимірна візуалізація гвинта

При наступних вхідних характеристиках

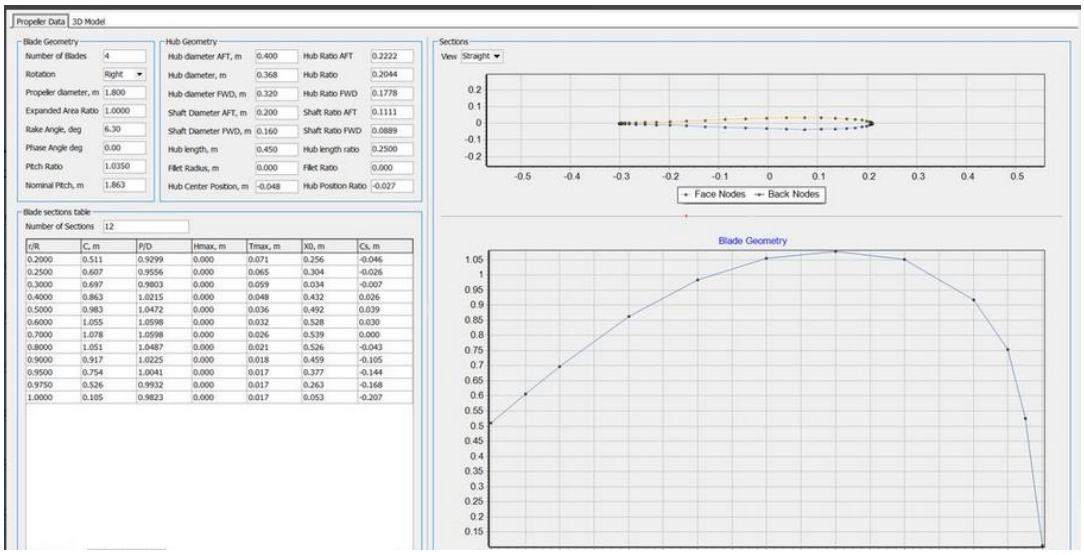


Рисунок В.3 – Вхідні характеристики гвинта

Отримуємо наступну тривимірну модель:

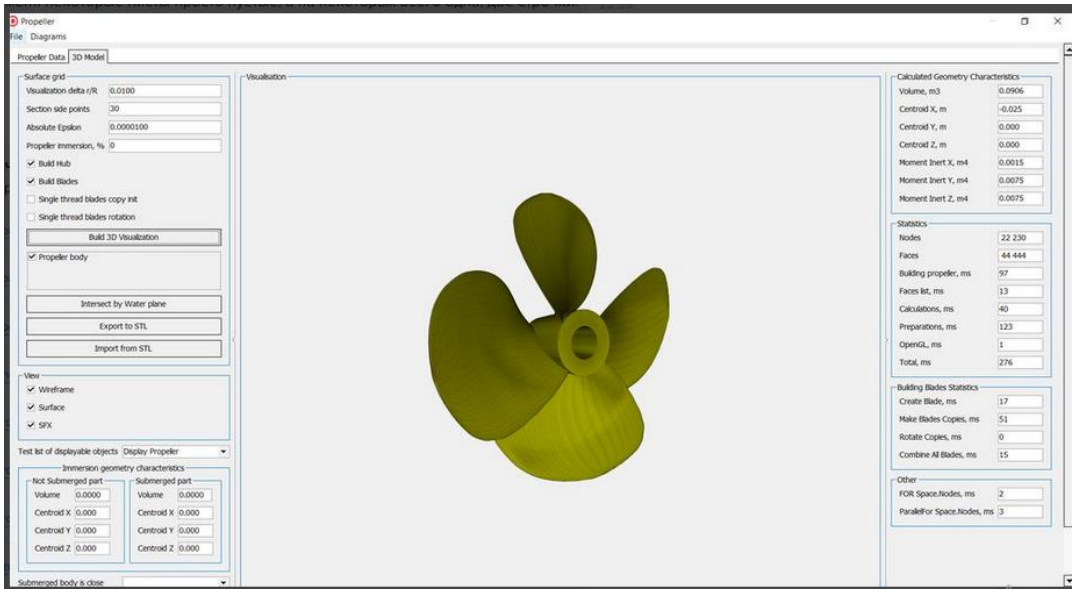


Рисунок В.4 – Тривимірна візуалізація гвинта

При додаванні рівня води модель буде перерізана на тому рівні, який вказано користувачем при побудові:

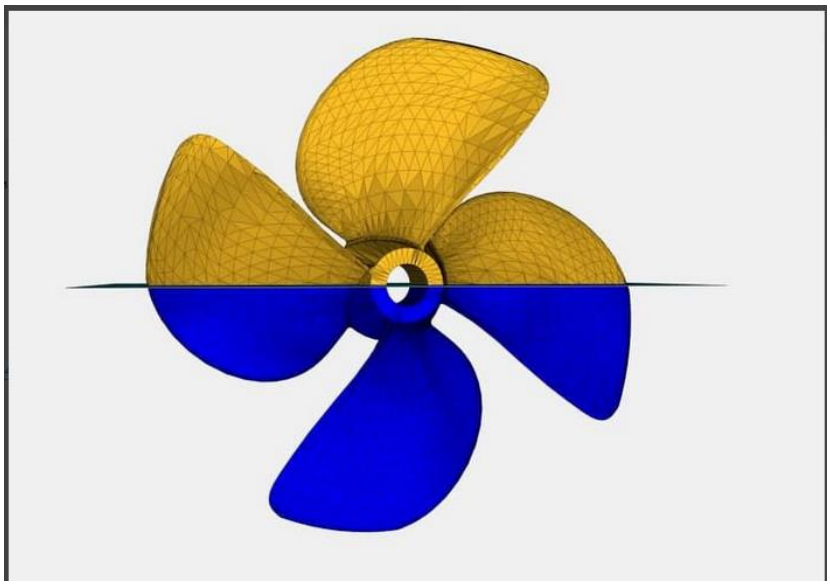


Рисунок В.5 – Тривимірна візуалізація гвинта при перерізі водною гладдю

Була додана можливість розглянути переріз водною гладдю з різних боків, навіть приховати самі частини перерізаного гвинта (Рисунок 3.15) та отримати геометричні характеристики кожної с частин гвинта (Рисунок 3.16).

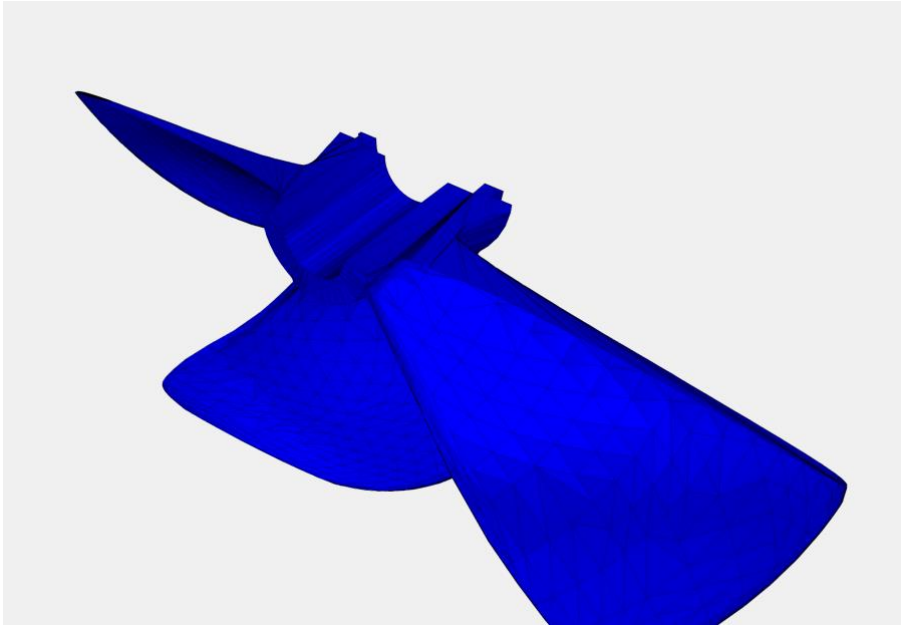


Рисунок В.6 – Перерізана «занурена» частина гвинта

Dry part	
Volume, m3	0.0339
Centroid X	0.0343
Centroid Y	0.0040
Centroid Z	0.2038
Mass, kg	0.0339
Submerged part	
Volume, m3	0.0339
Centroid X	0.0342
Centroid Y	-0.0040
Centroid Z	-0.2038
Mass, kg	0.0339

Рисунок В.7 – Геометричні характеристики частин розрізаного гвинта

ДОДАТОК Г. ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску програми користувач бачить перед собою першу вкладку програми.

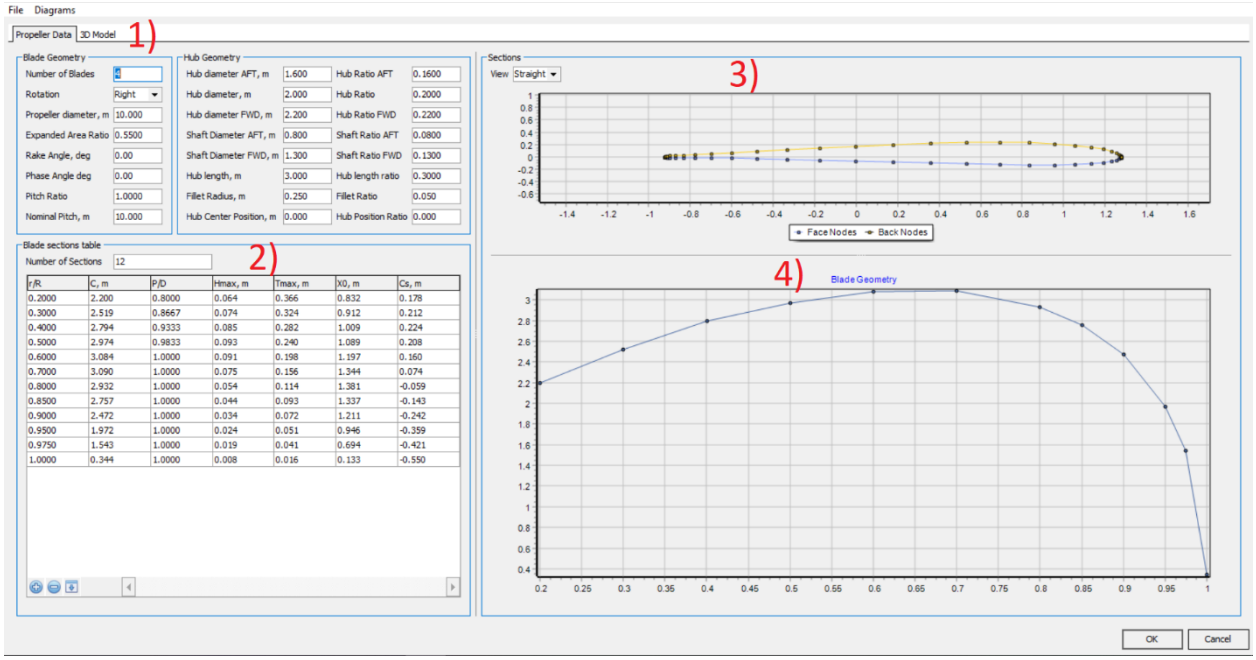


Рисунок Г.1 – Робота програми

Дві групи заповнюваних полів які відповідають за основні геометричні характеристики: BladeGeometry зберігає в собі вступні дані щодо геометрії лопаті гвинта і HubGeometry зберігає в собі вступні дані по геометрії маточини. Вони заповнені даними за замовчуванням які користувач може змінити за власним розсудом.

Таблиця перетинів яка переноситься в програму з креслення лопаті або з інших джерел. Так само заповнена за замовчуванням даними пересічний стандартного гвинта B-Series. У таблиці є три функції: 1. Додати нове перетин - інтерполює два сусідніх перерізів і створює нове з інтерпольованого даннимі.2. Видалити перетин - видаляє перетин з табліци.3. Заповнити таблицю перетинами за замовчуванням - видаляє всі нинішні перетину в таблиці і заповнює її перетинами гвинта B-Series по замовчуванням.

Графік який пов'язаний з таблицею перетинів і в реальному часі показує Вибраний в таблиці розтин. При зміні даних розтину ці зміни будуть наочно видно на графіку.

Графік наочно відображає динаміку зміни значень за всіма перетинах (показується обраний стовпчик перетинів)

Всі введені дані можуть бути як збережені в форматі XML так і знову завантажені з нього в програму. Це дозволяє створювати заготовки для різних гвинтів, без необхідності постійного ручного введення.

При необхідності користувач може викликати модульне вікно PropellerBladeDescription, у вкладці Diagrams, де знаходиться картинка з схематичним кресленням лопаті гвинта, де відображені всі вступні параметри. Це зручний спосіб щоб споживач не надто знайомий з кресленнями і де на них зображені необхідні йому параметри зміг зорієнтуватися і успішно перенести їх в програму.

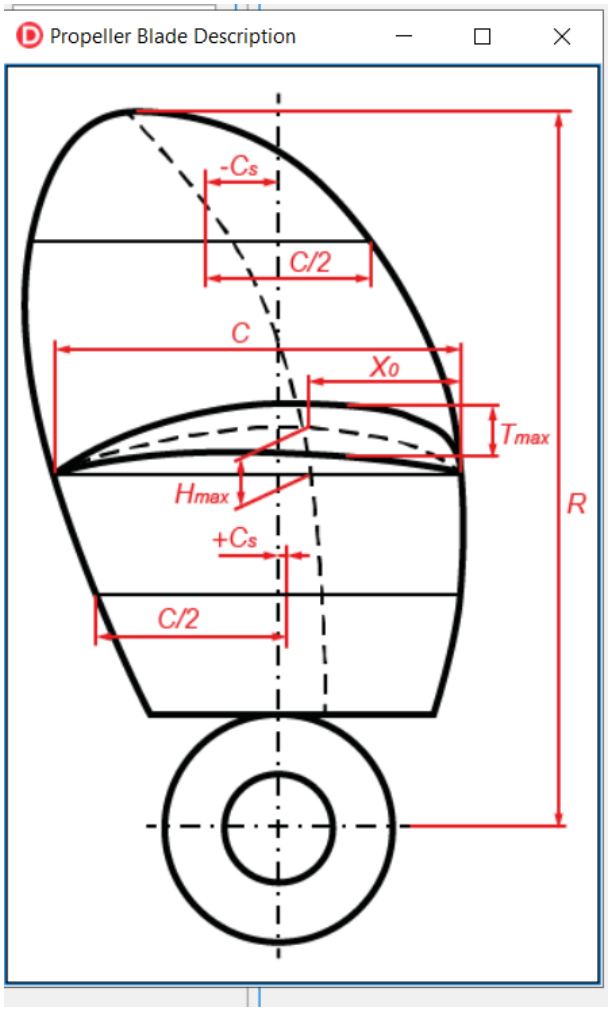


Рисунок Г.2 – Робота програми

При переході на друге вікно користувач повинен відразу звернути увагу на панель зліва.

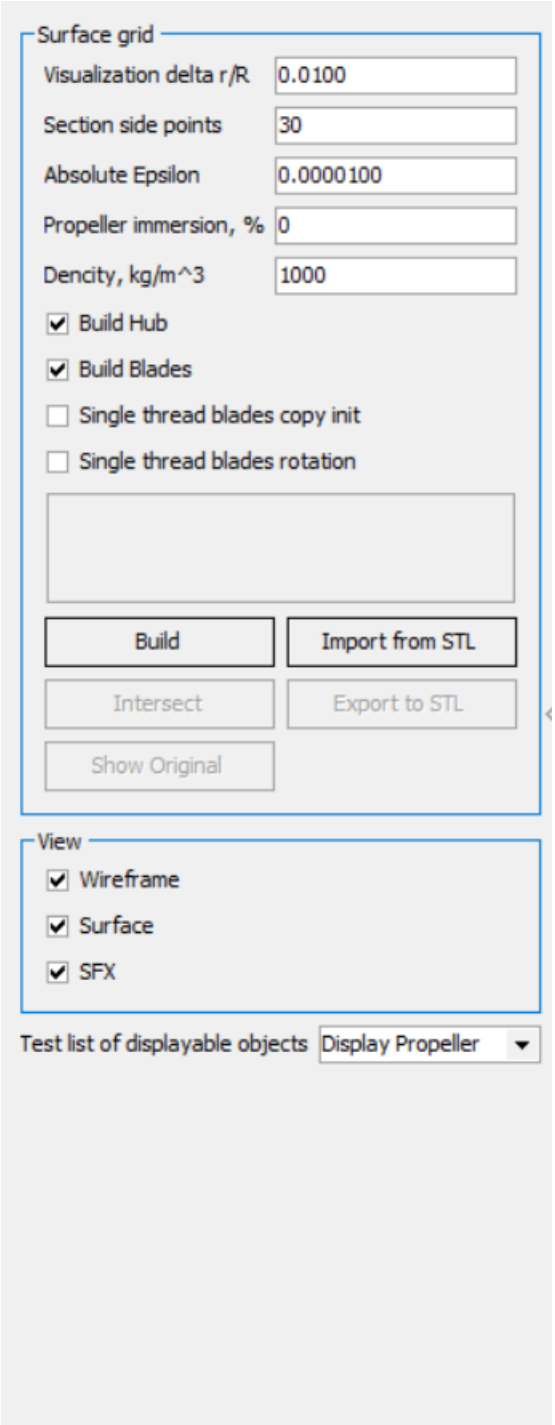


Рисунок Г.3 – Робота програми

Це основна панель побудови тривимірної моделі гвинта. Тут він може вибрати деталізацію моделі налаштувавши параметри: 1) Visualization delta r / R - проміжок між перетинами, чим він менший тим більше детальна модель.

2) Section side points - кількість полігонів на одній стороні перетину, чим їх більше тим більше деталізація і більш плавним буде гвинт.

3) BuildHub і BuildBlades - check-box дають змогу вказати будувати маточину або лопаті. Якщо є необхідність розглянути окремо ту чи іншу частину гвинта, можна заздалегідь це вказати.

4) Поки модель не побудована для взаємодії є тільки 2 кнопки: Buildi ImportfromSTL. Програма підтримує як базове побудова гвинта можливих форм на першій вкладці параметрам, так і можливість просто завантажити вже існуючу модель з формату STL, до того ж не обов'язково модель гвинта. Імпортувати можна будь-яку модель, наприклад корпус корабля або просто куб.

5) Check-boxes Wireframe, Surface, SFX- кожен із них відповідає за відображення того чи іншого елемента моделі і може міняти модель в реальному часі відключаючи сітку полігонів, полігони або візуальні ефекти на моделе.

Після побудови моделі або імпорту, панель виглядає ось так.

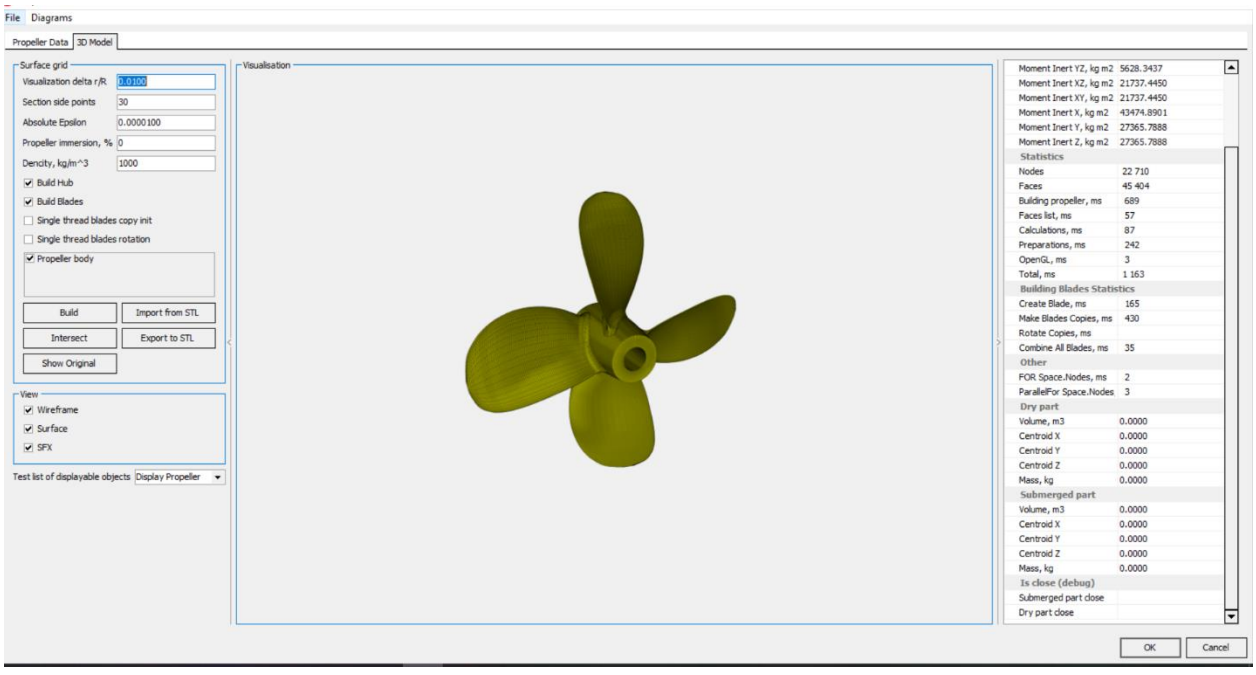


Рисунок Г.4 – Робота програми

Посередині екрана в зоні Visualisation будується модель гвинта яку можна обертати, наближати, віддаляти і розглядати з усіх ракурсів.

Стають доступні такі кнопки як:

2.1 Intersect- розрізає модель у відсотку який вказаний в полі Propellerimmersion візуалізує таким чином занурення гвинта в воду і його розсічення водною гладдю.

2.2 ExportSTL -Дозволяє експортувати побудовану модель в формат STL для подальшого використання в інших програмах або можливої завантаженні в інший сесії роботи з програмою.

2.3 ShowOriginal - дозволяє скасувати всі зміни в моделі і показати її первісний вигляд.

3) Справа ми можемо побачити геометричні характеристики побудованої моделі

Calculated Geometry Characteristics	
Volume, m3	12.4842
Centroid X, m	0.198
Centroid Y, m	0.000
Centroid Z, m	0.000
Mass, kg	12484.1584
Moment Inert YZ, kg m2	5628.3437
Moment Inert XZ, kg m2	21737.4450
Moment Inert XY, kg m2	21737.4450
Moment Inert X, kg m2	43474.8901
Moment Inert Y, kg m2	27365.7888
Moment Inert Z, kg m2	27365.7888
Statistics	

Рисунок Г.5 – Робота програми

При розсічення гвинт буде виглядати ось так:

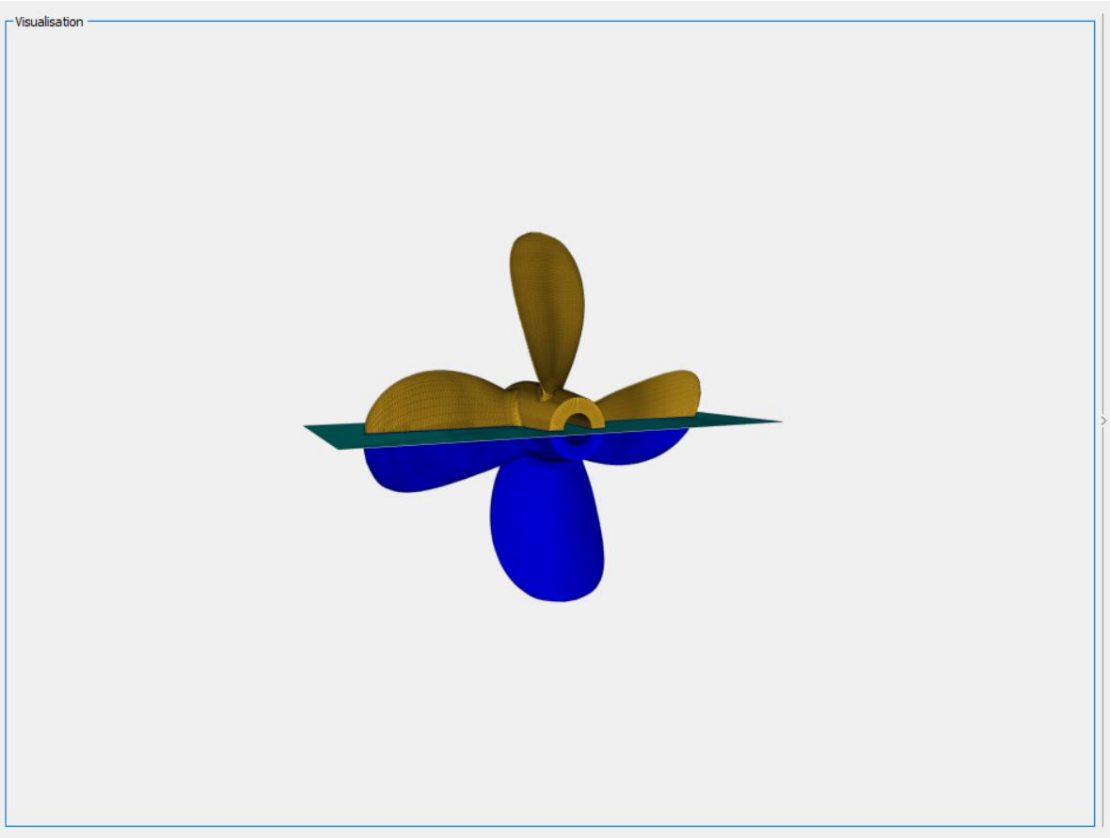


Рисунок Г.6 – Робота програми

Отримаємо так само геометричні характеристики кожної з частин гвинта в правій частині екрана.

Dry part	
Volume, m3	6.2421
Centroid X	0.1983
Centroid Y	0.1373
Centroid Z	0.9270
Mass, kg	6242.0792
Submerged part	
Volume, m3	6.2421
Centroid X	0.1983
Centroid Y	-0.1373
Centroid Z	-0.9270
Mass, kg	6242.0792
Is close (debug)	
Submerged part close	Yes
Dry part close	Yes

Рисунок Г.7 – Робота програми

ДОДАТОК Д. ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є необхідною частиною розробки продукту. За допомогою послідовного виконання тестів можна перевірити функціональну поведінку програми, зручність використання та правильність роботи функціоналу.

Перевагами тестування ПЗ є:

- знаходження та виправлення помилок до того, як продуктом почне користуватись аудиторія;
- мінімізація технічних втрат та швидкий розвиток процесу розробки через виконання тестування на декількох стадіях створення;
- оптимізація коду.

Розробка комп'ютерних програм має свої привілеї, так як не потрібно піклуватись про розмір екранів та через більшу потужність та швидкість обробка даних проходить швидше порівнюючи з іншими пристроями. Проте з іншої сторони це не означає, що процес тестування дуже простий.

В першу чергу потрібно проаналізувати та виконати тести роботи програмного забезпечення на різних операційних системах. Потім перевірити роботу з різними версіями інших програмних продуктів.

При дослідженні процесу тестування програми порівняння також необхідно ретельно перевірити роботу функціоналу, тому що продукт працює напряму з документами та будь-які некоректні зміни можуть привести до певних наслідків.

Враховуючи аналіз проведення тестування комп'ютерних програм та усі складності його виконання, робота розробки продукту може йти по графіку або навіть прискоритись, кінцевий результат надасть змогу користувачам без проблем використовувати продукт.

Тестування виконується зазвичай на декількох етапах життєвого циклу, коли розробляється програмне забезпечення. Технологія тестування програмного забезпечення містить такі етапи:

- визначення функціоналу, що підлягає та не підлягає тестуванню;
- формулювання підходів, що будуть використовуватись для даного продукту;
- написання тест кейсів;
- розробка критерію проходження тестів;
- визначення вимог середовища проведення тестування;
- проведення тестування та оцінка результатів;
- звітність результатів. Системні вимоги:
- операційна система Windows 7 або наступний версій;
- наявність програми Microsoft Word версії 2007 або наступних;
- наявність програми Adobe Acrobat Reader DC версії 5.1 або наступних.

Апаратні вимоги:

- достатньо вільної пам’яті комп’ютері для встановлення програми;
- 32 або 64-розрядний процесор з тактовою частотою 1GHz;
- оперативна пам’ять не менше 1Gb.

Тестування проводилось на ноутбуці компанії DELL з процесором Intel® Core i3-6060U, тактовою частотою 2GHz, операційною системою Windows 10 та при наявності програми Microsoft Word 2016 та програми Adobe Acrobat Reader DC версії 5.1.

Під час тестування даного програмного забезпечення було проведено перевірку усіх функціональних вимог та варіантів використання, враховано роботу інтерфейсу та перевірено усі запобіжні функції від некоректних дій користувача. Результати тестів наведені в таких таблицях:

- завантаження характеристик гвинта (таблиця Д.1);
- перевірка роботи при побудові гвинта (таблиця Д.2);
- візуалізація гвинта (таблиця Д.3).

Таблиця Д.1 – Завантаження характеристик гвинта

Мета тесту	Перевірити коректність вводу вхідних характеристик гвинта
Початковий стан	Програма запущена. Видно інтерфейс користувача.
Вхідна дані	Ввведені
Проведення тесту	Ввести вхідні дані в форму
Очікуваний результат	В текстових полях відображена введена інформація
Фактичний результат	В текстових полях відображена введена інформація

Таблиця Д.2 – Перевірка роботи при побудові гвинта

Мета тесту	Перевірити роботу програми при побудові гвинта
Початковий стан	В текстових полях відображена введена інформація
Вхідна дані	Ввведені
Проведення тесту	Натиснути кнопку Виконати обчислення
Очікуваний результат	В текстових полях відображені обчислені характеристики гвинта
Фактичний результат	В текстових полях відображені обчислені характеристики гвинта

Таблиця Д.3 – Візуалізація гвинта

Мета тесту	Візуалізувати гвинт
Початковий стан	Програма запущена. Видно інтерфейс користувача.
Вхідна дані	Ввведені

Проведення тесту	Натиснути кнопку Зображення
Очікуваний результат	Поява тривимірного зображення гвинта
Фактичний результат	Поява тривимірного зображення гвинта

Досліджено аналіз якості програмного забезпечення. Виявлено, що для даного продукту потрібно протестувати роботу інтерфейсу користувача та функціональні вимоги, так як обробка документів повинна видавати найбільш точні результати.

Окрім цього вказано опис процесу тестування, сформульовані системні та апаратні вимоги, що є необхідними для проведення даного етапу розробки продукту. Наведені вимоги до технічного та програмного забезпечення, на якому будуть проводитися тести.

Також детально описані проведені тести-кейси у вигляді таблиць. Вказано інформацію про мету тестування, початковий стан програми, вхідні дані, вказані кроки проведення тестів, а також очікуваний та фактичний результат. Проводилось тестування усіх функціональних вимог програмного забезпечення та робота виведення повідомлень на некоректні дія користувача.