Київський національний університет імені Тараса Шевченка
Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

Осіпьонок Максим Миколайович

УДК 004.021: 004.043: 004.422.639

ДИСЕРТАЦІЯ
РОЗРОБКА МОДЕЛІ ЄДИНОГО АЛГОРИТМІЧНОГО СЕРЕДОВИЩА ДЛЯ РОЗВ'ЯЗАННЯ КОМПЛЕКСУ ЗАДАЧ АДИТИВНОГО ВИРОБНИЦТВА

12 Інформаційні технології
122 Комп'ютерні науки

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ М. М. Осіпьонок

Науковий керівник Терещенко Василь Миколайович
доктор фізико-математичних наук, професор

Київ - 2026

АНОТАЦІЯ

Осіпьонок М.М. **Розробка моделі єдиного алгоритмічного середовища для розв'язання комплексу задач адитивного виробництва.** – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 - Комп'ютерні науки. – Київський національний університет імені Тараса Шевченка, Київ, 2026.

Дисертаційна робота присвячена розробці концепції моделі єдиного алгоритмічного середовища (МЄАС) для розв'язування комплексу алгоритмічних задач, що постають в процесу планування адитивного виробництва (АВ), тобто під час підготовки однієї або кількох цифрових тривимірних моделей до 3D-друку. Увага акцентується на наступному наборі задач, які постають при використанні більшості існуючих технологій АВ: генеруванні мешів, виборі оптимальної орієнтації для моделей та їх розміщення всередині робочої області принтера, генерування опорних конструкцій, слайсинг та побудова траєкторії друку.

Сфера застосування адитивного виробництва в сучасному світі поступово розширюється з інструменту швидкого прототипування до повноцінної компоненти складних виробничих ланцюжків. Перевагою АВ є здатність долати обмеження традиційних методів виробництва, таких як лиття або фрезування, шляхом створення деталі з внутрішніми каналами ти сітчастими структурами, в результаті чого надруковані деталі міцнішими при меншій вазі. В таких індустріях промисловості як автомобілебудування та аерокосмічна галузь, застосування деталей виготовлених шляхом адитивного виробництва дозволяє збільшити вантажопідйомність та знизити витрати пального. Сучасні геополітичні виклики та логістичні кризи підкреслюють ще одну перевагу АВ – можливість використання моделі «виробництво на вимогу», за якої створення необхідної деталі відбувається за лічені години локально в місці її використання, що є критично важливим у військовій справі, енергетиці та медицині.

На сьогодні існує велика кількість різноманітних технологій АВ, що орієнтовані на друк різними матеріалами – пластиком, керамікою, металами тощо. В основу цих технологій покладені різні фізичні явища, однак об'єднує їх процес планування АВ, що неодмінно передує безпосередньому друку моделі. Різні технології АВ накладають різні вимоги та обмеження щодо вхідних даних, тому розв'язання задачі планування АВ не може бути зведене до єдиного універсального алгоритму.

Програмні рішення, що існують на сьогодні, можна розділити на три категорії: відкриті системи масового сегмента (наприклад Cura та PrusaSlicer), промислові платформи (наприклад Autodesk Netfabb та Siemens NX) та пропрієтарні рішення від виробників обладнання (наприклад Formlabs PreForm та EOS Build).

Представники першої категорії є відкритими програмними продуктами та мають відносно низький поріг входу, що зробило їх стандартом для настільних 3D-принтерів аматорського сегмента. В межах своєї спеціалізації ці системи пропонують повноцінний робочий цикл, від завантаження моделі до генерування зрозумілих принтеру інструкцій, однак, незважаючи на відкритість вихідного коду, будь-яка суттєва модифікація вимагає високих навичок у програмуванні та глибокого розуміння ядра системи, в наслідок чого при розробці нових алгоритмів розв'язання задач АВ часто простішим є створення власних скриптів, а не адаптація архітектури існуючих програмних рішень.

Перевагою класу промислових платформ є широкий набір інструментів виправлення дефектів та покращення якості, гнучкі алгоритми пакування, а також нативна інтеграція з модулями термомеханічного аналізу. Але, з точки зору архітектури програмного забезпечення, такі системи є яскравим прикладом пакетно-алгоритмічного підходу, за якого функціональні модулі будуються з використанням різних, часто несумісних між собою бібліотек. Промислові платформи зазвичай дозволяють користувачеві забезпечити певний рівень автоматизації рутинних задач (наприклад, шляхом написання скриптів мовою

Python), ці можливості обмежуються автоматизацією стандартних дій в інтерфейсі.

Пропрієтарні рішення від виробників принтерів орієнтуються на автоматизацію базового процесу використання обладнання. В такому програмному забезпеченні більшість параметрів друку прихована від користувача заради стабільності результату та простоти експлуатації. Це дозволяє досягти високої повторюваності виробництва без глибоких знань технологічного процесу, проте повністю позбавляє систему універсальності та забороняє її використання з принтерами від інших виробників, а будь-яке розширення функціоналу з боку користувача є неможливим.

З наведеного огляду існуючих рішень видно, що жодне з них не є таким, яке б забезпечувало процес планування АВ від початку до кінця для будь-якої технології 3D-друку, зберігаючи при цьому властивості розширюваності та масштабованості. Тому актуальною на сьогодні проблемою є розробка архітектури мультиалгоритмічної платформи, яка б дозволяла ефективно розв’язувати послідовність задач планування АВ та була б відкритою до розширення.

Враховуючи велику кількість геометричних задач, що постають в процесі планування АВ, фундаментом розв’язання яких є алгоритми та задачі обчислювальної геометрії та комп’ютерної графіки, найбільш доцільним та перспективним шляхом побудови мультиалгоритмічного середовища з зазначеними властивостями є застосування моделі єдиного алгоритмічного середовища (МЄАС), оскільки використання МЄАС вже дозволило отримати значний приріст ефективності під час розв’язання взаємопов’язаних задач обчислювальної геометрії: побудова діаграми Вороного та тріангуляції Делоне, пошук найближчого сусіда тощо.

Однак, безпосереднє застосування МЄАС на основі теорії звідності в якості архітектурного ядра системи підтримки процесу планування АВ не дає бажаного ефекту, оскільки зазначений комплекс задач, алгоритми їх розв’язання

та структури даних, на яких вони базуються не є взаємопов'язаними на формальному рівні. Тому в цій дисертаційній роботі вперше пропонується розширити концепцію МЄАС шляхом введення інструментів керування обчисленнями на рівні архітектури середовища: менеджера структур даних (МСД) та неявним зведенням репрезентацій даних.

МСД є основним елементом запропонованої архітектури МЄАС, що відповідає за зберігання та синхронізацію спільних структур даних для різних алгоритмів. В процесі виконання алгоритми розв'язання задач процесу планування АВ не будують необхідні структури даних, а отримують їх від МСД. Якщо необхідна структура даних вже була побудована та знаходиться в актуальному стані, алгоритм може її використати одразу, якщо ні – то, МСД спершу будує або актуалізує необхідну структуру даних, зберігає її, і лише після цього повертає її алгоритму.

Робота МСД базується на введеній класифікації алгоритмів за характером змін, що вносяться в геометрію, над якою вони працюють. *Немодифікаційні* алгоритми не змінюють геометрію чи структури даних (наприклад, пошук найближчого сусіда), *атрибутивні* алгоритми додають, модифікують або видаляють властивості геометричних об'єктів або їх складових частин (наприклад, призначення кольору вершини трикутника), *топологічно-тріангуляційні* алгоритми виконують топологічні зміни (наприклад, перебудова тріангуляції), *геометричні* алгоритми вносять зміни в геометричне місце точок, що описує об'єкт, над яким вони оперують (наприклад, побудова більш органічного мешу за рахунок заокруглення кутів), *просторово-трансформаційні* алгоритми впливають лише на матрицю трансформації об'єкта (наприклад, поворот або перенесення об'єкта в просторі) та *комплексні* (або *радикальні*) алгоритми, що здійснюють суттєві перетворення структури моделі. Кожен алгоритм в МЄАС в явному вигляді декларує, до яких класів змін він належить, завдяки чому МСД може слідкувати за актуальністю наявних структур даних.

Також, в межах запропонованої архітектури, ядро МЄАС відповідає за автоматичне визначення оптимального шляху отримання необхідної репрезентації даних. При цьому МЄАС виконує необхідні проміжні обчислення або трансформації даних «за кадром», тому розробнику алгоритму непотрібно власноруч слідкувати за способом отримання потрібних даних.

В роботі проведено огляд основних актуальних підходів до розв'язання зазначеного вище комплексу задач процесу планування АВ. При цьому, однією з ключових вимог до архітектури МЄАС є масштабованість системи, зокрема, шляхом додавання нових задач та алгоритмів, тому функціональні можливості запропонованої моделі єдиного алгоритмічного середовища не обмежується наведеним вище переліком задач. Також, з метою забезпечення додаткової гнучкості системи, окремо розглянуто загальні алгоритми генерування мешів на основі функції відстані зі знаком та алгоритми виконання булевих операцій над мешами, які дозволяють реалізовувати прикладні задачі процесу планування АВ, які є актуальними лише для деяких технологій друку або спеціалізованих робочих процесів (наприклад, додавання відступу до моделі).

Окремо акцентується увага на структурах даних, що використовуються розглянутими алгоритмами в процесі роботи, а також проаналізовано, за який рахунок інтеграція цих алгоритмів в єдиній системі на базі запропонованої архітектури МЄАС дозволяє прискорити загальний час їх виконання.

Також в дисертації досліджено підхід до побудови ефективних алгоритмів розв'язання деяких задач обчислювальної геометрії з використанням графу Ріба (ГР) для плоского многокутника. Зокрема, в роботі запропоновано новий алгоритм побудови декомпозиції довільного плоского многокутника на набір монотонних многокутників з одночасною їх тріангуляцією. Встановлено, що за умови наявності ГР в МСД, обчислювальна складність розробленого алгоритму складає $O(n)$, де n – кількість вершин многокутника, що є ефективнішим за існуючі алгоритми тріангуляції. Показано, що використання застосування ГР

дозволяє оптимізувати задачі процесу планування АВ при роботі на рівні слайсів моделі.

Розроблено алгоритм перевірки подібності многокутників, який працює за лінійний час та використовує константний обсяг додаткової пам'яті, на основі якого запропоновано новий підхід до врахування подібності сусідніх слайсів моделі в алгоритмах, що виконують процедуру декомпозиції многокутника. Також, в роботі запропоновано алгоритм відновлення декомпозиції многокутника на основі існуючої декомпозиції для подібного многокутника. В сукупності, ці два алгоритми дозволяють уникати повторну побудову декомпозиції або тріангуляції для однакових або дуже схожих контурів моделі, що забезпечує економію часу при обробці великої кількості послідовних слайсів.

Ефективність запропонованих алгоритмів як складових компонентів МЄАС підтверджено практичними експериментами. Швидкість роботи процедури декомпозиції довільного многокутника на монотонні з одночасною їх тріангуляцією було порівняно з найбільш вживаними бібліотеками мови програмування C++, які містять алгоритми побудови тріангуляції: CGAL, PolyPartition та Geometric Tools. Тестування, проведене на многокутниках з кількістю вершин від 10^2 до 10^7, показало, що запропонований алгоритм в поєднанні з МЄАС в середньому є в 2-3 рази швидшим за PolyPartition, в залежності від розміру вхідного многокутника, та щонайменше в 5 разів швидшим за інші розглянуті бібліотеки. Експериментальні дослідження оптимізації процедури декомпозиції многокутника шляхом врахування подібності сусідніх слайсів були проведені для задачі декомпозиції довільного многокутника на опуклі, що часто використовується в алгоритмах планування траєкторії друку в адитивному виробництві. Показано, що застосування такої оптимізації дозволяє скоротити час виконання декомпозиції, причому, зі збільшенням кількості слайсів ефект від використання оптимізації стає більш помітним. Наприклад, час обробки 2000 слайсів моделі Stanford Bunny

зменшився 32.2%, а час обробки 4000 слайсів аналогічної моделі зменшився на 47.8%, тобто операція була виконана майже вдвічі швидше.

Ключові слова: 3D-моделювання, адитивне виробництво, код, композиція, математична модель, математична оптимізація, множина даних, модель єдиного алгоритмічного середовища, обчислювальна геометрія, оптимізація, параметрична оптимізація, представлення поверхні, просторові задачі, регуляризація, швидкість обробки.

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

1. Tereshchenko, V., & Osiponok, M. (2024). Process planning in additive manufacturing: a review of problems and methods of their solution. *Bulletin of Taras Shevchenko National University of Kyiv. Series: Physics and Mathematics*, (1), 128–136. https://doi.org/10.17721/1812-5409.2024/1.24 (Scopus)
2. Tereshchenko, V., Prishlyak, A., & Osiponok, M. (2025). Constructing Efficient Algorithms for Solving Certain Computational Geometry Problems Using Reeb Graphs. *Cybernetics and Systems Analysis, 61*(6), 924–933. https://doi.org/10.1007/s10559-025-00825-4 (Scopus)
3. Osiponok, M., & Tereshchenko, V. (2025a). Fast identification of similar polylines as an optimization technique in computational geometry problems. *Information Technology: Computer Science, Software Engineering and Cyber Security*, (4), 178–185. https://doi.org/10.32782/it/2025-4-21
4. Осіпьонок, М. (2023). *Актуальність застосування моделі єдиного алгоритмічного середовища до комплексу задач адитивного виробництва.* Proceedings of the 2nd International Scientific and Practical Internet Conference «Way Science», April 3-4, 2023, Dnipro, Ukraine (pp. 288-289).
5. Osiponok, M., & Tereshchenko, V. (2023). *A review of the approaches to solve the nesting problem in additive manufacturing.* III International Scientific Symposium "Intelligent Solutions" (Satellite) Proceedings, September 28, 2023, Kyiv, Ukraine (pp. 97-100).

6. Осіпьонок, М. (2024). *Розвиток адитивного виробництва: технологічні та соціальні аспекти*. Матеріали XXIX Всеукраїнської наукової конференції молодих істориків науки, техніки і освіти та спеціалістів за темою: «наука для відбудови України», 19 квітня 2024, Київ, Україна (с. 182-184).
7. Osiponok, M., & Tereshchenko, V. (2025b). *Designing a model of a unified algorithmic environment for AM process planning*. Proceedings of XXIII International Scientific – Practical Conference «Shevchenkivska Vesna – 2025», April 10, 2025, Kyiv, Ukraine (p. 105).
8. Osiponok, M., & Tereshchenko, V. (2025c). *On incremental construction of polygon subdivisions.* Information Technologies and Security. Book of Extended Abstracts of the XXV International Scientific and Practical Conference ITS-2025, December 11, 2025, Kyiv, Ukraine (pp. 232-235).

ANNOTATION

Osiponok M.M. **Development of a model of a unified algorithmic environment for solving a complex of additive manufacturing problems.** – Manuscript.

Dissertation thesis for acquiring the degree of a Doctor of Philosophy by specialty 122 - Computer Science. - Taras Shevchenko National University of Kyiv, Kyiv, 2026.

The dissertation is devoted to the development of a concept for a model of a unified algorithmic environment (MUAE) for solving a set of algorithmic problems that arise in the additive manufacturing (AM) process planning, i.e., during the preparation of one or more digital three-dimensional models for 3D printing. The focus is on the following set of problems that arise when using most existing AM technologies. The primary issues addressed include mesh generation, optimal model

orientation and placement within the printing vessel, generation of support structures, slicing of models, and the construction of printing trajectories.

The scope of additive manufacturing is increasingly evolving from a tool primarily used for rapid prototyping to an integral part of complex production chains. One of the main advantages of AM is its ability to overcome the limitations of traditional manufacturing methods, such as casting or milling. It enables the creation of parts with internal channels and lattice structures, resulting in components that are both stronger and lighter. In industries like automotive and aerospace, additive manufacturing enables the production of parts that have a higher load capacity and lower fuel consumption. Current geopolitical challenges and logistical crises emphasize another benefit of AM: the ability to employ an «on-demand manufacturing» model. This approach allows for parts to be created locally at the point of use within hours, which is crucial for military, energy, and medical applications.

Today, there are numerous additive manufacturing technologies that use a variety of materials, including plastics, ceramics, and metals. While these technologies are based on different physical principles, they all share a common step known as AM process planning, which is essential before proceeding with the actual printing of a model. Each AM technology has its own specific requirements and limitations regarding input data, meaning that the challenge of AM planning cannot be addressed with a single universal algorithm or a limited set of those.

The software solutions that exist today can be divided into three categories: open systems for the mass market (e.g., Cura and PrusaSlicer), industrial platforms (e.g., Autodesk Netfabb and Siemens NX), and proprietary solutions from equipment manufacturers (e.g., Formlabs PreForm and EOS Build).

Software products in the first category are open-source, which leads to a low entry barrier, making them a standard choice for amateur desktop 3D printers. These systems provide a complete workflow tailored for their specific functions, ranging from loading a model to generating printer-compatible instructions. However, despite the accessible source code, making significant modifications demands advanced

programming skills and a comprehensive understanding of the system's core. Consequently, when developing new algorithms to address additive manufacturing (AM) challenges, it is often more practical to create custom scripts rather than adapt the existing software architecture.

Industrial platforms offer several advantages, including a wide range of tools for fixing mesh defects and improving quality, flexible packaging algorithms, and seamless integration with thermomechanical analysis modules. However, from a software architecture standpoint, these systems often exemplify a batch-algorithmic approach. This means that functional modules are created using various, often incompatible libraries. While industrial platforms do allow users to automate some routine tasks (for instance, with a Python scripts), these automation capabilities are generally limited to standard actions within the user interface.

Proprietary solutions from printer manufacturers are designed to automate the basic processes of using their equipment. In these software applications, most printing parameters are hidden from the user to prioritize stability and ease of use. This approach ensures high production repeatability without requiring in-depth knowledge of the technology. However, it limits the system's versatility, making it incompatible with printers from other manufacturers. Additionally, users are unable to expand the functionality of the software.

From the overview of existing solutions above, it is clear that none of them provide a complete AM planning process for all 3D printing technologies while maintaining scalability and expandability. Therefore, the current problem is to develop a multi-algorithmic platform architecture that would allow for the effective solution of a sequence of AM planning tasks, would be scalable, and open to extension.

Given the large number of geometric problems that arise in the AM process planning, the basis for solving which is algorithms and problems of computational geometry and computer graphics, the most appropriate and promising way to build a multi-algorithmic environment with the specified properties is to use a unified algorithmic environment model (MUAE), since the use of MUAE has already made it

possible to achieve a significant increase in efficiency when solving interrelated computational geometry problems: construction of Voronoi diagrams and Delaunay triangulation, nearest neighbor search, etc.

However, the direct application of MUAE, which is based on the reducibility theory as the architectural core of the AM process planning support system, does not produce the desired effect, since the specified set of tasks, the algorithms for their solution, and the data structures on which they are based are not formally interrelated. Therefore, this dissertation proposes for the first time to expand the concept of MUAE by introducing tools for managing computations at the environment architecture level: data structure cache (DSC) and implicit reduction of data representations.

The DSC is the main element of the proposed MUAE architecture, responsible for storing and synchronizing shared data structures across different algorithms. During execution, the algorithms for solving AM problems do not build the necessary data structures on their own, but obtain them from the DSC. If the required data structure has already been built and is up to date, the algorithm can use it immediately. Otherwise, the DSC first builds or updates the required data structure, caches it, and only then returns it to the algorithm.

The work of DSC is based on the introduced classification of algorithms according to the nature of changes made to the geometry on which they operate. *Non-modifying* algorithms do not change the geometry or data structure (e.g., nearest neighbor search), *attributive* algorithms can add, modify, or remove properties of geometric objects or their assembly components (e.g., assigning a color to a triangle vertex), *topological-triangulation* algorithms perform topological changes (e.g., triangulation rearrangement), *geometric* algorithms make changes to the geometric location of points describing the object they operate on (e.g., constructing a more organic mesh by rounding corners), *transformation* algorithms only affect the transformation matrix of the object (e.g., rotation or translation of the object in space), and *complex* (or *radical*) algorithms are thouse that perform significant transformations of the model structure. Each algorithm in the MUAE explicitly declares which classes

of changes it belongs to, allowing the DSC to monitor the relevance of the data structures contained in the cache.

Also, within the proposed architecture, the MUAE core is responsible for automatically determining the optimal path for obtaining the required data representation. At the same time, MUAE performs the necessary intermediate calculation or data transformation steps «behind the scenes», so the algorithm developer does not need to manually care about the way how the required data is obtained.

The dissertation reviews the main existing approaches to solving the above-mentioned set of problems in the AM planning process. At the same time, one of the key requirements for the MUAE architecture is the scalability of the system, particulary, by adding new problems and algorithms. Therefore, the functionality of the proposed model of a unified algorithmic environment is not limited to the above list of tasks. Also, to provide additional flexibility of the system, general mesh generation algorithms based on a signed distance function and algorithms for performing Boolean operations on meshes are considered separately, which allow the implementation of specific problems of the AM planning process that are relevant only for some printing technologies or specialized work processes (for example, applying an offset to the geometry).

Separate attention is paid to the data structures used by the considered algorithms, and an analysis is provided of how the integration of these algorithms into a single system based on the proposed MUAE architecture allows for an acceleration of their overall execution time.

The dissertation also explores an approach to constructing effective algorithms for solving certain computational geometry problems, particularly using a Reeb graph (RG) of a planar polygon. In particular, the author proposes a new algorithm for decomposing an arbitrary planar polygon into a set of monotone polygons with simultaneous triangulation. It has been established that, given that RG is available in the DSC, the computational complexity of the developed algorithm is $O(n)$, where n

is the number of vertices of the polygon, which is more efficient than existing triangulation algorithms. It has been shown that the use of RG allows optimizing the AM process planning algorithms that operate on a slice level.

An algorithm for checking the similarity of polygons has been developed. The algorithm operates in linear time and uses a constant amount of additional memory. Utilizing this algorithm, a new optimization approach has been proposed for taking into account the similarity of neighboring slices of a model in algorithms that perform the polygon decomposition procedure. The dissertation also proposes an algorithm for the reconstruction of a polygon decomposition based on the existing decomposition for a similar polygon. Together, these two algorithms make it possible to avoid rebuilding the decomposition or triangulation for identical or very similar model contours, which saves time when processing a large number of consecutive slices.

The effectiveness of the proposed algorithms as components of the MUAE has been confirmed by practical experiments. The performance of the procedure for decomposing an arbitrary polygon into monotone polygons with simultaneous triangulation was compared with the most commonly used C++ programming language libraries that contain triangulation construction algorithms: CGAL, PolyPartition, and Geometric Tools. Testing conducted on polygons with a number of vertices ranging from 10^2 to 10^7 showed that the proposed algorithm, in combination with MUAE, is on average 2-3 times faster than PolyPartition, depending on the size of the input polygon, and at least 5 times faster than other libraries considered. Experimental studies on optimizing the polygon decomposition procedure by taking into account the similarity of neighboring slices were conducted for the task of decomposing an arbitrary polygon into convex ones, which is often used in print path planning algorithms in additive manufacturing. It has been shown that the use of such optimization reduces the decomposition execution time, and the effect of optimization becomes more noticeable as the number of slices increases. For example, the processing time for 2,000 slices of the Stanford Bunny model decreased by 32.2%, and

the processing time for 4,000 slices of a similar model decreased by 47.8%, meaning that the operation was performed almost twice as fast.

Keywords: 3D-modelling, additive manufacturing, code, composition, computational geometry, data set, mathematical model, mathematical optimization, model of a unified algorithmic environment, optimization, parametric optimization, processing speed, regularization, spatial problems, surface representation.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Інтерфейс програмування застосунків, від англ. application programming interface

BJ - Струменеве нанесення в’яжучого, від англ. Binder Jetting

CAD - система автоматизованого проєктування і розрахунку, від англ. Computer-aided design

CCW – поворот проти годинникової стрілки, від англ. counterclockwise rotation

CMS – алгоритм Cubical Marching Squares

CPU – центральний процесор, від англ. central processing unit

CW – поворот за годинниковою стрілкою, від англ. clockwise rotation

DED - Пряме підведення енергії, від англ. Directed Energy Deposition

MC – алгоритм Marching Cubes

ME - Екструзія матеріалу, від англ. Material Extrusion

MJ - Струменеве нанесення матеріалу, від англ. Material Jetting

MS – алгоритм Marching Squares

NURBS – неоднорідний раціональний B-сплайн

PBF - Синтез на підкладці, від англ. Powder Bed Fusion

RAM - пам’ять з довільним доступом, від англ. Random access memory

SL - Листова ламінація, від англ. Sheet Lamination

SLA – стереолітографія, від англ. stereolithography

VP - Фотополімеризація у ванні, від англ. Vat Photopolymerization

АВ – адитивне виробництво

ГР – граф Ріба

МСД – менеджер структур даних

МЄАС – модель єдиного алгоритмічного середовища

ОТД – обмежена тріангуляція Делоне

РСПЗ – реберний список з подвійними зв’язками

ООП – об’єктно-орієнтоване програмування

ВСТУП

Актуальність теми. Перші технології адитивного виробництва (АВ) з'явились в середині 80-х років XX сторіччя. Спершу сфера застосування АВ була обмежена виготовленням пластикових прототипів, а сама індустрія мала назву «швидке прототипування». Однак, з часом все більш очевидними ставали технологічні переваги, які надає процес виготовлення моделі шляхом пошарового накладання матеріалу, а саме можливість будувати деталі зі складною внутрішньою структурою, яку до цього неможливо було досягнути традиційними технологіями виробництва. Водночас відбувався розвиток технологій АВ та розширення доступного спектру матеріалів. З'явилися такі технології, як SLM та WAAM, що дозволяють будувати функціональні металічні деталі зі складною внутрішньою геометрією, що привернуло увагу до АВ технологічних гігантів з таких індустрій, як автомобілебудування та аерокосмічна промисловість. Станом на сьогодні, технології адитивного виробництва знаходять своє застосування також в ювелірній промисловості, медицині, будівництві та навіть в кіноіндустрії для виготовлення декорацій.

Результатом стрімкого розвитку адитивного виробництва стала поява великої кількості технологій АВ, що базуються на різних фізичних процесах та мають власні вимоги та обмеження до моделі (або моделей), що друкується. Так, в дослідженні (VoxelMatters, 2019) виділено понад 100 різних технологій АВ. Наслідком такого різноманіття є поява великої кількості вузькоспеціалізованих програмних продуктів для підготовки моделі або кількох моделей до друку. Цей процес в індустрії АВ прийнято називати процесом планування адитивного виробництва. Використання подібного програмного забезпечення є прийнятним в межах науково-дослідних лабораторій, оскільки в таких умовах немає строгих вимог до автоматизації та відтворюваності процесу. Проте, для використання АВ в якості рівноправного компонента в складних виробничих ланцюжках критично важливим є ефективне розв'язання всього комплексу задач процесу планування

АВ в межах єдиної екосистеми, що з одного боку забезпечує автоматизацію та відтворюваність процесів, а з іншого боку є гнучкою та масштабованою.

Дослідженням окремих задач АВ, а також розробкою фундаментального базису для їх розв'язання займались L.C. Aleardi, M. Campen, O. Cheong, M. De Berg, K. Deb, E.W. Dijkstra, D. Ding, R. Dwivedi, P.E. Hart, C.-C. Ho, X. Jiang, Y. Jin, B. Joe, D.-S. Kim, L. Kobbelt, P.-T. Lan, W.E. Lorensen, E. Lutters, M. Mares, D. Meagher, R. Minetto, M. Overmars, G. Rozenberg, S. Schaefer, A. Scherzinger, R. Tarjan, R. Vaidya, M. van Kreveld, N. Volpato, I. Wald, S.W. Yang, F. Yu, Y. Zhang, T. Zhao та багато інших. Серед вітчизняних науковців даною проблематикою займались А. В. Анісімов, Д. В. Коцур, О. В. Панкратов, О. О. Пришляк, Т. Є. Романова, Ю. Г. Стоян, Ю. Є. Стоян, В. М. Терещенко, А. Л. Фісуненко, М. І. Чернов, А. М. Чугай та інші.

Існуючі на сьогодні програмні рішення лише частково задовольняють описані вище вимоги. Так, відкриті системи масового сегмента, наприклад PrusaSlicer (Prusa Research, 2025), орієнтовані здебільшого на найдоступніші технології АВ, що робить неможливим їх застосування, зокрема, для друку металами. Також, будь-яке їх розширення є нетривіальною задачею та вимагає серйозних навичок програмування та ретельного вивчення архітектури системи. Промислові платформи, такі як Autodesk Netfabb (Autodesk, n.d.), пропонують потужний набір інструментів для розв'язання задач процесу планування АВ, але використання пакетно-алгоритмічного підходу робить цей процес досить повільним та ресурсозатратним. Також, архітектура таких систем не дозволяє виконувати масштабування шляхом додавання нових задач та алгоритмів, а можливості автоматизації обмежені скриптуванням стандартних дій в інтерфейсі системи. При цьому, вартість однієї ліцензії користувача такої системи вимірюється тисячами доларів США в рік. Пропрієтарні рішення від виробників обладнання, наприклад Formlabs PreForm (Formlabs, n.d.), забезпечують автоматизацію базових сценаріїв використання обладнання, однак вони є

несумісними з принтерами від інших виробників та закриті до масштабування. Детальний огляд існуючих програмних рішень наведено в підрозділі 1.2.

Відповідно, виконання процесу планування АВ вимагає залучення великої кількості програмних систем, навіть за умови застосування однієї технології АВ. При цьому, як зазначається в дослідженні Carnegie Mellon University (Rosenthal, 2016), існуючі програмні рішення не використовують такі концепції як модульність, що обмежує можливості щодо автоматизації виробничих процесів. Зазначенні вище обмеження існуючих підходів, а також вимоги індустрії щодо масштабування та автоматизації підкреслюють актуальність побудови архітектури єдиного комплексного середовища, що від початку до кінця забезпечує процес планування адитивного виробництва.

Прикладом розв’язання подібної проблеми для комплексу взаємопов’язаних задач обчислювальної геометрії є модель єдиного алгоритмічного середовища (МЄАС), що запропонована в роботі (Терещенко, 2011). Застосування МЄАС дозволило одночасно вирішити групу взаємопов’язаних задач, що суттєво підвищує продуктивність обробки великих об’ємів даних. Як буде показано в розділі 2 цієї дисертаційної роботи, саме задачі та алгоритми обчислювальної геометрії є фундаментом більшості підходів до розв’язання комплексу задач планування АВ, тому перспективним напрямком розробки архітектури ефективної універсальної платформи для планування АВ є саме побудова моделі єдиного алгоритмічного середовища, що враховує залежності між задачами та автоматично оптимізує обчислювальні процеси системи.

Зв’язок роботи з науковими програмами, планами, темами.

Робота є складовою частиною наукових досліджень, які ведуться на кафедрі математичної інформатики факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка на виконання теми проєкту Міністерства освіти і науки України «Комп’ютерне моделювання, візуалізація та супроводження комплексних індивідуальних

програм медичної реабілітації за стандартами міжнародної класифікації функціонування» (НДФ 24БФ015-01, 01.01.2024-31.12.2026)

Мета і завдання дослідження. Метою дисертаційного дослідження є побудова моделі єдиного алгоритмічного середовища, що забезпечує ефективне розв'язання комплексу задач Адитивного Виробництва (АВ) та є масштабованим як шляхом додавання нових задач, так і шляхом додавання нових алгоритмів розв'язання існуючих задач. Відповідно до мети сформульовано такі завдання:

1. Провести огляд комплексу задач АВ та сучасних підходів до їх розв'язання.
2. Розробити архітектуру моделі єдиного алгоритмічного середовища для розв'язання комплексу задач АВ, яка дозволить ефективно розв'язувати низку задач, між якими немає взаємозв'язку в розумінні теорії звідності.
3. Створити ефективні методи інтеграції існуючих алгоритмів розв'язання комплексу задач АВ в розроблену архітектуру МЄАС. Довести, що інтеграція зазначених алгоритмів в МЄАС дозволяє прискорити їх роботу.
4. Розробити нові ефективні алгоритми розв'язання окремих задач обчислювальної геометрії, та застосувати їх для оптимізації розв'язання комплексу задач АВ.
5. Виконати програмну реалізацію розроблених алгоритмів в межах МЄАС та продемонструвати їх практичну ефективність в порівнянні з існуючими підходами.

Об'єктом дослідження є процес підготовки цифрових моделей до 3D-друку.

Предметом дослідження є архітектура, алгоритмічна база та структури даних для побудови уніфікованого програмного середовища, що забезпечує ефективне розв'язання комплексу задач процесу планування АВ.

Методи дослідження. Дисертаційна робота використовує наступні методи:

- Прикладної теорії алгоритмів
- Обчислювальної геометрії та комп'ютерної графіки

- Теорії графів
- Методів оптимізації
- Програмування

Наукова новизна отриманих результатів полягає в наступному:

Вперше:

- розроблено архітектуру моделі єдиного алгоритмічного середовища для розв’язання комплексу задач адитивного виробництва, розширену шляхом введення інструментів керування обчисленнями на рівні архітектури, що дозволило вперше отримати МЄАС для розв'язання задач, між якими немає взаємозв’язку в розумінні теорії звідності.
- розроблено лінійний алгоритм побудови декомпозицію плоского многокутника на множину тріангульованих y-монотонних многокутників в межах МЄАС. Показано, що запропонований алгоритм дозволяє прискорити виконання попередньої обробки в алгоритмі (Dwivedi & Kovacevic, 2004) розв’язання задачі побудови траєкторії друку.
- розроблено алгоритм оцінки подібності двох многокутників на площині, що враховує особливості форми многокутників та відстань Гаусдорфа між їх границями. Доведено, що алгоритм має лінійну оцінку швидкості роботи та константу оцінку додаткової пам’яті
- розроблено алгоритм відновлення довільної хордової декомпозиції многокутника на основі відомої декомпозиції подібного многокутника
- розроблено оптимізацію алгоритмів побудови траєкторії друку, що використовують підхід декомпозиції вхідного многокутника, який описує контур слайсу моделі, на множину многокутників з заданими властивостями (y-монотонні, опуклі тощо) шляхом врахування

подібності між сусідніми слайсами моделі та повторного використання існуючої траєкторії

Практичне значення одержаних результатів. Робота має як теоретичні так і практичні результати. Запропонований підхід до розширення концепції моделі єдиного алгоритмічного середовища може бути застосованим не лише до комплексу задач АВ, але й до інших проблемних областей, в яких постають різнорідні алгоритмічні задачі. Продемонстровано, що розроблена МЄАС дозволяє розв'язувати наступні задачі процесу планування АВ: генерування мешів, орієнтація та позиціонування моделей, побудова опорних конструкцій, слайсинг та побудова траєкторії друку. Запропонована архітектура МЄАС є гнучкою, вона дозволяє інтегрувати як існуючі алгоритми розв'язання зазначених задач, при цьому нові задачі та алгоритми можуть бути додані без внесення змін в архітектуру та без втрати властивостей системи. Проведене практичне тестування реалізації запропонованих алгоритмів показує, що в поєднанні з МЄАС вони дозволяють досягнути кращої швидкодії, ніж існуючі на сьогодні альтернативи, що робить запропонований підхід застосовним для побудови високоефективних промислових систем підготовки моделей до 3D-друку конвеєрного типу.

Особистий внесок здобувача. Всі результати, представлені в дисертації, отримані автором самостійно, викладені у вигляді тверджень і теорем, які строго доведено з використанням допоміжних результатів. При обґрунтуванні наведено всі посилання на використані джерела. З публікацій, виконаних у співавторстві, до дисертації включено лише особисті результати здобувача.

В роботі (Tereshchenko & Osiponok, 2024) персональний внесок автора полягає в проведенні огляду актуальних задач АВ та існуючих алгоритмів їх розв'язання, а також в формулюванні проблематики поточного підходу до цих задач в комплексі. Внесок автора в (Tereshchenko et al., 2025) полягає в побудові алгоритму декомпозиції многокутника на тріангульовані y-монотонні

многокутники, формулювання та доведення теореми про часову складність запропонованого алгоритму, його практична реалізація та проведення порівняння реалізації з існуючими бібліотеками. В (Osiponok & Tereshchenko, 2025a) автор сформулював процедуру перевірки подібності двох многокутників, обґрунтував її часову складність, виконав практичну реалізацію та оцінку ефективності запропонованої процедури. В роботі (Osiponok & Tereshchenko, 2023) автором було виконано огляд актуальних підходів до розв'язання задачі розміщення моделей в межах робочої області 3D принтера. В (Osiponok & Tereshchenko, 2025b) автором було висвітлено концепцію застосування МЄАС для розв'язання комплексу задач адитивного виробництва. Внесок автора в роботу (Osiponok & Tereshchenko, 2025c) полягає в виборі структур даних та побудові алгоритму відновлення декомпозиції, реалізації цього алгоритму та перевірці його ефективності шляхом виконання практичних експериментів.

Апробація результатів дисертації. Основних положення та результати дисертаційного дослідження доповідалися на наукових семінарах, що проходили на факультеті комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. Також, основі результати дослідження доповідались на наступних всеукраїнських та міжнародних наукових конференціях:

- 2nd International Scientific and Practical Internet Conference «Way Science». Dnipro, Ukraine. April 3-4, 2023.
- III International Scientific Symposium "Intelligent Solutions" (Satellite). Kyiv / Uzhhorod, Ukraine. September 28, 2023.
- XXIX Всеукраїнська наукова конференція молодих істориків науки, техніки і освіти та спеціалістів за темою: «наука для відбудови України». Київ, Україна. 19 квітня 2024.
- XXIII International Scientific – Practical Conference «Shevchenkivska Vesna – 2025». Kyiv, Ukraine. April 10, 2025.

- XXV International Scientific and Practical Conference ITS-2025. Kyiv, Ukraine. December 11, 2025.

Публікації. Основні результати дисертації опубліковані в 8 наукових роботах, серед яких: 2 статті в виданнях, що індексуються наукометричною базою Scopus, 1 стаття в фаховому вітчизняному виданні категорії «Б», а також 5 тез конференцій.

Структура дисертації. Дисертаційна робота складається із вступу, п’яти розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить 182 с., основний текст роботи викладено на 143 с. Список використаних джерел містить 116 найменувань та оформлений відповідно до стилю APA style, який віднесено до переліку рекомендованих стилів оформлення списку наукових публікацій у дисертації відповідно до наказу Міністерства освіти і науки України від 12.01.2017 № 40 "Про затвердження вимог до оформлення дисертацій" (зі змінами, внесеними згідно з Наказом від 31.05.2019 № 759).

РОЗДІЛ 1. ТЕХНОЛОГІЇ АДИТИВНОГО ВИРОБНИЦТВА ТА ТЕОРЕТИЧНЕ ПІДҐРУНТЯ ПОБУДОВИ КОМПЛЕКСТИХ МУЛЬТИАЛГОРИТМІЧНИХ СИСТЕМ

В цьому розділі буде здійснено огляд основних концепцій, понять та фізичних принципів, що лежать в основі адитивного виробництва. Зокрема, будуть розглянуті ключові ідеї та підходи, що характеризують основні технології 3D-друку, а також охарактеризовано процес планування адитивного виробництва. Крім того, розділ містить огляд наявних програмних рішень, що використовуються для підтримки цього процесу.

В підрозділах 1.3 – 1.5 наведено основні математичні та алгоритмічні означення, що застосовуються у подальших розділах дослідження для аналізу, побудови та оцінки ефективності розроблених алгоритмів.

В підрозділі 1.6 розглядається концепція моделі єдиного алгоритмічного середовища (МЄАС) як підходу до побудови ефективних мультиалгоритмічних обчислювальних систем. Окрему увагу приділено аналізу останніх наукових досліджень, що застосовують концепцію МЄАС у таких галузях, як обчислювальна геометрія, комп’ютерна графіка, обробка зображень та машинне навчання.

1.1 Технології адитивного виробництва

Адитивне виробництво (АВ), також відоме як 3D-друк, це узагальнена назва сукупності процесів створення тривимірних об'єктів на основі цифрових моделей шляхом пошарового додавання матеріалу. На відміну від традиційних підходів, де створення форми обмежене можливостями інструментів чи спеціального оснащення, адитивні технології дозволяють синтезувати необхідний об’єкт безпосередньо з цифрових даних.

Історія розвитку АВ бере початок у 1980-х роках. У 1984 році було розроблено першу машину для стереолітографії (SLA), що започаткувало

революцію у промисловості (Ганєєв та ін., 2023). Основною метою створення SLA-машини була оптимізація та прискорення процесу виготовлення прототипів нових компонентів, сприяння їх візуалізації, а також забезпечення можливості оцінки взаємодії нових деталей з вже існуючими (Wohlers et al., 2023).

З поступовим розвитком технологій 3D-друку все більш очевидними ставали переваги цієї технології порівняно з традиційними методами виробництва, зокрема: мінімізація виробничих відходів, можливість децентралізації виробничих потужностей, масова персоналізація виробництва, а також здатність виготовляти складні геометричні форми, що є неможливими або надзвичайно складними для традиційних методів виробництва (Huang et al., 2012; Kwok et al., 2017). Сьогодні АВ знаходить застосування у найрізноманітніших сферах: в аерокосмічній промисловості, автомобілебудуванні та медицині (Chiu, 2020; Salmi, 2021; Valdivieso, 2021).

На сьогоднішній день існує значна кількість технологій адитивного виробництва. Так, в дослідженні (VoxelMatters, 2019) виокремлено понад сто різних технологій АВ. Така велика варіативність частково пояснюється патентними обмеженнями: компанії часто змушені шукати шляхи обхідних рішень, що призводить до появи формально нових технологій, які за своєю суттю та принципом роботи є подібними до вже існуючих. Окремо виділяють 4D-друк, під яким розуміють застосування традиційних технологій АВ з програмованими матеріалами, що можуть змінювати свої властивості або форму з плином часу під впливом зовнішнього середовища (зміна температури, вологості, тиску тощо). З огляду на це, технології АВ прийнято розділяти на 7 класів за принципом їх роботи, відповідно до стандарту ISO/ASTM 52900:2021 (International Organization for Standardization, 2021). Нижче буде наведено короткий опис кожного класу та наведено приклади відповідних технологій АВ. Детальний опис розглянутих технологій АВ разом з наочними ілюстраціями можна знайти в роботах (Wong & Hernandez, 2012) та (Ганєєв та ін., 2023).

1.1.1 Фотополімеризація у ванні (Vat Photopolymerization, VP)

Технології фотополімеризації у ванні використовують рідкі фоточутливі смоли, що тверднуть під впливом світлового випромінювання певної довжини хвилі. Серед основних технологій АВ цього класу виділяють наступні:

- SLA (Stereolithography) є найстарішою технологією 3D-друку. В якості матеріалу друку використовуються рідкі фотополімерні смоли. В процесі друку платформа занурюється у ванну зі смолою, а УФ-лазер точково сканує поверхню, викликаючи полімеризацію. Після завершення шару платформа зміщується по вертикалі, і процес повторюється для наступного слайсу. Технологія SLA забезпечує надзвичайно високу точність, гладкість поверхні та можливість створення дрібних деталей, що робить її незамінною в стоматології та ювелірній справі.
- DLP (Digital Light Processing) є близькою до технології SLA, але замість лазера застосовується проектор, що засвічує весь шар одночасно, формуючи зображення слайсу за допомогою цифрового мікродзеркального пристрою. Це дозволяє значно пришвидшити процес друку, оскільки час формування шару не залежить від кількості деталей на платформі. Деталі, отримані методом DLP, мають високу роздільну здатність, хоча на поверхні іноді можна помітити «пікселізацію» від матриці проектора.

1.1.2 Екструзія матеріалу (Material Extrusion, ME)

Клас ME об'єднує технології, в яких матеріал вибірково подається через сопло або екструдер. Це найбільш поширений клас технологій АВ завдяки відносній простоті обладнання та доступності матеріалів. Основними технологіями класу ME є:

- FDM (Fused Deposition Modeling). Технологія FDM використовує термопластичні полімери у вигляді твердої нитки, яку називають філаментом. Філамент нагрівається до напіврідкого стану і вкладається шарами, що застигають при охолодженні. FDM широко використовується

для створення прототипів та функціональних деталей з пластиків ABS, PLA, PETG та нейлону. Недоліками методу є анізотропія механічних властивостей (деталь міцніша вздовж волокон, ніж між шарами) та помітна шаруватість поверхні. Сучасні системи дозволяють використовувати два екструдера для друку основним матеріалом та окремим матеріалом для опорних структур.

- DIW (Direct Ink Writing) виконує друк шляхом пошарової екструзії в'язких матеріалів (в рідкому або пастоподібному стані) через сопло під тиском повітря або поршня. На відміну від технології FDM, DIW-друк зазвичай не використовує нагрівання для плавлення. Стабільність форми отриманої деталі досягається за рахунок властивостей самого матеріалу, або як результат подальшої додаткової обробки. Технологія дозволяє працювати з широким спектром матеріалів, що включають керамічні пасти, гідрогелі з живими клітинами для біодруку та композитні суміші.

1.1.3 Струменеве нанесення матеріалу (Material Jetting, MJ)

Технологія струменевого нанесення матеріалу базується на принципах, схожих на роботу звичайного струменевого принтера, але в тривимірному просторі. Друкуюча голівка наносить краплі рідкого фотополімеру, які миттєво затверджуються під впливом ультрафіолетової лампи. Такий підхід дозволяє досягати високої точності та гладкості поверхні. Особливістю MJ є можливість одночасного друку різними матеріалами, комбінуючи тверді та гнучкі, прозорі та кольорові полімери в одному виробі.

1.1.4 Струменеве нанесення в'яжучого (Binder Jetting, BJ)

Процес друку методом струменевого нанесення в'яжучого схожий на технологію MJ. Замість рідкого фотополімеру використовується матеріал у вигляді порошку, на який друкуюча голівка вибірково наносить в'яжучу речовину (клей). Процес відбувається без нагрівання, що дозволяє уникнути температурних деформацій. Однак, безпосередньо після друку деталь є крихкою і потребує подальшої обробки: інфільтрації іншим матеріалом (наприклад,

бронзою для сталевих деталей) або спікання в печі для надання остаточної міцності.

1.1.5 Пряме підведення енергії (Directed Energy Deposition, DED)

Технології цього класу подають матеріал безпосередньо в зону дії джерела енергії (наприклад, лазера). Серед основних технологій класу DED виділяють наступні:

- LENS / DED (Laser Engineered Net Shaping). Металевий порошок подається через сопло безпосередньо у фокус потужного лазера, де створюється ванна розплаву. Особливістю технології LENS є можливість не лише створювати нові деталі, але й ремонтувати зношені компоненти готових виробів, додаючи метал на існуючу поверхню, а також створювати функціонально-градієнтні матеріали шляхом зміни суміші порошків в процесі друку.
- WAAM (Wire Arc Additive Manufacturing). Технологія WAAM використовує зварювальну дугу для плавлення металевого дроту. Процес фактично являє собою автоматизоване дугове зварювання, під час якого роботизований маніпулятор пошарово наплавляє розплавлений метал для формування тривимірного об’єкта. WAAM відрізняється надзвичайно високою продуктивністю наплавлення та низькою вартістю матеріалів, оскільки для друку використовується звичайний зварювальний дріт. Це робить технологію WAAM безальтернативною для виготовлення великогабаритних металевих конструкцій довжиною у кілька метрів, таких як елементи фюзеляжу літаків або гребні гвинти суден. Цей процес не обмежений розміром робочої камери і дозволяє створювати масивні деталі значно швидше та дешевше. Однак, отримані вироби мають низьку роздільну здатність і виражену хвилястість поверхні. Як наслідок, надруковані деталі потребують обов'язкової подальшої механічної обробки (фрезерування) для досягнення кінцевих розмірів та задовільної якості поверхні.

1.1.6 Синтез на підкладці (Powder Bed Fusion, PBF)

Категорія PBF охоплює методи друку, в яких теплова енергія вибірково сплавляє ділянки шару порошкового матеріалу. Ключовими технологіями з цієї категорії є:

- Селективне лазерне спікання (SLS). Метод використовує порошкоподібні полімери, найчастіше поліамід (нейлон), які спікаються під дією лазера. Робоча область принтера попередньо нагрівається майже до температури плавлення матеріалу, а лазер додає енергію лише в ті точки, які потрібно затвердити. Важливою перевагою SLS є відсутність необхідності у додаткових опорних структурах, оскільки деталь підтримується неспеченим порошком, що залишається в камері принтера. Це дозволяє створювати складні геометричні форми зі з'єднаними або рухомими частинами та високою механічною міцністю.
- Селективна лазерна плавка (SLM). Технологія SLM працює з металевими порошками, що повністю розплавляються потужним лазером. Процес відбувається в камері, заповненій інертним газом, щоб запобігти окисленню металу. SLM дозволяє отримувати вироби з монолітною структурою та механічними властивостями, що не поступаються або навіть перевершують литі аналоги. На відміну від SLS-друку, технологія SLM вимагає побудови додаткових опорних конструкцій, за допомогою яких відбувається відведення тепла для уникнення деформації деталі від залишкового механічного напруження. Метод широко застосовується в аерокосмічній галузі та медицині для створення легких, але міцних компонентів та імплантатів.
- Електронно-променева плавка (EBM). Технологія EMB також використовує металевий порошок, але джерелом енергії виступає потік електронів у вакуумній камері. Висока енергія електронного променю дозволяє досягати високих температур, що робить метод ідеальним для обробки тугоплавких металів, таких як титанові сплави. Процес EBM

відбувається при підвищених температурах, що знижує залишкові напруження в деталі та усуває потребу в термічній обробці після друку. Однак, поверхня деталей, виготовлених методом EBM, зазвичай є більш шорсткою, в порівнянні з лазерними методами.

1.1.8 Листова ламінація (Sheet Lamination, SL)

SL є гібридним класом методів виробництва, що поєднує адитивний (додавання шарів) та субтрактивний (вирізання контуру) підходи. Основним представником класу SL є технологія виготовлення об’єктів ламінуванням.

- Виготовлення об'єктів ламінуванням (LOM). Технологія LOM використовує твердий листовий матеріал, такий як папір, пластикова плівка або металева фольга. Листи послідовно склеюються між собою під тиском та нагріванням, після чого лазер (або ніж) вирізає контур деталі на кожному шарі. Зайвий матеріал навколо деталі не видаляється під час друку і слугує природною опорою. LOM дозволяє швидко та дешево створювати великогабаритні макети, але має обмеження щодо складності внутрішньої геометрії.

1.2 Процес планування та комплекс задач АВ. Огляд існуючих програмних рішень

Під процесом планування адитивного виробництва прийнято розуміти етап підготовки цифрової моделі до її фізичного виробництва одним із методів АВ. Етап планування АВ є перехідною ланкою між абстрактною геометричною моделлю, побудованою без врахування специфіки технологій АВ, та набором машинозрозумілих інструкцій для керування апаратним забезпеченням 3D-принтера, результатом застосування яких є фізична репрезентація побудованої моделі (можливо, з виконанням додаткової пост-обробки).

Через фундаментальні відмінності у фізиці процесів різних технологій АВ, розв’язання задачі планування АВ не може бути зведене до єдиного універсального алгоритму. Різноманітність технологій та матеріалів породжує

широкий спектр специфічних задач, які інженер повинен вирішити на етапі планування друку.

Під комплексом задач АВ в цій дисертаційній роботі будемо розуміти наступний набір задач, що є актуальними для більшості технологій 3D-друку:

Задача 1. *Генерування мешів* полягає в побудові полігональної сітки, що задає границю тривимірної моделі. Бажана модель (вхідні дані) може задаватись у вигляді параметричної поверхні, функції відстані зі знаком, або булевою операцією (об'єднанням, перетином або різницею) над іншими мешами.

Задача 2. *Розміщення моделей в межах робочої області принтера* полягає у виборі оптимальної, згідно обраного критерія, орієнтації для набору тривимірних моделей та їх розміщення в межах робочої області принтера без перетинів та замикань.

Задача 3. *Генерування опорних конструкцій* полягає в побудові додаткового мешу для заданої моделі відповідно до її орієнтації та позиції, який, в залежності від специфіки обраної технології АВ, може запобігати її руйнуванню під дією гравітації в процесі друку, забезпечувати відведення тепла та міцне кріплення до платформи принтера.

Задача 4. *Слайсинг* полягає в побудові перерізів моделей, що друкуються, горизонтальними площинами, які відповідають шарам нанесення матеріалу в процесі друку. Відстань між площинами (товщина слайсу) може бути фіксованою, або динамічно адаптуватись до деталізації поверхні моделі на відповідній висоті.

Задача 5. *Побудова траєкторії друку* полягає в визначенні шляху (траєкторії), за яким друкуючий інструмент буде наносити матеріал для заданого перерізу моделі. Результуюча траєкторія може бути суцільною або складатись з кількох окремих частин.

Важливо зауважити, що інтерпретація цих задач часто залежить від обраної технології АВ. Розглянемо, наприклад, задачу вибору просторової орієнтації та

розташування моделі всередині робочої області принтера. Відомо, що орієнтація деталі має безпосередній вплив на її механічні властивості, якість результуючої поверхні та час друку (Kok et al., 2018). Для досягнення максимальної міцності при використанні технології FDM, деталь намагаються розмістити таким чином, щоб вектори дії максимального навантаження не співпадали з напрямком зчеплення слайсів – віссю z, що підтверджується результатами дослідження (Eryildiz, 2021). В той же час, для ефективності промислового застосування порошкового друку, зокрема технологією SLS, важливо досягнути високої щільності заповнення робочої області принтера деталями (Sinterit, 2025).

Іншим прикладом відмінностей в підході до процесу планування АВ, залежно від обраної технології друку, є генерування додаткових опорних конструкцій. Технологія полімерного друку порошком SLS не потребує додаткових опор (оскільки природною опорою слугує неспечений порошок з попередніх слайсів). Разом з тим, технології полімерного друку FDM та SLA потребують опор для утримання нависаючих ділянок моделі від падіння або усадки під дією гравітації. При використанні металевого друку технологіями SLM або EBM, опорні структури виконують набагато складнішу функцію, ніж просто утримання геометрії. Вони мають забезпечити жорстке кріплення деталі до платформи для уникнення ефекту скручування внаслідок внутрішніх залишкових напружень, приклад якого наведено на рисунку 1.1 (Kishor et al., 2026). Також, в металевому АВ опори відіграють роль радіатора, створюючи канал відведення тепла між зоною розплаву та охолоджуваною платформою побудови.

Ринок програмного забезпечення для процесу планування АВ еволюціонував паралельно з розвитком апаратного забезпечення та виникненням нових технологій АВ. Сьогодні існує велика кількість спеціалізованих програмних рішень, які можна розділити на три категорії: відкриті системи масового сегмента, промислові платформи та пропрієтарні рішення від виробників обладнання.

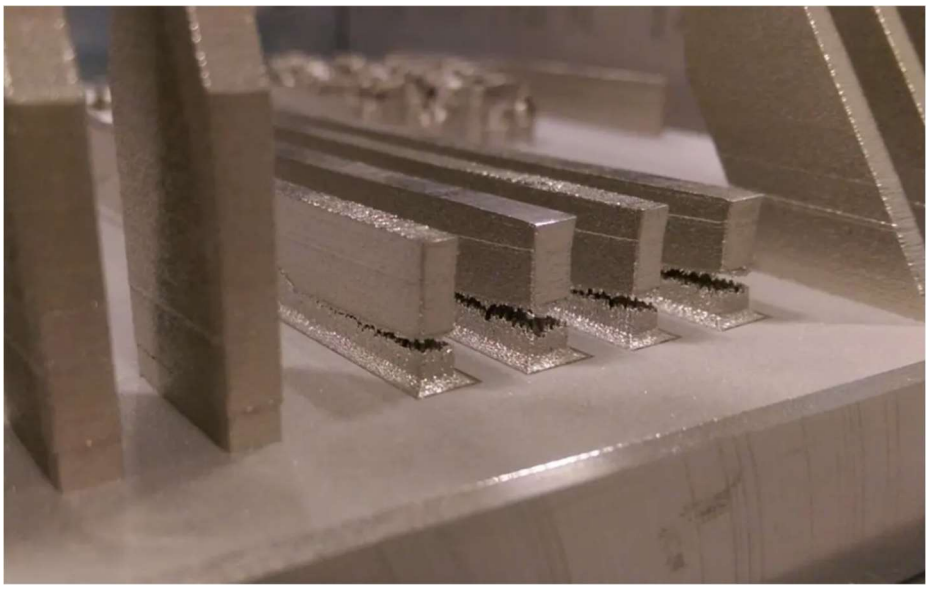

Рисунок 1.1. Приклад впливу залишкових напружень в металевому адитивному виробництві (Kishor et al., 2026)

До першої категорії відносяться такі програмні продукти як Cura (Ultimaker, 2025) та PrusaSlicer (Prusa Research, 2025), що є орієнтованими на найбільш доступні широкому загалу технології, зокрема екструзію полімерів (MJ) та фотополімеризацію (VPP). Представники цієї категорії є відкритими програмними продуктами та мають відносно низький поріг входу, що зробило їх стандартом для настільних 3D-принтерів аматорського сегмента. В межах своєї спеціалізації ці системи пропонують повноцінний робочий цикл, від завантаження моделі до генерування зрозумілих принтеру інструкцій. Однак, незважаючи на відкритість вихідного коду, будь-яка суттєва модифікація вимагає високих навичок у програмуванні та глибокого розуміння ядра системи. На практиці це призводить до того, що дослідникам часто простіше створити власні спеціалізовані скрипти, ніж намагатися адаптувати архітектуру цих програм під нетипові адитивні процеси.

Комерційні промислові платформи, представлені такими рішеннями, як Autodesk Netfabb (Autodesk, n.d.) та Siemens NX (Siemens AG, 2025), розроблені для обслуговування складних промислових процесів, насамперед порошкових технологій (PBF, BJ) та прямого підведення енергії (DED). Вони пропонують потужний набір інструментів для виправлення дефектів геометричних моделей,

оптимізації пакування, а також інтеграцію з модулями термомеханічного аналізу. Проте такі системи будуються з використанням пакетно-алгоритмічного підходу, де функціональність розбита на окремі закриті модулі. Хоча вони зазвичай дозволяють користувачеві забезпечити певний рівень автоматизації рутинних задач (наприклад, шляхом написання скриптів мовою Python), ці можливості обмежуються автоматизацією стандартних дій в інтерфейсі. Дослідник не має доступу до зміни фундаментальної логіки обчислень або геометричного ядра, що значно ускладнює впровадження інноваційних алгоритмів в межах існуючої комерційної екосистеми.

Окремо розглядають програмне забезпечення, створене виробниками 3D-принтерів виключно для власного обладнання, наприклад Formlabs PreForm (Formlabs, n.d.) та EOS Build (EOS GmbH, 2025). Такі системи фактично є «чорним ящиком», де більшість критичних параметрів друку прихована від користувача заради стабільності результату та простоти експлуатації. Це дозволяє досягти високої повторюваності виробництва без глибоких знань технологічного процесу, проте повністю позбавляє систему універсальності. Пропрієтарне програмне забезпечення неможливо використовувати з обладнанням інших виробників, а будь-яке розширення функціоналу з боку користувача заблоковано на рівні архітектури та захисту інтелектуальної власності виробника. Такий підхід створює ефект «замкненості на вендорі», що є неприйнятним для наукових розробок у галузі нових алгоритмів, методів оптимізації, матеріалів чи технологій виробництва.

Проведений аналіз дозволяє стверджувати, що на сьогоднішній день не існує універсального програмного рішення, яке могло б ефективно розв’язувати комплекс задач етапу планування для всіх технологій адитивного виробництва. Ключовою проблемою для науково-дослідного сектора є архітектурна обмеженість існуючих платформ: використання пакетно-алгоритмічного підходу створює значні обмеження для гнучкого розширення систем та інтеграції нових математичних моделей.

1.3 Основні означення

В цьому розділі подано основні математичні та геометричні поняття й властивості, які в подальшому будуть застосовуватись під час аналізу та розробки алгоритмічної основи моделі єдиного алгоритмічного середовища. Метою цього розділу є ознайомлення з базовими поняттями та введення позначень, які в подальшому будуть зустрічатись в тексті дисертаційної роботи. Детальний опис наведених понять та властивостей можна знайти в роботах (De Berg et al., 2008), (Мальцев, 1970), та (Пришляк & Терещенко, 2024).

Нехай $\mathbb{R}^m$ – декартовий простір дійсних чисел розмірності m з початком координат в точці O.

Означення 1.1. Точкою p простору $\mathbb{R}^m$ будемо називати кортеж з m дійсних чисел $p = (x_1, x_2, \dots, x_m)$.

Для спрощення індексації координат, в часткових випадках будемо використовувати позначення $p = (x, y)$ для простору $\mathbb{R}^2$, та $p = (x, y, z)$ для простору $\mathbb{R}^3$.

Множину n точок простору $\mathbb{R}^m$ будемо позначати $\{p_1, p_2, \dots, p_n\}$, де кожна точка p_i має координати $p_i = (x_{i1}, x_{i2}, \dots, x_{im}), i = \overline{1, n}$.

Кожну точку p простору $\mathbb{R}^m$ можна розглядати як m-вимірний вектор-стовпець $\vec{p}$, що починається в точці O та закінчується в точці p:

$$\vec{p} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}.$$

Відповідний вектор-рядок будемо позначати $\vec{p}^T = (x_1, x_2, \dots, x_m)$, де оператор $\cdot^T$ позначає операцію транспонування.

Означення 1.2. Евклідовою нормою вектора $\vec{p}$ (або довжиною вектора $\vec{p}$) простору $\mathbb{R}^m$ будемо називати величину, що обчислюється наступним чином:

$$\|\vec{p}\| = \sqrt{\vec{p}\,\vec{p}^T} = \sqrt{\sum_{i=1}^{m} x_i^2}.$$

Означення 1.3. Мангеттенською нормою (або L_1-нормою) вектора $\vec{p}$ простору $\mathbb{R}^m$ будемо називати величину, що обчислюється наступним чином:

$$\|\vec{p}\|_1 = \sum_{i=1}^{m} |x_i|.$$

Сума векторів $\vec{p}_1$ та $\vec{p}_2$ простору $\mathbb{R}^m$ визначається як вектор $\vec{p} = \vec{p}_1 + \vec{p}_2$ з простору $\mathbb{R}^m$, що має координати $\vec{p} = (x_{11} + x_{21}, x_{12} + x_{22}, \dots, x_{1m} + x_{2m})^T$. Добуток скаляру $\lambda \in \mathbb{R}$ та вектора $\vec{p} \in \mathbb{R}^m$ є вектор $\lambda\vec{p} = (\lambda x_1, \lambda x_2, \dots, \lambda x_m)^T$.

Означення 1.4. Скалярним добутком двох векторів $\vec{p}_1$ та $\vec{p}_2$ простору $\mathbb{R}^m$ називають скалярну величину $\vec{p}_1 \cdot \vec{p}_2 = \vec{p}_1^T \vec{p}_2 = x_{11}x_{21} + x_{12}x_{22} + \dots + x_{1m}x_{2m}$.

Властивість 1.1. Скалярний добуток двох векторів $\vec{p}_1$ та $\vec{p}_2$ простору $\mathbb{R}^m$ обчислюється за формулою $\vec{p}_1 \cdot \vec{p}_2 = \|\vec{p}_1\|\|\vec{p}_2\| \cos\angle(\vec{p}_1, \vec{p}_2)$, де $\angle(\vec{p}_1, \vec{p}_2)$ – кут між векторами $\vec{p}_1$ та $\vec{p}_2$.

Означення 1.5. Вектори $\vec{p}_1$ та $\vec{p}_2$ простору $\mathbb{R}^m$ називають *колінеарними* (або *паралельними*), якщо $\exists\lambda \in \mathbb{R}, \lambda \neq 0 : \vec{p}_1 = \lambda\vec{p}_2$.

Означення 1.6. Два вектора $\vec{p}_1$ та $\vec{p}_2$ з простору $\mathbb{R}^m$ називають *перпендикулярними*, якщо $\vec{p}_1 \cdot \vec{p}_2 = 0$.

Означення 1.7. Векторним добутком двох векторів $\vec{p}_1 = (x_1, y_1, z_1)^T$ та $\vec{p}_2 = (x_2, y_2, z_2)^T$ з простору $\mathbb{R}^3$ називають наступний вектор:

$$\vec{p}_1 \times \vec{p}_2 = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}.$$

Для двох векторів $\vec{p}_1 = (x_1, y_1)^T$ та $\vec{p}_2 = (x_2, y_2)^T$ з простору $\mathbb{R}^2$ визначимо оператор $\times_Z$, що повертає z-координату векторного добутку векторів з координатами $(x_1, y_1, 0)^T$ та $(x_2, y_2, 0)^T$:

$$\vec{p}_1 \times_Z \vec{p}_2 = x_1 y_2 - x_2 y_1.$$

Означення 1.8. Впорядкована пара векторів $\vec{p}_1$ та $\vec{p}_2$ з простору $\mathbb{R}^2$ утворює *лівий поворот*, якщо $\vec{p}_1 \times_Z \vec{p}_2 > 0$, та *правий поворот*, якщо $\vec{p}_1 \times_Z \vec{p}_2 < 0$. Якщо $\vec{p}_1 \times_Z \vec{p}_2 = 0$, то вектори $\vec{p}_1$ та $\vec{p}_2$ є колінеарними.

Лівий поворот також називають *поворотом проти годинникової стрілки* (*англ. counterclockwise rotation, CCW*), а правий поворот називають поворотом за годинниковою стрілкою (*англ. clockwise rotation, CW*). Приклад пар векторів, що утворюють лівий та правий повороти зображено на рисунку 1.2.

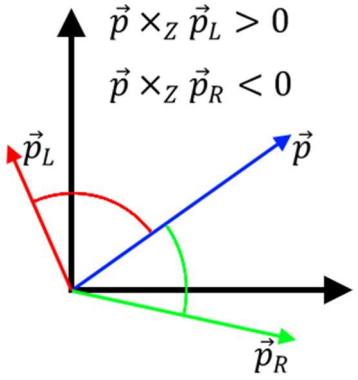

Рисунок 1.2. Лівий та правий поворот пари векторів

Означення 1.9. Нехай задано три точки $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$, та $p_3 = (x_3, y_3)$ з простору $\mathbb{R}^2$. Трикутник $p_1p_2p_3$ називається *лівоповоротним*, якщо рухаючись від точки p_1 до p_2 і потім до p_3 в точці p_2 відбувається лівий поворот. Невироджений трикутник, що не є лівоповоротним називається *правоповоротним*.

Орієнтацію трикутника $p_1p_2p_3$ можна визначити, обрахувавши наступний визначник:

$$\Delta_{p_1p_2p_3} = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = \overrightarrow{p_2 - p_1} \times_Z \overrightarrow{p_3 - p_1}$$

Якщо $\Delta_{p_1p_2p_3} > 0$, то трикутник $p_1p_2p_3$ є лівоповоротним. Інакше, якщо $\Delta_{p_1p_2p_3} < 0$, то трикутник $p_1p_2p_3$ є правоповоротним. Випадок $\Delta_{p_1p_2p_3} = 0$ означає, що трикутник є $p_1p_2p_3$ виродженим.

Означення 1.10. Нехай задано три точки $p_1 = (x_1, y_1, z_1), p_2 = (x_2, y_2, z_2)$, та $p_3 = (x_3, y_3, z_3)$ з простору $\mathbb{R}^3$. *Вектором нормалі* або *нормаллю* трикутника $p_1p_2p_3$ називають вектор $\overrightarrow{n_{p_1p_2p_3}}$, перпендикулярний до площини, в якій міститься трикутник $p_1p_2p_3$.

В алгоритмах обчислювальної геометрії прийнято обчислювати нормаль трикутника за формулою $\overrightarrow{n_{p_1p_2p_3}} = \overrightarrow{p_2 - p_1} \times \overrightarrow{p_3 - p_1}$, тим самим фіксуючи орієнтацію трикутника в просторі: вважається, що вектор нормалі спрямований назовні. Ця властивість є важливою, зокрема, в $3D$-моделюванні, оскільки узгоджені нормалі дозволяють описати межу тіла, заданого трикутниками.

Означення 1.11. *Прямою* в просторі $\mathbb{R}^m$, що проходить через точки p_1 та p_2 називають множину точок L_{p_1,p_2}, яка задається наступним чином:

$$L_{p_1,p_2} = \{p \in \mathbb{R}^m | p = tp_1 + (1-t)p_2, t \in \mathbb{R}\}$$

Означення 1.12. *Променем* в просторі $\mathbb{R}^m$, що починається в точці p_1 та проходить через точку p_2 називають множину точок R_{p_1,p_2}, яка задається наступним чином:

$$R_{p_1,p_2} = \{p \in \mathbb{R}^m | p = p_1 + t(p_2 - p_1), t \in \mathbb{R}\}$$

При цьому, вектор $\overrightarrow{d_{R_{p_1,p_2}}} = \overrightarrow{p_2 - p_1}$ називають напрямком R_{p_1,p_2}.

Означення 1.13. *Відрізком* з кінцями в точках $p_1, p_2 \in \mathbb{R}^m$ називають множину точок $\overline{p_1p_2}$, яка задається наступним чином:

$$\overline{p_1p_2} = \{p \in \mathbb{R}^m | p = p_1 + t(p_2 - p_1), t \in [0,1]\}$$

Означення 1.14. *Параметричним відрізком*, побудованим на точках $p_1, p_2 \in \mathbb{R}^m$ з параметрами t_{min}, $t_{max} \in \mathbb{R}^* = \mathbb{R} \cup \{\pm\infty\}$ називають множину точок $\overline{p_1p_2}[t_{min}, t_{max}]$, яка задається наступним чином:

$$\overline{p_1p_2}[t_{min}, t_{max}] = \{p \in \mathbb{R}^m | p = p_1 + t(p_2 - p_1), t \in [t_{min}, t_{max}]\}$$

Не важко бачити, що параметричний відрізок є досить гнучкою структурою, яка дозволяє узагальнити пряму, промінь та відрізок. При цьому дозволяється, щоб точки p_1 та p_2 знаходились за межами параметричного відрізка.

Означення 1.15. Множина $A \subset \mathbb{R}^m$ називається *опуклою*, якщо $\forall p_1, p_2 \in A$, $\forall p \in \overline{p_1 p_2}: p \in A$.

Означення 1.16. *Опуклою оболонкою* $CH(S)$ множини $S \subset \mathbb{R}^m$ називають найменшу опуклу підмножину $A(S) \subset \mathbb{R}^m$, що $A \subseteq S$.

Означення 1.17. *Ламаною* називають криву, що визначена послідовністю з n точок простору $\mathbb{R}^m$: $p_1, p_2, \dots, p_n$, де кожна пара послідовних точок сполучається відрізком $\overline{p_i p_{i+1}}, i = \overline{1, n-1}$. Точки $p_1, p_2, \dots, p_n$ називають *вершинами* ламаної, а відповідні відрізки – *ланками* ламаної.

Означення 1.18. Замкнену ламану з вершинами $p_1, p_2, \dots, p_n, p_{n+1}$, $p_1 = p_{n+1}$ називають *многокутником* або *полігоном*. Ланки такої ламаної називають *ребрами*. Для спрощення позначень $(n+1)$-шу вершину ламаної зазвичай не записують.

Означення 1.19. Многокутник називають *простим*, якщо він не містить самоперетинів.

Означення 1.20. Многокутник називають *плоским* або *планарним*, якщо всі його вершини лежать в одній площині. Зокрема, будь-який многокутник в $\mathbb{R}^2$ є плоским.

Теорема 1.1 (Формула площі Гаусса). Нехай задано многокутник P з n вершин $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \dots, p_n = (x_n, y_n)$ в просторі $\mathbb{R}^2$. *Орієнтованою площею* многокутника P називають значення:

$$S^{\pm}(P) = \frac{1}{2}\sum_{i=1}^{n}(x_{i+1} + x_i)(y_{i+1} - y_i), \text{ де } p_{n+1} = p_1.$$

Тоді *площу* многокутника P можна обчислити за формулою $S(P) = |S^{\pm}(P)|$.

Доведення теореми можна знайти в роботі (Art of Problem Solving, n.d.).

Означення 1.21. Нехай задано невироджений многокутник P з n вершин в просторі $\mathbb{R}^2$. Кажуть, що вершини многокутника P задані в порядку обходу проти годинникової стрілки, якщо $S^{\pm}(P) > 0$, а сам многокутник P називають *орієнтованим проти годинникової стрілки*. Інакше вершини многокутника P задані в порядку обходу за годинниковою стрілкою, а сам многокутник називають *орієнтованим за годинниковою стрілкою*.

Означення 1.22. *Многокутником з дірками* або *з отворами* називають зв'язну область на площині, границя якої складається з одного зовнішнього простого многокутника та одного або більше внутрішніх простих многокутників, що повністю містяться всередині зовнішнього многокутника та не мають перетинів між собою та з зовнішнім многокутником.

В алгоритмах обчислювальної геометрії прийнято задавати многокутник з дірками як набір простих многокутників, в якому зовнішній та внутрішні многокутники мають різну орієнтацію. Зазвичай, зовнішній многокутник є орієнтованим проти годинникової стрілки, в той час як внутрішні многокутники є орієнтованими за годинниковою стрілкою.

Означення 1.23. Плоский многокутник називають *опуклим*, якщо його внутрішня область є опуклою множиною.

Означення 1.24. Плоский многокутник називають *монотонним* відносно деякої прямої L, що лежить в площині многокутника, якщо будь-яка пряма, перпендикулярна до L перетинає многокутник не більше двох разів. Зазвичай, в просторі $\mathbb{R}^2$, в якості прямої L виступає одна з координатних осей. В такому випадку многокутник називають *X-монотонним*, або *Y-монотонним*.

Означення 1.25. Геометричне місце точок, відстань від яких до відрізка $\overline{p_1 p_2}$ в просторі $\mathbb{R}^m$ не перевищує δ називають *капсулою* та позначають $S_\delta(\overline{p_1 p_2})$. У випадку $m = 2$ також використовують назву *стадіон*.

1.4 Модель обчислень та поняття складності алгоритмів

Базовим елементом теорії складності обчислень є поняття *моделі обчислення*, що складається з множини допустимих операцій, які можуть бути використані при обчисленні, та оцінку вартості цих операцій (Savage, 1998). Визначення модель обчислення дозволяє оцінювати вартість виконання окремо взятого алгоритму, та порівняти її з вартістю виконання альтернативних алгоритмів, що розв'язують одну й ту ж задачу.

Всі оцінки алгоритмів, що будуть наведені в цій роботі виконано в рамках моделі машини з довільним доступом до пам'яті (*RAM-машини*), яка моделюється як пара зв'язаних між собою детермінованих скінченних автоматів: центрального процесора (*CPU*) та пам'яті з довільним доступом (*RAM*), кожна комірка якої може зберігати лише одне дійсне значення та має адресу, яка її однозначно ідентифікує (Savage, 1998).

CPU підтримує наступні елементарні операції, вартість яких є одиничною:

1. Арифметичні операції: $+, -, *, /$
2. Логічні операції порівняння двох дійсних чисел: $<, \leq, >, \geq, =$
3. Непряма адресація пам'яті за її адресою

В разі необхідності додатково можуть бути введені й інші елементарні операції (алгебраїчні, логічні, тригонометричні тощо).

Означення 1.26. *Часовою складністю алгорит*му називають суму вартостей всіх елементарних операцій, які виконує алгоритм в залежності від розміру вхідних даних.

Означення 1.27. *Просторовою складністю алгоритму* називають максимальну кількість комірок *RAM*, що використовуються алгоритмом в процесі його виконання.

Часова та просторова оцінки складності алгоритму зазвичай виражаються як функція від розміру вхідних даних $n \in \mathbb{Z}_+$. Варто зауважити, що для складних алгоритмів розмір вхідних даних може визначатись не як скалярне значення, а як

кортеж $(n_1, n_2, \dots, n_m) \in \mathbb{Z}_+^m$, однак подальші означення, наведені в цьому підрозділі вимагають саме скалярну оцінку розміру вхідних даних для забезпечення однозначності порівняння алгоритмів.

Означення 1.28. Нехай $n \in \mathbb{Z}_+$ - розмір вхідних даних алгоритму розв'язку деякої задачі T, а функція $f(n)$ виражає залежність часу роботи алгоритму від розміру вхідних даних. Гранична поведінка $f(n)$ при $n \to +\infty$ називається *асимптотичною часовою складністю алгоритму* розв'язання задачі T. Аналогічним чином визначається і *асимптотична просторова складність* алгоритму. Асимптотичну складність алгоритму прийнято обчислювати з точністю до множників та доданків нижчого порядку.

Для позначення верхньої оцінки складності алгоритму (*складність у найгіршому випадку*) використовується O-нотація:

$$O\big(f(n)\big) = \{g(n) | \exists c > 0 \ \wedge \ \exists n_0 > 0 : \ |g(n)| \leq c|f(n)|, \forall n \geq n_0\}$$

Аналогічним чином, для позначення нижньої оцінки складності алгоритму (*складність у найкращому випадку*) використовується Ω-нотація:

$$\Omega\big(f(n)\big) = \{g(n) | \exists c > 0 \ \wedge \ \exists n_0 > 0 : \ |g(n)| \geq c|f(n)|, \forall n \geq n_0\}$$

Для позначення співпадінні верхньої та нижньої оцінки складності алгоритму з точністю до деякого множника прийнято використовувати Θ-нотацію:

$$\Theta\big(f(n)\big) = \{g(n) | \exists c_1, c_2 > 0 \wedge \ \exists n_0 > 0 : c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|, \forall n \geq n_0\}$$

Також розрізняють середню оцінку складності алгоритму (складність в середньому випадку), що визначається як усереднена вартість роботи алгоритму по всім можливим вхідним даним.

Означення 1.29. Задача T називається P-повною, якщо існує алгоритм її розв'язання, що має поліноміальну оцінку складності в найгіршому випадку $O(n^k)$, де $k \in \mathbb{Z}$. Інакше задача T називається NP-повною.

При аналізі рекурсивних алгоритмів, часто функція оцінки часової складності $f(n)$ виражається у вигляді рекурентного співвідношення вигляду $f(n) = af\left(\frac{n}{b}\right) + h(n)$, де $a, b \geq 1$ – константи, $h(n)$ – оцінка часової складності розділення вхідних даних для підзадач та злиття їх результатів. Така ситуація часто зустрічається в алгоритмах побудови деревовидних структур даних, або при використанні алгоритмічної стратегії «розділяй та володарюй». Для такого випадку відомо класичний результат для визначення оцінки часової складності, що має назву Майстер-метод (Bentley et al., 1980):

Теорема 1.2. Майстер-метод (Bentley et al., 1980). Нехай оцінка часової складності задачі T виражається за допомогою рекурентного співвідношення $f(n) = af\left(\frac{n}{b}\right) + h(n)$, де $a, b \geq 1$ – константи, $h(n)$ – оцінка часової складності розділення вхідних даних для підзадач та злиття їх результатів. Тоді мають місце наступні оцінки часової складності алгоритму розв'язання задачі T:

1. Якщо $h(n) = O\left(n^{\log_b a - \varepsilon}\right)$, де $\varepsilon > 0$ – деяка константа, тоді $f(n) = O(n^{\log_b a})$
2. Якщо $h(n) = \Theta(n^{\log_b a})$, тоді $f(n) = \Theta(n^{\log_b a} \log n)$
3. Якщо $h(n) = \Omega(n^{\log_b a + \varepsilon})$, де $\varepsilon > 0$ – деяка константа, та функція $h(n)$ задовольняє умову $ah\left(\frac{n}{b}\right) \leq ch(n)$ для деякої константи $c < 1$ при достатньо великих значеннях n, тоді $f(n) = \Theta(h(n))$.

Доведення теореми 1.2 та детальне пояснення майстер-методу наведено в роботах (Bentley et al., 1980) та (Cormen et al., 2009, Chapter 4.5).

Нехай функції $f(n)$ та $g(n)$ – оцінки складності деяких двох задач. Будемо позначати $f \preccurlyeq g$, якщо виконується умова $f \in O(g(n))$. Мають місце наступні теореми.

Теорема 1.3. Оцінка суперпозиції задач (Cormen et al., 2009, Chapter 3.1) Нехай задача T_1 має асимптотичну оцінку складність $O(f_1(n))$, та її вихідний результат розмірності n є входом для задачі T_2, що має асимптотичну оцінку

складності $O(f_2(n))$. Тоді при виконанні умови $f_1 \preccurlyeq f_2$ отримуємо, що асимптотична оцінка складності суперпозиції задач T_1 та T_2 складає $O(f_2(n))$. Інакше, якщо виконується умова $f_2 \preccurlyeq f_1$, то суперпозиція задач T_1 та T_2 має оцінку складності $O(f_1(n))$.

Доведення. Розглянемо перший випадок $f_1 \preccurlyeq f_2$, адже другий випадок доводиться аналогічним чином за рахунок симетричності.

Умова $f_1 \preccurlyeq f_2$ означає, що $f_1 \in O(f_2(n))$. Тобто $\exists c > 0 \wedge n_0 > 0$, для яких справедлива нерівність $|f_1(n)| \leq c|f_2(n)|$, $\forall n \geq n_0$. Суперпозиція задач T_1 та T_2 має складність $f_1(n) + f_2(n)$. Тоді маємо, що $|f_1(n) + f_2(n)| \leq (c+1)|f_2(n)|$, $\forall n \geq n_0$, тобто $f_1(n) + f_2(n) \in O(f_2(n))$ згідно з означенням O-нотації. ■

Теорема 1.4. (Cormen et al., 2009, Chapter 3.1) Нехай маємо m задач $T_1, T_2, \dots, T_m$, які мають асимптотичні оцінки складності $O\big(f_1(n)\big), O\big(f_2(n)\big), \dots, O\big(f_m(n)\big)$ відповідно. Тоді їх суперпозиція має складність $\mathrm{O}\big(f(n)\big)$, де $f(n) = \max\limits_{1 \leq i \leq m} f_i(n) \stackrel{\text{def}}{=} f \in \{f_1, f_2, \dots, f_m\}, f_i \preccurlyeq f, i = \overline{1,m}$.

Доведення. Скористаємося методом математичної індукції. База індукції при $m = 2$ виконується за рахунок теореми 1.3. Перехід: припустимо, що твердження виконується для $m = k$. Тобто, для k задач $T_1, T_2, \dots, T_k$ з оцінками складності $O\big(f_1(n)\big), O\big(f_2(n)\big), \dots, O(f_k(n))$ їх суперпозиція має оцінку складності $O(g(n))$, де $g(n) = \max\limits_{1 \leq i \leq m} f_i(n)$. Випадок $m = k+1$ можна розглядати як суперпозицію $(k+1)$-ї задачі з оцінкою складності $O(f_{k+1}(n))$ з суперпозицією попередніх k задач, що, згідно з припущення, мають складність $O\big(g(n)\big)$. З теореми про оцінку суперпозиції задач слідує, що при $g \preccurlyeq f_{k+1}$ складність такої суперпозиції складає $O(f_{k+1}(n))$, а при $f_{k+1} \preccurlyeq g$ загальна оцінка складає $O(g(n))$. Тобто, суперпозиція $(k+1)$ задач має асимптотичну оцінку складності $O\big(f(n)\big) = \max\{f_{k+1}(n), g(n)\}$. ■

Визначення верхніх і нижніх оцінок складності є одним із основних, але водночас найскладніших етапів їх аналізу та проектування алгоритмів. Під час

такого аналізу часто застосовують метод спрощення, коли для отримання оцінки на алгоритм або множину допустимих вхідних даних накладають додаткові обмеження. Широковживаним є метод *зведення* задач (Cormen et al., 2009). Інтуїтивно, задача T є звідною до задачі T', якщо для будь-якого можливого набору вхідних даних її можна подати як випадок задачі T', розв'язок якої дає можливість визначити розв'язок T.

Означення 1.30. Задача T є звідною до задачі T', якщо:

1. Існує алгоритм перетворення вхідних даних задачі T у вхідні дані задачі T'
2. На перетворених вхідних даних розв'язується задача T'
3. Існує алгоритм, що перетворює розв'язок задачі T' в розв'язок задачі T

Нехай асимптотична оцінка часу виконання кроків 1 та 3 складає $O(r(n))$, де n – розмір вхідних даних задачі T. Тоді кажуть, що задача T є $r(n)$-звідною до задачі T', що позначається наступним чином: $T \propto_{r(n)} T'$. Відношення звідності задач $\propto_{r(n)}$ не є симетричним відношенням. Дві задачі T та T' називають *еквівалентними*, якщо одночасно виконуються умови $T \propto_{r(n)} T'$ та $T' \propto_{r(n)} T$. Якщо алгоритми, що виконують кроки 1 та 3 є P-повними, то кажуть, що задача T є *поліноміально звідною* до задачі T'.

Твердження 1.1. (Нижня оцінка методом зведення). Якщо відомо, що розв'язання задачі T вимагає $f(n)$ часу, і $T \propto_{r(n)} T'$, то задачу T' можна розв'язати за час не менший за $f(n) - O(r(n))$.

Твердження 1.2. (Верхня оцінка методом зведення). Якщо відомо, що розв'язання задачі T' вимагає $f(n)$ часу, і $T \propto_{r(n)} T'$, то задачу T можна розв'язати за час, що не перевищує $f(n) + O(r(n))$.

Детальне обґрунтування тверджень 1.1 та 1.2 наведено в роботах (Cormen et al., 2009, Chapter 34.3) та (Терещенко, 2011).

1.5. Ефективне представлення геометричних об’єктів в пам’яті комп’ютера

Важливим питанням при побудові мультиалгоритмічної програмної системи, що виконує обробку великих об’ємів геометричних даних є ефективне представлення геометричних об’єктів в пам’яті комп’ютера.

Одним із фундаментальних підходів до організації геометричної інформації в пам’яті комп’ютера, що використовується при розв’язанні задач обчислювальної геометрії та комп’ютерного моделювання є *індексне представлення* геометричних об’єктів. Його сутність полягає у відокремленні геометричного змісту (координати вершин) від топологічної структури. За такого представлення вершини зберігаються у вигляді масиву координат, тоді як кожен геометричний об’єкт описується набором індексів з набору вершин.

Використання такого підходу надає низку переваг, які є особливо важливими для програмних комплексів, що розраховані на обробку деталізованих геометричних об’єктів, які можуть складатись з десятків мільйонів вершин:

1. За індексного представлення координати спільних вершин кількох примітивів зберігаються лише один раз, що дозволяє істотно зменшити обсяг пам’яті, необхідної для опису складних геометричних об’єктів.
2. Індексна структура сприяє покращенню локальності посилань у пам’яті. (Nystrom, 2014). Оскільки вершини та індекси зберігаються у компактних лінійних масивах, операції над геометричними об’єктами (обхід, обчислення нормалей, тощо) виконуються з високою ефективністю завдяки зменшенню кількості кеш-промахів, що дає змогу прискорити обробку меш-структур на рівні CPU.
3. Будь-яка зміна координат вершини автоматично застосовується до всіх об’єктів, що її використовують. Завдяки цьому спрощується реалізація операцій редагування, деформації, тощо.
4. Індексне представлення дозволяє ефективно відновити топологічні зв’язки у вигляді реберного списку з подвійними зв’язками (РСПЗ), «крилатого»

представлення (Baumgart, 1972), або подібних структур даних, оскільки порівняння вершин відбувається на рівні цілочислових індексів, а не координат – чисел з рухомою комою.

5. Уніфікація представлення для різних типів об’єктів спрощує реалізацію універсальних алгоритмів, що можуть обробляти геометричні об’єкти різної розмірності та топології.
6. Індексне представлення відповідає формату представлення вхідних даних сучасних графічних API: OpenGL (Khronos Group, 2025a), DirectX (Microsoft, 2025), та Vulkan (Khronos Group, 2025b), що дозволяє ефективно організувати буфер вершин та буфер індексів, уникаючи необхідності конвертувати дані.

Враховуючи наведені вище переваги: компактність, структурну простоту та обчислювальну ефективність, алгоритми розроблені в цій роботі будуть проектуватись для роботи з індексним представленням геометричних даних.

1.6. Модель єдиного алгоритмічного середовища

Проблему неефективності мультиалгоритмічних програмних комплексів, що побудовані з використанням традиційного пакетно-алгоритмічного підходу вперше було досліджено в роботах (Tereshchenko & Anisimov, 2010) та (Терещенко, 2011). Ідея пакетно-алгоритмічного підходу полягає у використанні набору окремо взятих алгоритмів та структур даних у вигляді бібліотек, які часто є незалежними одна від одної. Архітектура бібліотек часто проектується опираючись на «принцип відкритості-закритості» (Meyer, 1997, pp 57-61), відомий постулат об’єктно орієнтованого програмування (ООП), який означає, що компоненти програми мають бути відкритими до розширення та закритими для змін. Такий підхід дозволяє збільшити надійність бібліотеки, оскільки внутрішня реалізація є захищеною від змін користувача, та може бути протестована. Однак, часто така архітектура значно обмежує можливість безшовної інтеграції між кількома бібліотеками. Наприклад, структури даних, побудовані однією бібліотекою, не завжди можуть бути використані іншою. Як

зазначається в (Tereshchenko, 2010), результатом використання такого підходу є набір алгоритмів з власними структурами даних, методами попередньої обробки, та навіть форматом вхідних та вихідних даних. Схематично, така модель представлена на рисунку 1.3. Як наслідок, під час роботи такої програмної системи часто доводиться виконувати зайві або надлишкові обчислення, та конвертувати данні з одного формату в інший. Таким чином, традиційний пакетно-алгоритмічний підхід не дозволяє розробляти мультиалгоритмічні програмні комплекси, що були б ефективні за своїм дизайном.

В якості альтернативи пакетно-алгоритмічному підходу, в роботах (Tereshchenko & Anisimov, 2010), (Tereshchenko, 2010) та (Терещенко, 2011) було запропоновано модель єдиного алгоритмічного середовища (МЄАС). Концепція МЄАС полягає в проектуванні програмного комплексу з врахуванням взаємозв'язків між задачами, що постають в проблемній області, що досліджується. Опираючись, зокрема, на поняття звідності задач, таких підхід дозволяє виокремити спільні частини декількох алгоритмів, наприклад попередню обробку, виконати її один раз, та використати наявну інформацію про звідність задач, для уникнення зайвих обчислень.

Вперше ефективність використання МЄАС було продемонстровано на взаємозв'язаних задачах обчислювальної геометрії на площині: побудові опуклої оболонки, тріангуляції, пошук найближчого сусіда, тощо. В роботах (Tereshchenko, 2010) та (Терещенко, 2011) показано, що використання МЄАС на базі діаграми Вороного та алгоритмічної стратегії «розділяй та володарюй» дозволяє отримати значний приріст ефективності при розв'язуванні задач обчислювальної геометрії. Також, в роботі (Коцур, 2019) розроблено МЄАС для деяких задач аналізу та обробки зображень (сегментація, відслідковування об'єктів, побудова серединної осі об'єкта на зображенні, тощо). Автором показано, що застосування МЄАС з етапом оптимізації дозволяє на порядок прискорити алгоритми розв'язання задач обробки зображень. Крім того, існує ряд досліджень щодо побудови єдиного алгоритмічного фреймворку для задач

машинного навчання та аналізу даних, зокрема для тестування різноманітних алгоритмів навчання з підкріпленням (Chen et al., 2021), для уніфікації варіаційних методів виявлення опцій з методами, що використовують лапласіан в некерованому машинному навчанні (Aggarwal et al., 2023), та для оптимізації аналізу великих даних шляхом узагальнення багатьох відомих підходів до роботи з великими наборами даних (Hong et al., 2016).

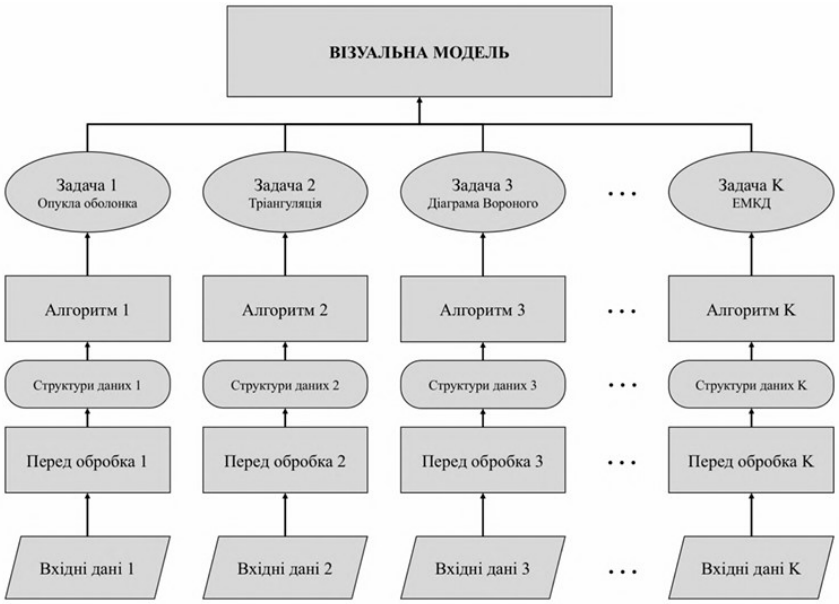

Рисунок 1.3 Схематичне представлення пакетно-алгоритмічного підходу (Терещенко, 2011)

Висновки до розділу 1

В першому розділі дослідження здійснено комплексний аналіз сучасного стану технологій адитивного виробництва (АВ) та особливостей процесу підготовки моделей до 3D-друку. Проведений огляд дозволив систематизувати основні технології АВ за їх класом згідно стандарту ISO/ASTM 52900:2021 (International Organization for Standardization, 2021), а також за типом та станом матеріалу, який підтримує та чи інша технологія АВ. Результати систематизації представлені у вигляді узагальнюючої схеми (рисунок 1.4), що використовує наступні позначення: вузли прямокутної форми позначають технології друку, що використовують матеріал у твердому вигляді, овальні – матеріалами у рідкому стані, а шестикутні – порошкоподібні матеріали. Тип матеріалу ідентифікується

за кольором контуру вузла, де синій відповідає полімерам, червоний – металам, зелений – кераміці, а помаранчевий – іншим специфічним матеріалам. Наявність кількох контурів у вузла означає, що відповідна технологія може працювати з кількома типами матеріалів.

Показано, що через фундаментальні відмінності у фізичних принципах різних технологій 3D-друку, задачі підготовки цифрової моделі до виробництва, такі як вибір просторової орієнтації та генерація опорних структур, є специфічними для кожної технології та не можуть бути зведені до єдиного універсального алгоритму. Це створює значні виклики для побудови ефективного універсального програмного забезпечення, здатного розв'язувати комплекс задач планування для всіх технологій АВ.

Огляд наявних на ринку програмних рішень показав, що сучасний процес планування АВ, зазвичай, вимагає застосування кількох незалежних додатків для розв’язання різних задач. Існуючі системи базується на традиційному пакетно-алгоритмічному підході, який передбачає використання набору незалежних закритих модулів та бібліотек, що не лише ускладнює розширення системи та інтеграцію нових задач та алгоритмів, але й зумовлює обчислювальну неефективність, оскільки під час роботи таких комплексів часто виникає необхідність виконувати зайві або надлишкові обчислення та конвертувати дані з одного формату представлення в інший.

В підрозділах 1.3 та 1.4 наведена основна теоретична та методологічна база дослідження, що використовується в подальших розділах цієї дисертації.

Нарешті, в підрозділах 1.5 та 1.6 розглядаються існуючі підходи до побудови ефективних алгоритмів розв’язання задач обчислювальної геометрії, та мультиалгоритмічних платформ, що призначені для розв’язання комплексу взаємопов’язаних задач з певної проблемної області. Підрозділ 1.5 обґрунтовує доцільність застосування індексного представлення геометрії для оптимізації

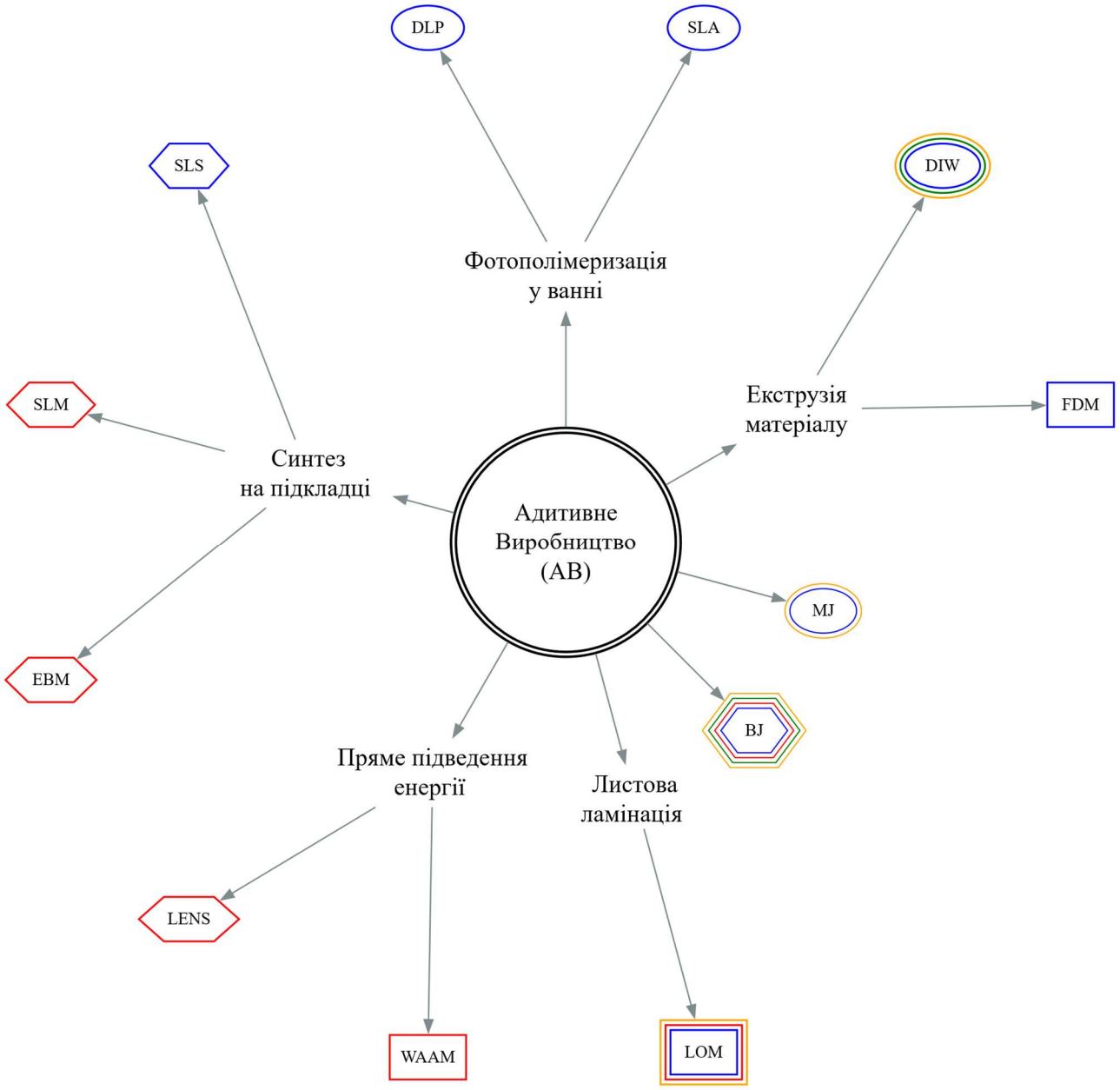

Рисунок 1.4. Схематичне зображення основних технологій адитивного виробництва

використання пам’яті та прискорення доступу до даних. В підрозділі 1.6 розглядаються переваги моделі єдиного алгоритмічного середовища (МЄАС) як основи для створення високоефективних систем. Завдяки врахуванню взаємозв’язків між задачами, спираючись на теорію звідності, МЄАС дозволяє мінімізувати обчислювальні витрати шляхом усунення надлишкових операцій.

РОЗДІЛ 2. КОМПЛЕКС ЗАДАЧ ПРОЦЕСУ ПЛАНУВАННЯ АДИТИВНОГО ВИРОБНИЦТВА

Як зазначалось в підрозділі 1.2, на сьогодні не існує універсального програмного рішення, яке б ефективно розв'язувало комплекс задач процесу планування адитивного виробництва. Окрім того, не існує несуперечливого і водночас ненадлишкового набору задач, який би повною мірою задовольняв вимоги всіх існуючих методів 3D-друку, що є наслідком різноманітності фізичних процесів, покладених в основу технологій АВ.

В цьому розділі буде формалізована постановка основних задач процесу планування АВ (задачі 1-5 розділу 1.2), які є актуальними для більшості існуючих технологій. З огляду на специфіку окремо взятих виробничих процесів розглянутий набір задач не є вичерпним, однак він дає достатнє усвідомлення варіативності алгоритмічних підходів, що застосовуються під час підготовки моделей до друку. Це, своєю чергою, створює підґрунтя для формулювання критеріїв до універсальної системи планування процесу адитивного виробництва та розроблення її архітектури на основі МЄАС у наступних розділах роботи.

2.1 Генерування мешів (Задача 1)

Задача генерування мешів постає на різних етапах процесу планування адитивного виробництва. По-перше, дизайн вхідних моделей для друку зазвичай розробляється в системах автоматизованого проєктування і розрахунку (САПР, *англ.* Computer-aided design, CAD), вихідними даними яких є файл, що містить інформацію про представлення границі моделі у вигляді орієнтованих граней та кривих. Прикладом такого файлового формату є .STEP (International Organization for Standardization, 2024). Однак де-факто стандартом представлення моделей в адитивному виробництві є формат STL (Library of Congress, 2025), що являє собою набір трикутників та їх нормалей, заданих орієнтацією трикутника відповідно до означення 1.10. Таким чином, важливою задачею процесу планування АВ є конвертування даних з CAD формату в STL.

Іншою варіацією задачі генерації мешів є задачі модифікації геометрії моделі. До таких модифікацій можна віднести додавання відступу, заокруглення ребер та кутів, генерування пустот всередині моделі, булеві операції над моделями (об'єднання, перетин та різниця моделей) тощо.

Підходи до розв'язання задачі генерування мешів доцільно розділити на дві групи: загальні та спеціалізовані. Загальні підходи можуть бути застосовані до широкого спектра постановок задачі генерування мешів незалежно від конкретних умов або типу вхідних даних. Натомість, спеціалізовані підходи орієнтовані на розв'язання окремих, чітко визначених постановок задачі та враховують специфіку конкретних геометричних або технологічних вимог.

2.1.1 Конвертування CAD в STL

CAD-дані описують геометрію довільної форми та кривизни з використанням аналітичних поверхонь, зокрема поверхонь Безьє або неоднорідних раціональних B-сплайнів (NURBS). Як наслідок, меш, отриманий у результаті конвертації, є лише наближенням поверхні вихідної CAD-моделі. Точність такої апроксимації може бути задана параметрами алгоритму конвертації або встановлюватися автоматично з врахуванням розмірів моделі та технічних характеристик апаратного обладнання, наприклад товщини лазерного променя.

Вхідні дані: модель в CAD форматі.

Вихідні дані: меш трикутників, що з наперед заданою або автоматично визначеною точністю апроксимує геометрію вхідної моделі з врахуванням її початкової просторової орієнтації.

Поширеним підходом до конвертації CAD-даних в меш трикутників є формування полігональних контурів на основі граничних кривих, подальша тріангуляція обмеженої ними області та уточнення отриманої тріангуляції з врахуванням точок, що знаходяться на криволінійних поверхнях. Алгоритм, що

використовує такий підхід запропоновано в роботі (Yang & Choi, 2010). Процес побудови мешу складається з наступних чотирьох етапів:

1. Побудова обмеженої тріангуляції Делоне (ОТД), де в якості обмежень виступають ребера, що апроксимують межі граней CAD-моделі.
2. Побудова *kD*-дерева для вершин ОТД.
3. Уточнення тріангуляції шляхом додавання внутрішніх точок поверхні.
4. Видалення надлишкових трикутників, розташованих поза межами вхідної моделі.

На першому етапі алгоритм перетворює криві, що задають границю моделі, на ламані лінії, які її апроксимують. В процесі побудови ламаних, відрізки послідовно додаються до ОТД, формуючи початкову структуру меша.

На другому етапі здійснюється побудова *kD*-дерева для множини вершин відрізків, отриманих на попередньому кроці. Для зберігання тріангуляції використовується модифікована структура даних interworld (Kim et al., 2006). Оскільки вершини трикутників в цій структурі подано у вигляді звичайного масиву, *kD*-дерево реалізується у вигляді перестановки його елементів, що дає змогу уникнути додаткових витрат пам’яті та цілком відповідає принципам ефективного представлення геометричних об’єктів в пам’яті комп’ютера, описаним в підрозділі 1.5.

На третьому етапі відбувається апроксимація кривизни поверхні. Для цього параметрична область поверхні дискретизується за допомогою регулярної сітки. Якщо відхилення центральної точки комірки сітки від відповідної ділянки поверхні перевищує наперед задану похибку апроксимації, виконується подальше уточнення розбиття. Після досягнення необхідної точності ОТД доповнюється новими трикутниками, одна з вершин яких відповідає доданій точці на поверхні моделі, тоді як інші вершини визначаються відповідно до поточної конфігурації тріангуляції.

На завершальному етапі здійснюється фільтрація ОТД: трикутники, що розташовані поза межами поверхні моделі, видаляються. Автори зазначають, що обчислювальна складність алгоритму в найгіршому випадку становить $O(n\log n)$, де n – кількість відрізків, які задають границю поверхні.

Альтернативний підхід до конвертування CAD-даних в меш трикутників запропоновано в роботі (Yu et al., 2021). Як зазначають автори, меші, отримані внаслідок перетворення з CAD-форматів, часто мають специфічні дефекти, зокрема самоперетинами, неузгодженістю суміжних трикутників та наявністю дірок. Для усунення цих та інших недоліків на практиці застосовують різноманітні методи відновлення мешів, детальний огляд яких наведено в роботі (Campen et al., 2012). Водночас, точне відтворення поверхні не завжди є можливим без наявності додаткової інформації про модель. З огляду на це, в роботі Yu et al. запропоновано підхід до синхронізації та вирівнювання меша в процесі конвертування, що забезпечує об’єднання мешів окремих кривих і поверхонь CAD-даних у цілісну структуру без погіршення якісних характеристик результуючого меша.

2.1.2 Булеві операції над мешами

Окремою постановкою задачі генерування мешів є виконання булевих операцій над ними. Зазвичай, розглядають три базові булеві операції: об'єднання, перетин та різницю, які формально визначаються наступним чином.

Означення 2.2. Нехай M_1 та M_2 – деякі меші (множини трикутників) в просторі $\mathbb{R}^3$. Тоді множина $M_\cup$, для якої виконується умова $\forall x \in \mathbb{R}^3\ x\ \in M_\cup \Leftrightarrow \mathrm{x} \in \mathrm{M}_1 \vee \mathrm{x} \in \mathrm{M}_2$ називається об'єднанням мешів M_1 та M_2. Множина $M_\cap$, для якої виконується умова $\forall x \in \mathbb{R}^3\ x\ \in M_\cap \Leftrightarrow \mathrm{x} \in \mathrm{M}_1 \wedge \mathrm{x} \in \mathrm{M}_2$. Різницею мешів називають множину $M_\setminus$, для якої виконується умова $\forall x \in \mathbb{R}^3\ x\ \in M_\setminus \Leftrightarrow \mathrm{x} \in \mathrm{M}_1 \wedge \mathrm{x} \notin \mathrm{M}_2$.

Булеві операції в задачах процесу планування АВ використовуються в алгоритмах генерування опорних конструкцій, внутрішніх порожнин, побудови складних поверхонь у вигляді композиції примітивів тощо.

Задачу виконання булевої операції над двома мешами можна формалізувати таким чином:

Вхідні дані: M_1, M_2 *– два меша трикутників, а також задана булева операція, яку необхідно виконати (об'єднання, перетин або різниця).*

Вихідні дані: результуючий меш $M_\cup, M_\cap$ *або* $M_\setminus$*, відповідно до заданої операції.*

Одним із поширених підходів до реалізації булевих операцій є використання алгоритмів відтинання, зокрема описаних в (Scherzinger et al., 2017). Однак більш доцільним є застосування спеціалізованих алгоритмів, розроблених безпосередньо для розв'язання задачі виконання булевих операцій над мешами. Один із таких алгоритмів запропоновано в роботі (Jiang et al., 2016).

На першому кроці роботи алгоритм апроксимує спільну область двох мешів шляхом побудови дерев октантів (Meagher, 1980) для кожного з мешів та виділяє їх спільну частину. У вузлах сформованого спільного октодерева визначаються пари трикутників, що належать різним мешам і перетинаються між собою. Для кожної такої пари обчислюються відрізок перетину, після чого початкова тріангуляція обох моделей деталізується шляхом додавання нових вершин та ребер на відрізку перетину трикутників.

Далі визначається просторове положення вершин трикутників, що перетинаються, відносно іншого меша. Відповідно до результатів класифікації вершинам присвоюються мітки: «всередині», «ззовні» або «на поверхні». Використовуючи ці мітки, формуються наступні п'ять множин трикутників:

- $M_{M_1 in M_2}$ – множина трикутників з M_1, що розташовані всередині M_2
- $M_{M_1 out M_2}$ – множина трикутників з M_1, що розташовані ззовні M_2

- $M_{M_2 in M_1}$ – множина трикутників з M_2, що розташовані всередині M_1
- $M_{M_2 out M_1}$ – множина трикутників з M_2, що розташовані ззовні M_1
- $M_{on M_1 M_2}$ – множина трикутників, що належать одночасно M_1 та M_2

Тоді результати булевих операцій визначаються такими співвідношеннями:

$$M_{\cup} = M_{M_1 out M_2} \cup M_{M_2 out M_1} \cup M_{on M_1 M_2}$$

$$M_{\cap} = M_{M_1 in M_2} \cup M_{M_2 in M_1} \cup M_{on M_1 M_2}$$

$$M_{\setminus} = M_{M_1 out M_2} \cup M_{M_2 in M_1}$$

2.1.3 Загальні алгоритми генерування мешів

В цьому підрозділі розглядаються алгоритми генерування мешів, які можуть бути безпосередньо застосовані або адаптовані до широкого класу задач, що виникають як в процесі планування АВ (зокрема, розглянуті в підрозділах 2.1.1 та 2.1.2 задачі конвертування CAD-даних в меш та виконання булевих операцій над мешами), так і загалом під час роботи з тривимірними геометричними об'єктами.

Вхідними даними для алгоритмів цього є функція відстані зі знаком, що дає змогу задавати цільову поверхню в неявному вигляді.

Означення 2.1. Нехай $D \subset R^N$ – підмножина Евклідового простору розмірності $N \geq 2$. Позначимо через $\partial D \neq \emptyset$ – границю D в R^N, $\overline{D}$ – замикання множини D в R^N, $d(x, y)$ – метрику простору R^N. Функцією відстані зі знаком $f(x)$, де $x \in R^N$, для множини D визначають наступним чином:

$$f(x) = \begin{cases} d_{\partial D}(x), & x \in \overline{D} \\ -d_{\partial D}(x), & x \in R^N \setminus \overline{D} \end{cases}, \text{ де } d_{\partial D}(x) = \inf_{y \in \partial D} d(x, y).$$

Тоді загальну задачу генерування меша можна сформулювати наступним чином:

Вхідні дані: $f(x)$ - функція відстані зі знаком для множини $D \subset R^3$, яка задає границю моделі.

Вихідні дані: меш трикутників, що репрезентує ∂D.

Одним із перших підходів до розв'язання задачі генерування меша за заданою функцією відстані зі знаком є алгоритм Marching Cubes (MC), описаний в роботі (Lorensen & Cline, 1987). Він розроблявся як метод відновлення тривимірних моделей за медичними даними, зокрема результатами комп'ютерної томографії та магнітно-резонансної томографії. Згодом алгоритм набув широкого застосування в різних галузях, зокрема в адитивному виробництві. На сьогодні MC залишається одним із найпоширеніших методів побудови мешів на основі функції відстані зі знаком.

На першому етапі алгоритму MC простір дискретизується рівномірною тривимірною сіткою із заздалегідь заданим кроком. У вершинах сітки обчислюються значення функції відстані зі знаком $f(x)$. Очевидно, що ті вокселі, у вершинах яких значення $f(x)$ змінює знак, мають непорожній перетин із межею ∂D. Варто зауважити, що обернене твердження є хибним у загальному випадку. Найпростішим контрприкладом є ситуація, коли функція $f(x)$ задає область D, що повністю збігається з одним із вокселів сітки. Проте, на практиці подібні випадки обходяться шляхом вибору достатньо малого кроку дискретизації, що забезпечує належну точність апроксимації.

Алгоритм аналізує всі 256 можливих конфігурацій знаків значень $f(x)$ у вершинах кубів, які шляхом повороту та відображення зводяться до 15 базових випадків, які наведено на рисунку 2.1.

Після визначення конфігурації, що відповідає кожній комірці сітки, алгоритм розміщує вершини меша на ребрах відповідно до конфігурації. Положення вершин меша визначається за допомогою лінійної інтерполяції значень функції відстані зі знаком на кінцях цього ребра. Далі до меша додаються трикутники, що відповідають встановленій конфігурації, формуючи локальну апроксимацію поверхні в межах даної комірки.

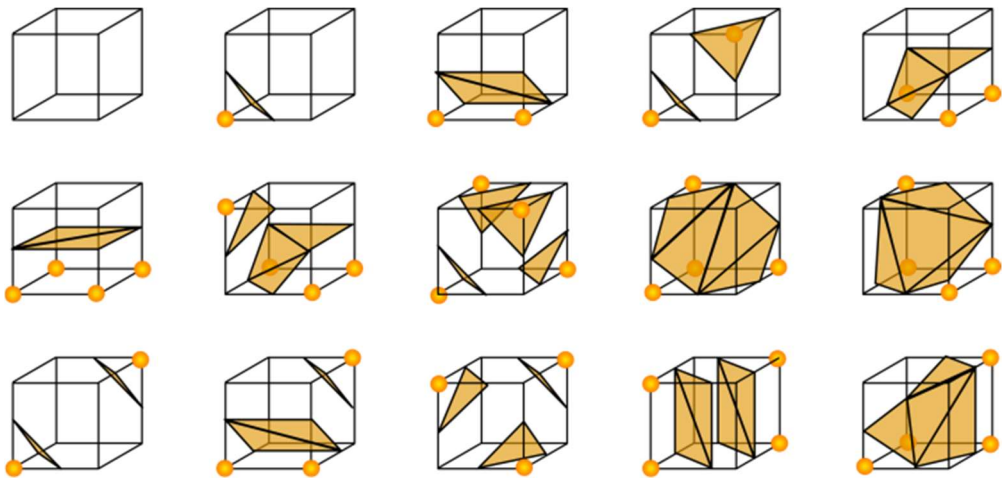

Рисунок 2.2. 15 базових конфігурацій вузлів кубів алгоритму MC (Ryoshoru, 2023)

Суттєвим недоліком такого підходу є низька якість апроксимації ділянок поверхні з різко вираженими геометричними особливостями. Зокрема, у разі застосування алгоритму Marching Squares (MS), який є двовимірним аналогом MC, для побудови квадрата, замість очікуваних прямих кутів отримується фігура зі зрізаними кутами (рисунок 2.2).

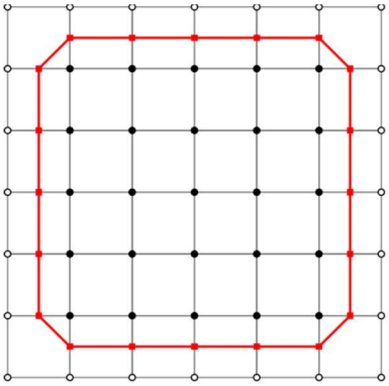

Рисунок 3.2. Апроксимація квадрата алгоритмом MS (Newgas, 2018)

Очевидно, що якість апроксимації можна підвищити шляхом зменшення довжини ребра комірки дискретизації. Проте навіть за такого підходу досягти точної передачі прямих кутів неможливо, тоді як кількість кубів, що обробляються алгоритмом, істотно зростає, що призводить до збільшення обчислювальних витрат.

Іншим недоліком алгоритму MC є наявність неоднозначних конфігурацій вузлів, які потребують застосування певної стратегії їх вирішення, оскільки

непослідовне трактування неоднозначних конфігурацій може спричинити утворення розривів у меші. Вибір такої стратегії може відрізнятися залежно від вимог до алгоритму та сфери його застосування. При цьому стратегії, що враховують конфігурації вузлів сусідніх кубів істотно ускладнюють можливість паралельної реалізації алгоритму, адже в такому разі обробка окремих кубів не є незалежною процедурою.

З метою подолання зазначених вище недоліків в роботах (Kobbelt et al., 2001), (Ho et al., 2005) та (Schaefer et al., 2007) було запропоновано низку альтернативних підходів. Серед них особливої уваги заслуговує алгоритм Cubical Marching Squares (CMS), запропонований в (Ho et al., 2005), оскільки він забезпечує коректну репрезентацію гострих ділянок поверхні, водночас зберігаючи незалежність обробки окремих кубів.

Ключова ідея алгоритму CMS полягає у зведенні тривимірної задачі полігоналізації до двовимірної: замість аналізу тривимірного куба, як це передбачено в алгоритмі MC, розглядаються шість двовимірних квадратів, отриманих шляхом розгортання граней куба.

Для дискретизації простору алгоритмiCMS використовує дерево октантів (Meagher, 1980), підрозбиття якого виконується доти, доки всі вершини-листки не матимуть однозначної конфігурації вузлів і не будуть містити «складну» поверхню, евристичний критерій оцінки якої також пропонується авторами алгоритму. Після дискретихації, для всіх вершин-листків генерується набір сегментів із використанням алгоритму MS. На останньому етапі роботи алгоритм відновлює поверхню за побудованими сегментами та виконує її тріангуляцію.

2.2 Розміщення моделей в межах робочої області принтера (Задача 2)

Задача розміщення моделей в межах робочої області 3D-принтера належить до ширшого класу оптимізаційних задач – задач пакування. З прикладної точки зору вона є важливим етапом процесу планування АВ, оскільки

раціональне використання робочого об’єму забезпечує збільшення кількості деталей, що виготовляються одночасно, і, відповідно, підвищує продуктивність виробничого процесу. При цьому значна частина задач пакування належить до класу NP-складних задач (Pankratov et al., 2020). Як наслідок, більшість досліджень задач пакування спрямовані на пошук ефективних оптимізаційних алгоритмів, здатних за прийнятний обчислювальний час знаходити розв’язки, що адекватно апроксимують глобально-оптимальний.

Формально задачу розміщення моделей в межах робочої області принтера можна сформулювати наступним чином:

Вхідні дані: множина тривимірних моделей та розміри робочої області принтера.

Вихідні дані: множина матриць трансформацій, застосування яких до відповідних моделей з вхідної множини забезпечує їх розміщення в межах робочої області принтера.

Цікавий підхід до розв’язання цієї задачі запропоновано в роботі (Lutters et al., 2012). Ключова ідея алгоритму полягає у використанні процедури, що моделює феномен гранулярної конвекції, також відомий як ефект бразильського горіха (Rosato et al., 1987) – фізичне явище сегрегації суміші частинок різного розміру під час струшування, внаслідок якого більші частинки піднімаються догори.

Запропонований алгоритм складається з двох основних етапів:

1. Оптимізація орієнтації деталей.
2. Побудова початкового розміщення з подальшим його покращенням.

Для визначення оптимальної орієнтації моделі автори розглядають наступні критерії:

i. Мінімізація об’єму мінімального обмежувального паралелепіпеда, вирівняного за осями координат.

ii. Мінімізація площі поверхні деталі, яка не є паралельною або перпендикулярною до напрямку друку, за алгоритмом з роботи (Lan et al., 1997).

Перший критерій (i) сприяє більш компактному розміщенню моделі в робочому просторі принтера, в той час як другий критерій (ii) дозволяє зменшити кількість нависаючих елементів, що потребують опорних структур, та покращити якість результуючої поверхні.

Алгоритм Lutters et al. об'єднає зазначені критерії в єдиний скалярний критерій шляхом введення вагових коефіцієнтів. Такий підхід надає можливість користувачу алгоритму регулювати відносний внесок кожного критерію в процес вибору орієнтації деталі залежно від технології друку, властивостей матеріалу, вимог до якості поверхні, необхідності мінімізації опорних структур та інших технологічних обмежень.

Після визначення оптимальної орієнтації деталей алгоритм формує початкове розміщення в межах робочої області принтера випадковим чином, а далі імітує застосування вібрацій до моделей. Моделювання вібрацій є обчислювально складною процедурою, оскільки на кожному кроці необхідно забезпечувати відсутність перетинів та взаємних замикань між деталями. З метою зменшення обчислювальної складності перевірок на перетин алгоритм застосовує різні методи апроксимації геометрії моделей, зокрема: обмежувальні сфери, мінімальні обмежувальні паралелепіпеди, орієнтовані мінімальні обмежувальні паралелепіпеди, дискретні орієнтовані многогранники (k-DOP) та випуклі оболонки. Вибір конкретного способу апроксимації може змінюватися під час виконання алгоритму, що дозволяє керувати компромісом між точністю перевірок на перетин, якістю підсумкового розміщення деталей та витратами часу на його побудову.

Інший підхід до розв'язання задачі пакування в контексті адитивного виробництва запропоновано в роботі (Pankratov et al., 2020). Автори формулюють задачу пакування як задачу оптимізації, цільовою функцією якої є мінімізація

об’єму контейнера, необхідного для розміщення заданої множини моделей. Як обмеження розглядаються умови належності всіх моделей контейнеру та відсутності перетинів між ними.

Для аналітичного подання зазначених обмежень автори застосовують апарат phi-функцій (Chernov et al., 2010) та квазі-phi-функцій (Stoyan et al., 2019). Зазначається, що отримана оптимізаційна задача містить $O(N^2)$ нелінійних нерівностей та $O(N^2)$ змінних, де N – кількість вхідних моделей.

З огляду на квадратичне зростання кількості змінних та обмежень, застосування універсальних програмних пакетів глобальної оптимізації для безпосереднього розв’язання цієї задачі є обчислювально витратним і потребує значного часу. У зв’язку з цим авторами запропоновано алгоритм пошуку локального оптимуму, результат якого може бути використаний або як початкове наближення для подальшого пошуку глобального оптимуму, або як практично прийнятна апроксимація розв’язку задачі пакування.

2.3 Генерування опорних конструкцій (Задача 3)

Під час друку необхідно забезпечити її надійне з’єднання з платформою принтера або з попередньо сформованими шарами. В протилежному випадку, в залежності від обраної технології АВ, істотно зростає ймовірність виникнення дефектів таких як провисання матеріалу, викривлення геометрії або навіть повного руйнування деталі в процесі виготовлення.

В багатьох галузях, де застосується адитивне виробництво, зокрема в авіабудуванні, медицині та інших високотехнологічних сферах, навіть незначне відхилення геометрії поверхні від проєктної є неприпустимим. Для забезпечення виготовлення якісних деталей складної форми разом із основною моделлю часто необхідно формувати та друкувати додаткові опорні структури, які стабілізують процес побудови та запобігають появі дефектів.

Задачу генерування опорних конструкцій будемо формулювати наступним чином:

Вхідні дані: модель із фіксованою орієнтацією.

Вихідні дані: опорні структури, що забезпечують технологічну можливість коректного виготовлення вхідної моделі за заданих умов друку.

Одним із поширених підходів до побудови опорних конструкцій є використання наперед заданих геометричних примітивів, які комбінуються між собою. Зокрема, в роботі (Vaidya & Anand, 2016) в якості базових елементів застосовуються зрізані октаедри та ромбічні додекаедри з внутрішніми порожнинами.

На першому етапі алгоритму виконується вокселізація моделі: початково геометрія апроксимується кубами, після чого кубічна воксельна структура трансформується в представлення, де заповнені вокселі відображуються в обрані геометричні примітиви. Також на цьому кроці визначаються інтерфейсні примітиви, тобто ті, що безпосередньо контактують із поверхнею моделі.

Серед інтерфейсних примітивів виділяються групи сусідніх елементів розміром 2×2 та 3×3, розташовані на однаковій висоті. Для кожної такої групи формуються вертикальні опорні структури.

На наступному етапі, на основі отриманої конфігурації примітивів будується зважений граф, який використовується для побудови опор для інтерфейсних примітивів, що не належать жодній з утворених груп. Для кожного такого елемента визначається найближчий примітив, який з’єднує одну з вже побудованих вертикальних опор із платформою принтера або моделлю. Між двома вершинами графа, що відповідають цим елементам, знаходиться найкоротший шлях за допомогою алгоритму (Dijkstra, 1959). На примітивах, що утворюють знайдений шлях, формується додаткова опорна конструкція. Після побудови опори ваги ребер графа оновлюються, і процес повторюється до повного забезпечення підтримки всіх інтерфейсних елементів.

Інший підхід до генерування деревовидних опорних структур запропоновано в роботі (Zhang et al., 2020). На першому етапі алгоритм знаходить нависаючі ділянки моделі, для яких необхідне формування опор. Як критерій нависання автори використовують кут нахилу трикутників відносно напряму друку: такими, що потребують опори, вважаються трикутники, кут нахилу яких перевищує $45°$.

На визначених трикутниках будуються точки кріплення майбутніх опор до поверхні моделі. Для цього трикутники, що підлягають підтримці, проєктуються на площину XOY. На площині будується рівномірна двовимірна сітка, після чого визначається такий її кут повороту, який мінімізує кількість вузлів сітки, що потрапляють в середину проєкцій нависаючих трикутників. Точки приєднання опор до моделі отримують шляхом зворотної проекції відповідних точок сітки на поверхню тривимірної моделі.

Математична модель деревовидних опорних структур описується на основі теорії L-систем (Rozenberg & Salomaa, 1976). З метою забезпечення технологічної придатності згенерованих дерев до друку, автори виділяють чотири базові типи параметричних L-систем, для яких фіксуються допустимі кути розгалуження гілок: $11.25°$, $15°$, $22.5°$ та $45°$. Параметрами систем є початкова довжина стовбура та коефіцієнт масштабування довжини стовбура на кожному етапі розгалуження.

Процес синтезу опор формалізується як задача багатокритеріальної оптимізації: цільовою функцією, що мінімізується, є об’єм дерев та кількість перетинів із моделлю. Задача залежить від п’яти змінних: тип L-системи, початкової довжини стовбура, коефіцієнт масштабування, позиція дерева в просторі та його орієнтація.

Для розв’язання побудованої оптимізаційної задачі використовується генетичний алгоритм NSGA-II (Deb et al., 2002), результатом роботи якого є множина Парето-оптимальних розв’язків. Для кожного з отриманих варіантів

виконується симуляція теплообміну з метою оцінювання термічних деформацій, після чого обирається конфігурація дерева, що забезпечує мінімальний рівень теплової деформації.

На завершальному етапі здійснюється постобробка деревовидної структури: гілки, що перетинаються з моделлю, рекурсивно розділяються на менші, згідно з правилами L-системи, а їхні кінцеві елементи замінюються конічними фрагментами для спрощення подальшого видалення опор після друку.

2.4 Слайсинг (Задача 4)

На завершальному етапі роботи з тривимірною моделлю під час планування АВ виконується її слайсинг – розбиття геометрії на послідовність тонких шарів площинами вздовж осі z. Отримані слайси є прототипами фізичних шарів, які формуються принтером в процесі пошарового виготовлення деталі.

Розрізняють два основні підходи до слайсингу: рівномірний та адаптивний. У випадку рівномірного слайсингу товщина всіх шарів є сталою по всій висоті моделі. Натомість, адаптивний слайсинг передбачає змінну товщину шарів залежно від локальних геометричних особливостей деталі. Зокрема, для підвищення точності апроксимації контуру моделі в зонах із високою кривизною або деталізацією доцільно застосовувати меншу товщину шарів. Водночас, під час формування геометрично однорідних ділянок або елементів із менш жорсткими вимогами до точності (наприклад, опорних структур) товщину шарів можна збільшувати, що дозволяє скоротити тривалість процесу друку без суттєвої втрати якості.

З врахуванням обох зазначених підходів, задачу слайсингу можна формалізувати таким чином:

Вхідні дані: тривимірна модель та функція $h(z)$*, яка визначає висоту розташування слайсу з номером* z*. У випадку рівномірного слайсингу функція*

$h(z)$ *має вигляд* $h(z) = h_0 + z\delta$, *де* h_0 *– висота першого слайсу, а* δ *– відстань між сусідніми слайсами.*

Вихідні дані: впорядкований набір многокутників, отриманих як перетини моделі з площинами, перпендикулярними до осі z, *розташованими на висотах, визначених функцією* $h(z)$.

Асимптотично оптимальний алгоритм слайсингу запропоновано в роботі (Minetto et al., 2017). Авторами доведено, що в найгіршому випадку обчислювальна складність запропонованого підходу становить $O(n \log k + k + m)$, де n – загальна кількість трикутників меша, k – кількість слайсів, а m – кількість перетинів трикутників мешу з площинами слайсинга. Крім того, в роботі Minetto et al. запропоновано модифікації базового алгоритму для спеціальних випадків: коли трикутники моделі попередньо впорядковані за зростанням їх мінімальної z – координати, а також для випадку рівномірного слайсингу. Виконання хоча б однієї з цих умов дозволяє зменшити асимптотичну оцінку складності алгоритму до $O(n + k + m)$.

Запропонована процедура слайсингу складається з наступних етапів:

- Перетин площин з трикутниками моделі. На цьому кроці алгоритм визначає всі окремі відрізки, що утворюються внаслідок перетину кожної площини слайсингу з трикутниками меша.
- Відновлення замкнених контурів на основі отриманих відрізків На цьому кроці окремі невпорядковані відрізки перетинів об’єднуються в прості многокутники.
- Визначення орієнтації контурів. На останньому кроці для набору простих многокутників визначається коректна орієнтація для зовнішніх та внутрішніх контурів слайсу.

Для ефективного визначення всіх перетинів площин із трикутниками моделі автори застосовують техніку замітаючої площини. На підготовчому етапі

формується список L, де L_i – множина трикутників мешу, мінімальна z-координата яких розташована між i-ю та $(i+1)$-ю площинами слайсингу.

В загальному випадку, побудова списку L здійснюється шляхом визначення для кожного трикутника відповідного індексу за допомогою бінарного пошуку серед площин слайсингу, за рахунок чого обчислювальна складність алгоритму складає $O(n \log k + k)$. Але, якщо трикутники є впорядкованими за зростанням мінімальної z-координати, список L можна сформувати за допомогою техніки двох вказівників, що дозволяє покращити час роботи алгоритму до лінійного $O(n + k)$.

Аналогічно, лінійну складність можна отримати у випадку рівномірного слайсингу. Індекс i відповідного списку для кожного трикутника визначається безпосередньо як ціла частина від ділення його мінімальної z-координати на товщину шару (можливо, з попереднім зсувом всієї моделі таким чином, щоб мінімальна z-координа серед всіх трикутників дорівнювала 0).

Після побудови списку L, алгоритм послідовно обробляє площини, починаючи з найнижчої. В процесі обробки підтримується множина A «активних» трикутників (тих, що перетинаються з поточною площиною). Використовуючи список L, множина A оновлюється на кожному кроці, після чого, для всіх трикутників з A обчислюються відрізки перетину із поточною площиною слайсингу. Результатом такого обходу є невпорядкований набір неорієнтованих відрізків, які утворюються перетинами трикутників мешу з площинами слайсингу.

Наступним кроком є відновлення замкнених контурів за набором невпорядкованих відрізків. Для розв’язання цієї задачі автори використовують хеш-таблицю, у якій ключем є точка на площині слайсу, а значенням – пара точок, з’єднаних із цією точкою відрізками. Після додавання всіх відрізків до хеш-таблиці, алгоритм обирає довільний ключ, вилучає його та переходить вздовж одного з інцидентних ребер до наступної точки. Таким чином формується

перший відрізок контуру. Далі процедура послідовного переходить від вершини до вершини, поки не повернеться до початкової точки. Якщо після побудови одного контуру хеш-таблиці є непорожньою, описаний процес повторюється.

На завершальному етапі визначається орієнтація отриманих контурів. Для цього пропонується підхід на основі перевірки належності точки многокутнику (Volpato et al., 2013), що дозволяє розрізнити зовнішні межі та внутрішні отвори многокутника, утвореного визначеними контурами.

2.5 Побудова траєкторії друку (Задача 5)

Наступним етапом процесу планування АВ після слайсингу є генерування траєкторії руху інструменту для заповнення внутрішньої області кожного контуру моделі. Вимоги до характеристик такої траєкторії суттєво залежать від специфіки обраної технології 3D-друку. Наприклад, технології SLS або ME часто допускають розриви траєкторії та вільні переміщення інструменту (без нанесення матеріалу). Для технологій АВ на основі лазерного спікання існують принтери, що оснащені кількома лазерами, для яких наявність розривів в траєкторії спрощує синхронізацію паралельної роботи різних лазерів. В той же час, існують технології (зокрема WAAM), які вимагають максимально неперервну траєкторію друку для забезпечення високої якості поверхні. В таких методах часті зупинки та запалювання дуги призводять до нерівномірного формування геометрії, що накопичує похибки у вертикальному напрямку під час нанесення наступних шарів.

Загальну постановку задачі генерування траєкторії друку можна сформулювати так:

Вхідні дані: набір многокутників, що описує слайс моделі.

Вихідні дані: траєкторія руху інструмента, що забезпечує повне заповнення внутрішніх областей многокутників.

Більшість ефективних алгоритмів розв'язання задачі побудови траєкторії базуються на стратегії декомпозиції складної геометрії слайсу на простіші області.

Підхід, запропонований в (Dwivedi & Kovacevic, 2004), виконує розбиття контуру слайсу на множину y-монотонних многокутників. Процедура побудови траєкторії починається зі зміщення вхідного многокутника всередину на відстань, що дорівнює роздільній точності друку. Після цього визначаються увігнуті вершини, які розділяються на групи висхідних та низхідних вершин. Далі, від кожної з цих вершин проводяться горизонтальні лінії до перетину з правою границею многокутника, утворюючи монотонне розбиття. Для кожного з отриманих монотонних многокутників формується зигзагоподібна траєкторія, яка будується шляхом послідовного з'єднання точок перетину з рівновіддаленими горизонтальними прямими. На завершальному етапі визначаються точки перетину траєкторій окремих многокутників розбиття, їхні подовження відтинаються, та формується єдиний замкнений шлях обходу внутрішньої області слайсу.

Авторами встановлено, що верхня оцінка складності кроку декомпозиції многокутника складає $O(n \log n)$, де n – кількість вершин в контурі слайсу. Обчислювальна складність побудови траєкторії складає $O(m^2)$, де m – розмір найбільшої з окремих траєкторій для монотонних многокутників. Замкненість побудованої траєкторії є важливою характеристикою розглянутого алгоритму, що робить його застосовним для WAAM та подібних технологіях.

Зигзагоподібний шаблон є ефективним для заповнення великих внутрішніх ділянок слайсу, але його суттєвим недоліком є низька точність відтворення контурів. Причиною цього є похибки дискретизації на границях контуру за рахунок відступу. Для подолання цієї проблеми в роботі (Ding et al., 2014) запропоновано альтернативну, гібридну стратегію заповнення, що комбінує контурний шаблон для зовнішніх меж та зигзагоподібний шаблон для внутрішньої області слайсу.

На першому кроці алгоритму виконується декомпозиція многокутника на опуклі. Спершу визначаються вершини «вирізи», внутрішній кут яких перевищує 180°, наявність яких означає, що многокутник не є опуклим. Для усунення вирізів, алгоритм розглядає їх в порядку зменшення внутрішнього кута, визначає точки перетину променів, спрямованих з вершини-вирізу у внутрішню частину многокутника в напрямку, заданому ребрами, що є інцидентними поточній вершині. В залежності від властивостей ламаної, обмеженої отриманими точками перетину, застосовується одна з трьох процедур підрозбиття: виріз-бісектор, виріз-виріз або виріз-вершина. Цей процес виконується рекурсивно, поки кількість вирізів не дорівнюватиме нулю.

Далі обирається оптимальний напрямок зигзагоподібного заповнення для кожного з отриманих опуклих многокутників, який мінімізує кількість поворотів інструменту. Авторами доведено, що оптимальний напрямок завжди паралельний одному з ребер багатокутника.

Для побудови траєкторії в кожному опуклому многокутнику визначається лівий ланцюг (ланцюг між найвищою на найнижчою точками опуклої оболонки при обході проти годинникової стрілки). Траєкторією заповнення кожного з многокутників є зміщений всередину лівий ланцюг та зигзагоподібна траєкторія для решти многокутника. Зрештою алгоритм об’єднує окремі траєкторії для опуклих многокутників в єдину траєкторію за допомогою ліній, за якими відбувалась його декомпозиція на першому кроці алгоритму.

Інший підхід до вирішення проблеми генерування траєкторії друку, зокрема для великогабаритних деталей та спільної роботи кількох інструментів, описано в роботі (Zhao et al., 2024).

Автори пропонують метод розбиття слайсу, що базується на використанні так званих дрібних примітивів. Отримані примітиви організовуються в набори на основі їх суміжності та мостових з’єднань. Далі набори структуруються в примітиви першого рівня та другого рівня. На їх основі виділяються монотонні

відносно осі y структурні примітиви, для кожного з яких обчислюється мінімальний обмежувальний прямокутник, в межах якого формується траєкторія заповнення. Подібно до гібридного методу Ding et al., зовнішній контур траєкторії будується на основі зсунутого контуру примітиву, а внутрішня область заповнюється зигзагоподібним шаблоном.

Також, авторами описано підхід до інтеграції розробленого алгоритму з системами планування послідовності друку кількома інструментами. Використовуючи пошуковий алгоритм на основі kD-дерева, запропонований підхід мінімізує непродуктивні шляхи переміщення інструментів та уникає зіткнень між ними в процесі паралельної роботи. За даними авторів, цей метод розбиття демонструє точність 99.5% та сприяє усуненню поверхневих дефектів під час використання технологій АВ класу прямого підведення енергії.

Висновки до розділу 2

В цьому розділі було наведено формалізоване визначення та здійснено аналіз ключових задач процесу планування АВ, які є актуальними для більшості існуючих технологій 3D-друку. Показано, що алгоритмічний процес підготовки моделей до друку вимагає послідовного розв’язання низки складних геометричних та оптимізаційних задач. Проведений огляд існуючих підходів до розв’язання зазначеного комплексу задач процесу планування АВ засвідчує високу варіативність алгоритмічних підходів, що застосовуються на кожному з цих етапів, що є прямим наслідком відмінностей у фізичних процесах, покладених в основу різних методів 3D-друку.

З огляду на технологічну різноманітність та специфіку окремо взятих виробничих процесів, на сьогодні не існує універсального програмного рішення, яке б ефективно розв’язувало весь комплекс задач процесу планування АВ. Розглянуті в підрозділі 1.2 існуючі програмні комплекси розробляються з використанням пакетно-алгоритмічного підходу, що передбачає використання набору окремих незалежних бібліотек та алгоритмів. Такі бібліотеки часто

оперують власними структурами даних, методами попередньої обробки та форматами представлення вхідних та вихідних даних, що ускладнює їх ефективну інтеграцію в єдину систему. Як наслідок, в процесі роботи подібні мультиалгоритмічні програмні системи постійно конвертують дані з одного формату в інший та виконують надлишкові обчислення, що робить традиційний пакетно-алгоритмічний підхід неефективним для розробки складних програмних комплексів, зокрема для підтримки процесу планування адитивного виробництва.

Враховуючи велику кількість геометричних задач, що постають в процесі планування АВ, фундаментом розв’язання яких є алгоритми та задачі обчислювальної геометрії та комп’ютерної графіки, найбільш доцільним та перспективним шляхом подолання зазначених вище недоліків є застосування моделі єдиного алгоритмічного середовища, оскільки використання МЄАС вже дозволило отримати значний приріст ефективності під час розв’язання взаємопов’язаних задач обчислювальної геометрії (Терещенко, 2011). Однак, безпосереднє застосування МЄАС на основі теорії звідності в якості архітектурного ядра в системі підтримки процесу планування АВ не дає бажаного ефекту, оскільки розглянуті в цьому розділі задачі, алгоритми їх розв’язання та структури даних, на яких вони базуються не є взаємопов’язаними на формальному рівні. Наслідком цього є необхідність розширення класичної концепції МЄАС з врахуванням специфічних алгоритмічних та технологічних взаємозв’язків між задачами предметної області АВ, в якій інтеграція задач виконується шляхом застосування спільних структур даних та повторному використанні результатів попередніх обчислень на наступних етапах планування АВ. Відповідно, має місце наступне твердження:

Твердження 2.1. Архітектуру системи комп'ютерного моделювання та візуалізації, яка б забезпечувала ефективне розв’язання комплексу задач АВ (задачі 1-5) в межах єдиної алгоритмічної платформи, з наданням прямого доступу до спільних структур даних та набору алгоритмічних інструментів, та

мала властивості гнучкості, мінімізації надлишкових обчислень, узгодженості, відкритості до масштабування та могла працювати з задачами довільної розмірності можна побудувати на основі Моделі Єдиного Алгоритмічного Середовища.

РОЗДІЛ 3. ПРОЕКТУВАННЯ АРХІТЕКТУРИ МОДЕЛІ ЄДИНОГО АЛГОРИТМІЧНОГО СЕРЕДОВИЩА

В цьому розділі розглядається проєктування архітектури моделі єдиного алгоритмічного середовища (МЄАС), орієнтованої на підтримку комплексного процесу планування адитивного виробництва. Метою розділу є формалізація архітектурних принципів, структурних компонентів та механізмів взаємодії алгоритмів та структур даних, які дозволяють поєднати відкритість, модульність і обчислювальну ефективність у межах єдиного програмного середовища. Особливу увагу приділено специфіці використання МЄАС для комплексу задач планування адитивного виробництва, та їх відмінностям від попередніх сфер застосування МЄАС: можливість модифікації геометричних об’єктів, робота з даними різної розмірності, необхідність підтримки складних робочих процесів, що налаштовуються користувачем програмного комплексу.

В підрозділі 3.1 проаналізовано вимоги та критерії до архітектури МЄАС, з врахуванням контексту АВ, необхідності масштабування за кількістю задач, алгоритмів та обсягом даних.

В підрозділі 3.2 введено ключові компоненти МЄАС та формалізовану декларативну структуру опису алгоритму в межах МЄАС з точки зору їх впливу на стан геометричних даних, що в подальшому буде використано для побудови ефективного процесу координації різних алгоритмів та підтримки валідності структур даних.

Підрозділ 3.3 присвячено розгляду менеджера структур даних (МСД) як ключового інфраструктурного компоненту МЄАС, що відповідає за підтримку узгодженості спільних структур даних, керує їх валідністю та реалізує механізм відкладеного оновлення. Показано, як використання класифікації модифікацій з підрозділу 3.2.1 дозволяє мінімізувати надлишкові обчислення та ізолювати розробників алгоритмів від необхідності детально розуміти внутрішню будову інших алгоритмів МЄАС.

Підрозділ 3.4 розкриває концепцію неявного зведення даних як узагальнення класичної ідеї зведення задач в МЄАС. Запропоновано розглядати зведення не лише на рівні задач, але й на рівні репрезентації даних, що дає змогу автоматично визначати найефективніший спосіб побудови відсутніх структур даних.

Нарешті, у підрозділі 3.5 описано графово-вузлову архітектуру як декларативний інтерфейс МЄАС, що надає користувачу можливість візуальної побудови, аналізу та оптимізації робочих процесів. Показано, як поєднання графово-вузлової моделі з вузлами керування, МСД та принципу неявного зведення даних дозволяє моделювати складні, в тому числі ітеративні сценарії планування АВ.

3.1 Аналіз критеріїв та вимог до МЄАС

Враховуючи різноманітність наведеного вище комплексу задач, що виникають на етапі планування адитивного виробництва, МЄАС, що забезпечує ефективне їх розв’язання бути гнучкою, легко масштабованою, простою у використанні. Також, враховуючи стрімку динаміку розвитку адитивного виробництва, важливо забезпечити відкритість МЄАС до розширення: як можливість додавання нових задач, так і нових алгоритмів, що вирішують розглянуті в цій роботі задачі. Таким чином, можна сформулювати наступний перелік вимог до МЄАС:

1. МЄАС має забезпечувати комплексний процес планування АВ.

   Іншими словами, побудована МЄАС повинна щонайменше забезпечувати ефективне розв’язання розглянутих вище задач, що виникають в процесі планування АВ.
2. МЄАС має масштабуватись за кількістю задач, алгоритмів, та об’ємом вхідних даних.

   Мається на увазі, що архітектура побудованої моделі має дозволяти користувачу додавати нові задачі, що є специфічними для конкретної

технології або робочого процесу; додавати альтернативні алгоритми розв’язання розглянутих вище задач, причому робити це ефективно для великого об’єму вхідних даних.

3. Архітектура МЄАС має базуватись на універсальних структурах даних, що є спільними для реалізованих алгоритмів.

   Структури даних, що використовуються в МЄАС мають бути спільними для всіх алгоритмів, що їх використовують.
4. Архітектура МЄАС має забезпечувати сумісність вхідних та вихідних даних між різними алгоритмами. Алгоритми мають використовувати уніфіковані формати репрезентації вхідних та вихідних даних.
5. МЄАС має уникати надлишкових обчислень та виконувати їх лише в разі необхідності.

   Структура даних, використовується в процесі роботи кожного з алгоритмів має будуватись лише один раз для одного набору вхідних даних.
6. МЄАС має підтримувати роботу «в режимі налагодження», даючи можливість зовнішньому оператору налаштувати оптимальні параметри алгоритмів та провести верифікацію отриманих результатів.

Остання вимога є важливою саме в контексті АВ, оскільки попередні дослідження МЄАС, наприклад, для комплексу задач обчислювальної геометрії (Терещенко, 2010) розглядають детерміновані задачі, тобто такі, для кожного набору вхідних даних яких існує однозначно визначений результат. В той же час більшість задач, що розглядаються в цій роботі є оптимізаційними, тобто серед множини можливих розв’язків необхідно знайти найкращий згідно з певним критерієм. Алгоритми розв’язання таких задач часто мають велику кількість вхідних параметрів, що дозволяють керувати балансом між якістю знайденого результату та затратами часу та пам’яті, які необхідні алгоритму на його побудову. Також, часто застосовуються евристичні алгоритми, що знаходять певну апроксимацію оптимального розв’язку з деякою заданою точністю. МЄАС має забезпечити можливість визначення оптимальної конфігурації таких

алгоритмів зовнішнім оператором (яким може виступати як людина, так і алгоритм машинного навчання).

3.2 Структура МЄАС

З наведеного вище огляду комплексу задач АВ, а також сформульованих критеріїв побудови МЄАС (розділи 2 та 3.1) стає очевидним, що структура МЄАС з робіт (Терещенко, 2011) та (Коцур, 2019) лише частково покриває потреби проблемної області АВ, що досліджується в цій роботі:

- По-перше, існуючі розробки МЄАС для роботи з геометричними алгоритмами, як то задачі обчислювальної геометрії або обробки зображень, розраховані на роботу з даними однієї розмірності (2D або 3D). В той же час, підтримка комплексного процесу планування АВ вимагає ефективної роботи як з двовимірними даними (плоскі многокутники, розбиття площини тощо), так і з тривимірними об'єктами (меші трикутників, що задають геометрію моделей).
- По-друге, в згаданих вище роботах геометричні об'єкти залишаються незмінними в процесі роботи МЄАС. Тобто результатом роботи алгоритмів МЄАС є або певна інформація, визначена на основі вхідних даних (наприклад, визначення позиції певного об'єкта на зображенні), або ж утворення певної надбудови над вхідними даними (наприклад, побудова тріангуляції або евклідового мінімального кістякового дерева), але в обох випадках вхідні дані (набір точок або зображення) залишаються незмінними. Натомість, в цій роботі розглядаються задачі, зокрема булеві операції над тривимірними моделями, генерування мешів та побудова опорних конструкцій, які передбачають модифікацію вхідних даних моделі.
- Підтримка «режиму налагодження» є новою концепцією в дослідженнях, пов'язаних з МЄАС та має бути врахованою на етапі проектування архітектури програмного середовища.

В той же час, загальна концепція МЄАС (*спільна попередня обробка – спільні структури даних - алгоритми*), висвітлена в (Терещенко, 2011) залишається підґрунтям для побудови ефективного засобу розв'язання комплексу задач АВ.

Першим етапом роботи МЄАС є попередня обробка, що полягає в читанні вхідних даних, їх верифікації та приведенні до спільного представлення. В цій роботі вхідними даними є одна або більше тривимірна модель, якими задано межу об'єктів, які необхідно надрукувати. В якості єдиного формату репрезентації тривимірних геометричних об'єктів для алгоритмів МЄАС будемо використовувати меш трикутників, що де-факто є стандартом в індустрії АВ. Для файлового представлення таких даних прийнято застосовувати формат *STL* (Library of Congress, 2025), проте в якості вхідних даних можуть виступати й інші файлові формати.

Також, можливими вхідними даними є моделі, побудовані в CAD системах (*від англ. Computer-Aided Design, комп'ютерне проектування*), представлені у вигляді *граничного подання* (*B-Rep*), що зазвичай складається з кривих Безьє, NURBS, та подібних поверхонь. Такі моделі можуть бути збережені в файлових форматах STEP, IGES та ін. Для підтримки файлових B-Rep форматів в МЄАС необхідно виконати попередню обробку – конвертацію з B-Rep формату в меш трикутників. Детальніше цей процес було розглянуто в розділі 2 цієї дисертації. Останнім кроком попередньої обробки є виправлення дефектів мешу. Оскільки вхідними даними МЄАС можуть виступати довільні меші трикутників, отримані безпосередньо від користувача, або як результат конвертування з іншого представлення, важливо на етапі попередньої обробки позбутись від можливих помилок, які можуть мати місце в меші: дірки (пропущені трикутники, які порушують замкненість моделі), інвертовані нормалі, самоперетини трикутників і так далі.

Після виконання попередньої обробки виконується заданий користувачем робочий процес обробки моделей. Цей процес полягає в почерговому

застосуванні обраних алгоритмів до оброблених вхідних даних, з можливістю призупинити цей процес для оцінки якості результату. За необхідності, допустимі внесення корегувань до майбутніх кроків процесу або повернення назад. Застосування низки алгоритмів потребує використання однієї або більше спільних структур даних. Для забезпечення ефективності цього процесу та уникнення надлишкових обчислень в цій роботі пропонується використати *менеджер структур даних* (*МСД*), який виступатиме ключовим структурним елементом архітектури МЄАС. Деталі реалізації МСД будуть розглянуті в подальших розділах цієї роботи.

3.2.1 Класифікація алгоритмів

Введемо наступну класифікацію для алгоритмів, що реалізовані в МЄАС:

- За розмірністю простору: оскільки МЄАС для розв'язання комплексу задач АВ потребує роботи як з двовимірними так і тривимірними даними, природньо розділити алгоритми на класи за розмірністю простору їх вхідних даних.
- За типом обробки вхідних даних алгоритми будемо класифікувати як *модифікаційні* – такі, що змінюють дані, над якими вони працюють, та *немодифікаційні*, застосування яких залишає вхідні дані без змін. Наприклад, процедура пошуку найближчого сусіда не змінює дані, на яких вона працює, в той час як алгоритм заокруглення гострих кутів меша, очевидно, вносить зміни до відповідних ділянок тріангуляції.

В класі модифікаційних алгоритмів окремо виділимо *класи ознак* за типом змін, що вносяться алгоритмами, з врахуванням різниці між геометрією, топологією, атрибутами та положенням.

До класу *атрибутивних* модифікаційних алгоритмів будемо відносити алгоритми, що змінюють метаінформацію, асоційовану з геометричними об'єктами, не торкаючись при цьому їхньої форми, топології та інших властивостей. Прикладом таких змін є нанесення текстур, атрибутів видимості,

розфарбовування вершин тощо. Алгоритми цього класу не порушують коректність геометрії та можуть застосовуватись без ризику накопичення похибок.

Топологічно-тріангуляційні алгоритми зберігають границю об’єкта як множину точок простору, але змінюють спосіб його дискретного представлення, перебудовуючи вершини, трикутники, ребра та топологію без зміни геометрії об’єкта. Прикладом такого алгоритму є повторна тріангуляція плоских граней тривимірної моделі з метою покращення їх якості (наприклад, побудова тріангуляції Делоне). Зазвичай, такі алгоритми застосовуються для оптимізації продуктивності системи та зменшення похибки чисельних обчислень.

Клас *геометричних* модифікаційних алгоритмів містить алгоритми, що змінюють певні геометричні властивості об’єктів, над якими вони працюють (або їх складових). До таких властивостей відносяться форма об’єкта, зміна кривизни (наприклад, $\sqrt{3}$-підрозбиття (Kobbelt, 2000)), додавання або вилучення складових частин об’єкта, деформація тощо.

Просторово-трансформаційні алгоритми змінюють виключно глобальне положення, орієнтацію та масштаб об’єкта в просторі, але залишають сталою його геометрію, топологію та атрибути. Прикладом таких алгоритмів є позиціонування та зіставлення геометричних тривимірних моделей.

До найзагальнішого класу *комплексних* (або *радикальних*) алгоритмів будемо відносити алгоритми, що можуть одночасно змінювати геометрію, топологію, глобальне положення та атрибути геометричного об’єкта. Алгоритми цього класу фактично можуть переконструювати модель, вирішуючи задачу, що не зводиться до локальних змін геометрії або просторового розташування. Прикладом таких алгоритмів є булеві операції над тривимірними моделями, реконструкція поверхні за хмарою точок. Зокрема, до цього класу алгоритмів будемо відносити й читання геометричних об’єктів з файлів на диску.

Важливо зауважити, що в межах наведеної вище класифікації свідомо вживається термін «алгоритм», а не «задача», попри те, що геометричні операції природно описуються саме як задачі. Однак, формулювання через задачу було б корисним лише за умови, що всі алгоритми, що розв'язують певну задачу, здійснюють зміни одного й того самого типу. На практиці ситуація часто є іншою: різні алгоритми розв'язання однієї задачі можуть належати до різних класів за характером внесених змін. Наприклад, задачу згладжування моделі можна розв'язувати як вносячи лише локальні геометричні зміни, зберігаючи при цьому топологію, так і більш агресивними підходами, що змінюють геометричну та топологічну структури об'єкта. Вживання терміну «алгоритм» функціонує як скорочення виразу «конкретні алгоритми, що реалізують розв'язання відповідної задачі», а класифікація спрямована не на задачі як абстрактні математичні формулювання, а саме на методи їх розв'язання та на те, які зміни ці методи фактично вносять в структуру даних.

3.3 Менеджер структур даних

Розглянемо детальніше концепцію *менеджера структур даних* (*МСД*), введену в розділі 3.2. Як зазначалося вище, модель єдиного алгоритмічного середовища (МЄАС), орієнтована на комплексний процес планування адитивного виробництва, повинна забезпечувати ефективну співпрацю широкого спектра різнотипних алгоритмів, що послідовно застосовуються до множини вхідних даних. Ці алгоритми можуть працювати з об'єктами різної розмірності та покладаються на різні структури даних, що відображають спосіб подання моделі на певному етапі обробки.

Оскільки модифікаційні алгоритми змінюють стан даних (в межах відповідних класів змін, описаних у попередньому розділі), структури даних, побудовані на попередніх етапах, можуть втрачати актуальність і вимагати часткового або повного оновлення. Унаслідок цього ефективність системи

істотно залежить від здатності МЄАС підтримувати узгодженість спільних структур даних під час послідовного застосування різних алгоритмів.

МЄАС, що розробляється в цій роботі не обмежується фіксованим сценарієм планування АВ, а натомість має на меті розробити універсальний інструмент, який може бути адаптованим до будь-якого робочого процесу. Для цього МЄАС має включати механізм, що, з одного боку, надає алгоритмам можливість використовувати спільні структури даних без надмірних обчислень і повторної попередньої обробки, а з іншого - гарантує узгодженість і валідність цих структур після будь-яких модифікацій, незалежно від того, які саме алгоритми (та в якому порядку) використовуються.

Таким чином, розробник окремого алгоритму звільняється від необхідності перевіряти коректність структури даних, яку він використовує: ця відповідальність покладається на МЄАС. Такий підхід суттєво підвищує модульність і відкритість системи до зовнішнього розширення, спрощуючи інтеграцію нових задач та алгоритмів, які є специфічними для робочого процесу або технології АВ конкретного користувача, та не розглядаються в цій роботі.

Функціонально, МСД виконує роль проміжного шару в архітектурі МЄАС, який:

1. Підтримує узгодженість множини взаємопов’язаних структур даних, побудованих для різних геометричних об’єктів.
2. Відстежує залежності між цими структурами, та визначає, які з них залишаються коректними після застосування певного алгоритму, а які потребують часткового або повного оновлення.
3. Забезпечує автоматичну побудову або інкрементальне оновлення структур даних в момент їх запиту алгоритмом.
4. Виступає єдиним джерелом істини для всіх алгоритмів в межах МЄАС, виключаючи дублювання, неузгодженість та надлишкові обчислення, що повністю відповідає SSOT архітектурі (Yllemo, 2023).

5. Ізолює розробника алгоритму від необхідності контролювати стан структур даних, оскільки відповідальність за їх коректність покладається на МСД.

Робота МСД ґрунтується на введеній вище класифікації алгоритмів (модифікаційні та немодифікаційні), та на класах можливих змін (атрибутивні, топологічно-тріангуляційні, геометричні, просторово-трансформаційні, комплексні). Віднесення алгоритму до одного або кількох класів змін дозволяє МСД визначити характер впливу алгоритму на стан даних. Ця інформація використовується МСД для прогнозування наслідків застосування алгоритму та керування валідністю наявних структур даних.

Для кожної структури даних S, реалізованої в МЄАС (наприклад, kD-дерево, AABB-ієрархія, граф Ріба, маска атрибутів, тощо), МСД зберігає *профіль чутливості* до різних класів змін. Такий профіль визначає, як структура даних має реагувати на модифікації певного типу. Зокрема:

- Атрибутивні модифікації часто не впливають на геометричні структури даних, але впливають на атрибутивні буфери.
- Геометричні модифікації вимагають повного або часткового оновлення деревоподібних структур даних.
- Топологічно-тріангуляційні модифікації зазвичай спричиняють повну перебудову індексних структур даних.
- Просторово-трансформаційні модифікації можуть вимагати як застосування трансформації до всіх вузлів деревовидних структур даних, або можуть оброблятись шляхом підтримки додаткової матриці трансформації, що акумулює всі модифікації цього типу.
- Комплексні модифікації приводять до повної інвалідності всіх пов’язаних структур даних.

Формально, з кожною структурою даних S асоціюється функція R_S:

$$R_S: C \to \{NoAction, LocalUpdate, Rebuild\}, \text{ де:}$$

$C -$ множина класів змін, $R_S(\mathrm{c}) -$ реакція структури S на модифікацію класу c.

В процесі роботи, або в момент завершення роботи, алгоритм A сигналізує МСД про класи виконаних ним змін. МСД обробляє цей сигнал наступним чином:

1) Аналізується типи здійснених модифікацій
2) Для кожної побудованої структури даних S, що є асоційованою з даними, які були змінені алгоритмом A застосовується відповідна реакція R_S. При цьому, структура даних S маркується в один з наступних станів:
   a) *Валідний* стан, якщо внесені зміни не зачіпають S ($R_S(c) = NoAction$), та до виконання алгоритму A структура даних S знаходилась в валідному стані.
   b) *Частково невалідний* стан, якщо має місце один з наступних випадків:
      i) Можливе інкрементальне оновлення S ($R_S(c) = LocalUpdate$), та до виконання алгоритму A структура даних S знаходилась в валідному стані.
      ii) $R_S(c) \neq Rebuild$, та до виконання алгоритму A структура даних S знаходилась в частково невалідному стані.
   c) *Повністю невалідний* стан, якщо S потрібно повністю перебудувати ($R_S(c) = Rebuild$), або якщо до виконання алгоритму A структура даних S знаходилась в повністю невалідному стані.

У випадку часткової невалідності та $R_S(c) = LocalUpdate$ також фіксується опис змін (наприклад, список вершин та зміни їх координат). Пізніше, збережена інформація буде використана для виконання локального оновлення.

МСД працює за принципом *відкладеного оновлення*. Це означає, що оновлення виконується не одразу в момент отримання сигналу про внесені зміни та необхідності застосування відповідної реакції. Натомість, структура

залишається в маркованому стані до моменту, коли інший алгоритм звертається до МСД за доступом до неї. В момент такого звернення МСД виконує часткове або повне оновлення структури даних (відповідно до її маркування). Такий підхід забезпечує наступні переваги:

- Відсутність непотрібних обчислень, адже деякі структури даних можуть так і не знадобитись на подальших кроках роботи системи (оскільки декларація алгоритмом залежності від структури даних не гарантує, що вона знадобиться на конкретному наборі вхідних даних).
- Мінімізація перерахунків у випадку серії модифікацій, що можуть зробити надлишковими попередні оновлення (наприклад, кілька змін координат однієї вершини може оброблятись як одна модифікація).

Формально, обробка звернення певного алгоритму до МСД за структурою даних S описується наступним відображенням:

$$\mathrm{Get}(S) = \begin{cases} S, & \text{якщо стан } S - \text{валідний} \\ LocalUpdate(S, D), & \text{якщо стан } S - \text{частково невалідний} \\ Rebuild(S), & \text{якщо стан } S - \text{повністю невалідний} \end{cases}, \text{ де}$$

$D = D(S)$ −журнал необроблених змін, що були застосовані до даних, асоційованих зі структурою даних S.

Таким чином, робота МСД базується на трьох ключових ідеях:

1) *Типи модифікацій → типи реакцій*: класифікація модифікаційних алгоритмів використовується для визначення впливу виконання алгоритму на різні структури даних.
2) *Маркування стану валідності структури даних*: після кожної модифікації даних МСД визначає мінімальну необхідну ступінь оновлення кожної структури даних.
3) *Відкладене оновлення*: фактична синхронізація даних виконується не в момент зміни їх стану, а в момент фактичного використання структури даних.

Як результат, МСД дозволяє оптимізувати сумарну обчислювальну вартість виконання композиції алгоритмів та уникнути дублювання обчислень, забезпечує незалежність алгоритмів та підтримує модульність та розширюваність МЄАС.

3.4 Неявне зведення даних

Однією з ключових ідей моделі єдиного алгоритмічного середовища є переосмислення задач як взаємопов'язаних операцій, де результат однієї задачі використовується для прискорення вирішення інших, пов'язаних задач. Класичним прикладом застосування такого підходу є МЄАС для деяких задач обчислювальної геометрії на площині (Терещенко, 2011), що використовує теорію звідності задач в якості своєї основи. Звідність в контексті МЄАС означає можливість отримати необхідні дані на основі розв'язків задач, які було отримано раніше. Така операція часто є «дешевшою» з точки зору асимптотичної оцінки складності, в порівнянні з побудовою необхідного розв'язку з нуля. Наприклад, в зазначеній вище роботі діаграма Вороного розглядалась як універсальна структура даних, побудова якої дозволяє знайти розв'язок пов'язаних задач (наприклад, побудова опуклої оболонки або тріангуляції Делоне) за лінійний час.

Масштабуючи зазначений вище підхід, в цій роботі пропонується розглядати правила зведення не лише на рівні задач, але й на рівні репрезентацій даних (як структур даних так і форматів їх представлення). Ідея полягає в тому, що МЄАС, використовуючи МСД як сховище структур даних, має змогу не лише підтримувати існуючі структури, але й планувати побудову відсутніх структур або репрезентацій даних за допомогою застосування послідовності зведень із вже наявних представлень (у випадку, коли безпосередня побудова «з нуля» є обчислювально складнішою). Надалі цю концепцію будемо *називати неявним зведенням даних*.

Позначимо множину типів репрезентацій як $S = \{S_1, S_2, \dots, S_m\}$, в яку входять як структури даних (наприклад, діаграма Вороного або граф Ріба), так і

формати представлення (наприклад, представлення простого многокутника як набір монотонних многокутників).

Множину правил зведення (редукції) будемо позначати R. Кожне правило $r \in R$ формально задається як трійка $r = (S_s, S_t, A_r)$, де:

- $S_s, S_t \in S$ – початковий та результуючий типи репрезентації
- A_r – алгоритм, що реалізує відповідне перетворення.

Означення 3.1. *Редукційним графом* $G_r = (S, R)$ МЄАС будемо називати зважений орієнтований граф, вершинами якого є елементи множини S, а ребра будуються на основі правил редукції в напрямку від S_t до S_s з вагою $C(A_r)$, де оператор $\mathrm{C}(\cdot)$ позначає оцінку вартості виконання алгоритму A_r.

В залежності від алгоритму, в якості оцінки вартості можна використовувати як асимптотичну оцінку складності, так і очікувану складність в середньому. Наприклад, якщо A_r є алгоритмом Quick Hull побудови опуклої оболонки, то в якості оцінки його складності доцільно використовувати не асимптотичну оцінку $O(n^2)$, а $O(nk)$, де k – очікувана кількість вершин опуклої оболонки (Пришляк & Терещенко, 2024). Аналогічно, для алгоритмів, що використовують хеш-таблиці та подібні структури, доречно використовувати амортизаційну оцінку складності, а не оцінку в найгіршому випадку.

Варто зауважити, що може існувати кілька способів зведення однієї репрезентації в іншу, кожен з яким має місце за різних вхідних даних. Такі випадки враховуються шляхом додавання множинних ребер різної ваги в графі G_r.

Нехай під час виконання деякого робочого процесу вузлу-алгоритму потрібно отримати репрезентацію $S_{req} \in S$. Наявні в МСД репрезентації утворюють підмножину $S' \subseteq S$. Тоді задачу побудови S_{req} можна сформулювати наступним чином: знайти таку послідовність перетворень, що дозволяє отримати S_{req} з множини наявних репрезентацій S' з мінімальними очікуваними ресурсними витратами, або вирішити, що ефективніше буде побудувати S_{req} з нуля.

Сформульована задача є задачею пошуку найменшого шляху в зваженому графі G_r, де вершиною-джерелом є S_{req}, а S' є множиною цільових вершин. Тоді шукана послідовність перетворень задається знайденим шляхом в зворотному порядку вершин. Розв'язувати цю задачу можна класичним алгоритмом пошуку найкоротшого шляху в зваженому графі (Dijkstra, 1959), що шукає найкоротший шлях до всіх вершин графу. Також можна кілька разів виконати алгоритм A* (Hart et al., 1968), що знаходить найкоротший шлях між заданою парою вершин. Такий підхід може бути оптимальнішим в контексті МЄАС, оскільки редукційний граф за своєю природою не містить великої кількості вершин та є достатньо розрідженим. При цьому, не залежно від вибору алгоритму, пошук зупиняється, якщо поточний шлях «важчий» за побудову S_{req} без використання попередніх обчислень.

Розглянемо приклад роботи неявного зведення даних. Нехай деякий алгоритм в процесі своєї роботи використовує Евклідове мінімальне кістякове дерево (ЕМКД) для деякої множини точок на площині. МЄАС реалізує алгоритм побудови ЕМКД, що має складність $O(n \log n)$, де n - кількість точок. Редукційний граф містить наступні правила зведення:

- Тріангуляцію Делоне для множини з n точок на площині можна побудувати за час, що не перевищує $O(n)$ в найгіршому випадку, якщо для цієї множини точок відома діаграма Вороного. (Терещенко, 2011)
- ЕМКД для множини з n точок на площині можна побудувати за час, що не перевищує $O(n)$ в найгіршому випадку, якщо для цієї множини точок відома тріангуляція Делоне використовуючи алгоритм 2 з роботи (Mares, 2004).

Підграф G_r, що відповідає описаним вище правилам зображено на рисунку 3.1.

Нехай МСД містить діаграму Вороного для множини точок, що розглядається. Тоді редукційний граф G_r містить шлях з вершини ЕМКД до діаграми Вороного через тріангуляцію Делоне з загальною оцінкою ваги $O(n)$, що менше за оцінку складності побудови ЕМКД без використання додаткових

даних - $O(n \log n)$. Відповідно, застосувавши принцип неявного зведення даних в МЄАС, шукане Евклідове мінімальне кістякове дерево буде знайдено за час $O(n)$.

Важливо, що розглянутий вище процес виконується «неявно», безпосередньо на рівні МЄАС, тобто не вимагає жодного втручання розробника алгоритму або користувача програмного комплексу.

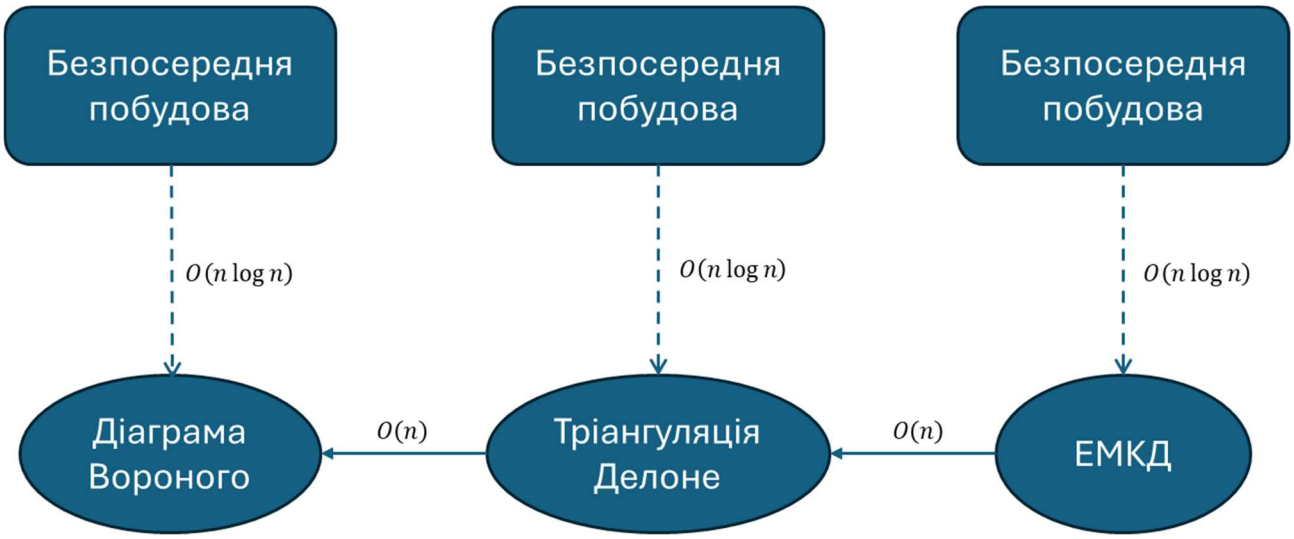

Рисунок 3.4. Підграф G_r

Розширимо наведену вище концепцію до можливості підтримки функцій оцінок вартості, що залежать від довільної кількості параметрів, причому в якості параметрів можуть використовуватись не лише характеристики геометричних об'єктів (наприклад, кількість вершин), а й властивості середовища, в якому оперує МЄАС (наприклад, максимальна кількість паралельних потоків, які може підтримувати процесор). Таке розширення необхідне для можливості практичного застосування неявного зведення даних в комплексній програмній системі, що орієнована на виконання складних геометричних алгоритмів, в тому числі з можливістю паралельного або розподіленого виконання.

В термінах сформульованої вище математичної моделі, введення оцінки вартості операції зведення означає, що $C(A_r)$ розглядається не як деяка аналітична функція, а як програмна функція в розумінні теорії програмування, тобто як обчислювальна процедура, що задається програмним кодом. Така

функція може містити умовні переходи, допоміжні обчислення, виклики інших функцій, що дозволяє реалізувати кусково-задану або контекстно-залежну логіку оцінювання вартості виконання редукції. Разом з тим, час її виконання має бути асимптотично незначним у порівнянні з часом виконання як самого алгоритму A_r, так і решти алгоритмів системи.

Процес пошуку оптимального шляху в редукційному графі відбувається в під час виконання робочого процесу для конкретного набору даних. Це означає, що значення аргументів функції $C(A_r)$ є відомими, а вартість виконання відповідної редукції може бути обчислена шляхом обрахунку фактичного значення $C(A_r)$ в процесі роботи алгоритму пошуку найкоротшого шляху.

Важливо відзначити, що логіка вибору між альтернативними алгоритмами реалізується на рівні структури редукційного графа, в той час як функція $C(A_r)$ відповідає лише за оцінювання вартості виконання редукції за конкретного стану системи. Такий підхід дозволяє будувати функції оцінки вартості редукції не враховуючи наявні альтернативні алгоритми, що робить їх простішими в підтримці, адже додавання нового альтернативного алгоритму не вимагатиме їх оновлення.

З врахуванням функцій оцінки вартості, наведений вище приклад (рисунок 3.1) можна узагальнити у вигляді наступної теореми:

Теорема 3.1. Модель єдиного алгоритмічного середовища для взаємозв'язаних задач обчислювальної геометрії на основі діаграми Вороного (Терещенко, 2011) може бути реалізованою в термінах розширеної МЄАС з цієї дисертаційної роботи.

Доведення. Сформулюємо загальну схему побудови редукційного графу, що описує структуру МЄАС з роботи (Терещенко, 2011). В якості репрезентацій даних використовуються результати розв'язання взаємопов'язаних задач обчислювальної геометрії (опукла оболонка, тріангуляція Делоне, діаграма Вороного, ЕМКД та інші). Окремою репрезентацією є результат виконання попередньої обробки, що обраховується лише один раз та надалі зберігається в МСД у вигляді зваженої зчепленої черги. В якості правил зведення виступають

процедури злиття (рекурсивного підйому), для кожної з яких $C(A_r) = O(\log^2 N)$, згідно з теоремою 3.4 з роботи (Терещенко, 2011). Побудований таким чином редукційний граф при першому виконанні неявного зведення виконає попередню обробку, побудову зваженої зчепленої черги, та одну з процедур злиття. Надалі, обробка кожного наступного звернення полягатиме виключно у виконанні необхідної процедури злиття. Таким чином, побудований редукційний граф цілком реалізує функціонал МЄАС з роботи (Терещенко, 2011). ■

3.5 Графово-вузлова архітектура інтерфейсу МЄАС

Як зазначалось в попередніх розділах, однією з фундаментальних вимог до сучасних систем планування АВ є можливість адаптації та налаштування робочого процесу. З точки зору користувача це означає можливість самостійно визначати послідовність задач (та алгоритмів, що їх розв'язують), які застосовуються до вхідних даних. Такий підхід дозволяє оптимізувати робочий процес під конкретну технологію АВ, матеріал, тип виробу та будь-які інші специфічні вимоги виробництва.

Однак така відкритість має і зворотний бік: користувач може поєднувати задачі неоптимальним чином, або у спосіб, що призводить до логічних або геометричних конфліктів. Як приклад можна розглянути сценарій, за якого після генерування опорних конструкцій користувач виконує геометричну модифікацію моделі. Наприклад, такою модифікацією може бути невеликий зсув для компенсації втрат матеріалу під час постобробки, що може істотно вплинути на механічну стабільність моделі та оптимальність вибору контактних ділянок згенерованих опорних структур. В результаті опори, створені до зміни геометрії можуть стати некоректними та потребувати повторної генерації.

Проблема полягає в тому, що користувач може не усвідомлювати взаємозв'язків між алгоритмами, структурами даних та характером змін, що будуть внесені до структур даних в МСД відповідно до їх профілю чутливості. У відкритих системах неможливо гарантувати, що користувач не створить

неоптимальну або непослідовну конфігурацію робочого процесу, яка призведе до масованих інвалідацій структур даних в МСД.

Як наслідок, виникає потреба у візуалізації та прогнозуванні інвалідацій структур даних, що дозволяє виявити неефективні комбінації задач (та алгоритмів) і спростити оптимізацію робочого процесу. Для побудови такого інструменту в цьому дослідженні пропонується застосувати *графово-вузлову архітектуру* (Hosick, 2014), яка широко застосовується, зокрема, в комп'ютерній графіці та CAD-системах.

Графово-вузлова архітектура (англ. node graph architecture) – це підхід до організації обчислювальних процесів, за якого система моделюється як орієнтований граф, вузли якого відповідають окремим операціям, функціональним компонентам або обчислювальним модулям, а ребра описують потоки даних або керування (сигнали активації, умови виконання тощо) між ними. Виконання обчислювального процесу, заданого у вигляді *графово-вузлової моделі* полягає в почерговому «запуску» вузлів в порядку, заданому графом. Такий підхід дозволяє подати алгоритми у декларативному вигляді, що забезпечує чітку формалізацію залежностей, гнучкість та модульність програмної системи, та можливість паралельного виконання незалежних задач.

Графово-вузлова архітектура часто використовується в графічних редакторах як засіб опису властивостей елементів зображення або сцени, та взаємодії між ними. Зокрема, в тривимірному графічному редакторі з відкритим доступом Blender (Blender Foundation, 2025) використовується графово-вузлове представлення в підсистемах матеріалів та процедурного моделювання. Рисунок 3.2 ілюструє застосування такого підходу до опису текстурованого матеріалу.

В контексті МЄАС та зазначеної вище проблеми відкритих програмних систем графово-вузлова архітектура виконує одразу кілька функцій. По-перше, вона надає користувачеві можливість візуалізувати побудований робочий процес в наочній формі – як граф, де кожен вузол відповідає окремій задачі: позиціонуванню, генерації опор, слайсингу тощо. Така візуалізація дозволяє

побачити залежності між задачами, зрозуміти, які дані використовуються на кожному з етапів роботи, та оцінити, де саме можуть виникати неефективні поєднання задач. По-друге, граф виступає інструментом прогнозування наслідків модифікацій: оскільки кожен вузол, що відповідає певній задачі, декларує класи змін, які він може вносити під час свого виконання, МСД може на основі цієї інформації визначити структури даних, які необхідно буде оновити або перебудувати після виконання конкретної задачі. Як наслідок, коли користувач змінює порядок вузлів або додає новий вузол між існуючими, система негайно відображає, як саме змінюється процес інвалідації структур даних, даючи змогу оцінити вплив внесених змін на обчислювані витрати робочого процесу.

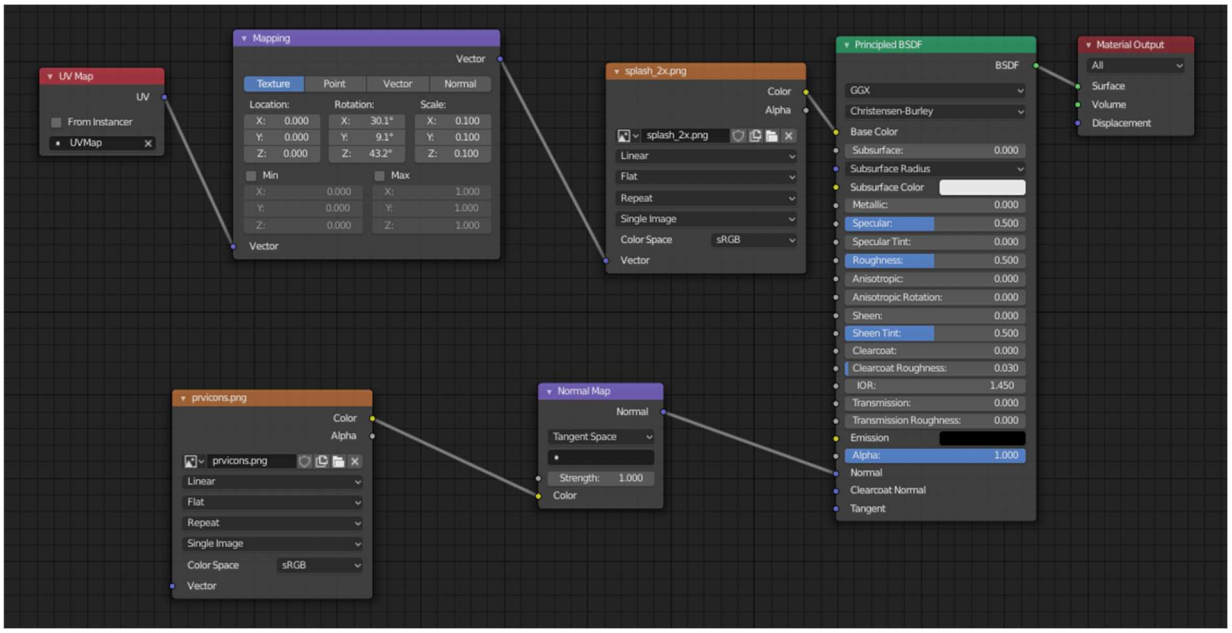

Рисунок 3.2. Графово-вузлове представлення матеріалу (Blender Foundation, n.d.).

Важливо, що використання такої архітектури забезпечує користувача засобом оптимізації робочого процесу, який не вимагає глибокого розуміння внутрішньої реалізації алгоритмів. Повертаючись до прикладу, наведеного на початку цього розділу, якщо користувач переміщає операцію зсуву перед генерацією опорних конструкцій, система автоматично покаже, що тепер немає необхідності відновлювати опори після модифікації геометрії, що, у свою чергу, суттєво знижує обчислювальні витрати. Таким чином, графово-вузлова

архітектура виконує роль не лише інтерфейсу, але й механізму аналітичного супроводу, який дозволяє користувачу будувати ефективні робочі процеси.

Зазначимо, що на рівні графово-вузлової архітектури МСД не розглядається як вузол графа, оскільки він є інфраструктурним компонентом МЄАС, що завжди існує в єдиному екземплярі, не бере безпосередньої участі в потоці керування, визначеному графом, та автоматично реагує на кожне виконання модифікаційного алгоритму, незалежно від структури графа.

Формалізуємо описаний вище підхід. Робочий процес в межах МЄАС будемо представляти у вигляді орієнтованого графу $G = (V, E)$, де V – множина вузлів, та E – множина орієнтованих ребер, що описують залежності між вузлами.

Означення 3.2. *Вузлом-алгоритмом* $v \in V$ будемо називати структурний елемент графа G, що представляє окремий алгоритм обробки геометричних даних.

Кожен вузол-алгоритм визначається як трійка $v = (A_v, I_v, O_v)$, де:

- A_v – алгоритм, що виконується вузлом v
- I_v – множина вхідних даних вузла v, включаючи посилання на необхідні структури даних в МСД
- O_v– множина вихідних даних, що є результатом роботи вузла v

Кожен алгоритм з МЄАС в явному вигляді декларує класи модифікацій, які він *може* здійснити протягом своєї роботи, що дає можливість МСД оцінити ймовірні наслідки виконання відповідного вузла.

Варто звернути увагу, що алгоритми декларують саме можливі класи модифікацій, що дає змогу оцінити ефективність робочого процесу в найгіршому випадку до початку його виконання, використовуючи його опис у вигляді графу G. При цьому, робота МСД здійснюється під час виконання алгоритму, тобто на фактичні інвалідації структур даних впливають лише реально внесені модифікації геометричних об’єктів або їх властивостей.

Часто програмні рішення на базі графово-вузлової архітектури обмежують можливість будувати довільні графово-вузлові моделі, накладаючи вимогу ациклічності на граф, що її представляє. Таке обмеження насправді спрощує математичну модель, усуває проблему нескінченного обходу та дозволяє гарантувати скінченність виконання будь-якої конфігурації вузлів. Проте, в контексті 2розробки практично-орієнтованої МЄАС для розв'язання комплексу задач АВ, таке обмеження унеможливить побудову ітераційних робочих процесів (наприклад, побудова опорних конструкцій – оцінка міцності опор – посилення опорних конструкцій). Як наслідок, в цій роботі також будемо розглядати *вузли керування*, які дозволяють моделювати розгалуження потоку обчислень. При цьому, вихідні ребра такого вузла можуть вести як у невідвідані раніше вузли, так і до вузла підграфа, що вже був відвіданий в процесі виконання робочого процесу.

Вузли керування застосовуються для управління порядком виконання вулзів графово-вузлової моделі. Вони не здійснюють жодних модифікацій геометричних об'єктів або пов'язаних з ними структур даних, але під час обрахунку умови розгалуження вони можуть використовувати структури даних з МСД. Тому, в загальному випадку, використання вузлів керування не є безкоштовним з точки зору обчислювальної складності робочого процесу.

Означення 3.3. *Вузлом керування* арності k будемо називати елемент $v_c \in V$, який визначається як пара $v_c = (P_{v_c}, O_{v_c})$, де:

$P_{v_c} = (p_1, p_2, \dots, p_k)$ – впорядкована послідовність предикатів (немодифікаційних алгоритмів),

$O_{v_c} = (e_1, e_2, \dots, e_{k+1})$ – впорядкована множина виходів.

Вихід $e_i, i = \overline{1,k}$ активується, якщо предикат p_i істинний, а попередні предикати $e_j, j = \overline{1,k-1}$ є хибними. Якщо всі k предикатів є хибними, то активується останній вихід e_{k+1}.

Виконання вузла керування відбувається за схемою, аналогічною конструкції if – else if - … - else в програмуванні, що дозволяє моделювати як

розгалуження, так і цикли. Зокрема, якщо один із виходів спрямовано до вузла, від якого існує транзитивний шлях до самого вузла керування, то утворюється ітераційний процес, умовою зупинки якого є активація одного з інших виходів.

Висновки до розділу 3

В цьому розділі було запропоновано архітектуру моделі єдиного алгоритмічного середовища, адаптовану до різнорідних задач процесу планування адитивного виробництва. На основі аналізу комплексу задач планування АВ та поточного стану розвитку галузі 3-D друку, сформульовано ключові вимоги до МЄАС: гнучкість, масштабованість, відкритість до розширення та ефективна підтримка роботи з алгоритмами, що модифікують геометричні дані.

Запропоновано класифікацію алгоритмів за характером змін, які вони вносять в дані. Введена класифікаціє виступає використовується для аналізу наслідків виконання композиції алгоритмів та прогнозування впливу окремих кроків робочого процесу на стан структур даних. Цей результат є ключовим для подальшого керування узгодженістю даних в МЄАС.

Введення менеджера структур даних як окремого інфраструктурного шару МЄАС забезпечує централізоване керування спільними структурами даних та ефективну підтримку їх валідності. Оновлення відбувається з використанням принципу відкладених обчислень. Завдяки цьому розробники алгоритмів звільняються від необхідності самостійно відстежувати актуальність структур даних, що, у свою чергу, сприяє модульності системи та полегшує її зовнішнє розширення шляхом додавання нових алгоритмів.

Суттєвим доповненням до існуючих досліджень моделі єдиного алгоритмічного середовища (Терещенко, 2011) та (Коцур, 2019) є введення механізму неявного зведення даних, який дозволяє розглядати різні репрезентації та структури даних як взаємопов’язані елементи. Зв’язки між репрезентаціями

подаються у вигляді правил зведення. Формалізація правил зведення виконана у вигляді орієнтованого редукційного графа створює можливість автоматизованого вибору найефективнішого способу отримання необхідних даних: шляхом прямої побудови або зведення, з використанням вже наявних репрезентацій.

Розширена архітектура МЄАС, що включає описані вище концепції та їх взаємодію наочно представлена на рисунку 3.3. На рисунку схематично проілюстровано роль кожного з архітектурних елементів МЄАС та показано, яким чином використання МСД дозволяє підтримувати узгодженість даних в під час виконання робочого процесу.

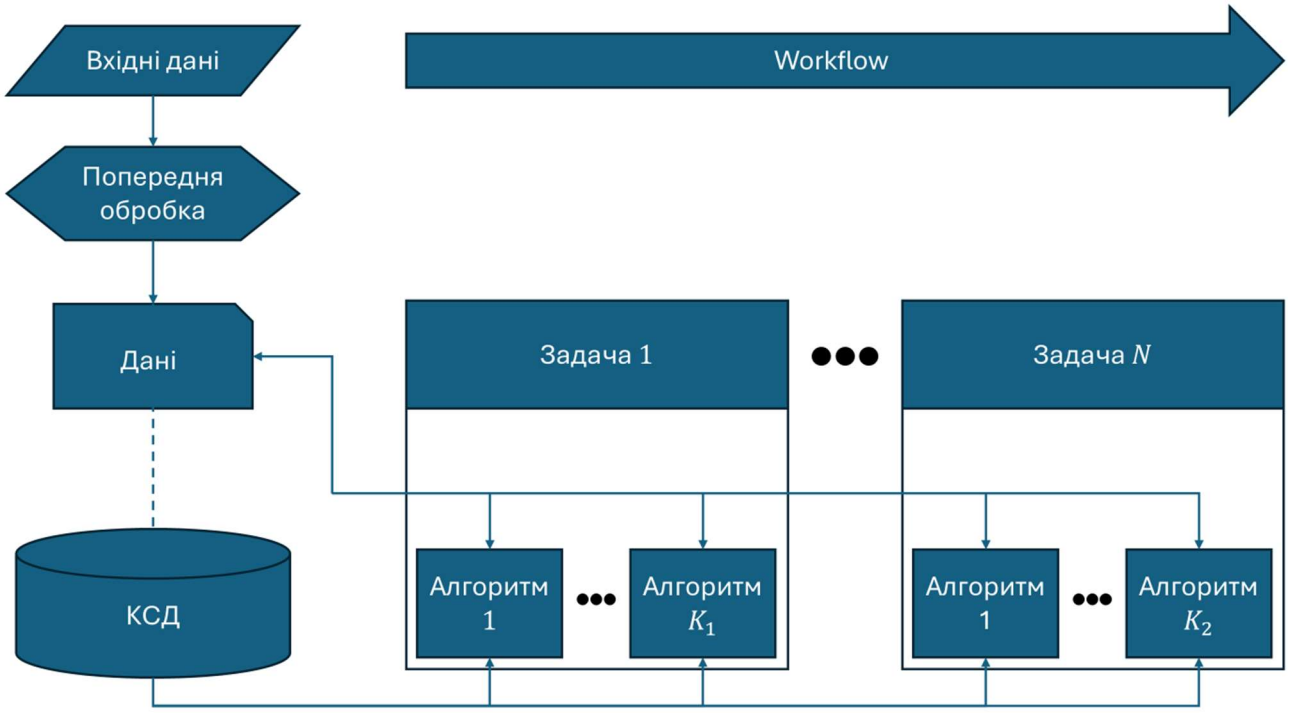

Рисунок 3.3. Схематичне зображення виконання робочого процесу

Викладені в цьому розділі результати формулюють архітектурну основу та формалізацію алгоритмічних процесів в МЄАС. В подальших розділах цієї роботи будуть розглянуті конкретні приклади задач та алгоритмів, які доречно реалізувати в МЄАС для забезпечення комплексного процесу планування АВ.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ КОМПЛЕКСУ ЗАДАЧ ПЛАНУВАННЯ АДИТИВНОГО ВИРОБНИЦТВА В МЄАС

В попередньому розділі було сформульовано теоретичні засади та архітектуру моделі єдиного алгоритмічного середовища, введено класифікацію алгоритмів за типом модифікацій геометричних даних та описано механізм керування наявними структур даних. Метою цього розділу є проєктування реалізації конкретних алгоритмічних задач планування АВ в межах запропонованої розширеної архітектури. Також, в цьому розділі будуть запропоновані нові алгоритми та методи оптимізації, застосування яких, разом з МЄАС, дозволяє отримати значний приріст швидкості розв’язання окремих задач АВ.

В підрозділі 4.1 проаналізовано шляхи можливої інтеграції існуючих алгоритмів розв’язання ключових задач планування АВ, огляд яких було наведено в розділі 2, в програмний комплекс на основі запропонованої архітектури МЄАС. Для кожного алгоритму визначено клас змін, відповідно до класифікації з розділу 3.2.1. Проаналізовано структури даних, що використовуються тим чи іншим алгоритмом з точки зору можливості її повторного використання через інтерфейс МСД. Зазначено можливі шляхи прискорення розглянутих алгоритмів за допомогою застосування принципу неявного зведення даних.

В підрозділі 4.2 наведено приклад ефективної інтеграції структур даних в запропоновану архітектуру МЄАС на прикладі графу Ріба (ГР) для плоского многокутника. Проведено аналіз властивостей ГР та його застосувань в задачах обчислювальної геометрії. Запропоновано алгоритм декомпозиції довільного плоского многокутника на множину y-монотонних многокутників, з можливістю одночасної їх тріангуляції. Встановлено асимптотичну оцінку складності зазначеного алгоритму в найгіршому випадку.

Підрозділ 4.3 присвячений розробці швидкого методу визначення геометричної подібності сусідніх слайсів моделі. Запропоновано алгоритм

відновлення декомпозиції многокутника, що описує слайс моделі, за існуючою декомпозицією подібного до нього слайса.

4.1 Інтеграція існуючих алгоритмів з МЄАС

Проаналізуємо існуючі підходи, розглянуті в розділі 2 цього дослідження, відповідно до запропонованої системи класів модифікацій, а також розглянемо їх на предмет сумісності з запропонованою архітектурою МЄАС.

4.1.1 Алгоритми конвертування CAD-репрезентацій та мешів

Першим етапом процесу планування АВ є отримання мешу, що відображає геометрію моделі з відповідної CAD-репрезентації або неявного представлення. В контексті архітектури МЄАС, цей крок виконується на етапі попередньої обробки (рисунок 3.3), але в процесі своєї роботи алгоритми можуть будувати структури даних, які доцільно залишити в МСД для повторного використання. Наприклад, алгоритм (Yang & Choi, 2010) будує kD-дерево для вершин мешу, яке може бути застосованим в подальшому для розв'язання задачі «пошук найближчого сусіда». Іншим прикладом є алгоритм CMS (Ho et al., 2005), що будує дерево октантів. В процесі генерування трикутників побудоване дерево може бути доповнене індексами відповідних трикутників, та збережене в МСД.

Також, алгоритми генерування мешів можуть використовуватись в процесі розв'язання інших задач АВ, зокрема, генерування опорних конструкцій, виправлення артефактів мешу, та побудови нового мешу на основі існуючого (наприклад, для створення відступу або заокруглення кутів). Відповідно до наведеної в розділі 3 класифікації, алгоритми цієї категорії належать до класу комплексних (радикальних) алгоритмів, оскільки вони формують первинну геометрію та топологію об'єкта, або повністю її перезаписують.

Для підтримки булевих операцій над мешами в МЄАС будемо використовувати алгоритм (Jiang et al., 2016), який було детально розглянуто в розділі 2. Зазначений алгоритм реалізує набір базових булевих операцій

(об’єднання, перетин та різниця) шляхом побудови дерева октантів для вхідних мешів, та аналізу їх спільної частини.

Згідно з запропонованою класифікацією, цей алгоритм належить до топологічно-тріангуляційних та геометричних класів, оскільки булева операція виконується шляхом вилучання непотрібних трикутників, та додавання нових. Тобто, змінюється як геометрія моделі, так і тріангуляція, що її репрезентує. Важливо, що алгоритм вносить зміни до окремих ділянок мешу, що дозволяє уникнути зміни всієї тріангуляції, що дозволяє МСД виконувати локальні оновлення таких структур даних, як BVH, шляхом видалення вузлів, що відповідають видаленим трикутникам, та додавання вузлів для нових трикутників, після чого виконується перевірка необхідності балансування дерева.

Також, використання розробленої архітектури МЄАС з МСД дозволяє оптимізувати роботу алгоритму (Jiang et al., 2016) шляхом повторного використання дерева октантів, яке могло бути побудоване в результаті застосування алгоритму CMS або, наприклад, при перевірці колізій між моделями. В такому випадку алгоритм безпосередньо переходить до визначення пар трикутників, що перетинаються.

4.1.2 Алгоритми розміщення моделі всередині робочої області принтера

Задача розміщення моделей в межах робочої області принтера є критичною для масштабування та підвищення продуктивності АВ. Розглянуті алгоритми розв’язання цієї задачі (Lutters et al., 2012) та (Pankratov et al., 2020) працюють над апроксимованою геометрією, що дозволяє спростити обчислення, та в якості кінцевого результату повертають матрицю трансформації координат мешу з простору моделі до глобальної системи координат. Відтак, обидва алгоритми належать до просторово-трансформаційного класу.

Алгоритм (Lutters et al., 2012) виконує велику кількість перевірок на перетин при зміні положення та орієнтації моделі, використовуючи апроксимації справжньої геометрії мінімальні обмежувальними сферами та паралелепіпеди, випуклі оболонки тощо. За традиційного підходу кожна трансформація моделі вимагала б перерахунку всіх її координат, та, відповідно, перебудови апроксимації. Натомість, застосування запропонованої архітектури МЄАС дозволяє залишити геометричні дані сталими. Оновлення структур даних виконується на рівні МСД, що, за рахунок застосування принципу відкладених обчислень та можливості застосувати обернену трансформацію до запиту (тобто трансформувати параметри запиту з глобальної системи координат в локальну систему координат моделі) дозволяє уникнути обчислювально-складної повної перебудови.

Також, МЄАС дозволяє ефективно апроксимувати модель у вигляді композиції базових геометричних примітивів (сфери, циліндри тощо) для застосування алгоритму (Pankratov et al., 2020). За наявності дерева октантів або kD-дерева в МСД, таке представлення моделі можна будувати шляхом пошуку оптимального обмежувального примітива для листових вузлів дерева.

4.1.3 Генерування опорних конструкцій

Алгоритми генерування опорних конструкцій, що розглядаються в цій роботі, виконують аналіз поверхні для виявлення ділянок, що потребують побудови опор. Такий аналіз не впливає на геометрію чи топологію моделі, однак, може змінювати її атрибути – виділяти ділянки, які необхідно посилити опорними конструкціями. Відповідно, застосування цієї групи алгоритмів виконує лише атрибутивні модифікації відносно моделей, для яких генеруються опори. Сам результат генерації є новим мешем, відносно якого процедура генерування опор є комплексною операцією (за аналогією до алгоритмів з підрозділу 4.1.1).

Алгоритм (Vaidya & Anand, 2016) використовує вокселізацію моделі для аналізу оптимальних шляхів побудови опор. Процес побудови вокселізації є

обчислювально витратним, однак, МСД дозволяє уникнути його, якщо попередні етапи робочого процесу (наприклад, термомеханічне моделювання, пакування, або генерування мешу) вже використовували воксельну сітку достатньої точності. В такому разі алгоритм Vaidya & Anand може використати її безпосередньо, або виконати процедуру деталізації існуючої сітки для отримання більш якісної апроксимації поверхні моделі.

Альтернативний підхід з роботи (Zhang et al., 2020) виконує велику кількість запитів проєктування точок та трасування променів для визначення ділянок, в яких деревовидні опорні конструкції приєднуються до моделі. Ефективність виконання таких запитів досягається за рахунок використання структур даних BVH або kD-дерев. Ці ж структури даних лежать в основі більшості з розглянутих вище алгоритмів, тому, з високою ймовірністю, алгоритму (Zhang et al., 2020) не доведеться виконувати їх побудову, оскільки їх можна буде отримати безпосередньо з МСД.

4.1.4 Слайсинг

Традиційно, слайсинг є останнім етапом обробки моделі як тривимірного об’єкту, але є дослідження використання набору слайсів при розв’язанні інших задач АВ, наприклад для побудови опорних структур в роботі (Jin et al., 2015). Результатом слайсингу є впорядкований набір двовимірних орієнтованих многокутників, які описують поверхню моделі. Алгоритм (Minetto et al., 2017), що використовує техніку замітаючої прямої, є асимптотично оптимальним: його складність складає $O(n \log k + k + m)$, де n – загальна кількість трикутників меша, k – кількість слайсів, та m – кількість перетинів трикутників з площинами.

Виконання слайсингу не змінює геометрію або топологією тривимірної моделі, однак потенційно може вносити атрибутивні зміни. Розглядаючи слайсинг в контексті запропонованої архітектури МЄАС, важливою є варіація алгоритму Minetto et al., яка дозволяє отримати лінійну обчислювальну

складність, якщо впорядкування трикутників мешу за мінімальною z координатою вершин є відомим.

Припустимо, що в МСД міститься воксельна репрезентація мешу (яка могла залишитись після генерування опор, пакування тощо). Шукане впорядкування трикутників мешу за мінімальною z будемо розглядати як одну з можливих репрезентацій даних, для якої може бути застосованим принцип неявного зведення. Важливо, що воксельна сітка зберігається у вигляді впорядкованої послідовності вокселів, що дозволяє ітеруватись по вокселях в порядку від найнижчого до найвищого без додаткових обчислень. Набори вокселів на одній висоті будемо розглядати як окремі групи. Побудуємо алгоритм, що використовує ідею алгоритму сортування комірками, детальний опис якого можна знайти в (Cormen et al., 2009, Chapter 8.4):

Алгоритм 4.1: Побудова впорядкованого списку трикутників мешу за мінімальною z координатою

Вхідні дані: *воксельна сітка, z-індекси якої знаходяться в інтервалі* $[z_{min}, z_{max}]$.

Вихідні дані: res *– впорядкований масив трикутників*

1 **for** $z = z_{min}$ **to** z_{max} **do begin**
2 $l_z \leftarrow$*triangles from voxels on layer* z
3 sort l_z
4 append l_z to res
5 **End**

Теорема 4.1. Попередня обробка алгоритму слайсинга з роботи (Minetto et al., 2017) в межах МЄАС може бути виконана за алгоритмом 4.1 за час $O(n + z_{max} - z_{min})$ в середньому.

Доведення. Алгоритм 4.1 використовує впорядкованість вокселів вздовж осі z. За рахунок цієї властивості трикутники в списку l_z перед виконанням 3-го рядка

псевдокоду є невпорядкованими відносно один одного, однак є впорядкованими відносно інших груп вокселів з відмінним z-індексом. При визначенні алгоритму сортування в 3-му рядку доцільно перевірити розмір списку l_z. При достатньо малих значеннях l_z (наприклад, якщо розмір списку складає $O(\sqrt{n})$, де n – кількість трикутників в меші) є сенс застосувати алгоритм сортування вставками, а для великих списків застосувати алгоритм сортування злиттям. У випадку мешу з «хорошою» тріангуляцією, яка забезпечує рівномірний розподіл трикутників вздовж осі z, обчислювальна складність алгоритму 4.1 буде відповідати середній обчислювальній складності алгоритму сортування комірками $O(n + z_{max} - z_{min})$, якщо кількість вокселів в сітці складає $O(n)$. ■

З точки зору реалізації, алгоритму 4.1 потрібно додатково підтримувати значення статусу кожного трикутника: «оброблений» чи «необроблений». Це необхідно, щоб ефективно відфільтровувати попередньо оброблені трикутники при побудові списку l_z, оскільки один трикутник може належати кільком вокселям. Тому цей алгоритм відноситься до атрибутивного класу.

4.1.4 Побудова траєкторії

Фінальним кроком процесу планування АВ є генерування траєкторії друку, найчастіше у вигляді орієнтованої ламаної або набору ламаних, для кожного слайсу моделі. Після генерації траєкторія конвертується в спеціалізовану мову програмування G-code, стандартизовану як ISO 6983-1:2009 (International Organization for Standardization, 2009), та подається на вхід принтеру.

Розглянемо алгоритм побудови траєкторії, описаний в роботі (Dwivedi & Kovacevic, 2004), що використовує метод декомпозиції довільного многокутника без на набір монотонних многокутників. Для кожного монотонного многокутника алгоритм генерує зигзагоподібну траєкторію, які потім зливаються в єдину замкнену траєкторію для кожної зв’язної частини вхідного многокутника. Властивість монотонності є ключовою в роботі алгоритму. Вона

забезпечує детермінованість перетинів межі многокутника та вектору нахилу зигзагоподібного шаблону.

В контексті запропонованої архітектури МЄАС, декомпозиція многокутника на монотонні відносно деякої прямої розглядається як окрема репрезентація вхідного многокутника. Важливо, що побудова такої декомпозиції може бути виконана як за допомогою класичного алгоритму, що використовує техніку замітаючої прямої, детальний опис якого можна знайти в (De Berg et al., 2008), так і з використанням підходу на основі графу Ріба (Tereshchenko et al., 2025). Останній алгоритм є цікавим з точки зору побудови ефективної мультиалгоритмічної системи, оскільки, як буде показано в наступних підрозділах цієї дисертації, він дозволяє отримати шукану декомпозицію за час $O(n)$ для многокутника з n вершинами, тоді як пряма побудова вимагає $O(n\log n)$ часу. Також, буде показано, що граф Ріба може бути застосованим для розв'язку інших задач обчислювальної геометрії: побудови монотонних ланцюгів (як попередню обробку в задачі локалізації точки), тріангуляції та пошуку мінімальної множини точок-опор для лінійчатих многокутників.

4.2 Ефективна інтеграція структур даних в МЄАС на прикладі графу Ріба

В попередніх розділах було сформульовано архітектурні принципи МЄАС, зокрема введено поняття МСД та неявного зведення представлень, що дозволяє координувати роботу різних алгоритмів в межах єдиного обчислювального середовища. Запропонований підхід передбачає явне виокремлення структур даних як самостійних об'єктів середовища, які підтримують коректність свого стану впродовж виконання робочого процесу. Тому особливого інтересу набувають інструменти, які, з одного боку, не є загальноприйнятими в класичних роботах з обчислювальної геометрії, а з іншого – дозволяють істотно скоротити обчислювальні витрати для низки задач за рахунок повторного використання результатів попередніх обчислень.

Одним з таких інструментів є граф Ріба (ГР), побудований для плоского многокутника відносно функції висоти. ГР застосовуються в топологічному

аналізі даних, зокрема для топологічного зіставлення форм (Ramamurthi & Chattopadhyay, 2022), однак особливий випадок ГР для плоских многокутників залишається недостатньо дослідженим. Водночас, граф Ріба є компактним представленням залежності між формою многокутника та його топологічною структурою. Як буде показано в наступних підрозділах, ця властивість може бути використана для прискорення розв'язання як загальних задач обчислювальної геометрії – локалізації точки та побудові тріангуляції, так і задач, що є специфічними для АВ – слайсингу.

4.2.1 Означення графу Ріба та відповідних структур даних

Нехай x, y – координати точки в деякій декартовій системі координат. Функцію $h(x, y) = y$ будемо називати функцією висоти. Граф Ріба для заданого плоского многокутника визначається як орієнтований планарний граф, що перетинає кожну компоненту рівня $h^{-1}(c)$ рівно в одній точці, а множина його вершин збігається з множиною вершин многокутника.

Очевидно, що з означення графу Ріба для відношення еквівалентності, заданого функцією висоти, випливає необхідність того, щоб функція $h(x, y)$ набувала різних значень у різних вершинах многокутника. Іншими словами, всі вершини многокутника мають розташовуватися на різних висотах. У випадку, коли для даного многокутника ця умова не виконується, її можна забезпечити шляхом повороту системи координат на деякий достатній кут за годинниковою стрілкою. Тому надалі вважаємо, що всі y-координати вершин многокутника є попарно різними. Крім того, в межах цієї роботи будемо вважати, що послідовність вершин, яка задає зовнішній цикл (границю) многокутника, впорядкована відповідно до обходу проти годинникової стрілки, тоді як послідовності вершин, що задають отвори (дірки), впорядковані за обходом за годинниковою стрілкою.

Для побудови графу Ріба вершини многокутника поділяють на наступні класи:

- V – вершини (початкові вершини), які є нижніми кінцями відповідних ребер, і ці ребра утворюють поворот проти годинникової стрілки.
- Λ – вершини (кінцеві вершини), які є верхніми кінцями відповідних ребер, і ці ребра утворюють поворот проти годинникової стрілки.
- Y – вершини (вершини розщеплення), які є нижніми кінцями відповідних ребер, але не є V – вершинами.
- ʎ – вершини (вершини злиття), які є верхніми кінцями відповідних ребер, але не є Λ – вершинами.
- Звичайні вершини, які є нижнім кінцем одного ребра та верхнім кінцем іншого.

Аналогічно, вершини графу Ріба поділяються на наступні типи:

- Джерелами називають вершини степені 1, з яких виходить ребро. Ці вершини відповідають початковим вершинам многокутника.
- Стоками називають вершини степені 1, в які входить ребро. Ці вершини відповідають кінцевим вершинам многокутника.
- Y – вершинами називають вершини степені 3, що мають одне вхідне ребро, та два вихідних ребра.
- ʎ – вершинами називають вершини степені 3, що мають два вхідних ребра, та одне вихідне ребро.
- Звичайні вершини, які є вершинами степені 2 з одним вхідним та одним вихідним ребром.

Плоский многокутник будемо представляти у вигляді трьох масивів:

- $vertices$ – невпорядкований масив вершин многокутника
- $next$ – масив індексів наступних вершин. Інакше кажучи, вершина $vertices[next[i]]$ є наступною після вершини $vertices[i]$ відповідно до порядку обходу вершин многокутника.

- $prev$ – масив індексів попередніх вершин. Інакше кажучи, вершина $vertices[prev[i]]$ є попередньою вершиною перед $vertices[i]$ відповідно до порядку обходу вершин многокутника.

Такий підхід дозволяє ефективно описувати як прості многокутники, так і многокутники з дірками однією структурою даних.

Зазначимо, що наведенні нижче алгоритми легко адаптуються до традиційного представлення многокутника у вигляді реберного списку з подвійними зв'язками (РСПЗ). Однак використання компактних лінійних структур даних на основі масивів має кілька переваг (Aleardi & Devillers, 2018). Зокрема, такий підхід дає змогу застосовувати індексну арифметику під час розробки алгоритмів і організовувати елементи структури даних у послідовних ділянках пам'яті, що дозволяє процесору ефективніше використовувати кеш.

Згідно з теоремою 2, наведеною в (Tereshchenko & Prishlyak, 2024), множина вершин ГР збігається з множиною вершин відповідного плоского многокутника. З огляду на це, для подання графа Ріба доцільно застосовувати список суміжності, де індекси вершин відповідають індексам елементів масиву $vertices$.

Для подання тріангуляції плоского многокутника в цій роботі використовується масив $triangles$, що містить трійки індексів вершин трикутників в масиві $vertices$. Отже, трійка $triangles[t] = (i_t, j_t, k_t)$ визначає трикутник, вершинами якого є $vertices[i_t]$, $vertices[j_t]$ та $vertices[k_t]$.

4.2.2 Алгоритм побудови графу Ріба

Побудову графу Ріба для плоского многокутника будемо виконувати методом замітаючої прямої. На першому етапі алгоритму виконується сортування вершин многокутника за зростанням координати y та визначення класу кожної вершини відповідно до класифікації, наведеній в розділі 4.2.1. Далі розпочинається процес «замітання» вершин многокутника горизонтальною прямою, починаючи з найнижчої. Точками події цього процесу є вершини

многокутника, а статусом процедури замітання є множина ребер многокутника, що перетинаються з поточною горизонтальною прямою, впорядкована за x координатою точок перетину. При цьому припускається, що ребра многокутника спрямовані від нижчої вершини до вищої. Обробка кожної точки події виконується наступним чином:

i. Визначення найвищої обробленої вершини, яка є видимою з поточної. Якщо така вершина існує, то до графа Ріба додається ребро між нею та поточною вершиною.
ii. Визначення ребер многокутника, що входять у поточну вершину, та їхнє видалення зі статусу, якщо такі ребра присутні.
iii. Визначення ребер многокутника, що виходять з поточної вершини, та їхнє додавання до статусу, якщо такі ребра присутні.

Для ефективного підтримання статусу замітання можна використовувати збалансоване бінарне дерево пошуку, що забезпечить виконання операцій пошуку, додавання та вилучення ребер зі статусу за час $O(\log n)$, де n - кількість вершин многокутника.

Ключовим етапом обробки події в процесі замітання є визначення найвищої раніше обробленої вершини, яка є видимою з поточної вершини. Для вирішення цієї задачі введемо таку характеристику ребра: $visible[e]$ - найвища оброблена вершина, для якої горизонтальний відрізок між ребром e та цією вершиною повністю знаходиться всередині многокутника. Нехай e_l - ребро, що знаходиться безпосередньо ліворуч від поточної вершини, а e_r - ребро, що розташоване безпосередньо праворуч від поточної вершини. Тоді найвищою обробленою вершиною, яка є видимою з поточної вершини, буде та з вершин $visible[e_l]$ та $visible[e_r]$, яка має більшу y координату. Цей підхід схожий на використання масиву $helper$ в алгоритмі тріангуляції y-монотонного многокутника, що описаний в (De Berg et al., 2008), але враховує ребра по обидві сторони від поточної вершини.

Теорема 4.2. Граф Ріба для плоского многокутника з n вершин у найгіршому випадку може бути побудований методом замітання за час, що не перевищує $O(n\log n)$.

Доведення: Сортування вершин за y координатою вимагає $O(n\log n)$ часу. Розглянемо процес обробки події під час процесу замітання. На кожному з n кроків можливі наступні операції:

- *Вставка*. На кожному кроці до дерева може бути додано щонайбільше два ребра (при обробці V або Y – вершин).
- *Пошук*. Не більше двох запитів пошуку для визначення e_l та e_r.
- *Видалення*. Не більше двох запитів видалення (при обробці Λ та Λ – вершин).

Оремо кожна з цих операцій виконується за час $O(\log n)$. Для кожної вершини кількість викликів кожної операції обмежена двома, тому загальний час обробки однієї вершини складає $O(\log n)$. Як результат, обробка всіх вершин займає $O(n\log n)$ часу. Таким чином, процедура побудови графа Ріба для плоского многокутника з n вершин є композицією двох послідовних етапів, кожен з яких виконується за час $O(n\log n)$. Отже, загальний час виконання алгоритму також становить $O(n\log n)$. ■

З визначення Λ-вершини многокутника слідує, що вершини цього типу можуть міститись лише в многокутнику з дірками. Відповідно, для простого многокутника ГР є деревом (за побудовою, лише Λ-вершини графу замикають дві гілки графу в один ланцюг). Такий ГР також називають *контурним деревом*. Контурні дерева часто розглядають як окрему структуру даних, алгоритм побудови якої наведено в роботі (Carr et al., 2003). Однак, цей алгоритм не має асимптотичної переваги в порівнянні з алгоритмом побудови ГР, тому в межах МЄАС будемо розглядати контурні дерева для плоских многокутників як частковий випадок графу Ріба, та виконувати побудову алгоритмом, що описаний вище в цьому розділі.

4.2.3 Властивості графу Ріба та його використання для розв'язку деяких задач обчислювальної геометрії

Розглянемо властивості ГР для плоского многокутника, які в наведеній архітектурі МЄАС можуть виступати в ролі правили зведення репрезентацій, або як теоретична основа окремих низькорівневих алгоритмів обчислювальної геометрії (чи їх оптимізації).

Важливою властивістю графа Ріба, доведеною в роботі (Tereshchenko & Prishlyak, 2024), є можливість розбиття плоского многокутника на y-монотонні прості многокутники за допомогою діагоналей, що сполучають Y-вершини з вершинами, від яких виходить ребро ГР до відповідної Y-вершини, а також між λ-вершинами та вершинами, до яких проведено ребро графа Ріба з λ-вершини. З цієї властивості безпосередньо випливає одне з можливих застосувань графа Ріба - регуляризація розбиття (у вигляді плоского многокутника) для використання методу ланцюгів локалізації точки (Терещенко, 2013).

В роботі (Cavanna et al., 2017) досліджено використання ГР для плоского многокутника при побудові мінімальної множини точки опор для лінійчатого многокутника. Під лінійчатим многокутником розуміється плоский многокутник та набір хорд, що попарно не перетинаються, кінці яких розміщені на границі многокутника.

Розглянемо інше можливе застосування графу Ріба для плоского многокутника – декомпозицію многокутника на множину y-монотонних многокутників з одночасною побудовою їх тріангуляції. Алгоритм, що буде запропоновано в цьому розділі виконує одночасне розв'язання зазначених задач, але, з метою спрощення подачі інформації, спочатку окремо розглянемо алгоритм декомпозиції плоского многокутника на y-монотонні многокутники з використанням ГР. Після цього буде наведено алгоритм побудови тріангуляції,

який може бути безпосередньо інтегрований у процес виконання процедури декомпозиції.

Позначимо через $order[i]$ порядковий номер вершини з індексом i в послідовності вершин многокутника, впорядкованих за зростанням y-координати. Вважатимемо, що масив $order$ є відомим, оскільки сортування вершин було виконано на етапі попередньої обробки даних під час побудови графу Ріба, та може бути отримано з МСД.

Ідея алгоритму полягає у послідовному переборі вершин многокутника в «основного» циклі, починаючи з вершини з найменшою y-координатою. Для кожної необробленої вершини будемо будувати y-монотонний многокутник, найнижчою точкою якого є ця вершина. В процесі декомпозиції, для кожної вершини алгоритм підтримує лічильник її відвідин. Значення цього лічильника надалі використовується в основному циклі: вершини, для яких кількість відвідин дорівнює степеню відповідної вершини в графі Ріба, пропускаються (вони вважаються обробленими), оскільки всі y-монотонні многокутники, що містять ці вершини, вже були виділені алгоритмом. Для кожної поточної вершини в основному циклі алгоритм визначає лівий і правий ланцюги відповідного y-монотонного многокутника.

Алгоритм виокремлення монотонних ланцюгів виконується на орієнтованому графі, який будується на основі ребер многокутника та його графу Ріба наступним чином: до графа включаються всі ребра многокутника, а також ті ребра ГР, які або виходять із вершини типу Λ, або входять у вершину типу Y. Орієнтація ребер задається від вершини з меншою y-координатою до вершини з більшою y-координатою. Надалі цей граф будемо називати доповненим вертикально орієнтованим графом многокутника (доповнений ВГ). Приклад доповненого ВГ для плоского многокутника з дірками зображено на рисунку 4.1.

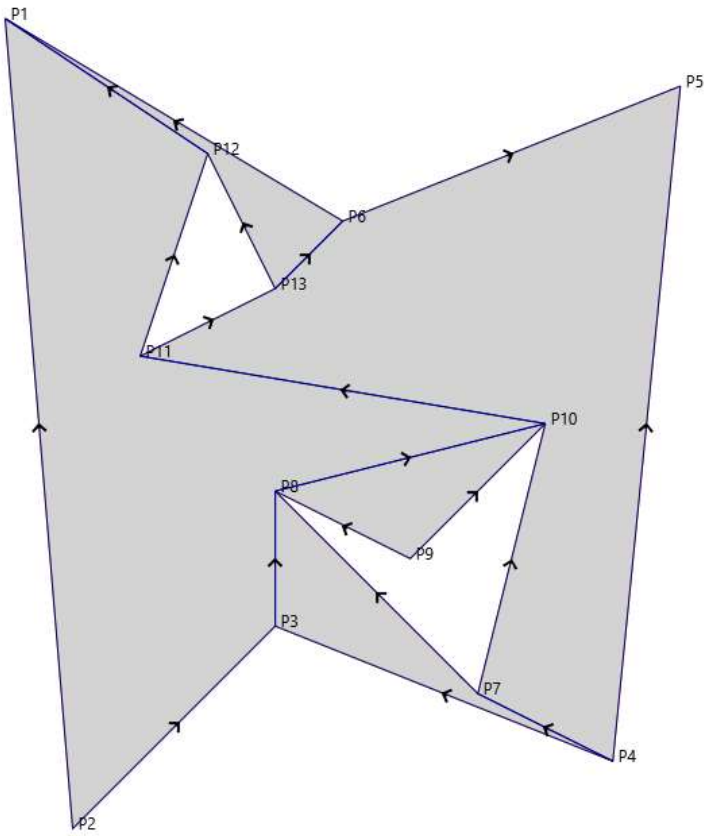

Рисунок 4.1. Многокутник та відповідний доповнений вертикально-орієнтований граф

На першому кроці обробки поточної вершини алгоритм визначає стартові ребра лівого та правого ланцюгів. Лівому ланцюгу відповідає найлівіше ребро, що виходить із поточної вершини, тоді як правому – ребро, що розташоване безпосередньо праворуч від нього. Подальша побудова ланцюгів здійснюється шляхом підйому вздовж ребер доповненого ВГ відповідно до наведених нижче правил:

- Якщо поточна вершина належить до класу Λ – вершин, або в доповненому ВГ існує ребро між нею та Y – вершиною, то ланцюг доповнюється ребром із доповненого ВГ, що виходить із цієї вершини. Для лівого ланцюга обирається найправіше таке ребро, а для правого – найлівіше.
- В іншому випадку ланцюг доповнюється наступним ребром з многокутника.

Таким чином, алгоритм використовує жадібну стратегію: під час побудови лівого ланцюга він рухається якомога правіше, а під час побудови правого – якомога лівіше. Зазначений процес триває доти, доки обидва ланцюги не зійдуться в одній вершині.

Розглянемо модифікацію описаного вище алгоритму, що дозволяє паралельно виконувати тріангуляцію y-монотонного многокутника під час побудови лівого та правого ланцюгів. Результат тріангуляції подається у вигляді масиву трійок індексів вершин трикутників. Зазначимо, що процес «підйому» по ланцюгах буде здійснюватися одночасно для обох ланцюгів. Для цього достатньо обирати ланцюг, що будується на поточній ітерації, з врахуванням y координати наступних вершин ланцюга. До ланцюгів, що формуються таким чином, можна застосувати підхід, який використовується в класичному алгоритмі тріангуляції y-монотонного многокутника, що описано, зокрема, в (De Berg et al., 2008): після додавання чергової вершини до одного з ланцюгів тріангуляція доповнюється відповідними ребрами між цією вершиною та частиною раніше оброблених вершин. Основні відмінності від класичного алгоритму полягають у тому, що обхід вершин здійснюється знизу вгору, а не навпаки, відповідно до орієнтації ребер у ГР, а також технічною необхідністю додавати до структури даних трикутники, а не окремі ребра. Остання відмінність потребує визначення третьої вершини трикутника, що на перший погляд може виглядати як ускладнення задачі порівняно з підходом De Berg et al., однак цей підхід дозволяє отримати результат у вигляді простіших структур даних, переваги яких були детально розглянуті в розділі 2.3.

Для розуміння того, як потрібно модифікувати алгоритм тріангуляції y-монотонного многокутника з врахуванням необхідності визначення третьої вершини, слід звернути увагу на те, що в класичному алгоритмі, під час розгортання стека, фактично будується верхній дотичний відрізок для трійки точок – вершин трикутника, одна з яких на даний момент ще є невідомою. Нижній дотичний відрізок для цієї трійки точок є відрізком між вершиною зі стеку та

шуканою третьою вершиною. Цей відрізок повинен бути верхнім опорним відрізком для останнього з уже побудованих трикутників, вершинами якого є ці дві вершини.

Таким чином, для визначення третьої вершини трикутника можна ввести додатковий масив $tangent$, де $tangent[i]$ позначає індекс вершини, яка в парі з i-ю вершиною утворює верхній дотичний відрізок попереднього побудованого трикутника з вершиною i. Якщо ж при обробці вершини i не було побудовано жодного трикутника, то $tangent[i]$ є індексом вихідної вершини ребра доповненого ВГ, за яким обхід потрапив до вершини i.

Об'єднавши процедуру підйому по лівому та правому ланцюгах, можна отримати наступний алгоритми.

Алгоритм 4.2*: *Додавання вершини до тріангуляції
Вхід: $vertices$ – масив вершин многокутника, $stack$ – поточний стек вершин, $tangent$ – допоміжний масив, описаний вище, cv – індекс поточної вершини, cc – індекс поточного ланцюга.
Вихід: оновлена тріангуляція

```
1  top ← top(stack)
2  if top.c = cc  begin
3        pv ← top.v
4        tangent[cv] ← pv
5        while stack не порожній do begin
6              if (трійка vertices[pv], vertices[cv], vertices[top.v]
                 орієнтована проти годинникової стрілки та cc – лівий
                 ланцюг) або (трійка
                 vertices[pv], vertices[cv], vertices[top.v] орієнтована за
                 годинниковою стрілкою та cc – правий ланцюг) begin
7                    break
8              end
9              top ← top(stack)
10             pop(stack)
11             yield triangle (cv, top.v, tangent[top.v])
12             tangent[cv] ← tangent[top.v]
13             pv ← tangent[top.v]
14       end
15       push(stack, cv, cc)
16 Else
17       while stack не порожній do begin
```

```
18              top ← top(stack)
19              pop(stack)
20              yield triangle (cv, top.v, tangent[top.v])
21          end
22          tangent[cv] ← top.v
23          push(stack, cv, cc)
24  End
```

Наведена вище процедура додавання вершини до тріангуляції (алгоритм 4.2) уточнює поточну тріангуляцію з врахуванням нової вершини. Ця процедура під назвою $AddVertexToTriangulation$ буде використана в наступному алгоритмі, що будує декомпозицію многокутника на y-монотонні многокутники та їх тріангуляцію.

Алгоритм 4.3*: *Побудова декомпозиції та тріангуляції
Вхід: efg – доповнений ВГ, $vertices$ – масив вершин многокутника, $order$ – масив y-впорядкованих індексів вершин многокутника.
Вихід: $monotone\_polygons$ – список y-монотонних многокутників, $triangles$ – тріангуляція

```
 1  for i = 0 to size(vertices) do begin
 2          cv ← order[i]
 3          while efg[cv] містить ребра do begin
 4                  left ← efg[cv][0]
 5                  remove edge(efg[cv], 0)
 6                  right ← efg[cv][0]
 7                  decrement_capacity_or_remove_edge(efg[cv], 0)
 8                  stack ← empty_stack()
 9                  if order[left] < order[right] do begin
10                          push(stack, left, 0)
11                          tangent[left] ← cv
12                          pci = 1
13                  else
14                          push(stack, right, 1)
15                          tangent[right] ← cv
16                          pci = 0
17                  end
18                  chains ← list[2]
19                  append(chains[0], left)
20                  append(chains[1], right)
21                  while end(chain[0]) ≠ end(chain[1]) do begin
```

```
22              if efg[end(chains[0])] порожній або
                (efg[end(chains[1])] не порожній та
                order[begin(efg[chains[1]])] <
                order[begin(efg[chains[0]])]) do begin
23                  nv ← efg[end(chains[1])]
24                  nvi ← 0
25                  cc ← 1
26              else
27                  nv ← efg[end(chains[0])]
28                  nvi ← size(efg[end(chains[0])]) − 1
29                  cc ← 0
30              end
31              if pci ≠ −1 та order[chains[pci][0]] < order[nv] do
                begin
32                  triangles ← AddVertexToTriangulation(vertices,
                    stack, tangent, chains[pci][0], pci)
33                  pci = −1
34              End
35              decrement_capacity_or_remove_edge(
                efg[end(chains[cc])], nvi)
36              append(chains[cc], nv)
37              triangles ← AddVertexToTriangulation(vertices,
                stack, tangent, nv, cc)
38          end
39          while stack не порожній do begin
40              top ← top(stack)
41              yeild triangle (end(chains[0]), top, tangent[top. v])
42              pop(stack)
43          end
44          monotone_polygons ←злити монотонні ланцюги chains[0] та
            chains[1]
45      End
46  End
```

Теорема 4.3 Плоский многокутник з n вершин може бути розділений на множину тріангульованих y-монотонних многокутників в межах МЄАС з допомогою алгоритму 4.3 за час, що не перевищує $O(n)$ в найгіршому випадку.

Доведення: В процесі виконання процедури підйому, кожне ребро доповненого ВГ відвідується двічі у випадку, якщо воно було додане як ребро графа Ріба, а в іншому разі відповідне ребро відвідується один раз. Враховуючи планарність ГР

та доповненого ВГ, обхід графу здійснюється за час $O(n)$. Розгортання стеку в процесі побудови тріангуляції також потребує $O(n)$ часу, як показано в роботі (De Berg et al., 2008). Отже, загальна часова складність запропонованого алгоритму становить $O(n)$. ■

Алгоритм 4.3 побудови тріангуляції з використанням графа Ріба може бути застосований не лише в задачах, які безпосередньо використовують тріангуляцію як модель даних. Він є може бути застосованим при розробці ефективних алгоритмів розв'язання інших задач обчислювальної геометрії або оптимізації.

Як приклад можна розглянути задачу побудови видимої області простого многокутника з заданої точки. Алгоритм розв'язання цієї задачі з часовою складністю $O(n)$, наведений у роботі (Joe & Simpson, 1987), є асимптотично оптимальним. Проте, на практиці часто трапляються випадки, коли з точки запиту видимою є лише невелика частина многокутника. До таких випадків належать, зокрема, зіркові многокутники, якщо точка запиту не належить їх ядру, а також схеми будівель або міських вулиць, які часто виникають у класичній задачі картинної галереї (англ. Art Gallery Problem) та її варіаціях.

Алгоритм Joe & Simpson виконує повний обхід многокутника, що є неефективним в зазначених випадках. Одним із можливих шляхів оптимізації цього алгоритму є попередня обробка у вигляді побудови тріангуляції. Це дає змогу уникнути виконання повного обходу заздалегідь невидимої частини полігону під час пошуку наступного видимого ребра шляхом переходу через трикутники, які утворюють межу між видимою та невидимою областями многокутника. В контексті запропонованої архітектури МЄАС, така оптимізація може бути реалізована у вигляді правила зведення, яке вимагає наявність побудованого ГР в МСД.

4.3 Оптимізація алгоритмів побудови траєкторії друку

В цьому розділі буде продовжено дослідження способів оптимізації задачі побудови траєкторії друку. В розділі 4.2 було показано, яким чином інтеграція

графу Ріба в МСД дозволяє оптимізувати процес побудови y-монотонної декомпозиції многокутників, яка застосовується при побудові траєкторії алгоритмом (Dwivedi & Kovacevic, 2004). Проте, існують альтернативні підходи до побудови траєкторії, що використовують ідею декомпозиції вхідного многокутника на інші типи результуючих многокутників.

Зокрема, в роботі (Ding et al., 2014) виконується декомпозиція на множину опуклих многокутників. Побудова такої декомпозиції є обчислювально складнішою, в порівнянні з y-монотонним розбиттям. На практиці найчастіше застосовуються наступні алгоритми розв'язання цієї задачі: підхід з роботи (Keil & Snoeyink, 1998) будує оптимальне розбиття з точки зору кількості многокутників, однак не може бути застосованим до многокутників з дірками, а час роботи алгоритму може сягати $O(n^3)$, де n – кількість вершин многокутника. Іншим популярним підходом є алгоритм (Hertel & Mehlhorn, 1985), що не гарантує оптимальної кількості многокутників, але дає можливість виконати декомпозицію многокутника з дірками, та має часову складність $O(n^2)$.

Алгоритми Dwivedi & Kovacevic та Ding et al. успішно розв'язують задачу побудови траєкторії для кожного окремого слайсу моделі. Однак, розглядаючи ці підходи в більш широкому контексті, не важко помітити, що ці алгоритми розглядають слайси як множину незалежних об'єктів. При цьому, вхідними даними цих для цих алгоритмів виступає набір *послідовних* слайсів - горизонтальних перерізів моделі, що друкується. Під час друку кожен наступним шар опирається на попередні, тому многокутники, що їх описують, є подібними в певному розумінні. Метою цього підрозділу є розробка методу оптимізації зазначених вище алгоритмів як складових частин МЄАС, що дозволить врахувати подібність сусідніх шарів моделі та уникнути зайвих обчислень.

4.3.1 Критерій близькості многокутників

В залежності від вимог, на сьогодні використовуються різноманітні підходи до перевірки подібності многокутників. Найбільш поширеними є використання

метрик Гаусдорфа (D_H) та Чамфера (D_C), які визначають відстань між многокутниками P з n_P вершин та P' з $n_{P'}$ вершин наступним чином:

$$\overrightarrow{D_H}(P, P') = \sup_{p \in P} \inf_{p' \in P'} \|p - p'\|$$

$$D_H(P, P') = \max\left(\overrightarrow{D_H}(P, P'), \overrightarrow{D_H}(P', P)\right)$$

$$D_C(P, P') = \frac{1}{2n_P} \sum_{p \in P} \min_{p' \in P'} \|p - p'\| + \frac{1}{2n_{P'}} \sum_{p' \in P'} \min_{p \in P} \|p - p'\|$$

При цьому, значення $\overrightarrow{D_H}(P, P')$ називають односторонньою відстанню Гаусдорфа.

Існують дослідження більш просунутих метрик, таких як метрика PoLiS (Avbelj et al., 2015). Також, досить просто можна перевірити, чи відрізняються многокутники з точністю до трансформації, що зберігає кути многокутника: переносу, повороту, масштабу або їх композиції. Ідея такої перевірки полягає в наступному: за таких операцій послідовність кутів одного многокутника має бути циклічним зсувом послідовності кутів іншого многокутника. Розглядаючи ці послідовності як рядки, оригінальна задача зводиться до перевірки, чи є один рядок циклічним зсувом іншого, яку можна розв'язати за лінійний час, використавши алгоритм Кнута-Морріса-Пратта (Knuth et al., 1977), вхідним рядком якого виступає конкатенація послідовності кутів одного многокутника сама з собою, а вхідним словом – послідовність кутів іншого многокутника.

Суттєвим недоліком більшості наведених вище підходів є трактування многокутників виключно як певну множину точок, без врахування деталей їх геометричної форми. Так, для кожної точки чорного багатокутника, зображеного на рисунку 4.2, можна з достатньою точністю підібрати відповідну точку на зеленому багатокутнику, і навпаки. Відповідно до розглянутих вище критеріїв подібності, різниця між такими багатокутниками є мінімальною. Однак, взявши до уваги їх форму, легко помітити, що чорний многокутник є трапецією, чого не можна сказати про зелений многокутник.

Рисунок 4.2. Приклад многокутників

Як було зазначено вище, у цьому підрозділі основну увагу приділено алгоритмам побудови траєкторії Dwivedi & Kovacevic та Ding et al., які виконують декомпозицію множини довільних многокутників на множину многокутників із наперед відомими властивостями. Процедури декомпозиції, що застосовуються в цих алгоритмах, нелінійно залежать від кількості вершин у наборі многокутників і виконуються окремо для кожного слайсу. Такий підхід є неефективним, оскільки більшість сусідніх слайсів у контексті AB не мають суттєвих геометричних відмінностей між собою.

Однак, враховуючи специфіку задачі, критерій визначення подібності сусідніх слайсів має враховувати не лише близькість множин точок, що описують відповідні многокутники, а й деталі їх форми.

Для формулювання критерію схожості многокутників, що задовольняє зазначену вище вимогу, необхідно ввести поняття області близькості та побудувати алгоритм перевірки повторюваності їхньої форми. Нехай a – деякий відрізок на площині, а δ – задана точність перевірки схожості. Геометричне місце точок площини, відстань від яких до відрізка a а не перевищує δ, називатимемо стадіоном (*англ. stadium*) і позначатимемо $S_\delta(a)$. Стадіон є об'єднанням прямокутника, в якому відрізок a а виступає серединною лінією, та двох півкіл радіусом δ. Тоді область δ – близькості для многокутника P, заданого відрізками $p_1, p_2, \dots, p_n$, визначимо як об'єднання відповідних δ – стадіонів:

$$\bigcup_{\delta} P = \bigcup_{i=1}^{n} S_\delta(p_i).$$

Отже, $\bigcup_\delta P$ є геометричним місцем точок, відстань від яких до многокутника P не перевищує δ. Приклад області δ – близькості для простого многокутника наведено на рисунку 4.3.

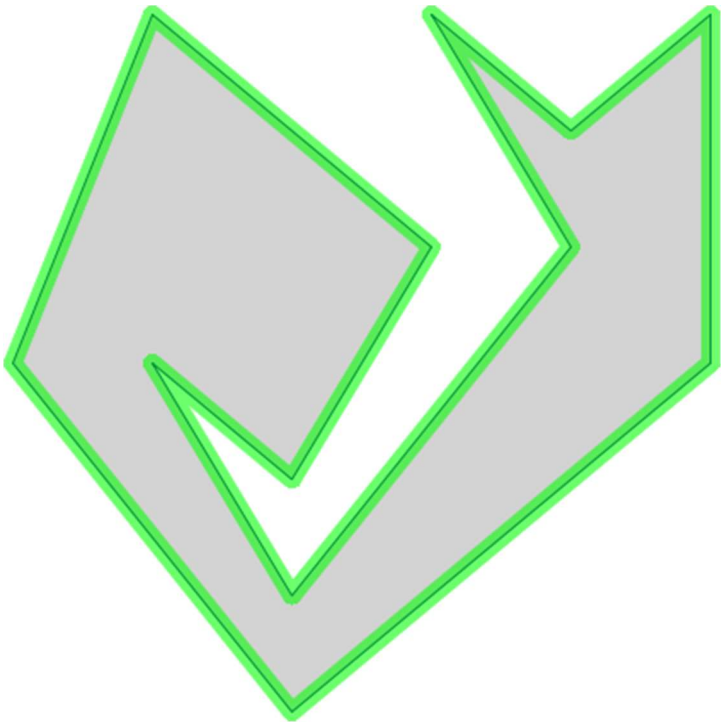

Рисунок 4.3. Простий многокутник та його область δ – близькості (зображена зеленим)

Область δ – близькості дає змогу оцінювати односторонню відстань Гаусдорфа $\overrightarrow{D_H}(P, P')$, оскільки для будь-якої точки многокутника P', який повністю міститься всередині $\bigcup_\delta P$, гарантовано існує точка на многокутнику P, відстань до якої не перевищує δ. Ця властивість має важливе практичне значення, оскільки обрахунок відстані Гаусдорфа в явному вигляді є обчислювально важкою задачею. Алгоритми, що забезпечують лінійний час роботи, відомі лише для окремого випадку, коли обидва багатокутники є опуклими (Atallah, 1983), тоді як універсальні підходи, зокрема алгоритм, запропонований у роботі (Bai et al., 2011), потребують побудови структур даних, таких як R-дерева, що істотно впливає на загальний час виконання.

Запропонованому в цьому розділі алгоритму не потрібно визначати точне значення відстані Гаусдорфа. Натомість, достатньо перевірити виконання умови $D_H(P, P') \leq \delta$, яка за означенням області δ – близькості, є еквівалентною одночасному виконанню умов $P' \subseteq \bigcup_\delta P$ та $\subseteq \bigcup_\delta P'$.

Спираючись на введене поняття області δ – близькості, побудуємо процедуру, яка дозволяє перевірити, чи повторює заданий многокутник форму іншого многокутника з точністю δ. Нехай задано многокутник P з n_P вершинами, для якого побудовано область δ – близькості, а також деякий многокутник P' з $n_{P'}$ вершинами. Ідея алгоритму перевірки полягає у пошуку ділянки многокутника P, що відповідає кожному ребру многокутника P'. При цьому накладання таких ділянок ненульової довжини не допускається. Крім того, в процесі роботи алгоритму можуть додатково перевірятися властивості многокутника, зокрема опуклість, монотонність, ізотетичність тощо, якщо це є релевантним у контексті розв'язуваної задачі. В разі, якщо алгоритму вдалось знайти відповідні ділянки для всіх ребер многокутника P', та його ребра повністю містяться в частині області δ – близькості, що відповідає ділянці многокутника P, многокутник P' вважаємо таким, що повторює форму многокутника P з точністю δ.

Зауважимо, що належність многокутника P' до області δ – близькості многокутника P сама по собі не гарантує виконання сформульованої вище умови «повторюваності» форми многокутника P многокутником P'. Зокрема, у випадку, коли внутрішній кут між двома суміжними ребрами многокутника P є гострим, многокутник P' може «зрізати» відповідну ділянку, не доходячи до спільної вершини цих ребер у многокутнику P. Саме з огляду на це, описана вище процедура виконує пошук відповідних ділянок між ребрами многокутників, або їх частинами.

4.3.2 Алгоритм порівняння многокутників

Нехай многокутник P складається з k_P поліліній: простих многокутників $P_1, P_2, \dots, P_{k_p}$. Серед них принаймні один простий многокутник визначає зовнішню границю многокутника P, тоді як решта простих многокутників розташовані всередині цієї межі та утворюють отвори (дірки). Для визначеності вважатимемо, що вершини простих многокутників, які задають зовнішню границю P впорядковані проти годинникової стрілки, тоді як вершини многокутників-отворів орієнтовані за годинниковою стрілкою. Аналогічно припустимо, що многокутник P' складається з k'_P простих многокутників $P'_1, P'_2, \dots, P'_{k_{P'}}$.

Алгоритм, який реалізує описану в розділі 4.3.1 процедуру, доцільно поділити на такі етапи:

- Пошук точки (допускаються як вершини, так і точки на ребрі) на многокутнику P', що відповідає деякій початковій вершині одного з простих многокутників $P_i, i = \overline{1, k_P}$.
- Одночасний обхід вершин полілінії $P_i, i = \overline{1, k_P}$, та ребер многокутника P', які відповідають ребрам P_i відповідно до конфігурації многокутника P.

Перший етап алгоритму можна інтерпретувати як окремий випадок задачі близькості, а саме задачу пошуку найближчого сусіда. Відповідно, одним із можливих підходів до її розв'язання є застосування класичних алгоритмів, що базуються на використанні kD-дерев та подібних структур даних, побудова яких вимагає нелінійного часу. Тому нижче будуть запропоновані швидші методи, які враховують контекст та специфіку розглядуваної задачі.

Розглянемо два можливі підходи до реалізації першого кроку алгоритму. Перший підхід є ефективним у випадку, коли $k_P \ll n$. Для кожного простого многокутника $P'_1, P'_2, \dots, P'_{k_{P'}}$ побудуємо мінімальні обмежувальні паралелепіпеди $B'_1, B'_2, \dots, B'_{k_{P'}}$, та представимо отриманий набір у вигляді ієрархії

обмежувальних об'ємів (BVH-дерева), у листових вузлах якого зберігаються обмежувальні паралелепіпеди многокутників, що утворюють P'.

Далі, для кожного простого многокутника $P_i, i = \overline{1, k_P}$ обчислюємо його мінімальний обмежувальний паралелепіпед B_i та виконуємо спуск по BVH-дереву з метою пошуку елемента листового вузла, для якого коефіцієнт Жаккара (Jaccard, 1901) є максимальним: $J_{P_i} = \max\limits_{j=1,k_{P'}} J(B_i, B'_j)$, де $J(B_i, B'_j)$ – відношення площі перетину паралелепіпедів B_i та B'_j до площі їх об'єднання. Якщо отримане значення J_{P_i} є меншим за заданий пороговий рівень, або якщо під час спуску по дереву не було відвідано жодного листового вузла, вважаємо, що для многокутника P_i відсутній відповідник серед набору $P'_1, P'_2, \dots, P'_{k_{P'}}$. В іншому випадку початкову вершину p_i^s многокутника P_i обираємо як його першу вершину, а як відповідну їй вершину $p_i^{s'}$ – найближчу до неї точку того многокутника з набору $P'_1, P'_2, \dots, P'_{k_{P'}}$, для якого досягається максимум значення J_{P_i}.

Інший можливий підхід використовує специфіку задачі слайсингу. Многокутники P та P' отримуються в результаті перетину полігональної сітки горизонтальними площинами, розташованими на невеликій відстані одна від одної. В цьому випадку точку-відповідник доцільно шукати на ребрах, утворених перетином тих трикутників, перетин ребер яких формує вершину многокутника P, або ж на ребрах одного із суміжних трикутників, що розташовані на висоті слайсу P'.

Подавши меш у вигляді списку граней з подвійними зв'язками (наприклад, тривимірний варіант РСПЗ) або іншої структури даних, заснованої на концепції напівребра (*англ.* half-edge), можна використати інформацію про суміжність для ітеративного переходу до сусідніх трикутників у напрямку зростання координати висоти. Детальне пояснення та огляд таких структур даних можна зайти в роботі (Yin & Goh, 2019). В такий спосіб будується локальна ділянка многокутника P',

проєкція якої на площину XY належить δ – околу вершини многокутника P. Шуканою точкою-відповідником обирається точка побудованої ділянки P', що є найближчою до вершини многокутника P, з якої було розпочато обхід суміжних трикутників.

Не обмежуючи загальності, вважатимемо, що для кожного простого многокутника з набору $P_1, P_2, \dots, P_{k_p}$ було визначено пару стартових точок. В протилежному випадку многокутники, для яких стартову точку не знайдено, вважаються новими, і на наступних етапах роботи алгоритму можуть бути проігноровані. З врахуванням цього, результатом першого кроку алгоритму є набір точок $p_1^{s'}, p_2^{s'}, \dots, p_k^{s'}$, що належать многокутнику P', які вважаються відповідниками набору стартових вершин многокутника P.

Для реалізації другого етапу виконаємо обхід многокутників $P_i, i = \overline{1, k_P}$. Для кожного з цих многокутників застосуємо підхід, аналогічний алгоритмічній техніці двох вказівників. Для цього введемо наступні допоміжні змінні: i та i' – індекси поточного ребра многокутників P та P' відповідно, а також t та t' – барицентричні координати, що вказують на поточну позицію на ребрах i та i' відповідно. Зокрема, $t = 0$ позначає початок ребра i, а $t = 1$ – його кінець.

Обхід кожної пари многокутників розпочинається з стартової вершини p^s та її відповідника $p^{s'}$. Індекс i ініціалізується порядковим номером ребра, що виходить із вершини p^s, а координата t – значенням 0. Індекс i' ініціалізується порядковим номером ребра, якому належить $p^{s'}$, але не є його кінцевою вершиною. Параметру t' присвоюється координата точки $p^{s'}$ на цьому ребрі в барицентричній системі координат.

На кожній ітерації циклу розглядається параметричний відрізок $P_i[t, 1]$, початковою точкою якого є $(1 - t)p_i + tp_{i+1}$, а кінцевою є вершина p_{i+1}, що є наступною після p_i вершиною многокутника P згідно з обраним порядком обходу. Для цього відрізка визначається його перетин зі стадіоном $S_\delta(P'_{i'}[t', 1])$ радіусу δ, побудованим для ребра многокутника P' з індексом i'. Перетин на

початковій точці параметричного відрізка враховується лише в тому випадку, якщо решта відрізка знаходиться за межами стадіону. Можливі наступні випадки:

Випадок 1. Якщо $P_i[t,1]$ не перетинає $S_\delta(P'_{i'}[t',1])$ в жодній точці, процедура зупиняється з результатом $false$.

Випадок 2. $P_i[t,1]$ перетинає $S_\delta(P'_{i'}[t',1])$ лише в своїй початковій точці (рисунок 4.4 – А). В такому разі перейдемо до наступного ребра многокутника P'. Цю процедуру продовжуємо, поки перетин $P_i[t,1]$ з новим стадіоном продовжує знаходитись в початковій точці відрізка $P_i[t,1]$, та з цієї точки є видимим відрізок $P'_{i'}[t',1]$, де множину перешкод складають попередньо відвідані при обробці цього випадку ребра P'. Якщо в процесі обробки цього випадку буде здійснено повний обхід P', процедура зупиняється з результатом $true$.

Випадок 3. Якщо $P_i[t,1]$ повністю знаходиться всередині $S_\delta(P'_{i'}[t',1])$ (рисунок 4.4 – В), тоді визначимо точку на відрізку $P'_{i'}[t',1]$, що є найближчою до кінця відрізку $P_i[t,1]$. Переходимо до наступного ребра многокутника P, збільшивши індекс i на одиницю, та присвоюємо змінній t значення 0. Змінній t' присвоюємо барицентричну координату знайденої точки на i'-му ребрі многокутника P'. У випадку, якщо знайдена точка є кінцевою вершиною цього ребра, переходимо до наступного ребра P'.

Випадок 4. $P_i[t,1]$ перетинає $S_\delta(P'_{i'}[t',1])$ в одній точці (рисунок 4.4 – С). Цей випадок обробляється схожим чином. Визначаємо точку на відрізку $P'_{i'}[t',1]$, що є найближчою до точки перетину $P_i[t,1]$ та $S_\delta(P'_{i'}[t',1])$. Змінним t та t' присвоюємо барицентричну координати точки перетину на i-му ребрі многокутника P, ребрі многокутника i'-му ребрі многокутника P' відповідно. В разі необхідності переходимо до наступних ребер многокутників.

В процесі виконання обходу, точка $(1-t)p_i + tp_{i+1}$ завжди міститься в середині $S_\delta(P'_{i'}[t',1])$, адже на початку циклу відстань між стартовою вершиною

та її відповідником не перевищує δ за побудовою, а в разі виходу за межі області δ – близькості алгоритм зупиниться за рахунок *випадку 1*. Тому, з врахуванням опуклості $S_\delta(P'_{i'}[t',1])$, випадок перетину $P_i[t,1]$ та $S_\delta(P'_{i'}[t',1])$ в більш ніж одній точці з ненульовою барицентричною координатою є неможливим.

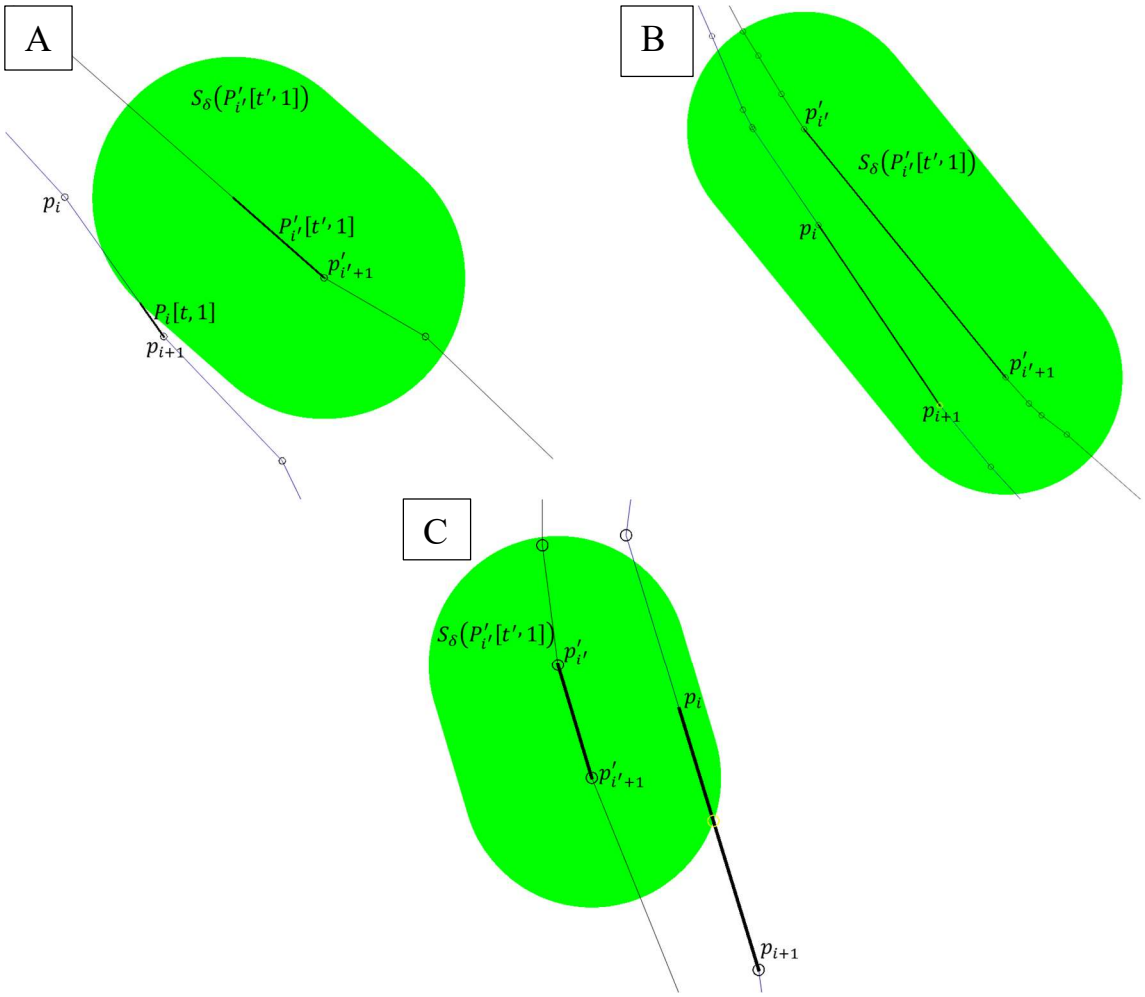

Рисунок 4.4. Можливі конфігурації перетину $P_i[t,1]$ та $S_\delta(P'_{i'}[t',1])$

Обхід зупиняється, коли досягнута стартова точка хоча б одного з многокутників. Якщо при цьому стартова точка іншого многокутника не була досягнута, необхідно додатково перевірити, чи не порушує залишок цього многокутника критерій близькості з підрозділу 4.3.1. Для цього можна застосувати процедуру, аналогічну до обробки *випадку 2* основного обходу.

Успішне завершення обходу (результат $true$) означає виконання умови $P \subseteq \bigcup_\delta P'$. Після цього виконаємо обернену перевірку, змінивши місцями вхідні

многокутники, щоб перевірити умову $P' \subseteq \bigcup_\delta P$. Виконання обох цих умов свідчить про виконання нерівності $D_H(P, P') \leq \delta$, з додатковим обмеженням на «повторюваність» форми многокутників.

Наведемо псевдокод процедури, що виконує описаний вище обхід многокутників. В псевдокоді оператор $|\cdot|$ використовується для позначення довжини відрізка або кількості вершин многокутника, в залежності від контексту. Функція $intersect$ знаходить перетин відрізка та стадіону, або повертає значення NaN (*від англ. Not a number*) якщо перетину немає. Також припускаємо, що многокутники задані циклічно впорядкованою послідовністю своїх вершин, індексація яких починається з нуля. Наприклад, многокутник P з n_p вершин задається послідовністю $p_0, p_1, \ldots, p_{n_p-1}$, та $p_{n_p} \equiv p_0$.

***Алгоритм 4.4*: Обхід P відповідно до $\bigcup_\delta P'$**
Вхід: *P, P' - многокутники; i – індекс вихідного ребра вершини p_i^s; i' - індекс ребра P', якому належить точка $p^{s'}$, але не є його кінцевою вершиною; t'-барицентрична координата $p^{s'}$ на ребрі i'; δ – точність перевірки*
Вихід: *$True$ в разі успішного завершення обходу, $False$ – інакше*

```
1   t ← 0
2   while не виконано повний обхід хоча б одного з многокутників do
    begin
3       t_seg ← intersect(P_i[t, 1], S_δ(P'_i'[t', 1])) // t_seg - барицентрична
        координата точки перетину на відрізку P_i[t, 1]
4       if t_seg is NaN then
5           return False
6       else if t_seg == 0 then
7           do
8               i' ← (i' + 1) mod |P'|
9               t' ← 0
10          while не виконано повний обхід P' and intersect(P_i[t, 1],
            S_δ(P'_{i'+1}[0,1])) == 0 and P'_{i'+1}[0,1] є видимим з (1 −
            t)p_i + tp_{i+1}
11          Continue
12      else if t_seg == 1 then
13          i ← (i + 1) mod |P|
14          t ← 0
15      else
```

16 $t \leftarrow t + \frac{t_{seg}|P_i[t,1]|}{|(p_i,p_{i+1})|}$ // Трансформуємо барицентричну координату на відрізку $P_i[t,1]$ в барицентричну координату на відрізку p_i, p_{i+1}

17 **end**

18 $t_{stad} \leftarrow nearest\_point((1-t)p_i + tp_{i+1},\ P'_{i'}[t',1])$ // барицентрична координата точки на відрізку $P'_{i'}[t',1]$, що є найближчою до точки $(1-t)p_i\ +\ tp_{i+1}$, з врахуванням змін в рядках 12-17

19 **if** $t_{stad} == 1$ **then**

20 $i' \leftarrow (i'+1)\ mod\ |P'|$

21 $t' \leftarrow 0$

22 **else**

23 $t' \leftarrow t' + \frac{t_{stad}\left|P'_{i'}[t',1]\right|}{\left|\left(p'_{i'},p'_{i'+1}\right)\right|}$ // Трансформуємо барицентричну координату на відрізку $P'_{i'}[t',1]$ в барицентричну координату на відрізку $p'_{i'}, p'_{i'+1}$

24 **End**

25 **End**

26 **return** $True$

Теорема 4.4. Для двох простих многокутників P та P', що мають відповідно n_P та $n_{P'}$ вершин, перевірка виконання умови $D_H(P,P') \leq \delta$ із додатковим обмеженням на «повторюваність» форми многокутників може бути здійснена за допомогою алгоритму 4.4 за час, що не перевищує $O(n_P + n_{P'})$ в найгіршому випадку.

Доведення. Розглянемо, як змінюється стан змінних i, i', t та t' під час кожної ітерації процедури обходу многокутників.

Випадок 1 є тривіальним, оскільки за нього обхід зупиняється. У другому випадку перехід до наступного ребра многокутника P' відбувається принаймні один раз. Для перевірки видимості ребра $i'+1$ визначається орієнтація трійки точок $< (1-t)p_i + tp_{i+1}, p'_{i'+1}, p'_{i'+2} >$. Якщо орієнтація цієї трійки узгоджується з раніше визначеними орієнтаціями для ребер з множини перешкод, то це ребро є видимим. Оскільки перевірка для одного ребра займає $O(1)$ часу, вона не впливає на загальну складність алгоритму.

Варіанти, за яких випадки 2 та 3 завершуються з результатом $false$, також не впливають на оцінку складності. Тому необхідно дослідити лише варіанти випадків 3 та 4, за яких обхід продовжується. Зазначимо, що в обох випадках алгоритм переходить до наступного ребра хоча б одного з многокутників: у випадку 3 індекс ребра першого многокутника i завжди збільшується на одиницю, а у випадку 4 хоч і не вимагається, щоб точка перетину $P_i[t, 1]$ та $S_\delta(P'_{i'})$, або найближча до неї точка на ребрі $P'_{i'}$ були кінцями ребер, проте в іншому разі на наступній ітерації алгоритм не знайде жодного перетину і зупиниться через випадок 1.

Таким чином, кожна ітерація обходу виконує перехід до наступного ребра хоча б одного з многокутників. Всього може бути відвідано n_P та $n_{P'}$ ребер, тому процедура обходу виконується за час, що не перевищує $O(n_P + n_{P'})$.

Додаткова перевірка впорядкованості послідовності точок за полярним кутом відносно іншої точки виконується за лінійний час. Отже, перевірка умови $\overrightarrow{D_H}(P, P')$ виконується за час, що лінійно залежить від загальної кількості вершин у многокутниках. Аналогічним чином виконується перевірка умови $\overrightarrow{D_H}(P', P)$, тому загальний час роботи процедури також складає $O(n_P + n_{P'})$. ■

Наслідок. Оскільки плоский многокутник може бути представлений у вигляді набору простих многокутників, що задають його зовнішню та внутрішню границі, то алгоритм 4.4 може бути окремо застосованим до кожного з цих многокутників окремо. Тому, якщо множина початкових вершин є відомою, описаний вище алгоритм може бути застосованим для порівняння довільних плоских многокутників за лінійний час.

Теорема 4.5. Для двох простих многокутників P та P', що мають відповідно n_P та $n_{P'}$ вершин, перевірка виконання умови $D_H(P, P') \leq \delta$ із додатковим обмеженням на «повторюваність» форми многокутників може бути здійснена за допомогою алгоритму 4.4 з використанням $O(1)$ додаткової пам'яті в найгіршому випадку.

Доведення. В процесі роботи алгоритм підтримує фіксований набір скалярних змінних: індекси поточних ребер i та i', барицентричні координати t та t', та фіксовану кількість допоміжних змінних. Перевірка видимості (10й рядок псевдокоду) вимагає не більше $O(1)$ додаткової пам'яті, оскільки вона виконується шляхом перевірки узгодженості орієнтації ребер. Алгоритм реалізований ітеративно, без виконання рекурсивних викликів. Кожна ітерація циклу працює лише з поточними ребрами многокутників P та P', та областю δ – близькості, що параметрично представлена відповідним стадіоном, та використовується для аналітичного пошуку відстані та перевірки перетину. Таким чином, жодна інформація про вже пройдені ребра не накопичується в процесі роботи алгоритму. З наведеного випливає, що незалежно від кількості вершин n_P та $n_{P'}$, алгоритм використовує лише сталу кількість додаткової пам'яті для зберігання проміжних значень і керуючих змінних. Отже, просторова складність алгоритму 4.4 одночасного обходу двох простих многокутників дорівнює $O(1)$. ■

4.3.3 Відновлення декомпозиції подібних многокутників

Теореми 4.4 та 4.5 у сукупності встановлюють обчислювальну ефективність запропонованої в розділі 4.3.2 процедури перевірки подібності многокутників. Було показано, що час роботи алгоритму лінійно залежить від сумарної кількості вершин вхідних многокутників. Така оцінка складності є асимптотично оптимальною для процедури, що виконує обхід многокутника, оскільки кожне ребро має бути відвідане щонайменше один раз. Водночас, алгоритм 4.4 не потребує побудови допоміжних структур даних і використовує лише сталу кількість додаткової пам'яті, що робить його придатним для застосування в задачах обчислювальної геометрії з великими об'ємами вхідних даних та жорсткими обмеженнями на обчислювальні ресурси. Поєднання лінійної часової складності та константних просторових витрат дозволяє

розглядати запропонований підхід як ефективний базовий примітив для оптимізації алгоритмів.

В цьому розділі буде показано, як алгоритм 4.4 може бути застосований, для відновлення декомпозиції многокутника, якщо існує подібний многокутник з раніше побудованою декомпозицією. Запропонований підхід може бути застосованим до різних типів декомпозиції (на монотонні многокутники, на опуклі многокутники, тріангуляції тощо), однак вимагає, щоб декомпозиція не додавала нових вершин в многокутник.

Задачу відновлення декомпозиції многокутника будемо розглядати як процес інкрементальногї побудови розбиття многокутника. Для кожної хорди існуючого розбиття будемо визначати початкову та кінцеву точку подібної хорди на новому многокутнику, та перевіряти, чи не утворює така хорда перетин з контуром многокутника або іншими наявними хордами. Хорди, для яких буде знайдено такий перетин ігноруються, але процес реконструкції буде продовжуватись далі, оскільки можливі випадки, за яких локальна перебудова декомпозиції деякої області многокутника може бути виконана швидше за повну повторну декомпозицію.

Ключовою алгоритмічною проблемою такого підходу є побудова ефективного способу визначення перетину чергової потенційної хорди, з іншими хордами та ребрами многокутника. Хоча визначення точки перетину між двома відрізками є відносно простою задачею, вона суттєво ускладнюється зі зростанням кількості відрізків або за наявності динамічних змін у сцені, таких як додавання, видалення чи рух відрізків. Існують численні дослідження, зосереджені на окремих варіантах споріднених задач. Одним із класичних прикладів є алгоритм (Bentley & Ottman, 1979), який знаходить всі пари перетинів між n відрізками за час $O((n+k)\log n)$, де k – фактична кількість утворених перетинів. В порівнянні з тривіальним перебором всіх можливих пар, що має складність $O(n^2)$, такий підхід забезпечує суттєве прискорення у випадку, якщо значення k достатньо мале. В роботі (De Berg et al., 2017) досліджується пошук

перетинів між відрізками виключно в межах заданої області. Крім того, в (Buchin et al., 2022) запропоновано структуру даних, яка ефективно підраховує точки або відрізки, що знаходяться в середині простого многокутника, що є видимими з заданої точки або відрізка. Також, окремі дослідження присвячені спеціальним випадкам, у яких відрізки завжди є паралельними до однієї з координатних осей. Зокрема, (Imai & Asano, 1984) запропонували алгоритм, який знаходить перетини в динамічній множині вирівняних по осям відрізків, що може оновлюватися шляхом додавання або вилучення, але не здатний одночасно обробляти обидва типи запитів. Іншу структуру даних, яка дозволяє повідомляти всі перетини відрізків, вирівняних по осям в межах заданого прямокутника та водночас підтримує додавання нових відрізків, запропоновано в роботі (Petrova & Tarjan, 2016).

В найбільш загальній постановці, задачу можна сформулювати таким чином. Спочатку задано простий многокутник P, який складається з n вершин $p_1, p_2, \dots, p_n$. Підрозбиття многокутника будується інкрементально, шляхом додавання хорд, що сполучають його вершини. На кожному кроці надходить нова хорда – кандидат у вигляді пари індексів вершин многокутника (i, j). Якщо відрізок, що з'єднує вершини p_i та p_j, повністю лежить в середині многокутника P і не перетинає жодної з раніше доданих хорд, то хорда $p_i p_j$ додається до розбиття. Метою є побудова алгоритму, здатного ефективно підтримувати ітеративний процес побудови підрозбиття. Задачу будемо розглядати в режимі онлайн. Це означає, що окремі запити не можуть бути згруповані та оброблені спільно, а майбутні запити залишаються невідомими.

Перш ніж перейти безпосередньо до побудови алгоритму, наведемо короткий огляд структур даних, які будуть в ньому використовуватись.

Першою структурою даних є *ієрархія обмежувальних об'ємів* (*Bounding Volume Hierarchy, BVH)* – універсальна деревоподібна структура даних, у якій геометричні об'єкти зберігаються в листових вузлах, тоді як решта вузів містять деякі об'єднані обмежувальні об'єми своїх нащадків. Вперше вона була

запропонована в роботі (Rubin & Whitted, 1980). В контексті задачі підтримки розбиття, в листових вузлах *BVH* зберігатимуться ребра многокутника. З огляду на загальний характер цієї структури існує багато підходів до побудови *BVH*. Для задачі підтримки розбиття будемо використовувати *BVH* на основі *евристики площі поверхні* (*Surface Area Heuristic, SAH*), яку вперше було запропоновано в роботі (MacDonald & Booth, 1989), а ефективний алгоритм побудови наведено в (Wald, 2007). Такий вибір обґрунтовано тим, що *SAH*-евристика на практиці часто забезпечує вдалий компроміс між часом побудови структури даних та глибиною отриманого дерева.

Іншою структурою даних, що буде застосована при побудові алгоритму, є *MinMax Segment Tree*, що є різновидом дерева відрізків (De Berg et al., 2007), в якому кожен вузол одночасно зберігає як мінімальне, так і максимальне значення відповідного діапазону. З теоретичної точки зору така структура є еквівалентною підтримці двох окремих дерев відрізків – одного для мінімумів та іншого для максимумів, але на практиці використання єдиної ітеративної реалізації дерева відрізків забезпечує дещо вищу продуктивність. Покращення зумовлене зменшенням кількості кадрів стеку, що створюються в процесі виконання запиту, та кращою локальністю даних (Lin et al., 2021).

Для розуміння роботи алгоритму важливо звернути увагу на те, що початкову задачу можна розділити на дві незалежні підзадачі, що безпосередньо випливає з визначення запиту перевірки кандидата на хорду:

i. Визначення того, чи лежить задана хорда – кандидат (i, j) всередині многокутника P, та перевірка її перетину з ребрами многокутника (за винятком ребер, інцидентних вершинам p_i або p_j).
ii. Визначення того, чи перетинає задана хорда – кандидат (i, j) будь-які раніше розглянуті допустимі хорди.

Тобто, якщо задана хорда – кандидат (i, j) задовольняє обидві наведені вище умови, то пара (i, j) визначає коректну хорду, яка має бути додана до розбиття.

Звернемо увагу, що перша умова (i.) перевіряється на статичній геометрії, адже сам многокутник P жодним чином не змінюється в результаті додавання нових хорд, в той час як друга умова (ii.) вимагає динамічної перевірки з врахуванням доданих раніше хорд.

Перша умова (i.) може бути розглянута як задача трасування променів: маючи дві вершини многокутника, необхідно визначити, чи не перетинає промінь, що виходить з однієї вершини до іншої, інших ребер многокутника на своєму шляху. При цьому, необхідно додатково перевірити напрямок променя, який має бути спрямований у внутрішню область многокутника. Виконання цієї умови можна перевірити, розглянувши знак z-компоненти векторного добутку інцидентних ребер з вектором-напрямком променя та між собою. Припускаючи, що вершини многокутника є впорядкованими в порядку обходу проти годинникової стрілки, зазначену перевірку можна подати у вигляді наступного предиката:

$$GoesInside\left(\vec{v}_{in}, \vec{v}_{out}, \vec{d}\right) = \begin{cases} \vec{v}_{in} \times_z \vec{d} \geq 0 \; and \; \vec{v}_{out} \times_z \vec{d} \geq 0, if \; \vec{v}_{in} \times_z \vec{v}_{out} \geq 0 \\ \vec{v}_{in} \times_z \vec{d} \geq 0 \; or \; \vec{v}_{out} \times_z \vec{d} \geq 0, if \; \vec{v}_{in} \times_z \vec{v}_{out} < 0 \end{cases},$$

де $\vec{v}_{in} = p_i - p_{i-1}$, $\vec{v}_{out} = p_{i+1} - p_i$ та $\vec{d} = p_j - p_i$.

Задачу трасування променів можна розв’язати за допомогою ієрархії обмежувальних об’ємів на основі *SAH*-евристики, в листових вузлах якої містяться ребра P, а в решті вузлів – мінімальні орієнтовані обмежуючі паралелепіпеди. Для цього достатньо виконати обхід дерева BVH в порядку згори до низу, виконуючи перевірку перетинів із ребрами в тих листових вузлах, обмежувальні паралелепіпеди яких перетинаються променем.

Кожна хорда, для якої виконується перша умова, ділить многокутник на дві секції, що дозволяє розглядати другу умову (ii.) як інтервальну задачу. Нехай (i, j) – хорда-кандидат, а через (k, l) позначимо деяку хорду, перетин з якою необхідно перевірити. Для визначеності припустимо, що $i < j$ та $k < l$. Дві секції

многокутника P можна представити у вигляді трьох інтервалів: $[1, i-1]$, $[i, j]$ та $[j+1, n]$ (при цьому, перший та третій інтервали можуть бути порожніми). Тоді існує чотири різні конфігурації взаємного розташування хорд (i, j) та (k, l):

1. $k < l \leq i < j$
2. $i \leq k < l \leq j$
3. $k < i < l < j$
4. $i < k < j < l$

Ці чотири конфігурації проілюстровано на рисунку 4.5.

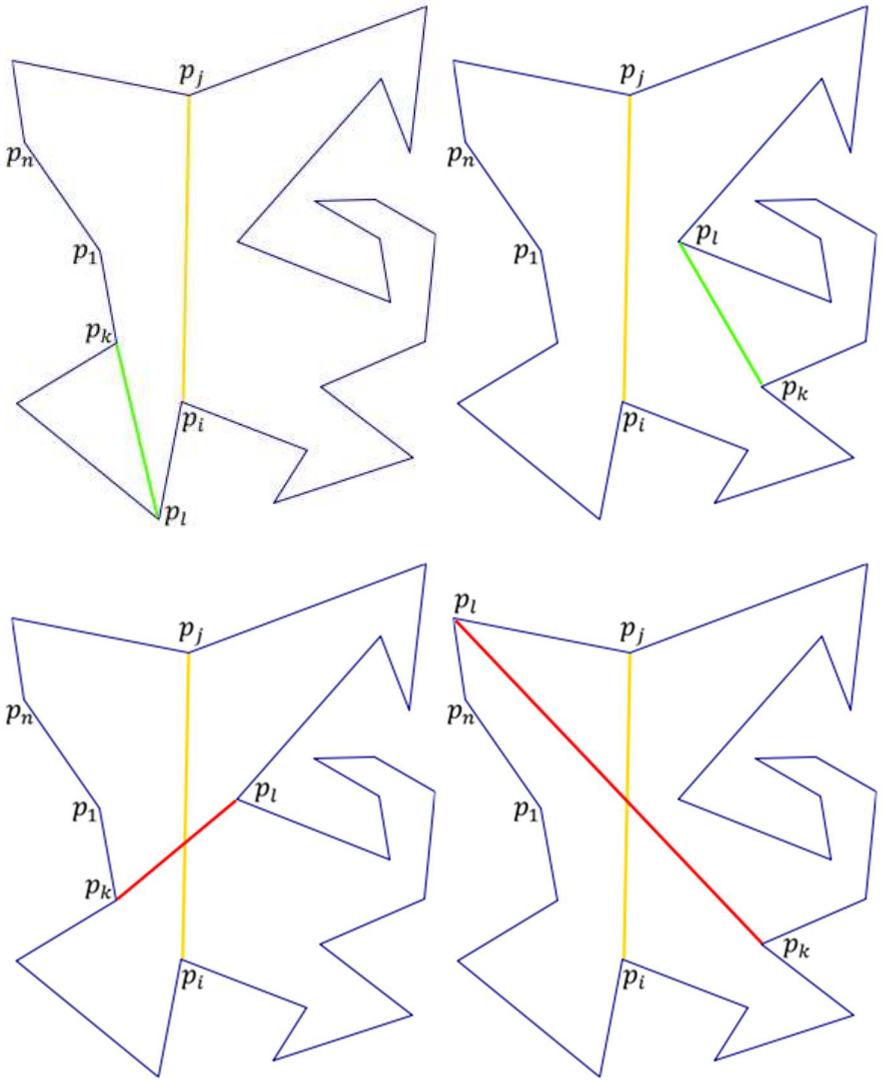

Рисунок 4.5. Чотири різні конфігурації розміщення хорд (i, j) та (k, l)

Для розв'язання поставленої задачі необхідно ефективно визначати третю та четверту конфігурації – ті, за яких відрізки $p_i p_j$ та $p_k p_l$ перетинаються. В

термінах інтервальної задачі, це випадки, коли існує ребро, що з'єднує вершину з ланцюга $p_{i+1} \dots p_{j-1}$ або з вершиною ланцюга $p_1 \dots p_{i-1}$ або з вершиною ланцюга $p_{j+1} \dots p_n$. З цього слідує, що для визначення наявності деякої хорди, яка в комбінації з (i, j) розміщена в третій та четвертій конфігурації достатньо перевірити, чи є хоч одна хорда з однією вершиною в ланцюзі $p_{i+1} \dots p_{j-1}$ та іншою вершиною в $p_1 \dots p_{i-1}$ або $p_{j+1} \dots p_n$. Для ефективної обробки масових запитів такого вигляду будемо підтримувати для кожної вершини індекс найлівішої та найправішої вершин, що сполучені хордою з даною вершиною. Тоді відповідь на такий запит зводиться до задачі пошуку мінімуму та максимуму на заданому інтервалі. Ефективно обробляти такі запити можна за допомогою структури даних *MinMax Segment Tree*, яка зберігає мінімальний та максимальний індекси вершин, з якими кожна вершина з'єднана попередніми хордами.

Формальніше, під час ініціалізації дерева, кожен вузол дерева отримує початкові значення $(+inf, -inf)$, а кожна хорда-кандидат обробляється за таким алгоритмом:

***Алгоритм 4.5*: Обробка хорди-кандидата (i, j)**
Вхід: *$tree$ – MixMax Segment Tree на інтервалі $[1, n]$; i – індекс першої вершини хорди-кандидата, j – індекс другої вершини хорди-кандидата*
Вихід: *$True$, якщо пара (i, j) задає коректну хорду, $False$ – інакше*

```
1    I ← [i + 1, j − 1]
2    I_min, I_max ← query(tree, I)
3    if I_min > i ma I_max < j do begin
4        updateMax(tree, i, j)
5        updateMin(tree, j, i)
6        return True
7    End
8    return False
```

Теорема 4.6. Динамічну перевірку перетину q хорд-кандидатів з іншими хордами в простому многокутнику P з n вершин можна виконати алгоритмом 4.5 за час, що не перевищує $O(n + q \log n)$ в найгіршому випадку.

Доведення. Попередньою обробкою алгоритму є побудова дерева відрізків для n елементів. В загальному формулюванні, асимптотична складність побудови такого дерева складає $O(n \log n)$ (De Berg et al., 2007) за рахунок сортування відрізків. Однак, в дереві відрізків для алгоритму 4.5 одиничними інтервалами є цілочислові індекси вершин. Іншими словами, множина відрізків є впорядкованою за своєю природою, тому виконання попередньої обробки полягає у побудові бінарного дерева пошуку з $O(n)$ вершинами, що вимагає $O(n)$ часу.

Обробка кожної хорди-кандидата полягає у виконанні одного запиту пошуку по дереву відрізків, та, в разі виконання умови $I_{min} > i \; ma \;\; I_{max} < j$ (3й рядок псевдокоду алгоритму 4.5), двох запитів оновлення значення в одиничному інтервалі. Запити обох видів виконуються за час $O(\log n)$, тому q таких запитів будуть оброблені за час $O(q \log n)$.

Об'єднавши витрати на побудову дерева відрізків та обробку q хорд-кандидатів, отримуємо загальну асимптотичну складність $O(n + q \log n)$. ■

Побудована процедура відновлення підрозбиття, в поєднанні з алгоритмом 4.4 перевірки подібності двох многокутників дозволяє виконувати оптимізацію низки геометричних задач, що використовують техніку декомпозиції многокутника на менші многокутники з наперед визначеними властивостями (монотонні, опуклі тощо).

Висновки до розділу 4

В цьому розділі було проаналізовану реалізацію комплексу задач планування АВ в межах запропонованої в третьому розділі архітектури МЄАС. Отримані результати підтверджують ефективність запропонованих у розділі 3 архітектурних принципів та концепцій, зокрема використання МСД та неявного зведення даних.

Показано, що існуючі підходи до розв'язання комплексу задач процесу планування АВ природно інтегруються в запропоновану МЄАС. Продемонстровано, що використання МСД дозволяє повторно застосовувати обчислювально важкі для побудови структури даних (kD-дерева, дерева октантів, воксельні сітки тощо), створені під час розв'язання попередніх задач робочого процесу. Досліджено способи оптимізації існуючих алгоритмів інструментами МЄАС. Зокрема, показано, що застосування принципу неявного зведення даних дозволяє прискорити алгоритм слайсингу та отримати лінійну обчислювальну складність за умови наявності воксельної сітки в МСД.

Обґрунтовано доцільність використання графу Ріба як компактної репрезентації топології слайсів моделі. Досліджено можливості використання графу Ріба як інструмент прискорення розв'язання деяких задач обчислювальної геометрії в межах запропонованої архітектури МЄАС. Зокрема, розроблено алгоритм побудови декомпозиції плоского многокутника на y-монотонні многокутники з одночасною їх тріангуляцією. Встановлено, що обчислювальна складність запропонованого алгоритму складає $O(n)$, де n – кількість вершин многокутника, що є ефективнішим за існуючі алгоритми тріангуляції.

Розроблено новий підхід до врахування подібності сусідніх слайсів моделі, як оптимізацію процесу побудові траєкторії друку. Введено поняття області δ – близькості, на основі якого розроблено алгоритм перевірки подібності многокутників, який працює за лінійний час та використовує константний обсяг додаткової пам'яті. Запропоновано алгоритм відновлення декомпозиції многокутника на основі існуючої декомпозиції для подібного многокутника. В сукупності, ці два алгоритми дозволяють уникати повторну побудову декомпозиції або тріангуляції для однакових або дуже схожих контурів моделі, що забезпечує економію часу при обробці великої кількості послідовних шарів.

Розглянуті рішення підкреслюють гнучкість і масштабованість розробленої архітектури МЄАС. Зазначені в розділі 3 підходи до ізольованості структур даних в МСД та використання принципу неявного зведення даних

продемонстровано на практиці. Отримані результати доводять, що розширена МЄАС ефективно координує роботу різнорідних алгоритмів процесу планування АВ, мінімізуючи надлишкові обчислення та забезпечуючи узгодженість даних під час виконання робочого процесу.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА РЕАЛІЗАЦІЯ

В розділі 3 даної роботи було представлено архітектуру моделі єдиного алгоритмічного середовища (МЄАС), яка була розширена та адаптована до специфіки досліджуваної проблемної області – процесу планування адитивного виробництва. В розділі 4 показано, яким чином існуючі підходи до розв'язання комплексу задач адитивного виробництва можуть бути ефективно реалізовані в рамках розробленої архітектури МЄАС. Запропоновано нові підходи до розв'язання деяких задач або оптимізації розглянутих алгоритмів. В цьому розділі дослідження будуть розглянуті основні аспекти практичної реалізації запропонованої архітектури МЄАС з врахуванням вимог до ефективності та масштабованості програмної системи, а також досліджено ефективність запропонованих в розділі 4 алгоритмів та методів оптимізації, в порівнянні з існуючими підходами.

5.1 Реалізація та тестування алгоритмів на основі графу Ріба для плоского многокутника

5.1.1 Побудова графу Ріба

Алгоритм побудови графу Ріба для плоского многокутника, ідею якого вперше було описано в роботі (Tereshchenko & Prishlyak, 2024), та деталізовано в підрозділі 4.2.2 цього дослідження було реалізовано мовою програмування *C++*. В якості структури даних, що підтримує статус процедури плоского замітання було використано контейнер *std::set* зі стандартної бібліотеки шаблонів мови *C++* з кастомним компаратором, що порівнює x координати точок перетинів двох відрізків (ребер многокутника) з поточною горизонтальною прямою. Також, було застосовано бібліотеку *Mathematics*, яка є складовою частиною геометричного фреймворку Geometric Tools (2025), що містить реалізацію базових геометричних примітивів та операцій над ними. Графічний інтерфейс додатку було розроблено на основі фреймворку Qt (The Qt Company, 2024).

На сьогодні відсутні загальноприйняті набори тестових многокутників, які б знаходились у відкритому доступі, містили многокутники з різною кількістю вершин (від десятків та сотень до десятків мільйонів вершин), в тому числі з дірками. Як наслідок, в межах цієї роботи для тестування та перевірки ефективності алгоритму побудови графу Ріба було побудовано власний набір тестових многокутників, які генерувались наступним чином:

1. Додаток Random Polygon Generator (Auer et al., 2020), що реалізує алгоритми генерування многокутників, описані в роботі (Auer & Held, 1996).
2. Алгоритм генерування простих многокутників на заданому наборі точок, описаний в роботі (Osiponok & Tereshchenko, 2019).
3. Шаблонний многокутник, утворений шляхом віднімання шестикутних дірок з середини ромбовидної основи.

Зазначеними підходами було отримано набір різних плоских многокутників, кількість вершин в яких варіюється від 100 до 10 000 000.

На кожному тестовому многокутнику алгоритм побудови ГР з підрозділу 4.2.2 було виконано 1000 разів, з метою мінімізації впливу випадкових зовнішніх факторів (фонова активність сервісів операційної системи тощо) на виміри. Середнє значення часу виконання було обчислене для кожного многокутника, що дозволило отримати більш стабільні та надійні результати, усуваючи випадкові варіації, викликані зовнішніми чинниками. На основі отриманих даних було сформовано графік залежності часу побудови ГР від розміру вхідного многокутника, що представлено на рисунку 5.1.

Результати проведеного тестування підтверджують, що запропонований алгоритм та розроблена реалізація можуть бути ефективно застосованими на практиці для розв’язку задач в реальному часі. Так, на многокутниках, що складаються з 10^6 вершин, алгоритм в середньому виконував свою задачу за 600 мс, а на многокутниках з великою деталізацією контуру розміром 10^7 вершин

побудова виконувалась за 3.7 с в середньому. Приклад побудованого ГР для многокутника з дірками, що складається з 33х вершин представлено на рисунку 5.2.

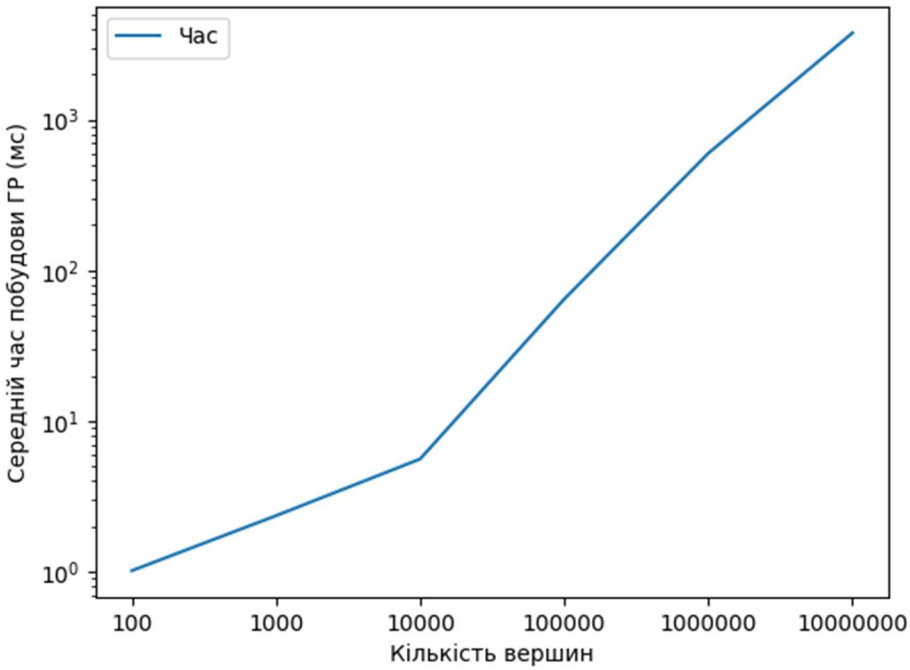

Рисунок 5.1. Залежність часу побудови ГР від кількості вершин многокутника

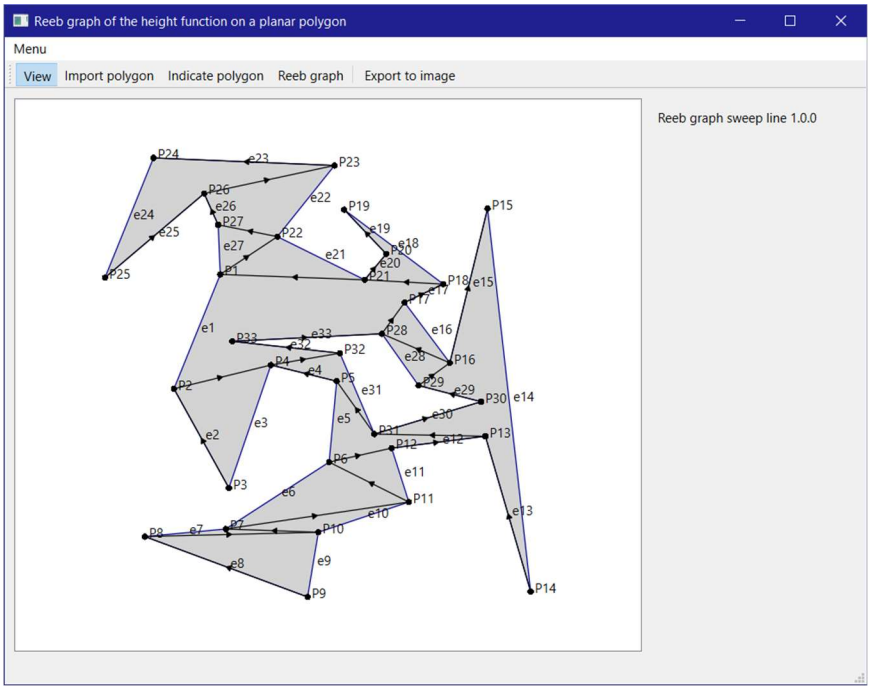

Рисунок 5.2. Побудований ГР для многокутника з дірками

5.1.2 Побудова y-монотонної декомпозиції та тріангуляції

Тестування алгоритму 4.3 декомпозиції плоского многокутника на набір y-монотонних многокутників з одночасною побудовою їх тріангуляції було виконано на наборі тестових многокутників, побудову якого було описано в підрозділі 4.1.1. Приклади побудованої декомпозиції многокутника на y-монотонні частини та їх тріангуляцію зображено на рисунках 5.3 (А) та 5.3 (Б) відповідно.

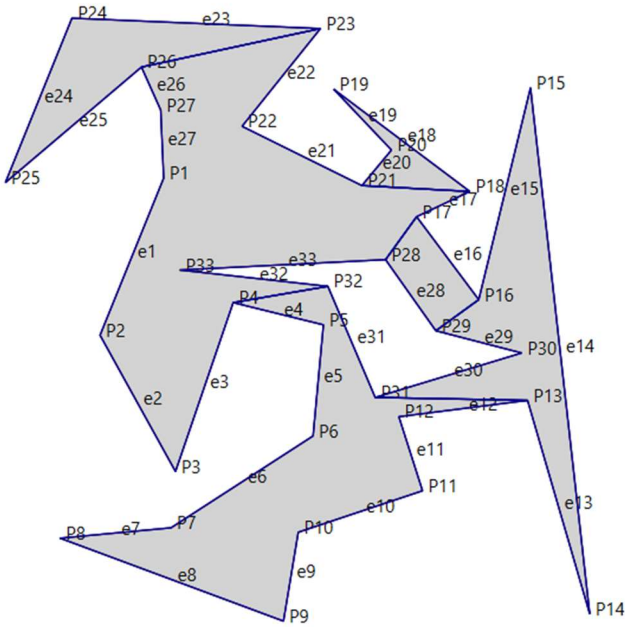

Рисунок 5.3 (А). Декомпозиція многокутника на y-монотонні частини

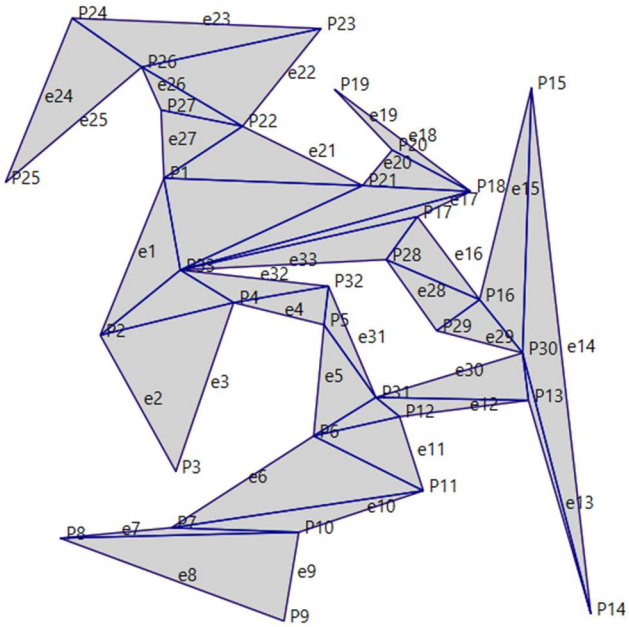

Рисунок 5.3 (Б). Тріангуляція y-монотонних многокутників

Для порівняння ефективності розробленої імплементації алгоритму з існуючими альтернативними підходами були обрані найбільш вживані бібліотеки, що реалізують той чи інший алгоритм тріангуляції плоского многокутника (як з дірками, так і без них), а саме:

- Бібліотека CGAL (The CGAL Project, 2024), що містить два оператори (CGAL::Constrained_triangulation_2 та

CGAL::Constrained_Delaunay_triangulation_2) побудови тріангуляції з обмеженнями, яким в якості обмежень можна задати ребра многокутника.

- Бібліотека Geometric Tools Mathematics (Geometric Tools, 2025), що містить оператор побудови тріангуляції ОТД
- Бібліотека earcut.hpp (Mapbox, n.d.), що реалізує модифікацію квадратичного алгоритму тріангуляції earcut (Eberly, 2024), оптимізованого хешуванням кривих z-порядку та розширеного для підтримки многокутників з дірками.
- Бібліотека PolyPartition (Fratric, 2021), що реалізує алгоритм тріангуляції за допомогою розбиття на y-монотонні многокутники, який описано в (De Berg et al., 2008).

Порівняння швидкості роботи вищезазначених бібліотек було проведено як з повною тріангуляцією з використанням графу Ріба (тобто час побудови графа Ріба включається в загальний час тріангуляції), так і окремо. Останній випадок є цікавим, оскільки він дозволяє оцінити ефективність використання ГР в МЄАС, імітуючи наявність попередньо побудованого графу в МСД. З метою забезпечення точності тестування, кожен з зазначених алгоритмів 100 разів виконувався на кожному з тестових многокутників, і до уваги брався середній час виконання алгоритму.

За результатами тестування, найшвидшим виявився запропонований в цій роботі підхід (алгоритм 4.3), за умови попередньої побудови графа Ріба (випадок застосування МЄАС з МСД). В середньому, цей підхід є в 2-3 рази швидшим за реалізацію алгоритму тріангуляції шляхом розбиття многокутника на y-монотонні частини з бібліотеки PolyPartition, та в 5-6 разів швидше за алгоритми тріангуляції з бібліотеки CGAL. Однак, якщо врахувати час побудови графу Ріба, то PolyPartition працював на 20% швидше на всіх тестах, окрім найбільшого многокутника – в цьому випадку побудова графа Ріба та тріангуляція за його допомогою зайняла на 22% менше часу, в порівнянні з PolyPartition. Інші розглянуті бібліотеки значно поступаються в швидкодії як підходу,

запропонованому в цій статті, так і бібліотекам CGAL та PolyPartition. Графік залежності часу виконання тріангуляції алгоритмом на основі графу Ріба та зазначеними вище бібліотеки зображено на рисунку 5.4.

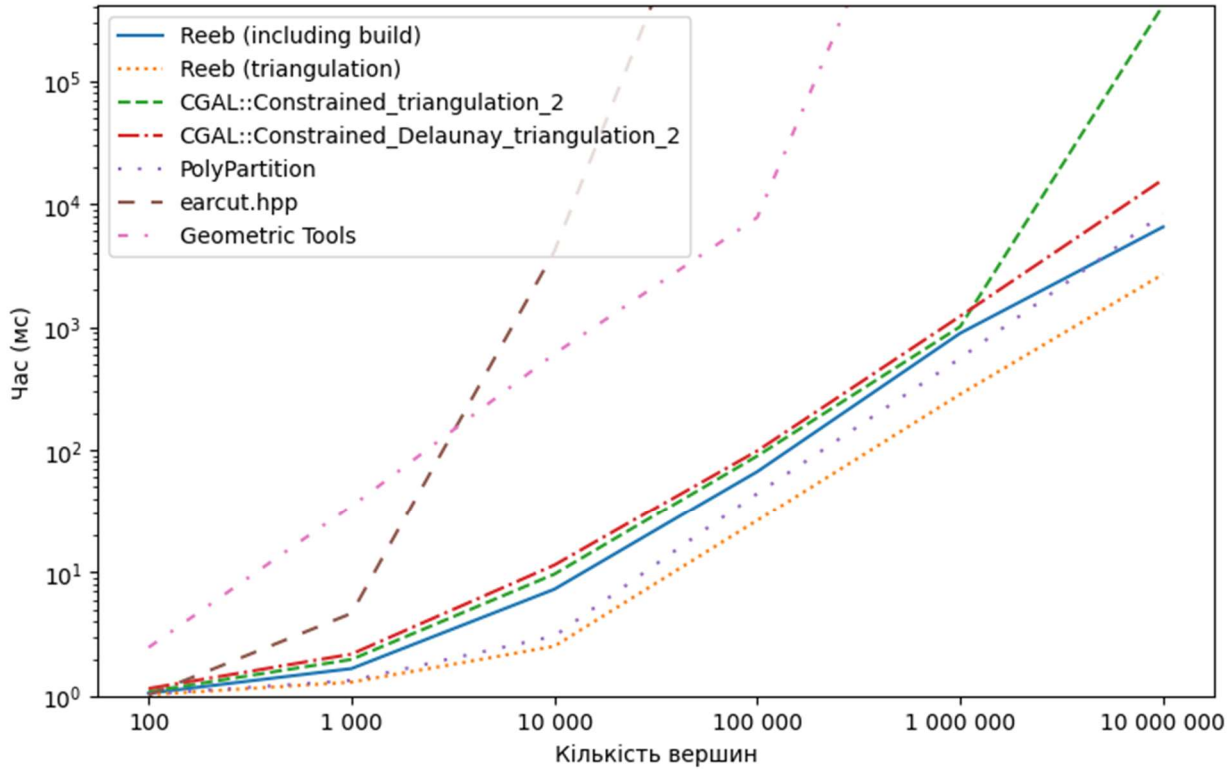

Рисунок 5.4. Графік залежності часу тріангуляції від розміру многокутника

Таким чином, алгоритм побудови тріангуляції на основі ГР є в рази швидшим за аналоги за умови наявності ГР в МСД, що підтверджує ефективність застосування ГР як одного з інструментів при побудові геометричних алгоритмів, зокрема в задачах планування АВ. Також, тріангуляція з врахуванням часу побудови ГР лише трохи поступається найшвидшому з існуючих підходів, що були розглянуті в цій роботі - класичному алгоритму тріангуляції з використанням y-монотонності многокутника з роботи (De Berg et al., 2008), реалізованому в бібліотеці PolyPartition. Це пояснюється додатковими витратами на підтримку y-монотонної декомпозиції у вигляді окремої структури даних. Однак, навіть в цьому випадку, зі збільшенням кількості вершин многокутника

різниця в швидкості роботи двох алгоритмів зменшується, а на великих многокутниках з 10^7 вершин розроблений алгоритм виявився швидшим за PolyPartition.

5.2 Реалізація та тестування алгоритму порівняння многокутників з заданою точністю

Запропоновану в підрозділі 4.3.2 процедуру визначення порівняння многокутників з заданою точністю δ, що враховує форму многокутників, було реалізовано мовою програмування C++ з використанням бібліотек Geometric Tools (2025) та BVH (Pérard-Gayot, 2025). Тестування алгоритму виконувалось на популярних тестових 3D-моделей: Stanford Bunny, Stanford Armadillo та Utah Teapot з відкритого дата сету (Jacobson, 2017), з різною товщиною слайсу та різним значенням параметру δ. Для зручності, зазначені тестові моделі було масштабовано до розмірів $19.97 \times 16.28 \times 20$, $110.05 \times 131.1 \times 100$ та $80.43 \times 39.38 \times 50$ відповідно. Порогове значення коефіцієнта Жаккара було зафіксовано на рівні 0.8.

Для кожного тесту було підраховано кількість повністю близьких слайсів, тобто таких, що для всіх поліліній алгоритму вдалось визначити відповідний полілайн на наступному слайсі, та кількість частково близьких слайсів, для яких алгоритмом було знайдено відповідник хоча б для одного полілайну.

Результати тестування демонструють, що для більшості слайсів алгоритму вдалося визначити відповідні полілайни на наступному слайсі. В залежності від складності геометрії моделі, товщини слайсу та значення параметру δ, цей показник варіюється від 55.825% при $\delta = 0.15$ та 67% при $\delta = 0.3$ для моделі Stanford Armadillo, до $\sim 97\%$ для простішої моделі Stanford Bunny.

Якщо ж врахувати слайси, для полілайнів яких відповідність було побудовано частково, загальний відсоток успішно оброблених слайсів для всіх тестів лежить в межах від 96.72% для моделі Utah Teapot при $\delta = 0.15$ та

товщині слайсу 0.02, до 99.44% для цієї ж моделі при $\delta = 0.3$ та товщині слайсу 0.005. Повністю результати тестування представлені в таблиці 5.1.

Назва моделі	Товщина слайсу	Значення δ	Загальна кількість слайсів	Кількість повністю близьких слайсів	Кількість частково близьких слайсів
Bunny	0.01	0.3	2000	1950	29
Bunny	0.01	0.15	2000	1943	35
Bunny	0.005	0.15	4000	3939	37
Bunny	0.005	0.05	4000	3878	65
Teapot	0.02	0.3	2500	2190	254
Teapot	0.02	0.15	2500	2129	289
Teapot	0.005	0.3	10000	9685	259
Teapot	0.005	0.15	10000	9571	367
Armadillo	0.025	0.3	4000	2680	1252
Armadillo	0.025	0.15	4000	2233	1664
Armadillo	0.01	0.3	10000	8245	1687
Armadillo	0.01	0.15	10000	7935	1985

Таблиця 5.1. Результати тестування алгоритму порівняння многокутників

Аналіз суміжних слайсів, серед яких алгоритм не ідентифікував відповідність між полілайнами, показує, що такі ситуації переважно характерні для ділянок моделі з різкими перепадами кривизни та подібними геометричними

особливостями. Керувати точністю обробки таких областей можна шляхом зміни значення параметру δ.

Розглянемо, наприклад, модель Stanford Bunny з товщиною слайсу 0.005 та значеннями $\delta = 0.05$ та $\delta = 0.15$ (рисунок 5.5). В обох випадках, виділені червоним ділянки, що відповідають слайсам, для яких алгоритму не вдалось визначити відповідні полілайни, знаходяться в місцях екстремумів (локальних піків) геометрії та на ділянках, де спостерігається різкий перепад кривизни. Проте, товщина таких ділянок при більшому значенні δ є помітно меншою.

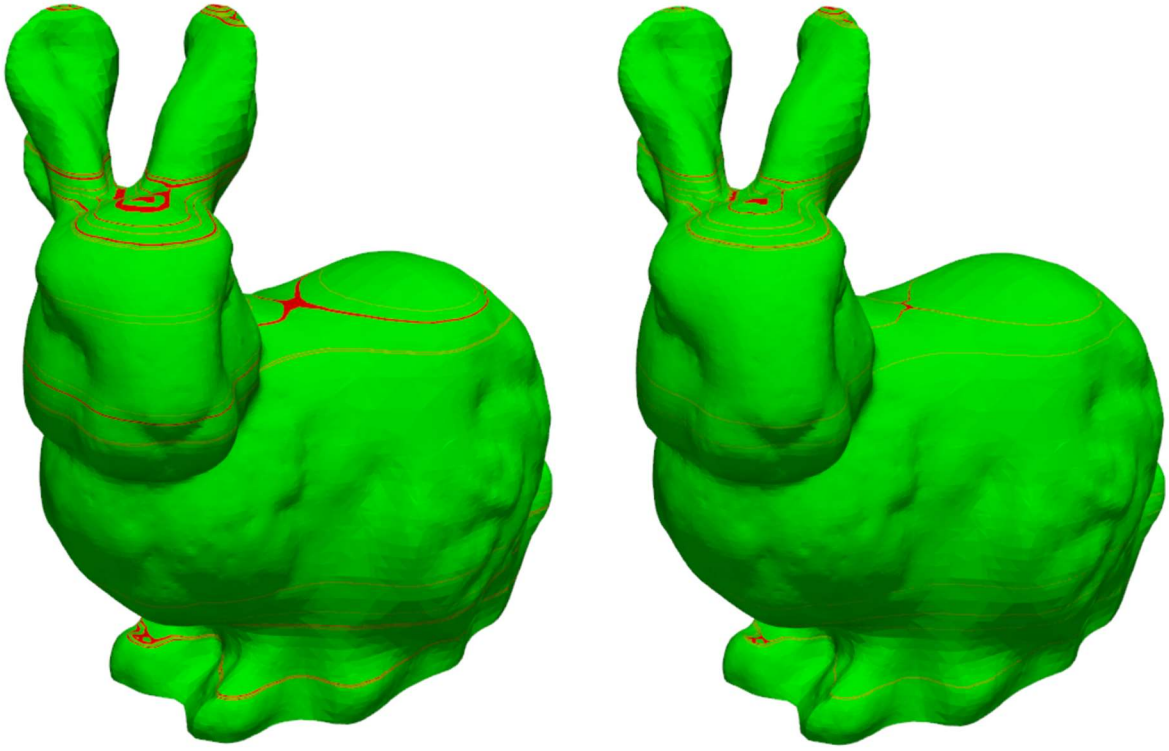

Рисунок 5.5. Результат роботи алгоритму на моделі Stanford Bunny з $\delta = 0.05$ (ліворуч), та з $\delta = 0.15$ (праворуч)

Перевірку ефективності запропонованого підходу в якості методу оптимізації алгоритмів, що виконують декомпозицію вхідного многокутника було проведено для задачі розбиття множини простих многокутників на опуклі. Порівняння було виконано з алгоритмом (Hertel & Mehlhorn, 1985). Реалізація зазначеного алгоритму було використано з бібліотеки PolyPartition (Fratric, 2021).

Відновлення декомпозиції виконувалось алгоритмом, описаним в розділі 4.3.3. В якості критерія оцінки ефективності використано середній час побудови декомпозиції за 100 виконань алгоритму на однакових вхідних даних.

Результати експериментів показали, що застосування запропонованого методу оптимізації дозволяє отримати приріст ефективності обробки послідовності слайсів від 30% до 50%, залежно від тестових даних та кількості оброблюваних слайсів. Наприклад, результати порівняння двох підходів на слайсах моделі Stanford Bunny, подані у вигляді стовпчикової діаграми на рисунку 5.6, демонструють, що при обробці 2000 слайсів час виконання зменшується на 32.2%, а при 4000 слайсах – на 47.8%, тобто операція виконується майже вдвічі швидше.

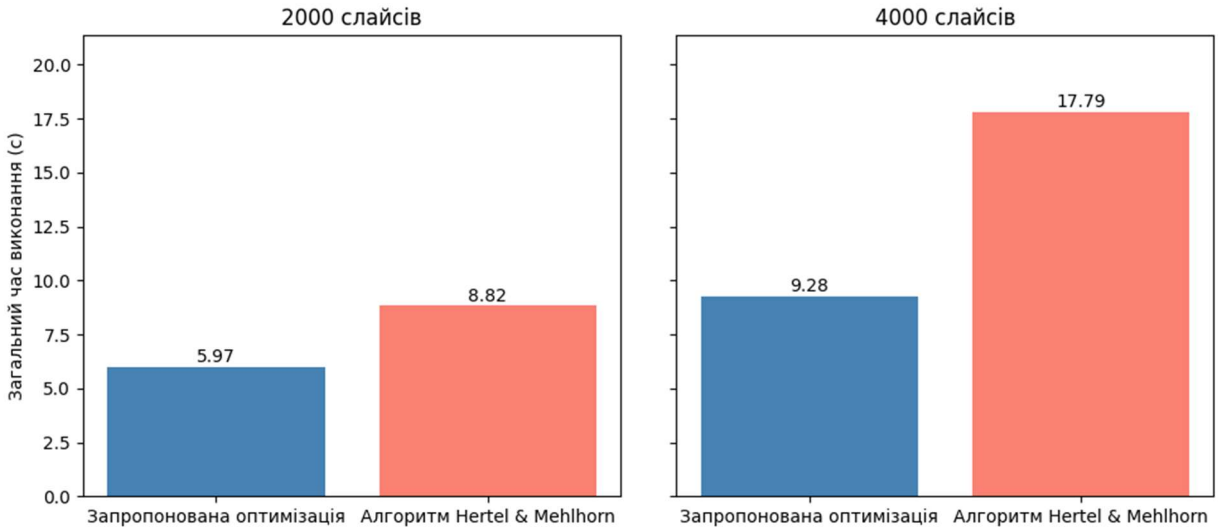

Рисунок 5.6. Порівняння часу декомпозиції набору слайсів з застовуванням запропонованої оптимізації та без неї

5.3 Реалізація та тестування алгоритму відновлення декомпозиції

Алгоритм відновлення декомпозиції шляхом додавання хорд між вершинами плоского многокутника, запропонований в підрозділі 4.3.3, було реалізовано мовою програмування *C++* з використанням бібліотеки BVH (Pérard-Gayot, 2025), що реалізовує ієрархію обмежуючих об’ємів на основі *SAH*-

евристики. Для структури даних *MinMax Segment Tree* було розроблено власну нерекурсивну реалізацію.

Тестування проводилось на простих многокутниках з набору, процес генерування якого описано в підрозділі 5.1.1. Кількість вершин в тестових многокутниках варіювалася від 10^4 до 10^6. Підрозбиття кожного з многокутників було виконано рівномірно розподіленими хордами. В якості метрики ефективності використовувався середній час, необхідний для обробки різної кількості запитів для кожного з многокутників. Результати тестування представлені на рисунку 5.7, де вісь часу відображена в логарифмічному масштабі.

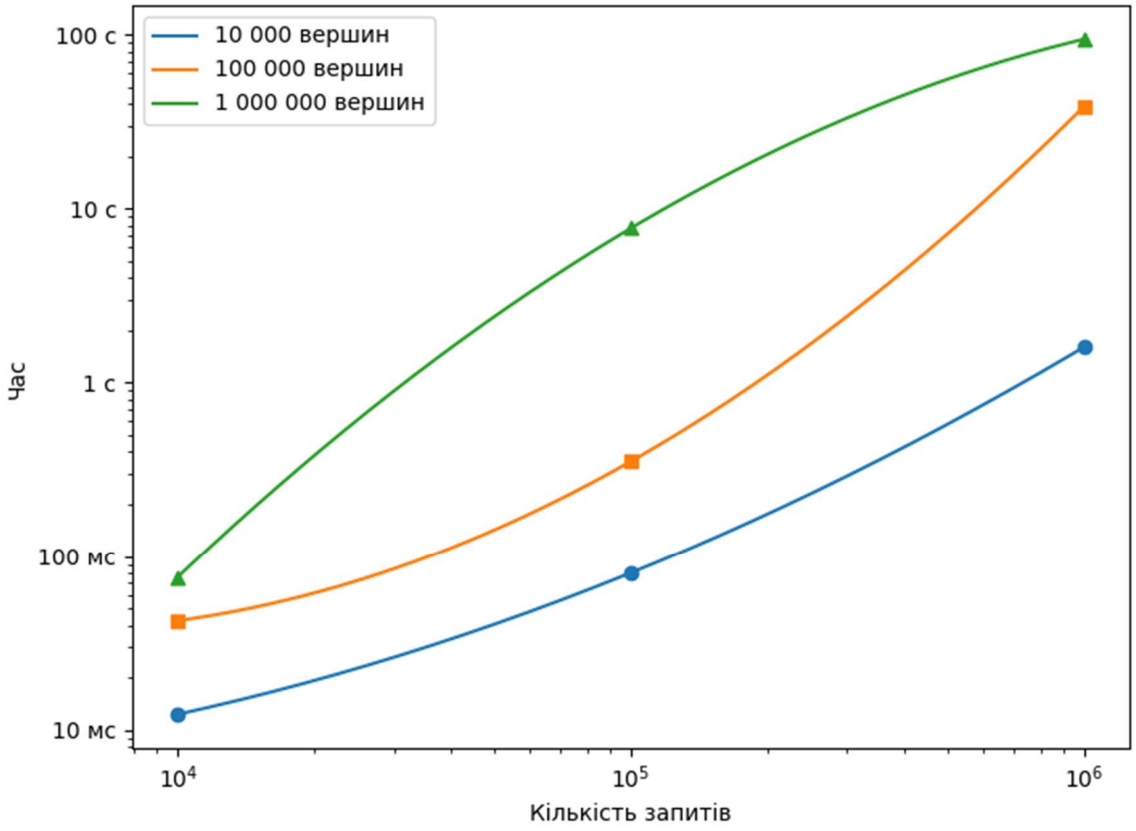

Рисунок 5.7. Середній час, необхідний для обробки 10^4, 10^5 та 10^6 запитів

Результати тестування показують, що навіть для найважчих многокутників з високою деталізацією контуру, обробка 10^6 запитів зайняла приблизно 94 секунди. Тобто для таких многокутників середній час обробки одного запиту становить трохи менше 0.1 мс. Продемонстрована швидкодія підтверджує

ефективність запропонованого підходу розв’язання задачі відновлення декомпозиції навіть у сценаріях реального часу.

Інше спостереження стосується нерівномірності середнього часу виконання перших 10^5 запитів на многокутниках з кількістю вершин до 10^5, та перших 10^4 запитів на многокутниках, що мають 10^6 вершин, з середнім часом виконання решти запитів. Це, ймовірно, пов'язано з особливостями проведення тестування та вимірюванні швидкодії. Тестові запити були попередньо згенерованими, оскільки генерування безпосередньо в процесі проведення вимірів призведе до суттєвих похибок в оцінці часу виконання запиту за рахунок відносно високої обчислювальної складності генерування випадкових чисел. З врахуванням цього, зазначені спостереження пояснюються тим, що тестова робоча станція має достатній обсяг кеш-пам'яті для розміщення деяких попередньо згенерованих запитів. Однак, зі збільшенням кількості запитів, процесор повинен чесно отримувати дані з оперативної пам'яті, що призводить до зниження продуктивності. Тому, хоча результати тестування для більшої кількості запитів є більш надійними, можна очікувати швидшого часу відгуку, якщо очікувана кількість запитів, що обробляються, є невеликою з самого початку, а задача, що розв’язується, допускає групування запитів.

ВИСНОВКИ

В дисертації було досліджено актуальну науково-прикладну проблему побудови універсальної алгоритмічної платформи для розв’язання комплексу задач адитивного виробництва. Головним результатом дисертації є побудова моделі єдиного алгоритмічного середовища на основі МСД та з використанням принципу неявного зведення репрезентацій, яке здатне ефективно розв’язувати задачі процесу планування АВ, що подані у вигляді робочого процесу (композиції задач). Основна увага приділялась задачам генерування мешів (в різних постановках), оптимізації орієнтації та розміщення моделей в середні робочої області принтера, генерування опорних конструкцій, слайсингу та побудові траєкторії друку, оскільки ці задачі є актуальними для більшості технологій АВ, однак запропонована МЄАС передбачає можливість масштабування шляхом додавання нових задач.

В дисертації отримано наступні результати. Вперше:

1. Побудовано архітектуру моделі єдиного алгоритмічного середовища для розв’язання комплексу задач адитивного виробництва, розширену шляхом введення інструментів керування обчисленнями на рівні архітектури (МСД, неявне зведення репрезентацій), що дозволило вперше отримати МЄАС, що об’єднує задачі, між якими немає залежності в розумінні теорії звідності. Ключовим елементом архітектури виступає менеджер структур даних (МСД), що відповідає за збереження спільних структур даних та підтримку їх в актуальному стані. Робота МСД ґрунтується на розробленій класифікації високорівневих геометричних алгоритмів за характером змін, які вони вносять в дані. Оновлення структур даних відбувається відкладено, в момент, коли структура даних запитується одним із алгоритмів. На рівні архітектури МЄАС реалізовано формальний підхід до автоматизації процесу зведення пов’язаних репрезентацій даних. Правила зведення формалізуються у вигляді редукційного графа - орієнтованого графа, вузлами якого є репрезентації, та з кожним ребром якого

асоціюється функція оцінки обчислювальної вартості виконання зведення. Процес зведення відбувається неявно, оптимальний шлях зведення автоматично визначається пошуком найдешевшого шляху в редукційному графі для кожного конкретної окремо взятої репрезентації. Доведено, що модель єдиного алгоритмічного середовища для взаємопов'язаних задач обчислювальної геометрії (Терещенко, 2011) може бути реалізована в термінах розробленої архітектури розширеної МЄАС (теорема 3.1).

2. Розроблено оптимізацію алгоритму розв'язання задачі слайсингу (Minetto et al., 2017) при роботі в межах МЄАС, для якого показано, що за наявності воксельної сітки в МСД можна досягнути лінійного часу роботи в середньому (алгоритм 4.1).
3. Розроблено алгоритм декомпозиції плоского многокутника на набір y-монотонних тріангульованих многокутників з використанням графу Ріба, що має лінійну асимптотичну оцінку складності за умови існування ГР (алгоритм 4.3). Відповідний результат сформульовано у вигляді теореми 4.3.
4. Розроблено оптимізацію алгоритму розв'язання задачі побудови траєкторії друку (Dwivedi & Kovacevic, 2004) при роботі в межах МЄАС, з використанням графу Ріба та алгоритму 4.3, що дозволило зменшити асимптотичну оцінку часу виконання попередньої обробки з $O(n \log n)$ до $O(n)$, де n – кількість вершин многокутника.
5. Розроблено алгоритм оцінки подібності двох простих многокутників на площині (алгоритм 4.4). Ключовою перевагою розробленого алгоритму є врахування особливостей форми многокутників, а не лише перевірка близькості в теоретико-множинному розумінні (з використанням метрики Гаусдорфа). Доведено, що алгоритм має лінійну оцінку швидкості роботи та константу оцінку додаткової пам'яті. Відповідні результати сформульовані у вигляді теорем 4.4 та 4.5.
6. Розроблено алгоритм підтримки хордової декомпозиції простого многокутника в режимі онлайн. Показано, що обробка одного запиту

додавання хорди може бути зведена до двох підзадач - трасування променя та пошуку мінімуму та максимуму на заданому відрізку. Першу підзадачу розв'язано за допомогою ієрархії обмежуючих об'ємів, а другу – за допомогою дерева відрізків, у вузлах якого одночасно підтримується мінімальне та максимальне значення (алгоритм 4.5). Показано, що асимптотична оцінка складності виконання q запитів на многокутнику з n вершин складає $O(n + q\log n)$.

7. Розроблено оптимізацію алгоритмів розв'язання задачі побудови траєкторії друку, що виконують декомпозицію многокутника, зокрема (Dwivedi & Kovacevic, 2004), (Ding et al., 2014) та (Zhao et al., 2024), шляхом врахування подібності між сусідніми слайсами моделі з використанням алгоритмів 4.4 та 4.5.

Розроблені в дисертації алгоритми були реалізовані в МЄАС мовою програмування C++ та компілятора MSVC 2022 19.44 (Update 14). Тестування проводилось на машині з процесором Intel® Core™ i5-13600K.

Дослідження ефективності алгоритму 4.3 було проведено в порівнянні найбільш вживаними на сьогодні бібліотеками, які дозволяють розв'язувати задачу тріангуляції, зокрема CGAL, GeometricTools та PolyPatrition. Результати порівняння показують, що за умови попереднього обрахунку ГР запропонований алгоритм є в рази швидшим порівняно з розглянутими бібліотеками: в залежності від розміру вхідного многокутника, він є в 2-3 рази швидшим за реалізацію алгоритму тріангуляції шляхом розбиття многокутника на y-монотонні частини з бібліотеки PolyPartition, та в 5-6 разів швидше за алгоритми тріангуляції з бібліотеки CGAL.

Практичні експерименти з алгоритмом 4.4 на тестових 3D-моделях показали, що алгоритм здатний знаходити повні відповідності більш ніж у половині слайсів, а з урахуванням часткових збігів кількість слайсів, для яких алгоритм визначає хоча б один схожий многокутник сягає 96%, що дозволяє суттєво знизити обчислювальні витрати в задачах, де використовується метод

декомпозиції довільного многокутника на набір многокутників із деякими заданими властивостями. Також експериментальні результати підтвердили ефективність застосування розробленого алгоритму як способу оптимізації задач обчислювальної геометрії, що виконують декомпозицію набору многокутників на многокутники з заданими властивостями. Зокрема, такий підхід дозволив отримати приріст продуктивності на 30% - 50% на розглянутих тестових даних.

Тестування розробленого підходу до відновлення декомпозиції як задачі підтримки хордового розбиття з використанням алгоритму 4.5 та ієрархії обмежуючих об'ємів для трасування променя підтвердило ефективність цього підходу для роботи в реальному часі. На найбільших тестових многокутниках з 10^6 вершин середній час обробки одного запиту алгоритмом склав 0.1 мс, тобто обробка 10^6 запитів була виконана менш ніж за 100 секунд. На менших многокутниках швидкість обробки аналогічної кількості запитів була ще вищою: 39 секунд для многокутників з 10^5 вершин, та всього 1.6 секунди для многокутників з 10^4 вершин.

Доцільними напрямками подальшого розширення побудованої моделі єдиного алгоритмічного середовища є її адаптація до паралельних обчислень. В поточному стані МЄАС може працювати в паралельному середовищі, але лише з паралелізмом на рівні алгоритму або на рівні незалежних геометричних об'єктів. Можливість одночасно виконувати кілька алгоритмів над однією моделлю може бути корисною з практичної точки зору (наприклад, вона дозволить виконувати конвертацію CAD даних в STL та паралельно виправляти дефекти цієї конвертації), однак вона потребує проведення додаткових досліджень. Також доречним напрямком розвитку запропонованої МЄАС є її застосування до інших проблемних областей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Aggarwal, V., Chen, J., & Lan, T. (2023) A unified algorithm framework for unsupervised discovery of skills based on determinantal point process. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23)* (pp. 67925–67947). Curran Associates Inc. https://doi.org/10.52202/075280-2971

Aleardi, L.C., & Devillers, O. (2018). Array-based compact data structures for triangulations: Practical solutions with theoretical guarantees. *Journal of Computational Geometry, 9*(1), 247–289. https://doi.org/10.20382/jocg.v9i1a8

Art of Problem Solving. (n.d.). *Shoelace Theorem*. AoPS Wiki. Retrieved October 9, 2025, from https://artofproblemsolving.com/wiki/index.php/Shoelace_Theorem

Atallah, M. J. (1983). A linear time algorithm for the Hausdorff distance between convex polygons. *Information Processing Letters, 17*(4), 207–209. https://doi.org/10.1016/0020-0190(83)90042-x

Auer, T., & Held, M. (1996). Heuristics for the Generation of Random Polygons. In F. Fiala, E. Karanakis, & J.-R. Sack (Eds.), *Canadian Conference on Computational Geometry* (pp. 38–43). McGill-Queen's University Press. https://doi.org/10.1515/9780773591134-009

Auer, T., Gschwandtner, M., Heimlich, M., Held, M., & Palfrader, P. (2020). *Random Polygon Generator (RPG)* [Computer software]. GitHub. https://github.com/cgalab/genpoly-rpg

Autodesk. (n.d.). *Autodesk Fusion with Netfabb: Additive manufacturing, design, and simulation* [Computer software]. Autodesk, Inc. https://www.autodesk.com/products/netfabb/overview

Avbelj, J., Muller, R., & Bamler, R. (2015). A Metric for Polygon Comparison and Building Extraction Evaluation. *IEEE Geoscience and Remote Sensing Letters, 12*(1), 170–174. https://doi.org/10.1109/lgrs.2014.2330695

Bai, Y.-B., Yong, J.-H., Liu, C.-Y., Liu, X.-M., & Meng, Y. (2011). Polyline approach for approximating Hausdorff distance between planar free-form curves. *Computer-Aided Design, 43*(6), 687–698. https://doi.org/10.1016/j.cad.2011.02.008

Baumgart, B. G. (1972). *Winged Edge Polyhedron Representation* (Report No. STAN-CS-72-320). Stanford University. http://infolab.stanford.edu/pub/cstr/reports/cs/tr/72/320/CS-TR-72-320.pdf

Bentley, J., & Ottmann, T. (1979). Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers, C–28*(9), 643–647. https://doi.org/10.1109/tc.1979.1675432

Bentley, J., Haken, D., & Saxe, J. (1980). A general method for solving divide-and-conquer recurrences. *ACM SIGACT News, 12*(3), 36–44. https://doi.org/10.1145%2F1008861.1008865

Blender Foundation. (2025). *Blender - The Free and Open Source 3D Creation Software* (Version 4.3) [Computer software]. https://www.blender.org/

Blender Foundation. (n.d.). *Import & Export of Node Shaders*. Blender Manual. Retrieved June 17, 2025, from https://docs.blender.org/manual/en/2.81/addons/import_export/io_node_shaders_info.html

Buchin, K., Custers, B., van der Hoog, I., Löffler, M., Popov, A., Roeloffzen, M., & Staals, F. (2022). Segment Visibility Counting Queries in Polygons. In S. W. Bae, & H. Park (Eds.), *LIPIcs ISAAC 2022* (No. 58; Vol. 248, p. 58:1-58:16). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPICS.ISAAC.2022.58

Campen, M., Attene, M., & Kobbelt, L. (2012). A Practical Guide to Polygon Mesh Repairing. *Eurographics*. https://api.semanticscholar.org/CorpusID:42550171

Carr, H., Snoeyink, J., & Axen, U. (2003). Computing contour trees in all dimensions. *Computational Geometry, 24*(2), 75–94. https://doi.org/10.1016/s0925-7721(02)00093-7

Cavanna, N. J., Khoury, M., & Sheehy, D. R. (2017). Supporting Ruled Polygons. *arXiv*. https://doi.org/10.48550/ARXIV.1707.00826

Chen, R., Wu, X., Pan, Y., Yuan, K., Li, L., Ma, T., Liang, J., Zhang, R., Wang, K., Zhang, C., Peng, S., Zhang, X., Du, Z., Guo, Q., & Chen, Y. (2021) *Eden: A Unified*

Environment Framework for Booming Reinforcement Learning Algorithms. arXiv. https://doi.org/10.48550/arXiv.2109.01768

Chernov, N., Stoyan, Yu., & Romanova, T. (2010). Mathematical model and efficient algorithms for object packing problem. *Computational Geometry, 43*(5), 535–553. https://doi.org/10.1016/j.comgeo.2009.12.003

Chiu, B. W. (2020). *Additive manufacturing applications and implementation in Aerospace* [Master's thesis, Massachusetts Institute of Technology]. DSpace@MIT. https://dspace.mit.edu/bitstream/handle/1721.1/126950/1191622655-MIT.pdf

Cormen, T.H., Leiserson, C.E., Rivest, R.L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). The MIT Press.

De Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer.

De Berg, M., Gudmundsson, J., & Mehrabi, A. D. (2017). Finding Pairwise Intersections Inside a Query Range. *Algorithmica, 80*(11), 3253–3269. https://doi.org/10.1007/s00453-017-0384-3

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197. https://doi.org/10.1109/4235.996017

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269–271. https://doi.org/10.1007/bf01386390

Ding, D., Pan, Z., Cuiuri, D., & Li, H. (2014). A tool-path generation strategy for wire and arc additive manufacturing. *The International Journal of Advanced Manufacturing Technology, 73*(1–4), 173–183. https://doi.org/10.1007/s00170-014-5808-5

Dwivedi, R., & Kovacevic, R. (2004). Automated torch path planning using polygon subdivision for solid freeform fabrication based on welding. *Journal of Manufacturing Systems, 23*(4), 278–291. https://doi.org/10.1016/s0278-6125(04)80040-2

Eberly, D. (2024). *Triangulation by ear clipping*. Geometric Tools. https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf

EOS GmbH. (2025). *EOS Build. Data preparation and job optimization for industrial additive manufacturing* (Version 24) [Computer software]. https://www.eos.info/enablement/software

Eryildiz, M. (2021). Effect of Build Orientation on Mechanical Behaviour and Build Time of FDM 3D-Printed PLA Parts: An Experimental Investigation. *European Mechanical Science, 5*(3), 116–120. https://doi.org/10.26701/ems.881254

Formlabs. (2025). *Formlabs PreForm. Powerful 3D Print Preparation Software* (Version 3.45.0) [Computer software]. https://formlabs.com/software/preform

Fratric, I. (2021). *PolyPartition: a lightweight C++ library for polygon partition and triangulation* [Computer software]. https://github.com/ivanfratric/polypartition

Geometric Tools. (2025). *Geometric Tools Engine* (Version 7.3) [Computer software]. https://www.geometrictools.com

Hart, P. E., Nilsson, N.J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. https://doi.org/10.1109/tssc.1968.300136

Hertel, S., & Mehlhorn, K. (1985). Fast triangulation of the plane with respect to simple polygons. *Information and Control, 64*(1–3), 52–76. https://doi.org/10.1016/s0019-9958(85)80044-9

Ho, C.-C, Wu, F., Chen, B., Chuang, Y., & Ouhyoung, M. (2005). Cubical Marching Squares: Adaptive Feature Preserving Surface Extraction from Volume Data. *Computer Graphics Forum, 24*(3), 537–545. https://doi.org/10.1111/j.1467-8659.2005.00879.x

Hong, M., Razaviyayn, M., Luo, Z.-Q., & Pang, J.-S. (2016). A Unified Algorithmic Framework for Block-Structured Optimization Involving Big Data: With applications in machine learning and signal processing. *IEEE Signal Processing Magazine, 33*(1), 57-77. https://doi.org/10.1109/MSP.2015.2481563

Hosick, E. (2014, February 20). Visual Programming Languages – Snapshots. *Interface Vision*. http://blog.interfacevision.com/design/design-visual-progarmming-languages-snapshots/

Huang, S. H., Liu, P., Mokasdar, A., & Hou, L. (2012). Additive Manufacturing and its societal impact: A literature review. *The International Journal of Advanced Manufacturing Technology, 67*(5–8), 1191–1203. https://doi.org/10.1007/s00170-012-4558-5

Imai, H., & Asano, T. (1984). Dynamic Segment Intersection Search With Applications. *25th Annual Symposium onFoundations of Computer Science, 1984*., 393–402. https://doi.org/10.1109/sfcs.1984.715940

International Organization for Standardization. (2009). *Automation systems and integration — Numerical control of machines — Program format and definitions of address words Part 1: Data format for positioning, line motion and contouring control systems* (ISO Standard No. 6983-1:2009). https://www.iso.org/standard/34608.html

International Organization for Standardization. (2021). *Additive manufacturing — General principles — Fundamentals and vocabulary* (ISO/ASTM Standard No. 52900:2021). https://www.iso.org/standard/74514.html

International Organization for Standardization. (2024). *Industrial automation systems and integration - Product data representation and exchange. Part 1: Overview and fundamental principles.* (ISO Standard No. 10303-1:2024). https://www.iso.org/standard/83105.html

Jaccard, P. (1901). Distribution de la Flore Alpine dans le Bassin des Dranses et dans quelques régions voisines. *Bulletin de la Societe Vaudoise des Sciences Naturelles, 37*(140), 241–272. https://doi.org/10.5169/seals-266440

Jacobson, A. (2017). *Common 3D test models* [Data set]. GitHub. https://github.com/alecjacobson/common-3d-test-models

Jiang, X., Peng, Q., Cheng, X., Dai, N., Cheng, C., & Li, D. (2016). Efficient Booleans algorithms for triangulated meshes of geometric modeling. *Computer-Aided Design and Applications, 13*(4), 419–430. https://doi.org/10.1080/16864360.2015.1131530

Jin, Y., He, Y., & Fu, J. (2015). Support generation for additive manufacturing based on sliced data. *The International Journal of Advanced Manufacturing Technology, 80*(9–12), 2041–2052. https://doi.org/10.1007/s00170-015-7190-3

Joe, B., & Simpson, R. B. (1987). Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics, 27*(4), 458–473. https://doi.org/10.1007/bf01937271

Keil, M., & Snoeyink, J. (2002). On the time bound for convex decomposition of simple polygons. *International Journal of Computational Geometry & Applications, 12*(3), 181–192. https://doi.org/10.1142/s0218195902000803

Khronos Group. (2025a). *OpenGL: The Industry's Foundation for High Performance Graphics* (Version 4.6) [Computer software]. https://www.opengl.org/

Khronos Group. (2025b). *Vulkan - Cross platform 3D Graphics* (Version 1.4.308) [Computer software]. https://www.vulkan.org/

Kim, D.-S., Kim, D., Cho, Y., & Sugihara, K. (2006). Quasi-triangulation and interworld data structure in three dimensions. *Computer-Aided Design, 38*(7), 808–819. https://doi.org/10.1016/j.cad.2006.04.008

Kishor, G., Sharma, Y. K., Mugada, K. K., & Mahto, R. P. (2026). A review on in-situ monitoring, process control and residual stress prediction using machine learning methods in metal-based laser additive manufacturing. *Progress in Additive Manufacturing*. https://doi.org/10.1007/s40964-026-01532-y

Knuth, D. E., Morris, Jr., J. H., & Pratt, V. R. (1977). Fast Pattern Matching in Strings. *SIAM Journal on Computing, 6*(2), 323–350. https://doi.org/10.1137/0206024

Kobbelt, L. (2000). √3-subdivision. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’00 (pp. 103–112). ACM Press. The 27th annual conference. https://doi.org/10.1145/344779.344835

Kobbelt, L. P., Botsch, M., Schwanecke, U., & Seidel, H.-P. (2001). *Feature sensitive surface extraction from volume data*. Proceedings of the 28th annual conference on Computer graphics and interactive techniques (pp. 57–66). ACM. SIGGRAPH01: The 28th International Conference on Computer Graphics and Interactive Techniques. https://doi.org/10.1145/383259.383265

Kok, Y., Tan, X. P., Wang, P., Nai, M. L. S., Loh, N. H., Liu, E., & Tor, S. B. (2018). Anisotropy and heterogeneity of microstructure and mechanical properties in metal additive manufacturing: A critical review. *Materials & Design, 139*, 565–586. https://doi.org/10.1016/j.matdes.2017.11.021

Kwok, T.-H., Ye, H., Chen, Y., Zhou, C., & Xu, W. (2017). Mass customization: Reuse of digital slicing for additive manufacturing. *Journal of Computing and Information Science in Engineering, 17*(2). https://doi.org/10.1115/1.4034010

Lan, P.-T., Chou, S.-Y., Chen, L.-L., & Gemmill, D. (1997). Determining fabrication orientations for rapid prototyping with Stereolithography apparatus. *Computer-Aided Design, 29*(1), 53–62. https://doi.org/10.1016/s0010-4485(96)00049-8

Library of Congress. (2025). *STL (STereoLithography) File Format Family*. Sustainability of Digital Formats. https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml

Lin, L., Xu, Z., Huan, H., Jian, Z., & Li-Xin, L. (2021). An Empirical Comparison of Implementation Efficiency of Iterative and Recursive Algorithms of Fast Fourier Transform. In M. Guan, & Z. Na (Eds.), *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST)*: Vol. 342 (pp. 73–81). Springer International Publishing. https://doi.org/10.1007/978-3-030-66785-6_9

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics, 21*(4), 163–169. https://doi.org/10.1145/37402.37422

Lutters, E., ten Dam, D., & Faneker, T. (2012). 3D Nesting of Complex Shapes. *Procedia CIRP, 3*, 26–31. https://doi.org/10.1016/j.procir.2012.07.006

MacDonald, J. D., & Booth, K. S. (1989). *Heuristics for ray tracing using space subdivision*. Proceedings of Graphics Interface ’89 (pp. 152–163). Canadian Information Processing Society. https://doi.org/10.20380/GI1989.22

Mapbox. (2023). *Earcut: A C++ port of earcut.js, a fast, header-only polygon triangulation library* (Version 2.2.4) [Computer software]. https://github.com/mapbox/earcut.hpp

Mares, M. (2004). Two linear time algorithms for MST on minor closed graph classes. *Archivum Mathematicum, 40*(3), 315–320. https://dml.cz/handle/10338.dmlcz/107914

Meagher, D. (1980). *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer* (Report No. IPL-TR-80-111). Rensselaer Polytechnic Institute. https://apps.dtic.mil/sti/tr/pdf/ADA117450.pdf

Meyer, B. (1997). *Object-Oriented Software Construction* (2nd ed.). Prentice-Hall, Inc.

Microsoft. (2025). *DirectX 12 Agility SDK* (Version 12) [Computer software]. https://devblogs.microsoft.com/directx/directx12agility/

Minetto, R., Volpato, N., Stolfi, J., Gregori, R. M. M. H., & da Silva, M. V. G. (2017). An optimal algorithm for 3D triangle mesh slicing. *Computer-Aided Design, 92*, 1–10. https://doi.org/10.1016/j.cad.2017.07.001

Newgas, A. (2018, April 15). Dual contouring tutorial. *Boris the Brave.* https://www.boristhebrave.com/2018/04/15/dual-contouring-tutorial/

Nystrom, R. (2014). *Game Programming Patterns*. https://gameprogrammingpatterns.com

Osiponok, M., & Tereshchenko, V. (2019). *The "Divide and Conquer" technique to solve the Minimum Area Polygonalization problem*. 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT) (pp. 336–339). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/atit49449.2019.9030463

Pankratov, A., Romanova, T., & Litvinchev, I. (2020). Packing Oblique 3D Objects. *Mathematics, 8*(7), 1130. https://doi.org/10.3390/math8071130

Pérard-Gayot, A. (2025). *bvh: A modern C++ BVH construction and traversal library* (Version 1) [Computer software]. https://github.com/madmann91/bvh

Petrova, K., & Tarjan, R. (2016). *A Dynamic Data Structure for Segment Intersection Queries: Extended Abstract*. 15th International Workshop on Algebraic and Combinatorial Coding Theory, Albena, Bulgaria (pp. 244–249)

Prusa Research. (2025). *PrusaSlicer. G-code generator for 3D printers (RepRap, Makerbot, Ultimaker etc.)* (Version 2.9.4) [Computer software]. https://github.com/prusa3d/PrusaSlicer

Ramamurthi, Y., & Chattopadhyay, A. (2022). Topological Shape Matching using Multi-Dimensional Reeb Graphs. In S. Biswas, S. Raman, & A. Roy-Chowdhury (Eds.), *Proceedings of the Thirteenth Indian Conference on Computer Vision, Graphics and Image Processing ICVGIP'22* (pp. 1–10). ACM. https://doi.org/10.1145/3571600.3571606

Rosato, A., Strandburg, K. J., Prinz, F., & Swendsen, R. H. (1987). Why the Brazil nuts are on top: Size segregation of particulate matter by shaking. *Physical Review Letters, 58*(10), 1038–1040. https://doi.org/10.1103/physrevlett.58.1038

Rosenthal, S. (2016). *Software Engineering for Additive Manufacturing* [Poster abstract]. Carnegie Mellon University. Software Engineering Institute. https://www.sei.cmu.edu/library/software-engineering-for-additive-manufacturing/

Rozenberg, G., & Salomaa, A. (1976). The Mathematical Theory of L Systems. *Advances in Information Systems Science, 6*, 161–206. https://doi.org/10.1007/978-1-4615-8249-6_4

Rubin, S. M., & Whitted, T. (1980). A 3-dimensional representation for fast rendering of complex scenes. *ACM SIGGRAPH Computer Graphics, 14*(3), 110–116. https://doi.org/10.1145/965105.807479

Ryoshoru. (2023). *The originally published 15 cube configurations* [Image]. Wikipedia. https://en.wikipedia.org/wiki/Marching_cubes#/media/File:MarchingCubesEdit.svg

Salmi, M. (2021). Additive manufacturing processes in medical applications. *Materials, 14*(1), 191. https://doi.org/10.3390/ma14010191

Savage, J.E. (1998). *Models Of Computation: Exploring the Power of Computing*. Addison-Wesley.

Schaefer, S., Ju, T., & Warren, J. (2007). Manifold Dual Contouring. *IEEE Transactions on Visualization and Computer Graphics, 13*(3), 610–619. https://doi.org/10.1109/tvcg.2007.1012

Scherzinger, A., Brix, T., & H. Hinrichs, K. (2017). An Efficient Geometric Algorithm for Clipping and Capping Solid Triangle Meshes. In A.P. Cláudio, D. Bechmann, & J. Braz, *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* (pp. 187–194). Springer. https://doi.org/10.5220/0006097201870194

Siemens AG. (2025). *Siemens NX software* (Version 2506) [Computer software]. https://www.siemens.com/en-us/products/cad-cam-software/

Sinterit. (2025, February 24). Pack density in SLS 3D printing. *Sinterit SLS Technology*. https://sinterit.com/blog/sls-technology/pack-density-in-sls-3d-printing/

Stoyan, Y., Pankratov, A., Romanova, T., Fasano, G., Pintér, J. D., Stoian, Y. E., & Chugay, A. (2019). Optimized Packings in Space Engineering Applications: Part I. In G. Fasano, & J.D. Pintér (Eds.), *Modeling and Optimization in Space Engineering: State of the Art and New Challenges* (pp. 395–437). Springer International Publishing. https://doi.org/10.1007/978-3-030-10501-3_15

Tereshchenko, V. (2010) *The system of common algorithmic space to create visual models of phenomena and processes.* Proceedings of the International Conference on Applications of Computer and Information Sciences to Nature Research (pp. 64–68). ACM. ACISNR ’10: Applications of Computer and Information Sciences to Nature Research. https://doi.org/10.1145/1868013.1868030

Tereshchenko, V. N., & Anisimov, A. V. (2010). Recursion and parallel algorithms in geometric modeling problems. *Cybernetics and Systems Analysis, 46*(2), 173–184. https://doi.org/10.1007/s10559-010-9196-z

Tereshchenko, V., & Prishlyak, O. (2024). Reeb graph of the height function on a planar polygon. *Bulletin of Taras Shevchenko National University of Kyiv. Series: Physics and Mathematics, 79*(2), 54–58. https://doi.org/10.17721/1812-5409.2024/2.9

Tereshchenko, V., Prishlyak, A., & Osiponok, M. (2025). Constructing Efficient Algorithms for Solving Certain Computational Geometry Problems Using Reeb Graphs. *Cybernetics and Systems Analysis, 61*(6), 924-933. https://doi.org/10.1007/s10559-025-00825-4

The CGAL Project. (2024). *CGAL: The Computational Geometry Algorithms Library* (Version 6.0.1) [Computer software]. https://www.cgal.org

The Qt Company. (2024). *Qt: Tools for Each Stage of Software Development Lifecycle* (Version 6.8.0) [Computer software]. https://www.qt.io

Ultimaker. (2025). *Cura. 3D printer / slicing GUI built on top of the Uranium framework* (Version 5.9.1) [Computer software]. https://github.com/Ultimaker/Cura

Vaidya, R., & Anand, S. (2016). Optimum Support Structure Generation for Additive Manufacturing Using Unit Cell Structures and Support Removal Constraint.

Procedia Manufacturing, 5, 1043–1059. https://doi.org/10.1016/j.promfg.2016.08.072

Valdivieso, C. (2021, September 9). The role of AM in the automotive industry. *3Dnatives*. https://www.3dnatives.com/en/the-role-of-am-in-the-automotive-industry/#!

Volpato, N., Franzoni, A., Luvizon, D. C., & Schramm, J. M. (2013). Identifying the directions of a set of 2D contours for additive manufacturing process planning. *The International Journal of Advanced Manufacturing Technology, 68*(1–4), 33–43. https://doi.org/10.1007/s00170-012-4706-y

VoxelMatters. (2019.). *Map of additive manufacturing technologies.* VoxelMatters. https://www.voxelmatters.com/map-of-additive-manufacturing-technologies/

Wald, I. (2007). *On fast Construction of SAH-based Bounding Volume Hierarchies*. In 2007 IEEE Symposium on Interactive Ray Tracing (pp. 33–40). IEEE. IEEE/ EG Symposium on Interactive Ray Tracing 2007. https://doi.org/10.1109/rt.2007.4342588

Wohlers, T., Gornet, T., Mostow, N., Campbell, I., Diegel, O., Kowen, J., Huff, R., Stucker, B., Fidan, I., Doukas, A., Drab, B., Drstvenšek, I., Eitsert, N., Espalin, D., Feldhausen, T., Ghany, K., Gillett-Crooks, M., Guo, D., Held, A., … Peels, J. (2023). History of Additive Manufacturing. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4474824

Wong, K. V., & Hernandez, A. (2012). A Review of Additive Manufacturing. ISRN *Mechanical Engineering, 2012*, 1–10. https://doi.org/10.5402/2012/208760

Yang, S. W., & Choi, Y. (2010). Triangulation of CAD data for visualization using a compact array-based triangle data structure. *Computers & Graphics, 34*(4), 424–429. https://doi.org/10.1016/j.cag.2010.02.001

Yin, J. & Goh, J. (2019, December 10). Half-Edge Data Structures. Geometry Processing Algorithms. *JERRY YIN*. https://jerryyin.info/geometry-processing-algorithms/half-edge/

Yllemo, H. (2023, August 17). Single Source of Truth (SSOT). *ALMBok*. https://almbok.com/kb/single_source_of_truth

Yu, F., Cao, J., Shan, J., Lo, S. H., & Guan, Z. (2021). PASM: Parallel aligned surface meshing. *International Journal for Numerical Methods in Engineering, 122*(15), 3705–3732. https://doi.org/10.1002/nme.6678

Zhang, Y., Wang, Z., Zhang, Y., Gomes, S., & Bernard, A. (2020). Bio-inspired generative design for support structure generation and optimization in Additive Manufacturing (AM). *CIRP Annals, 69*(1), 117–120. https://doi.org/10.1016/j.cirp.2020.04.091

Zhao, T., Yan, Z., Zhao, Y., Jia, Y., & Chen, S. (2024). Path planning in additive manufacturing with multi-robot collaboration based on structural primitive partitioning. *Advances in Engineering Software, 197*(32), 103754. https://doi.org/10.1016/j.advengsoft.2024.103754

Ганєєв, Т., Прибитько, І., Руденко, М., & Петренко, І. (2023). *Адитивні технології. Навчальний посібник*. НУ «Чернігівська політехніка»

Коцур, Д. В. (2019). *Побудова моделі єдиного алгоритмічного середовища на основі діаграми вороного для розв'язування комплексу задач обробки зображень* [Дисертація на здобуття наукового ступеня кандидата фізико-математичних наук, Київський національний університет імені Тараса Шевченка]. Каталог дисертацій. Факультет комп'ютерних наук та кібернетики. http://csc.knu.ua/uk/library

Мальцев А.И. (1970). *Основы линейной алгебры*. Наука.

Пришляк О., & Терещенко В. (2024). *Обчислювальна геометрія. Основні конструкції на площині. Навчальний посібник*. Київський національний університет імені Тараса Шевченка.

Терещенко, В. М. (2011). *Побудова єдиного алгоритмічного середовища для розв'язування комплексу задач обчислювальної геометрії* [Дисертація на здобуття наукового ступеня доктора фізико-математичних наук, Київський національний університет імені Тараса Шевченка]. Національна бібліотека України імені В. І. Вернадського. http://nbuv.gov.ua/

Терещенко, В.М. (2013). Модифікація методу ланцюгів для задачі локалізації точки. *Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки, 36*(2), 233–236. https://bphm.knu.ua/bphm/issue/view/36

Додаток 1. Реалізація алгоритмів 4.2 та 4.3

```
void _AddVertexToTriangulation(std::vector<StackNode>& io_stack,
                               std::vector<std::size_t>& io_edge,
                               std::vector<std::array<std::size_t, 3>>& io_triangulation,
                               const Polygon& i_polygon,
                               std::size_t i_current_vertex,
                               std::size_t i_current_chain)
{
  assert(!io_stack.empty());
  if (io_stack.back().chain_index != i_current_chain) {
    auto prev = io_stack.back();
    for (const auto& top : std::ranges::reverse_view(io_stack)) {
      assert(io_edge[top.vertex_index] != std::size_t(-1));
      io_triangulation.push_back({ i_current_vertex, top.vertex_index, io_edge[top.vertex_index] });
    }
    io_edge[i_current_vertex] = prev.vertex_index;
    io_stack.clear();

    io_stack.push_back({ i_current_vertex, i_current_chain });
  } else {
    auto last = io_stack.back();
    auto prev_vertex = last.vertex_index;
    io_edge[i_current_vertex] = prev_vertex;
    while (io_stack.size() > 0) {
      auto v1 = prev_vertex;
      auto v2 = i_current_vertex;
      auto v3 = io_edge[io_stack.back().vertex_index];
      assert(v3 != std::size_t(-1));
      auto orientation = Reeb::GetOrientation(i_polygon.vertices[v1],
                                              i_polygon.vertices[v2],
                                              i_polygon.vertices[v3]);
      if (i_current_chain == 0 && orientation == Reeb::TripleOrientation::CounterClockwise) {
        break;
      }
      if (i_current_chain == 1 && orientation == Reeb::TripleOrientation::Clockwise) {
        break;
      }
```

```
            last = io_stack.back();
            io_stack.pop_back();
            assert(io_edge[last.vertex_index] != std::size_t(-1));
            io_triangulation.push_back({ i_current_vertex, last.vertex_index, io_edge[last.vertex_index] });
            io_edge[i_current_vertex] = io_edge[last.vertex_index];
            prev_vertex = io_edge[last.vertex_index];
          }
          io_stack.push_back({ i_current_vertex, i_current_chain });
        }
      }
    }

  ReebGraphSweepLine::MonotoneDecompositionResult ReebGraphSweepLine::SplitIntoYMonotonePolygons(const
Polygon& i_polygon, const std::vector<TargetVerticesList>& i_reeb_graph) const
  {
    const auto num_vertices = i_polygon.vertices.size();
                auto& y_lexicographical_indexing =
MUAE::GetRepresentation(MUAE::Representations::R_Y_SORTED_INDEXING, i_polygon);

    const auto prev = InvertPolygon(i_polygon);
    const auto flipped_reeb_graph = _FlipGraph(i_reeb_graph);

    auto is_split = [&](std::size_t vertex) { return i_reeb_graph[vertex].size() == 2; };
    auto is_merge = [&](std::size_t vertex) { return flipped_reeb_graph[vertex].size() == 2; };

    std::vector<std::size_t> order(num_vertices);
    for (std::size_t i = 0; i < num_vertices; ++i) {
      order[y_lexicographical_indexing[i]] = i;
    }

    struct Edge
    {
      std::size_t target = 0;
      std::size_t capacity = 0;
    };
    std::vector<std::vector<Edge>> extended_reeb_graph(num_vertices);
    for (std::size_t i = 0; i < num_vertices; ++i) {
```

```
        auto next_vertex = i_polygon.next[i];
        auto prev_vertex = prev[i];
        if (order[prev_vertex] > order[i]) {
            extended_reeb_graph[i].push_back({ prev_vertex, 1 });
        }
        if (order[next_vertex] > order[i]) {
            extended_reeb_graph[i].push_back({ next_vertex, 1 });
        }
        if (is_merge(i)) {
            assert(i_reeb_graph[i].size() == 1);
            extended_reeb_graph[i].push_back({ i_reeb_graph[i][0], 2 });
        }
        if (const auto& outcoming_edges = i_reeb_graph[i]; !outcoming_edges.empty()) {
            for (const auto to : outcoming_edges) {
                if (is_split(to)) {
                    if (std::none_of(extended_reeb_graph[i].begin(), extended_reeb_graph[i].end(), [to](const Edge&
i_edge) { return i_edge.target == to; })) {
                        extended_reeb_graph[i].push_back({ to, 2 });
                    }
                }
            }
        }

        std::sort(extended_reeb_graph[i].begin(), extended_reeb_graph[i].end(), [&](const Edge& lhs, const Edge& rhs)
        {
            auto vector_lhs = i_polygon.vertices[lhs.target] - i_polygon.vertices[i];
            auto vector_rhs = i_polygon.vertices[rhs.target] - i_polygon.vertices[i];
            auto angle_lhs = std::atan2(vector_lhs[1], vector_lhs[0]);
            auto angle_rhs = std::atan2(vector_rhs[1], vector_rhs[0]);
            return angle_lhs > angle_rhs;
        });
    }

    auto decrement_capacity = [](std::vector<Edge>& io_edges, std::size_t i_index)
    {
        auto it = io_edges.begin() + i_index;
        if ((--it->capacity) == 0) {
```

```
        io_edges.erase(it);
    }
};

std::vector<std::vector<std::size_t>> result;
std::vector<std::array<std::size_t, 3>> triangulation;
std::vector<std::size_t> edge(num_vertices, std::size_t(-1));
for (std::size_t curr_index = 0; curr_index < num_vertices; ++curr_index) {
    const auto curr = y_lexicographical_indexing[curr_index];
    while (extended_reeb_graph[curr].size() >= 2) {
        auto left = extended_reeb_graph[curr].front().target;
        assert(extended_reeb_graph[curr].front().capacity == 1);
        extended_reeb_graph[curr].erase(extended_reeb_graph[curr].begin());
        auto right = extended_reeb_graph[curr].front().target;
        decrement_capacity(extended_reeb_graph[curr], 0);


        std::vector<StackNode> stack;
        std::size_t postponed_init_chain = std::size_t(-1);
        bool postponed_init_needed = true;
        if (order[left] < order[right]) {
            stack.push_back({ left, 0 });
            edge[left] = curr;
            postponed_init_chain = 1;
        } else {
            stack.push_back({ right, 1 });
            edge[right] = curr;
            postponed_init_chain = 0;
        }

        std::vector<std::size_t> chains[2];
        chains[0].push_back(left);
        chains[1].push_back(right);

        while (chains[0].back() != chains[1].back()) {
            std::array<std::size_t, 2> v01 = {
```

```
                extended_reeb_graph[chains[0].back()].empty() ? std::size_t(-1) :
extended_reeb_graph[chains[0].back()].back().target,
                extended_reeb_graph[chains[1].back()].empty() ? std::size_t(-1) :
extended_reeb_graph[chains[1].back()].front().target
            };
            assert(v01[0] != std::size_t(-1) || v01[1] != std::size_t(-1));
            std::size_t nexti = std::size_t(-1);
            if (v01[0] != std::size_t(-1) && v01[1] != std::size_t(-1)) {
               nexti = order[v01[0]] < order[v01[1]] ? 0 : 1;
            } else if (v01[0] != std::size_t(-1)) {
               nexti = 0;
            } else {
               nexti = 1;
            }

            if (postponed_init_needed && chains[postponed_init_chain].size() == 1 &&
order[chains[postponed_init_chain][0]] < order[v01[nexti]]) {
               postponed_init_needed = false;
               _AddVertexToTriangulation(stack, edge, triangulation, m_polygon, chains[postponed_init_chain][0],
postponed_init_chain);
            }

            decrement_capacity(extended_reeb_graph[chains[nexti].back()], nexti == 1 ? 0 :
(extended_reeb_graph[chains[0].back()].size() - 1));
            auto prev_vertex = chains[nexti].back();
            chains[nexti].push_back(v01[nexti]);

            if (v01[0] == v01[1] || chains[0].back() == chains[1].back()) {
               continue;
            }

            _AddVertexToTriangulation(stack, edge, triangulation, m_polygon, v01[nexti], nexti);
         }

         for (std::size_t inv_stack_index = 0; inv_stack_index < stack.size(); ++inv_stack_index) {
            auto stack_index = inv_stack_index;
            triangulation.push_back({ chains[0].back(), stack[stack_index].vertex_index,
edge[stack[stack_index].vertex_index] });
         }
```

```
            auto& monotone_polygon = result.emplace_back();
            monotone_polygon.reserve(chains[0].size() + chains[1].size());
            monotone_polygon.push_back(curr);
            std::copy(chains[1].begin(), chains[1].end(), std::back_inserter(monotone_polygon));
            std::copy(chains[0].rbegin() + 1, chains[0].rend(), std::back_inserter(monotone_polygon));
        }
    }

                MUAE::PutRepresentation(MUAE::Representations::R_MONOTONE_DECOMPOSITION, result,
i_polygon);
                MUAE::PutRepresentation(MUAE::Representations::R_TRIANGULATION, triangulation,
i_polygon);
    return { result, triangulation };
  }
```

Додаток 2. Реалізація алгоритму 4.5

```
bool OneSideHausdorffTest(std::span<const Reeb::Point> vertices_a,
                                                    std::span<const std::size_t> loop_a,
                                                    std::span<const Reeb::Point> vertices_b,
                                                    std::span<const std::size_t> loop_b,
                                                    double delta)
{
        const auto na = loop_a.size(), nb = loop_b.size();

        double best_d = std::numeric_limits<double>::infinity();
        std::size_t best_ii = 0;
        double best_tt = 0;
        for (std::size_t j = 0; j < nb; ++j) {
                gte::DCPPoint2Segment2<double> q;
                auto r = q(vertices_a[loop_a[0]], { vertices_b[loop_b[j]], vertices_b[loop_b[(j + 1) % nb]] });
                if (r.distance < best_d) {
                        best_ii = j;
                        best_tt = r.parameter;
                        best_d = r.distance;
                }
        }
```

```
	if (best_d > delta) {
		return false;
	}

	if (std::abs(best_tt - 1.) < EPS) {
		best_ii = (best_ii + 1) % nb;
		best_tt = 0;
	}

	std::size_t i = 0, ii = best_ii;
	double t = 0, tt = best_tt;

	std::array<bool, 2> first_edge_processed = { false, false };
	while (true) {
		gte::Segment2<double> segment;
		segment.p[0] = (1 - t) * vertices_a[loop_a[i]] + t * vertices_a[loop_a[(i + 1) % na]];
		segment.p[1] = vertices_a[loop_a[(i + 1) % na]];
		auto leftover_length = gte::Length(segment.p[1] - segment.p[0]);
		auto total_length = gte::Length(vertices_a[loop_a[(i + 1) % na]] - vertices_a[loop_a[i]]);

		gte::Capsule<2, double> capsule;
		capsule.segment = { (1 - tt) * vertices_b[loop_b[ii]] + tt * vertices_b[loop_b[(ii + 1) % nb]],
vertices_b[loop_b[(ii + 1) % nb]] };
		capsule.radius = delta;

		auto r = _IntersectSegmentAndCapsule2D(segment, capsule);

		if (!r.intersect) {
			return false;
		}

		std::size_t k = r.parameter[0] < EPS && r.numIntersections > 1 ? 1 : 0;

		gte::DCPPoint2Segment2<double> p2s;
		auto p2sr = p2s({ r.point[k][0], r.point[k][1] }, capsule.segment);
		if (p2sr.distance > delta + EPS) {
			return false;
```

```
                }

                if (r.numIntersections == 1 && std::abs(r.parameter[0]) < EPS) {
                        ii = (ii + 1) % nb;
                        tt = 0;
                        first_edge_processed[1] = true;
                } else {
                        if (std::abs(r.parameter[k] - 1) < EPS) {
                                i = (i + 1) % na;
                                t = 0;
                                first_edge_processed[0] = true;
                        } else {
                                t += r.parameter[k] * leftover_length / total_length;
                        }

                        if (std::abs(p2sr.parameter - 1.) < EPS) {
                                ii = (ii + 1) % nb;
                                tt = 0;
                                first_edge_processed[1] = true;
                        } else {
                                tt += p2sr.parameter * gte::Length(capsule.segment.p[1] - capsule.segment.p[0]) /
gte::Length(vertices_b[loop_b[(ii + 1) % nb]] - vertices_b[loop_b[ii]]);
                        }
                }

                if (i == 0 && std::abs(t) < EPS && first_edge_processed[0]) {
                        break;
                }
                if (ii == best_ii && std::abs(tt - best_tt) < EPS && first_edge_processed[1]) {
                        break;
                }
        }

        return _CheckIfRemainderIsSorted(vertices_a, vertices_b, i, ii, best_ii);
}
```