

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

ГОРОДЕЦЬКИЙ МИКОЛА ВАДИМОВИЧ

УДК 004.94

ДИСЕРТАЦІЯ

ГЕОМЕТРИЧНЕ МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ОБ'ЄКТА ПОЛІТОЧКОВИМИ
ПЕРЕТВОРЕННЯМИ НА ОСНОВІ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

122 Комп'ютерні науки
12 Інформаційні технології

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.В. Городецький

Науковий керівник
Сидоренко Юлія Всеволодівна, кандидат технічних наук, доцент

Київ – 2026

АНОТАЦІЯ

Городецький М.В. Геометричне моделювання деформації об'єкта політочковими перетвореннями на основі паралельних обчислень. — Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 «Комп'ютерні науки». — Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2026.

Дисертаційне дослідження присвячене розв'язанню науково-прикладної проблеми геометричного моделювання нелінійних деформацій двовимірних та тривимірних об'єктів шляхом розвитку методу політочкових перетворень та реалізації високоефективних паралельних алгоритмів на гетерогенних обчислювальних системах.

У цифрову епоху, коли фізична реальність дедалі тісніше переплітається з віртуальними моделями, адитивним виробництвом та концепціями Індустрії 4.0, геометричне моделювання стало фундаментальною основою авіакосмічної промисловості, біомедичної інженерії та систем віртуальної реальності. Проте фундаментальна проблема ефективної та точної деформації складних тривимірних об'єктів залишається відкритою. Існуючі методи часто досягають межі обчислювальних можливостей обладнання і виділеного бюджету на нього, коли зіштовхуються з викликами топологічної складності сучасних біонічних форм та вимогами до реального часу обробки масивів даних. Дослідження пропонує удосконалений підхід, що долає розрив між жорсткістю класичних параметричних підходів та непередбачуваністю нейромережевих методів, використовуючи потенціал сучасних архітектур паралельних обчислень.

Актуальність дисертаційного дослідження, присвяченого розвитку методу політочкових перетворень, зумовлена необхідністю подолання розриву між жорсткістю класичних параметричних підходів та непередбачуваністю сучасних нейромережових методів. Дослідження пропонує удосконалений погляд на проблему деформації простору, базуючись на критичному аналізі недоліків існуючих рішень та використовуючи потенціал сучасних архітектур паралельних обчислень. Доцільність роботи підтверджується нагальною потребою у створенні вітчизняних алгоритмічних ядер геометричного моделювання, здатних забезпечити незалежність та конкурентоспроможність у стратегічно важливих галузях науки та виробництва.

Метою дослідження є розробка способів та алгоритмів політочкових перетворень для моделювання нелінійних деформацій двовимірних і тривимірних об'єктів з використанням гетерогенних обчислювальних систем. Об'єктом дослідження є процеси геометричного моделювання деформацій дво- і тривимірних об'єктів.

У роботі проаналізовано сучасні методи деформації геометричних об'єктів, такі як: нейронні імпліцитні підходи: NeRF, Гаусовий сплаттінг, фізично обґрунтовані методи: метод скінчених елементів, модель мас-пружинної системи, системи частинок, а також геометрично деформативні та інтерполяційні методи: політканинні перетворення, політочкові перетворення, побудова інтерполяційних векторних полів та симплексна інтерполяція. Встановлено, що нейронні імпліцитні методи відкривають нові можливості для процесінгу в режимі реального часу, але мають значні обчислювальні витрати та обмеження у збереженні топології, фізично обґрунтовані підходи забезпечують високу точність, але обмежені продуктивністю в реальному часі, тоді як геометрично деформативні методи дають керованість і адаптивність.

Метод політочкових перетворень забезпечує низку вагомих переваг серед інших підходів: його базисом є скінченна множина контрольних точок, що

дозволяє ефективно керувати деформацією цілого об'єкта, а математичний апарат зводиться до систем лінійних рівнянь, роблячи розрахунки простими й прозорими на відміну від «чорного ящика» нейронних методів.

Цей апарат є природно паралельним і добре підходить для реалізації на багатоядерних та гетерогенних архітектурах, однак у вигляді послідовної реалізації на трикутникових сітках із мільйонами трикутників метод стає надто повільним, що й мотивувало розробку нових способів задання геометрії об'єкта та спеціалізованих схем паралелізації обчислень. На основі цього обґрунтовано використання методу політочкових перетворень як базового інструментарію, що має високий потенціал до паралелізації та точності.

Для моделювання двовимірних об'єктів розроблено формалізований полігональний спосіб задання вихідної геометрії. Замість ресурсоємного відслідковування перетинів прямих для кожної точки, об'єкт подається у вигляді ланцюжка відрізків, що утворюють полігон. Це удосконалило спосіб задання геометричного об'єкта при двовимірних політочкових перетвореннях за рахунок введення стека для відслідковування входження прямих при відображенні заданої ламаної, що зберігає топологічну цілісність об'єкта. Експериментально показано, що залежність між кутом твірних прямих у вершині та зміщенням точки має майже лінійний характер без осциляцій, що підтверджує стабільність методу. Щоб з'єднати точки в цілісний об'єкт після деформації, запропоновано застосувати модифікований метод параметричної інтерполяції Гауса. Його вдосконалено завдяки адаптації варіативного параметра α до форми кривої на кожному кроці, що дозволило суттєво (у понад 13 разів для тестових функцій) зменшити похибку інтерполяції.

У тривимірному просторі вирішено ключову проблему втрати однозначності під час трансформації вершин трикутнкової сітки. Вперше запропоновано спосіб представлення 3D-об'єкта, заданого трикутнковою сіткою на основі площин, що перетинаються, який дозволяє зберігати цілісність об'єкта після політочкових

перетворень. Проведено порівняльний аналіз трьох способів задання геометрії об'єкта. Встановлено, що метод представлення вершини як точки перетину площин дотичних трикутників забезпечує найкраще співвідношення між точністю реконструкції геометрії та обчислювальною ефективністю. Для розв'язання перевизначених систем лінійних алгебричних рівнянь, що виникають при перетині більше ніж трьох площин, застосовано метод найменших квадратів та псевдоінверсну матрицю Мура-Пенроуза. Виявлено проблему чисельної нестійкості у вироджених конфігураціях трикутничкової сітки, для розв'язання якої успішно застосовано метод регуляризації Тихонова з експериментально обґрунтованим оптимальним параметром $\lambda = 10^{-6}$.

Для забезпечення роботи алгоритмів у реальному часі з масивами даних до 80 млн полігонів обґрунтовано та реалізовано методи інженерного масштабування у гетерогенному середовищі. Встановлено, що продуктивність багатопотокових реалізацій на центральних процесорах обмежується архітектурою неоднорідного доступу до пам'яті. Розроблена NUMA-орієнтована стратегія з використанням OpenMP, яка дозволила прискорити виконання на 15–30% порівняно з базовою реалізацією.

Подальше радикальне прискорення досягнуто шляхом розробки масово-паралельного алгоритму для графічних процесорів на базі архітектури CUDA. Застосовано концепцію «одна площа — один потік» та проведено глибоку оптимізацію на мікроархітектурному рівні: заміну високолатентних операцій подвійної точності на апаратні інструкції, зниження тиску на регістрову пам'ять та впровадження кешування масиву базисних точок у спільній пам'яті стрімінгового мультипроцесора. Це дозволило переорієнтувати задачу з тієї, що обмежена пропускнуою здатністю пам'яті, на таку, що обмежена швидкістю обчислень, і досягти прискорення у ~ 14.8 разів відносно послідовного виконання на ЦП та у 2.11 разів відносно оптимізованого 192-ядерного кластера.

Практичного впровадження результати дисертаційного дослідження набули в ТОВ «БІ-ХАБ», де розроблене розширення для системи моделювання Blender застосовувалося для високоточної 3D-візуалізації банківських платіжних карток із відтворенням оптичних спецефектів. Отримані візуалізаційні результати використовувалися як еталонні 3D-моделі для виробництва. Загалом, результати апробації свідчать про практичну придатність запропонованих методів до впровадження в системи реального часу. Практична доцільність запропонованих методів визначається тим, що розроблене розширення: а) удосконалює спосіб задання геометричного об'єкта; б) забезпечує збереження цілісності 3D-об'єкта після політочкових перетворень; в) підвищує точність обчислень завдяки адаптації параметра в модифікованій Гаус-інтерполяції до локальної форми кривої; г) оптимізує обчислення політочкових перетворень, скорочуючи час підрахунків при збереженні цілісності геометрії. Сукупність зазначених властивостей має прикладну цінність для задач, де критичними є точність деформації, стабільність топології, цілісності та продуктивність, зокрема для застосувань, наближених до реального часу. Висновки, отримані на основі проведених досліджень, підтверджують доцільність їх використання.

Ключові слова: геометричне моделювання, моделювання деформацій, моделювання, політочкові перетворення, інтерполяція, дискретна модель поверхні, комп'ютерні інформаційні технології, перетворення простору, оптимізація, трикутникові сітки, паралельні обчислення, розподілені системи, інформаційні системи, 3D-графіка, GPU.

ABSTRACT

Horodetskyi M.V. Geometric modeling of object deformation by polypoint transformations based on parallel computing. — Qualifying scientific work as a manuscript.

Dissertation for the degree of Doctor of Philosophy in specialty 122 «Computer Science». — National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2026.

The dissertation research is devoted to solving the scientific and applied problem of geometric modeling of non-linear deformations of two-dimensional and three-dimensional objects by developing the polypoint transformation method and implementing high-performance parallel algorithms on heterogeneous computing systems.

In the digital era, where physical reality is increasingly intertwined with virtual models, additive manufacturing, and Industry 4.0 concepts, geometric modeling has become a fundamental cornerstone of the aerospace industry, biomedical engineering, and virtual reality systems. However, the fundamental problem of efficient and accurate deformation of complex three-dimensional objects remains open. Existing methods, facing the challenges of the topological complexity of modern bionic forms and real-time processing requirements for massive datasets, often reach the limits of hardware computational capabilities and the budgets allocated for them. The research proposes an enhanced approach that bridges the gap between the rigidity of classical parametric approaches and the unpredictability of neural network methods, leveraging the potential of modern parallel computing architectures.

The relevance of the dissertation research, dedicated to the development of the polypoint transformation method, is driven by the need to bridge the gap between the

rigidity of classical parametric approaches and the unpredictability of modern neural network methods. The research offers an enhanced perspective on the problem of space deformation, based on a critical analysis of the shortcomings of existing solutions and leveraging the potential of modern parallel computing architectures. The expediency of the work is confirmed by the urgent need to create national algorithmic kernels for geometric modeling capable of ensuring independence and competitiveness in strategically important fields of science and industry.

The aim of the research is to develop methods and algorithms for polypoint transformations for modeling non-linear deformations of two-dimensional and three-dimensional objects using heterogeneous computing systems. The object of the study is the processes of geometric modeling of deformations of two- and three-dimensional objects.

The paper analyzes modern methods of geometric object deformation, such as: neural implicit approaches (NeRF, Gaussian Splatting); physically-based methods (the finite element method, the mass-spring system model, and particle systems); and geometric deformation and interpolation methods (poly-cloth transformations, polypoint transformations, the construction of interpolating vector fields, and simplex interpolation). It has been established that neural implicit methods open new possibilities for real-time processing but entail significant computational costs and limitations in topology preservation; physically-based approaches provide high accuracy but are limited by real-time performance, whereas geometric deformation methods offer controllability and adaptability.

The polypoint transformation method provides a number of significant advantages over other approaches: its basis is a finite set of control points, which allows for effective control over the deformation of the entire object, and its mathematical apparatus is reduced to systems of linear equations, making the calculations simple and transparent, unlike the "black box" of neural methods.

This apparatus is naturally parallel and well-suited for implementation on multi-core and heterogeneous architectures; however, in the form of a sequential implementation on triangular meshes with millions of triangles, the method becomes excessively slow, which motivated the development of new methods for defining object geometry and specialized computational parallelization schemes. On this basis, the use of the polypoint transformation method is justified as a fundamental toolkit with high potential for parallelization and accuracy.

A formalized polygonal method for specifying initial geometry has been developed for 2D object modeling. Instead of resource-intensive tracking of line intersections for each point, the object is represented as a chain of segments forming a polygon. This improved the method of specifying a geometric object in 2D polypoint transformations by introducing a stack to track line inclusion when mapping a given polyline, which preserves the topological integrity of the object. It is experimentally shown that the relationship between the angle of the generating lines at the vertex and the point displacement is nearly linear without oscillations, confirming the stability of the method. To connect points into a coherent object after deformation, a modified Gaussian parametric interpolation method is proposed. It has been enhanced by adapting a variable parameter α to the curve shape at each step, which significantly reduced the interpolation error (by more than 13 times for test functions).

In three-dimensional space, the key problem of the loss of uniqueness during the transformation of triangular mesh vertices has been resolved. For the first time, a method for representing a 3D object defined by a triangular mesh based on intersecting planes has been proposed, which allows for the preservation of object integrity after polypoint transformations. A comparative analysis of three methods for specifying object geometry has been conducted. It was established that the method of representing a vertex as the intersection point of the planes of tangent triangles provides the best balance between geometry reconstruction accuracy and computational efficiency. To solve the overdetermined systems of linear algebraic equations that arise when more than three

planes intersect, the least squares method and the Moore-Penrose pseudoinverse matrix were applied. A problem of numerical instability in degenerate triangular mesh configurations was identified, for the resolution of which Tikhonov regularization with an experimentally justified optimal parameter was successfully applied.

To ensure real-time algorithm operation with datasets of up to 80 million polygons, engineering scaling methods in a heterogeneous environment have been substantiated and implemented. It was established that the performance of multi-threaded implementations on central processing units is limited by the Non-Uniform Memory Access (NUMA) architecture. A NUMA-aware strategy using OpenMP has been developed, which allowed for a 15–30% acceleration in execution compared to the baseline implementation.

Further radical acceleration was achieved through the development of a massively parallel algorithm for graphics processing units based on the CUDA architecture. The «one plane — one thread» concept was applied, and deep optimization at the microarchitectural level was performed: replacing high-latency double-precision operations with hardware instructions, reducing register pressure, and implementing caching of the basis point array in the shared memory of the streaming multiprocessor. This allowed for the reorientation of the task from being memory-bound to being compute-bound, achieving a speedup of ~14.8 times relative to sequential CPU execution and 2.11 times relative to an optimized 192-core cluster.

The results of the dissertation research have been practically implemented at "BI-HUB" LLC, where the developed extension for the Blender modeling system was applied for high-precision 3D visualization of bank payment cards with the reproduction of optical special effects. The obtained visualization results were used as reference 3D models for production. Overall, the validation results indicate the practical suitability of the proposed methods for implementation in real-time systems. The practical expediency of the proposed methods is determined by the fact that the developed extension: a) improves the method of specifying a geometric object; b) ensures the preservation of 3D

object integrity after polypoint transformations; c) increases calculation accuracy by adapting a parameter in the modified Gaussian interpolation to the local curve shape; d) optimizes the computation of polypoint transformations, reducing calculation time while maintaining geometric integrity. The combination of these properties provides applied value for tasks where deformation accuracy, topological stability, integrity, and performance are critical, particularly for near-real-time applications. The conclusions drawn from the research confirm the expediency of their application.

Keywords: geometric modeling, deformation modeling, modeling, polypoint transformations, interpolation, discrete surface model, computer information technologies, space transformation, optimization, triangular meshes, parallel computing, distributed systems, information systems, 3D graphics, GPU.

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Основні наукові результати дисертації, які опубліковані у періодичних наукових виданнях проіндексованих у базах WoS:

1. Ausheva, N. M., Sydorenko, I. V., Kaleniuk, O. S., Kardashov, O. V., & Horodetskyi, M. V. (2025). Implicit curves and surfaces modeling with pseudogaussian interpolation. *Radio Electronics, Computer Science, Control*, (1), 30–37. <https://doi.org/10.15588/1607-3274-2025-1-3>

Над роботою працювали:

- Аушева Н.М.: аналіз літературних джерел, впорядкування та аналіз даних, редагування тексту.
- Сидоренко Ю.В.: аналіз літературних джерел, впорядкування та аналіз даних, редагування тексту.
- Каленюк О.С.: формулювання гіпотези дослідження, аналіз літературних джерел, розробка методів дослідження з побудови псевдо-гаусівської інтерполяції, впорядкування та аналіз даних, створення графіків та ілюстрацій, написання основного тексту.
- Кардашов О.В.: аналіз літературних джерел, дослідження варіантів використання RBF в процесі псевдо-гаусівської інтерполяції, редагування тексту.
- Городецький М.В.: математична формалізація псевдо-гаусівської функції, дослідження обмежень запропонованого підходу, впорядкування та аналіз даних, створення ілюстрацій, редагування тексту.

Основні наукові результати дисертації, які опубліковані у наукових фахових виданнях України:

1. Sydorenko, Yu. V., Onysko, A. I., Shaldenko, O. V., & Horodetskyi, M. V. (2022). Interpolation of different types of spiral-like curves by gaus-interpolation methods.

Control Systems and Computers, 3(299), 1–10.

<https://doi.org/10.15407/csc.2022.03.003>

Над роботою працювали:

— Сидоренко Ю.В.: сформулювала постановку задачі та теоретично обґрунтував застосування звичайної й параметричних модифікацій Гаус-інтерполяції для спіралеподібних кривих.

— Онисько А.І.: реалізував програмну систему/алгоритми інтерполяції та забезпечив технічну підтримку моделювання і візуалізації результатів

— Шалденко О.В.: виконав комп'ютерний експеримент, підрахунок похибок і підготовку таблиць та графічних візуалізацій результатів.

— Городецький М.В.: і відповідав за постановку дослідження, порівняльний аналіз із методом Лагранжа та практичні рекомендації, зокрема для нерівномірного кроку інтерполяції.

2. Sydorenko, Iu. V., & Horodetskyi, M. V. (2024). Modification of the algorithm for defining polygonal geometry of an object for polypoint transformations. Control Systems and Computers, 4, 12–18. <https://doi.org/10.15407/csc.2023.04.012>

Над роботою працювали:

— Сидоренко Ю.В.: сформулювала постановку задачі та теоретично обґрунтував модифікацію Гаус-інтерполяції через підбір оптимального варіативного параметра α і узагальнив висновки.

— Городецький М.В.: реалізував програмне забезпечення для комп'ютерного експерименту, провів тестування на елементарних функціях і підготував графіки/порівняння похибок для різних значень α та для полінома Лагранжа.

3. Sydorenko, Iu. V., Kaleniuk, O. S., & Horodetskyi, M. V. (2024). Polypoint transformation dependency on the polyfiber configuration. Control Systems and Computers, 4, 3–9. <https://doi.org/10.15407/csc.2024.04.003>

Над роботою працювали:

— Сидоренко Ю.В.: провела обчислювальний експеримент (регулярні

багатокутники й рівнобедрений трикутник), зібрав дані та проаналізував залежність зміщення вершини від кута між формувальними прямими. — Каленюк О.С.: сформулював постановку задачі залежності результату політочкового перетворення від конфігурації політканини та обґрунтував застосування підходу до деформації трикутних сіток.

— Городецький М.В.: реалізував програмну частину моделювання й візуалізації, підготував таблиці/графіки та провів інтерпретацію результатів для висновку про придатність методу в 3D.

4. Horodetskyi, M. V., & Sydorenko, Iu. V. (2025). Methods of defining geometry of an object in three-dimensional space for polypoint transformations. *Elektronnoe modelirovanie*, 47(3), 3–11. <https://doi.org/10.15407/emodel.47.03.003>

Над роботою працювали:

— Городецький М.В.: реалізував програмну частину та провів обчислювальні експерименти з оцінкою точності/швидкодії трьох способів задання вершин трикутної сітки, зокрема із застосуванням псевдооберненої Мура–Пенроуза для неоднозначних перетинів площин.

— Сидоренко Ю.В.: сформулювала постановку задачі деформації трикутної сітки через політочкові перетворення та обґрунтував порівняльний аналіз методів (нормалі, ортогональні площини, площини суміжних трикутників) і підсумкові висновки щодо найкращого балансу точності та швидкодії.

5. Horodetskyi, M. V., & Kaleniuk, O. S. (2025). Parallel implementation of polypoint transformations with adjacent triangle plane intersections. *Elektronnoe modelirovanie*, 47(6), 3–10. <https://doi.org/10.15407/emodel.47.06.003>

Над роботою працювали:

— Городецький М.В.: виконав програмну реалізацію багатопотокового перетворення списку площин, провів експерименти на великій моделі та побудував апроксимаційні моделі залежності часу від кількості потоків.

— Каленюк О.С.: відповідав за постановку дослідження паралелізації,

інтерпретацію результатів через закон Амдала (оцінка частки паралельного коду) та формулювання висновків про межі прискорення.

Матеріали конференцій, які засвідчують апробацію матеріалів дисертації:

1. Сидоренко, Ю. В., Кривда, О. В., & Городецький, М. В. (2022, 20 грудня). Гаус–інтерполяція спіралеподібних кривих [Матеріали конференції]. У Наукові підсумки 2022 року: збірка наукових праць XI наукової конференції (р. 8). Харків: Технологічний центр. <https://ela.kpi.ua/handle/123456789/55490>
2. Сидоренко, Ю. В., & Городецький, М. В. (2024, 4–6 червня). Оптимізація методу розв’язання СЛАР для політочкових перетворень [Тези доповіді]. У Сучасні проблеми геометричного моделювання: тези 26-ї міжнародної науково-практичної конференції (р. 31). Мелітополь, Україна. <https://doi.org/10.33842/2313-125X-2024-13>
3. Сидоренко, Ю. В., & Городецький, М. В. (2025, 3–4 червня). Переваги методу політочкових перетворень порівняно з методами NeRF та Gaussian splatting [Тези доповіді]. У Сучасні проблеми геометричного моделювання: тези 27-ї міжнародної науково-практичної конференції (р. 47). Мелітополь, Україна. <https://magazine.mdpu.org.ua/index.php/spm/issue/download/133/60>

Матеріали, які додатково відображають наукові результати дисертації:

1. Sydorenko, Yu., Zalevska, O., Horodetskyi, M., & Spirintsev, D. (2022). Аналіз поведінки похибки обчислень при Гаус–інтерполяції функції Рунге. Сучасні проблеми моделювання, (23), 159–167. <https://doi.org/10.33842/2313-125X-2023-23-159-167>
2. Sydorenko, Yu., Zalevska, O., Horodetskyi, M., & Naidysh, A. (2022). Особливості розташування базисних вузлів Гаус-функції на прикладі спіралеподібних кривих. Сучасні проблеми моделювання, (23), 151–158. <https://doi.org/10.33842/2313-125X-2022-23>

ЗМІСТ

| | |
|--|----|
| АНОТАЦІЯ | 2 |
| ABSTRACT | 7 |
| СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ | 12 |
| ЗМІСТ | 16 |
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ | 21 |
| ВСТУП..... | 24 |
| РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ У КОМП'ЮТЕРНІЙ ГРАФІЦІ | 30 |
| 1.1 Методи деформації на основі неявних нейронних представлень..... | 30 |
| 1.1.1 Деформація об'єкта на основі нейронних полів випромінювання (NeRF) | 30 |
| 1.1.2 Деформація об'єкта на основі гаусового сплаттінгу | 32 |
| 1.2 Фізично обґрунтовані методи деформації | 35 |
| 1.2.1 Метод скінчених елементів (FEM) | 35 |
| 1.2.2 Модель мас-пружинної системи..... | 37 |
| 1.2.3 Системи частинок..... | 38 |
| 1.3 Метод довільної деформації форми | 41 |
| 1.4 Геометрично деформаційні методи моделювання | 44 |
| 1.4.1 Метод політканинних перетворень | 44 |
| 1.4.2 Метод політочкових перетворень..... | 46 |
| 1.4.3 Метод деформаційного моделювання геометричних об'єктів із використанням двовимірної інтерполяційної векторної поля деформації..... | 50 |
| 1.4.4 Метод побудови двовимірної інтерполяційної векторної поля..... | 52 |

| | |
|---|-----------|
| 1.4.5 Симплексна інтерполяція функції двох змінних для задач деформаційного моделювання..... | 53 |
| Висновки до першого розділу | 57 |
| РОЗДІЛ 2. МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ НА ПЛОЩИНІ | 59 |
| 2.1 Моделі побудови двовимірних об'єктів політочковими перетвореннями.. | 59 |
| 2.2 Полігональний спосіб задання геометрії об'єкта | 62 |
| 2.3 Вплив кута між твірними прямими політканини на перетворення точкового об'єкта | 68 |
| 2.4 Застосування інтерполяційних методів для забезпечення гладкості полігональних моделей | 73 |
| 2.5 Оптимізований алгоритм вибору параметра α функції гаусової інтерполяції | 80 |
| Висновки до другого розділу | 83 |
| РОЗДІЛ 3. МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ У БАГАТОВИМІРНОМУ ПРОСТОРИ..... | 84 |
| 3.1 Побудова тривимірних об'єктів політочковими перетвореннями | 84 |
| 3.2 Способи задання геометрії об'єкта для політочкових перетворень | 86 |
| 3.2.1 Спосіб задання геометрії об'єкта через перетин площин трикутника та його нормалей | 87 |
| 3.2.2 Спосіб задання геометрії об'єкта через перетин ортогональних площин трикутника | 88 |
| 3.2.3 Спосіб задання геометрії об'єкта через перетин площин дотичних трикутників..... | 89 |
| 3.3 Порівняльний аналіз трьох способів задання геометрії об'єкта..... | 91 |
| 3.4 Аналіз обчислювальної складності способів | 99 |
| 3.4.1 Алгоритм задання геометрії об'єкта через перетин площин трикутника та його нормалей | 99 |

| | |
|--|-----|
| 3.4.2 Алгоритм задання геометрії об'єкта через перетин ортогональних площин трикутника | 105 |
| 3.4.3 Алгоритм задання геометрії об'єкта через перетин площин дотичних трикутників | 108 |
| 3.4.4 Асимптотичний аналіз складності способів за нотацією Ландау | 113 |
| 3.5. Методологія вибору параметра регуляризації | 117 |
| 3.5.1 Визначення проблеми чисельної нестійкості у вироджених випадках | 117 |
| 3.5.2 Метод регуляризації для стабілізації розв'язку | 118 |
| 3.5.3 Експериментальне визначення оптимального значення регуляризації λ | 120 |
| 3.5.4 Аналіз результатів та обґрунтування вибору λ | 121 |
| Висновки до третього розділу | 126 |
| РОЗДІЛ 4. РОЗРОБКА МЕТОДІВ ПОЛІТОЧКОВИХ ПЕРЕТВОРЕНЬ НА ОСНОВІ ГЕТЕРОГЕННИХ ОБЧИСЛЕНЬ | 129 |
| 4.1 Багатопотокова паралелізація методу політочкових перетворень | 129 |
| 4.1.1 Архітектура рішення і алгоритм підготовки даних та запуску процесу обробки на багатоядерних CPU | 129 |
| 4.1.2 Експериментальна оцінка масштабованості та моделювання продуктивності паралельної реалізації методу політочкових перетворень | 132 |
| 4.2 Паралельна реалізація методу політочкових перетворень для розподілених систем на базі NUMA-архітектури | 137 |
| 4.2.1 Аналіз вузьких місць базової реалізації на архітектурах з неоднорідним доступом до пам'яті (NUMA) | 138 |
| 4.2.2 Розробка NUMA-орієнтованої стратегії паралелізації з використанням технології OpenMP | 140 |

| | |
|--|-----|
| 4.2.3 Експериментальне дослідження та порівняльний аналіз продуктивності pthreads та OpenMP для політочкових перетворень..... | 142 |
| 4.3 Прискорення методу політочкових перетворень з використанням GPU-архітектури CUDA | 146 |
| 4.3.1 Огляд архітектури та принципів організації паралельних обчислень у середовищі CUDA | 147 |
| 4.3.2 Постановка задачі та обґрунтування моделі паралелізму CUDA | 151 |
| 4.3.3 Реалізація гетерогенного обчислювального процесу для методу політочкових перетворень | 153 |
| 4.4 Дослідження методів оптимізації та підвищення ефективності гетерогенного обчислювального процесу для методу політочкових перетворень | 156 |
| 4.4.1 Оптимізація на рівні інструкцій та аналіз чисельної стабільності | 156 |
| 4.4.2 Підвищення рівня завантаженості стрімінгових мультипроцесорів .. | 159 |
| 4.4.3 Оптимізація підсистеми пам'яті..... | 160 |
| 4.4.4 Експериментальна оцінка продуктивності та порівняльний аналіз ... | 163 |
| Висновки до четвертого розділу | 166 |
| ВИСНОВКИ | 167 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ | 171 |
| ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОГО ДОСЛІДЖЕННЯ..... | 185 |
| ДОДАТОК Б. ЛІСТИНГ ПРОЦЕДУРИ GETPOLYPOINTPLANE | 186 |
| ДОДАТОК В. ЛІСТИНГ ПІДГОТОВКИ ТА ЗАПУСКУ ПРОЦЕДУРИ GETPOLYPOINTPLANE НА ВСІХ ДОСТУПНИХ ЯДРАХ ПРОЦЕСОРА..... | 188 |
| ДОДАТОК Г. ЛІСТИНГ ПІДГОТОВКИ ТА ЗАПУСКУ ПРОЦЕДУРИ GETPOLYPOINTPLANE OPENMP НА КЛАСТЕРІ AMAZON | 194 |

| | |
|--|-----|
| ДОДАТОК Г. ЛІСТИНГ НЕОПТИМІЗОВАНОГО CUDA-ЯДРА ДЕФОРМАЦІЇ ПЛОЩИН МЕТОДОМ ПОЛІТОЧКОВИХ ПЕРЕТВОРЕНЬ | 200 |
| ДОДАТОК Д. ЛІСТИНГ ОПТИМІЗОВАНОГО CUDA-ЯДРА ДЕФОРМАЦІЇ ПЛОЩИН МЕТОДОМ ПОЛІТОЧКОВИХ ПЕРЕТВОРЕНЬ | 205 |
| ДОДАТОК Е. ЛІСТИНГ РОЗШИРЕННЯ ДЛЯ ПРОГРАМНОГО ПАКЕТУ BLENDER | 211 |

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) — набір правил, протоколів та інструментів, що дозволяє різним програмним застосункам взаємодіяти, обмінюватися даними та отримувати доступ до функцій один від одного.

CAD (Computer-Aided Design) — система автоматизованого проєктування.

CAE (Computer-Aided Engineering) — загальна назва програм або програмних пакетів, призначених для інженерних розрахунків.

CUDA (Compute Unified Device Architecture) — програмно-апаратна архітектура паралельних обчислень, розроблена компанією Nvidia.

Embarrassingly parallel — ідеально паралельні обчислення; задачі, які легко розбиваються на незалежні підзадачі без необхідності комунікації між ними.

FEM (Finite Element Method) — метод скінченних елементів.

FFD (Free-Form Deformation) — метод довільної деформації форми.

First-Touch Policy — політика першого дотику; стратегія управління пам'яттю, при якій фізична пам'ять виділяється тому процесору, який першим звертається до неї.

FMA (Fused Multiply-Add) — операція об'єднаного множення—додавання, що виконується за один крок.

FP32 (single-precision Floating Point) — стандартний 32-бітний формат для представлення дійсних чисел у комп'ютерах.

FP64 (double-precision Floating Point) — стандартний 64-бітний формат для представлення дійсних чисел у комп'ютерах.

FPE (Floating Point Exception) — помилка при виконанні операцій з плаваючою комою.

Gaussian Splatting modeling — гаусове сплаттінгове моделювання; метод візуалізації та представлення складних тривимірних об'єктів.

GPGPU (General-Purpose computing on Graphics Processing Units) — техніка використання графічного процесора для загальних обчислень.

Хост (Host) — керуюча сторона гетерогенної обчислювальної системи (ЦП та ОЗП).

LIFO (Last In, First Out) — принцип організації даних «останнім прийшов — першим пішов».

MAE (Mean Absolute Error) — середня абсолютна помилка.

NeRF (Neural Radiance Fields) — нейронне поле випромінювання.

NUMA (Non-Uniform Memory Access) — Архітектура з неоднорідним доступом до пам'яті.

OpenMP (Open Multi-Processing) — набір директив компілятора, бібліотечних процедур та змінних середовища для програмування багатопотокових додатків на багатопроцесорних системах.

PBD (Physics-Based Deformation) — фізично обґрунтоване моделювання деформацій.

RBF (Radial Basis Function) — метод радіальної базисної функції.

RGB (Red Green Blue) — червоний, зелений, синій.

Pthreads (бібліотека) — стандартизований інтерфейс прикладного програмування (API) для створення та керування потоками в процесі в POSIX-сумісних операційних системах, таких як Linux, macOS та Solaris.

RMSE (Root Mean Square Error) — середньоквадратична помилка.

SIMD (Single Instruction, Multiple Data) — архітектура «одна інструкція — багато даних».

SIMT (Single Instruction, Multiple Threads) — архітектура «одна інструкція — багато потоків», що використовується в ГП.

SMP (Symmetric Multiprocessing) — симетрична багатопроцесорність.

SVD (Singular Value Decomposition) — сингулярний розклад матриці; метод, що використовується для розв'язання систем лінійних рівнянь (зокрема з псевдоінверсною матрицею).

Warp Divergence — дивергенція (розбіжність) варпа; ситуація, коли потоки виконання в одному варпі повинні виконати різні гілки коду.

АЛП (Арифметико-логічний пристрій) — блок процесора, що виконує арифметичні та логічні перетворення даних.

Блок потоків (Thread block) — логічна група потоків, які можуть взаємодіяти між собою через швидку спільну пам'ять та бар'єрну синхронізацію.

Варп (Warp) — найменша група потоків (зазвичай 32), які виконуються фізично одночасно на мультипроцесорі.

Глобальна пам'ять (Global Memory) — найбільший за обсягом, але найповільніший тип пам'яті GPU, доступний усім потокам та хосту.

ГОЗП (VRAM, (Video Random Access Memory) — графічно оперативно запам'ятовуючий пристрій; пам'ять пристроя (ГП)..

ГП (GPU, Graphics Processing Unit) — графічний процесор.

ОЗП (RAM, Random Access Memory) — оперативно запам'ятовуючий пристрій; пам'ять хоста (ЦП).

СЛАР — Система лінійних алгебричних рівнянь.

СМ (SM, Streaming Multiprocessor) — стрімінговий мультипроцесор; основний виконавчий блок в архітектурі ГП Nvidia.

Потік (Thread) — базова одиниця виконання, що має власний ідентифікатор, набір регістрів та лічильник команд.

ПТП — Політочкові перетворення.

Сітка (Grid) — сукупність усіх блоків потоків, що виконують одне ядро.

Спільна пам'ять (Shared Memory) — це швидка вбудована пам'ять, доступна всім потокам в одному блоці потоків; явно керується програмістом.

ЦП (CPU, Central Processing Unit) — центральний процесор.

ВСТУП

У цифрову епоху, коли фізична реальність дедалі тісніше переплітається з віртуальними моделями, адитивним виробництвом та концепціями Індустрії 4.0, крилатий вислів Натана Ротшильда «Хто володіє інформацією, той володіє світом» набуває нового, технократичного змісту. У контексті сучасного інжинірингу та науки можна перефразувати цю думку наступним чином: «Хто володіє найшвидшими методами опису та деформації геометричної інформації, той володіє ключем до передового виробництва». Геометричне моделювання більше не є просто допоміжним інструментом креслення; воно стало фундаментом, на якому ґрунтується авіакосмічна промисловість, біомедична інженерія, архітектурне проектування та системи віртуальної реальності [1].

Проте, незважаючи на експоненціальне зростання обчислювальних потужностей, фундаментальна проблема ефективної, точної та інтуїтивно зрозумілої деформації складних тривимірних об'єктів залишається відкритою. Існуючі методи, розроблені в епоху становлення CAD-систем, досягли межі своїх можливостей, при зіткненні з викликами топологічної складності сучасних біонічних форм та вимогами до реального часу обробки масивів даних, що налічують мільйони полігонів.

Актуальність дисертаційного дослідження, присвяченого розвитку методу політочкових перетворень, зумовлена необхідністю подолання розриву між жорсткістю класичних параметричних підходів та непередбачуваністю сучасних нейромережевих методів. Дослідження пропонує удосконалений погляд на проблему деформації простору, базуючись на критичному аналізі недоліків існуючих рішень та використовуючи потенціал сучасних архітектур паралельних обчислень. Доцільність роботи підтверджується нагальною потребою у створенні вітчизняних алгоритмічних ядер геометричного моделювання, здатних забезпечити

незалежність та конкурентоспроможність у стратегічно важливих галузях науки та виробництва.

Метою дослідження є розробка способів та алгоритмів політочкових перетворень для моделювання нелінійних деформацій двовимірних і тривимірних об'єктів з використанням гетерогенних обчислювальних систем.

Для досягнення поставленої мети необхідно розв'язати наступні **задачі**:

- Проаналізувати геометричні деформаційні методи та обґрунтувати вибір політочкових перетворень як базового інструмента моделювання нелінійних деформацій.
- Розробити математичні моделі побудови двовимірних геометричних об'єктів на площині на основі політочкових перетворень та дослідити полігональний спосіб задання геометрії об'єкта на якість та стійкість деформації.
- Розробити та дослідити інтерполяційні методи для підвищення гладкості полігональних каркасів геометричного об'єкта, та удосконалити спосіб Гаус-інтерполяції за рахунок адаптації варіативного параметру для зменшення похибок деформації.
- Розробити спосіб представлення 3D-об'єкта, заданого трикутною сіткою на основі площин, який дозволяє зберігати цілісність об'єкта при політочкових перетвореннях.
- Розробити багатопотокові паралельні алгоритми реалізації методу політочкових перетворень для багатоядерних ЦП, спроектувати NUMA-орієнтовану стратегію паралелізації та виконати експериментальне порівняння реалізацій на основі pthreads та OpenMP.

Об'єктом дослідження є процеси геометричного моделювання деформацій дво- і тривимірних об'єктів.

Предметом дослідження є методи полікоординатних перетворень для просторового моделювання деформацій дво- і тривимірних об'єктів.

Були використані наступні **методи дослідження**:

- Метод політочкових перетворень — застосовано для дослідження відображення деформаційних змін двовимірних та тривимірних об'єктів шляхом перетворення прямих та площин.
- Метод інтерполяції Гауса — застосовано для дослідження побудови інтерполяційної кривої за дискретними вузлами.
- Метод Жордана–Гауса — використано як чисельний метод розв'язання СЛАР, що виникає в гаусовій інтерполяції.
- Метод найменших квадратів і псевдообернення Мура–Пенроуза — використано для дослідження реконструкції вершини, коли точка перетину перетворених площин не існує.
- Метод регуляризації Тихонова — застосовано для стабілізації розв'язку в погано обумовлених/вироджених випадках через модифікацію системи додаванням λ .
- NUMA-орієнтовані методи оптимізації на CPU — застосовано керування прив'язкою потоків і ідіому «першого дотику».
- GPGPU/CUDA підхід до прискорення — проведено огляд архітектури CUDA та гетерогенної моделі Host/Device як основи обчислень загального призначення на GPU.

Наукова новизна отриманих результатів полягає в:

- Вперше запропоновано спосіб представлення об'єкта перетворень, який враховує топологію дискретного представлення, що дозволяє здійснювати політочкові перетворення поверхонь заданих полігональними сітками.
- Удосконалено спосіб задання геометричного об'єкта при двовимірних політочкових перетвореннях за рахунок введення стека для відслідковування входження прямих при відображенні заданої ламаної, що зберігає топологічну цілісність об'єкта.

- Удосконалено спосіб Гаус-інтерполяції за рахунок адаптації варіативного параметру α до форми кривої на кожному кроці, що дозволило підвищити точність обчислень.
- Удосконалено спосіб обрахунку політочкових перетворень за рахунок розпаралелювання обчислень, що призвело до значного скорочення часу підрахунків при збереженні цілісності об'єкта.

Практичне значення одержаних результатів роботи полягає у розробленні та впровадженні розширення до Blender (Додаток Е), яке забезпечує збереження топологічної цілісності 3D об'єктів під час політочкових перетворень зокрема завдяки представленню трикутничкової сітки через перетин площин дотичних трикутників, підвищує точність обчислень шляхом адаптації параметра α у Гаус-інтерполяції та скорочує час підрахунків завдяки застосуванню гетерогенних обчислень. *Впровадження* результатів роботи здійснено у ТОВ «БІ-ХАБ». Розширення використано для високоточної 3D-візуалізації банківських карток з оптичними спецефектами та отримання еталонних моделей для виробництва, що порівняно зі стандартними підходами Blender зменшило обчислювальні витрати, прискорило рендеринг і оптимізувало використання ОЗП. Впровадження підтверджено «Актом впровадження» від 29 січня 2026 року (Додаток А).

Особистий внесок здобувача у статті [2] полягає в систематизації огляду літератури, формуванні мотивації та обмежень запропонованого підходу, а також у підготовці пояснень, оформленні рисунків і формулюванні підсумкових практичних висновків. У статті [3] полягає в постановці дослідження, виконанні порівняльного аналізу та формуванні практичних рекомендацій, зокрема щодо нерівномірного кроку інтерполяції й вибору оптимального методу для різних спіралей. У статті [4] полягає у формулюванні постановки задачі та теоретичного обґрунтування полігонального подання об'єкта для подальшого застосування методу політочкових перетворень і узагальненні висновків щодо переваг. У статті [5] полягає в постановці та проведенні обчислювальних експериментів

(регулярні багатокутники й рівнобедрений трикутник), зборі та аналізі даних залежності деформації від кута між формувальними лініями і формулюванні практичного висновку про придатність підходу для 3D-деформацій. У статті [6] полягає в реалізації та експериментальній перевірці трьох способів задання геометрії трикутної сітки для політочкових перетворень. У статті [7] полягає в постановці задачі та теоретичному обґрунтуванні підходу з перетином площин суміжних трикутників, інтерпретації результатів і формуванні висновків щодо ефективності паралелізації.

Апробація результатів дисертаційної роботи проводилася в Національному технічному університеті України «Київський політехнічний інститут ім. Ігоря Сікорського» (Навчально-науковий інститут атомної і теплової енергетики, кафедра цифрових технологій в енергетиці) під час розроблення та тестування програмного розширення до комплексу Blender. Практична апробація та впровадження отриманих результатів здійснювалися в ТОВ «БІ-ХАБ», де створене розширення застосовувалося для формування високоточної анімаційної 3D-візуалізації платіжних банківських карток із відтворенням оптичних спецефектів; отримані візуалізаційні результати використовувалися як еталонні 3D-моделі для подальшого виробництва.

Основні положення і практичні результати дисертаційної роботи доповідалися та одержали схвалення на таких конференціях:

1. Наукові підсумки 2022 року: збірка наукових праць XI наукової конференції (р. 8). Харків: Технологічний центр;
2. Сучасні проблеми геометричного моделювання: тези 26-ї міжнародної науково-практичної конференції (р. 31). Мелітополь, Україна;
3. Сучасні проблеми геометричного моделювання: тези 27-ї міжнародної науково-практичної конференції (р. 47). Мелітополь, Україна.

Публікації. Основні результати дисертаційної роботи викладено в 11 наукових роботах з них: 6 публікацій у наукових фахових виданнях України

категорії «Б» (одна з яких проіндексована у наукометричній базі Web of Science), 3 роботи у працях і матеріалах міжнародних наукових конференцій, 2 публікації, які додатково висвітлюють дослідження.

Структура і обсяг роботи. Дисертація складається з вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Повний обсяг дисертації складає 216 сторінок, в тому числі: 147 сторінки основного тексту, 67 рисунків і 10 таблиць, список використаних джерел зі 122 найменувань і 7 додатків

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ У КОМП'ЮТЕРНІЙ ГРАФІЦІ

Моделювання деформаційних процесів є критично важливим завданням у сферах віртуальної реальності, інженерного проєктування та медицини [8], де ключовою вимогою є баланс між візуальною достовірністю та швидкодією в реальному часі. Існуючі підходи поділяються на три основні класи: методи на основі неявних нейронних представлень [9, 10, 11, 12], що забезпечують високий реалізм; фізично обґрунтовані методи (FEM) [13, 14, 15, 16, 17], які гарантують точність симуляції, але є ресурсомісткими; та геометричні методи, що пропонують ефективний компроміс між швидкістю обчислень та контрольованістю форми.

У цьому розділі здійснено порівняльний аналіз зазначених методів з метою обґрунтування вибору оптимального підходу для розв'язання задач деформації швидко і точно. Основний акцент зроблено на геометричному моделюванні, зокрема методах політочкових перетворень та побудові інтерполяційних векторних полів, які демонструють значний потенціал для створення швидких алгоритмів, придатних до ефективного розпаралелювання на сучасних обчислювальних архітектурах.

1.1 Методи деформації на основі неявних нейронних представлень

1.1.1 Деформація об'єкта на основі нейронних полів випромінювання (NeRF)

Нейронне поле випромінювання (NeRF) є сучасним методом неявного нейронного представлення тривимірних геометричних об'єктів, що дозволяє моделювати їхню форму та візуальні характеристики як неперервні функції

просторових координат та напрямів спостереження [9]. Завдяки параметризації нейронними мережами, NeRF здатен з високою точністю фіксувати складні геометричні структури та особливості їхнього візуального сприйняття.

При застосуванні методу NeRF для моделювання деформаційних змін об'єктів використовуються додаткові параметри кондиціонування, такі як поля деформацій або латентні коди, що дозволяють описувати гладкі та складні геометричні перетворення об'єктів без необхідності явних маніпуляцій з сітками. У загальному випадку функцію деформації NeRF можна записати у вигляді:

$$f_{\text{deform}}(x, d, z) : \mathbb{R}^3 \times \mathbb{R}^2 \times \mathbb{R}^n \rightarrow \mathbb{R}^4,$$

де x — вектор просторових координат точки об'єкта;

d — напрямок спостереження;

z — додатковий латентний вектор або поле деформації, що відповідає за контроль геометричних перетворень;

\mathbb{R}^3 — просторові координати точки в 3D;

\mathbb{R}^2 — напрямок спостереження;

\mathbb{R}^n — латентні вектори або параметри поля деформації;

\mathbb{R}^4 — результат, що містить параметри випромінювання та кольору (інтенсивність випромінювання та RGB).

Для здійснення деформації, модель NeRF навчена оптимізувати функцію випромінювання σ та кольору c , залежно від умов деформації z [11]:

$$(\sigma, c) = F_{\theta}(\mathbf{x}', \mathbf{d}), \quad \mathbf{x}' = \mathbf{x} + D_{\phi}(\mathbf{x}, \mathbf{z}),$$

де F_{θ} — нейронна мережа з параметрами θ , яка апроксимує нейронне поле випромінювання;

D_ϕ — нейронна мережа з параметрами ϕ , яка визначає поле деформації об'єкта;
 z — зазвичай описує певний стан або часовий момент об'єкта.

Параметри ϕ поля деформації навчаються таким чином, щоб мінімізувати різницю між прогнозованим і фактичним виглядом деформованого об'єкта за формулою:

$$\mathcal{L}(\theta, \phi) = \sum_{r \in R} |C(r) - \hat{C}(r, \theta, \phi)|^2,$$

де R — множина променів, використаних для навчання;

$C(r)$ — реальний колір променю;

$\hat{C}(r, \theta, \phi)$ — прогнозований колір, що визначається інтегруванням поля випромінювання вздовж променю з урахуванням деформації.

Використання NeRF для моделювання деформацій забезпечує високу гнучкість і точність у представленні складних змін форми геометричних об'єктів, роблячи його перспективним інструментом для застосування у галузях, що потребують деталізованого і реалістичного відображення динамічних змін об'єктів [12]. Однак, основними недоліками методу є значні обчислювальні витрати та труднощі в забезпеченні рендерингу в реальному часі, особливо для великих або високодеталізованих об'єктів або сцен.

1.1.2 Деформація об'єкта на основі гаусового сплаттінгу

Гаусове сплаттінгове моделювання (від англ. Gaussian splatting modeling) є сучасною методикою візуалізації та представлення складних тривимірних об'єктів, що ґрунтується на використанні дискретних гаусових функцій, розміщених у

ключових точках об'єкта. На відміну від традиційних методів, що базуються на полігональних сітках, гаус-сплаттінгове моделювання забезпечує монотонну, неперервну деформацію шляхом зміни параметрів цих функцій, зокрема позиції, масштабу й орієнтації. Завдяки цьому підходу стає можливим моделювати надзвичайно складні й деталізовані трансформації форми об'єктів з мінімальними обчислювальними потужностями.

Метод гаусового сплаттінгу представляє об'єкт як множину тривимірних гаусових функцій [10], що визначаються наступним чином:

$$G(x) = \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right),$$

де x — точка простору;

μ — центр гаусової функції, що відповідає опорній точці об'єкта;

Σ — матриця коваріації, що визначає масштаб та орієнтацію гаусіани.

Процес деформації об'єкта здійснюється шляхом зміни параметрів μ та Σ , що відповідає просторовим переміщенням, масштабуванню та обертанням окремих частин об'єкта.

У процесі моделювання гаусове сплаттінгове подання дозволяє точно контролювати геометричні зміни за рахунок маніпуляції такими параметрами:

1. позиція (μ): визначає переміщення точки в просторі, безпосередньо впливаючи на форму об'єкта;
2. масштаб (коваріація Σ): керує ступенем розмиття та охоплення, впливаючи на ступінь локальності чи глобальності деформації;
3. орієнтація (власні вектори матриці Σ): дозволяє здійснювати обертання гаусових функцій, що важливо для реалізації складних анізотропних деформацій.

Деформацію, представлену через гаусове сплаттінгове моделювання, можна описати як відображення [10]:

$$F(x) = \sum_{i=1}^N w_i G_i(x),$$

де w_i — вагові коефіцієнти, що задають значущість кожної гаусової функції в загальному представленні форми об'єкта;

N — загальна кількість точок-гаусіанів.

В процесі деформації ці вагові коефіцієнти можуть також змінюватися, що дозволяє зберігати точні геометричні деталі під час трансформації.

Серед вагомих переваг цього методу слід відзначити: збереження дрібних геометричних деталей завдяки монотонним трансформаціям; ефективна реалізація в реальному часі, що важливо для інтерактивних застосувань; висока сумісність із методами машинного навчання для автоматизації складних деформаційних задач.

Однак, метод має певні обмеження, зокрема: складність попереднього налаштування параметрів гаусових функцій для точної відповідності реальним об'єктам; висока обчислювальна складність при значному збільшенні кількості опорних точок (гаусіан).

Порівняно з методом гаусового сплатінгу, який моделює геометрію явно через дискретні гаусові ядра, NeRF має перевагу у моделюванні неперервних і складних деформацій, що дозволяє йому більш точно передавати нелінійні зміни форми. Водночас гаусовий сплатінг краще підходить для швидкого рендерингу у реальному часі та забезпечує вищий ступінь збереження тонких геометричних деталей.

1.2 Фізично обґрунтовані методи деформації

Фізично обґрунтоване моделювання деформацій (PBD) ґрунтується на законах механіки неперервних середовищ, що дозволяє досягати високої фізичної достовірності поведінки об'єкта при дії зовнішніх та внутрішніх сил. Основною ідеєю таких методів є розв'язання рівнянь руху або рівноваги з урахуванням матеріальних властивостей середовища — зокрема, модулів Юнга, коефіцієнтів Пуассона, густини, жорсткості та інших параметрів. Результатом є деформована форма об'єкта, яка узгоджується з фізичними законами збереження імпульсу, енергії тощо.

1.2.1 Метод скінчених елементів (FEM)

Найбільш точним методом для симуляції деформацій є метод скінчених елементів (від англ. Finite Element Method, FEM), який передбачає розбиття об'єкта на малорозмірні елементи (тетраедри, куби, призми), у межах яких розв'язується рівняння рівноваги [15]. FEM розглядає об'єкт як пружне тіло, що перебуває у рівновазі під дією зовнішніх сил. Деформація визначається прикладеними зусиллями та фізичними властивостями матеріалу. При досягненні мінімуму потенціальної енергії об'єкт переходить у стан рівноваги. Потенціальна енергія деформованого об'єкта визначається як:

$$E_p = E_s - W,$$

де E_s — енергія деформації, що накопичується в об'єкті у вигляді внутрішньої енергії;

W — робота зовнішніх сил.

Мінімізація функціоналу приводить до виникнення диференціальних рівнянь рівноваги. Оскільки їх аналітичне розв'язання, як правило, неможливе, використовуються чисельні методи. Процедура обчислення деформації об'єкта методом FEM включає такі етапи:

1. виведення рівняння рівноваги шляхом диференціювання функціонала потенціальної енергії за переміщенням матеріалу в об'єкті;
2. розбиття об'єкта на дискретні елементи (рис. 1.1);
3. вибір інтерполяційних функцій для елементів, які апроксимують деформаційне поле;
4. для кожного елемента представити компоненти рівняння рівноваги через інтерполяційні функції цього елемента та відповідні переміщення його вузлів;
5. об'єднання рівнянь рівноваги всіх елементів у єдину систему та розв'язання її для знаходження переміщень усього об'єкта;
6. використання переміщень вузлів та інтерполяційних функцій конкретного елемента для обчислення переміщень внутрішніх точок цього елемента, а отже — і визначення деформованих координат точок об'єкта.

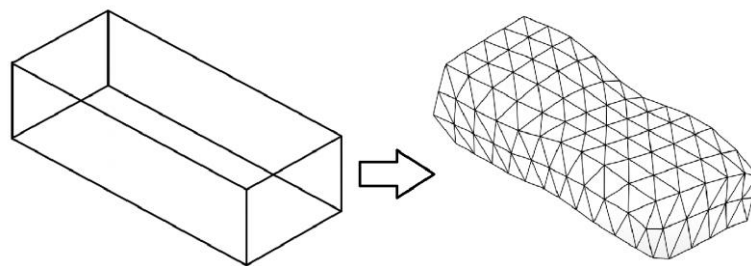


Рис. 1.1. Представлення об'єкта у вигляді скінченноелементної моделі

У матричній формі це рівняння набуває вигляду:

$$Ku = R,$$

де K — глобальна матриця жорсткості;

u — вектор переміщень вузлів;

R — вектор зовнішніх сил.

1.2.2 Модель мас-пружинної системи

Один з найпростіших і найраніше використаних підходів у PBD — модель мас-пружин, у якій об'єкт апроксимується системою точкових мас, з'єднаних ідеальними пружинами (рис. 1.2). Стан кожної точки описується її положенням $x_i \in \mathbb{R}^3$, швидкістю v_i , масою m_i , коефіцієнтом демпфування δ_i , силою зовнішнього впливу f_i та внутрішніми пружними зусиллями g_{ij} від сусідніх точок [16-17].

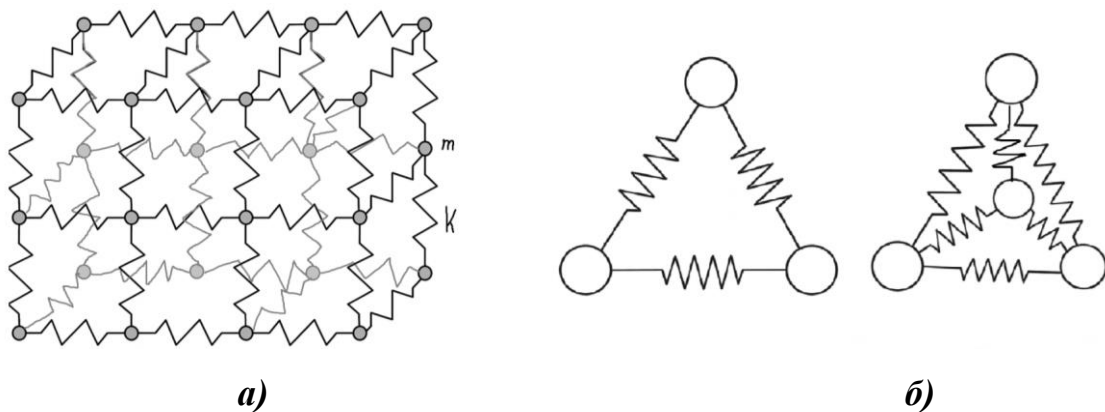


Рис. 1.2. Мас-пружинне представлення деформованого об'єкта.

- а) Фрагмент мас-пружинної моделі. Пружини, що з'єднують масові точки, чинять сили на суміжні масові точки, коли одну з них зміщено відносно положення рівноваги. б) Два типові варіанти з'єднання між масовими точками.

Динаміка системи описується другим законом Ньютона у вигляді:

$$m_i \frac{d^2 x_i}{dt^2} = -\delta_i \frac{dx_i}{dt} + \sum_j g_{ij} + f_i,$$

де m_i — маса точки i ;

x_i — положення точки i ;

δ_i — коефіцієнт демпфування;

g_{ij} — сила пружини між точками i та j ;

f_i — зовнішня сила, прикладена до точки i ;

t — час.

Для всієї системи мас складається система зчеплених звичайних диференціальних рівнянь другого порядку:

$$M \frac{d^2 x}{dt^2} + \Delta \frac{dx}{dt} + Kx = f,$$

де M — масова матриця;

Δ — матриця демпфування;

K — матриця жорсткості;

x — вектор положень точок;

f — вектор зовнішніх сил;

t — час.

Мас-пружинні моделі мають меншу точність, ніж FEM, але дозволяють досягти інтерактивної швидкодії і тому широко використовуються у моделюванні деформацій в режимі реального часу — наприклад, для тканин, м'язів обличчя.

1.2.3 Системи частинок

Ще одним варіантом фізичного моделювання є використання систем частинок, у яких кожна точка об'єкта моделюється як маса, що взаємодіє з іншими

через потенціальні сили [18]. Наприклад, використовується потенціал Леннарда-Джонса:

$$\phi_{LJ}(r) = \frac{B}{r^n} - \frac{A}{r^m},$$

де r — відстань між частинками;

B, A — коефіцієнти, що визначають інтенсивність відштовхування та притягання відповідно;

n, m — показники ступеня, що контролюють спад потенціалу з відстанню.

$$f(r) = -\frac{d\phi_{LJ}}{dr},$$

де $f(r)$ — сила взаємодії частинок на відстані r ;

ϕ_{LJ} — потенціал Леннарда-Джонса;

$\frac{d\phi_{LJ}}{dr}$ — похідна потенціалу, що визначає напрям і величину сили.

Перевагами таких моделей є висока гнучкість, здатність до моделювання нечітких, флуктуючих або частково взаємодіючих об'єктів (дим, полум'я, руйнування м'яких тіл), а також можливість включення нелінійних міжчастинкових взаємодій. Серед недоліків — низька точність у задачах моделювання пружної деформації твердих тіл, висока обчислювальна складність при великій кількості частинок та труднощі зі збереженням стабільності чисельного розрахунку.

Окрему увагу серед інженерно-прикладних реалізацій привертає рушій NVIDIA PhysX — бібліотека фізичної симуляції, яка активно застосовується у відеоіграх та віртуальних середовищах [19]. PhysX підтримує мас-пружинні системи, позиційно-орієнтовану динаміку та інші спрощені моделі для симуляції

тканин, частинок, м'яких тіл та фрагментації об'єктів. Завдяки GPU-акселерації, PhysX дозволяє виконувати симуляцію в реальному часі з високою стабільністю та прийнятною візуальною достовірністю, що робить його ефективним інструментом для інтерактивної графіки попри спрощення фізичних моделей.

Методи фізично обґрунтованого моделювання деформацій забезпечують високу фізичну достовірність поведінки об'єктів завдяки врахуванню матеріальних властивостей і дії зовнішніх сил. Серед них метод скінчених елементів (FEM) є найточнішим, однак потребує значних обчислювальних ресурсів і краще підходить для статичних або квазістатичних задач з малими деформаціями.

Мас-пружинні моделі поступаються в точності, але дозволяють ефективно реалізовувати симуляції в реальному часі завдяки простоті обчислень. Системи частинок забезпечують високу гнучкість для моделювання неklasичних середовищ — диму, полум'я, тканин — але обмежені у відтворенні точних пружних ефектів.

Інженерні середовища часто використовують комбіновані підходи, де спрощені фізичні моделі інтегруються з геометричними або візуальними методами. Прикладом є рушій NVIDIA PhysX, який реалізує позиційно-орієнтовану динаміку та GPU-прискорення для інтерактивного моделювання тканин, уламків, частинок тощо. Попри обмеження в точності, PhysX дозволяє забезпечити візуальну правдоподібність у режимі реального часу.

Таким чином, вибір методу фізично обумовленого моделювання визначається компромісом між точністю, швидкодією та вимогами до інтерфейсу користувача. Необхідність таких методів є особливо важливою для медичних симуляторів, анімації складних матеріалів, деформації тіл у динаміці та реалістичної візуалізації фізичних явищ у комп'ютерній графіці.

1.3 Метод довільної деформації форми

Метод довільної деформації форми FFD, запропонований Седербергом і Перрі [20], є універсальним інструментом для геометричної деформації об'єктів у комп'ютерній графіці. Основна ідея полягає у зануренні об'єкта в тривимірну ґратку контрольних точок. Деформація відбувається шляхом зміни положення цих контрольних точок, що викликає згладжену деформацію об'єкта в усьому об'ємі ґратки. FFD є методом геометричного моделювання, який можна класифікувати як тензорний метод, що базується на поліномах Бернштейна або B-сплайнах [21].

Геометрична побудова FFD

FFD базується на тензорному добутку базисних функцій, які визначають форму деформаційного об'єму. Нехай об'єкт занурено в паралелепіпед, визначений трійкою векторів \vec{U} , \vec{V} , \vec{W} і точкою початку координат \vec{O} . Будь-яка точка \vec{P} всередині цього об'єму може бути представлена у локальній системі координат як:

$$\vec{P} = \vec{O} + u\vec{U} + v\vec{V} + w\vec{W},$$

де $(u, v, w) \in [0, 1]^3$ — нормалізовані координати точки всередині деформаційного об'єму.

Локальні координати визначаються через векторний добуток:

$$u = \frac{(\vec{V} \times \vec{W}) \cdot (\vec{P} - \vec{O})}{(\vec{V} \times \vec{W}) \cdot \vec{U}}$$

$$v = \frac{(\vec{W} \times \vec{U}) \cdot (\vec{P} - \vec{O})}{(\vec{W} \times \vec{U}) \cdot \vec{V}}$$

$$w = \frac{(\vec{U} \times \vec{V}) \cdot (\vec{P} - \vec{O})}{(\vec{U} \times \vec{V}) \cdot \vec{W}}.$$

де P — довільна точка всередині деформаційного об'єму;

O — точка початку, від якої відкладені вектори U, V, W ;

U, V, W — вектори ребер паралелепіпеда;

u, v, w — нормалізовані локальні координати точки P у базисі $\{U, V, W\}$.

Формула FFD деформації

Після переміщення контрольних точок $\vec{P}_{i,j,k'}$, нове положення будь-якої точки \vec{P} визначається за допомогою тензорного добутку тривимірних поліномів Бернштейна:

$$\vec{F}(u, v, w) = \sum_{i=0}^l \sum_{j=0}^m \sum_{k=0}^n B_i^l(u) B_j^m(v) B_k^n(w) \cdot \vec{P}_{i,j,k'},$$

де $B_i^l(u) = \binom{l}{i} (1-u)^{l-i} u^i$ — поліном Бернштейна ступеня l ;

$\vec{P}_{i,j,k'}$ — нові положення контрольних точок після деформації;

добуток $B_i^l(u) B_j^m(v) B_k^n(w)$ — визначає вагу впливу точки $\vec{P}_{i,j,k'}$ на точку \vec{P} .

Ця формула гарантує гладке, глобальне перетворення об'єкта в залежності від змін локальної ґратки.

Властивості та розширення методу FFD

Метод довільної деформації форми має ряд важливих властивостей, що обумовлюють його широке застосування у задачах комп'ютерної графіки та візуалізації. До основних переваг даного методу належать:

1. можливість інтуїтивного керування деформацією об'єкта шляхом зміщення контрольних точок ґратки;

2. забезпечення глобальної або локальної дії деформації на об'єкт, що досягається завдяки регулюванню розміру та щільності контрольної решітки;
3. універсальність застосування до різних типів геометричного представлення об'єктів: полігональні мережі, параметричні поверхні та неявно задані об'єкти.

Разом із тим, класичний варіант FFD має певні обмеження, головним з яких є жорстка топологія контрольної ґратки (у вигляді прямокутного паралелепіпеда), що ускладнює його застосування до об'єктів складної конфігурації. Для усунення цього недоліку було запропоновано низку модифікацій базового методу:

1. EFFD (від англ. Extended Free-Form Deformation) розширений метод FFD, у якому замість паралелепіпеда використовується решітка довільної форми [22];
2. DFFD (від англ. Dirichlet FFD) — узагальнення класичного методу на випадок розсіяних даних за допомогою діаграм Вороного та тріангуляції Делоне, що забезпечує локальність впливу та гнучке управління формою [23];
3. AFFD (від англ. Animated FFD) — застосування анімаційного підходу на основі ключових кадрів, що дозволяє моделювати часову динаміку деформації [24];
4. B-spline FFD — реалізація FFD з використанням базису В-сплайнів замість поліномів Бернштейна, що дозволяє зменшити глобальний вплив зміщення окремої контрольної точки та покращити локальний контроль [25].

FFD широко застосовується у: комп'ютерній анімації — модифікація скелетних сіток та об'єктів персонажів, індустріальному проектуванні, CAD-системах — деформація твердих тіл при симуляції навантажень, медичному моделюванні — симуляції м'яких тканин (зокрема у віртуальній хірургії), текстурному морфінгу та 3D скульптурі. FFD також може бути поєднано з фізично обґрунтованими методами (наприклад, масово-пружинними моделями), при цьому контрольні точки FFD можуть бути масами, що обчислюються динамічно.

З огляду на складність і багатовимірність задач геометричного моделювання деформацій, виникає потреба у використанні підходів, що базуються на керованих

геометричних перетвореннях простору. Одним із перспективних напрямів є політочкові перетворення, які дозволяють здійснювати деформацію об'єктів через зміну просторового розташування множини контрольних точок без обмеження на топологію решітки.

1.4 Геометрично деформаційні методи моделювання

1.4.1 Метод політканинних перетворень

Політканинні перетворення становлять окремий клас нелінійних геометричних перетворень, що використовуються для моделювання деформацій геометричних об'єктів [26, 27]. Їх суть полягає у зміні простору, в якому розміщено об'єкт, за допомогою спеціально сконструйованої структури — політканини. Політканина визначається як сукупність прямих у площині або площин у просторі, що формують координатну систему, в межах якої кожній точці можна приписати набір політканинних координат [28]. Кількість таких координат залежить від розмірності й густини самої політканини.

У процесі моделювання об'єкт занурюється у простір, визначений політканинним базисом (рис. 1.3). Для кожної точки такого об'єкта, окрім декартових координат, додатково фіксується її відстань до кожної з координатних функцій — політканинні координати. Подальша деформація відбувається шляхом контрольованої зміни орієнтації координатних функцій політканини. В результаті цього змінюються й декартові координати точок, що перебувають у межах зміненого простору (рис. 1.4).

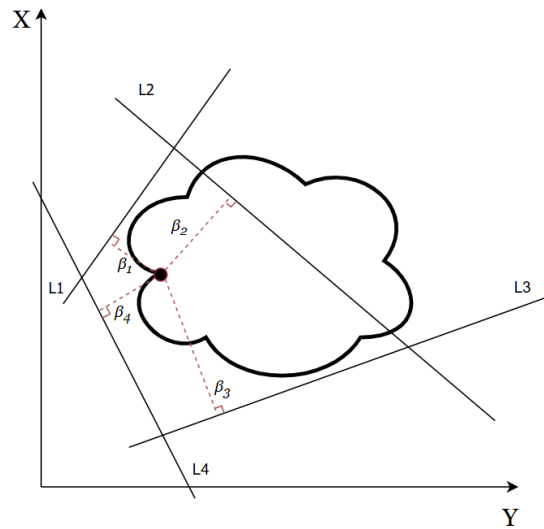


Рис. 1.3. Політканинні координати точки занурені у початковий базис

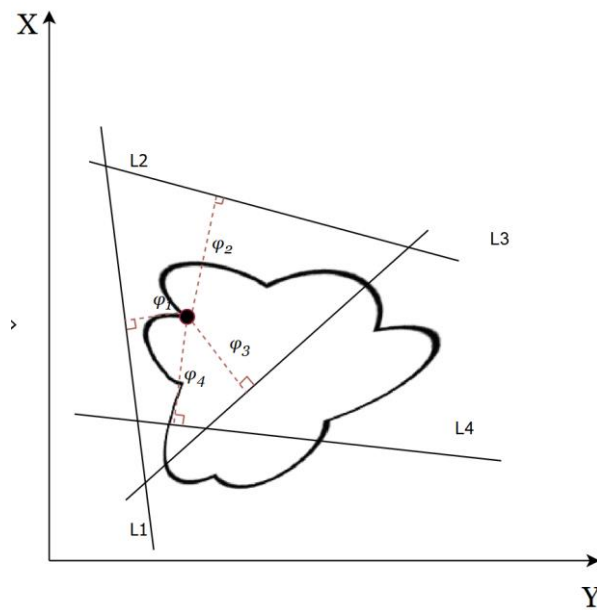


Рис. 1.4. Політканинні координати точки занурені у перетвореному базисі

Метод дозволяє реалізовувати як глобальні, так і локальні перетворення, в залежності від конфігурації координатного базису. Визначальною особливістю є відсутність необхідності явного завдання функції деформації — зміна геометрії об'єкта досягається опосередковано, через зміну конфігурації базису політканини.

Політканинні перетворення знайшли застосування у задачах конструювання плоских кривих, зокрема при використанні багатобазисного підходу, який дозволяє забезпечити неперервність похідних різних порядків на ділянках кривої. Це робить метод конкурентоспроможним порівняно зі сплайн-моделюванням, зокрема у випадках, де потрібне збереження або контроль над характером зміни кривини.

До практичних переваг методу відносяться:

1. можливість опису складних деформацій без явної параметризації об'єкта;
2. гнучкість у зміні координатної структури простору;
3. використання у зворотних задачах (коли задані прообраз і образ, а перетворення — невідоме).

Разом з тим, метод має ряд суттєвих обмежень. Зокрема, при збігові точки з точкою на координатній лінії або вузлом політканини її координатне представлення може містити нулі, що унеможлиблює однозначне визначення її положення після перетворення. Окрім того, складність побудови координатного базису та математичної інтерпретації політканин створює бар'єр для користувачів, які не мають достатньої підготовки у галузі прикладної геометрії. Це спонукає до розробки альтернативних підходів — наприклад, політочкових перетворень, де базис задається множиною контрольних точок.

Таким чином, метод політканинних перетворень є ефективним інструментом геометричного моделювання, особливо в задачах деформації, що вимагають гнучкого керування просторовою структурою, проте вимагає спеціалізованих математичних підходів для практичного застосування та автоматизації процесів.

1.4.2 Метод політочкових перетворень

Розвиток ідеї політканинних перетворень, зокрема застосування до них принципу двоїстості, призвів до формулювання нового класу перетворень — політочкових [29, 30, 31]. Основна відмінність цього підходу полягає в тому, що

координатний базис простору визначається не множиною функцій, а скінченною множиною точок, тоді як об'єктом перетворення виступає пряма на площині, а у тривимірному просторі — площина.

На площині політочковий базис задається набором точок з координатами (x_i^{Π}, y_i^{Π}) , де $i=1,2\dots p$. У цьому базисі пряма-прообраз визначається рівнянням:

$$ax_i^{\Pi} + by_i^{\Pi} = \beta_i, \quad i = 1, 2, \dots, p,$$

де A, b — параметри прямої-прототипу;

β_i — політочкові координати, що інтерпретуються як відстані від базисних точок до цієї прямої зі своїм знаком.

Пряма задається через нормалізовані коефіцієнти A, b . Політочковий базис задається набором точок: $(x_1^{\Pi}, y_1^{\Pi}), (x_2^{\Pi}, y_2^{\Pi}), \dots, (x_p^{\Pi}, y_p^{\Pi})$.

Після перетворення базис змінюється на новий — з точками (x_i, y_i) , у якому відповідна пряма-образ визначається рівнянням:

$$\varphi_i = Ax_i + By_i, \quad i = 1, 2, \dots, p,$$

де A, B — параметри нової прямої, які належить знайти.

Якщо перетворення розглядається у евклідовому просторі R^2 , то точки об'єкта-прообразу визначаються як точки перетину прямих-прообразів. Таким чином, у процесі перетворення політочкові координати прямих зазнають змін, що, відповідно, призводить до зміни координат точок їх перетину. Графічна ілюстрація такого політочкового перетворення на площині наведена на рисунку 1.5.

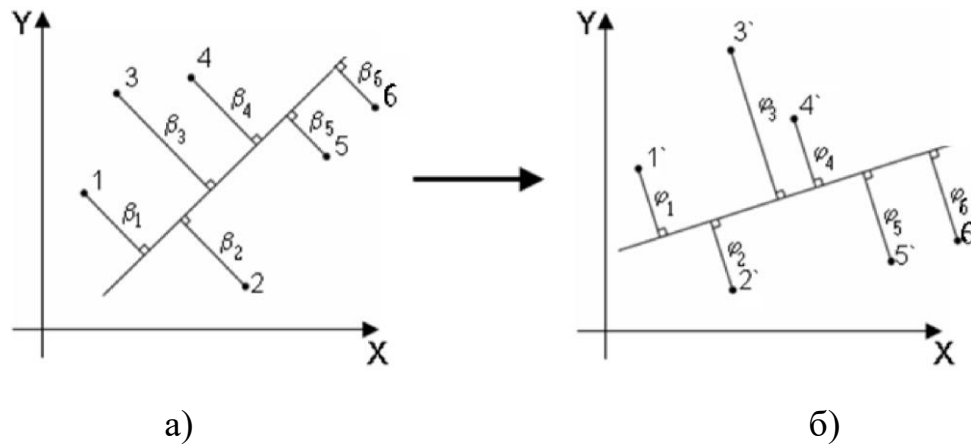


Рис. 1.5. Політочкове перетворення прямої: а) пряма-прообраз і шість політочкових координатів; б) шість політочкових координатів у новому базисі і пряма-образ

Виникає задача отримання однозначного розв'язку у просторі R^2 , що зводиться до встановлення функціональної залежності між політочковими координатами β_i (початковий базис) та φ_i (перетворений базис) для $i=1, 2, \dots, p$. Враховуючи, що політочкова координата прямої інтерпретується як міра її віддаленості від базисної точки, саме перетворення може бути виражене так:

$$\varphi_i = \omega_i \beta_i, \quad i = 1, 2, \dots, p,$$

що, підставляючи у попереднє рівняння, дає:

$$\omega_i \beta_i = Ax_i + By_i, \quad i = 1, 2, \dots, p$$

Таким чином, виникає система з p рівнянь відносно $p + 2$ невідомих (ω_i , A , B). Для її однозначного розв'язання вводять дві додаткові умови, зокрема, умову мінімізації відхилення коефіцієнтів ω_i від одиниці. Це забезпечує наближення нового положення прямої до початкового. Мінімізуючий функціонал можна взяти наступний:

$$J(\omega) = \sum_{i=1}^p (\omega_i - 1)^2 \rightarrow \min$$

Оптимізація цього функціоналу дозволяє знайти такі значення ω_i , які забезпечують найменше викривлення трансформованої прямої відносно її початкового положення.

Система рівнянь, розв'язок якої дозволяє визначити коефіцієнти перетвореної прямої, матиме вигляд:

$$\begin{cases} A \sum_{i=1}^p \frac{X_i^2}{\beta_i^2} + B \sum_{i=1}^p \frac{X_i Y_i}{\beta_i^2} + C \sum_{i=1}^p \frac{X_i}{\beta_i^2} = \sum_{i=1}^p \frac{X_i}{\beta_i} \\ A \sum_{i=1}^p \frac{Y_i X_i}{\beta_i^2} + B \sum_{i=1}^p \frac{Y_i^2}{\beta_i^2} + C \sum_{i=1}^p \frac{Y_i}{\beta_i^2} = \sum_{i=1}^p \frac{Y_i}{\beta_i}, \\ A \sum_{i=1}^p \frac{X_i}{\beta_i^2} + B \sum_{i=1}^p \frac{Y_i}{\beta_i^2} + C \sum_{i=1}^p \frac{1}{\beta_i^2} = \sum_{i=1}^p \frac{1}{\beta_i} \end{cases}$$

де x_i, y_i — декартові координати точок нового базису;

β_i — політочкові координати заданої прямої.

Внаслідок розв'язання згаданої системи отримують параметри A, B, C прямої у новому базисі, яку можна записати у загальній формі:

$$Ax + By + C = 0$$

Цей метод допускає використання різних типів функціоналів для досягнення різноманітних цілей трансформації.

Метод політочкових перетворень забезпечує низку вагомих переваг серед інших підходів: його базисом є скінченна множина контрольних точок, що дозволяє ефективно керувати деформацією цілого об'єкта, а математичний апарат

зводиться до систем лінійних рівнянь, роблячи розрахунки простими й прозорими на відміну від «чорного ящика» нейронних методів.

Цей апарат є природно паралельним і добре підходить для реалізації на багатоядерних та гетерогенних архітектурах, однак, у вигляді послідовної (однопотокової) реалізації на сітках із мільйонами трикутників метод стає надто повільним, що й мотивувало розробку нових способів задання геометрії об'єкта та спеціалізованих схем паралелізації обчислень.

1.4.3 Метод деформаційного моделювання геометричних об'єктів із використанням двовимірного інтерполяційного векторного поля деформації

Під векторним полем деформації розуміється сукупність векторів, які відповідають перенесенню кожної точки геометричного об'єкта у процесі його деформації. Більш формально, якщо кожній точці $M(x_1, x_2, \dots, x_n)$ області Ω n -вимірного простору поставлено у відповідність єдиний вектор $A(M) = A(x_1, x_2, \dots, x_n)$, то кажуть, що в області Ω задане векторне поле A [32, 33].

Для кожного методу моделювання деформаційних змін точкового об'єкта можна побудувати відповідне векторне поле. Наприклад, якщо початкова точка простору x , яка належить множині об'єктів T , після деформації переходить в точку x' , то векторне поле можна визначити формулою:

$$A(x) = x' - x$$

Особливої уваги заслуговують методи, які використовують векторні поля, отримані на основі інтерполяційних процедур із точкових базисів. Такі векторні поля дозволяють створювати гнучкі та ефективні способи моделювання складних деформаційних перетворень.

Застосування векторного поля деформації дозволяє розв'язати низку важливих задач геометричного моделювання. Інтерполяційне векторне поле характеризується неперервністю та гладкістю, що забезпечує візуальну реалістичність і математичну коректність модельованих деформацій. Використання простих методів інтерполяції для побудови такого поля дозволяє суттєво зменшити обчислювальні витрати, що є важливим для задач, які потребують моделювання в реальному часі та інтерактивних системах. Крім того, застосування інтерполяції забезпечує можливість локального керування деформаціями геометричних об'єктів, що дозволяє будь-які зміни базисних точок поширювати плавно та прогнозовано. Векторні поля деформації є універсальними, оскільки можуть бути створені за допомогою різноманітних математичних методів інтерполяції, зокрема поліноміальних, радіальних, барицентричних та інших, що дає змогу ефективно вирішувати специфічні задачі відповідно до особливостей модельованих об'єктів.

Для формалізації задачі деформаційного моделювання розглядається множина базисних точок, які є ключовими для визначення початкового та кінцевого станів об'єкта. Позначимо множину початкових базисних точок як набір координат на площині:

$$\{(x_i^{\Pi}, y_i^{\Pi})\}, \quad i = 1, 2, \dots, p,$$

де p — кількість базисних точок, які задаються початковими декартовими координатами.

Кінцевий базис після деформації задається аналогічно у вигляді множини точок із новими координатами:

$$\{(x_i, y_i)\}, \quad i = 1, 2, \dots, p$$

Таким чином, кожній початковій базисній точці (x_i^{Π}, y_i^{Π}) відповідає деформована базисна точка (x_i, y_i) .

Для опису процесу деформації вводиться поняття вектора переносу, який визначає зміщення кожної базисної точки під час деформації. Вектор переносу для кожної точки визначається за формулою:

$$\vec{d}_i = (x_i - x_i^{\Pi}, y_i - y_i^{\Pi}), \quad i = 1, 2, \dots, p,$$

де \vec{d}_i — вектор переносу i -ї точки базису.

1.4.4 Метод побудови двовимірного інтерполяційного векторного поля

Для побудови двовимірного інтерполяційного векторного поля існують різні математичні методи, що дозволяють за заданими базисними точками отримати неперервне поле деформації у двовимірному просторі. Серед основних методів можна виокремити середньозважену інтерполяцію, метод Шепарда та симплексну інтерполяцію.

Середньозважена інтерполяція належить до класу вагових методів і ґрунтується на обчисленні інтерпольованого значення як зваженої суми значень у базисних точках за неперервними ваговими коефіцієнтами [34, 35]. Вона базується на розрахунку середньозваженого значення, що враховує значення базисних точок з відповідними ваговими коефіцієнтами. Формула середньозваженої інтерполяції має вигляд:

$$F(x) = \frac{\sum_{i=1}^m f(x)_i \cdot k_i(x)}{\sum_{i=1}^m k_i(x)},$$

де $f_i(x)$ — значення опорної функції (або вектора переносу) в базисній точці x_i ;

$k_i(x)$ — неперервні вагові коефіцієнти, такі, що:

$$\lim_{|x-x_i| \rightarrow 0} (k_i(x)) = \infty,$$

де (x_i, y_i) , $i=1...m$ — базис інтерполяції, який заданий через множину із m точок та відповідних значень функції у цих точках.

Метод Шепарда є частковим випадком вагової інтерполяції, де вагові коефіцієнти визначаються як обернена величина відстані до базисних точок. Векторне поле за методом Шепарда задається формулою:

$$F(x) = \frac{\sum_{i=1}^m f(x)_i \cdot \frac{1}{\|x - x_i\|^q}}{\sum_{i=1}^m \frac{1}{\|x - x_i\|^q}},$$

де f_i — значення функції (або вектора переносу) в базисній точці x_i ;

q — додатний степінь, що керує ступенем впливу віддалених точок.

1.4.5 Симплексна інтерполяція функції двох змінних для задач деформаційного моделювання

Симплексна інтерполяція функції двох змінних є ефективним методом, що дозволяє здійснювати побудову неперервного векторного поля деформації на основі довільного набору точок, розташованих у загальному положенні на площині [36]. Це особливо важливо у задачах деформаційного моделювання геометричних об'єктів, де об'єкти зазнають складних локальних змін, що не можуть бути описані регулярними сітками.

Нехай маємо множину точок інтерполяційного каркасу $\{(x_i, y_i)\}_{i=1}^n$ кожній з яких відповідає значення деякої величини z_i . Потрібно побудувати функцію $F(x, y)$, яка б задовольняла умову:

$$F(x_i, y_i) = z_i, \quad i = 1, 2, \dots, n$$

та була неперервною в області, що утворюється опуклою оболонкою цієї множини точок. Для цього простір розбивається на симплекси — трикутники з вершинами у точках (x_i, y_i) [37]. На цьому симпліціальному комплексі будується інтерполяційна функція на R^2 , як показано на рисунку 1.6:

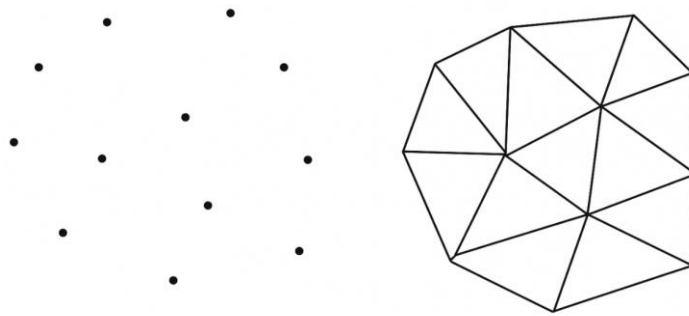


Рис. 1.6. Симпліціальний комплекс, побудований на точковому каркасі

У межах кожного симплексу S_{ijk} побудованого на трьох вершинах, (x_i, y_i) , (x_j, y_j) , (x_k, y_k) , узагальнена вагова інтерполяційна функція визначається формулою:

$$F_{ijk}(x, y) = \frac{f_{ij}(x, y)w(h^{ij}) + f_{jk}(x, y)w(h^{jk}) + f_{ki}(x, y)w(h^{ki})}{w(h^{ij}) + w(h^{jk}) + w(h^{ki})},$$

де $f_{ij}(x, y)$ — вагова функція на ребрі i, j ;

$w(h)$ — ваговий коефіцієнт, що є неперервною спадною функцією відстані h до відповідного ребра симплексу.

Кожна функція $f_{ij}(x, y)$ визначається за допомогою опорних значень у вершинах ребра як:

$$f_{ij}(x, y) = \frac{f_i(x, y)q(h_i^{ij}) + f_j(x, y)q(h_j^{ij})}{q(h_i^{ij}) + q(h_j^{ij})},$$

де $f_i(x, y), f_j(x, y)$ — опорні функції, визначені в вершинах i, j ;

$q(h)$ — ще один ваговий коефіцієнт, що також спадний і неперервний;

h_{ij}, h_{ji} — відстані від проекції точки (x, y) на ребро i, j до відповідних вершин.

Функції $w(h)$ та $q(h)$ мають властивості:

$$\lim_{h \rightarrow 0} w(h) = \infty, \quad \lim_{h \rightarrow 0} q(h) = \infty$$

що гарантує неперервність функції $F(x, y)$ навіть на межах симплексів за умови, що виконуються:

$$\begin{aligned} F_{ijk}(x, y) &\rightarrow f_{ij}(x, y), \quad \text{при } h_{ij} \rightarrow 0 \\ F_{ijk}(x, y) &= f_{ij}(x, y), \quad \text{при } h_{ij} = 0 \end{aligned}$$

У вершинах симплексу значення функцій:

$$\begin{aligned} f_{ij}(x, y) &\rightarrow f_i(x, y), \quad \text{при } h_{ij}^i \rightarrow 0 \\ f_{ij}(x, y) &= f_i(x, y), \quad \text{при } h_{ij}^i = 0 \end{aligned}$$

На рисунку 1.7 показано симплекс S_{ijk} із зазначеною точкою розрахунку, відстанями від цієї точки до ребер симплексу, а також відстанями від проекцій точки на ребра до відповідних вершин.

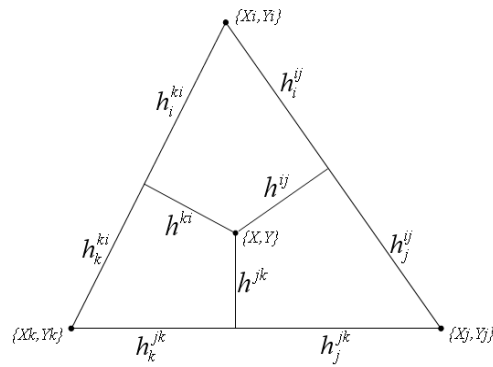


Рис. 1.7. Приклад симплексу S_{ijk}

На рисунку 1.8 проілюстровано результати симплексної інтерполяції для однакового набору точок каркасу за трьох варіантів побудови:

- а) використано константні опорні функції з оберненими ваговими коефіцієнтами;
- б) константні опорні функції з обернено-квадратичними ваговими коефіцієнтами;
- в) білінійні опорні функції з обернено-квадратичними ваговими коефіцієнтами.

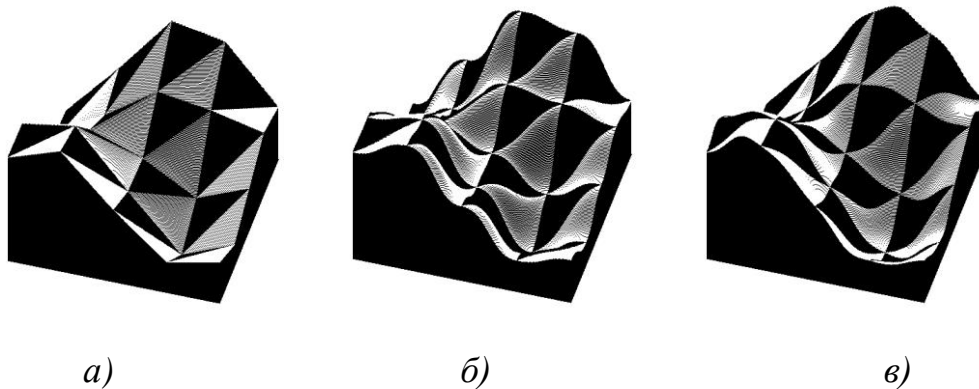


Рис. 1.8. Візуалізація впливу вибору типів опорних функцій і вагових коефіцієнтів на результат симплексної інтерполяції

Таким чином, симплексна інтерполяція дозволяє побудувати глобально неперервну функцію, яка використовується для опису векторного поля деформації. Кожне значення z_i може бути представлено як компонент вектора деформації (наприклад, Δx_i або Δy_i), а сама функція $F(x,y)$ — як згладжене поле деформації, що визначає зміщення точок геометричного об'єкта.

Цей метод є надзвичайно корисним для задач комп'ютерної графіки, моделювання об'єктів із нерегулярною сіткою, локальних деформацій, та інтерполяції результатів сканування об'єктів у деформованому стані.

Висновки до першого розділу

У першому розділі дисертації проаналізовано сучасні методи деформації геометричних об'єктів у комп'ютерній графіці, зокрема нейронні імпліцитні підходи, фізично обґрунтовані моделі, методи геометричних перетворень, а також інтерполяційні підходи з використанням векторних полів деформації. На основі проведеного аналізу сформульовано такі висновки:

Методи моделювання деформаційних змін геометричних об'єктів можна класифікувати за трьома основними напрямками: нейронно-імпліцитні, фізично орієнтовані та геометрично орієнтовані. Фізично обумовлені підходи, як FEM та мас-пружинні моделі, забезпечують високу точність, але мають обмеження щодо інтерактивної продуктивності.

Геометричні методи, зокрема метод довільної деформації форми (FFD), політканинні та політочкові перетворення, дозволяють досягати високої керованості деформаціями та адаптації до складних форм, забезпечуючи інтуїтивне керування процесом трансформації.

Методи на основі нейронних імпліцитних представлень, такі як NeRF та гаусовий сплатінг, відкривають нові можливості для високоточної реконструкції та деформації об'єктів без явної сіткової структури, однак вимагають значних

обчислювальних ресурсів, мають обмеження в режимі реального часу і не здатні зберігати топологію деформованого об'єкта;

Політочкові перетворення становлять перспективний напрямок моделювання геометричних деформацій завдяки можливості гнучкого перетворення простору на основі зміни положення множини контрольних точок (базисів) незалежно від топології сітки. Вони забезпечують достатню локальність та керованість трансформацій. Цей апарат є природно паралельним і добре підходить для реалізації на багатоядерних та гетерогенних архітектурах.

Побудова інтерполяційного векторного поля деформації за заданими точками базису дозволяє реалізувати ефективні моделі деформацій, які сумісні з політочковими перетвореннями, але мають ширший клас застосувань та кращу обчислювальну ефективність при використанні симплексної або вагової інтерполяції.

Разом з тим, практична реалізація політочкових перетворень потребує подальшого розвитку в напрямку формалізованого задання вихідної геометрії об'єкта до моменту застосування перетворення, що є критично важливим для забезпечення стабільності та коректності обчислень. Крім того, досі недостатньо розкритим залишається потенціал ефективної паралелізації політочкових перетворень на багатоядерних процесорах, графічних прискорювачах та обчислювальних кластерах. Ці аспекти становлять предмет подальшого дослідження, результати якого буде розглянуто в наступних розділах дисертації.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ НА ПЛОЩИНІ

У розділі розглядається вдосконалення методу політочкових перетворень для деформації двовимірних об'єктів із застосуванням полігонального задання геометрії, що дозволяє оптимізувати обчислювальні витрати. Для забезпечення гладкості та високої точності отриманих контурів запропоновано інтеграцію перетворень із модифікованим методом параметричної інтерполяції Гауса та розроблено алгоритм оптимізації його варіативного параметра.

2.1 Моделі побудови двовимірних об'єктів політочковими перетвореннями

Політочкові перетворення є одним з різновидів полікоординатних відображень, які використовуються для розв'язання задач відображення деформаційних змін геометричних об'єктів [28].

Об'єктом деформації при політочкових перетвореннях у двовимірному просторі \mathbb{R}^2 є пряма такого вигляду:

$$Ax + Bx + C = 0$$

Нехай задана пряма L_l , з коефіцієнтами A_l , B_l , C_l . Розв'язання задачі політочкових перетворень зводиться до розв'язання системи лінійних алгебричних рівнянь, яку можна записати у матричному вигляді:

$$\begin{bmatrix} A \sum_0^i \frac{X_i^2}{\beta_i^2} & B \sum_0^i \frac{X_i Y_i}{\beta_i^2} & C \sum_0^i \frac{X_i}{\beta_i^2} \\ A \sum_0^i \frac{Y_i X_i}{\beta_i^2} & B \sum_0^i \frac{Y_i^2}{\beta_i^2} & C \sum_0^i \frac{Y_i}{\beta_i^2} \\ A \sum_0^i \frac{X_i}{\beta_i^2} & B \sum_0^i \frac{Y_i}{\beta_i^2} & C \sum_0^i \frac{1}{\beta_i^2} \end{bmatrix} = \begin{bmatrix} \sum_0^i \frac{X_i}{\beta_i} \\ \sum_0^i \frac{Y_i}{\beta_i} \\ \sum_0^i \frac{1}{\beta_i} \end{bmatrix},$$

де (X_i, Y_i) — координата i -того базисного вузла;

β_i — відстань від прямої $L1$ до базиса (X_i, Y_i) .

Розв'язком системи є три коефіцієнти A, B, C , підставивши які у загальне рівняння прямої, отримаємо нову перетворену пряму L . На рисунку 2.1 представлено політочкові перетворення прямої:

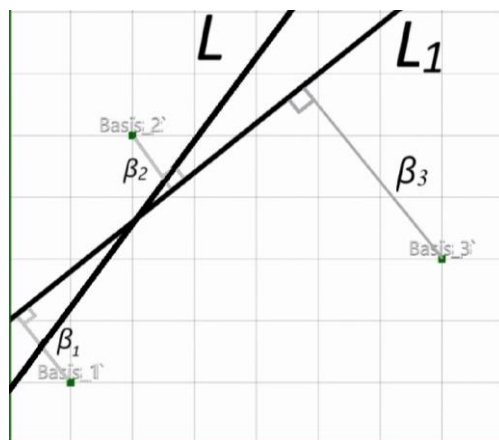


Рис. 2.1. Політочкові перетворення прямої L_1 з використанням трьох базисних точок

Оскільки за допомогою політочкових перетворень можна перенести з одного базису в інший одну пряму, то можна і декілька. Для прикладу побудуємо сітку з прямих, яка являє собою координатну сітку. Це множина ортогональних прямих з

рівним кроком. Керуючи точками базису, отримаємо змінену координатну сітку, як показано на рисунку 2.2-2.3.

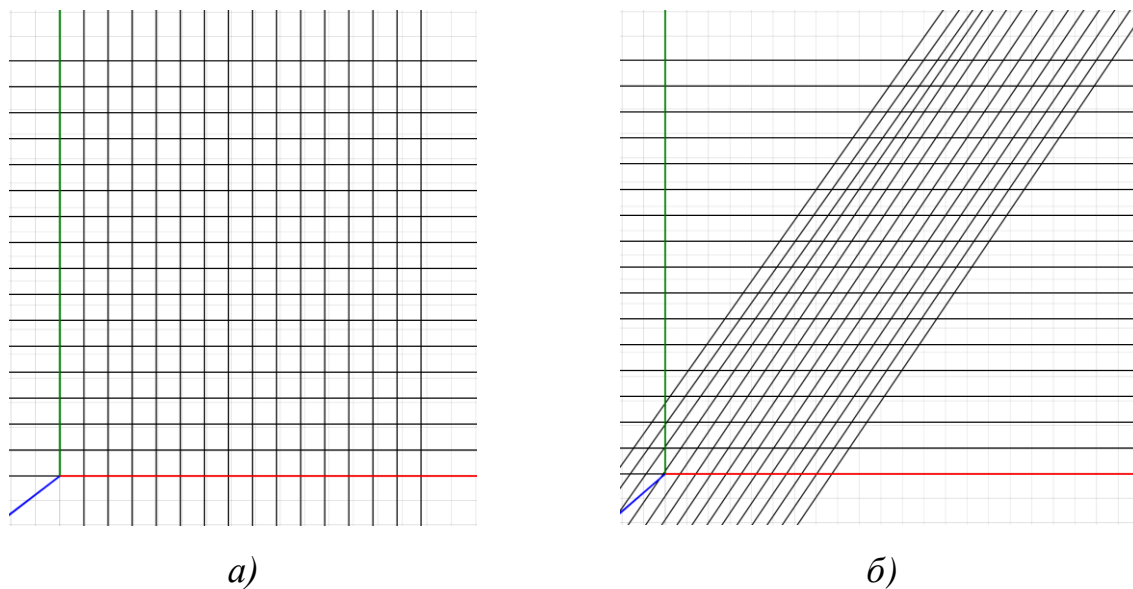


Рис. 2.2. Координатна сітка до деформації (а); після деформації (б)

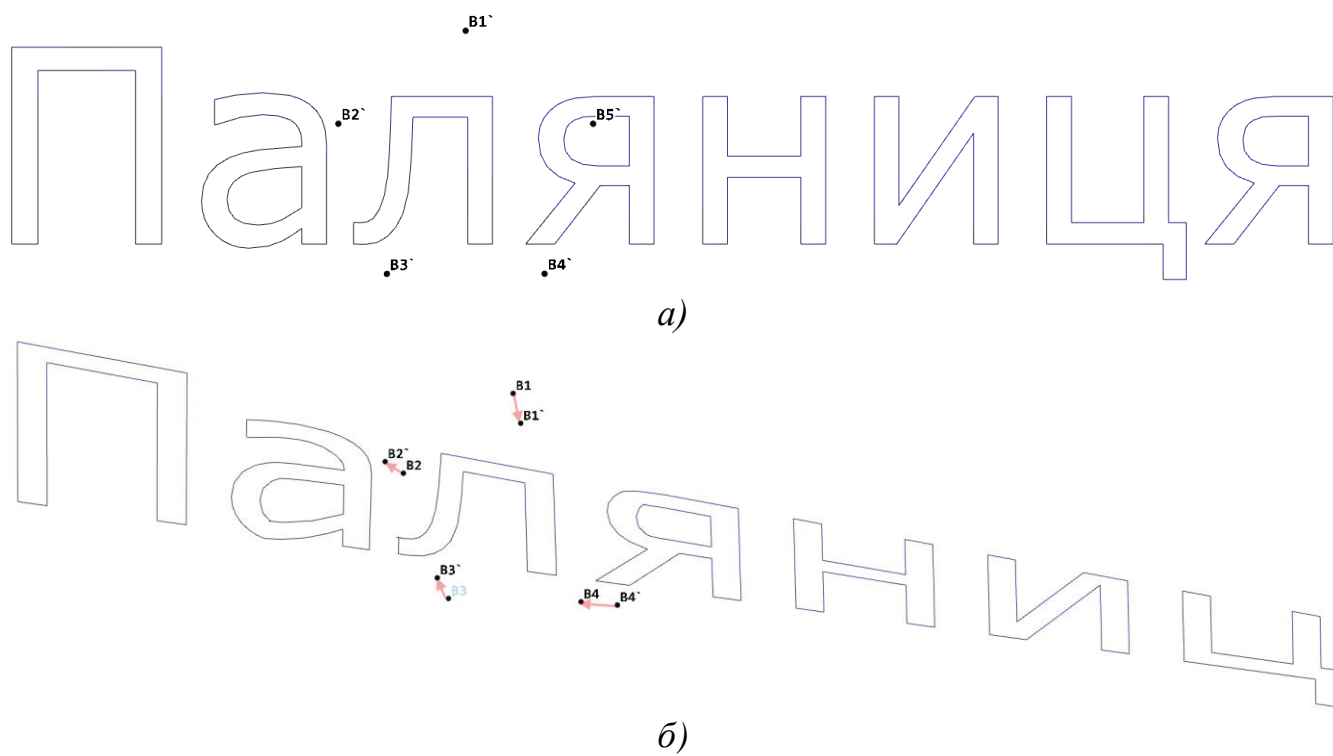


Рис. 2.3. Текст до деформації (а); після деформації (б)

Перетворення довільних двовимірних геометричних об'єктів можна проводити різними способами. Можна використати ту саму координатну сітку. Відобразити на ній об'єкт, розбити його на точки, а потім шукати на зміненій сітці відповідні точки. Цей процес ускладнюється тим, що на виході при роботі апарату політочкових перетворень результатом є коефіцієнти прямих. Тобто, деякі набори векторів з трьох чисел. І зрозуміти, де саме буде точка перетину двох конкретних прямих, ресурсозатратно.

Очевидно, що перед тим, як застосувати математичний апарат політочкових перетворень для деформування об'єкта на площині, необхідно спершу задати цей об'єкт у зручному для обробки вигляді.

2.2 Полігональний спосіб задання геометрії об'єкта

Існує підхід, який застосовується при роботі з політочковими перетвореннями [29, 38]. Суть його полягає в тому, що кожна точка об'єкта подається у вигляді перетину двох конкретних ортогональних прямих, і тоді нема потреби відслідковувати всі перетини всіх заданих прямих. Переносяться дві прямі у новий базис, знаходиться точка перетину. І так з кожною точкою. Але у цьому підході є недолік: оскільки точка представляється перетином двох прямих, то кількість обчислень зростає вдвічі, що в свою чергу призводить до збільшення часу обчислень а також використання зайвої пам'яті комп'ютера.

Розглянемо спосіб задання об'єкта за допомогою відрізків, які з'єднують сусідні точки, утворюючи полігон.

При реалізації такого підходу потрібно дотримуватися певних правил.

По-перше, відрізки повинні бути поєднані, тобто кінець одного відрізка повинен бути початком наступного (рис. 2.5). Це обумовлено тим, що метод політочкових перетворень працює з прямими, заданими у загальному вигляді. Тобто, перетворивши відрізки об'єкта «*KLMNPO*», отримаємо три

прямих k , m , p (рис. 2.4а). Застосувавши метод політочкових перетворень до згаданих прямих та знайшовши точки перетину прямих k , m , p , отримаємо зовсім інший об'єкт (рис. 2.4б). На рисунку 2.4а точки кінців відрізків L і M , N і O , K і P є окремими, на відміну від рисунка 2.4б, де таких точок немає, а замість них утворилися нові.

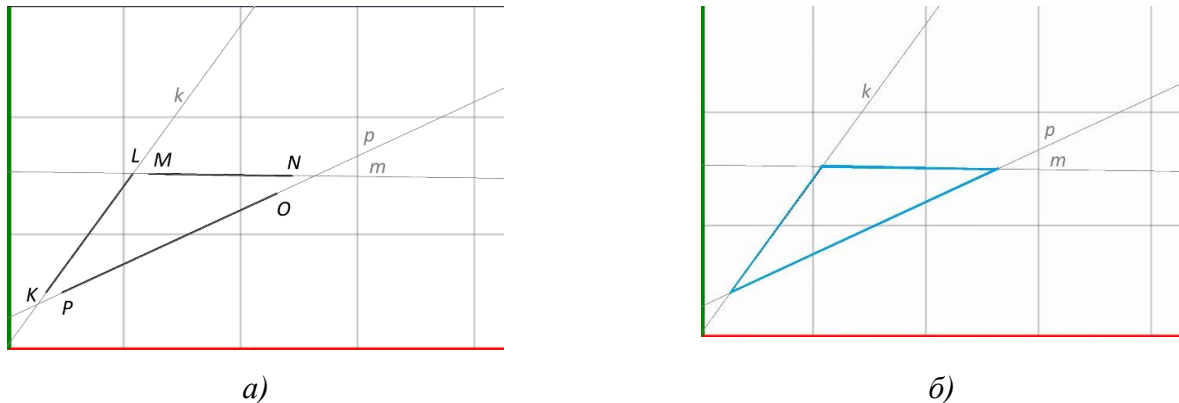


Рис. 2.4. Об'єкт « $KLMNPO$ », заданий прямими k , m , p (а); об'єкт, утворений через точки перетину прямих k , m , p (б)

По-друге, даний підхід вимагає, щоб об'єкт був замкнений. Тобто геометрія об'єкта має задаватись шляхом поєднання відрізків один за одним, утворюючи ланцюжок з відрізків (шлях), кінцем якого завжди повинен бути його початок.

Таким чином, алгоритм перетворення геометричного об'єкта має наступні кроки:

1. Задаємо початковий базис перетворень (певну кількість точок).
2. Задаємо послідовно точки відрізків, на які розбито об'єкт, що перетворюється. Точки кінця попереднього відрізка і початку наступного збігаються, тож їх не потрібно дублювати.
3. Точки потрапляють у стек типу LIFO, властивістю якого є обробка кожного елементу у порядку входження.

4. По кожній з двох послідовно заданих точках будуються прямі, які є об'єктами перетворень.
5. Проводиться зміна початкового базису шляхом переміщення базисних точок.
6. Проводяться розрахунки методом політочкових перетворень нових положень прямих.
7. На виході отримуємо точки перетину цих прямих, що і будуть утворювати форму перетвореного об'єкта.
8. Проводимо візуалізацію знайденого контуру, використавши той самий порядок точок, що і в ПЗ.

Розглянемо це на прикладі. Нехай потрібно задати об'єкт у двовимірному просторі за допомогою відрізків (далі ребер) для подальшого застосування методу політочкових перетворень.

Для цього кожне ребро задамо послідовно у напрямку слідування контуру об'єкта, використовуючи дві точки (x_i, y_i) , (x_j, y_j) для кожного. Пронумерувавши кожен кінець кожного ребра, отримаємо закритий полігон «ABCDEFGHA» (далі АВ-НА) (рис. 2.5). Відмітимо, що останнє ребро НА з'єднується з першим ребром АВ в точці А.

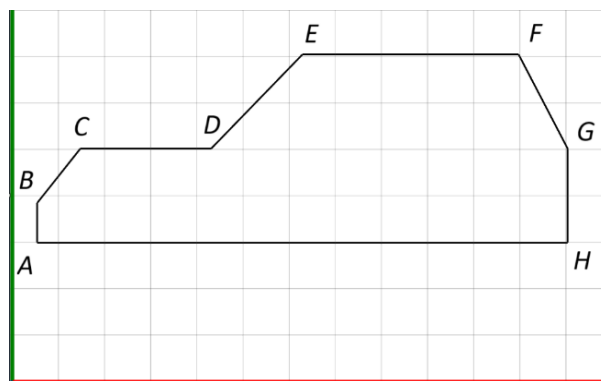


Рис. 2.5. Приклад замкненого полігону

Перетворимо полігон в прямі загального вигляду, застосуємо політочкові перетворення з використанням наступного мінімізуючого функціоналу:

$$S = \sum_{i=1}^p (\omega_i - 1)^2$$

Знайдемо точки перетину сусідніх прямих, розв'язавши систему алгебричних рівнянь:

$$\begin{cases} A_a x + B_a x = -C_a \\ A_b x + B_b x = -C_b \end{cases},$$

де $A_a, B_a, C_a, A_b, B_b, C_b$ — коефіцієнти сусідніх прямих.

Отримаємо нове положення прямих у перетвореному базисі (рис. 2.6).

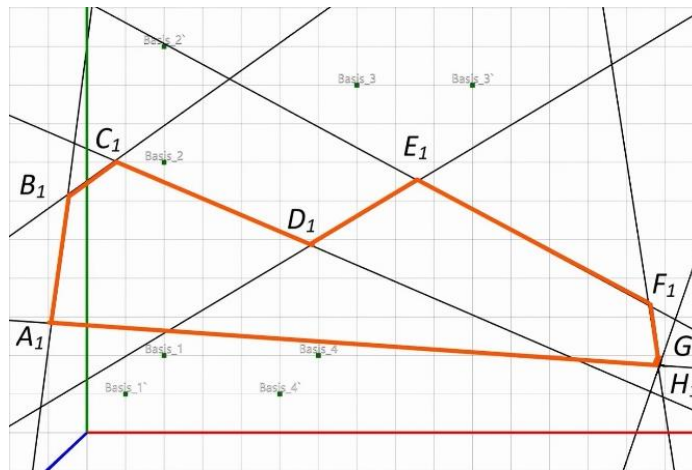


Рис. 2.6. Положення прямих у перетвореному базисі

Використавши точки перетину між сусідніми прямими із списку прямих утворимо новий полігон « $A_1 B_1 C_1 D_1 E_1 F_1 G_1 H_1 A_1$ » (далі $A_1 B_1 - H_1 A_1$), який є прообразом полігону АВ-НА (рис. 2.7).

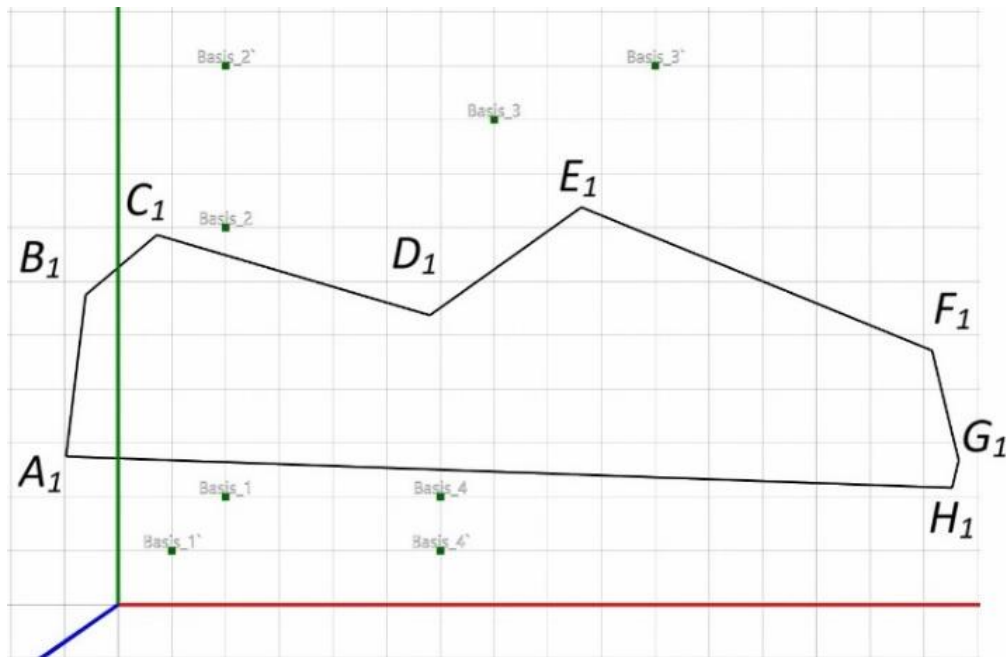


Рис. 2.7. Полігон « $A_1B_1C_1D_1E_1F_1G_1H_1A_1$ » після застосування політочкових перетворень

Таким чином, було отримано перетворений об'єкт після деформації. Перетворений полігон « $A_1B_1C_1D_1E_1F_1G_1H_1A_1$ », складений із тією самою кількістю ребер, що і початковий полігон « $ABCDEFGHNA$ », який побудовано за законом зміни точок базису.

Такий же підхід можна використати і для зважених політочкових перетворень [30]. Для цього потрібно застосувати інший функціонал, а саме функціонал такого вигляду:

$$S = \sum_{i=1}^p m(\beta_i)(\omega_i - 1)^2,$$

де $m(\beta_i)$ — функція від відстаней до прямої образу.

Наприклад, нехай t буде обернено-пропорційно залежати від β у парній степені. У цьому випадку точки об'єкта будуть тим більше притягуватись до точок базису, чим ближче вони до цих точок знаходяться. Причому, чим більший степінь, тим сильніший зв'язок.

Також на екрані можна розмістити декілька об'єктів і застосувати описаний підхід до задання кожного об'єкта. Наприклад, задаємо чотири об'єкта на одній сцені (рис. 2.8).

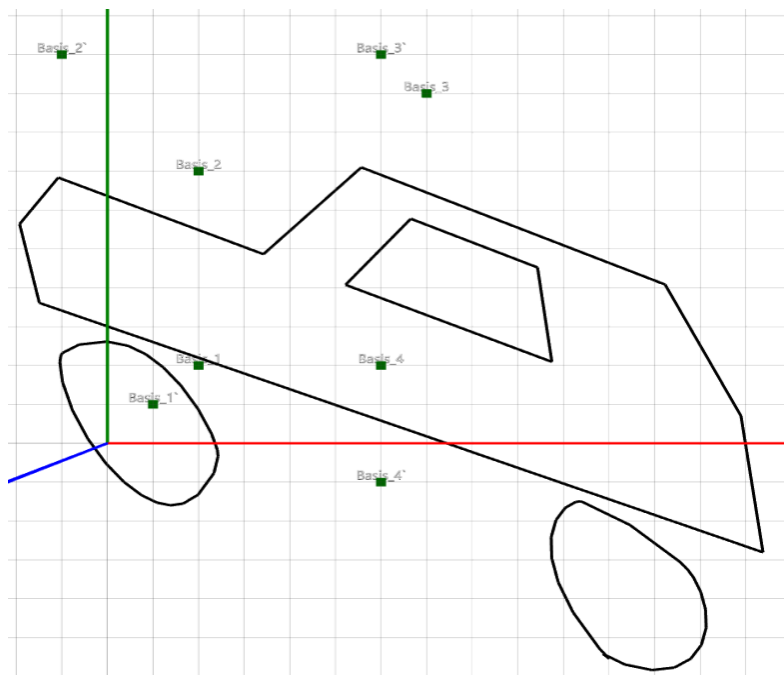


Рис. 2.8. Об'єкт з чотирьох полігонів після застосування політочкових перетворень

На рисунку 2.8 всі чотири об'єкти відобразились коректно. Причому, швидкість відображення об'єктів співрозмірна з їх кількістю і обчислюється долями мікросекунд, на відміну від відображення результатів без застосування полігону.

2.3 Вплив кута між твірними прямими політканини на перетворення точкового об'єкта

Досліджуючи полігональний спосіб задання геометрії для застосування методу політочкових перетворень [4], було виявлено потенційну проблему даного підходу, яка заключається в тому, що при заданні вершини через дві прямі, кут між ними хаотично змінюється і немає досліджень з приводу того, як саме результат деформації залежить від просторової конфігурації об'єкта деформації. В результаті, перетворення точок контуру залежить від відносної орієнтації прямих, що його формують і підлягають політочковому перетворенню. Щоб відповісти на ці запитання, проведено два розрахунки.

У першому розрахунку об'єктом перетворення є контур правильного n -кутника, початковим базисом політочкового перетворення є 5 точок B_j , $j=1..5$, рівномірно розташованих навколо об'єкта, а перетворені базисні точки B_j' представляють перетворення, що поєднує масштабування та обертання.

Точки контуру P_i формуються сусідніми ребрами правильного n -кутника, $i = 1, \dots, N$. Порахуємо, як результат перетворення першої точки P_1 залежить від кількості ребер у багатокутнику та кута, який утворюють два сусідні ребра (рис. 2.9).

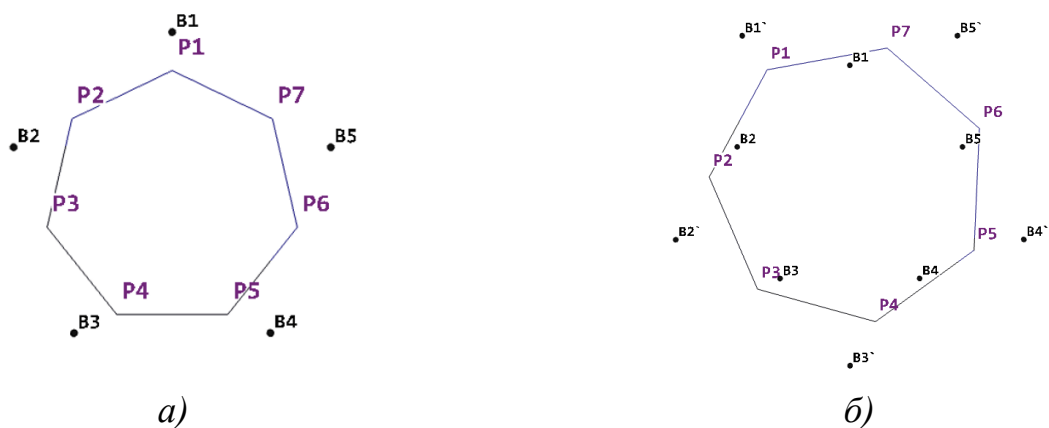


Рис. 2.9. Початковий базис і контур для деформації (а), обидва базиси та деформований контур (б)

Будь-яке практичне застосування політочкових перетворень для деформації трикутної сітки в 3D вимагає, щоб подібні об'єкти з різним рівнем деталізації деформувалися подібним чином. Якщо працювати з декількома рівнями деталізації однієї й тієї ж моделі одночасно, деформація цих рівнів зберігати стійкість, а отже, не повинна суттєво відрізнятись.

У таблиці 2.1 зібрано дані для дев'яти точок, кожна з яких представляє деформацію першої точки контуру правильного багатокутника.

Таблиця 2.1 Параметри точки P_1 для правильного n -кутника

| К-сть кутів в полігоні | Кут між лініями, що формують P_1 , у радіанах | Кут між лініями, що формують P_1 , у градусах | Відстань між початковим і перетвореним положенням P_1 |
|------------------------|---|---|---|
| 3 | 1.047 | 60 | 1.85269 |
| 4 | 1.57 | 90 | 1.85357 |
| 5 | 1.884 | 108 | 1.85399 |
| 6 | 2.094 | 120 | 1.85419 |
| 7 | 2.243 | ~128.57 | 1.8543 |
| 8 | 2.356 | 135 | 1.85437 |
| 9 | 2.443 | 140 | 1.85441 |
| 10 | 2.513 | 144 | 1.85444 |
| 15 | 2.722 | 156 | 1.85451 |

Ті ж дані представлені на графіку на рисунку 2.10:

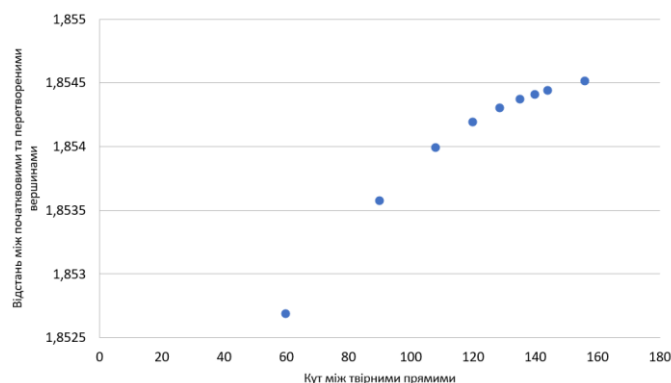


Рис. 2.10. Залежність між кутом твірних прямих, що формують P_1 , і відстанню між початковим положенням точки та її положенням після перетворення

Аналогічним способом проведено розрахунок із гострими кутами, що менші за $\pi/6$. Об'єктом деформації є рівнобедрений трикутник, що складається з вершин P_i , $i=1..3$, з варіативним кутом при вершині. Як і в попередньому розрахунку, початковий базис політочкового перетворення також складається з п'яти точок B_j , $j = 1..5$, рівномірно розташованих навколо об'єкта, а перетворені базисні точки B_j' також представляють перетворення, що поєднує масштабування та обертання.

Приклад такого розрахунку показано на рисунку 2.11.

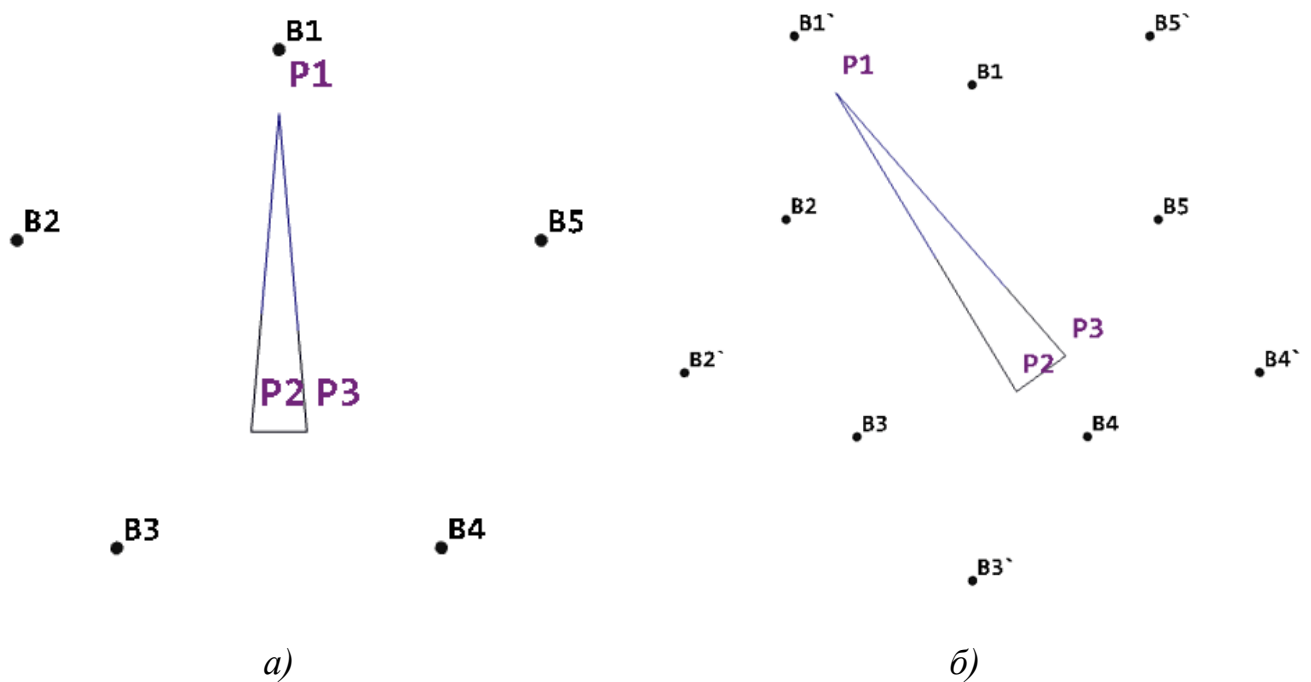


Рис. 2.11. Початковий базис і трикутник до деформації (а), обидва базиси та деформований трикутник (б)

У таблиці 2.2 зібрано дані для восьми точок, кожна з яких представляє деформацію першої точки гострокутного трикутника.

Таблиця 2.2 Залежність зміщення точки P_1 від кута при вершині

| Кут між лініями, що формують P_1 , у градусах | Відстань між початковим і перетвореним положенням P_1 |
|---|---|
| 10 | 1.85182 |
| 20 | 1.85189 |
| 30 | 1.85201 |
| 40 | 1.85219 |
| 50 | 1.85241 |
| 60 | 1.85269 |
| 80 | 1.85329 |
| 100 | 1.85382 |

Ті ж самі дані показані на рисунку 2.12:

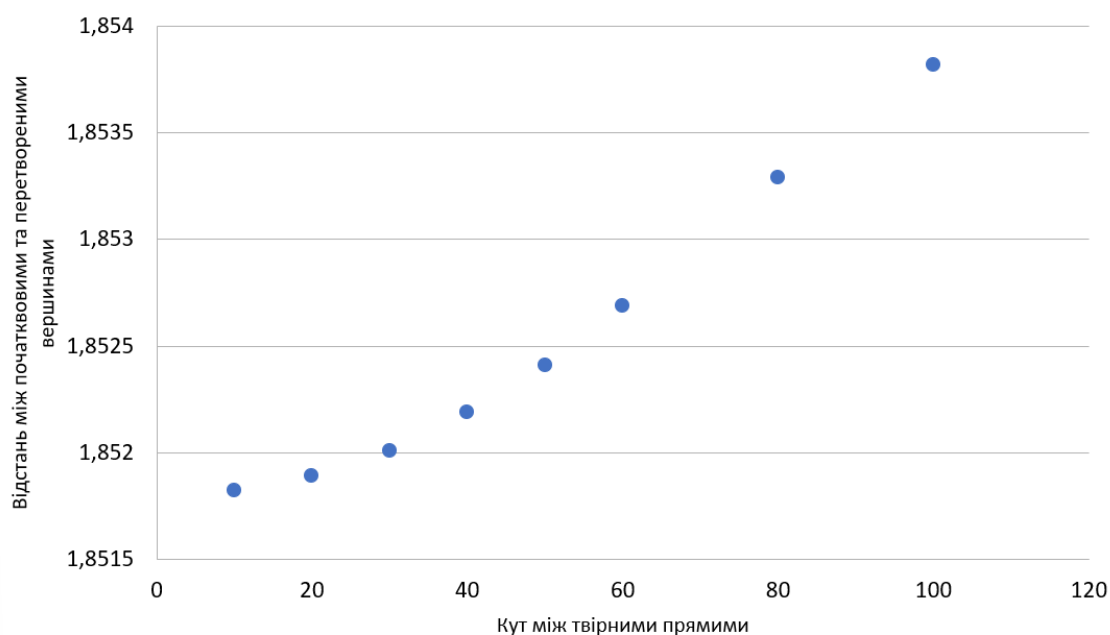


Рис. 2.12. Залежність відстані між початковим положенням точки та її положенням після перетворення від кута при вершині

Зібрані дані також показують, що досліджувана залежність, не має суттєвого впливу на використання алгоритму для деформації сіток у багатовимірному просторі.

Повний масив даних, зібраний як для правильних багатокутників, так і для рівнобедрених трикутників, наведено на рисунку 2.13:

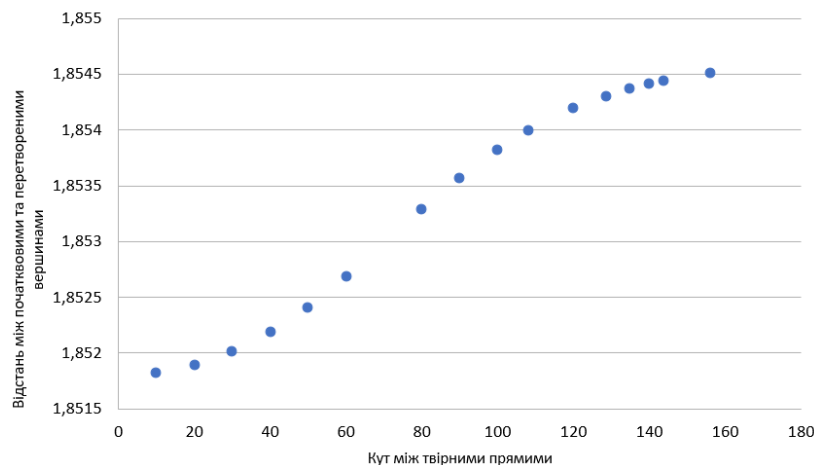


Рис. 2.13. Залежність між кутом твірних прямих у точці P_1 та відстанню між початковою точкою та перетвореною для всіх точок

Безперечно, певна залежність існує, оскільки алгоритм за задумом враховує топологію об'єкта деформації. Однак, як видно з графіка, ця залежність має майже лінійний характер — без осциляцій, вертикальних асимптот чи інших особливих точок. Графік візуально наближається до функції $\sin(A - \pi/2) + b$, де A — кут між утворюючими прямими, а b — значення відстані перетворення при куті $\pi/2$, але з набагато меншою амплітудою.

Отже, хоча залежність між кутом утворюючих прямих у кожній точці та деформацією дійсно є — і це особливість запропонованого методу задання об'єкта перетворення — вона, вочевидь, не перешкоджає застосуванню методу для деформації 2D контурів і 3D-сіток.

Разом з тим, після застосування методу політочкових перетворень виникає потреба у з'єднанні отриманих точок перетину прямих між собою. Попередньо розглянуто з'єднання найпростішими прямими, проте є й інші способи це зробити. Наприклад, Гаус-інтерполяції.

2.4 Застосування інтерполяційних методів для забезпечення гладкості полігональних моделей

Попередньо описаний метод полігонального задання геометрії демонструє певні обмеження у випадках, коли твірні лінії полігона формуються на основі відрізків між вершинами. Зокрема, при перетворенні таких відрізків у твірні прямі перед застосуванням політочкових перетворень, відбувається зміна кутів між суміжними сторонами полігона [4, 5]. Це призводить до порушення кутової узгодженості між сегментами, що в свою чергу негативно впливає на гладкість і безперервність деформованого об'єкта. Як проілюстровано на рисунку 2.14, порівняння вихідної форми (а) з результатом після політочкового перетворення (б) показує помітну зміну конфігурації об'єкта, зокрема — алогічну зміну кута між ребрами.

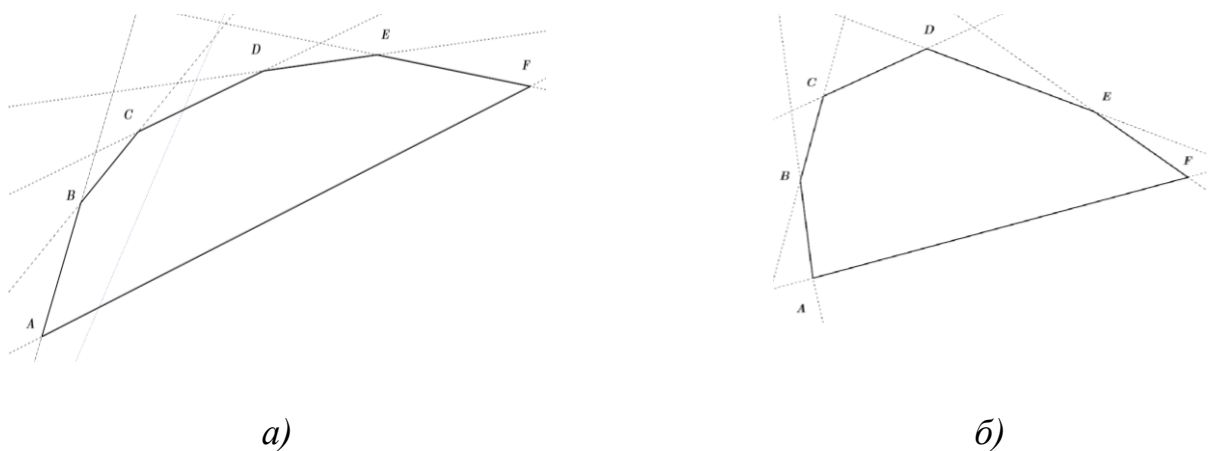


Рис. 2.14. Полігон до застосування політочкових перетворень (а), після застосування політочкових перетворень (б)

Для досягнення монотонного переходу між сусідніми сторонами полігона доцільно застосовувати інтерполяційні підходи, які дозволяють зберігати взаємну орієнтацію твірних ліній, навіть після деформації. Одним із перспективних рішень у цьому напрямку є використання криволінійної інтерполяції [39, 40, 41, 42] або згладжувальних функцій [43], що враховують топологію локального оточення кожної вершини. Такий підхід дозволяє значно знизити недоліки, пов'язані з розривами або зламами в об'єкті після трансформації, що особливо важливо при моделюванні об'єктів з підвищеними вимогами до візуальної якості.

Подальший розвиток цього підходу природним чином веде до застосування методів наближення, які надають формалізований математичний апарат для побудови гладких функцій або кривих за дискретно заданими даними. Зокрема, в задачах геометричного моделювання такі методи дають змогу не лише відтворювати криволінійні контури з високою точністю, а й керувати їх гладкістю та неперервністю, а також контролювати поведінку апроксимації в околі вузлових точок.

Основний зміст методів наближення полягає у заміні заданої функції $f(t)$ функцією $v(t)$, яка наближено відтворює її поведінку відповідно до певного математично визначеного критерію. У контексті геометричного моделювання актуальним є розгляд геометричних аспектів такого наближення, що зумовлює необхідність пошуку ефективних засобів апроксимації криволінійних форм.

Одним із найбільш поширених і ефективних інструментів наближення у цьому контексті є сплайн-функції. Їхня популярність обумовлена, насамперед, зручністю алгоритмічної реалізації та високим ступенем гладкості, який забезпечується при моделюванні криволінійних об'єктів. Завдяки цим властивостям, сплайни набули широкого застосування в задачах апроксимаційного характеру, і на сьогодні складно визначити клас задач у геометричному моделюванні, в яких би сплайнові методи не виявилися придатними.

Задача побудови математичного опису складних, дискретно заданих кривих була вперше розв’язана Ісааком Шенбергом [44]. Подальший розвиток теорії сплайн-функцій відбувався у напрямку підвищення степеня многочленів, які використовуються в окремих сегментах кривих (переважно непарного степеню), а також модифікації крайових умов. Унаслідок цього сформувався сучасний підхід, згідно з яким під сплайнами розуміють функції, що побудовані шляхом з’єднання окремих ділянок поліномів, з дотриманням умов згладженості між ними [45].

Для задання плоских контурів на основі дискретних множин точок широкого поширення набули сплайни першого ступеня, параболічні сплайни з додатковими вузлами, кубічні сплайни дефекту 1, ермітові кубічні сплайни, а також В-сплайни різного порядку [46, 47, 48, 49]. У технічних застосуваннях також використовуються різноманітні нелінійні сплайни, що дозволяють моделювати більш складні залежності та забезпечують розширену гнучкість моделі [50, 51, 52].

Однак при використанні сплайнів для моделювання геометричних обводів криволінійних об’єктів було виявлено суттєвий недолік — схильність до осциляцій, особливо на ділянках з різкою зміною кривини. Це створює додаткові труднощі в технічному моделюванні, оскільки виникає потреба у постобробці кривих: здійснюється аналіз знаку кривини, виключаються небажані точки перегину, а також відокремлюються ділянки опуклості й вгнутості. Для подолання зазначених недоліків були запропоновані альтернативні підходи, зокрема раціональні кубічні сплайни [53], а також так звані «напружені» (tension) сплайни, які забезпечують контроль над жорсткістю кривини [54].

Крім того, у низці практичних задач широко застосовуються кардинальні сплайни [55, 56], які побудовані на основі рівномірно розміщених вузлів, що суттєво спрощує їх реалізацію в чисельних алгоритмах. Водночас усі вищезгадані сплайни орієнтовані переважно на апроксимацію плоских кривих, які можуть бути подані у вигляді функціональної залежності $y=f(x)$, і, таким чином, є непридатними

для моделювання кривих, що містять ділянки з вертикальними дотичними або багатозначними проекціями.

З метою зменшення або повного усунення осциляційних ефектів, характерних для класичних інтерполяційних методів, у попередніх дослідженнях було запропоновано використання ймовірнісних функцій Гауса як альтернативного підходу до побудови згладжених кривих. Вперше метод інтерполяції на основі функцій Гауса був розроблений Сидоренко Ю. В.. У подальших публікаціях було здійснено параметризацію інтерполяційної функції, що базується на гаусовому розподілі, а також наведено практичні алгоритми побудови гладких інтерполяційних ліній із контрольованими характеристиками кривини [57]. Окрему увагу було приділено демонстрації прикладних можливостей цього підходу, зокрема в задачах хімічного моделювання та у фізичних дослідженнях [58, 59]. Проведений аналіз показав, що точність обчислень за допомогою гаусових інтерполяційних функцій суттєво залежить як від обраної математичної форми функції [3], так і від значення варіативного параметра α , який безпосередньо впливає на форму кривої [60] а також від особливості розташування базисних вузлів [61].

В стандартному вигляді інтерполяційний поліном Гауса записується як узагальнений многочлен $\varphi(x)$:

$$\varphi(x) = a_{00}\psi_1(x) + a_{11}\psi_2(x) + \dots + a_{nn}\psi_N(x),$$

де $\psi_i(x)$ — система експоненціальних функцій;

α_{mn} — варіативна змінна, яка була підібрана експериментальним шляхом [57] даючи середню похибку для багатьох видів кривих.

$$\alpha = \frac{\pi(n-1)}{(x_{\max} - x_{\min})^2},$$

де x_{max} , x_{min} — мінімальне і максимальне значення x .

Інтерполяційна функція Гауса визначається як сума з $\psi_i(x)$ (2.1):

$$\varphi(x) = \sum_{i=1}^n \psi_i(x), \quad (2.1)$$

де $\psi_i(x) = \tilde{y}_i e^{-a(x-x_i)^2}$, $i \in [1, N]$.

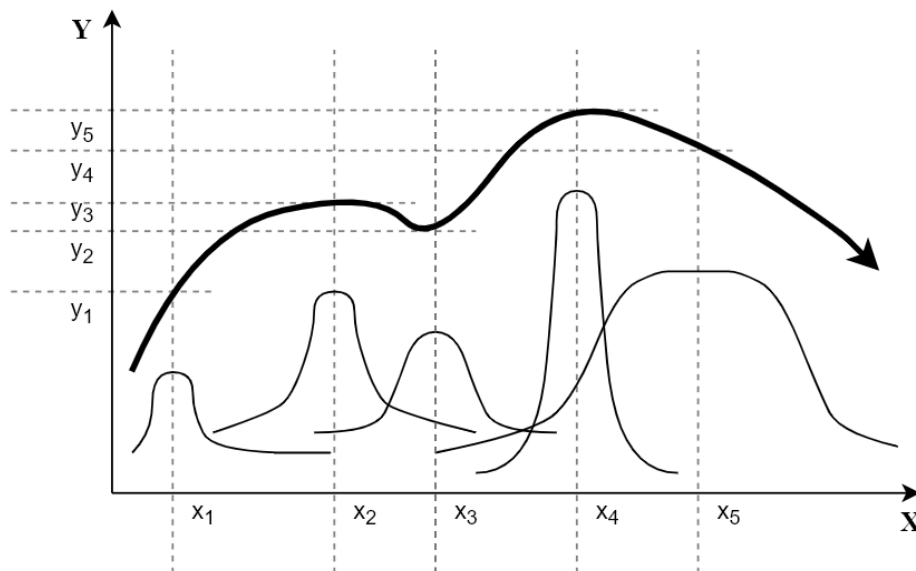


Рис. 2.15. Геометричний зміст Гаус-функції [посилання на бакалавр]

На рисунку 2.15 показаний приклад інтерполяційної кривої, яка складається з п'яти вузлів, в кожному з яких будується гаусіана $\psi_i(x)$.

Для заданої функціональної залежності $y_i = f(x_i)$ необхідно визначити базисні коефіцієнти $\tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_n$, що використовуються при побудові опорних функцій $\varphi_i(x)$. Визначення цих коефіцієнтів зводиться до розв'язання системи лінійних

Застосувавши параметричний метод інтерполяції Гауса до перетвореного набору точок методом політочкових перетворень отримаємо криву (рис. 2.16).

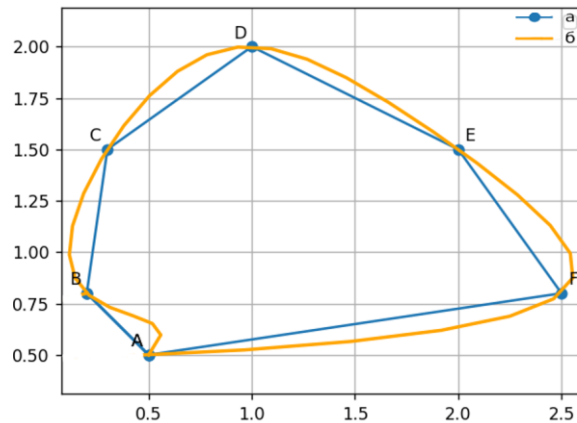


Рис. 2.16. Точковий каркас, з'єднаний прямими (а), інтерполяційна крива (б)

Крива обов'язково проходить через базові точки, за визначенням інтерполяції. Застосування даного методу добре показало себе на різних типах спіралей [3, 65].

2.5 Оптимізований алгоритм вибору параметра α функції гаусової інтерполяції

У аналітичному записі інтерполяційної функції Гауса присутній параметр α . Априорі він приймає значення (2.1), де n — кількість вузлів інтерполяції, X_{\max} , X_{\min} — відповідно максимальне та мінімальне значення аргумента X для формули метода Гауса або аргумента t для формули параметричного метода Гауса.

За допомогою розробленої програмної системи було отримано результати інтерполяції для ряду елементарних математичних функцій [66]. Наприклад, на рисунку 2.17 наведено результати інтерполяції функції \sqrt{x} за умови нерівномірного кроку між базисними вузлами:

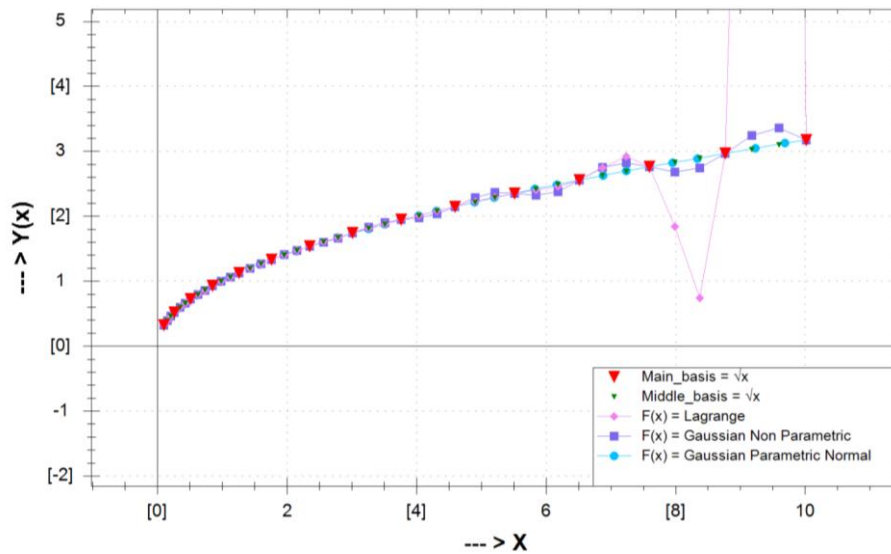


Рис. 2.17. Гаус-інтерполяція функції \sqrt{x}

Похибки інтерполяції в даному випадку склали:

1. поліном Лагранжа: 7,65939887629316;
2. функція Гауса: 0,0902131791036376;
3. параметрична функція Гауса: 0,132234862161009.

З метою зменшення похибки інтерполяції пропонується змінювати значення параметра α шляхом пошуку його оптимального значення. Ефективність алгоритмів різних варіацій методу Гауса може бути підвищена за рахунок довільного налаштування варіативного параметра α . Його значення може задаватися як вручну, так і за допомогою реалізованої у програмній системі функції прокручування. Таким чином, користувач має змогу впливати на точність інтерполяції для кожного з методів. Варто зауважити, що оптимальне значення параметра α , з точки зору мінімізації похибки, є залежним від конкретної функції і змінюється в залежності від її характеру.

У багатьох прикладних задачах отримані значення похибок є прийнятними з практичної точки зору. Проте в окремих випадках виникає потреба у підвищенні точності, коли навіть незначна різниця, наприклад на рівні 0,001, є критичною. У таких ситуаціях доцільно запровадити гранично допустиме значення похибки ϵ та

застосувати модифікований алгоритм визначення оптимального значення варіативного параметра α .

Оскільки результати комп'ютерного експерименту показали, що параметр α зазвичай набуває значень у межах від 0 до 2, було розроблено модифікований алгоритм його оптимізації, що ґрунтується на покроковому звуженні інтервалу пошуку. Алгоритм включає такі етапи:

1. параметр α змінюється в межах $[0; 2]$ з кроком 0,1;
2. на кожному кроці обчислюється максимальне відхилення (сума відхилень) у двох контрольних точках кожного відрізка інтерполяції;
3. визначається підінтервал значень α , на якому сума відхилень є мінімальною;
4. на цьому підінтервалі зменшується крок варіювання α (наприклад, до 0,01) і процедура повторюється;
5. процес ітераційного уточнення триває доти, доки зменшення суми відхилень на поточному кроці не перевищує задану граничну величину ε ;
6. як результат, приймається останнє обчислене значення суми відхилень.

Результатом роботи алгоритму є значення похибки інтерполяції при оптимальному значенні параметра α . Варто зазначити, що як інтервал варіювання α , так і крок його зміни та граничне значення похибки ε можуть змінюватися залежно від конкретних умов поставленої задачі. Застосування даного алгоритму дозволило отримати похибки для всіх трьох варіантів гаусових інтерполяційних функцій. Відповідні результати наведено на рисунку 2.18:

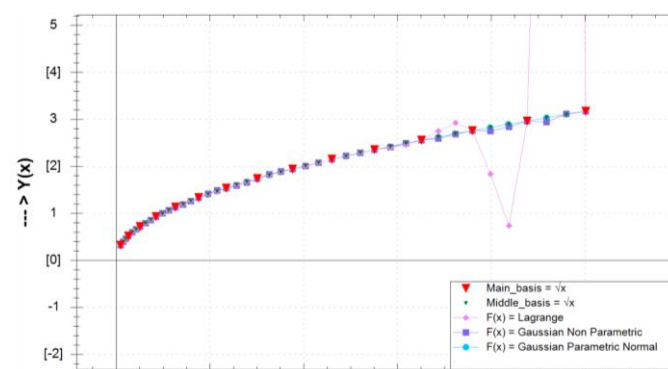


Рис. 2.18. Модифікована Гаус-інтерполяція функції \sqrt{x}

Після модифікації похибки інтерполяції склали:

1. поліном Лагранжа: 7,65939887629316;
2. функція Гауса: 0,0291129289894987;
3. параметрична функція Гауса: 0,0104161689302039.

З рисунку 2.18 видно, що похибка зменшилась для всіх трьох методів. Наприклад, для звичайної функції Гауса похибка зменшилась у 3 рази, для параметричної у 13 разів.

Висновки до другого розділу

У другому розділі удосконалено спосіб задання геометричного об'єкта при двовимірних політочкових перетвореннях за рахунок введення стека для відслідковування входження прямих при відображенні заданої ламаної, що зберігає топологічну цілісність об'єкта. Удосконалено спосіб Гаус-інтерполяції за рахунок адаптації варіативного параметру α до форми кривої на кожному кроці, що дозволило підвищити точність обчислень. Встановлено, що полігональний метод забезпечує ефективне розв'язання проблеми ресурсозатратності обчислень точок перетину прямих, проте має обмеження щодо кутової узгодженості ребер полігона після деформації, що потребує використання додаткових інтерполяційних методів. Показано, що розроблений оптимізаційний алгоритм вибору параметра α гаусової інтерполяції дозволяє зменшити похибки моделювання у 3 рази для звичайного метода Гаус-інтерполяції і у 13 разів для параметричного, що підтверджує перспективність подальших досліджень у напрямку оптимізації політочкових перетворень.

РОЗДІЛ 3. МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ГЕОМЕТРИЧНИХ ОБ'ЄКТІВ У БАГАТОВИМІРНОМУ ПРОСТОРИ

У цьому розділі розглядається розвиток та адаптація методу політочкових перетворень для моделювання нелінійних деформацій геометричних об'єктів у тривимірному просторі. Оскільки базовий математичний апарат методу оперує перетвореннями площин, ключовим завданням є розробка ефективних способів представлення багатополігональних 3D-моделей через сукупність площин.

3.1 Побудова тривимірних об'єктів політочковими перетвореннями

Під час моделювання динамічних процесів часто виникає завдання побудови геометричних об'єктів — як плоских, так і об'ємних — що змінюють свою форму внаслідок деформацій. Метод політочкових перетворень для нелінійної деформації площини у тривимірному просторі задається системою рівнянь виду:

$$\begin{bmatrix} A \sum_{i=1}^p \frac{X_i^2}{\gamma_i^2} & B \sum_{i=1}^p \frac{X_i Y_i}{\gamma_i^2} & C \sum_{i=1}^p \frac{X_i Z_i}{\gamma_i^2} & D \sum_{i=1}^p \frac{X_i}{\gamma_i^2} \\ A \sum_{i=1}^p \frac{Y_i X_i}{\gamma_i^2} & B \sum_{i=1}^p \frac{Y_i^2}{\gamma_i^2} & C \sum_{i=1}^p \frac{Y_i Z_i}{\gamma_i^2} & D \sum_{i=1}^p \frac{Y_i}{\gamma_i^2} \\ A \sum_{i=1}^p \frac{Z_i X_i}{\gamma_i^2} & B \sum_{i=1}^p \frac{Z_i Y_i}{\gamma_i^2} & C \sum_{i=1}^p \frac{Z_i^2}{\gamma_i^2} & D \sum_{i=1}^p \frac{Z_i}{\gamma_i^2} \\ \sum_{i=1}^p \frac{X_i}{\gamma_i^2} & B \sum_{i=1}^p \frac{Y_i}{\gamma_i^2} & C \sum_{i=1}^p \frac{Z_i}{\gamma_i^2} & D \sum_{i=1}^p \frac{1}{\gamma_i^2} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^p \frac{X_i}{\gamma_i} \\ \sum_{i=1}^p \frac{Y_i}{\gamma_i} \\ \sum_{i=1}^p \frac{Z_i}{\gamma_i} \\ \sum_{i=1}^p \frac{1}{\gamma_i} \end{bmatrix}, \quad (3.1)$$

де A, B, C, D — шукані коефіцієнти площини у гомогенному просторі;

X_i, Y_i, Z_i — координати кінцевого i -го базису;

γ_i — відстань від початкової площини до початкового i -го базису, або політочкова координата заданої площини;

p — кількість початкових та кінцевих базисів.

На рисунку 3.1 проілюстровано перетворення площини α в α' , де кількість початкових та кінцевих базисів $p = 3$.

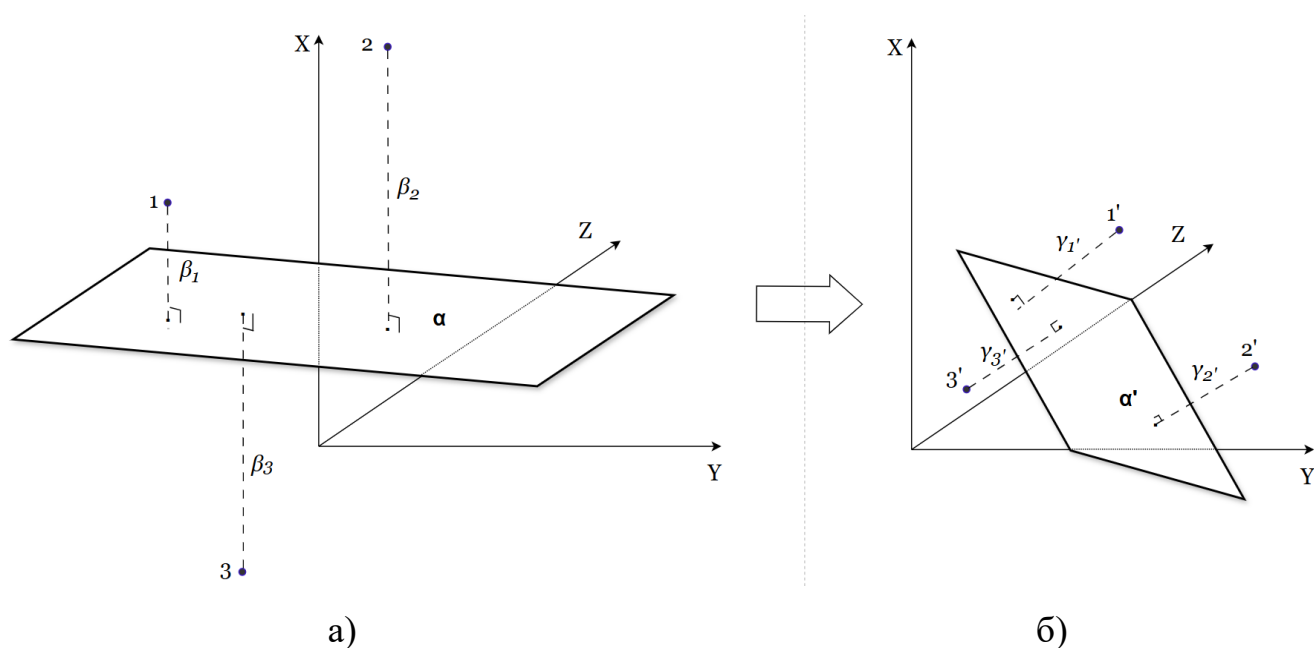


Рис. 3.1. Політочкове перетворення площини: а) площина-прообраз і три політочкових координати; б) три політочкові координати у новому базисі і площина-образ

У результаті розв'язання системи 3.1 визначаються параметри A, B, C, D площини в новому базисі, які можна подати у стандартній формі:

$$Ax + By + Cz + D = 0$$

Зазначений метод дозволяє застосовувати різні типи функціоналів залежно від поставлених цілей трансформації. В даному дослідженні використовується

мінімізуючий функціонал виду $J(\omega) = \sum_{i=1}^p (\omega_i - 1)^2 \rightarrow \min$ через його простоту та передбачуваність [67].

Запропонований підхід до трансформації площин передбачає його застосування до кожної окремої площини, з яких формується об'єкт. Однак у комп'ютерній графіці геометрія об'єктів, як правило, задається не площинами, а полігонами — наприклад, вексями або трикутниками. Це зумовлює необхідність вирішення двох важливих завдань: по-перше, визначення способу подання геометрії об'єкта у вигляді списку площин, придатного для використання методу політочкових перетворень; по-друге, розробки ефективної процедури відновлення деформованого об'єкта зі списку перетворених площин у форму триангульованої моделі. Подібний принцип дискретного подання поверхні через сітку та апроксимацію локальних елементів площинами використовується і в статико-геометричному методі, де це застосовується для формоутворення параметрів оболонки заданої форми [68].

Таким чином, актуальними є питання пошуку оптимального представлення об'єкта у вигляді набору площин, що відображають його структуру з достатньою точністю для деформаційних обчислень, та побудови алгоритмів, які дозволяють швидко та коректно реконструювати трикутну або полігональну сітку з деформованих площин після застосування політочкових перетворень.

3.2 Способи задання геометрії об'єкта для політочкових перетворень

Об'єктами політочкових перетворень виступають пряма на площині та площа у тривимірному просторі [26]. Відповідно, для застосування політочкових перетворень з метою моделювання деформації тривимірної сітки трикутників необхідно подати цю сітку у вигляді множини площин. Існує кілька підходів до такого подання, кожен із яких має власні переваги та недоліки. На рисунку 3.2

показаний прообраз моделі стендфордського кролика зануреного у модель сфери зліва та образ моделі кролика, отриманий через деформацію сфери справа.

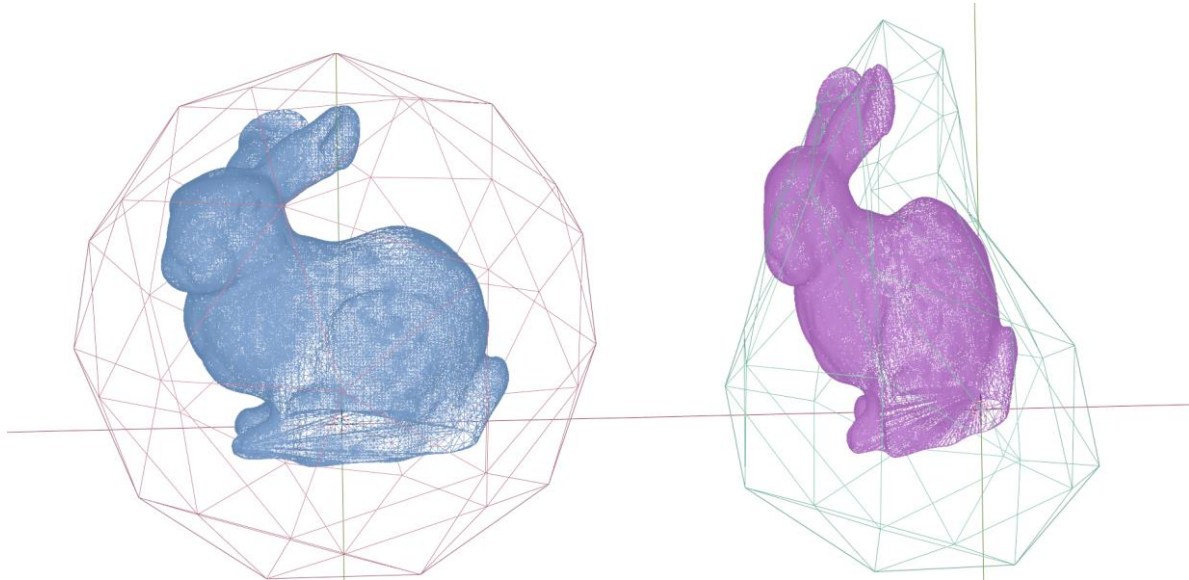


Рис. 3.2. Політочкове перетворення моделі кролика, зануреного у модель сфери

3.2.1 Спосіб задання геометрії об'єкта через перетин площин трикутника та його нормалей

Для усунення основного недоліку способу моделювання деформації трикуткової сітки, запропонованого раніше [30], що полягає у втраті однозначності трансформації вершин трикутників, а відтак і топології самої сітки, пропонується наступна модифікація. Суть модифікації полягає в тому, що перетворення вершини, спільного для кількох суміжних трикутників, визначається як середнє арифметичне координат відповідних вершин, які утворюються при незалежній деформації кожного трикутника.

Позначимо через $\alpha_1, \beta_1, \gamma_1$ — площини, що задають вершину A у трикутнику ABC, та $\alpha_2, \beta_2, \gamma_2$ — площини, що задають вершину A у трикутнику ABD. Після застосування політочкових перетворень ці площини переходять у площини $\alpha'_1, \beta'_1,$

γ_1' та α_2' , β_2' , γ_2' відповідно, визначаючи нові вершини ABC і ABD' . Координати кінцевого деформованої вершини A' визначаються як середнє арифметичне координат точок ABC' і ABD' . Вказаний процес схематично проілюстровано на рисунку 3.3.

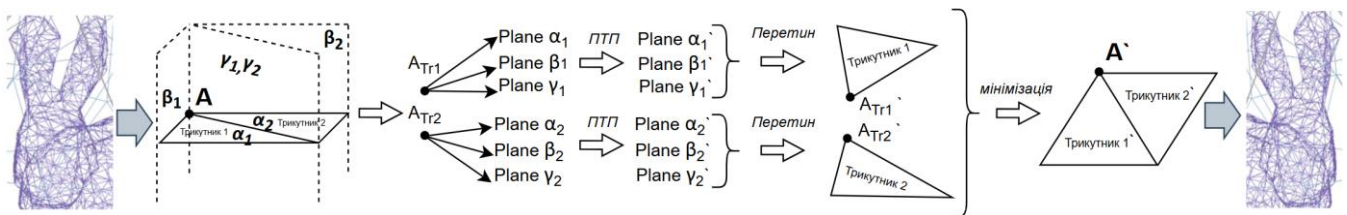


Рис. 3.3. Схема перетворення вершин A в A' за методом усереднення координат вершин суміжних трикутників

Запропонований модифікований спосіб забезпечує однозначність перетворення вершин, тим самим зберігаючи топологію трикуткової сітки. Крім того, цей підхід є ізотропним у тому сенсі, що результат деформації не залежить від вибору системи координат, в якій задані базисні точки та вершини трикуткової сітки.

Втім, питання про оптимальність методу усереднення координат для подолання неоднозначності перетворень залишається відкритим і потребує додаткових досліджень.

3.2.2 Спосіб задання геометрії об'єкта через перетин ортогональних площин трикутника

Наступний спосіб моделювання деформації трикуткової сітки не потребує додаткових модифікацій для збереження її топології, проте характеризується анізотропністю [69]. Процес моделювання деформації трикуткової сітки по вершинно, де кожна вершина задається ортогональними площинами, схематично

зображений на рисунку 3.4. У даному випадку результат деформації залежить від вибору ортогональних площин, які задають положення кожної вершини, а отже, залежить і від вибору системи координат. Питання оптимального вибору цих ортогональних площин наразі залишається відкритим і вимагає подальших досліджень.

У межах цієї роботи вершину з координатами (x, y, z) визначаємо як точку перетину ортогональних площин, заданих рівняннями: $(1, 0, 0, -x)$, $(0, 1, 0, -y)$ і $(0, 0, 1, -z)$. Такий підхід спрощує обчислювальні процедури, при цьому немає вагомих підстав вважати, що вибір інших ортогональних площин суттєво вплине на точність отриманої моделі деформації.

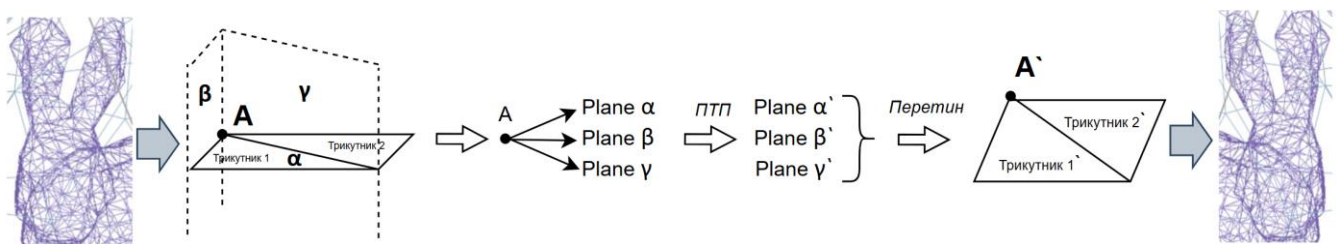


Рис. 3.4. Схема перетворення вершин A в A' заданої ортогональними площинами

Позначимо через α, β, γ площини, що визначають положення вершини A у трикутнику ABC , а через α', β', γ' — відповідні площини вершини A' після застосування політочкових перетворень. Вершина A' є результатом деформації вершини.

3.2.3 Спосіб задання геометрії об'єкта через перетин площин дотичних трикутників

Альтернативний спосіб моделювання деформації трикутничкової сітки полягає в тому, що кожна її вершина визначається як точка перетину площин

трикутників, які до неї дотичні [5]. Завдяки такому підходу, окрім однозначного збереження топології сітки, забезпечується і збереження її локальної геометрії. Процес моделювання деформації трикутничкової сітки зображений на рисунку 3.5.

При деформації площини дотичних трикутників трансформуються, і нове положення вершини визначається як перетин перетворених площин. У випадку, коли кількість дотичних трикутників дорівнює трьом, нове положення вершини визначається однозначно точкою перетину відповідних перетворених площин. Однак якщо до вершини дотичні чотири або більше трикутників, то, як правило, єдина точка перетину перетворених площин не існує. У такому випадку вершина визначається як точка, яка мінімізує суму квадратів відстаней до всіх перетворених площин, тобто знаходиться якнайближче до всіх перетворених площин одночасно.

Для визначення такого положення вершини доцільно застосовувати метод псевдоінверсної матриці Мура-Пенроуза (від англ. Moore–Penrose pseudoinverse [70, 71]), який дозволяє розв'язати поставлену задачу математичної оптимізації ефективно.

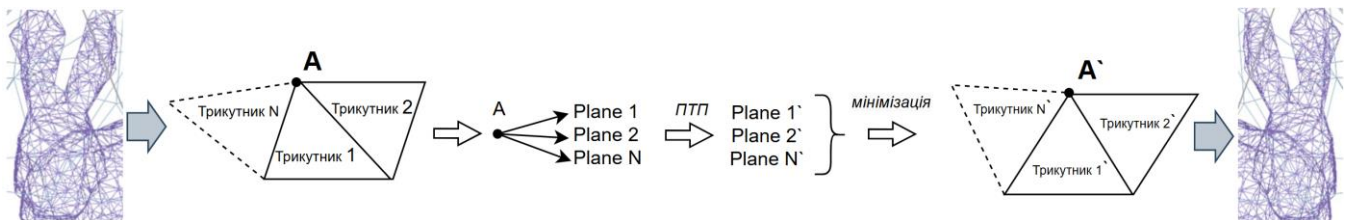


Рис. 3.5. Схема перетворення вершин A в A' методом перетину площин дотичних трикутників.

Позначимо через $Plane_1, Plane_2, \dots, Plane_N$ площини трикутників $1, 2, \dots, N$, які є дотичними до початкової вершини A . Через $Plane_1', Plane_2', \dots, Plane_N'$ позначено відповідні площини після політочкових перетворень. Тоді вершина A' визначає положення вершини A після деформації об'єкта.

3.3 Порівняльний аналіз трьох способів задання геометрії об'єкта

Після детального опису трьох способів задання трикутничкової сітки за допомогою множин площин, доцільно провести порівняльний аналіз точності моделювання заданих деформацій за допомогою політочкових перетворень цих площин і відповідної подальшої реконструкції вершин сітки. Для цього виконується серія чисельних експериментів, у яких досліджуваний об'єкт спочатку занурюється у початковий політочковий базис, а потім цей базис перетворюється наперед визначеним аналітичним перетворенням. Отриманий у такий спосіб результат політочкового перетворення порівнюється з еталонним об'єктом, координати вершин якого обчислюються безпосередньо за тією ж самою аналітичною формулою перетворення.

Наприклад, для перевірки точності моделювання деформації скручування при використанні третього способу задання трикутничкової сітки об'єкт занурюється у початковий точковий базис. Після цього кінцеве положення базису перетворення обчислюється відповідно до аналітично заданої формули деформації скручування. Паралельно з цим обчислюються координати вершин еталонного об'єкта, який безпосередньо трансформується тією ж формулою деформації. Далі виконується політочкове перетворення досліджуваного об'єкта, і отримані результати порівнюються з відповідними вершинами еталонного перетвореного об'єкта. Оцінка точності моделювання здійснюється шляхом розрахунку середньоквадратичної похибки (RMSE):

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (P_i - \hat{P}_i)^2} \quad .$$

де n — загальна кількість відповідних вершин об'єкта;

P_i — координати вершини еталонного перетвореного об'єкта;

\hat{P}_i — координати відповідної вершини об'єкта, отриманого за допомогою політочкового перетворення;

$(P_i - \hat{P}_i)^2$ — квадрат відстані (евклідової метрики) між відповідними точками.

Таким чином, метою наведених вище чисельних експериментів є встановлення того, який із запропонованих способів задання трикуткової сітки множинами площин забезпечує найменше відхилення результату політочкового перетворення від еталонної моделі деформації.

У якості тестового об'єкта для проведення експериментів використовується геометрична модель тора (рис 3.6-3.7), що складається з 400 трикутників і 200 вершин. Початковим політочковим базисом слугує модель одиничного куба, яка складається з 8 вершин. Модель тора вписується в одиничний куб таким чином, щоб кожна сторона куба доторкалася поверхні тора. Такий підхід обумовлений необхідністю забезпечити можливість передбачення результату деформації та візуальної оцінки точності проведених перетворень.

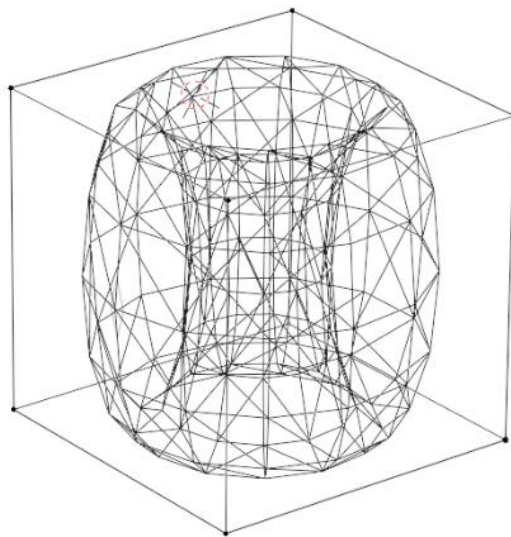


Рис. 3.6. Об'єкт перетворення (модель тора) та початковий базис (модель куба)

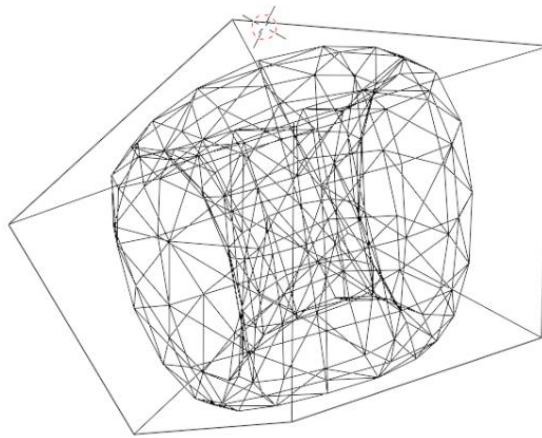


Рис. 3.7. Об'єкт перетворення (модель тора) та кінцевий базис (скручена модель куба, $d=2$)

Було проведено чисельний експеримент, в якому моделювалася деформація, задана такими аналітичними співвідношеннями:

$$\begin{cases} X_s(x, y, z) = \sin(\arctg(y/x) + \text{angle}) \sqrt{x^2 + y^2} \\ Y_s(x, y, z) = \cos(\arctg(y/x) + \text{angle}) \sqrt{x^2 + y^2}, \\ Z_s(x, y, z) = z \end{cases}$$

де x, y, z — початкові координати точки;

X_s, Y_s, Z_s — координати перетвореної точки;

$\text{Angle} = z/d$.

Наведене перетворення відповідає деформації скручування навколо осі Z [72], де параметр d визначає інтенсивність скручування: що меншим є значення d , то інтенсивнішим є скручування об'єкта. В експерименті було використано значення параметра $d \in [15; 2]$.

У рамках цього експерименту вимірювалось середньоквадратичне відхилення (RMSE) між вершинами об'єкта, отриманими за допомогою

політочкових перетворень на основі чотирьох різних способів задання трикуткової сітки, та відповідними вершинами еталонного об'єкта, отриманого прямим застосуванням наведених аналітичних формул. Результати цього експерименту наведені на рисунку 3.8, де *Method 1*, *2*, *3* — спосіб задання геометрії окремими трикутниками, ортогональними площинами та площинами, що перетинаються. RBF Метод — метод радіальної базисної функції, обґрунтування якого доведене в роботах [73, 74, 75].

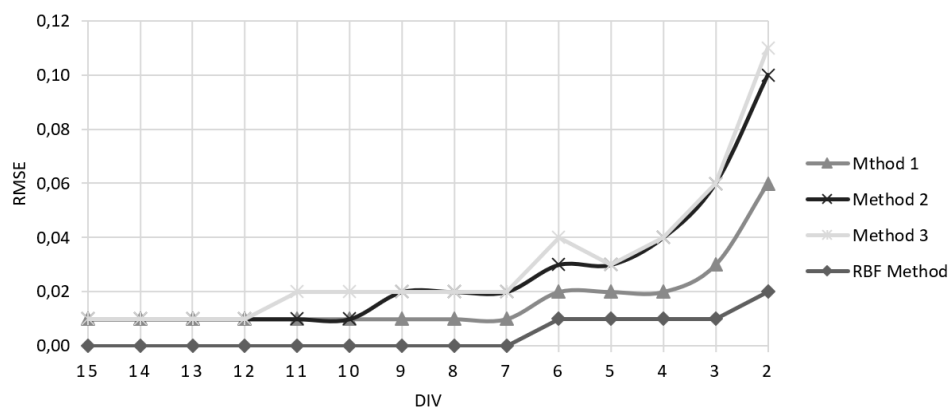


Рис. 3.8. Графік залежності *RMSE* від *division* при скручуванні об'єкта

На рисунку 3.9-3.11 показано моделі тора після застосування політочкових перетворень при дільнику $d = \{12, 6, 2\}$.

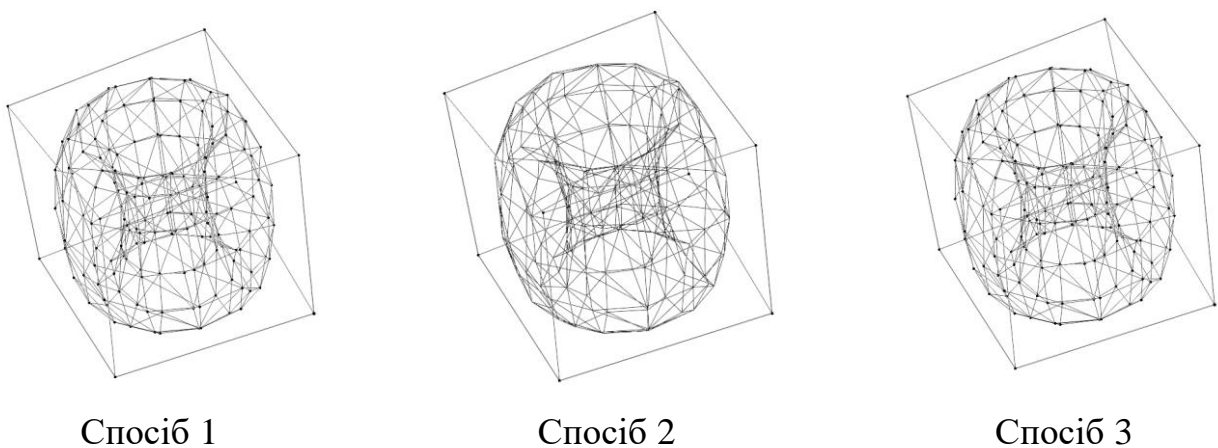
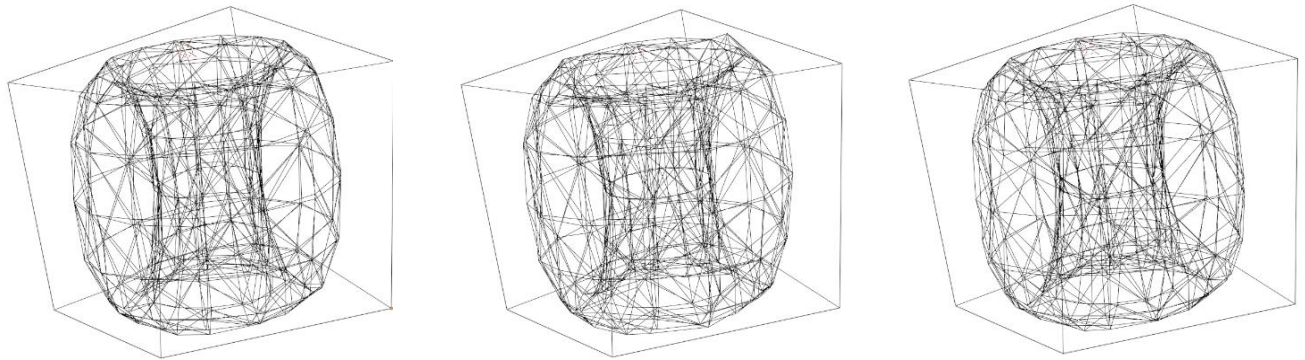


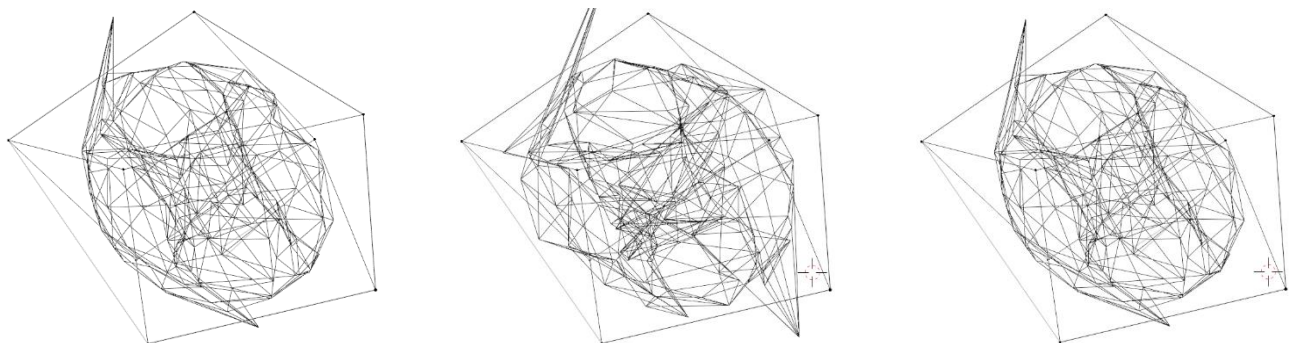
Рис. 3.9. Модель тора після моделювання деформації скрутки з $d = 12$



Спосіб 1

Спосіб 2

Спосіб 3

Рис. 3.10. Модель тора після моделювання деформації скрутки з $d = 6$ 

Спосіб 1

Спосіб 2

Спосіб 3

Рис. 3.11. Модель тора після моделювання деформації скрутки з $d = 2$

Як видно з рисунків, зі зменшенням параметра d (а відтак із зростанням кута повороту верхньої чотирикутної грані базисного куба) модель деформації втрачає внутрішню узгодженість: спостерігається непропорційне зміщення окремих вершин порівняно з іншими. Зазначений ефект притаманний усім трьом способам представлення об'єкта й зумовлений посиленням локальної анізотропії та погіршенням обумовленості задачі реконструкції за великих кутових деформацій, що, своєю чергою, обмежує діапазон їх практичного застосування.

У наступному експерименті було змодельовано нелінійну деформацію (неоднорідне зростання за об'ємом [76]), яка описується такими співвідношеннями:

$$\begin{cases} X_s(x, y, z) = x + \delta \\ Y_s(x, y, z) = y + \delta, \\ Z_s(x, y, z) = z + \delta \end{cases}$$

де x, y, z — координати старої точки;

X_s, Y_s, Z_s — координати перетвореної точки;

$\delta = f \cdot x \cdot y \cdot z$, де f — масштабний коефіцієнт.

Дана модель описує зростання об'єкта, інтенсивність якого залежить від об'єму, визначеного добутком координат точки.

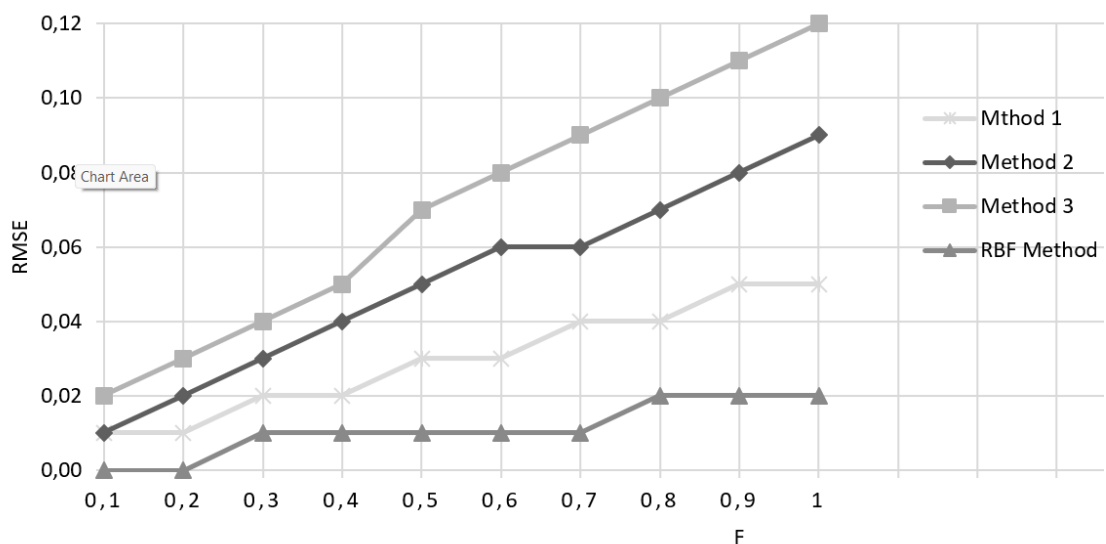
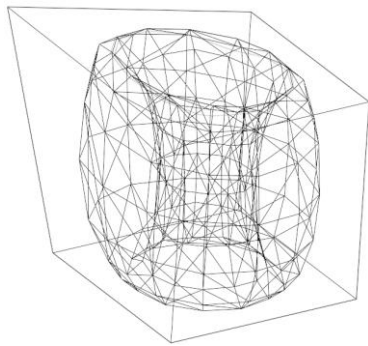


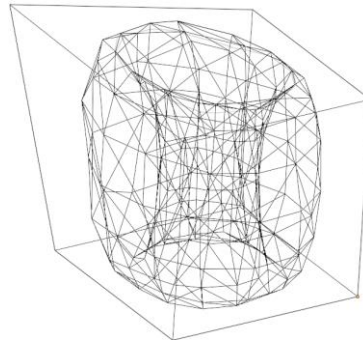
Рис. 3.12. Залежність RMSE моделі деформації від параметра f для випадку неоднорідного зростання за об'ємом

Залежність середньоквадратичної похибки моделювання для кожного з методів представлення вершин від значення параметра f наведено на рисунку 3.12.

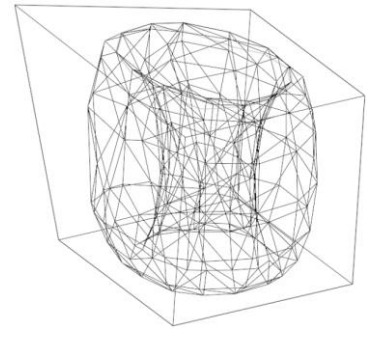
На рисунках 3.13–3.15 наведено приклади деформації моделі тора після застосування політочкових перетворень для значень параметра $f = \{0.1, 0.5, 1.0\}$.



Спосіб 1

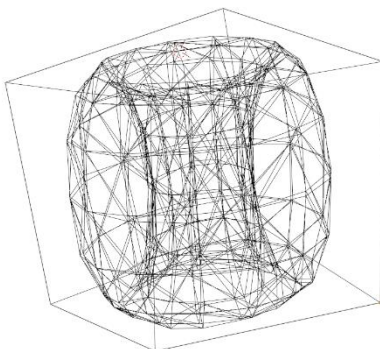


Спосіб 2

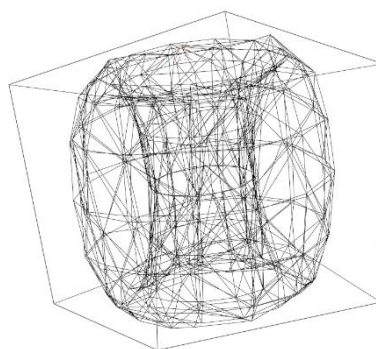


Спосіб 3

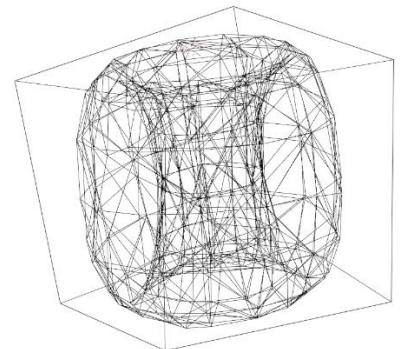
Рис. 3.13. Модель тора після моделювання деформації неоднорідного зростання за об'ємом з $f = 0.1$



Спосіб 1

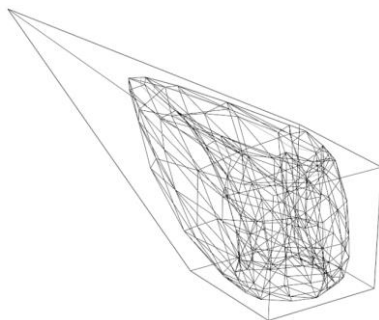


Спосіб 2

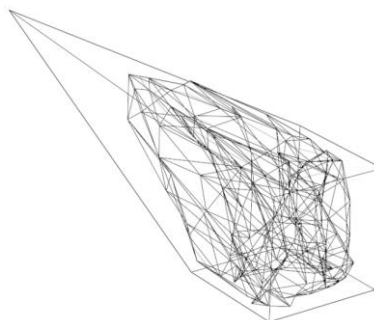


Спосіб 3

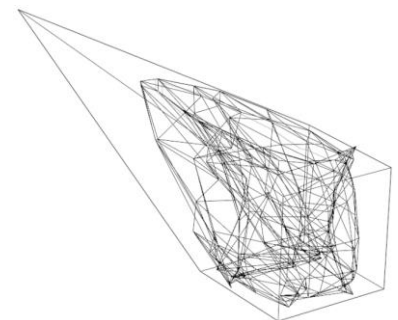
Рис. 3.14. Модель тора після моделювання деформації неоднорідного зростання за об'ємом з $f = 0.5$



Спосіб 1



Спосіб 2



Спосіб 3

Рис. 3.15. Модель тора після моделювання деформації неоднорідного зростання за об'ємом з $f = 1.0$

У процесі чисельних експериментів також було зафіксовано час обчислення для кожного з досліджуваних методів моделювання деформації. Результати вимірювань середнього часу виконання наведені в таблиці 3.1.

Таблиця 3.1. Середній час виконання розрахунків для кожного з методів

| | Спосіб 1 | Спосіб 2 | Спосіб 3 |
|---|----------|----------|----------|
| Середня швидкість виконання (мілісекунди) | 5.23 | 3.61 | 1.22 |

Слід зауважити, що представлені значення часу отримані для прямих реалізацій кожного із зазначених методів, без додаткових оптимізацій для прискорення їх виконання. Зокрема, у представленій реалізації не застосовувались методи паралельних або гетерогенних обчислень. Враховуючи специфіку задачі моделювання деформацій трикутникових сіток, використання саме таких підходів для оптимізації обчислень є перспективним напрямом і розглядається як предмет подальших досліджень.

Аналіз методів задання каркасів трикутникових сіток показав, що найбільш перспективним з точки зору оптимального співвідношення між точністю моделювання нелінійної деформації та обчислювальною ефективністю є метод 3 — задання вершин трикутнкової сітки як точок перетину площин трикутників, дотичних до цих вершин.

У першому експерименті при $d = 10$ похибка моделювання деформації для методу 3 була порівнянною з методами 1 та 2. Проте при зменшенні d у діапазоні $[9; 2]$ похибка методу 3 виявилася приблизно вдвічі меншою, ніж у методу 2, та утричі меншою, ніж у методу 1. У другому експерименті, при $f \in [0.2; 1.0]$, метод 3 також демонстрував нижчу похибку, ніж метод 2 (у два рази) та метод 1 (втричі).

Варто зазначити, що визначення положення окремої вершини, представленій як точка перетину площин, є однозначним лише у випадку, коли після перетворення три площини залишаються непаралельними. Для випадків із більшою кількістю площин у даній роботі запропоновано застосовувати метод псевдоінверсної матриці Мура-Пенроуза [70], який забезпечує однозначність

визначення положення вершини. Однак обчислювальна складність цього підходу зростає зі збільшенням кількості площин, що ускладнює ефективну паралелізацію обчислень у межах усього політочкового перетворення.

Вимірювання середнього часу виконання деформації для кожного з методів показало, що метод 3 забезпечує час виконання на рівні 1.22 мс, тоді як методи 1 та 2 потребували 5.23 мс та 3.61 мс відповідно.

3.4 Аналіз обчислювальної складності способів

Оцінювання обчислювальної складності методів моделювання деформацій є ключовим етапом при виборі алгоритмічного підходу для практичного застосування. Навіть метод, що забезпечує високу точність реконструкції геометрії, може виявитися непридатним для роботи з великими моделями через надмірні часові витрати або надмірне споживання обчислювальних ресурсів.

Оцінка обчислювальної складності проводиться для трьох алгоритмів, які використовуються для трьох методів задання геометрії

3.4.1 Алгоритм задання геометрії об'єкта через перетин площин трикутника та його нормалей

Алгоритм призначений для побудови набору площин, що описують локальну геометрію вершин трикуткової сітки шляхом представлення кожної вершини як точки перетину головної площини трикутника та двох допоміжних площин, ортогональних до його ребер, схематично зображений на рисунку 3.3. Такий підхід дозволяє описати топологію об'єкта у вигляді системи площин, що зберігають як інформацію про форму поверхні, так і орієнтацію в просторі. Процес побудови організовано у три послідовні етапи:

1. ініціалізація структури даних для представлення площин;

2. побудова площин для кожного трикутника та прив'язка їх до вершин;
3. завершення побудови та передача результатів.

На першому етапі виконується підготовка контейнерів для збереження результатів та проміжних обчислень, а саме:

1. створюється порожній список `planes`, який зберігатиме всі площини в порядку їх створення;
2. ініціалізується словник `planes_for_vertex_dict`, ключами якого є вершини (`Vertex`), а значеннями — порожні множини типу `TriPlaneSet`. Це забезпечує збереження зв'язків «вершина — множина площин» для подальших обчислень;
3. визначається допоміжна функція `Append_vertex_as_tri_plane(point, tri_plane)`, яка додає до словника набір трьох площин, пов'язаних із заданою вершиною.

Таким чином, на виході цього етапу маємо проініціалізовані структури, готові для заповнення даними у наступних кроках.

Другий етап є основною обчислювальною частиною алгоритму. Для кожного трикутника з сітки об'єкта виконується:

1. виділення координат вершин трикутника (P_1, P_2, P_3) із масиву вхідних вершин *vertexes* за індексами, вказаними у *triangles*;
2. побудова головної площини трикутника *triangle_plane*, що проходить через точки P_1, P_2, P_3 . Ця площина нормалізується для забезпечення числової стабільності подальших розрахунків;
3. генерація трьох допоміжних площин, ортогональних до ребер трикутника:
 - площина `p1p2_plane` — ортогональна до відрізка `p1p2`;
 - площина `p2p3_plane` — ортогональна до відрізка `p2p3`;
 - площина `p3p1_plane` — ортогональна до відрізка `p3p1`.

Кожна з цих площин будується як ортогональна до головної площини трикутника та вектора, що відповідає ребру.

4. Додавання площин у загальний список у визначеному порядку, що дозволяє відслідковувати індексацію (*plane_id* збільшується на 4 після кожного трикутника).

5. Прив'язка наборів площин до вершин:

- для вершини P1 зберігається трійка площин (triangle_plane, p1p2_plane, p3p1_plane);
- для вершини P2 — (triangle_plane, p1p2_plane, p2p3_plane);
- для вершини P3 — (triangle_plane, p2p3_plane, p3p1_plane).

В результаті кожна вершина буде представлена як точка перетину трьох площин, що зберігають локальну геометрію поверхні.

На заключному етапі алгоритму виконується перевірка узгодженості індексації площин: значення *plane_id* повинно відповідати індексу останньої площини у списку. Це гарантує, що всі площини були побудовані та пронумеровані без пропусків.

Результатом роботи функції є сукупність двох об'єктів:

- список усіх площин *planes*;
- словник відповідності вершин і трійок площин *planes_for_vertex_dict*.

Ці структури даних є вхідними для подальших етапів обчислень, таких як пошук нових положень вершин після деформацій через розв'язання задачі знаходження точки, найближчої до множини площин. На рисунку 3.16 представлена схема алгоритму.

Відновлення вершин після деформації об'єкта

Після того як поверхню подано через головні площини трикутників і три їхні ортогонали на ребрах, а самі площини пройшли політокове перетворення, потрібно відновити координати вершин деформованої сітки. У топології методу задання 1 кожна вершина представлена множиною трійок площин (по одній трійці від кожного інцидентного трикутника).

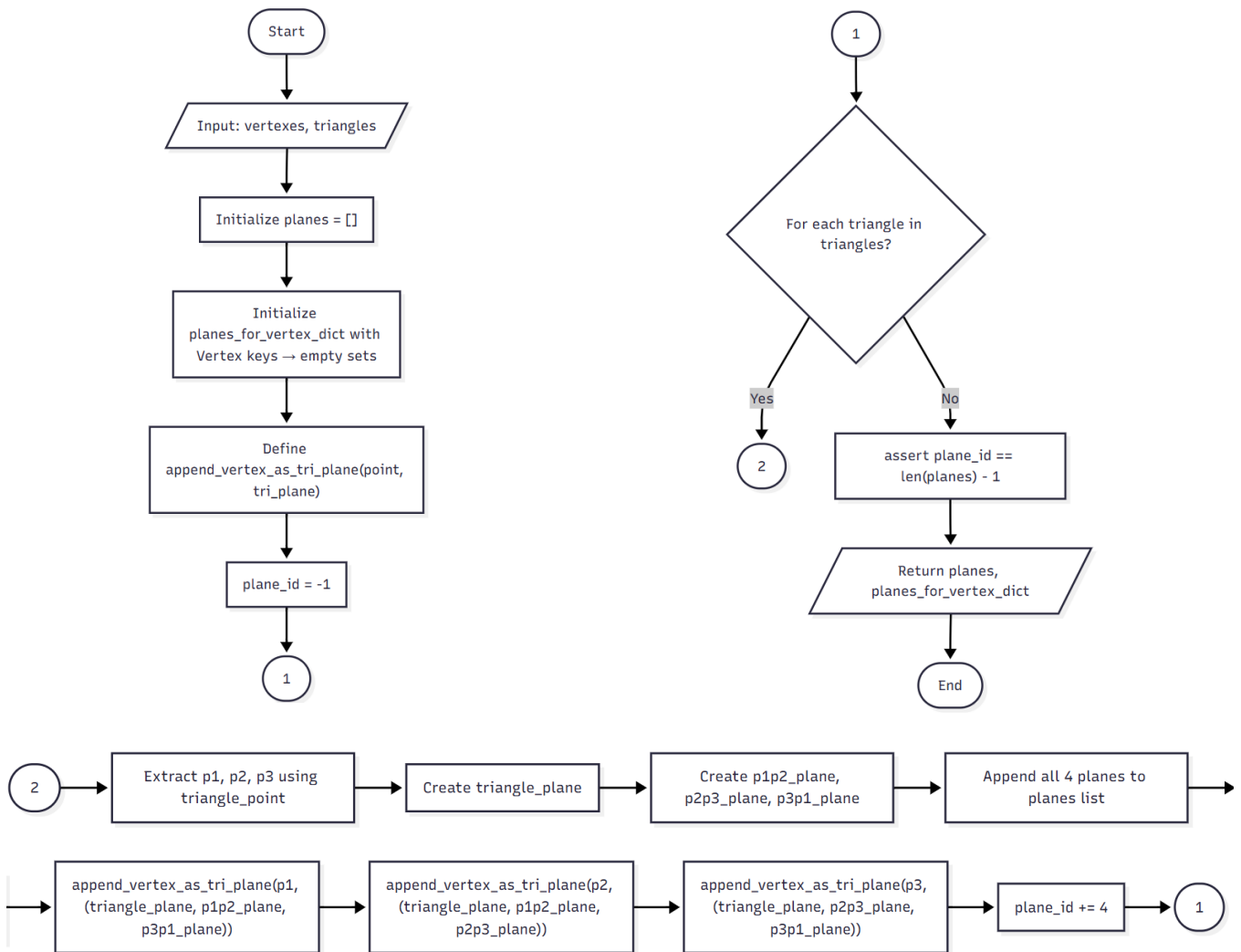


Рис. 3.16. Схема алгоритму задання геометрії для способу 1

Ідея алгоритму така: для кожної трійки беремо відповідні їй уже трансформовані площини, обчислюємо «під-вершину» як точку, що мінімізує суму квадратів відстаней до цих трьох площин, а потім усереднюємо всі під-вершини, що належать даній вершині (рис. 3.17).

Вхідні дані

1. Відображення «вершина — множина трійок площин до деформації».
2. Масив трансформованих площин з індексами, що збігаються з індексами вихідних площин.

Вихідні дані

Послідовність деформованих вершин (у тій самій логічній послідовності, що й вхідні).

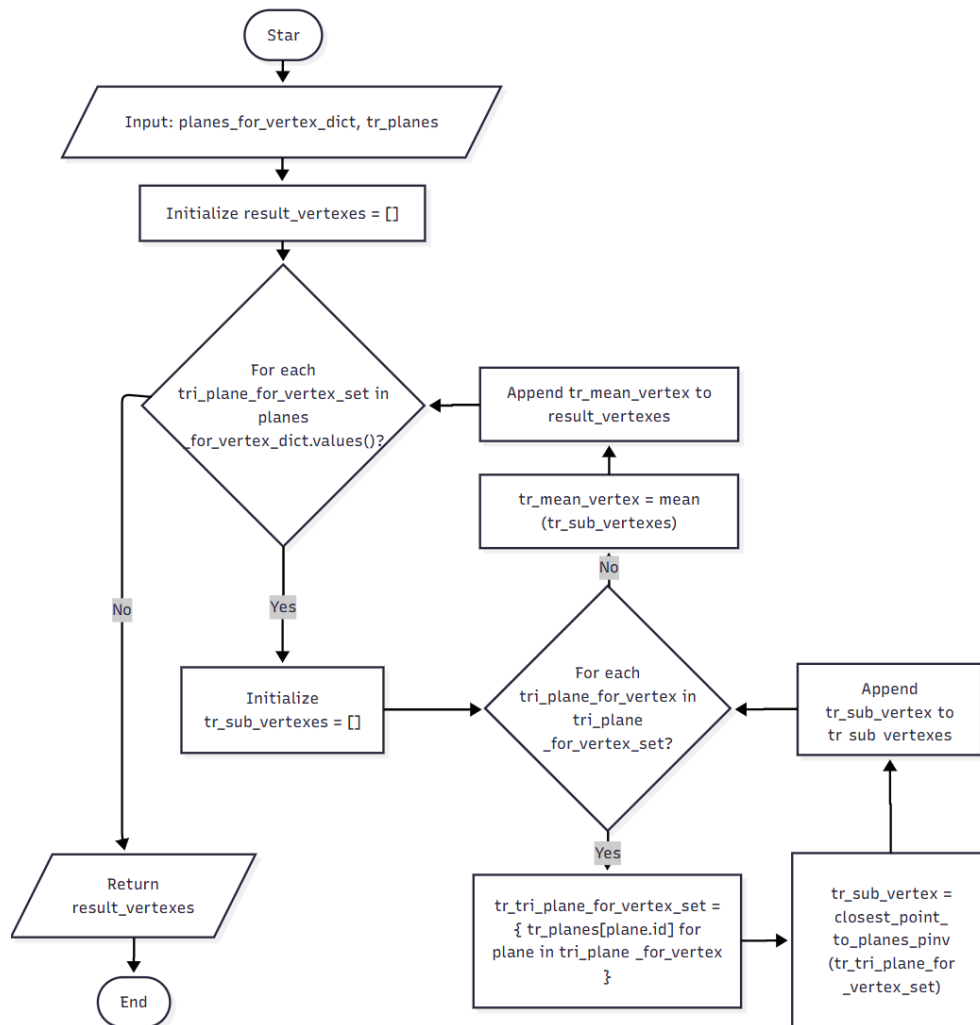


Рис. 3.17. Схема алгоритму відновлення геометрії для способу задання 1

Кроки алгоритму

1. Для фіксованої вершини перебрати всі її трійки площин. Кожну вихідну площину в трійці однозначно зіставити з трансформованою за її індексом (ідентифікатором).

На цьому забезпечується коректна відповідність «до/після» за рахунок індексації площин: кожна площина має свій сталий ідентифікатор, тому її образ після перетворення знаходиться без двозначностей.

2. Для кожної трансформованої трійки знайти «під-вершину» як точку, що найкраще узгоджує три площини (за нормою L_2). Практично це еквівалентно розв'язанню невеликої перевизначеної лінійної системи через псевдообернений оператор.

Перетин трьох площин трактують робастно: якщо геометрично точка перетину існує, обчислення її непотрібне; якщо ні — знаходиться така, що мінімізує суму квадратів відстаней до всіх трьох площин. Це усуває проблеми з виродженими або майже паралельними полігонами.

3. Зібрати всі під-вершини, пов'язані з цією вершиною, та обчислити їх середнє арифметичне — це й буде шукане положення деформованої вершини.

Усереднення зменшує локальні похибки окремих трійок і забезпечує узгодженість результату з усіма інцидентними трикутниками. За потреби звичайне середнє можна замінити на зважене (наприклад, з вагами, що враховують кути між нормаллями або кондиційність малої системи), але базовий алгоритм цього не вимагає.

4. Повторити кроки 1-3 для всіх вершин; повернути список отриманих точок.

У підсумку отримується поле деформованих вершин, яке є узгодженим із трансформованою системою площин та зберігає локальну геометрію поверхні, задану методом перетину площин трикутника і його нормалей.

3.4.2 Алгоритм задання геометрії об'єкта через перетин ортогональних площин трикутника

Даний алгоритм, який схематично представлено на рисунку 3.4, описує кожен вершину моделі як точку перетину трьох площин — дотичної до грані та двох ортогональних до її ребер.

Особливість цього підходу полягає в тому, що для кожної вершини зберігається лише одна трійка площин (*TriPlane*), яка формується під час першої зустрічі вершини в оброблюваних трикутниках. Це дозволяє уникнути дублювання та зменшити обсяг збережених даних.

Алгоритм складається з трьох послідовних частин:

1. ініціалізація структур даних — підготовка контейнерів для зберігання площин та зв'язків між ними і вершинами;
2. побудова площин для трикутників — створення головної та ортогональних площин, а також формування трійок площин для вершин;
3. фіналізація результатів — перевірка узгодженості індексації та повернення побудованих структур.

На першому етапі створюється порожній список *planes*, у який будуть послідовно додаватися всі площини. Словник *planes_for_vertex_dict* ініціалізується ключами-вершинами (*Vertex*), а значеннями є порожні об'єкти типу *TriPlane* — кортежі з трьох площин. Така структура гарантує, що кожна вершина матиме рівно три пов'язані площини.

Визначається допоміжна функція *Append_vertex_As_tri_plane(point, tri_plane)*, яка зберігає трійку площин у словнику для заданої вершини, перевіряючи, що кількість площин у трійці дорівнює трьом.

На другому етапі для кожного трикутника з вхідного списку *triAngles* виконується:

1. витягуються координати вершин $P1$, $P2$, $P3$ за індексами трикутника з масиву *vertexes*;
2. створюється головна площина *triangle_plane*, що проходить через точки $P1$, $P2$, $P3$. Площина нормалізується для забезпечення коректності подальших обчислень;
3. генеруються три ортогональні площини: *p1p2_plane* — ортогональна до відрізка $P1P2$; *p2p3_plane* — ортогональна до відрізка $P2P3$; *p3p1_plane* — ортогональна до відрізка $P3P1$.

Побудова відбувається шляхом обчислення векторного добутку нормалі головної площини та напрямного вектора відповідного ребра.

Усі чотири площини (головна та три ортогональні) додаються до списку *planes* у визначеному порядку, щоб зберегти коректну індексацію.

Для кожної вершини перевіряється, чи вже збережена її трійка площин у словнику *planes_for_vertex_dict*. Якщо ні — додається нова трійка.

На заключному етапі алгоритму після обробки всіх трикутників на виході з функції повертається: список площин *planes* і словник *planes_for_vertex_dict*, що зберігає для кожної вершини її трійку площин.

У попередньому алгоритмі (див. пункт 3.4.1) для кожної вершини зберігався набір усіх можливих трійок площин, отриманих з усіх суміжних трикутників. Це призводило до збільшення обсягу даних і потребувало додаткового усереднення для визначення кінцевого положення вершини після трансформацій. У поточному підході кожна вершина має рівно одну трійку площин, що зменшує обчислювальні витрати на етапі реконструкції і робить алгоритм більш ефективним при роботі з великими сітками, зберігаючи при цьому геометричну точність для ортогональних напрямків.

На рисунку 3.18 представлена схема алгоритму.

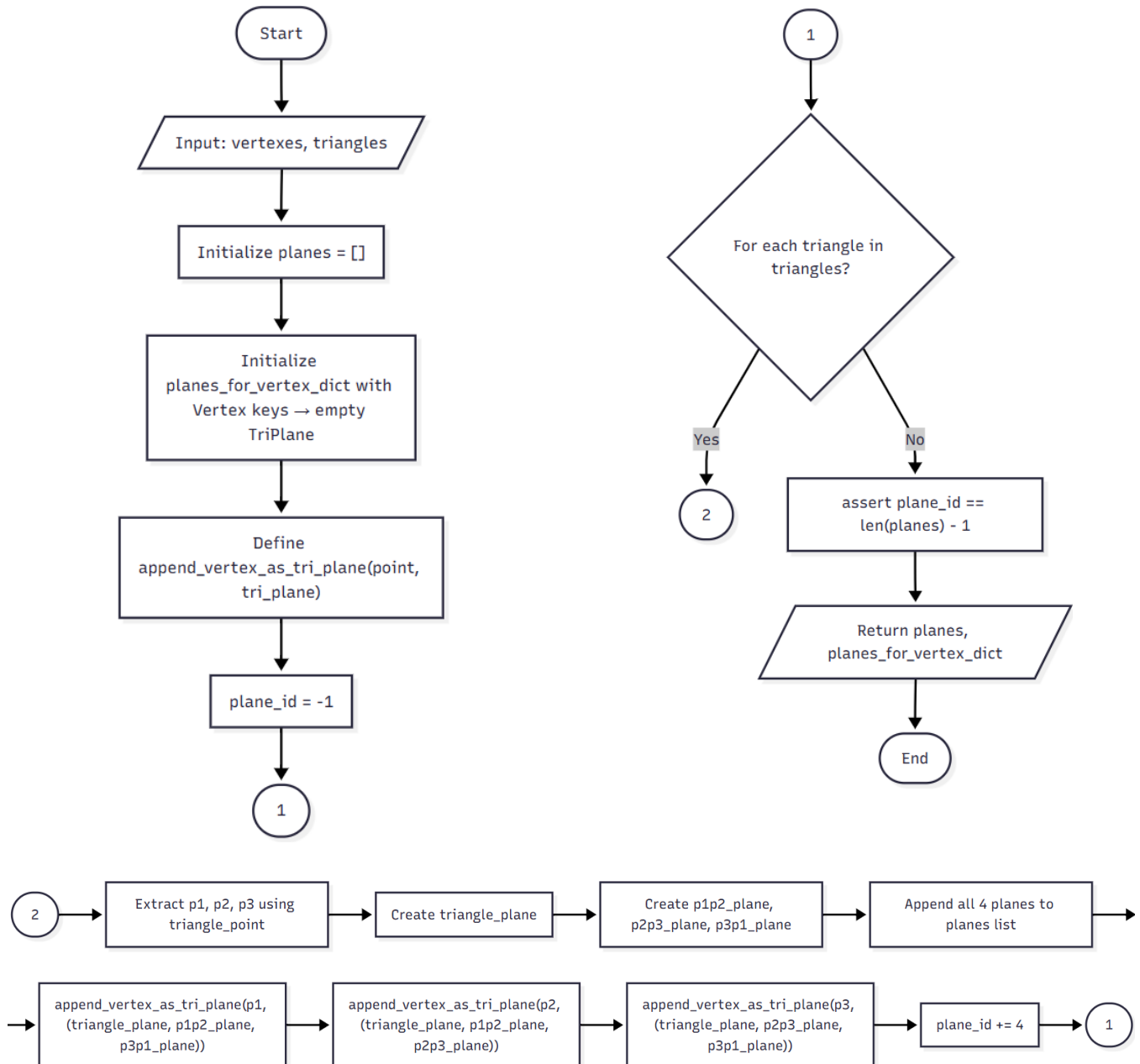


Рис. 3.18. Схема алгоритму задання геометрії способу 2

Алгоритм відновлення вершин після застосування політочкових перетворень геометрії об'єкта, заданої методом 2 ідентичний алгоритму відновлення вершин для геометрії, заданої методом 3 і буде описаний в підрозділі 3.4.3.

3.4.3 Алгоритм задання геометрії об'єкта через перетин площин дотичних трикутників

Алгоритм, схематично зображений на рисунку 3.5, буде для кожної грані сітки її дотичну площину та прив'язує ці площини до суміжних вершин. Вершина надалі відновлюється як точка, що мінімізує суму квадратів ортогональних відстаней до всіх дотичних площин, пов'язаних із цією вершиною. Така репрезентація локальної геометрії є лінійною за числом трикутників і не вимагає явної побудови ортогональних допоміжних площин уздовж ребер.

Процес побудови організовано у три послідовні етапи:

1. ініціалізація структури даних для представлення площин;
2. побудова площини для кожного трикутника та додавання її до множин, пов'язаних із трьома вершинами трикутника;
3. завершення побудови та передача результатів.

На першому етапі виконується підготовка контейнерів для збереження результатів та проміжних обчислень. На вхід подаються масиви *vertexes* і *triangles*. Створюється порожній список *planes*, що зберігатиме всі побудовані площини у порядку їх створення. Ініціалізується словник *planes_for_vertex_dict* з ключами типу *Vertex* та порожніми множинами як значеннями; це каркас для акумуляції всіх площин, інцидентних кожній вершині. Далі визначається допоміжна функція *append_vertex_as_planes(point, plane)*, яка виконує додавання площини до множини, пов'язаної з вершиною *point*, із перевіркою коректності ключа.

Другий етап є основною обчислювальною частиною алгоритму. Основний цикл проходить по всіх трикутниках сітки. Для кожного трикутника витягуються координати трьох вершин *p1*, *p2*, *p3*. На їх основі створюється площина *plane = Plane(id, p1, p2, p3)*, де коефіцієнти площини обчислюються за нормаллю, отриманою як векторний добуток двох напрямних у грані трикутника, а вільний

член визначається підстановкою однієї з вершин у рівняння площини. Побудована площина додається до списку *planes*. Далі ця ж площина послідовно асоціюється з кожною з трьох вершин трикутника викликами *append_vertex_as_planes(p1, plane)*, *append_vertex_as_planes(p2, plane)*, *append_vertex_as_planes(p3, plane)*. Індекс *id* узгоджується з порядком додавання, що полегшує подальше пряме звертання до трансформованих площин за їхніми ідентифікаторами.

На заключному етапі алгоритму після завершення обходу всіх трикутників функція повертає дві узгоджені структури: список площин *planes* і словник *planes_for_vertex_dict*. Перший є глобальним реєстром усіх побудованих дотичних площин, другий — відображенням «вершина — множина суміжних площин», яке на наступному етапі використовується для реконструкції положень вершин після перетворення площин. Такий інтерфейс повернення даних безпосередньо використовується подальшими процедурами деформації.

Обчислювальні і структурні властивості:

1. обсяг даних на кроці побудови є пропорційним числу трикутників (по одній площині на грань) і числу інцидентностей «грань — вершина» (по три прив'язки на грань). Тобто алгоритм лінійний щодо розміру сітки;
2. реконструкція вершини не потребує узгодження триплетів площин чи усереднення по варіантах: достатньо єдиної множини інцидентних площин, що спрощує реалізацію і прискорює обчислення.

У розділі 3.4.1 геометрія вершини задавалася перетином головної площини трикутника та двох допоміжних площин, ортогональних до його ребер. Це давало кожній вершині структурований триплет площин (*TriPlane*), з якого безпосередньо відновлювалась точка перетину; також контролювалася узгодженість індексації при додаванні чотирьох площин на грань. У підході алгоритму із цього розділу 3.4.3 відмовляємося від явних ортогональних допоміжних площин: замість цього використовуємо лише дотичні площини до граней і узгоджуємо вершину за всіма інцидентними площинами одразу. Це зменшує кількість побудов і спрощує

дані, але перекладає геометричні обмеження на розв’язок задачі найменших квадратів. Порівняно з варіантом, де для кожної вершини фіксується єдиний триплет (розділ 3.4.2), описаний тут підхід є більш універсальним щодо топологій із змінною валентністю та менше залежить від порядку обходу граней. Таким чином, методи 1 і 2 забезпечують жорсткі орієнтаційні обмеження за рахунок додаткових ортогональних площин, тоді як метод 3 мінімізує кількість примітивів і покладається на стійку реконструкцію із множини дотичних площин.

На рисунку 3.19 представлений алгоритм задання геометрії для способу 3.

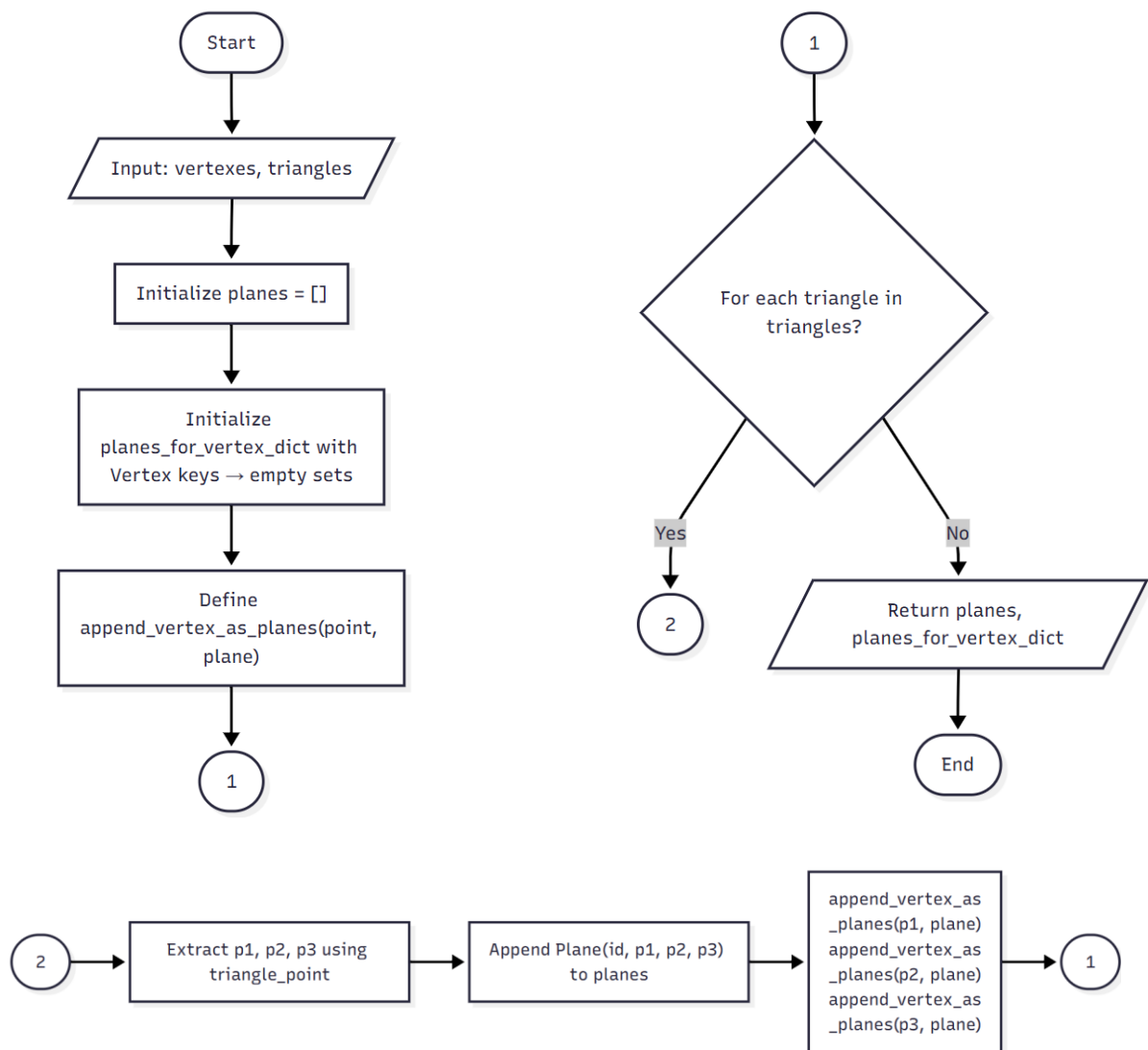


Рис. 3.19. Схема алгоритму задання геометрії для способу 3

Відновлення положення вершин після деформації як задача пошуку найближчої точки до множини площин

На попередніх підетапах (розділи 3.4.2-3.4.3) було сформовано представлення локальної геометрії у вигляді множин площин, інцидентних вершинам сітки: або як фіксовані трійки *TriPlane* для кожної вершини, або як множини дотичних площин до суміжних граней. На етапі деформації відповідні площини перетворюються методом політочкових перетворень, а положення вершин відновлюється як точка, що мінімізує суму квадратів відстаней до всіх перетворених площин. Схема, наведена нижче (рис. 3.20), реалізує саме цю процедуру.

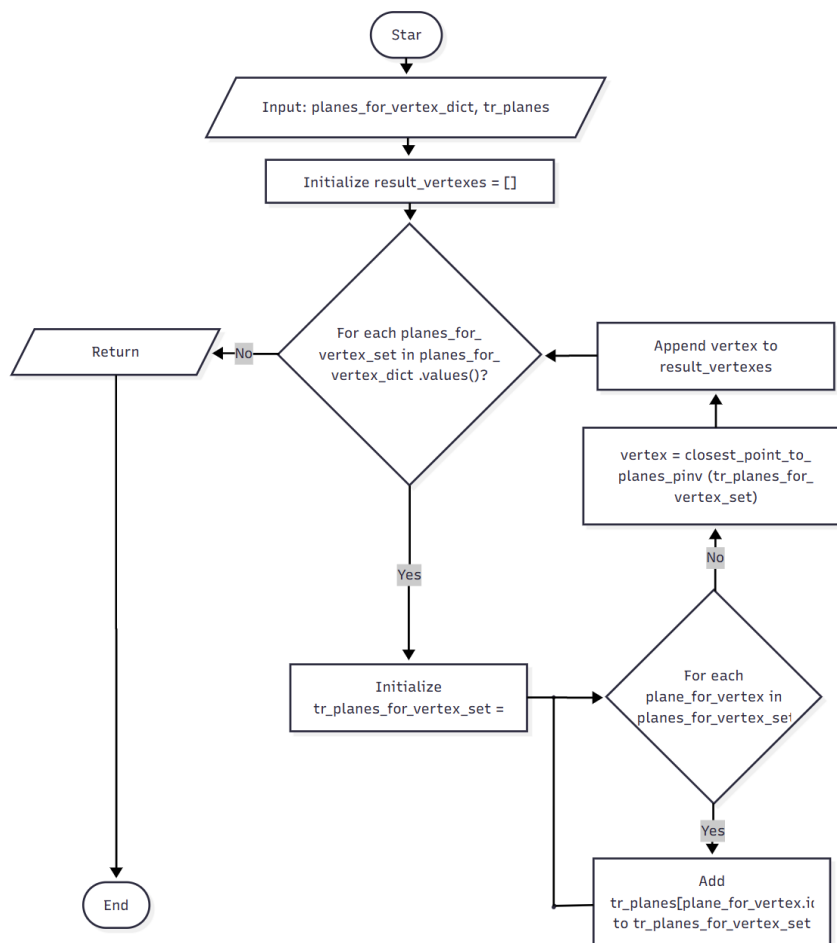


Рис. 3.20. Схема алгоритму відновлення геометрії для способу задання 1 та 2

На вхід алгоритми подається *planes_for_vertex_dict* — словник, що відображає вершину у множину пов'язаних з нею площин до деформації (для варіанта *TriPlane* — це множина з трьох площин; для варіанта дотичних площин — це всі суміжні дотичні площини); *tr_planes* — масив перетворених площин, індексованих за їхніми *id*.

Запишемо кроки алгоритму:

1. ініціалізується порожній список результатів *result_vertexes*;
2. для кожної вершини береться її набір початкових площин; за їхніми ідентифікаторами відбираються відповідні перетворені площини з *tr_planes* і акумулюються в локальну множину *tr_planes_for_vertex_set*;
3. обчислюється точка *vertex* як найближча (в евклідовій нормі) до всіх площин одночасно;
4. точка додається до *result_vertexes*. Після обходу всіх вершин повертається сформований масив оновлених координат.

Нехай для фіксованої вершини задано перетворені площини $\Pi_i : \mathbf{n}_i^\top \mathbf{x} + d_i = 0$ де $\|\mathbf{n}_i\| = 1$ [77, 78]. Шукаємо точку $\mathbf{x}^* \in \mathbb{R}^3$, що мінімізує суму квадратів орієнтованих відстаней:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \sum_i w_i (\mathbf{n}_i^\top \mathbf{x} + d_i)^2,$$

де $w_i > 0$ — ваги, що дозволяють компенсувати надійність площин. w_i можна обирати пропорційними площі суміжної грані або до косинуса дієдрального кута між сусідніми гранями, підсилюючи вклад площини.

Введемо матрицю $A \in \mathbb{R}^{k \times 3}$ з рядками $\sqrt{w_i} \mathbf{n}_i^\top$ та вектор $\mathbf{b} \in \mathbb{R}^k$ з компонентами $\mathbf{b}_i = -\sqrt{w_i} d_i$. Тоді маємо класичну задачу найменших квадратів:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2,$$

яка розв'язується через псевдообернену матрицю Мура-Пенроуза [70]:

$$\mathbf{x}^* = A^+ \mathbf{b} = (A^\top A)^{-1} A^\top \mathbf{b}.$$

У реалізації *closest_point_to_planes_pinv* ми використовуємо розклад SVD. Розкладаємо матрицю на прості частини, відсікаємо майже нульові сингулярні числа, за цим розкладом обчислюємо псевдообернену і отримуємо точку. Такий підхід дає стабільний результат навіть тоді, коли площини майже паралельні або майже лежать в одній площині.

Перед складанням A нормалі n_i приводяться до одиничної довжини; це робить внесок кожної площини інваріантним до масштабу параметризації.

У випадку близького до виродженого розміщення площин до нормальних рівнянь додається регуляризація Тихонова [79, 80] $(A^\top A + \lambda I)\mathbf{x} = A^\top \mathbf{b}$, $\lambda \ll 1$

3.4.4 Асимптотичний аналіз складності способів за нотацією Ландау

Обчислювальна складність безпосередньо пов'язана з масштабованістю алгоритму, тобто здатністю ефективно працювати при збільшенні розміру вхідних даних (кількості вершин, трикутників або площин) [81, 82]. Алгоритм з високою асимптотичною складністю може демонструвати прийнятну швидкість на невеликих сітках, але втрачатиме продуктивність на об'єктах із сотнями тисяч або мільйонами елементів [83]. Крім того, аналіз складності є необхідною передумовою для оптимізації та паралельної обробки. Нотація Ландау дозволяє виявити «вузькі місця» у структурі алгоритму, визначити, які обчислювальні операції є критичними за часом, та спрямувати зусилля на їх прискорення.

Проведемо аналіз вищезгаданих алгоритмів.

Позначемо:

V — кількість вершин;

T — кількість трикутників;

k_v — степінь вершини (кількість дотичних трикутників), $\sum_v k_v = 3T$ (для маніфолдових сіток без меж);

$\bar{N} = \frac{1}{V} \sum_v k_v$ — середній степінь вершини;

P — число площин, що трансформуються (P_{M1} — для алгоритму 1, P_{M2} — для алгоритму 2 і P_{M3} — для алгоритму 3);

p — кількість пар «початковий–кінцевий» базисів у політочковому перетворенні;

Так як політочкове перетворення однієї площини зводиться до формування системи лінійних рівнянь 4×4 по p базисах, $c_p = \Theta(p)$ — обчислювальна складність політочкового перетворення однієї площини.

Далі для кожного методу розділяємо витрати на три фази:

- побудова площин;
- перетворення площин;
- відновлення вершин.

1. Розрахуємо складність для алгоритма 1

1.а) Побудова площин

Для кожного трикутника створюється 4 площини (площина трикутника та три площини на основі нормалі до кожного ребра), див. ПЗ.4.1. Так як $P_{(M1)} = 4T$ Запишемо складність побудови як $\Theta(T)$.

1.б) Трансформація площин

Для кожної з $P_{(M1)}$ площин обчислювальна складність перетворення $c_p = \Theta(p)$. Тоді загальна обчислювальна складність політочкового перетворення відповідно $\Theta(P_{(M1)} \cdot c_p) = \Theta(T \cdot p)$.

1.в) Відновлення вершин

В алгоритмі 1 для вершини v зі степенем k_v маємо k_v триплетів площин (по одному від кожного дотичного трикутника). Кожен триплет розв'язується через псевдоінверсну матрицю Мура-Пенроуза 3×3 , тобто має константну складність. Після чого береться середнє по k_v підвершинах. Отже, відновлення вершин вартує $\sum_v \Theta(k_v) = \Theta(3T) = \Theta(T)$.

Загальна складність алгоритму 1 складає $\Theta(T) + \Theta(T \cdot p) + \Theta(T) = \Theta(T \cdot p)$.

2. Розрахуємо складність для алгоритма 2

2.а) Побудова площин

Для кожного трикутника створюється 3 ортогональні площини, див. ПЗ.4.2 .

Так як $P_{(M1)} = 3T$, запишемо складність побудови як $\Theta(V)$.

2.б) Трансформація площин

Для кожної з $P_{(M2)}$ площин обчислювальна складність перетворення $c_p = \Theta(p)$.

Тоді загальна обчислювальна складність політочкового перетворення відповідно $\Theta(P_{(M2)} \cdot c_p) = \Theta(3V \cdot p) = \Theta(V \cdot p)$.

2.в) Відновлення вершин

В алгоритмі 2 на вершину припадає один перетин трьох площин. Отже, відновлення вершин вартує $\Theta(V)$.

Загальна складність алгоритму 2 складає $\Theta(V) + \Theta(V \cdot p) + \Theta(V) = \Theta(V \cdot p)$.

3. Розрахуємо складність для алгоритма 3.

3.а) Побудова площин

Для кожного трикутника створюється 1 площини, див. ПЗ.4.2 . Тобто маємо складність побудови $\Theta(T)$.

3.б) Трансформація площин

Обчислювальна складність політочкового перетворення відповідно $\Theta(P_{(M3)} \cdot c_p) = \Theta(T \cdot p)$.

3.в) Відновлення вершин (наївна реалізація)

Розв'язання перевизначеної системи для кожної вершини за допомогою методів QR-розкладу або сингулярного розкладу (SVD) вимагає часу $\Theta(k_v^3)$. Таким чином, загальна складність відновлення всіх вершин визначається сумою: $\Theta\left(\sum_{v=1}^V k_v^3\right)$. Для типових трикутникових сіток без меж, середній ступінь вершини \bar{N} сталий (зазвичай 5–8), тож отримуємо асимптотичну оцінку $\Theta(V \cdot \bar{N}^3)$.

Відновлення вершин (оптимізована реалізація)

Оскільки кількість невідомих фіксована у трьохвимірному просторі, розв'язання системи нормальних рівнянь можна здійснювати за лінійний час відносно кількості площин [84]. Формування матриці нормальних рівнянь $\mathbf{A}_v^\top \mathbf{A}_v$ та правої частини займає $\Theta(k_v)$, а розв'язання малої системи розмірності 3×3 займає $\Theta(1)$ (таблиця 3.2).

Таблиця 3.2. Складність алгоритмів

| Алгоритм | Топологія подання | Площин на 1 елемент | Загалом площин P | Складність побудови | Складність перетворення | Складність відновлення | Складність загальна |
|----------|---------------------------------------|---------------------|--------------------|---------------------|-------------------------|------------------------|---------------------|
| 1 | Декілька <i>TriPlane</i> на 1 вершину | 4 / трикутник | $4T$ | $\Theta(T)$ | $\Theta(T \cdot p)$ | $\Theta(T)$ | $\Theta(T \cdot p)$ |
| 2 | 1 <i>TriPlane</i> на вершину | 3 / вершина | $3V$ | $\Theta(V)$ | $\Theta(V \cdot p)$ | $\Theta(V)$ | $\Theta(V \cdot p)$ |
| 3 | Дотичні трикутники | 1 / трикутник | T | $\Theta(T)$ | $\Theta(T \cdot p)$ | $\Theta(T)$ | $\Theta(T \cdot p)$ |

В такому разі складність реконструкції всіх вершин складає:

$$\Theta\left(\sum_{v=1}^V k_v\right) = \Theta(3T) = V \cdot \bar{N} = \Theta(T), \text{ оскільки } \sum_{v=1}^V k_v = 3T \text{ для замкнених сіток.}$$

Таким чином загальна складність алгоритму 3 складає $\Theta(T) + \Theta(T \cdot p) + \Theta(T) = \Theta(T \cdot p + 2T) = \Theta(T \cdot p)$.

В таблиці 3.2 наведено узагальнені результати розрахунків для розглянутих способів.

3.5. Методологія вибору параметра регуляризації

Ядром методу *getPolypointPlane* для обчислення перетвореної площини є розв’язання СЛАР виду $A \cdot X = B$ розмірності 4×4 . У цій системі A є матрицею, що формується шляхом накопичення сум, а X — вектором шуканих коефіцієнтів нової площини a', b', c', d' . Детальний аналіз реалізації алгоритму виявив, що елементи матриці A та вектора B формуються з доданків, що містять обернені величини $1/\gamma$ та $1/\gamma^2$, де γ позначає знакову відстань від вхідної базисної точки *orig_basis_p* до вхідної площини *plane*. Така побудова системи створює фундаментальну вразливість до чисельної нестійкості у вироджених геометричних конфігураціях [85, 86, 87].

3.5.1 Визначення проблеми чисельної нестійкості у вироджених випадках

Перша обумовленість системи виникає, коли хоча б одна вхідна базисна точка лежить точно на вхідній площині. У цьому випадку знакова відстань γ дорівнює нулю, що при спробі обчислення $1/\gamma^2$ призводить до ділення на нуль та

виключення з плаваючою комою (FPE). Це унеможливило подальше формування матриці A та, відповідно, розв'язання системи [88, 89].

Другий, більш поширений тип обумовленості пов'язаний з особливостями арифметики з плаваючою комою (зокрема, типом *double*). На практиці γ рідко буває точно нулем, однак може набувати малих значень (наприклад, 10^{-15} або 10^{-30}). Це призводить до того, що γ^2 стає катастрофічно малим, а обернені величини $1/\gamma^2$ — надвеликими. Внаслідок цього матриця A заповнюється домінуючими, надвеликими числами, що робить її погано обумовленою. Розв'язок СЛАР для такої матриці, наприклад, методом LU-розкладання, стає чисельно нестійким. У процесі обчислень накопичуються значні похибки, і результуючий вектор коефіцієнтів X містить нескінченність або просто аномально великі числа, що не мають фізичного сенсу [90, 91, 92, 93].

Третій тип проблеми обумовленості матриці пов'язаний не з відстанню γ , а з конфігурацією вихідних базисних точок. Навіть якщо всі значення γ є нормальними і не близькими до нуля, матриця A може стати сингулярною або близькою до неї через виродженість вихідного базису [86, 94]. Це трапляється, наприклад, якщо всі точки результуючого базису мають однакову x -координату або всі лежать на одній прямій [95, 96, 97, 98]. Така конфігурація призводить до лінійної залежності рядків або стовпців матриці A , внаслідок чого її детермінант стає нульовим або близьким до нуля. Це, у свою чергу, робить неможливим знаходження єдиного стабільного розв'язку системи $A \cdot X = B$ стандартними методами лінійної алгебри [99, 100].

3.5.2 Метод регуляризації для стабілізації розв'язку

Для подолання описаних трьох типів обумовленості та забезпечення робастності алгоритму при обробці довільних вхідних даних було обрано метод,

що широко використовується у розв'язанні обернених та погано обумовлених задач — метод регуляризації Тихонова [79, 80]. У лінійній алгебрі та статистиці цей підхід також відомий як «демпфування» або «гребенева регресія».

Суть методу полягає у модифікації вихідної системи $A \cdot X = B$. Замість неї розв'язується дещо зміщена, але гарантовано стабільна система:

$$(A + \lambda I) \cdot X = B,$$

де I — одинична матриця відповідної розмірності (4x4 у нашому випадку),

λ — мала позитивна скалярна величина, що є константою регуляризації (в Додатку Б-Е позначена як *reg_term*).

Обґрунтування ефективності цього методу полягає в наступному [101]. Матриця A , що формується в алгоритмі, за своєю побудовою є симетричною і, у невинроджених випадках, позитивно визначеною. Додавання позитивної константи λ до кожного діагонального елемента матриці A гарантує, що результуюча матриця $(A + \lambda I)$ стає строго позитивно визначеною. Згідно з фундаментальними теоремами лінійної алгебри, строго позитивно визначена матриця завжди є не винродженою тобто детермінант, що строго більший за нуль. Це гарантує існування єдиного розв'язку системи.

Практичний ефект від застосування регуляризації вирішує всі ідентифіковані проблеми. По-перше, він нейтралізує сингулярності типу 1 та 2: навіть якщо γ є нулем або надзвичайно малим числом, і діагональні елементи A (такі як a_1, b_2, c_3) прямують до нескінченності, додавання λ стабілізує їх, зберігаючи числову стійкість матриці. По-друге, метод вирішує проблему геометричної обумовленості: якщо вихідна матриця A була сингулярною через лінійну залежність її стовпців (і, відповідно, мала нульові власні числа), то модифікована матриця $(A + \lambda I)$ матиме

всі власні числа $\geq \lambda$. Це ефективно зсуває спектр матриці від нуля, роблячи її обумовленою.

3.5.3 Експериментальне визначення оптимального значення регуляризації λ

Вибір оптимального значення константи регуляризації λ (*reg_term*) є нетривіальною задачею, що вимагає балансування між чисельною стійкістю та точністю розв'язку. Цей вибір залежить від кількох ключових факторів. По-перше, це точність обчислень: оскільки в реалізації алгоритму використовується 64-бітна арифметика з плаваючою комою на базі процесорних інструкцій SSE2 (64-бітний *double* за стандартом IEEE 754), машинний епсилон ϵ становить приблизно $2.2 \cdot 10^{-16}$ [102]. Щоб λ мав стабілізуючий ефект, його значення має бути суттєво більшим за ϵ . По-друге, це масштаб вхідних даних: передбачається, що координати точок та коефіцієнти площин знаходяться у розумному діапазоні (наприклад, від 10^{-3} до 10^3). За умов роботи з даними кардинально іншого порядку (наприклад, 10^{10}) значення λ потребувало б відповідного масштабування.

Ключовим фактором є компроміс зміщення проти стійкості. З одного боку, занадто мале значення λ (наприклад, 10^{-20}) виявиться недостатньо сильним для стабілізації системи у важких випадках поганої обумовленості (проблеми типу 2 та 3). З іншого боку, занадто велике значення (наприклад, 1.0) хоч і гарантує абсолютну стабільність, але вносить значне зміщення (похибку) у розв'язок. Це призводить до того, що результуюча площина буде помітно неточною, що є неприйнятним.

Для емпіричного визначення оптимального λ було проведено серію обчислювальних експериментів. Мета полягала у знаходженні так званого плато — діапазону значень λ , в якому вже досягається 100% чисельна стійкість, але

похибка, що вноситься регуляризацією, залишається мінімальною, меншою за інші джерела похибок. Для цього було сформовано два тестові набори даних. Перший, «нормальний» набір, складався з 10 000 випадково згенерованих площин та базисів, де всі знакові відстані γ гарантовано були більшими за 0.1. Для цього набору існує еталонний розв’язок, обчислений без регуляризації. Другий, синтетичний набір, включав 1000 спеціально сконструйованих тестів, що моделювали всі вироджені випадки: $\gamma = 0$ (точка на площині), γ дуже близький до нуля (10^{-10} , 10^{-15}), а також вироджені конфігурації результуючих базисних точок, такі як колінеарні чи копланарні точки.

Методологія експерименту полягала у багаторазовому запуску алгоритму *getPolypointPlane* (Додаток Б) на обох наборах даних, варіюючи значення λ за логарифмічною шкалою $(10^{-18}, 10^{-15}, 10^{-12}, 10^{-10}, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1.0)$. Оцінка проводилася за двома ключовими метриками:

Стійкість — відсоток успішних запусків (що не призвели до нескінченності або невизначеності або програмних виключень) на синтетичному наборі;

Точність — середньоквадратична похибка (RMSE) отриманих коефіцієнтів площини порівняно з «еталонним» розв’язком на «нормальному» наборі.

3.5.4 Аналіз результатів та обґрунтування вибору λ

На рисунку 3.21 показаний графік, що ілюструє компроміс між точністю та стійкістю алгоритму в залежності від значення константи λ . На графіку вісь X відображає λ у логарифмічному масштабі. Ліва вісь Y показує відсоток успішних запусків (стійкість), а права вісь Y — середньоквадратичну похибку (RMSE) коефіцієнтів площини (точність), також у логарифмічному масштабі. Синя лінія показує, як стійкість різко зростає від низьких значень λ до 100% і залишається там. Червона лінія демонструє, як похибка залишається низькою, а потім починає

зростати при більших значеннях λ . Вертикальна пунктирна лінія позначає обране значення $\lambda = 10^{-6}$.

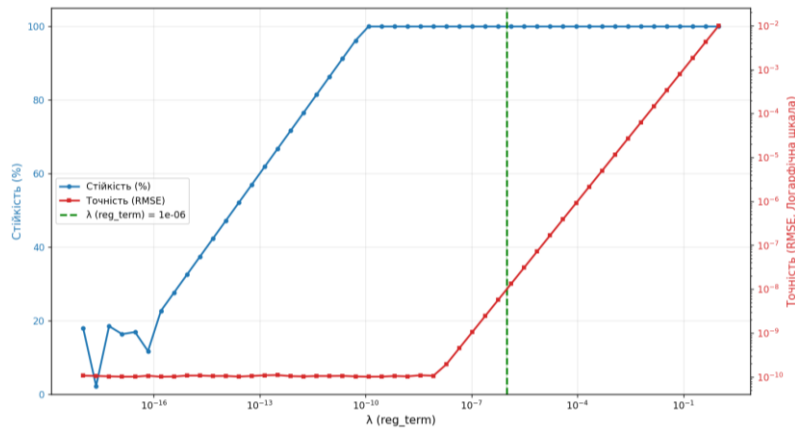


Рис. 3.21 Залежність стійкості та точності від λ

На рисунку 3.22 зображено залежність відсотка успішних запусків алгоритму (Стійкість) від константи регуляризації λ в діапазоні $[10^{-20}; 10^{-9}]$, причому вісь λ представлена у логарифмічному масштабі.

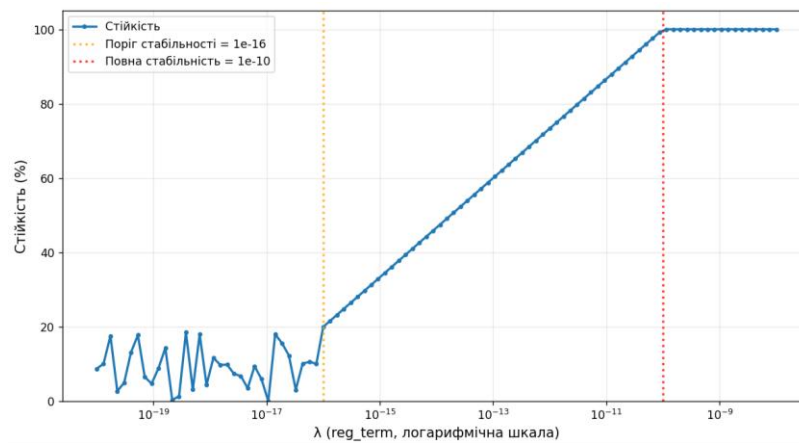


Рис. 3.22 Залежність стійкості від λ для малих λ

Синя лінія, власне, крива залежності стійкості від λ . Вона показує, що при дуже малих λ (близьких до 10^{-20}) стійкість практично нульова, потім спостерігаються коливання, а після певного порогу вона починає різко зростати.

Важливо розуміти, що залежність не є лінійною в абсолютному сенсі, а є лінійною на логарифмічній шкалі X .

При дуже малих λ ($< 10^{-17}$), його вплив на стабільність незначний, оскільки він губиться серед похибок обчислень з плаваючою комою, не здатний протистояти діленню на нуль або дуже малим γ . Коли λ наближається до машинного епсилону ($\approx 10^{-16}$) і далі, він починає ефективно змінювати властивості матриці A . У діапазоні $[10^{-16}; 10^{-10}]$ спостерігається плавний перехід від повної нестійкості до повної стабільності. Ця лінійна поведінка на логарифмічній шкалі λ у перехідній зоні свідчить, що реакція системи пропорційна порядку величини параметра, а не його абсолютному значенню, що є типовим для чисельних процесів.

На рисунку 3.23 зображено залежність середньоквадратичної похибки (RMSE) розв'язку від константи регуляризації λ у логарифмічних масштабах. При малих λ (до 10^{-9}) точність висока (RMSE близько 10^{-10}). Після $\lambda \approx 10^{-8}$ похибка починає значно зростати, демонструючи, як надмірна регуляризація погіршує точність розв'язку. Вертикальні лінії позначають початок погіршення 10^{-8} , значне погіршення 10^{-6} та обране значення $\lambda = 10^{-6}$.

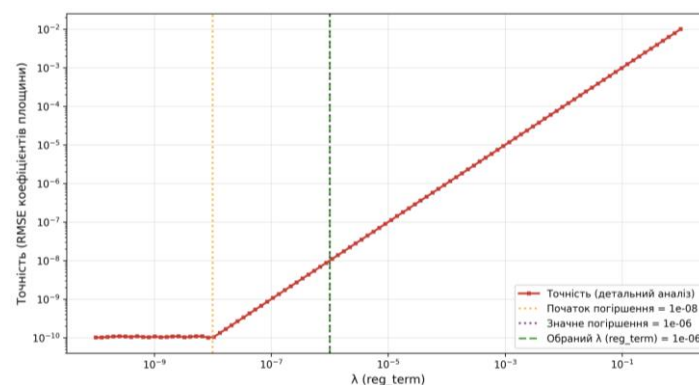


Рис. 3.23 Залежність точності від λ для великих λ

Представлені гістограми (рис. 3.24-3.25) візуалізують розподіл значень знакової відстані γ для двох типів тестових наборів даних: «Нормального» та

«Важкого», що є ключовим для обґрунтування необхідності чисельної регуляризації.

Перша пара діаграм відображає загальний розподіл γ . Для «Нормального» набору (рис. 3.24, ліворуч) розподіл значень γ є відносно рівномірним, зосередженим переважно в діапазоні від 0.1 до 1.0, із середнім значенням близько 0.554. Важливо, що в цьому наборі практично відсутні значення γ , близькі до нуля, що свідчить про його добре обумовлений характер.

Натомість для «Важкого» набору (рис. 3.24, праворуч) спостерігається яскраво виражений пік щільності розподілу в області малих значень γ , а середнє значення значно зміщене до 0.117. Це підтверджує, що «Важкий» набір цілеспрямовано містить велику кількість вироджених випадків, що створюють чисельні проблеми.

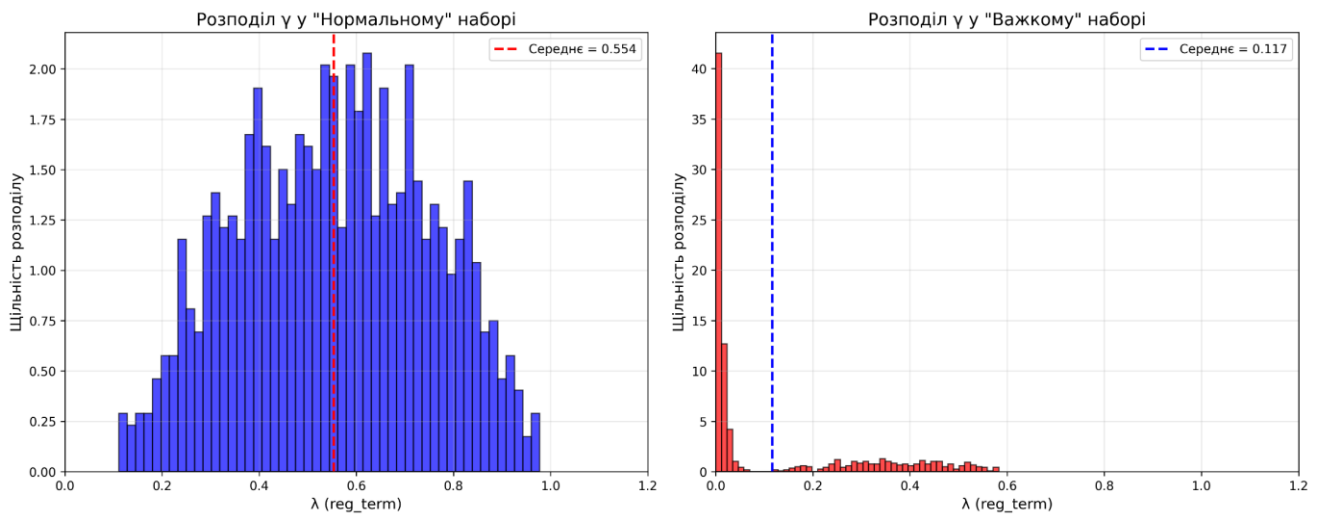


Рис. 3.24 Розподіл γ у «нормальному» наборі та «важкому»

Друга пара діаграм (рис. 3.25) пропонує детальніший аналіз розподілу малих значень γ з використанням логарифмічної шкали по осі X, що дозволяє дослідити критичні діапазони.

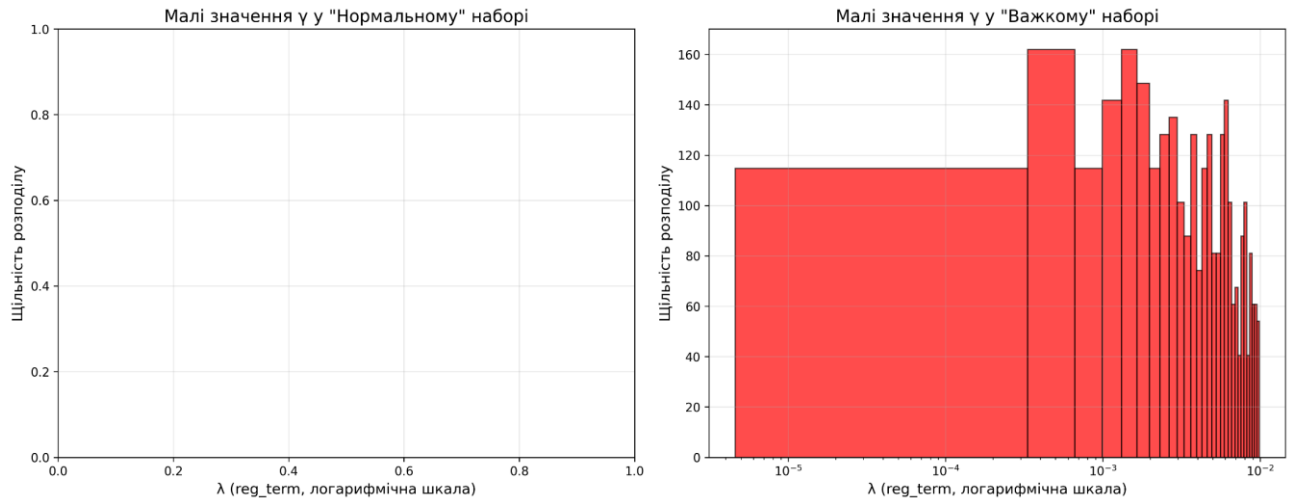


Рис. 3.25 Розподіл малих значень γ у «нормальному» наборі та «важкому»

Для «Нормального» набору (рис. 3.25, ліворуч) графік є порожнім, демонструючи повну відсутність значень γ в діапазоні $[10^{-5}; 10^{-2}]$. Це є очікуваним результатом і підтверджує, що «Нормальний» набір не містить критично малих γ . У випадку «Важкого» набору (рис. 3.25, праворуч) графік показує значну кількість випадків, розподілених по кількох порядках малих значень γ . Це свідчить про ретельну розробку «Важкого» набору, який тестує алгоритм не лише на нульових, а й на майже сингулярних значеннях, що є критично важливим для оцінки стійкості.

Проведений аналіз та експериментальні дослідження підтверджують, що проблема чисельної нестійкості при обробці вироджених геометричних конфігурацій є неминучою для даного типу алгоритмів, що базуються на розв'язанні СЛАР. Прямий підхід виявляється вразливим до випадків, коли базисні точки лежать на площині (або близько до неї), що призводить до ділення на нуль або поганої обумовленості матриці системи. Впровадження методу регуляризації Тихонова продемонструвало свою високу ефективність у вирішенні цієї проблеми.

Експериментальне визначення оптимального параметра регуляризації λ дозволило ідентифікувати безпечне плато $\lambda \in [10^{-10}; 10^{-5}]$, в якому одночасно досягається повна чисельна стійкість та зберігається висока точність обчислень.

Обране значення $\lambda = 10^{-6}$ є теоретично та емпірично обґрунтованою константою. Воно є достатньо великим, щоб подолати чисельні неточності, пов'язані з машинним епсилоном, та забезпечити стійкість алгоритму на «важкому» наборі даних. Водночас воно є достатньо малим, щоб не вносити суттєвого зміщення (похибки) у розв'язок для невироджених випадків на «важкому» наборі даних, де внесена похибка є значно меншою за очікувані шуми у вхідних даних.

Незважаючи на ефективність регуляризації, неоптимальний вибір константи може призвести до недостатньої стійкості або надмірної втрати точності, особливо для даних із змінним масштабом.

Висновки до третього розділу

У третьому розділі дисертаційної роботи було вперше запропоновано спосіб представлення об'єкта перетворень, який враховує топологію дискретного представлення, що дозволяє здійснювати політочкові перетворення поверхонь заданих полігональними сітками. Ключовою задачею було визначення оптимального способу подання геометрії полігональної моделі у вигляді набору площин, що є необхідною умовою для застосування методу політочкових перетворень.

У ході дослідження було проаналізовано три підходи до представлення геометрії об'єкта:

1. Представлення кожного трикутника через його власну площину та три допоміжні площини, утворені нормаллю та сторонами. Було виявлено, що цей метод призводить до порушення топологічної цілісності сітки. Для вирішення цієї проблеми запропоновано модифікацію, що полягає в усередненні координат спільних вершин, обчислених для кожного суміжного трикутника незалежно.

2. Представлення кожної вершини як точки перетину трьох взаємно ортогональних площин. Цей підхід забезпечує збереження топології сітки, однак є анізотропним (залежним від вибору системи координат) і не враховує локальну геометрію поверхні, що може знижувати якість моделювання складних деформацій.
3. Представлення кожної вершини як точки перетину площин усіх дотичних до неї трикутників. Цей метод не лише зберігає топологію, але й безпосередньо враховує локальну структуру сітки в процесі деформації. Для випадків, коли до вершини дотичні більше трьох трикутників, було запропоновано використовувати метод псевдоінверсної матриці Мура-Пенроуза для знаходження точки, що мінімізує суму квадратів відстаней до всіх перетворених площин.

Для порівняльного аналізу розроблених методів було проведено серію чисельних експериментів на прикладі моделі тора. Моделювалися два типи аналітично заданих деформацій: *скручування* навколо осі та *неоднорідне зростання за об'ємом*. Точність кожного методу оцінювалася шляхом обчислення середньоквадратичної похибки (RMSE) між результатом політочкового перетворення та еталонним об'єктом, деформованим за прямою аналітичною формулою.

Результати експериментального дослідження показали наступне:

За точністю моделювання третій метод продемонстрував значну перевагу над іншими. При інтенсивних деформаціях скручування (параметр $d \in [9; 2]$) його похибка була приблизно вдвічі меншою, ніж у другого методу, та втричі меншою, ніж у першого. Аналогічна тенденція спостерігалася і для деформації неоднорідного зростання.

За обчислювальною ефективністю третій метод також виявився найшвидшим серед запропонованих, із середнім часом виконання 1.22 мс, що значно перевершує

показники першого (5.23 мс) та другого (3.61 мс) методів у рамках проведених тестів на неоптимізованих реалізаціях.

На основі проведеного аналізу точності, візуальної якості результатів та обчислювальної складності можна зробити висновок, що спосіб задання вершин трикутничкової сітки як точок перетину площин дотичних трикутників є найбільш перспективним. Він забезпечує оптимальне співвідношення між високою точністю моделювання нелінійних деформацій, збереженням топології та локальної геометрії об'єкта, а також високою обчислювальною ефективністю.

Обґрунтовано вибір методу регуляризації Тиханова для вирішення проблеми погано обумовлених СЛАР а також експериментально показано доцільність використання константи регуляризації $\lambda = 10^{-6}$.

Разом з тим, подальші дослідження будуть спрямовані на оптимізацію алгоритмів, зокрема шляхом застосування методів паралельних та гетерогенних обчислень для підвищення продуктивності при роботі з високополігональними моделями.

РОЗДІЛ 4. РОЗРОБКА МЕТОДІВ ПОЛІТОЧКОВИХ ПЕРЕТВОРЕНЬ НА ОСНОВІ ГЕТЕРОГЕННИХ ОБЧИСЛЕНЬ

Емпіричний та теоретичний аналіз з розділу 3 показав, що подання геометрії «вершина як перетин площин дотичних трикутників» (метод 3) забезпечує найкращий компроміс між точністю моделювання нелінійних деформацій та обчислювальною ефективністю серед розглянутих способів (метод 1: площа трикутника + нормалі до ребер; метод 2: ортогональні площини у вершині). Зокрема, при інтенсивних скрутках та неоднорідному зростанні метод 3 давав у 2–3 рази менший RMSE, ніж альтернативи, і водночас демонстрував менший середній час виконання на неоптимізованій реалізації (≈ 1.22 мс проти 3.61-5.23 мс) [6]. Саме ці результати мотивують прийняти метод 3 як базове представлення для подальшої інженерної оптимізації та масштабування обчислень.

Важливо підкреслити, що оцінки часу у вказаних експериментах отримані для послідовних або наївних реалізацій без спеціальних засобів прискорення. Відповідно, реальний потенціал продуктивності способу 3 лишається повністю не розкритим і на пряму пов'язаний із можливістю розпаралелити два домінуючі етапи (1) масову трансформацію площин політочковими перетвореннями та (2) відновлення положення вершин як розв'язок малих задач найменших квадратів.

4.1 Багатопотокова паралелізація методу політочкових перетворень

4.1.1 Архітектура рішення і алгоритм підготовки даних та запуску процесу обробки на багатоядерних CPU

Метод політочкових перетворень площин застосовується для деформації геометричних об'єктів у тривимірному просторі. У загальному вигляді деформація

площини задається рівнянням 3.1. Сутність підходу полягає в поданні кожної вершини триангульованого каркаса як перетину множини площин, утворених інцидентними трикутниками. Для вершини A , що визначається чотирма площинами $\alpha, \beta, \gamma, \theta$, після застосування політочкових перетворень одержуємо перетворені площини $\alpha', \beta', \gamma', \theta'$, перетин яких задає шукану вершину A' (рис. 4.1-4.2).

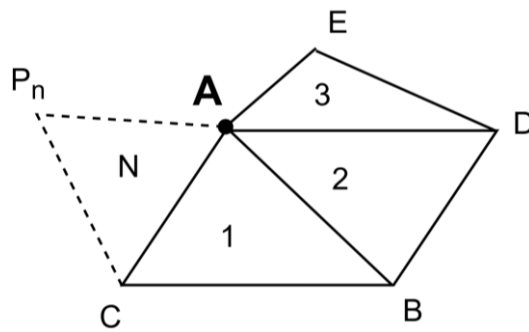


Рис. 4.1. Вершина A , задана площинами, утвореними з $\triangle ABC$, $\triangle ABD$, $\triangle ADE$ та ін.

Таким чином, обчислювальна задача зводиться до перетворення списку вхідних площин у список перетворених площин із подальшим відновленням вершин як їх перетинів.

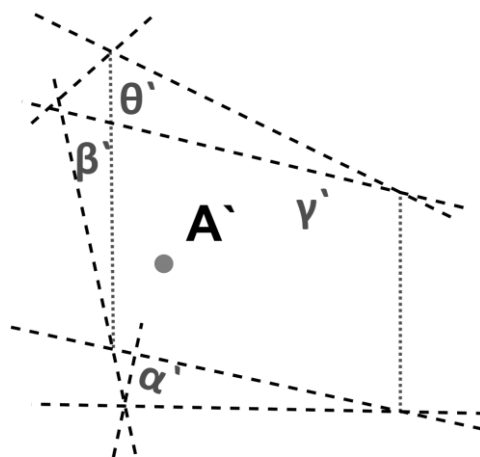


Рис. 4.2. Площини $\alpha', \beta', \gamma', \theta'$ після застосування ПТП та шукана вершина A'

Ключова властивість політочкових перетворень у цій постановці — незалежність перетворення окремих площин: результат для будь-якої площини не залежить від результатів для інших. Це дає можливість прямого розпаралелювання без синхронізацій усередині критичного ядра обчислень [5].

Запропонований алгоритм має такі кроки:

1. розбити вхідний список площин на N підсписків приблизно однакового розміру;
2. створити N паралельних потоків виконання;
3. у кожному потоці виконати обчислення політочкових перетворень для свого підписку, результати записати в тимчасовий буфер;
4. дочекатися завершення всіх потоків (бар'єр);
5. об'єднати тимчасові списки в єдиний результуючий список.

Завдяки такій організації накладні витрати обмежуються створенням потоків і завершальним злиттям результатів, проте і це не є вузьким місцем даного підходу, адже необхідна кількість потоків виконання може бути створена один раз перед першими розрахунками.

Реалізація алгоритму псевдокодом наведена у Додатку В. Схема алгоритму показана на рисунку 4.3, де:

- *struct Plane* — структура, що зберігає інформацію про площину, де a , b , c , d поля, що відповідають коефіцієнтам площини $ax+by+cz+d=0$;
- *threadChunkApproach* — процедура розбиття множини площин на підмножини (підсписки) і запуск процедури *getPolypointPlane* в окремому потоці виконання.
- *getPolypointPlane* — процедура перетворення площини методом політочкових перетворень. Для кожної вхідної площини та пари базисів (*origBasises*, *resBasises*) формується зважена система нормальних рівнянь $AX=B$ розміру 4×4 , де накопичення елементів A і B відбувається як суми по опорних точках x , y , z .

collectThreadResults — процедура, що об'єднує списки з перетвореними площинами в єдиний результуючий список.

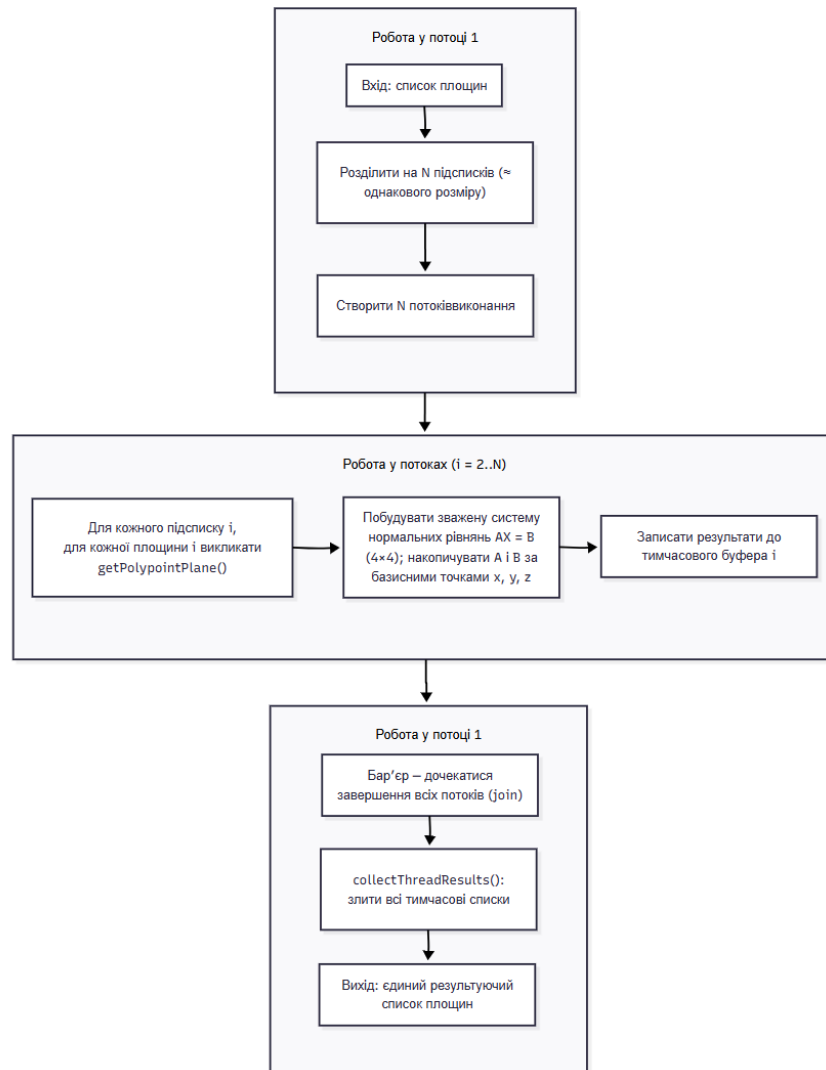


Рис. 4.3. Схема алгоритму запуску процедури *getPolypointPlane*

4.1.2 Експериментальна оцінка масштабованості та моделювання продуктивності паралельної реалізації методу політочкових перетворень

Для оцінювання масштабованості було використано триангульовану модель тора з приблизно 50 млн полігонів. Запуски виконувалися на конфігурації з максимально 24 потоками, причому для кожного числа потоків $P \in [1; 24]$

здійснювалося по 100 прогонів із усередненням часу виконання. Основні метрики: (а) середній час виконання; (б) прискорення $S(P)$ відносно послідовного виконання. Графік залежності часу від P наведено на рис. 4.4.

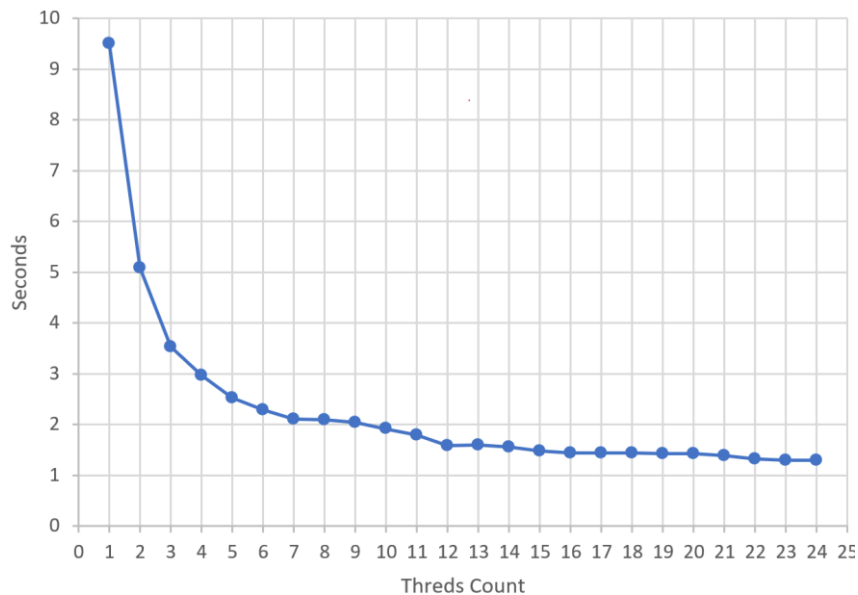


Рис. 4.4. Графік залежності усередненого із 100 прогонів часу виконання від кількості потоків

Отримані криві демонструють стрімке зменшення часу до ділянки насичення при 12-16 потоках із 24 доступних. Подальше збільшення P дає незначний приріст, а усереднений час наближається до горизонтальної асимптоти $\approx 1,016$ с.

Виникає питання, яка апроксимаційна модель описує дану залежність. Для того, щоб застосовувати закон Амдала [103], треба переконатися, що паралельна частина і послідовна добре розділені: кількість ядер впливає на паралельну частину лінійно, а на послідовну — не впливає взагалі. Для цього побудовано дві моделі, які протирічать цьому твердженню: раціональна (4.1) і гіперболічна (4.2). Було використано бібліотеку *SciPy* для *Python* і підібрано параметри a , b , c для загальних моделей (4.1) та (4.2).

$$f(x) = \frac{a}{x - b} + c, \quad (4.1)$$

$$f(x) = ax^{-b} + c, \quad (4.2)$$

де a , b , c — шукані параметри.

Отримані параметри підставлено та отримано моделі (4.3) і (4.4).

$$f(x) = \frac{7.655}{x - 0.098} + 1.016, \quad (4.3)$$

$$f(x) = 8.429x^{-1.07} + 1.063. \quad (4.4)$$

На рисунках 4.5-4.6 відображено по два набори даних. Перший набір *Data* - це набір точок, отриманий експериментальним шляхом, другий набір це крива, побудована за моделями (4.3) і (4.4).

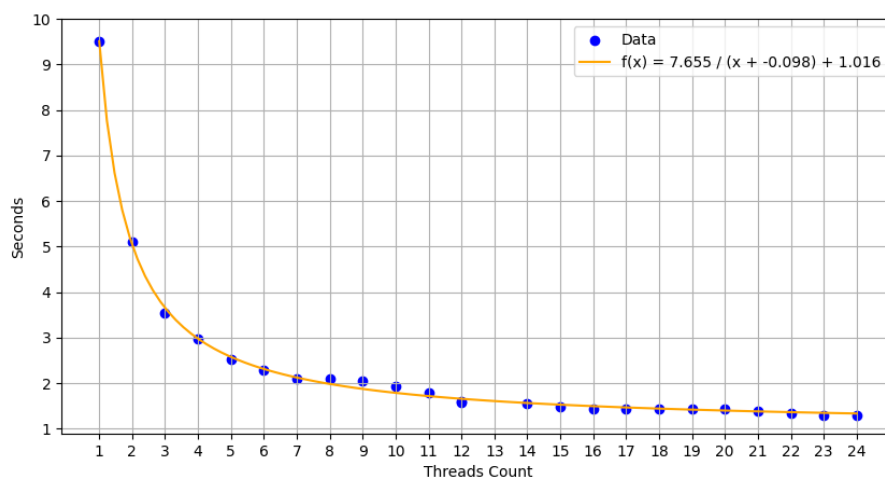


Рис. 4.5. Апроксимація використовуючи раціональний підхід

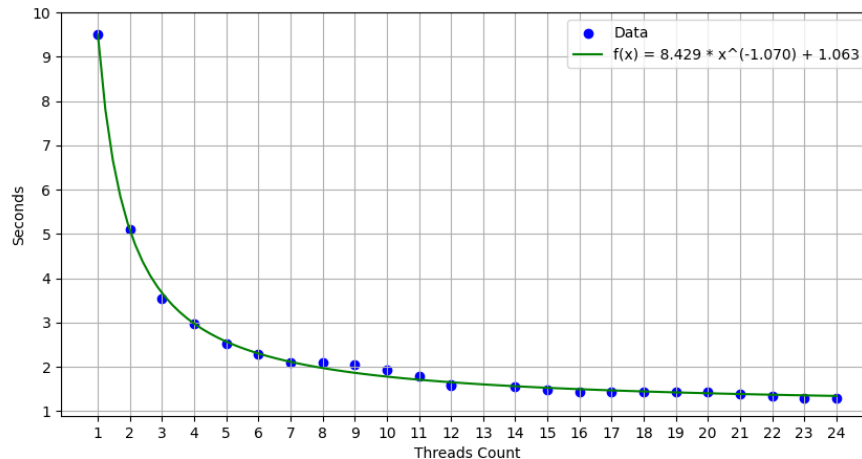


Рис. 4.6. Апроксимація використовуючи гіперболічний підхід

Отримані криві за формулами 4.3 і 4.4 візуально співпадають, про що свідчать графіки на рисунках 4.3-4.4. Для визначення кращої моделі апроксимації пораховано MAE за формулою:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_{i-факт.} - y_{i-прог.}|,$$

де $y_{i-факт.}$ — фактичне значення;

$y_{i-прог.}$ — прогнозоване значення.

Отримано MAE для раціональної моделі 0.049 та 0.05 для геперболічної.

Зважаючи на мале значення b у першій моделі 0.098 і на те, що у другій моделі b близьке до одиниці 1.07, час роботи алгоитму можна розділити на час роботи частини, що паралелиться ідеально $\frac{a}{x}$, і такої частини, що не паралелиться взагалі c .

Продовжуючи дослідження отриманих даних було проведено порівняння отриманих даних із законом Амдала. Щоб отримати графік закону Амдала, потрібно обчислити теоретичне прискорення за формулою:

$$S(P) = \frac{1}{(1-f) + \frac{f}{P}},$$

де $S(P)$ — прискорення на P потоках,

f — частка роботи програми, що виконується паралельно,

P — кількість потоків.

Зробивши усі необхідні перетворення та підібравши параметр $f = 0.9$ отримали класичний закон Амдала для багатопотокової реалізації політочкових перетворень (рис. 4.7).

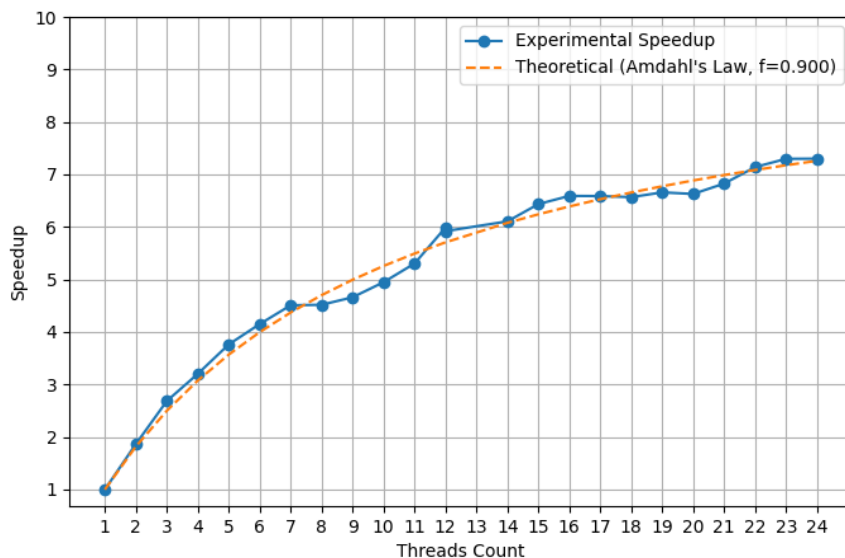


Рис. 4.7. Експериментальне і теоретичне прискорення для алгоритму політочкових перетворень

З рисунку 4.7 видно, експериментальні точки (синя лінія) досить добре збігаються з теоретичною кривою (помаранчева пунктирна лінія). Це ілюструє, що модель Амдала добре описує поведінку запропонованого паралельного алгоритму політочкових перетворень. Максимальне прискорення $\approx 7.5x$ при 24 потоках, проте

при такій кількості потоків плато не видно. Частка програми, що виконується паралельно $f = 0.9$, що означає, 10% коду виконується послідовно. Так як більшість роботи проводиться паралельно, то має сенс пришвидшувати саме паралельну частину, а саме те — як організоване паралельне обчислення. Потрібно зазначити, що при різних розмірах систем рівнянь, час обчислення трансформації однієї точки буде змінюватись.

4.2 Паралельна реалізація методу політочкових перетворень для розподілених систем на базі NUMA-архітектури

Попередній аналіз, представлений у підрозділі 4.1, продемонстрував, що базова багатопотокова реалізація методу політочкових перетворень на основі `threads` підпорядковується закону Амдала, де частка паралельної роботи становить приблизно 90%. Це свідчить про високий теоретичний потенціал масштабованості алгоритму. Однак емпіричні дані виявили досягнення плато продуктивності при використанні 12–16 потоків, що вказує на існування обмежувальних факторів, які не враховуються класичною моделлю Амдала. Цей підрозділ є логічним продовженням дослідження, що переходить від аналізу алгоритмічної масштабованості до інженерної оптимізації, спрямованої на подолання вузьких місць, зумовлених сучасною архітектурою багатопроцесорних систем, а саме — неоднорідним доступом до пам'яті. Для подолання цього обмеження варто розглянути застосування розподілених обчислювальних систем із неоднорідним доступом до пам'яті (від англ. Non-uniform memory access, NUMA).

Архітектура NUMA стала де-факто стандартом для сучасних для сучасних розподілених обчислювальних систем [104], витіснивши класичну модель симетричної багатопроцесорності SMP [105]. Основний принцип NUMA полягає в тому, що система складається з кількох вузлів, кожен з яких об'єднує один або декілька процесорних сокетів та власний банк оперативної пам'яті (рис. 4.8).

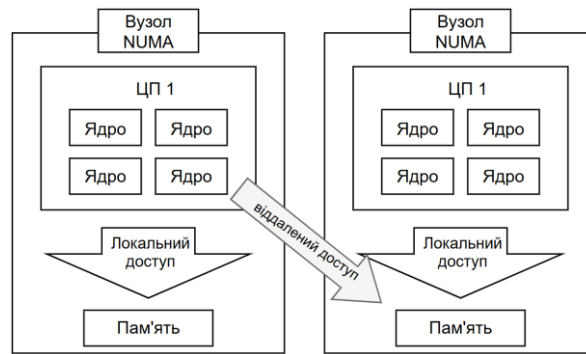


Рис. 4.8. Схема NUMA кластера з двома вузлами

Ключовою характеристикою такої архітектури є те, що доступ процесорного ядра до локальної пам'яті відбувається значно швидше, ніж до віддаленої пам'яті, що фізично підключена до іншого вузла [106].

Хоча закон Амдала успішно моделює обмеження, пов'язані з послідовною частиною коду, він ідеалізує підсистему пам'яті, розглядаючи її як єдиний ресурс з рівномірним часом доступу для всіх процесорних ядер. Ця ідеалізація не відповідає реаліям NUMA-систем [107, 108]. Розбіжність між плавною асимптотичною кривою, яку прогнозує закон Амдала, та різким виходом на плато, що спостерігається в експериментах з pthreads-реалізацією, є ключовою проблемою, що потребує пояснення. Це свідчить про перехід від домінування алгоритмічних обмежень до домінування архітектурних вузьких місць.

4.2.1 Аналіз вузьких місць базової реалізації на архітектурах з неоднорідним доступом до пам'яті (NUMA)

Для застосунків, які не враховують топологію NUMA, існує кілька фундаментальних факторів, що обмежують продуктивність та масштабованість:

- штраф за доступ до віддаленої пам'яті;
- конкуренція за шину міжз'єднань та пропускну здатність пам'яті;
- негативний вплив міграції потоків.

Штраф за доступ до віддаленої пам'яті — це найважливіша проблема NUMA [109]. Коли потік, що виконується на ядрі в одному NUMA-вузлі, звертається до даних, розташованих у пам'яті іншого вузла, запит має пройти через високошвидкісне міжз'єднання (наприклад, Intel UltraPath Interconnect або AMD Infinity Fabric). Цей шлях вносить значну додаткову затримку. Кількісні оцінки показують, що доступ до віддаленої пам'яті може бути в 1.5–3 рази повільнішим порівняно з доступом до локальної пам'яті [105]. Для обчислювально інтенсивних завдань, які постійно працюють з великими обсягами даних, такий штраф може нівелювати переваги від залучення додаткових процесорних ядер.

Міжсокетна шина, хоч і є високошвидкісною, має обмежену пропускну здатність. У ситуації, коли багато потоків, що працюють на ядрах різних NUMA-вузлів, одночасно звертаються до даних, сконцентрованих у пам'яті одного вузла, виникає ефект вузького горла. Це призводить до насичення пропускну здатності як міжз'єднання, так і контролера пам'яті цільового вузла, що різко обмежує подальшу масштабованість [110].

Стандартні планувальники операційних систем (наприклад, Linux Completely Fair Scheduler) за замовчуванням оптимізовані для забезпечення загального балансування навантаження та справедливості розподілу ресурсів [111, 112]. З цією метою вони можуть динамічно переміщувати потоки між різними процесорними ядрами. У NUMA-середовищі така міграція може бути вкрай шкідливою. Потік, який працював з даними у своїй локальній пам'яті, після міграції на ядро в іншому NUMA-вузлі раптово опиняється далеко від своїх даних. Усі подальші звернення до них стають віддаленими та повільними. Це руйнує локальність даних і призводить до значних, а головне — непередбачуваних та невідтворюваних падінь продуктивності [113].

На основі вищезазначених факторів формулюється наступна дослідницька гіпотеза: насичення продуктивності та подальше плато, що спостерігаються у pthreads-реалізації методу політочкових перетворень, зумовлені саме NUMA-

ефектами. Коли кількість потоків перевищує кількість фізичних ядер, доступних в одному NUMA-вузлі (для тестового стенду `s7a.metal-48x1` це 96 фізичних ядер на вузол), планувальник операційної системи починає розміщувати нові потоки на ядрах другого сокета. Оскільки базова реалізація є NUMA-неорієнтованою, вона не керує ані розміщенням потоків, ані розподілом даних. Це неминуче призводить до хаотичного доступу до пам'яті, де значна частина звернень стає віддаленою [114]. Зростаючі штрафи за латентність та конкуренція за міжз'єднання нівелюють обчислювальну потужність додаткових ядер, що й проявляється у вигляді плато на графіку продуктивності.

4.2.2 Розробка NUMA-орієнтованої стратегії паралелізації з використанням технології OpenMP

Для перевірки висунутої гіпотези та подолання виявлених вузьких місць була розроблена альтернативна реалізація паралельного алгоритму з використанням технології OpenMP, що надає інструменти для явного керування поведінкою програми в NUMA-середовищі.

Вибір OpenMP як інструменту для NUMA-оптимізації зумовлений його високорівневим, декларативним підходом до паралельного програмування. На відміну від `pthread`, де для керування прив'язкою потоків до конкретних ядер необхідно використовувати низькорівневі, платформно-залежні системні виклики (наприклад, `pthread_setaffinity_np` в Linux) [115], OpenMP надає стандартизований та портативний набір директив та змінних середовища [116, 117]. Це дозволяє програмісту описувати політику розміщення потоків, а не реалізовувати її механізм, що значно спрощує розробку, підвищує читабельність коду та забезпечує його переносимість між різними архітектурами та операційними системами.

Стратегія реалізації та керування прив'язкою потоків

Основою розробленої NUMA-орієнтованої стратегії є комбінація механізму прив'язки потоків OpenMP та принципу першого дотику для забезпечення локальності даних [116].

Для керування розміщенням потоків (від англ. Thread Affinity) були використані стандартні змінні середовища OpenMP 4.0 та новіших версій [117]:

OMP_PLACES=cores — ця змінна інструктує середовище виконання OpenMP розглядати кожне фізичне ядро процесора як окреме «місце» (place), до якого може бути прив'язаний потік.¹⁶ Це забезпечує максимальну гранулярність контролю, оскільки тестовий стенд не використовує Hyper-Threading (1 vCPU = 1 фізичне ядро).

OMP_PROC_BIND=close — ця змінна визначає політику прив'язки. Значення close змушує OpenMP розміщувати потоки паралельної області якомога ближче один до одного в списку доступних місць. На практиці це означає, що потоки будуть послідовно заповнювати ядра спочатку одного NUMA-вузла, і лише після вичерпання ядер у першому вузлі почнуть розміщуватися на ядрах наступного [116]. Така політика є оптимальною для завдань, де потоки активно обмінюються даними або використовують спільні ресурси (наприклад, кеш L3), оскільки вона максимізує ймовірність того, що взаємодіючі потоки будуть знаходитись в межах одного сокета, мінімізуючи затримки [117].

Продуктивність у NUMA-системах критично залежить не лише від розміщення потоків, а й від розташування даних, з якими вони працюють. Операційна система Linux за замовчуванням використовує політику першого дотику (від англ. First-Touch Policy): фізична сторінка пам'яті виділяється в тому NUMA-вузлі, на ядрі якого виконувався потік, що вперше звернувся до цієї сторінки. Для того, щоб використати цю особливість, етап підготовки даних (розбиття вхідного масиву площин на частини для кожного потоку) також був паралелізований. Кожен потік OpenMP ініціалізує та записує дані у свою власну частину вихідного буфера. Завдяки комбінації OMP_PROC_BIND=close та

паралельної ініціалізації, дані для потоків $0 \dots k-1$ (де k — кількість ядер на сокет) будуть гарантовано розміщені в пам'яті першого NUMA-вузла, а дані для потоків $k \dots N-1$ — у пам'яті другого, забезпечуючи максимальну локальність доступу до даних протягом усього обчислення [118].

Вибір політики `OMP_PROC_BIND=close` є свідомим інженерним компромісом. Вона максимізує локальність даних та швидкість синхронізації для потоків в межах одного сокета. Однак, при повному завантаженні одного вузла, всі його потоки починають конкурувати за пропускну здатність єдиного контролера пам'яті. Альтернативна політика *spread*, яка розподіляє потоки по черзі між сокетами [112], могла б забезпечити вищу сукупну пропускну здатність пам'яті за рахунок одночасного використання обох контролерів. Проте це відбулося б ціною значно вищих затримок для будь-якої синхронізації або обміну даними між потоками, що виконуються на різних сокетах. Враховуючи, що алгоритм політочкових перетворень містить етап бар'єрної синхронізації та фінального об'єднання результатів, було зроблено припущення, що мінімізація латентності доступу до даних та вартості синхронізації є більш критичним фактором для даної задачі, ніж максимізація сукупної пропускну здатності.

4.2.3 Експериментальне дослідження та порівняльний аналіз продуктивності pthreads та OpenMP для політочкових перетворень

Для кількісної оцінки ефективності запропонованої NUMA-орієнтованої стратегії було проведено серію експериментів, що порівнюють продуктивність оптимізованої реалізації на OpenMP з базовою реалізацією на pthreads.

Експерименти проводилися на високопродуктивній хмарній платформі, характеристики якої є критично важливими для коректної інтерпретації NUMA-ефектів у масштабі. Специфікації тестового стенду наведені в таблиці 4.1. Конфігурація зі 192 фізичними ядрами, розділеними на два NUMA-

вузли по 96 ядер, є ідеальною для тестування масивного паралелізму та дозволяє чітко відстежити ефекти при перетині межі між вузлами.

Таблиця 4.1. Характеристики тестового стенду

| Параметр | Значення |
|--------------------------------|--------------------------------------|
| Платформа | AWS VM Instance c7a.metal-48xl |
| Процесор (CPU) | AMD EPYC 9R14 (4th Gen) |
| Кількість vCPU (фізичних ядер) | 192 |
| Ядер на vCPU | 1 (SMT/Hyper-Threading вимкнено) |
| NUMA-вузлів | 2 |
| Ядер на NUMA-вузол | 96 |
| Оперативна пам'ять (RAM) | 384 GiB DDR5 |
| Операційна система | Ubuntu 20.04 LTS (ядро 5.4) |
| Компілятор | GCC 9.3.0 з прапором оптимізації -O3 |

Експериментальну модель побудовано на базі триангульованого кролика Stanford Bunny (рис. 3.2). Сумарна кількість трикутників (площин) дорівнює 80 мільйонів. Початковий та кінцевий базиси складаються з 50ти точок.

Представлення результатів

Вимірювання часу виконання проводилися для обох реалізацій при зміні кількості потоків від 1 до 192. Для кожного значення кількості потоків виконувалося по 100 запусків, результати яких усереднювалися для мінімізації випадкових флуктуацій. Через великий обсяг даних, у таблиці 4.2 наведено вибірку результатів, що ілюструє загальну динаміку. Повні графіки представлені на рисунках 4.9 і 4.10.

Таблиця 4.2. Порівняння часу виконання для базової (pthreads) та NUMA-орієнтованої (OpenMP) реалізацій (вибірка)

| Кількість Потоків | Open MP (сек) | Standard pthreads (сек) | Зменшення часу OpenMP | Прискорення OpenMP |
|----------------------|---------------|-------------------------|-----------------------|--------------------|
| 1 | 19.29 | 20.93 | 7.83 % | 1.08 |
| 10 | 9.65 | 11.83 | 18.41 % | 1.23 |
| 30 | 4.94 | 6.25 | 20.95 % | 1.27 |
| 50 | 4.19 | 5.05 | 16.97 % | 1.20 |
| 70 | 3.60 | 4.66 | 22.88 % | 1.30 |
| 90 | 3.42 | 4.02 | 14.88 % | 1.17 |
| 96 (межа NUMA-вузла) | 3.32 | 3.89 | 14.67 % | 1.17 |
| 100 | 3.26 | 3.82 | 14.55 % | 1.17 |
| 120 | 3.08 | 3.65 | 15.60 % | 1.18 |
| 150 | 2.88 | 3.34 | 13.78 % | 1.16 |
| 180 | 2.77 | 3.19 | 13.20 % | 1.15 |
| 192 | 2.75 | 3.17 | 13.43 % | 1.16 |

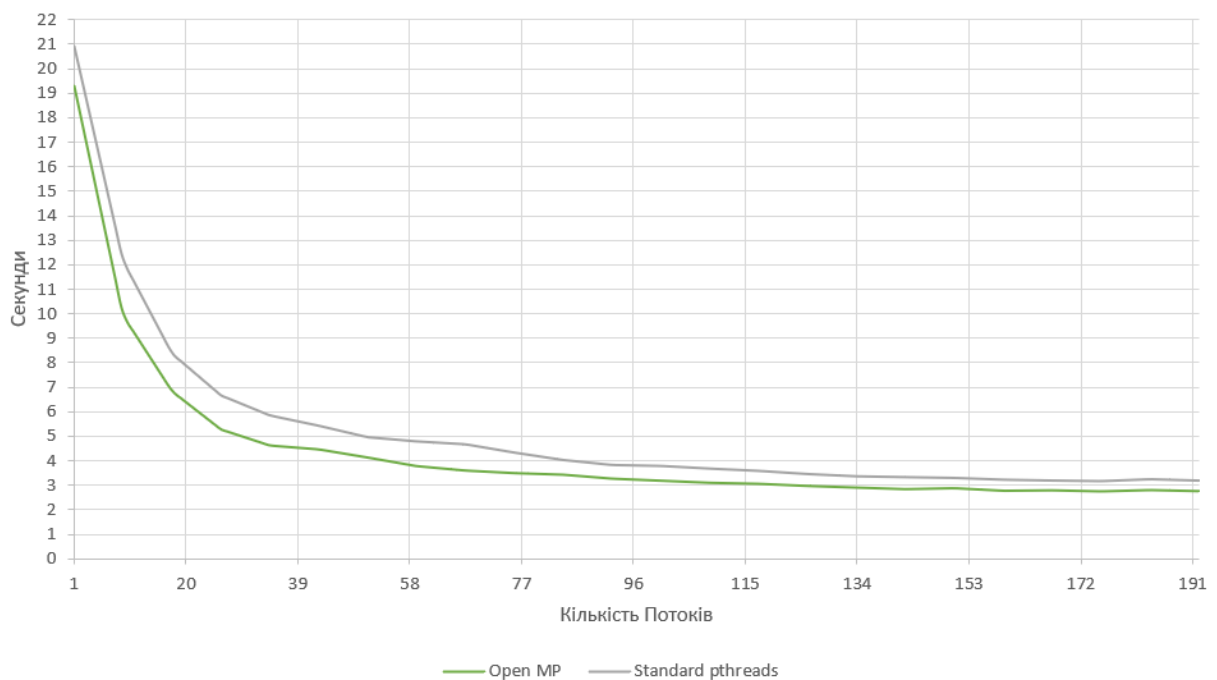


Рис. 4.9. Графік залежності часу виконання від кількості потоків (1-192)

На графіку наведено порівняльну характеристику часу виконання алгоритму для технологій OpenMP та Standard pthreads. Обидві криві спадають, але крива Open MP стабільно знаходиться нижче, демонструючи менший час виконання у всьому діапазоні до 192 потоку.

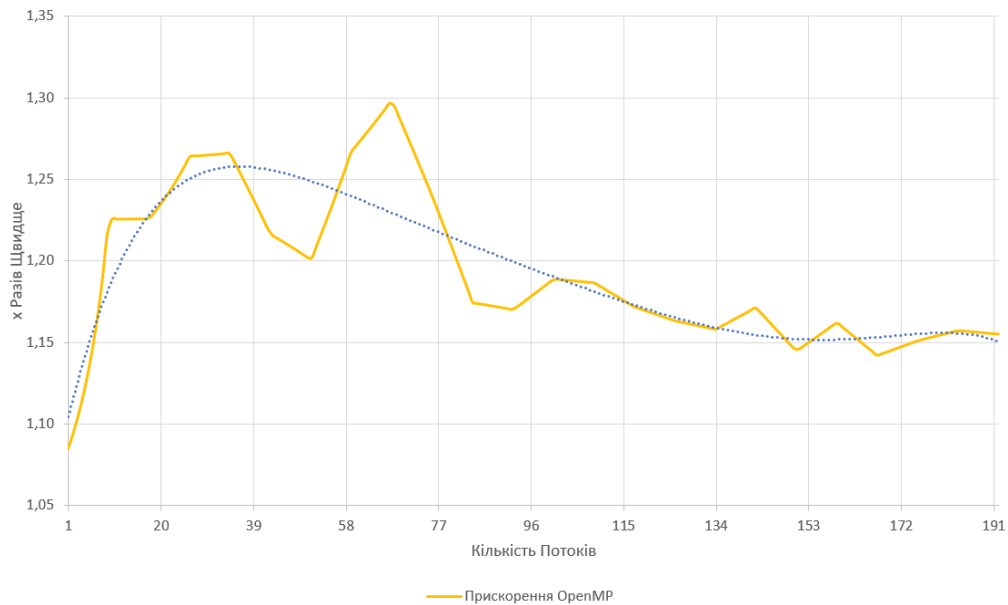


Рис. 4.10. Відносне прискорення NUMA-орієнтованої реалізації порівняно з базовою (1-192). Переривчаста лінія позначає лінію тренда

На рисунку представлено, що графік показує немонотонну криву прискорення OpenMP відносно pthreads. Крива демонструє складну динаміку з кількома піками та спадами, досягаючи максимального значення $\sim 1.30x$ в районі 70 потоків. Аналіз немонотонної кривої виявляє складну взаємодію програми з NUMA-архітектурою тестового стенду. Поведінку кривої можна розділити на дві основні ділянки, межею яких є заповнення першого NUMA-вузла (96 потоків). На першій ділянці (1–96 потоків) перевага OpenMP, що використовує політику `OMP_PROC_BIND=close`, зумовлена жорсткою прив'язкою потоків, що запобігає їх міграції та гарантованою локальністю даних завдяки принципу першого дотику. Флуктуації на цій ділянці та досягнення глобального піку $1.30x$ при 70 потоках

відображають оптимальний баланс між паралелізмом та початком насичення внутрішньовузлових ресурсів — пропускної здатності кешу L3 та контролера пам'яті.

При переході на другий NUMA-вузол (97+ потоків) pthreads — реалізація зазнає невдачі через хаотичний віддалений доступ до пам'яті. На противагу, OpenMP-стратегія коректно розміщує дані локально на другому вузлі, дозволяючи успішно масштабуватися, що підтверджується стабілізацією відносного прискорення на рівні 1.15x-1.18x. Незначне зниження ефективності порівняно з піком на одному вузлі є очікуваним і пояснюється накладними витратами на міжвузлову взаємодію, зокрема дорожчою бар'єрною синхронізацією, трафіком когерентності кешів та дисбалансом навантаження між сокетам.

Аналіз отриманих даних однозначно підтверджує висунуту гіпотезу. NUMA-орієнтована реалізація на OpenMP демонструє стабільно кращу продуктивність у всьому діапазоні тестованих кількостей потоків (від 1 до 192). Середнє зменшення часу виконання становить близько 15-20%, а максимальне *відносне* прискорення (OpenMP порівняно з pthreads) сягає 1.30x (тобто, 30%) при 70 потоках.

Важливо підкреслити, що цей приріст продуктивності досягнуто не за рахунок масштабування обчислювальних ресурсів у розподіленій системі, а виключно завдяки більш ефективному використанню вже наявних апаратних ресурсів. Це є прямим доказом того, що NUMA-ефекти були значним, хоча й прихованим, вузьким місцем у базовій pthreads-реалізації, особливо при масштабуванні до великої кількості ядер.

4.3 Прискорення методу політочкових перетворень з використанням GPU-архітектури CUDA

Попередні дослідження, викладені у розділах 4.1 та 4.2, продемонстрували ефективність паралелізації методу політочкових перетворень на багатоядерних

CPU-архітектурах. Базова реалізація на pthreads (підрозділ 4.1) показала значне прискорення, однак зіткнулася з насиченням продуктивності при досягненні 12–16 потоків на 24х ядерному CPU і 96 потоків на 192х ядерному CPU, що вказувало на алгоритмічні та системні обмеження. Подальша NUMA-орієнтована оптимізація з використанням OpenMP дозволила нівелювати частину цих вузьких місць, зокрема пов'язаних з неоднорідним доступом до пам'яті, що забезпечило додатковий приріст продуктивності у 15–30%.

Незважаючи на досягнуті успіхи, обидва CPU-орієнтовані підходи впираються у фундаментальні обмеження сучасної архітектури центральних процесорів: обмежену кількість обчислювальних ядер та високу вартість синхронізації і управління пам'яттю у NUMA-системах при спробах реалізації масового паралелізму. Досягнуте плато масштабованості свідчить, що подальше суттєве прискорення алгоритму потребує переходу до принципово іншої обчислювальної парадигми. Такою парадигмою є обчислення загального призначення на графічних процесорах, що надають апаратну базу для масового паралелізму на рівні тисяч спеціалізованих обчислювальних ядер.

4.3.1 Огляд архітектури та принципів організації паралельних обчислень у середовищі CUDA

Технологія CUDA (від англ. Compute Unified Device Architecture), вперше представлена компанією NVIDIA в 2007, стала революційним кроком у розвитку високопродуктивних обчислень, забезпечивши перехід від спеціалізованого використання графічних процесорів виключно для візуалізації до обчислень загального призначення (від англ. GPGPU — General-Purpose computing on Graphics Processing Units). CUDA програма може бути виконана лише на графічних процесорах від компанії NVIDIA [119].

Гетерогенна архітектура обчислювальних систем

Сучасні обчислювальні системи все частіше будуються за гетерогенним принципом, що передбачає спільне використання процесорів різних архітектур для вирішення однієї задачі. У контексті CUDA така система складається з двох головних компонентів:

Хост (Host) — базується на центральному процесорі (ЦП) та оперативній пам'яті (ОЗП). Він відповідає за керування логікою програми, введення і виведення даних, підготовку середовища та запуск обчислювальних ядер.

Пристрій (Device) — представлений графічним процесором (ГП) з власною високошвидкісною пам'яттю (ГОЗП). Він виступає як високопродуктивний апаратний прискорювач, що бере на себе виконання найбільш ресурсомістких ділянок коду.

Архітектурна відмінність між ЦП та ГП продиктована різними цілями проєктування. Центральні процесори оптимізовані для мінімізації затримки при виконанні послідовних інструкцій. Значну частину площі кристала ЦП займають блоки передбачення розгалуджень та великі багаторівневі кеші, що дозволяє швидко виконувати складну логіку. Натомість ГП спроектовані для максимізації пропускної здатності [119, 120]. Вони містять тисячі простіших АЛП, згрупованих у стрімінгові мультипроцесори (СМ). Така архітектура дозволяє приховувати затримки доступу до пам'яті шляхом миттєвого перемикання між тисячами активних потоків. Якщо один потік очікує на дані з пам'яті, планувальник ГП миттєво передає керування іншому потоку, який готовий до виконання, забезпечуючи повне завантаження обчислювальних потужностей. На рисунку 4.11 показано спрощену архітектуру гетерогенної системи.

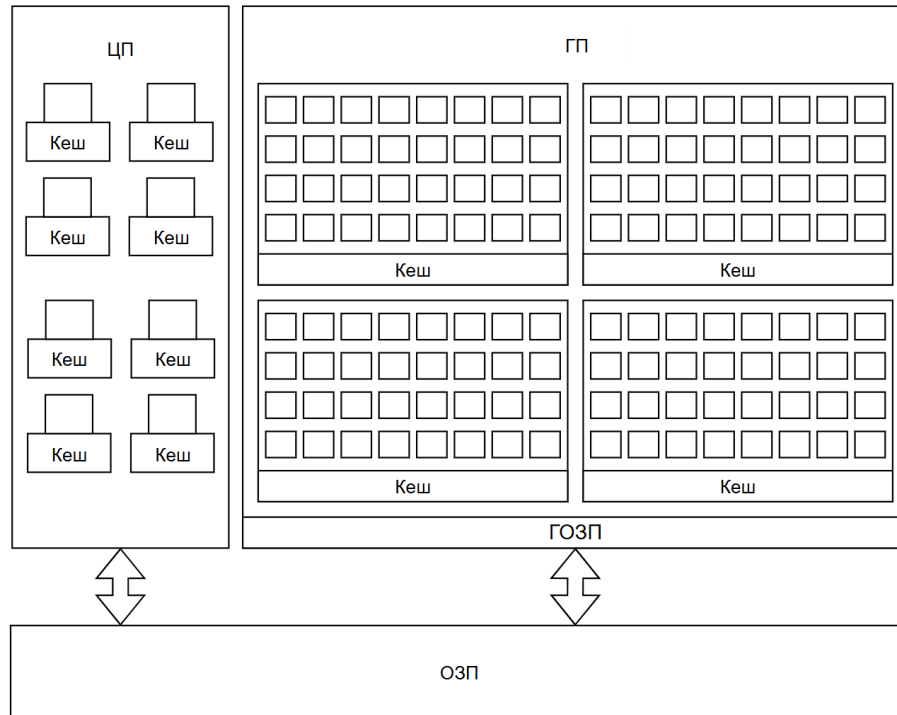


Рис. 4.11. Спрощену архітектуру гетерогенної системи з ЦП та ГП

ЦП складається з восьми ядер. Кожне ядро має свій кеш. Для простоти показано лише один рівень кешу. Тоді як ГП складається з декількох процесорів, які об'єднані в процесорні групи.

Модель виконання SIMT та ієрархія потоків

Програмна модель CUDA базується на концепції ядра (kernel) — функції, код якої пишеться мовою C/C++ і виконується на пристрої N разів одночасно різними потоками. Для ефективного масштабування на графічних процесорах різної потужності CUDA використовує трирівневу ієрархію потоків:

Потік (від англ. thread) — базова одиниця виконання. Кожен потік має власний ідентифікатор, набір регістрів та лічильник команд. Потоки в контексті ЦП і ГП реалізовані по-різному, тому важливо розрізняти два види потоків.

Блок потоків (від англ. *thread block*) — логічна група потоків, які можуть взаємодіяти між собою через швидку спільну пам'ять (від англ. *Shared Memory*) та бар'єрну синхронізацію.

Сітка (від англ. *grid*) — Сукупність усіх блоків, що виконують одне ядро. Блоки в сітці є незалежними один від одного, що дозволяє виконувати їх у довільному порядку на доступних стрімінгових мультипроцесорах.

На апаратному рівні архітектура NVIDIA реалізує модель SIMT (від англ. *Single Instruction, Multiple Threads*). Потоки об'єднуються у групи по 32 одиниці, які називаються варпами (від англ. *warps*). Усі потоки одного варпа виконують одну й ту ж інструкцію одночасно, але над різними даними. Варп можна уявити як один рух ткацького верстатата, який затягує наступну нитку. Це є ключовою відмінністю від векторних інструкцій SIMD (від англ. *Single Instruction, Multiple Data*) на ЦП, оскільки в SIMT кожен потік може мати власну логіку розгалуження.

Однак, варто зазначити явище дивергенції варпа (від англ. *warp divergence*). Якщо потоки всередині одного варпа переходять на різні гілки умовного оператора (наприклад, *if-then-else*), виконання цих гілок серіалізується: спочатку виконуються потоки, що пішли гілкою *then*, поки інші сплять, а потім навпаки. Це суттєво знижує продуктивність, тому при проєктуванні алгоритмів для ГП важливо мінімізувати розгалуження всередині варпів.

Модель пам'яті CUDA

Ефективність CUDA-додатків часто обмежується не швидкістю обчислень, а пропускнуою здатністю пам'яті. CUDA надає програмісту доступ до складної ієрархії пам'яті [121], кожна ланка якої має свої характеристики швидкості та обсягу.

Глобальна пам'ять (*Global Memory*) — найбільший за обсягом, але найповільніший тип пам'яті, доступний усім потокам та хосту. Критично важливим

тут є патерн доступу, що називається *coalescing* (об'єднання). Якщо потоки варпа звертаються до послідовних адрес пам'яті, апаратне забезпечення об'єднує ці запити в одну транзакцію, максимізуючи пропускну здатність. Хаотичний доступ призводить до значного падіння швидкодії.

Спільна пам'ять (*Shared Memory*) — це керована програмістом пам'ять, розташована безпосередньо на чипі мультипроцесора. Вона на порядки швидша за глобальну пам'ять і є основним засобом комунікації між потоками в межах одного блоку. Використання спільної пам'яті як кешу для часто використовуваних даних є стандартною технікою оптимізації.

Регістри (*Registers*) — це найшвидший тип пам'яті, індивідуальний для кожного потоку. Кількість регістрів обмежена; надмірне їх використання може зменшити кількість активних варпів на мультипроцесорі, що негативно вплине на можливість приховування затримок.

Константна та текстурна пам'ять — це Спеціалізовані види пам'яті, що використовують кешування для прискорення доступу лише для читання.

4.3.2 Постановка задачі та обґрунтування моделі паралелізму CUDA

Ключова властивість методу політочкових перетворень, ідентифікована в підрозділі 4.1.1, полягає у повній незалежності обчислень для кожної окремої площини. Результат перетворення однієї площини не залежить від результатів обробки будь-яких інших площин. В термінології паралельних обчислень, дана задача класифікується як природно паралельна (від англ. *embarrassingly parallel*).

Подібна властивість демонструє високу відповідність архітектурі сучасних ГП та програмній моделі CUDA.

Стратегія відображення обчислень. Перенесення обчислень на GPU вимагає чіткого відображення елементів задачі на апаратну ієрархію CUDA:

Обчислювальна сітка (Grid/Blocks) — це загальний обсяг роботи, що визначається кількістю площин *planeCount* (Додаток Г), декомпонується на 1D-сітку (Grid) обчислювальних блоків (Blocks). Кожен блок, у свою чергу, складається з фіксованої кількості потоків. Для забезпечення обробки кожної площини окремим потоком застосовується стандартне 1:1 відображення, де глобальний індекс потоку *idx* обчислюється як $blockIdx.x * blockDim.x + threadIdx.x$. Цей індекс безпосередньо використовується для доступу до масивів площин: *inPlanes[idx]* та *outPlanes[idx]* (Додаток Г).

Ефективність CUDA-ядра критично залежить від коректного розміщення даних у різних типах пам'яті ГП. Вхідні площини *inPlanes* та результуючі площини *outPlanes* — це великі масиви даних. Оскільки кожен потік *idx* читає *inPlanes[idx]* і пише в *outPlanes[idx]*, доступ до цих масивів у глобальній пам'яті є лінійним та послідовним. Це дозволяє апаратному забезпеченню ГП об'єднувати окремі запити від потоків одного варпу в одну широку транзакцію. Такий патерн доступу є найбільш ефективним для глобальної пам'яті.

Початковий та кінцевий базис (*origBasises*, *resBasises*, Додаток Г), — на відміну від площин, — це відносно малі масиви, які є однаковими для всіх потоків. У наївній реалізації кожен з тисяч потоків буде багаторазово читати ці масиви з повільної глобальної пам'яті. Це створює вузьке місце через високу латентність та некешовані, неузгоджені запити до пам'яті. Таким чином, ці масиви є головними кандидатами на оптимізацію шляхом кешування у швидших типах пам'яті, що буде розглянуто у 4.3.4.

Взаємодія між центральним процесором (хост) та графічним процесором (пристрій) інкапсульована у хост-функції-обгортці *gpu::deformPlanesPolypoint* (Додаток Г). Ця функція виконує повний життєвий цикл ГП-обчислення, а саме:

1. переміщення даних з хоста на пристрій;
2. конфігурація та запуск ядра;
3. копіювання результатів з пристрою на хост.

Під час переміщення даних з хоста на пристрій ініціюється асинхронне копіювання вхідних масивів площин (*inPlanes*) та обох наборів базисних точок (*origBasises*, *resBasises*) з оперативної пам'яті хоста (ОЗП) у глобальну пам'ять пристрою (ГОЗП).

Конфігурація та запуск ядра складаються з таких кроків як обчислення параметрів сітки (*gridSize*, *blockSize*) та виклик CUDA-ядра *deformPlanesPolypointKernel*<<<*gridSize*, *blockSize*>>>(...) (Додаток Г) для виконання безпосередньо на ГП.

Крок копіювання результатів з пристрою на хост вимагає примусової синхронізації (*cudaDeviceSynchronize*), що гарантує завершення ядром усіх обчислень перед початком копіювання даних.

Загальний час виконання ГП-реалізації складається не лише з часу роботи ядра, але й з накладних витрат на копіювання даних по шині PCIe, що є важливим фактором при аналізі загального прискорення.

4.3.3 Реалізація гетерогенного обчислювального процесу для методу політочкових перетворень

Для емпіричної перевірки ефективності ГП-обчислень було розроблено початкову версію обчислювального ядра *deformPlanesPolypointKernel* (Додаток Г), яка базується на прямому портуванні послідовної логіки, реалізованої для ЦП. Процес обробки розпочинається з ідентифікації потоком свого глобального індексу в межах 1D-сітки, що однозначно відповідає індексу оброблюваної площини у масиві вхідних даних, завантажених у ГОЗП. Алгоритм виконання кожного потоку складається з трьох послідовних етапів. На першому етапі відбувається ітеративний прохід по всіх парах базисних точок. У кожній ітерації потік зчитує координати точок з глобальної пам'яті, обчислює знакову відстань γ від поточної площини до точки початкового базису та акумулює внески у коефіцієнти системи

нормальних рівнянь (змінні $a1...d4$, $r1...r4$), використовуючи відповідні формули методу найменших квадратів.

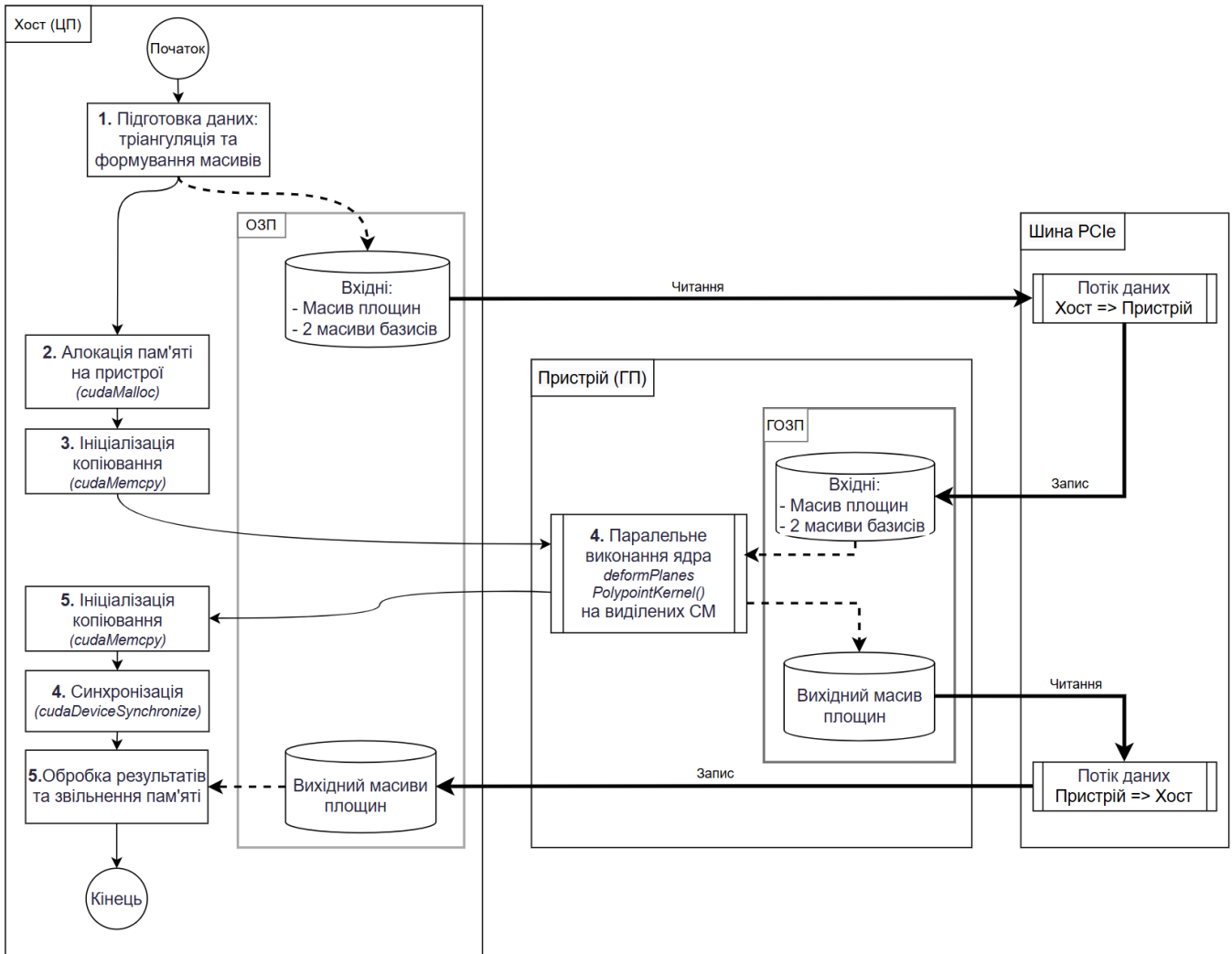


Рис. 4.12. Структурна схема організації гетерогенного обчислювального процесу та потоків даних між хостом і пристроєм для методу політочкових перетворень

На другому етапі, після завершення циклу накопичення, сформовані коефіцієнти організовуються у матрицю 4×4 та вектор вільного члена, при цьому до діагональних елементів додається регуляризаційний член для забезпечення чисельної стабільності (див. П.3.5.2). На фінальному етапі виконується розв'язання отриманої системи лінійних алгебричних рівнянь методом Гауса безпосередньо в

регістрах потоку, а знайдені нові коефіцієнти площини записуються у вихідний масив у ГОЗП.

На рисунку 4.12 представлена схема організації гетерогенних обчислень, що ілюструє послідовність етапів обробки даних та взаємодію між підсистемами хоста і пристрою. Процес ініціюється на хості підготовкою та лінеаризацією структур даних, після чого здійснюється виділення пам'яті та передача вхідних масивів у ГОЗП через шину PCIe. Центральним елементом є паралельне виконання ядра *deformPlanesPolypointKernel* на графічному процесорі, де кожен потік незалежно обробляє окрему площину, використовуючи дані з ГОЗП. Завершальна фаза алгоритму керується хостом, який ініціює зворотне копіювання перетворених площин в ОЗП та виконує бар'єрну синхронізацію для гарантування цілісності результатів перед звільненням ресурсів.

Детальний аналіз профілю виконання даної реалізації дозволив ідентифікувати низку критичних вузьких місць, що суттєво обмежують продуктивність. По-перше, значні накладні витрати створює блок арифметичних операцій, зокрема багаторазове використання операції ділення з подвійною точністю (double) при обчисленні $\frac{1}{\gamma}$ та $\frac{1}{\gamma^2}$. Стандартна реалізація ділення для 64-бітних чисел з плаваючою комою на архітектурі ГП характеризується високою латентністю та низькою пропускнуою здатністю інструкцій. По-друге, спостерігається неефективне використання ресурсів СМ через високий тиск на регістрову пам'ять. Для зберігання проміжних сум матриці нормальних рівнянь використовується велика кількість окремих скалярних змінних (*a1*, *b1*, тощо), що змушує компілятор виділяти значний обсяг регістрів на кожен потік. Це, у свою чергу, обмежує максимальну кількість активних варпів, які можуть одночасно перебувати на СМ, знижуючи здатність ГП приховувати латентність доступу до пам'яті. По-третє, критичним фактором є патерн роботи з пам'яттю: у кожній ітерації циклу всі потоки одночасно звертаються до масивів *origBasises* та

resBasises, що знаходяться у повільній глобальній пам'яті ГОЗП. Оскільки ці дані не кешуються явним чином, виникає значний затор через надлишкові зчитування, що призводить до простою обчислювальних блоків.

В результаті експериментального запуску даної базової версії на тестовому стенді (детальна конфігурація якого та параметри наборів даних будуть наведені у наступних розділах і залишатимуться незмінними для забезпечення коректності порівнянь) було зафіксовано час виконання 2.13757 с. Цей показник приймається за еталонне значення для оцінки ефективності подальших оптимізацій.

4.4 Дослідження методів оптимізації та підвищення ефективності гетерогенного обчислювального процесу для методу політочкових перетворень

Емпіричні результати базової ГП-реалізації (2.14с) продемонстрували значний потенціал для покращення. Детальний аналіз профілю виконання за допомогою інструментарію NVIDIA Nsight Compute дозволив сформулювати робочу гіпотезу: основним стримуючим фактором продуктивності є висока латентність доступу до глобальної пам'яті (від англ. Memory Latency Bound) у поєднанні з недостатнім паралелізмом інструкцій та низькою завантаженістю мультипроцесорів (від англ. Low Occupancy) через надмірне використання регістрів. Для перевірки цієї гіпотези та досягнення максимальної ефективності було розроблено та реалізовано стратегію поетапної оптимізації (реалізація наведена у Додатку Д).

4.4.1 Оптимізація на рівні інструкцій та аналіз чисельної стабільності

Першим кроком оптимізації стала заміна стандартних арифметичних операцій з плаваючою комою подвійної точності на спеціалізовані внутрішні

інструкції графічного процесора, такі як використання множення-додавання з одноразовим округленням FMA і `__drcp_rn`. У базовій реалізації (Додатку Г) обчислення зворотної відстані $\frac{1}{\gamma}$ та $\frac{1}{\gamma^2}$ виконувалося за допомогою стандартного оператора ділення `/`. Ця операція на архітектурі CUDA є однією з найбільш витратних. У оптимізованій версії було застосовано інструкцію `__drcp_rn(γ)` (від англ. device reciprocal round-to-nearest), яка виконує апаратне обчислення оберненої величини. Це дозволяє уникнути генерації складної послідовності команд ділення, забезпечуючи значно меншу кількість тактових циклів на виконання.

Попри те, що інструкція `__drcp_rn` відповідає стандарту IEEE 754 [122], перехід на іншу мікроархітектуру (з x86 на PTX) та можливі відмінності у послідовності операцій як, наприклад, використання FMA на ГП, теоретично можуть призводити до появи розбіжностей у молодших бітах мантиси. Для заміру похибки із використанням `__drcp_rn` було проведено порівняльний аналіз результуючої геометрії. Розрахунок середньоквадратичної похибки (MSE) відхилення координат вершин порівняно з еталонною ЦП-реалізацією показав повну відсутність значущих розбіжностей (похибка дорівнює 0 або знаходиться на межі машинної точності). Це підтверджує, що для даного класу задач оптимізація арифметики не впливає на візуальну якість та геометричну точність моделі.

Заміна операції ділення дозволила зменшити час виконання ядра з 1.90 с до 1.45 с, що становить приріст продуктивності у $\sim 30\%$. Це підтверджує, що ядро було частково обмежене латентністю арифметичного конвеєра.

Додатково до експерименту із заміною ділення на інструкцію `__drcp_rn`, було перевірено вплив агресивніших оптимізацій апаратної арифметики NVIDIA — режиму Fast Math (компіляція з дозволом швидких, але потенційно менш точних перетворень та апроксимацій) — на числову стабільність результату.

Для цього виконувалось попарне порівняння вихідних коефіцієнтів прямиї (A,B,C,D), обчислених GPU-ядром у двох конфігураціях: *Fast Math ON* та *Fast*

Math OFF, на геометрії тора, що містить 50 мільйонів трикутників; для кожного коефіцієнта оцінювалась абсолютна та відносна похибка між двома режимами.

Отримані результати показали, що розбіжності носять виключно характер числових відмінностей у межах машинної точності: медіанні абсолютні похибки становлять 4.55×10^{-13} (A), 2.27×10^{-13} (B), 5.69×10^{-14} (C) та 1.42×10^{-14} (D), а середні відносні похибки — лише $3.4 \times 10^{-5}\%$ (A), $6.2 \times 10^{-5}\%$ (B), $1.9 \times 10^{-5}\%$ (C) та $7 \times 10^{-6}\%$ (D). Максимальні відхилення також залишаються малими: найбільша абсолютна становить 2.11×10^{-10} , тоді як найбільша відносна похибка не перевищує $3.865 \times 10^{-3}\%$ (рис. 4.13).

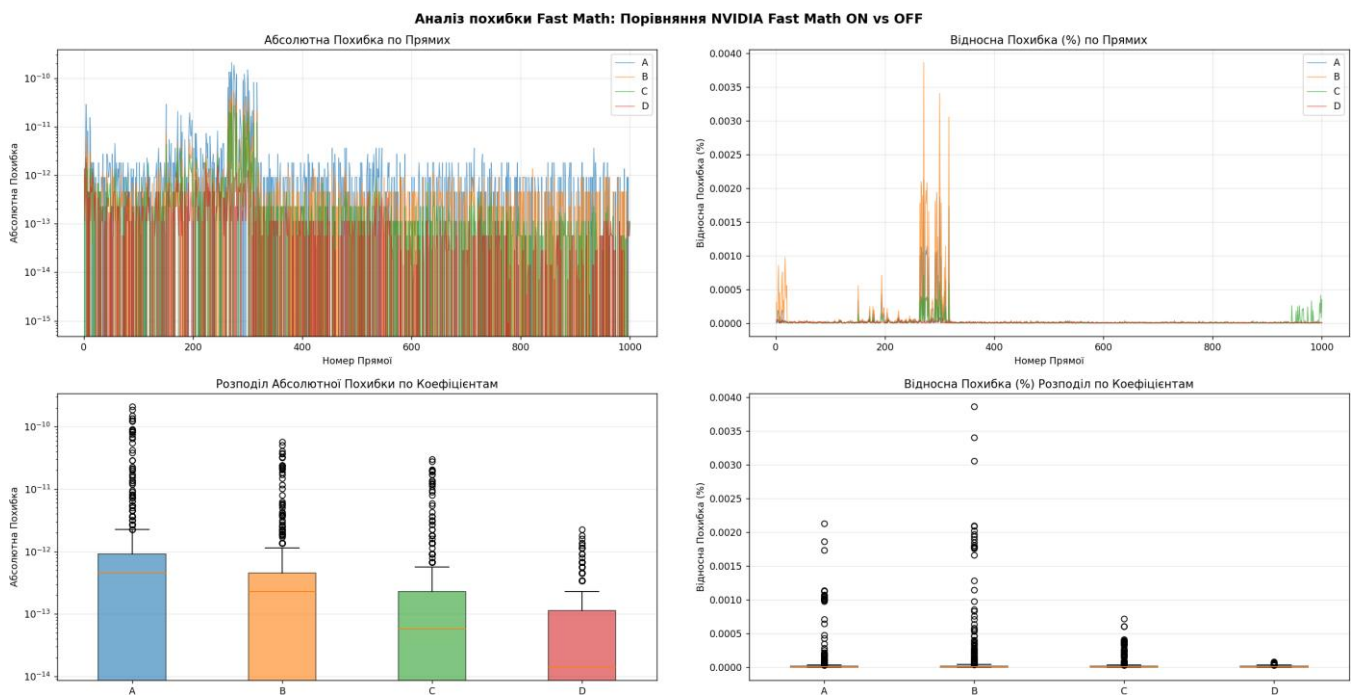


Рис. 4.13. Порівняння геометрії, отриманої з використанням технології *Fast Math*, із геометрією без *Fast Math*

Характерно, що пікові відхилення локалізуються в поодиноких прямим (зокрема, для прямої 271), що узгоджується з очікуваною чутливістю до перестановок операцій та використання FMA/апроксимацій у *Fast Math* для окремих комбінацій вхідних значень. Таким чином, як і у випадку з `__drcp_rn`,

ввімкнення Fast Math не призводить до значущої деградації точності для цього класу задач і не має практичного впливу на геометричну коректність результату, оскільки зафіксовані розбіжності є суттєво меншими за пороги, здатні проявитися на рівні візуальної якості або метричних характеристик моделі.

4.4.2 Підвищення рівня завантаженості стрімінгових мультипроцесорів

Аналіз метрик використання ресурсів виявив, що наївна реалізація страждає від надмірного тиску на регістрову пам'ять. Велика кількість скалярних змінних ($a1...d4$, $r1...r4$) змушувала компілятор резервувати значний обсяг регістрів для кожного потоку. Оскільки обсяг регістрового файлу на СМ є фіксованим, це призводило до обмеження максимальної кількості активних варпів, які можуть одночасно перебувати на мультипроцесорі. Недостатня завантаженість унеможливлювала ефективне приховування латентності пам'яті: коли всі активні варпи блокувалися в очікуванні даних, обчислювальні блоки простоювали.

Для вирішення цієї проблеми було реструктуризовано код. Змінні-акумулятори було замінено на компактні масиви $acc[14]$ та $rhs[4]$ (Додатку Е), що дозволило компілятору *nvcc* ефективніше розподіляти життєвий цикл змінних та повторно використовувати регістри. Це призвело до незначного прямого прискорення (з 1.45 с до 1.40 с), але створило передумови для наступного кроку.

Завдяки вивільненню регістрів стало можливим збільшити розмірність блоку з 256 до 512 потоків. Збільшення кількості потоків у блоці при зменшеному споживанні регістрів на потік дозволило планувальнику варпів підтримувати більшу чергу готових до виконання варпів. Це значно покращило механізм приховування латентності (від англ. *latency hiding*): поки одна група варпів очікує завершення транзакцій з пам'яттю, СМ миттєво перемикається на виконання інструкцій іншої групи.

4.4.3 Оптимізація підсистеми пам'яті

Незважаючи на попередні оптимізації, профілювання показало, що шина глобальної пам'яті залишається перевантаженою. Причиною став патерн доступу до масивів базисних точок *origBasises* та *resBasises*. Ці масиви є спільними для всіх потоків і мають високий коефіцієнт повторного використання: кожен потік зчитує одні й ті самі дані в циклі. Читання їх з глобальної пам'яті (ГОЗП) призводило до генерування надлишкового трафіку по шині даних.

Було реалізовано механізм програмно-керованого кешування з використанням спільної пам'яті (від англ. *shared memory*), яка є на порядки швидшою за глобальну. Розроблено схему кооперативного завантаження (Додаток Е). На початку роботи ядра кожен потік блоку завантажує свою частину масивів базисних точок з ГОЗП у спільної пам'ять. У основному обчислювальному циклі кожного блока всі звернення відбуваються виключно до спільної пам'яті блока.

Ця оптимізація дозволила усунути надмірне навантаження на шину пам'яті (від англ. *Memory Wall Effect*) для базисних точок. Час виконання скоротився з ~1.40 с до ~1.30 с. Важливішим результатом є зміна характеру навантаження: задача переорієнтувалася з такої, що обмежена пропускнуою здатністю пам'яті, у таку, що обмежена швидкістю обчислень, що свідчить про повну утилізацію обчислювального потенціалу ГП для даного алгоритму. Результат моделювання деформації скручування моделі стендфорського кролика показано на рисунку 4.15; лопати вітряка показано на рисунку 4.16.

Аналіз профілю виконання оптимізованого ядра (рис. 4.14) демонструє, що метрика «Пропускна здатність СМ» досягає 90.7%, в той час як завантаження підсистеми пам'яті становить лише 6.68%.

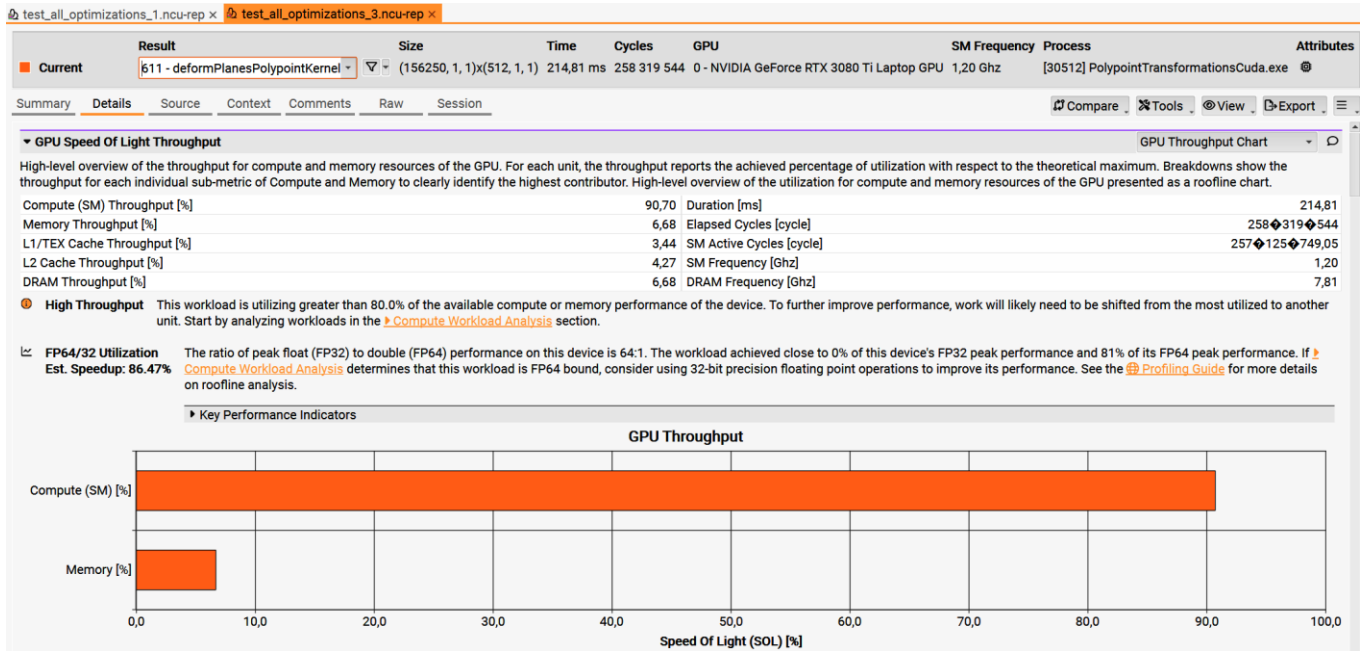


Рис. 4.14. Аналіз профілю виконання оптимізованого CUDA ядра *deformPlanesPolypointKernel* з програмного комплексу Nvidia Nsight Compute

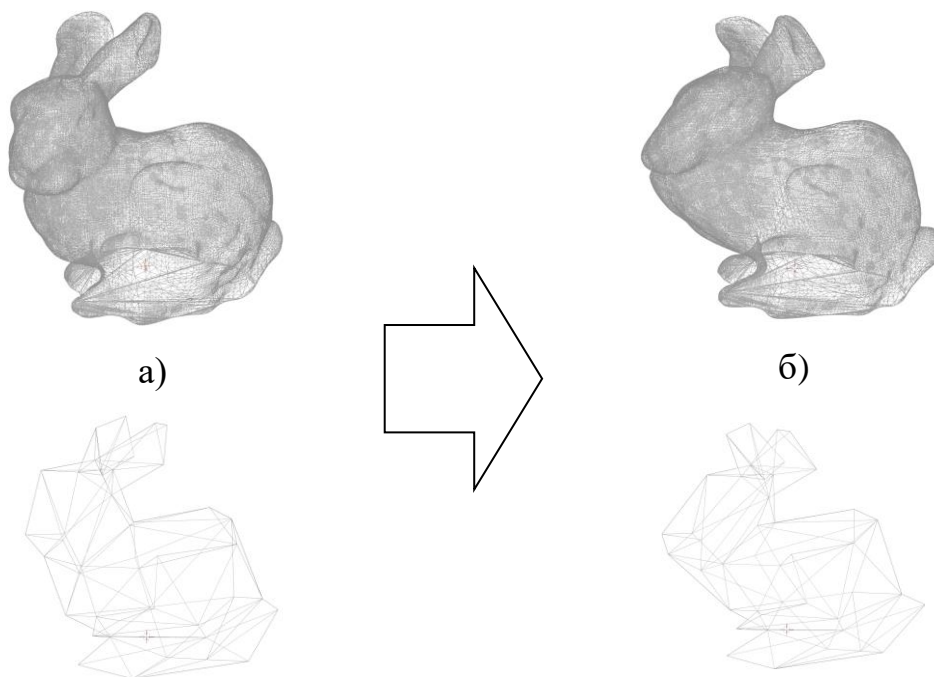


Рис. 4.15. а) багатоплігональна модель стендфорського кролика до деформації з вхідним базисом; б) модель після деформації та вихідним базисом

Це свідчить про повну переорієнтацію профілю задачі з такої, що обмежена пропускнуою здатністю пам'яті (від англ. Memory-Bound) у базовій версії, в таку, що обмежена швидкістю обчислень (від англ. Compute-Bound). Високий показник використання блоків FP64 (81%) вказує на те, що реалізація ефективно утилізує ресурси арифметичних співпроцесорів подвійної точності, досягаючи межі апаратної продуктивності пристрою.

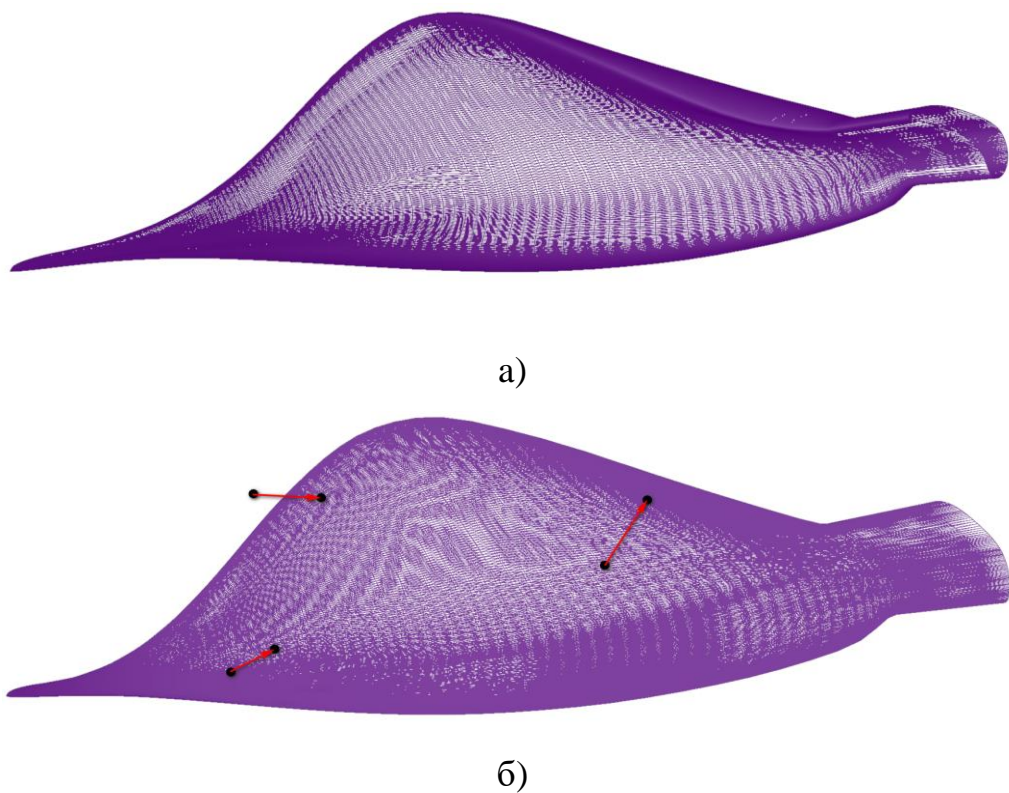


Рис. 4.16. а) лопать вітряка до застосування деформації; б) лопать вітряка після застосування деформації

Аналіз мікроархітектурних показників показує, що хоча ми досягли високого завантаження, вузьким місцем стала пропускна здатність внутрішніх конвеєрів кеш-пам'яті (L1/Tex Throttle). Це свідчить про те, що ядро

deformPlanesPolypointKernel настільки швидко обробляє дані, що фізичні лінії передачі даних всередині чіпа працюють на межі своїх можливостей.

4.4.4 Експериментальна оцінка продуктивності та порівняльний аналіз

Для верифікації теоретичних розрахунків та оцінки ефективності запропонованих методів оптимізації було проведено серію натурних експериментів. Метою даного етапу є кількісне порівняння продуктивності розробленого гетерогенного алгоритму з попередніми ЦП-орієнтованими реалізаціями, а також аналіз впливу кожного етапу оптимізації на загальний час виконання. Детальні технічні характеристики ГП, що використовувався як основний обчислювальний вузол, наведено в таблиці 4.3.

Таблиця 4.3. Апаратна специфікація графічного прискорювача

| Параметр | Значення |
|---|---------------------------------------|
| Модель ГП | NVIDIA GeForce RTX 3080 Ti Laptop GPU |
| Архітектура | Ampere (GA103) |
| Кількість шейдерних блоків (CUDA Cores) | 7424 |
| Кількість RT ядер | 58 |
| Обсяг / Тип відеопам'яті (ГОЗП) | 16 ГБ / GDDR6 |
| Шина пам'яті | 256 біт |
| Пропускна здатність пам'яті | ~512 ГБ/с |
| Теоретична продуктивність (FP32) | ~18.7 TFLOPS |
| Теоретична продуктивність (FP64) | ~0.292 GFLOPS (співвідношення 1:64) |

Програмна складова тестового стенду представлена у таблиці 4.4.

Таблиця 4.4. Програмна складова тестового стенду

| Параметр | Значення |
|--------------------|---|
| Операційна система | Windows 11 25H2 |
| MSVC toolset | 14.39 (v143) |
| Драйвер NVIDIA | 581.57 |
| CUDA Toolkit | 12.8 |
| Прапори компіляції | -gencode=arch=compute_86, code=«sm_86,compute_86» -use_fast_math -O2 |

Для забезпечення коректності порівняльного аналізу з результатами, отриманими у підрозділах 4.1.2 та 4.2.3, використовувалися ідентичні синтетичні та реальні набори геометричних даних:

- високополігональна триангульована модель тора, що містить 50 мільйонів трикутників.
- масштабована модель Stanford Bunny, деталізована до 80 мільйонів полігонів.

В обох випадках розмірність базисних множин (початкового та кінцевого) становила 10 точок.

У ході експерименту фіксувався час виконання ядра *deformPlanesPolypointKernel* для набору даних Stanford Bunny на кожному етапі оптимізації, описаному в підрозділах 4.4.1 – 4.4.3. Результати вимірювань (усереднені за 100 прогонами) наведено в таблиці 4.5.

Таблиця 4.5. Динаміка продуктивності CUDA-ядра в процесі оптимізації

| Етап оптимізації | Опис заходу | Час виконання (с) | Прискорення (до етапу) | Сумарне прискорення |
|------------------------------------|-----------------------------------|-------------------|------------------------|---------------------|
| Baseline (Базова версія) | Наївна реалізація | 2.14 | — | — |
| Етап 1 (Інструкції) | Використання __drcp_rn / fma | 1.45 | +32.2% | 1.47x |
| Етап 2 (Регістри) | Зменшення кількості скалярів | 1.40 | +3.5% | 1.53x |
| Етап 3 (Пам'ять) | Кешування у __shared__ пам'яті | 1.30 | +7.1% | 1.65x |

Аналіз даних таблиці 4.4 дозволяє зробити наступні висновки:

1. Найсуттєвіший приріст продуктивності (понад 30%) було отримано на першому етапі за рахунок заміни високолатентних операцій ділення подвійної точності на апаратні інструкції. Це підтверджує, що базова версія була значною мірою обмежена швидкістю арифметичного конвеєра.
2. Оптимізація використання регістрів (Етап 2) дала незначний прямий приріст, проте вона стала критично важливою передумовою для підвищення завантаженості, що дозволило ефективніше приховувати латентність на наступних кроках.
3. Впровадження кешування базисних точок у спільну пам'ять (Етап 3) дозволило досягти фінального часу виконання 1.30 с.

Порівняння фінального результату (1.30 с) з найкращою реалізацією на ЦП (OpenMP, 192 потоки, час виконання ~2.75 с, див. табл. 4.2) демонструє, що використання одного ГП забезпечує прискорення у 2.11 раза порівняно з кластером із 192 фізичних ядер CPU. Якщо ж порівнювати з однопотоковою версією на ЦП (~19.29 с), то сумарне прискорення сягає ~14.8 раза.

Узагальнюючи отримані результати, можна констатувати, що архітектура ГП є більш ефективною платформою для реалізації методу політочкових перетворень.

Перевага у співвідношенні ціна/продуктивність та можливість масового паралелізму робить використання ГП економічно вигіднішою альтернативою масштабуванню ЦП-кластерів.

Висновки до четвертого розділу

У четвертому розділі обґрунтовано та реалізовано інженерне масштабування методу політочкових перетворень у гетерогенному середовищі, що удосконалило спосіб обрахунку політочкових перетворень за рахунок розпаралелювання обчислень, що призвело до значного скорочення часу підрахунків при збереженні цілісності об'єкта. Як базове подання прийнято «вершина як перетин площин дотичних трикутників», оскільки воно забезпечує оптимальне співвідношення точності та обчислювальної ефективності.

Для ЦП запропоновано багатопотокову архітектуру, що спирається на незалежність обробки площин і добре описується законом Амдала (частка паралельної роботи близько 0.9). Показано, що раннє насичення продуктивності у розподілених систем зумовлене NUMA-ефектами; NUMA-орієнтована реалізація на OpenMP із керуванням прив'язкою потоків і ідіоми першого дотика забезпечила стабільно кращі результати порівняно з pthreads (орієнтовно 15–20% часу, до ~30% у найкращому випадку).

Подальше прискорення досягнуто перенесенням обчислень на ГП (CUDA) з відображенням «одна площа — один потік» та поетапною оптимізацією ядра за результатами профілювання. Усунення дорогих FP64-ділень, зменшення регістрового тиску та кешування базисів у спільну пам'ять знизили час ядра з 2.14 с до 1.30 с і перевели реалізацію обмежену пропускнуою здатністю пам'яті в таку, що обмежена швидкістю обчислень, забезпечивши перевагу над найкращим ЦП-варіантом та підтвердивши доцільність гетерогенного підходу для прискорення методу політочкових перетворень.

ВИСНОВКИ

У дисертаційній роботі вирішено науково-прикладну проблему геометричного моделювання деформацій геометричних об'єктів з використанням точкового базису довільної конфігурації на основі паралельних обчислень. *Значення для науки* полягає в подальшому розвитку методів моделювання деформацій геометричних об'єктів методом політочкових перетворень. *Значення для практики* полягає в створенні нових способів задання геометрії для моделювання деформації, які зберігають топологію сітки та можуть працювати у просторі довільної розмірності, а також ефективно паралелізуються на сучасних обчислювальних архітектурах. Їх програмна реалізація дозволяють суттєво прискорити обчислення деформацій складних моделей, що розширює можливості їх використання в системах комп'ютерної графіки та CAD/CAE для інтерактивного відтворення деформацій.

При виконанні поставлених завдань отримано такі основні теоретичні та практичні результати:

1. Здійснено аналіз сучасних методів моделювання деформацій геометричних об'єктів (нейронно-імпліцитних, фізично орієнтованих і геометричних підходів). Виявлено недоліки існуючих технологій деформації, зокрема обмеження інтерактивної швидкодії фізичних моделей та залежність від регулярної сітки в геометричних методах (політканинних перетвореннях). Обґрунтовано перспективність політочкових перетворень як перспективних для подальшого розвитку завдяки поєднанню високої швидкодії та достатньої точності моделювання.

2. Запропоновано модель формування двовимірних об'єктів на основі політочкових перетворень. Розроблено формалізований полігональний спосіб

задання вихідної геометрії об'єкта перед застосуванням деформації, що забезпечує збереження топології та стабільність обчислень при подальших трансформаціях.

3. Досліджено вплив кута між твірними прямими на результати політочкового перетворення об'єкта. Встановлено, що хоча між кутом утворюючих прямих у кожній точці та характером деформації існує залежність, вона має майже лінійний характер без осциляцій, особливих точок чи вертикальних асимптот.

5. Розроблено оптимізований алгоритм вибору параметра α для гаусового інтерполяційного згладжування. Автоматизовано процес налаштування параметра: алгоритм дозволяє визначити оптимальне значення α , яке балансує між згладженістю форми та збереженням геометричних деталей деформованого об'єкта, що покращує якість моделювання без ручного підбору параметрів.

6. Удосконалено застосування політочкових перетворень до тривимірних об'єктів. Запропоновано новий спосіб задання геометрії 3D-об'єкта, представленого трикутнковою сіткою, шляхом визначення положення вершини як точки перетину площин, побудованих за дотичними до суміжних трикутників. Цей підхід забезпечує цілісність об'єкта після політочкової деформації, дозволяє уникнути розривів у сітці та гарантує однозначне відновлення геометрії. На відміну від відомих способів на основі нормалей або ортогональних площин, запропонований спосіб демонструє кращу стійкість до числових похибок і точніше зберігає форму об'єкта після перетворення.

7. Проведено порівняльний аналіз запропонованих способів подання геометрії. Показано, що спосіб представлення вершини як точки перетину площин дотичних трикутників забезпечує найкраще співвідношення точності реконструкції поверхні та обчислювальної ефективності. Встановлено, що цей підхід мінімізує похибку відновлення форми за найменших обчислювальних витрат, тому саме він використаний як базовий у подальшій реалізації.

8. Проаналізовано обчислювальну складність трьох способів задання геометрії. Отримано аналітичні оцінки трудомісткості кожного з способів

відновлення геометрії для політочкових перетворень. Визначено, що всі запропоновані алгоритми мають поліноміальну (не експоненційну) складність, що підтверджує їх придатність для практичного застосування на моделях високої розмірності, а отримані оцінки вказують напрямки можливих оптимізацій.

9. Забезпечено чисельну стабільність обчислень при деформаціях. Виявлено проблему вироджених випадків (коли базисні площини близькі до паралельних), що призводить до нестійкого визначення вузлових точок. Для її розв'язання запропоновано метод стабілізації на основі псевдоінверсної матриці Мура-Пенроуза з регуляризацією Тихонова, який гарантує однозначне знаходження позиції вершини навіть у випадках відсутності єдиної точки перетину площин. Експериментально визначено оптимальний діапазон значення параметра регуляризації $\lambda=10^{-6}$, за якого досягається стійкий результат без суттєвої втрати точності моделювання.

10. Розроблено та реалізовано паралельний алгоритм виконання політочкових перетворень на багатоядерних процесорах. Багатопотокова ЦП-реалізація продемонструвала близьку до лінійної масштабованість прискорення при збільшенні числа ядер (частка паралельних обчислень $\sim 90\%$ за законом Амдала). Виявлено, що продуктивність на ЦП-платформах обмежується ефектами NUMA: за відсутності спеціалізованих заходів відбувається передчасне насичення прискорення при залученні великої кількості потоків.

11. Оптимізовано паралельну реалізацію для NUMA-архітектур. Запропоновано NUMA-орієнтовану стратегію багатопотокової обробки на ЦП із використанням OpenMP: закріплення потоків за вузлами пам'яті та політики «first touch». Застосування цієї стратегії дозволило знизити вплив нерівномірного доступу до пам'яті і прискорити виконання алгоритму на 15–20% (до 30% у кращому випадку) порівняно з базовою реалізацією. Отримано кращу продуктивність, ніж у альтернативної реалізації на основі Pthreads, що підтверджує ефективність вибраного підходу до оптимізації.

12. Реалізовано високопродуктивну ГП-версію алгоритму політочкових перетворень на архітектурі CUDA. Застосовано схему розпаралелювання “одна площа — один потік” та проведено поетапну оптимізацію ядра на основі аналізу профілю виконання. Усунення повільних операцій ділення з подвійною точністю, зниження регістрового тиску та кешування базисних даних у спільній пам’яті дозволило скоротити час виконання ядра на тестовому стенді з 2.14с до 1.30с (сумарне прискорення $\sim 1,65\times$). В результаті досягнуто прискорення методу приблизно у 2.1 раза відносно найпродуктивнішої реалізації на ЦП-кластері (192 ядра) та у ~ 14.8 раза відносно послідовного виконання на ЦП. Це підтверджує переваги гетерогенного підходу та економічну доцільність використання ГП для прискорення політочкових перетворень у задачах моделювання деформацій.

Запропоновані в дисертації методи та алгоритми можуть бути застосовані під час комп’ютерного проєктування й інженерного аналізу виробів, де потрібні контрольовані нелінійні деформації геометрії зі збереженням топологічної цілісності сітки та можливістю інтерактивної/високопродуктивної обробки багатополігональних моделей.

Подальший розвиток виконаних досліджень доцільний в напрямку розробки та вдосконалення методів підвищення точності і чисельної стійкості політочкових перетворень, а також їх подальшого масштабування на базі сучасних гетерогенних обчислювальних архітектурах таких як OpenCL, CUDA, AMD ROCm.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wolter, F.-E., Reuter, M., & Peinecke, N. (2007). Geometric modeling for engineering applications. In E. Stein, R. de Borst, & T. J. R. Hughes (Eds.), *Encyclopedia of computational mechanics: Part 1. Fundamentals*. John Wiley & Sons. <https://doi.org/10.1002/0470091355.ECM013.PUB2>
2. Ausheva, N. M., Sydorenko, I. V., Kaleniuk, O. S., Kardashov, O. V., & Horodetskyi, M. V. (2025). Implicit curves and surfaces modeling with pseudogaussian interpolation. *Radio Electronics, Computer Science, Control*, (1), 30–37. <https://doi.org/10.15588/1607-3274-2025-1-3>
3. Sydorenko, Yu. V., Onysko, A. I., Shaldenko, O. V., & Horodetskyi, M. V. (2022). Interpolation of different types of spiral-like curves by gaus-interpolation methods. *Control Systems and Computers*, 3(299), 1–10. <https://doi.org/10.15407/csc.2022.03.003>
4. Sydorenko Iu.V., Horodetskyi M.V. (2023). Modification of the Algorithm for Defining Polygonal Geometry of an Object for Polypoint Transformations. *Control Systems and Computers*, 4, 12-18. <https://doi.org/10.15407/csc.2023.04.012>
5. Sydorenko, Iu. V., Kaleniuk, O. S., & Horodetskyi, M. V. (2024). Polypoint transformation dependency on the polyfiber configuration. *Control Systems and Computers*, 4, 3–9. <https://doi.org/10.15407/csc.2024.04.003>
6. Horodetskyi, M. V., & Sydorenko, Iu. V. (2025). Methods of defining geometry of an object in three-dimensional space for polypoint transformations. *Elektronnoe modelirovanie*, 47(3), 3–11. <https://doi.org/10.15407/emodel.47.03.003>
7. Horodetskyi, M. V., & Kaleniuk, O. S. (2025). Parallel implementation of polypoint transformations with adjacent triangle plane intersections. *Elektronnoe modelirovanie*, 47(6), 3–10. <https://doi.org/10.15407/emodel.47.06.003>
8. Tzafestas, C. S., Koumpouros, Y., & Birbas, K. (2002). Paracentesis modeling and VR-based interactive simulation with haptic display for clinical skill training and

- assessment. Proceedings of the International Conference on Integrated Modeling and Analysis in Applied Control and Automation (I-MAACA).
9. Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing scenes as neural radiance fields for view synthesis. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 5451–5460.
<https://doi.org/10.48550/arXiv.2003.08934>
 10. Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3D Gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics, 42(4), Article 193. <https://doi.org/10.1145/3592433>
 11. Pumarola, A., Corona, E., Pons-Moll, G., & Moreno-Noguer, F. (2021). D-NeRF: Neural radiance fields for dynamic scenes. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10318–10326.
<https://arxiv.org/abs/2011.12948>
 12. Сидоренко, Ю. В., & Городецький, М. В. (2025, 3–4 червня). Переваги методу політочкових перетворень порівняно з методами NeRF та Gaussian splatting [Тези доповіді]. У Сучасні проблеми геометричного моделювання: тези 27-ї міжнародної науково-практичної конференції (р. 47). Мелітополь, Україна.
<https://magazine.mdpu.org.ua/index.php/spm/issue/download/133/60>
 13. Turner, M. J., Clough, R. W., Martin, H. C., & Topp, L. J. (2012). Stiffness and deflection analysis of complex structures. Journal of the Aeronautical Sciences, 23(9), 805–823. <https://doi.org/10.2514/8.3664>
 14. Müzel, S. D., Bonhin, E. P., Guimarães, N. M., et al. (2020). Application of the finite element method in the analysis of composite materials: A review. Polymers, 12(4), 971. <https://doi.org/10.3390/polym12040818>
 15. Safadi, M. O. (2023). Stress analysis with an introduction to finite element methods (1st ed.). Cognella Academic Publishing.

16. Nealen, A., Müller, M., Keiser, R., Boxerman, E., & Carlson, M. (2006). Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4), 809–846.
17. Chen, Y., Zhu, Q., Kaufman, A., & Muraki, S. (1998). Physically-based animation of volumetric objects. In *Proceedings of the Computer Animation '98 (CA '98)* (p. 154). IEEE. <https://doi.org/10.1109/CA.1998.681920>
18. Allen, M. P., & Tildesley, D. J. (2017). *Computer simulation of liquids* (2nd ed.). Oxford University Press.
<https://doi.org/10.1093/oso/9780198803195.001.0001>
19. Courtecuisse, H., Jung, H., Allard, J., Duriez, C., Lee, D. Y., & Cotin, S. (2010). Using the PhysX engine for physics-based virtual surgery with force feedback. *International Journal of Computer Assisted Radiology and Surgery*, 5(3), 301–308. <https://doi.org/10.1002/rcs.266>
20. Sederberg, T. W., & Parry, S. R. (1986). Free-form deformation of solid geometric models. *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*, 151–160.
<https://doi.org/10.1145/15886.15903>
21. Gu, J., & Wang, L. (2014). Free-Form Deformation of Parametric Surfaces Based on Extension Function with Platform. *Journal of Computational Information Systems*, 10(15), 6789–6797.
22. Coquillart, S. (1990). Extended free-form deformation: A sculpturing tool for 3D geometric modeling. *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '90)*, 187–196.
<https://doi.org/10.1145/97879.97900>
23. Moccozet, L., & Magnenat-Thalmann, N. (1997). Dirichlet free-form deformations and their application to hand simulation. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 149–154.
<https://doi.org/10.1145/258734.258821>

24. Reich, S., & van de Panne, M. B. (1997). Animated free-form deformation: An interactive animation technique. *Proceedings of the 8th Annual Conference on Computer Graphics (VCG '97)*, 177–183.
25. Feng, J., Gu, L., & He, X. (2000). B-spline free-form deformation of polygonal objects. *Proceedings of the 15th International Conference on Pattern Recognition*, 2, 110–113. <https://doi.org/10.1109/GMAP.2000.838272>
26. Бадаєв Ю. І. Політканинні перетворення у конструюванні геометричних об'єктів / Ю. І. Бадаєв, Ю. О. Дорошенко // *Прикладна геометрія та інженерна графіка*. – 1996. – № 60. – С. 32–38.
27. Дорошенко Ю. О. Комп'ютерний синтез геометричної моделі розповсюдження лісових пожеж на основі політканинних перетворень / Ю. О. Дорошенко // *Прикладна геометрія та інженерна графіка*. – 1998. – № 64. – С. 101–105.
28. Дорошенко Ю. О. Політканинне перетворення точок, що належать КБПТ / Ю. О. Дорошенко // *Прикладна геометрія та інженерна графіка*. – 2000. – № 67. – С. 89–91.
29. Бадаєв, Ю. І., & Сидоренко, Ю. В. (2000). Деформаційне конструювання об'єктів водного транспорту за допомогою політочкових перетворень. *Водний транспорт: Збірник наукових праць*, 140–143.
<https://scholar.google.com/scholar?cluster=8342402359139230242>
30. Сидоренко, Ю. В., & Бадаєв, Ю. І. (2000). Конструювання геометричних об'єктів засобами політочкових перетворень. *Прикладна геометрія та інженерна графіка*, (66), 44–47.
31. Sydorenko, I., & Zalevska, O. (2020). Підвищення точності алгоритму політочкових перетворень. *Прикладна геометрія та інженерна графіка*, 97, 129–135. <https://doi.org/10.32347/0131-579x.2020.97.129-135>

32. Sydorenko, Yu. V., & Kalenyuk, O. S. (2007). Моделювання векторного поля деформації геометричних об'єктів. In *Dynamical system modelling and stability investigation: International scientific conference abstracts* (p. 326). Kyiv, Ukraine.
33. Sydorenko, Yu. V., & Kalenyuk, O. S. (2009). Моделювання векторного поля деформації точкового об'єкта інтерполяцією методом Шепарда. *Прикладна геометрія та інженерна графіка*, (82), 180–184.
34. Sydorenko, Yu. V., & Kalenyuk, O. S. (2009). Моделювання векторного поля деформації точкового об'єкту симплексною ваговою інтерполяцією. In *Dynamical system modelling and stability investigation: International scientific conference abstracts* (p. 246). Kyiv, Ukraine.
35. Sydorenko, Yu. V., & Kalenyuk, O. S. (2009). Модифікований спосіб симплексної вагової інтерполяції функції багатьох змінних. *Праці таврійського державного агротехнологічного університету. Том 4. Прикладна геометрія та інженерна графіка*, (44), 102–106.
36. Сидоренко, Ю. В., & Каленюк, О. С. (2009а). Симплекса інтерполяція функції багатьох змінних. *Геометричне модулювання та комп'ютерні технології: теорія, практика, освіта: міжнар. наук.-практ. конф.: наукові нотатки* (с. 314–319). Х.: НТУ «ХП».
37. Сидоренко, Ю. В., & Каленюк, О. С. (2009b). Моделювання векторного поля деформації точкового об'єкту симплексною ваговою інтерполяцією. In *Dynamical system modelling and stability investigation: міжнар. наук. конф.: тези доповідей* (с. 246).
38. Сидоренко, Ю. В., Кривда, О. В., & Лещинська, І. В. (2020). Система моделювання конструктивних елементів вентиляційних систем політочковими перетвореннями. *Опір матеріалів і теорія споруд*, (104), 221–223. <https://orcid.org/0000-0002-8737-4595>
39. Аушева, Н. М., Сидоренко, Ю. В., Демчишин, А. А., & Каленюк, О. С. (2025). Побудова періодичних кривих комбінованим експоненційно-алгебраїчним

інтерполюючим поліномом. Вісник Запорізького національного університету. Фізично-математичні науки, (1), Article 01.

<https://doi.org/10.26661/2786-6254-2025-1-01>

40. Iben, H., O'Brien, J. F., & Shewchuk, J. R. (2009). Refolding planar polygons. *Discrete & Computational Geometry*, 41(3), 444–460.
41. Floater, M. S., & Kos, G. (2021). On planar polynomial geometric interpolation. *Mathematics of Computation*, 90(327), 325–347.
<https://doi.org/10.48550/arXiv.2012.01049>
42. Yan, Z., & Yang, X. (2026). Visualization of curvature monotonicity regions of 3D Bézier curves. *Computer-Aided Design*, 23(1), 41–55.
<https://doi.org/10.14733/cadaps.2026.41-55>
43. Tan, J., Xing, Y., Fan, W., & Hong, P. (2018). Smooth orientation interpolation using parametric quintic-polynomial-based quaternion spline curve. *Journal of Computational and Applied Mathematics*, 330, 256–266.
[45https://doi.org/10.1016/j.cam.2017.07.007](https://doi.org/10.1016/j.cam.2017.07.007)
44. Schoenberg, I. J. (1946). Contributions to the problem of approximation of equidistant data by analytic functions. *Quarterly of Applied Mathematics*, 4(1), 45–99.
45. Корнейчук, Н. П. (1984). Сплайны в теории приближения. М.: Наука.
46. Аушева, Н. М., & Демчишин, А. А. (2019). Формування ортогональних сіток на основі фундаментального сплайну. *Наукові праці МДПУ імені Б. Хмельницького. Серія: Фізико-математичні науки*, 16(10), 16–22.
<https://doi.org/10.33842/2313-125X/2019/16/10/16>
47. Аушева, Н. М., & Данько, Ю. А. (2020). Конструювання дискретних сіток та поверхонь на основі ізотропних В-сплайнів. *Наукові праці МДПУ імені Б. Хмельницького. Серія: Фізико-математичні науки*, 19(3), 10–17.
<https://doi.org/10.33842/2313-125X/2020/19/3/10>
48. Бадаєв, Ю. І. (2004). Криві на основі політканинних В-сплайнів. *Праці Таврійського державного агротехнологічного університету*, 28(4), 16–20.

49. Шепель, В. П., Білицька, Н. В., Гетьман, О. Г., & Грищенко, І. А. (2008). Моделювання кубічних кривих Без'є із застосуванням методу Кунса. Прикладна геометрія та інженерна графіка, 79, 68–72.
50. Shelevytskyi, I. V. (2015). Splines in digital processing data and signals. Kryvyi Rih: Kryvorizkyi State Pedagogical University.
<https://doi.org/10.31812/0564/40>
51. Ванін, В. В., Незенко, А. Й., & Козлов, С. О. (2023). Метод побудови фактичних поверхонь крила літака в процесі його виготовлення та експлуатації. Сучасні проблеми моделювання, 25, 158–168. <https://doi.org/10.33842/2313-125X-2023-25-158-168>
52. Wang, R.-H. (2001). Multivariate spline functions and their applications. Kluwer Academic Publishers. <https://doi.org/10.1007/978-94-017-2378-7>
53. Grošelj, J., & Šadl Praprotnik, A. (2025). Rational C^1 cubic Powell–Sabin B-splines with application to representation of ruled surfaces. Journal of Computational and Applied Mathematics, 457, Article 116292. <https://doi.org/10.1016/j.cam.2024.116292>
54. Rahman, K., Shi, J., & Deng, J. (2026). Quintic uniform algebraic hyperbolic tension B-spline collocation method for solving multi-dimensional Kuramoto-Tsuzuki equation. Journal of Computational and Applied Mathematics, 478, Article 117392. <https://doi.org/10.1016/j.cam.2026.117392>
55. 107. Holladay J.C. Smoothest curve approximation// Math.Tables Aids Comput.-1957.-N11.-P. 233-243.
56. Holladay, J. C. (1957). Smoothest curve approximation. Mathematical Tables and Other Aids to Computation, 11(60), 233–243.
57. Сидоренко Ю. В. (2001). Побудова гладких ліній за допомогою параметризованих функцій Гаусса. Прикладна геометрія та інженерна графіка, 69, 63–67.
58. Сидоренко, Ю. В. (2014). Параметрична інтерполяційна функція Гауса. У Комп'ютерне моделювання в хімії, технологіях і системах сталого розвитку.

КМХТ, 2014: Збірник наукових статей Четвертої міжнар. наук.-прак. конф. (с. 67–73). Київ: НТУУ «КПІ».

<https://ela.kpi.ua/handle/123456789/18714>

59. Сидоренко, Ю. В., & Третьак, В. А. (2015). Обчислення коефіцієнта об'ємної теплоємності при моделюванні процесів плавлення сплавів. *ScienceRise*, 5(2/10), 60–64. <https://doi.org/10.15587/2313-8416.2015.42632>
60. Sydorenko, Yu., Zalevska, O., Horodetskyi, M., & Spiritsev, D. (2022). Аналіз поведінки похибки обчислень при Гаус-інтерполяції функції Рунге. *Сучасні проблеми моделювання*, (23), 159–167. <https://doi.org/10.33842/2313-125X-2023-23-159-167>
61. Sydorenko, Yu., Zalevska, O., Horodetskyi, M., & Naidysh, A. (2022). Особливості розташування базисних вузлів Гаус-функції на прикладі спіралеподібних кривих. *Сучасні проблеми моделювання*, (23), 151–158. <https://doi.org/10.33842/2313-125X-2022-23>
62. Сидоренко, Ю. В., & Городецький, М. В. (2024). Оптимізація методу розв'язання СЛАР для політочкових перетворень [Тези доповіді]. У *Сучасні проблеми геометричного моделювання: тези 26-ї міжнародної науково-практичної конференції* (р. 31). Мелітополь, Україна. <https://doi.org/10.33842/2313-125X-2024-13>
63. Shlafman, S., Tal, A., & Katz, S. (2002). Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum*, 21(3), 219–228. <https://doi.org/10.1111/1467-8659.00581>
64. Floater, M. S., Gillette, A., & Sukumar, N. (2020). Mean value coordinates and their use in geometric modelling. *Computer Aided Geometric Design*, 82, Article 101924. <https://doi.org/10.1016/j.cagd.2005.06.004>
65. Сидоренко, Ю. В., Кривда, О. В., & Городецький, М. В. (2022, 20 грудня). Гаус-інтерполяція спіралеподібних кривих [Матеріали конференції]. У *Наукові*

- підсумки 2022 року: збірка наукових праць XI наукової конференції (р. 8). Харків: Технологічний центр. <https://ela.kpi.ua/handle/123456789/55490>
66. Сидоренко, Ю. В., & Городецький, М. В. (2020). Аналіз роботи алгоритму інтерполяційної функції Гауса на елементарних алгебраїчних функціях. Сучасні проблеми моделювання, (19), 138–145.
<https://doi.org/10.33842/22195203/2020/19/138/145>
67. Sydorenko, Iu. V., & Zalevska, O. (2020). Підвищення точності алгоритму політочкових перетворень. Прикладна геометрія та інженерна графіка, 97, 44–51.
<https://doi.org/10.32347/0131-579x.2020.97.129-135>
68. Ковальов, С., Ботвіновська, С., & Колган, А. (2025). Геометричне моделювання безмоментної оболонки заданої форми. Management of Development of Complex Systems, 61, 193–201. <https://doi.org/10.32347/2412-9933.2025.61.193-201>
69. Каленюк О.С. (2006). Аналіз властивостей простих та модифікованих політочкових перетворень. Магістерська робота. Нац. техн. ун-т України «Київ. політехн. ін-т ім. Ігоря Сікорського», 12(9), 39-42.
<https://doi.org/10.1371/journal.pone.0184206>
70. Barata, J. C. A., & Hussein, M. S. (2012). The Moore–Penrose pseudoinverse: A tutorial review of the theory. Brazilian Journal of Physics, 42(1-2), 146–165.
<https://doi.org/10.1007/s13538-011-0052-z>
71. Liu, T., Chen, M., Song, Y., Li, H., & Lu, B. (2017). Quality improvement of surface triangular mesh using a modified Laplacian smoothing approach avoiding intersection. PLoS ONE, 12(9), Article e0184206.
<https://doi.org/10.1371/journal.pone.0184206>
72. Bookstein, F. L. (1991). Morphometric Tools for Landmark Data: Geometry and Biology. Cambridge University Press.
<https://doi.org/10.1017/CBO9780511573064>

73. Fasshauer, G. E. (2007). *Meshfree Approximation Methods with MATLAB*. World Scientific Publishing Company. <https://doi.org/10.1142/6437>
74. Buhmann, M. D. (2003). *Radial Basis Functions: Theory and Implementations*. Cambridge University Press.
<https://doi.org/10.1017/CBO9780511543241>
75. Wendland, H. (2004). *Scattered Data Approximation*. Cambridge University Press.
<https://doi.org/10.1017/CBO9780511617539>
76. Gomes, J., & Velho, L. (1997). *Warping and Morphing of Graphical Objects*. Morgan Kaufmann.
77. Garland, M., & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, 209–216.
<https://doi.org/10.1145/258734.258849>
78. Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., & Lévy, B. (2010). *Polygon Mesh Processing*. AK Peters/CRC Press.
<https://doi.org/10.1201/b10688>
79. Choi, H. G., Thite, A. N., & Thompson, D. J. (2007). Comparison of methods for parameter selection in Tikhonov regularization with application to inverse force determination. *Journal of Sound and Vibration*, 304(3–5), 894–917.
<https://doi.org/10.1016/j.jsv.2007.03.040>
80. Tikhonov, A. N., & Arsenin, V. Y. (1977). *Solutions of ill-posed problems* (V. Kaplan, Trans.). W. H. Winston & Sons. (Original work published 1974).
<https://doi.org/10.1137/1021044>
81. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). MIT Press.
<https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/>
82. Knuth, D. E. (1997). *The art of computer programming, Volume 1: Fundamental algorithms* (3rd ed.). Addison-Wesley Professional.


83. Grama, A., Gupta, A., Karypis, G., & Kumar, V. (2003). Introduction to parallel computing (2nd ed.). Pearson Education.
84. Zhou, K., Gong, M., Huang, S. S., & Guo, B. (2010). Data-parallel octrees for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 17(5), 669–681. <https://doi.org/10.1109/TVCG.2010.75>
85. Higham, N. J. (2002). Accuracy and stability of numerical algorithms (2nd ed.). Society for Industrial and Applied Mathematics.
<https://doi.org/10.1137/1.9780898718027>
86. Golub, G. H., & Van Loan, C. F. (2013). Matrix computations (4th ed.). Johns Hopkins University Press.
87. Shewchuk, J. R. (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3), 305–363. <https://doi.org/10.1007/PL00009321>
88. Zhong, W. B., Luo, X. C., Chang, W. L., Cai, Y. K., Ding, F., Liu, H. T., & Sun, Y. Z. (2020). Toolpath interpolation and smoothing for computer numerical control machining of freeform surfaces: A review. *International Journal of Automation and Computing*, 17(1).
<https://doi.org/10.1007/s11633-019-1211-5>
89. Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1), 5–48.
<https://doi.org/10.1145/103162.103163>
90. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.
<https://mitpress.mit.edu/9780262046305/>
91. Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 1–14. <https://doi.org/10.1145/3592433>

92. Крылов, В. И., Бобков, В. В., & Монастырный, П. И. (1972). Вычислительные методы высшей математики (И. П. Мысовских, Ред.; Т. 1). Высшая школа.
93. Молчанов, И. Н. (1987). Машинные методы решения прикладных задач. Алгебра, приближение функций. Наукова думка.
94. Boissonnat, J.-D., Dyer, R., & Ghosh, A. (2013). The stability of Delaunay triangulations. *International Journal of Computational Geometry & Applications*, 23(04–05), 303–334.
<https://doi.org/10.1142/S0218195913600078>
95. Самарский А.А. Введение в численные методы [Текст]: Учебн. пособие для вузов / А.А. Самарский. – М.: Наука., 1987. – 288 с.
96. Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics.
97. Trefethen, L. N., & Bau, D. (1997). *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics.
98. Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations*. Johns Hopkins University Press.
99. Higham, N. J., & Mary, T. (2020). A new approach to probabilistic rounding error analysis. *SIAM Journal on Scientific Computing*, 42(2), A281–A306.
100. Demmel, J., Grigori, L., Hoemmen, M., & Langou, J. (2022). Communication-optimal numerical linear algebra. *Acta Numerica*, 31, 1–70.
101. Тихонов, А. Н., Гончарский, А. В., Степанов, В. В., & Ягола, А. Г. (1990). Численные методы решения некорректных задач. Наука.
102. Muller, J. M., Brunie, N., de Dinechin, F., Jeannerod, C. P., Joldes, M., Lefèvre, V., Melquiond, G., Revol, N., & Torres, S. (2018). *Handbook of Floating-Point Arithmetic* (2nd ed.). Birkhäuser Basel. <https://doi.org/10.1007/978-3-319-76526-6>
103. Hill, M. D., & Marty, M. R. (2008). Amdahl's Law in the Multicore Era. *Computer*, 41(7), 33–38. <https://doi.org/10.1109/MC.2008.209>


104. Shapovalova, S. I., & Baranichenko, O. M. (2025, May). Distributed component-oriented production system for controlling of hierarchical object. *System Technologies*, 4(159), 3–10. <https://doi.org/10.34185/1562-9945-4-159-2025-01>
105. Hennessy, J. L., & Patterson, D. A. (2017). *Computer Architecture: A Quantitative Approach* (6th ed.). Morgan Kaufmann. ISBN:978-0-12-811905-1
106. Drepper, U. (2007). What Every Programmer Should Know About Memory. Red Hat, Inc. <https://people.freebsd.org/~lstewart/articles/cpumemory.pdf>
107. Dubois, M., Annavaram, M., & Stenström, P. (2012). *Parallel Computer Organization and Design*. Cambridge University Press.
<https://doi.org/10.1017/9781009447621>
108. Hager, G., & Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press. ISBN 9781439811924
109. Kleen, A. (2005). A NUMA API for Linux. Novell / SUSE Labs.
<https://halobates.de/numaapi3.pdf>
110. McCalpin, J. D. (1995). Memory Bandwidth and Machine Balance in Current High Performance Computers. TCCA Newsletter.
111. Ilsche, T., Schöne, R., Bielert, M., Gocht, A., & Hackenberg, D. (2017). lo2s — Multi-core System and Application Performance Analysis for Linux. In *IEEE International Conference on Cluster Computing (CLUSTER)*.
<https://doi.org/10.1109/CLUSTER.2017.116>
112. Schöne, R., Molka, D., & Werner, M. (2014). Wake-up Latencies for Processor Idle States on Current x86 Processors. *Computer Science – Research and Development*.
<https://doi.org/10.1007/s00450-014-0270-z>
113. Curtis-Maury, M., Singh, K., McKee, S. A., Blagojevic, F., Nikolopoulos, D. S., De Supinski, B. R., & Schulz, M. (2007). Identifying energy-efficient concurrency levels using machine learning. 2007 IEEE International Conference on Cluster Computing, 488–495. <https://doi.org/10.1109/CLUSTER.2007.4629265>

114. Lepers, B., Quema, V., & Fedorova, A. (2015). Thread and memory placement on NUMA systems: Asymmetry matters. In 2015 USENIX Annual Technical Conference (USENIX ATC 15) (pp. 277–289). USENIX Association. <https://www.usenix.org/conference/atc15/technical-session/presentation/lepers>
115. Мочурад, Л. І., & Бойко, Н. І. (2021). Використання технології OpenMP для розрахунку електростатичного поля систем електронної оптики. Системи обробки інформації, (3), 193–200. <https://doi.org/10.15421/40290326>
116. Бут, А. Ю., & Кравець, Н. С. (2025). Застосування OpenMP для побудови паралельних програм. Scientific Collection «InterConf», (181). <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf>
117. OpenMP Architecture Review Board. (2024). OpenMP API Specification, Version 6.0. <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf>
118. Terboven, C., an Mey, D., Schmidl, D., & Jin, H. (2008). Data and thread affinity in OpenMP programs. Computer Science - Research and Development, 22(3–4), 115–120. <https://doi.org/10.1145/1366219.1366222>
119. Nickolls, J. (2007). GPU parallel computing architecture and CUDA programming model. 2007 IEEE Hot Chips 19 Symposium (HCS), 1–12. <https://doi.org/10.1109/HOTCHIPS.2007.7482491>
120. Kirk, D. B., & Hwu, W. W. (2016). Programming Massively Parallel Processors: A Hands-on Approach (3rd ed.). Morgan Kaufmann.
121. Lin, C.-H., Liu, J.-C., & Yang, P.-K. (2020). Performance Enhancement of GPU Parallel Computing Using Memory Allocation Optimization. 2020 14th International Conference on Ubiquitous Information Management and Communication (IMCOM), 1–5. <https://doi.org/10.1109/IMCOM48794.2020.9001771>
122. NVIDIA Corporation. (2025). *CUDA C++ programming guide* (Version 13.1). NVIDIA Developer Documentation. <https://docs.nvidia.com/cuda/cuda-c-programming-guide>

ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОГО ДОСЛІДЖЕННЯ



ЗАТВЕРДЖУЮ
Директор ТОВ «БІ-ХАБ»
В.В. Савченко



Вих. №2901/26-01

АКТ ВПРОВАДЖЕННЯ

результатів дисертаційної роботи Городецького Миколи Вадимовича на тему «Геометричне моделювання деформації об'єкта політочковими перетвореннями на основі паралельних обчислень», яка виконана в Національному технічному університеті України «Київський політехнічний інститут ім. Ігоря Сікорського» (Навчально-науковий Інститут атомної і теплової енергетики, кафедра цифрових технологій в енергетиці)

“29” січня 2026 року

За результатами робіт, проведених та опублікованих на здобуття наукового ступеня доктора філософії Городецьким Миколою Вадимовичем за темою «Геометричне моделювання деформації об'єкта політочковими перетвореннями на основі паралельних обчислень», претендентом було створено розширення до програмного комплексу Blender, яке дозволило:


- удосконалити спосіб задання геометричного об'єкту;
- зберігати цілісність 3D-об'єкту після політочкових перетворень, вперше використовуючи задання цього об'єкта через перетин площин дотичних трикутників;
- удосконалити спосіб Гаус-інтерполяції для підвищення точності обчислень, а саме, за рахунок адаптації варіативного параметру α до форми кривої на кожному кроці обчислювань;
- оптимізувати спосіб обрахунку політочкових перетворень, що призвело до значного скорочення часу підрахунків при збереженні цілісності об'єкту.

Наведене розширення для програмного комплексу Blender застосовувалося ТОВ «БІ-ХАБ» з метою формування високоточної анімаційної 3D-візуалізації платіжних банківських карток із відтворенням оптичних спецефектів (зокрема варіацій кольору, текстурних характеристик, райдужних/інтерференційних ефектів та ефекту переливу фольги). Сформовані візуалізаційні результати надалі використовувалися як еталонні 3D-моделі для виробництва смарт-карток.

Порівняно з типовими інструментами деформації геометрії Laplacian Deform, Mesh Deform, Surface Deform та Lattice (FFD), зазначене розширення забезпечило скорочення загальної тривалості циклу «розробка дизайну — підготовка до виробництва — випуск готової продукції» за рахунок:

1. зменшення обчислювальних витрат на обробку геометричних і візуалізаційних даних;
2. скорочення часу синтезу (рендерингу) анімаційних сцен;
3. оптимізації споживання оперативної пам'яті (RAM) серверними процесорами під час виконання відповідних задач.

Директор з розвитку бізнесу
ТОВ «БІ-ХАБ»



Л.Б. Половинкіна

ДОДАТОК Б. ЛІСТИНГ ПРОЦЕДУРИ GETPOLYPOINTPLANE

Нижче наведена процедура *getPolypointPlane*. На вхід приймає площину *plane*, яку необхідно перетворити та два списки з політочкових базисів: список з початкових *origBasises* та список з результуючими *resBasises*.

```
Plane getPolypointPlane(
    const Plane &plane,
    const vector<Eigen::Vector3d> &origBasises,
    const vector<Eigen::Vector3d> &resBasises)
{
    double a1 = 0, b1 = 0, c1 = 0, d1 = 0, r1 = 0;
    double b2 = 0, c2 = 0, d2 = 0, r2 = 0;
    double c3 = 0, d3 = 0, r3 = 0;
    double d4 = 0, r4 = 0;

    for (size_t i = 0; i < origBasises.size(); ++i) {
        const Vector3d &orig_basis_p = origBasises[i];
        const Vector3d &res_basis_p = resBasises[i];

        double gamma = plane.signDistance(orig_basis_p);
        double gamma_squared = gamma * gamma;

        double x = res_basis_p.x();
        double y = res_basis_p.y();
        double z = res_basis_p.z();

        a1 += x * x / gamma_squared;
        b1 += x * y / gamma_squared;
        c1 += x * z / gamma_squared;
        d1 += x / gamma_squared;

        b2 += y * y / gamma_squared;
```

```

    c2 += y * z / gamma_squared;
    d2 += y / gamma_squared;

    c3 += z * z / gamma_squared;
    d3 += z / gamma_squared;

    d4 += 1 / gamma_squared;

    r1 += x / gamma;
    r2 += y / gamma;
    r3 += z / gamma;
    r4 += 1 / gamma;
}

Matrix4d A;
A << a1 + reg_term, b1, c1, d1,
    b1, b2 + reg_term, c2, d2,
    c1, c2, c3 + reg_term, d3,
    d1, d2, d3, d4 + reg_term;

Vector4d B(r1, r2, r3, r4);
Vector4d X = A.lu().solve(B);

return {plane.id, X(0), X(1), X(2), X(3)};
}

```

ДОДАТОК В. ЛІСТИНГ ПІДГОТОВКИ ТА ЗАПУСКУ ПРОЦЕДУРИ GETPOLYPOINTPLANE НА ВСІХ ДОСТУПНИХ ЯДРАХ ПРОЦЕСОРА

Нижче наведена багатопотокова реалізація методу політочкових перетворень із заданням геометрії способом пересічних площин (Спосіб 3).

```
namespace geom {

constexpr double reg_term = 1e-6;

struct Plane
{
    int id;
    double a, b, c, d;

    double signDistance(const Eigen::Vector3d &point) const
    {
        return a * point.x() + b * point.y() + c * point.z() + d;
    }
};

Plane getPolypointPlane(
    const Plane &plane,
    const vector<Eigen::Vector3d> &origBasises,
    const vector<Eigen::Vector3d> &resBasises)
{
    double a1 = 0, b1 = 0, c1 = 0, d1 = 0, r1 = 0;
    double b2 = 0, c2 = 0, d2 = 0, r2 = 0;
    double c3 = 0, d3 = 0, r3 = 0;
    double d4 = 0, r4 = 0;
```

```

for (size_t i = 0; i < origBasises.size(); ++i) {
    const Eigen::Vector3d &orig_basis_p = origBasises[i];
    const Eigen::Vector3d &res_basis_p = resBasises[i];

    double gamma = plane.signDistance(orig_basis_p);
    double gamma_squared = gamma * gamma;

    double x = res_basis_p.x();
    double y = res_basis_p.y();
    double z = res_basis_p.z();

    a1 += x * x / gamma_squared;
    b1 += x * y / gamma_squared;
    c1 += x * z / gamma_squared;
    d1 += x / gamma_squared;

    b2 += y * y / gamma_squared;
    c2 += y * z / gamma_squared;
    d2 += y / gamma_squared;

    c3 += z * z / gamma_squared;
    d3 += z / gamma_squared;

    d4 += 1 / gamma_squared;

    r1 += x / gamma;
    r2 += y / gamma;
    r3 += z / gamma;
    r4 += 1 / gamma;
}

Eigen::Matrix4d A;

```

```

A << a1 + reg_term, b1, c1, d1, //
    b1, b2 + reg_term, c2, d2, //
    c1, c2, c3 + reg_term, d3, //
    d1, d2, d3, d4 + reg_term;

Eigen::Vector4d B(r1, r2, r3, r4);
Eigen::Vector4d X = A.lu().solve(B);

return {plane.id, X(0), X(1), X(2), X(3)};
}

using PlaneList = vector<geom::Plane>;

} // namespace geom

struct ElapsedTimer
{
    chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();

    double elapsedSec()
    {
        chrono::duration<double> dur = chrono::high_resolution_clock::now() - start;
        return dur.count();
    }
};

auto readPlanesBin(auto filePath)
{
    // Assume data stores as list of double.
    // Each 4 double are 1 Plane(a, b, c, d).

    // Open the binary file

```

```

ifstream file{ filePath, ios::binary };

streamsize fileSize = [&file] {
    file.seekg(0, ios::end);
    auto size = file.tellg();
    file.seekg(0, ios::beg);
    return size;
}();

// Calculate the number of elements
size_t num_elements = fileSize / sizeof(double);

// Read the data into a vector
vector<double> data(num_elements);
file.read(reinterpret_cast<char *>(data.data()), fileSize);

vector<geom::Plane> inPlanes;
// Convert to Planes
inPlanes.reserve(data.size() / 4);
for (auto &&chunk : data | views::chunk(4)) {
    inPlanes.emplace_back(0, chunk[0], chunk[1], chunk[2], chunk[3]);
}

return inPlanes;
}

geom::PlaneList collectThreadResults(
    const geom::PlaneList &inPlanes, const vector<geom::PlaneList> &threadResults)
// Collect results from all threads
geom::PlaneList result;
result.reserve(inPlanes.size());
for (const auto &partialResult : threadResults) {
    result.insert(result.end(), partialResult.begin(), partialResult.end());
}

```

```

    }
    return result;
}

```

```

geom::PlaneList threadChunkApproach(
    const geom::PlaneList &inPlanes, const auto &basis_in, const auto &basis_out, auto
threadCount)
{
    vector<jthread> threads;
    size_t chunkSize = inPlanes.size() / threadCount;

    auto threadResultsPtr = std::make_unique<vector<geom::PlaneList>>(threadCount);
    auto &threadResults = *threadResultsPtr;

    // Start parallel execution
    for (size_t i = 0; i < threadCount; ++i) {
        auto start = inPlanes.begin() + i * chunkSize;
        auto end = (i == threadCount - 1) ? inPlanes.end() : start + chunkSize;

        threads.emplace_back([&, start, end, i] {
            for (auto &&[_ , inPlane] : ranges::subrange(start, end) | views::enumerate) {
                threadResults[i].push_back(getPolypointPlane(inPlane, basis_in, basis_out));
            }
        });
    }

    std::ranges::for_each(threads, &jthread::join);
    return collectThreadResults(inPlanes, threadResults);
}

int main()
{
    constexpr auto runEachExperement = 3;

```



```

auto currPath = fs::current_path();
auto planesFile = currPath.append(«in_planes.npy»);
cout << format(«Planes file: {} \n», planesFile.string());

auto timer = ElapsedTimer{ };
auto inPlanes = readPlanesBin(planesFile);

cout << format(«Read {} planes in {} seconds \n», inPlanes.size(), timer.elapsedSec());
cout << format(«Last plane: {} {} {} {} \n», inPlanes.back().a, inPlanes.back().b,
inPlanes.back().c, inPlanes.back().d);
cout << endl;
auto basis_in = vector<Eigen::Vector3d>{
    {0.0, 0.0, 1.0}, {0.0, 1.0, 1.0}, {0.0, 0.0, 0.0}, {0.0, 1.0, -0.0},
    {1.0, 0.0, 1.0}, {1.0, 1.0, 1.0}, {1.0, 0.0, 0.0}, {1.0, 1.0, -0.0}};
auto basis_out = vector<Eigen::Vector3d>{
    {0.0, 0.0, 1.0}, {0.2, 0.2, 1.0}, {0.0, 0.0, 0.0}, {0.0, 1.0, -0.0},
    {0.2, -0.2, 1.0}, {1.18, 0.78, 1.0}, {1.0, 6.12e-17, 0.0}, {1.0, 1.0, -0.0}};
// serialApproach(inPlanes, basis_in, basis_out);
for (size_t threadCount = 1; threadCount <= thread::hardware_concurrency(); threadCount++)
    std::vector<double> times;

    for (int i = 0; i < runEachExperement; ++i) {
        ElapsedTimer timer;
        auto result = threadChunkApproach(inPlanes, basis_in, basis_out, threadCount);
        times.push_back(timer.elapsedSec());
    }
    auto elapsedSec = std::accumulate(times.begin(), times.end(), 0.0) / runEachExperement;
    cout << format(«{} {}; sec», threadCount, elapsedSec) << endl;
}
return 0;
}

```

ДОДАТОК Г. ЛІСТИНГ ПІДГОТОВКИ ТА ЗАПУСКУ ПРОЦЕДУРИ GETPOLYPOINTPLANE OPENMP НА КЛАСТЕРІ AMAZON

Скрипт розгортання 2х інстансів на AWS кластері.

```

pip install awscli --upgrade
aws configure
# Get a suitable Ubuntu AMI
REGION=US
AMI_ID=$(aws ec2 describe-images \
  --region «$REGION» \
  --owners 099720109477 \
  --filters «Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*» \
    «Name=architecture,Values=x86_64» \
  --query 'sort_by(Images, &CreationDate)[-1].ImageId' \
  --output text)

echo «Using AMI: $AMI_ID»
#Create a security group for SSH + intra-cluster traffic
VPC_ID= vpc-01234567890abcdef
REGION=US

SG_ID=$(aws ec2 create-security-group \
  --region «$REGION» \
  --group-name c7a-openmp-sg \
  --description «SG for c7a.metal-48xl OpenMP nodes» \
  --vpc-id «$VPC_ID» \
  --query 'GroupId' \
  --output text)

echo «Security group: $SG_ID»

```

```

MY_IP=$(curl -s https://checkip.amazonaws.com)/32
aws ec2 authorize-security-group-ingress \
  --region «$REGION» \
  --group-id «$SG_ID» \
  --ip-permissions
«IpProtocol=tcp,FromPort=22,ToPort=22,IpRanges=[{ CidrIp=$MY_IP,Description=\»SSH  from  my
IP\«}]»

# Allow *all* traffic between instances in this SG
aws ec2 authorize-security-group-ingress \
  --region «$REGION» \
  --group-id «$SG_ID» \
  --ip-permissions «IpProtocol=-1,UserIdGroupPairs=[{ GroupId=$SG_ID,Description=\»Intra-
SG traffic\«}]»

# Launch 2 × c7a.metal-48xl instances with user-data to install GCC
aws ec2 run-instances \
  --region «$REGION» \
  --image-id «$AMI_ID» \
  --count 2 \
  --instance-type c7a.metal-48xl \
  --key-name «$KEY_NAME» \
  --subnet-id «$SUBNET_ID» \
  --security-group-ids «$SG_ID» \
  --user-data file://horodetskyi-mv-data-openmp.sh \
  --tag-specifications 'ResourceType=instance,Tags=[{ Key=Name,Value=openmp-c7a-node}]'

aws ec2 describe-instances \
  --region «$REGION» \
  --filters «Name=tag:Name,Values=openmp-c7a-node» «Name=instance-state-
name,Values=running» \
  --query
'Reservations[].Instances[].[ID:InstanceId,PrivateIp:PrivateIpAddress,PublicIp:PublicIpAddress]' \

```

```

--output table

# Compile with -O3 -fopenmp and run (on each instance)
ssh -i «$KEY_FILE» ubuntu@$NODE1_PUBLIC_IP

cd /home/ubuntu
g++ -O3 -fopenmp openmp_approach.cpp -o openmp_approach

export OMP_NUM_THREADS=$(nproc) # 192 core
./openmp_approach
ssh -i «$KEY_FILE» ubuntu@172.31.42.242

cd /home/ubuntu
g++ -O3 -fopenmp openmp_approach.cpp -o openmp_approach
export OMP_NUM_THREADS=$(nproc)
./openmp_approach

```

Лістинг алгоритму політочкових перетворень модифікований для роботи з OpenMP API

```

static geom::PlaneList openmpApproach(
    const geom::PlaneList &inPlanes,
    const vector<Eigen::Vector3d> &basis_in,
    const vector<Eigen::Vector3d> &basis_out)
{
    geom::PlaneList result(inPlanes.size());

    // First-touch happens here.
    const size_t planesSize = inPlanes.size();

    ElapsedTimer timer;
    #pragma omp parallel for schedule(static)

```

```

for (size_t i = 0; i < planesSize; ++i) {
    result[i] = geom::getPolypointPlane(inPlanes[i], basis_in, basis_out);
}
// cout << std::format(«OpenMP. Deformation took: {} \n», timer.elapsedSec());
return result;
}

Plane getPolypointPlane(
    const Plane &plane,
    const vector<Eigen::Vector3d> &origBasises,
    const vector<Eigen::Vector3d> &resBasises)
{
    double a1 = 0, b1 = 0, c1 = 0, d1 = 0, r1 = 0;
    double b2 = 0, c2 = 0, d2 = 0, r2 = 0;
    double c3 = 0, d3 = 0, r3 = 0;
    double d4 = 0, r4 = 0;

    for (size_t i = 0; i < origBasises.size(); ++i) {
        const Eigen::Vector3d &orig_basis_p = origBasises[i];
        const Eigen::Vector3d &res_basis_p = resBasises[i];

        double gamma = plane.signDistance(orig_basis_p);
        double gamma_squared = gamma * gamma;

        double x = res_basis_p.x();
        double y = res_basis_p.y();
        double z = res_basis_p.z();

        a1 += x * x / gamma_squared;
        b1 += x * y / gamma_squared;
        c1 += x * z / gamma_squared;
        d1 += x / gamma_squared;
    }
}

```

```

    b2 += y * y / gamma_squared;
    c2 += y * z / gamma_squared;
    d2 += y / gamma_squared;

    c3 += z * z / gamma_squared;
    d3 += z / gamma_squared;

    d4 += 1 / gamma_squared;

    r1 += x / gamma;
    r2 += y / gamma;
    r3 += z / gamma;
    r4 += 1 / gamma;
}

Eigen::Matrix4d A;
A << a1 + reg_term, b1, c1, d1, //
    b1, b2 + reg_term, c2, d2, //
    c1, c2, c3 + reg_term, d3, //
    d1, d2, d3, d4 + reg_term;

Eigen::Vector4d B(r1, r2, r3, r4);
Eigen::Vector4d X = A.lu().solve(B);

return {plane.id, X(0), X(1), X(2), X(3)};
}

int main()
{
    // Make Eigen single-threaded to avoid nested parallelism:
    Eigen::setNbThreads(1);

```

```

constexpr int runEachExperement = 3;
auto currPath = fs::current_path();
auto planesFile = currPath.append(«in_planes.npy»);
cout << std::format(«Planes file: { }\n», planesFile.string());
auto timer = ElapsedTimer{ };
auto inPlanes = readPlanesBin(planesFile);
increasePlanesAount(inPlanes);

// clang-format off
cout << std::format(«Read { } planes in { } seconds\n», inPlanes.size(), timer.elapsedSec());
cout << std::format(«Last plane: { } { } { } { }\n»,
                    inPlanes.back().a, inPlanes.back().b, inPlanes.back().c, inPlanes.back().d);
cout << endl;
auto basis_in = vector<Eigen::Vector3d>{ };
auto basis_out = vector<Eigen::Vector3d>{ };
const auto maxThreads = omp_get_max_threads();
for (size_t threadCount = 1; threadCount <= maxThreads; threadCount++) {
    std::vector<double> times;
    for (int i = 0; i < runEachExperement; ++i) {
        omp_set_num_threads(threadCount);
        ElapsedTimer timer;
        auto result = openmpApproach(inPlanes, basis_in, basis_out);
        times.push_back(timer.elapsedSec());
    }

    auto elapsedSec = std::accumulate(times.begin(), times.end(), 0.0) / runEachExperement;
    cout << std::format(«{ }; { }; sec\n», threadCount, elapsedSec);
}

return 0;
}

```

ДОДАТОК І. ЛІСТИНГ НЕОПТИМІЗОВАНОГО CUDA-ЯДРА ДЕФОРМАЦІЇ ПЛОЩИН МЕТОДОМ ПОЛІТОЧКОВИХ ПЕРЕТВОРЕНЬ

```

#include <cuda_runtime.h>
#include <thrust/device_vector.h>
#include «common.h»
namespace gpu {

__device__ double signDistance(const geom::Plane &p, const double3 &point)
{
    return p.a * point.x + p.b * point.y + p.c * point.z + p.d;
}

__device__ void solve4x4(const double A[16], const double B[4], double X[4])
{
    // Simple Gauss elimination for small 4x4 system

    double M[4][5]; // 4x4 matrix + RHS
#pragma unroll
    for (int i = 0; i < 4; ++i) {
#pragma unroll
        for (int j = 0; j < 4; ++j) {
            M[i][j] = A[i * 4 + j];
        }
        M[i][4] = B[i];
    }

    // Forward elimination
#pragma unroll
    for (int i = 0; i < 4; ++i) {
        double pivot = M[i][i];
#pragma unroll

```



```

        for (int j = i; j <= 4; ++j) {
            M[i][j] /= pivot;
        }
#pragma unroll
        for (int k = i + 1; k < 4; ++k) {
            double factor = M[k][i];
#pragma unroll
            for (int j = i; j <= 4; ++j) {
                M[k][j] -= factor * M[i][j];
            }
        }
    }

    // Back substitution
#pragma unroll
    for (int i = 3; i >= 0; --i) {
        X[i] = M[i][4];
#pragma unroll
        for (int j = i + 1; j < 4; ++j) {
            X[i] -= M[i][j] * X[j];
        }
    }
}

__device__ geom::Plane getPolypointPlaneCUDA(
    const geom::Plane &plane, const double3 *origBasises, const double3 *resBasises, int
basisCount)
{
    double a1 = 0, b1 = 0, c1 = 0, d1 = 0, r1 = 0;
    double b2 = 0, c2 = 0, d2 = 0, r2 = 0;
    double c3 = 0, d3 = 0, r3 = 0;
    double d4 = 0, r4 = 0;

```

```

for (int i = 0; i < basisCount; ++i) {
    const double3 &orig_basis_p = origBasises[i];
    const double3 &res_basis_p = resBasises[i];

    double gamma = signDistance(plane, orig_basis_p);
    double gamma_squared = gamma * gamma;
    double x = res_basis_p.x;
    double y = res_basis_p.y;
    double z = res_basis_p.z;
    a1 += x * x / gamma_squared;
    b1 += x * y / gamma_squared;
    c1 += x * z / gamma_squared;
    d1 += x / gamma_squared;
    b2 += y * y / gamma_squared;
    c2 += y * z / gamma_squared;
    d2 += y / gamma_squared;
    c3 += z * z / gamma_squared;
    d3 += z / gamma_squared;
    d4 += 1.0 / gamma_squared;
    r1 += x / gamma;
    r2 += y / gamma;
    r3 += z / gamma;
    r4 += 1.0 / gamma;
}
// clang-format off
double A[16] = {
    a1 + consts::reg_term, b1, c1, d1,
    b1, b2 + consts::reg_term, c2, d2,
    c1, c2, c3 + consts::reg_term, d3,
    d1, d2, d3, d4 + consts::reg_term
};
// clang-format on

```

```

double B[4] = {r1, r2, r3, r4};
double X[4];
solve4x4(A, B, X);

return geom::Plane{plane.id, X[0], X[1], X[2], X[3]};
}

// Deform each plain on CUDA thread
__global__ void deformPlanesPolypointKernel(
    const geom::Plane *inPlanes,
    geom::Plane *outPlanes,
    const double3 *origBasises,
    const double3 *resBasises,
    int basisCount,
    int planeCount)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx >= planeCount) {
        return;
    }

    outPlanes[idx] = getPolypointPlaneCUDA(inPlanes[idx], origBasises, resBasises, basisCount);
}

void deformPlanesPolypoint(
    __out geom::PlaneList &outPlanes,
    const geom::PlaneList &inPlanes,
    const geom::BasisList &origBasises,
    const geom::BasisList &resBasises)
{
    // Must be the same size for Polypoint transformation!

```

```

assert(origBasises.size() == resBasises.size());

int planeCount = inPlanes.size();
int basisCount = origBasises.size();

// Upload data to GPU
thrust::device_vector<geom::Plane> d_inPlanes = inPlanes;
thrust::device_vector<geom::Plane> d_outPlanes(planeCount);
thrust::device_vector<double3> d_origBasises = origBasises;
thrust::device_vector<double3> d_resBasises = resBasises;

// Launch kernel
int blockSize = 256;
int gridSize = (planeCount + blockSize - 1) / blockSize;

deformPlanesPolypointKernel<<<gridSize, blockSize>>>(<
    thrust::raw_pointer_cast(d_inPlanes.data()),
    thrust::raw_pointer_cast(d_outPlanes.data()),
    thrust::raw_pointer_cast(d_origBasises.data()),
    thrust::raw_pointer_cast(d_resBasises.data()),
    basisCount,
    planeCount);
cudaDeviceSynchronize();

// Download result from GPU
outPlanes.resize(planeCount);
thrust::copy(d_outPlanes.begin(), d_outPlanes.end(), outPlanes.begin());
}

} // namespace gpu

```

ДОДАТОК Д. ЛІСТИНГ ОПТИМІЗОВАНОГО CUDA-ЯДРА ДЕФОРМАЦІЇ ПЛОЩИН МЕТОДОМ ПОЛІТОЧКОВИХ ПЕРЕТВОРЕНЬ

```

#include <cuda_runtime.h>
#include <thrust/device_vector.h>
#include «common.h»
namespace gpu {
__device__ double signDistance(const geom::Plane &p, const double3 &point)
{
    // p.a * point.x + p.b * point.y + p.c * point.z + p.d;
    double t = fma(p.c, point.z, p.d); // p.c*point.z + p.d
    t = fma(p.b, point.y, t); // p.b*point.y + t
    return fma(p.a, point.x, t); // p.a*point.x + t
}

__device__ void solve4x4(const double A[16], const double B[4], double X[4])
{
    // Simple Gauss elimination for small 4x4 system

    double M[4][5]; // 4x4 matrix + RHS
#pragma unroll
    for (int i = 0; i < 4; ++i) {
#pragma unroll
        for (int j = 0; j < 4; ++j) {
            M[i][j] = A[i * 4 + j];
        }
        M[i][4] = B[i];
    }

    // Forward elimination
#pragma unroll
    for (int i = 0; i < 4; ++i) {

```

```

    double pivot = M[i][i];
#pragma unroll
    for (int j = i; j <= 4; ++j) {
        M[i][j] /= pivot;
    }
#pragma unroll
    for (int k = i + 1; k < 4; ++k) {
        double factor = M[k][i];
#pragma unroll
        for (int j = i; j <= 4; ++j) {
            M[k][j] = fma(-factor, M[i][j], M[k][j]); // M[k][j] -= factor * M[i][j];
        }
    }
}

// Back substitution
#pragma unroll
for (int i = 3; i >= 0; --i) {
    X[i] = M[i][4];
#pragma unroll
    for (int j = i + 1; j < 4; ++j) {
        X[i] = fma(-M[i][j], X[j], X[i]); // X[i] -= M[i][j] * X[j];
    }
}

// clang-format off
__device__ geom::Plane getPolypointPlaneCUDA(
    const geom::Plane &plane, const double3 *origBasises, const double3 *resBasises, int
basisCount)
{
    double acc[14] = {0.0};

```

```

double rhs[4] = {0.0};

for (int i = 0; i < basisCount; ++i) {
    const double3 &orig_basis_p = origBasises[i];
    const double3 &res_basis_p = resBasises[i];

    const double gamma      = signDistance(plane, orig_basis_p);
    const double gamma_inv   = __drcp_rn(gamma);
    const double gamma_sq_inv = gamma_inv * gamma_inv;

    const double x = res_basis_p.x;
    const double y = res_basis_p.y;
    const double z = res_basis_p.z;

    // Accumulate with FMA
    acc[0] = fma(x * x, gamma_sq_inv, acc[0]); // a1 = x * x / gamma_squared;
    acc[1] = fma(x * y, gamma_sq_inv, acc[1]); // b1 = x * y / gamma_squared;
    acc[2] = fma(x * z, gamma_sq_inv, acc[2]); // c1 = x * z / gamma_squared;
    acc[3] = fma(x,    gamma_sq_inv, acc[3]);   // d1 = x / gamma_squared;

    acc[4] = fma(y * y, gamma_sq_inv, acc[4]); // b2 = y * y / gamma_squared;
    acc[5] = fma(y * z, gamma_sq_inv, acc[5]); // c2 = y * z / gamma_squared;
    acc[6] = fma(y,    gamma_sq_inv, acc[6]);   // d2 = y / gamma_squared;

    acc[7] = fma(z * z, gamma_sq_inv, acc[7]); // c3 = z * z / gamma_squared;
    acc[8] = fma(z,    gamma_sq_inv, acc[8]);   // d3 = z / gamma_squared;

    acc[9] = fma(1.0,  gamma_sq_inv, acc[9]); // d4 = 1 / gamma_squared;

    // RHS
    rhs[0] = fma(x, gamma_inv, rhs[0]);          // r1 = x / gamma;
    rhs[1] = fma(y, gamma_inv, rhs[1]);          // r2 = y / gamma;

```

```

    rhs[2] = fma(z, gamma_inv, rhs[2]);          // r3 = z / gamma;
    rhs[3] = fma(1.0, gamma_inv, rhs[3]);        // r4 = 1 / gamma;
}

const double A[16] = {
    acc[0] + consts::reg_term, acc[1], acc[2], acc[3],
    acc[1], acc[4] + consts::reg_term, acc[5], acc[6],
    acc[2], acc[5], acc[7] + consts::reg_term, acc[8],
    acc[3], acc[6], acc[8], acc[9] + consts::reg_term,
};

double X[4];
solve4x4(A, rhs, X);

return geom::Plane{plane.id, X[0], X[1], X[2], X[3]};
}

// clang-format on

// Deform each plain on CUDA thread
__global__ void deformPlanesPolypointKernel(
    const geom::Plane *inPlanes,
    geom::Plane *outPlanes,
    const double3 *origBasises,
    const double3 *resBasises,
    int basisCount,
    int planeCount)
{
    extern __shared__ double3 sharedMemory[]; // dynamic shared memory
    double3 *sharedOrigBasises = sharedMemory;
    double3 *sharedResBasises = sharedMemory + basisCount;

```



```

// Only one thread per block loads origBasises and resBasises into shared memory
for (int i = threadIdx.x; i < basisCount; i += blockDim.x) {
    sharedOrigBasises[i] = origBasises[i];
    sharedResBasises[i] = resBasises[i];
}
__syncthreads(); // wait for all threads to finish copying

int idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx >= planeCount) {
    return;
}

outPlanes[idx] = getPolypointPlaneCUDA(
    inPlanes[idx], sharedOrigBasises, sharedResBasises, basisCount);
}

void checkErrorCUDA(const char *msg)
{
    cudaError_t err = cudaGetLastError();
    if (err != cudaSuccess) {
        std::cerr << «CUDA Error: » << msg << «: » << cudaGetErrorString(err) << std::endl;
        exit(EXIT_FAILURE);
    }
}

void deformPlanesPolypoint(
    __out geom::PlaneList &outPlanes,
    const geom::PlaneList &inPlanes,
    const geom::BasisList &origBasises,
    const geom::BasisList &resBasises)
{
    // Must be the same size for Polypoint transformation!

```

```

assert(origBasises.size() == resBasises.size());

int planeCount = inPlanes.size();
int basisCount = origBasises.size();

// Upload data to GPU
thrust::device_vector<geom::Plane> d_inPlanes = inPlanes;
thrust::device_vector<geom::Plane> d_outPlanes(planeCount);
thrust::device_vector<double3> d_origBasises = origBasises;
thrust::device_vector<double3> d_resBasises = resBasises;
// Launch kernel
int blockSize = 512;
int gridSize = (planeCount + blockSize - 1) / blockSize;
size_t sharedMemSize = 2 * basisCount * sizeof(double3);
std::cout << «CUDA SM config: blockSize: » << blockSize << «, gridSize: » << gridSize
          << «, sharedMemSize: » << sharedMemSize << std::endl;
deformPlanesPolypointKernel<<<gridSize, blockSize, sharedMemSize>>>(
    thrust::raw_pointer_cast(d_inPlanes.data()),
    thrust::raw_pointer_cast(d_outPlanes.data()),
    thrust::raw_pointer_cast(d_origBasises.data()),
    thrust::raw_pointer_cast(d_resBasises.data()),
    basisCount,
    planeCount);
checkErrorCUDA(«Kernel launch failed»);
cudaDeviceSynchronize();
checkErrorCUDA(«Kernel execution failed»);
// Download result from GPU
outPlanes.resize(planeCount);
thrust::copy(d_outPlanes.begin(), d_outPlanes.end(), outPlanes.begin());
}

} // namespace gpu

```

ДОДАТОК Е. ЛІСТИНГ РОЗШИРЕННЯ ДЛЯ ПРОГРАМНОГО ПАКЕТУ BLENDER

```

try:
    import numpy as np
    from scipy.optimize import minimize
except ImportError:
    # Install modules for Blender's Python env
    python_exe = os.path.join(sys.prefix, "bin", "python.exe")
    subprocess.call([python_exe, "-m", "ensurepip"])
    subprocess.call(
        [
            python_exe,
            "-m",
            "pip",
            "install",
            "--target",
            "C:\\Program Files\\Blender Foundation\\Blender 4.3\\4.3\\python\\lib\\site-packages",
            "scipy",
            "numpy",
        ]
    )

import numpy as np
from scipy.optimize import minimize
import math
import bpy
from bpy.types import BlendDataObjects, Object
from copy import deepcopy
import traceback

```

```

# Helper functions
# def get_mesh_data(obj: Object):
#     """Get vertices and faces from a Blender object."""
#     mesh = obj.data
#     vertices = np.array([v.co for v in mesh.vertices])
#     faces = [tuple(p.vertices) for p in mesh.polygons]
#     return vertices, faces

# def set_mesh_data(obj: Object, vertices):
#     """Set vertices in a Blender object."""
#     mesh = obj.data
#     for i, coord in enumerate(vertices):
#         mesh.vertices[i].co = coord

def get_mesh_data(obj: Object):
    """Get vertices and faces from a Blender object."""
    mesh = obj.data
    vertices = [[v.co.x, v.co.y, v.co.z] for v in mesh.vertices]
    faces = [
        [
            p.vertices[0],
            p.vertices[1],
            p.vertices[2],
        ]
        for p in mesh.polygons
    ]
    return vertices, faces

def set_mesh_data(obj: Object, vertices: list):
    """Set vertices in a Blender object."""

```

```

if obj.mode == "EDIT":
    bpy.ops.object.mode_set(mode="OBJECT")

mesh = obj.data
for i, coord in enumerate(vertices):
    mesh.vertices[i].co = coord

mesh.update()
mesh.validate()

bpy.ops.object.mode_set(mode="EDIT")

# Main function
def deform_mesh(
    input_name,
    deformation_basis_from_name,
    deformation_basis_to_name,
    output_name,
    topology,
):
    input_obj = bpy.data.objects[input_name]
    basis_from_obj = bpy.data.objects[deformation_basis_from_name]
    basis_to_obj = bpy.data.objects[deformation_basis_to_name]
    output_obj = bpy.data.objects[output_name]

    # Get mesh data
    input_vertices, input_faces = get_mesh_data(input_obj)
    print("Input Vertices:", input_vertices[0])
    basis_from_vertices, basis_from_faces = get_mesh_data(basis_from_obj)
    basis_to_vertices, basis_to_faces = get_mesh_data(basis_to_obj)

```

```

# print("Input Vertices:", input_vertices)
# print("Input Triangles:", input_faces)

# Build planes from faces
in_planes, in_planes_for_vertex_dict = build_planes(
    input_vertices, input_faces, topology
)

# Get transformed planes
start_time = time.time()
tr_planes = get_polypoint_planes_list(
    in_planes, orig_basises=basis_from_vertices, res_basises=basis_to_vertices
)
tr_vertexes = get_transformed_vertexes(
    in_planes_for_vertex_dict, tr_planes, topology
)
print(f"Transformation took: {time.time() - start_time} seconds")

# Apply deformation to output object
set_mesh_data(output_obj, tr_vertexes)

# Parameters
# DEFORMATION_INPUT = "thorus_80v"
# DEFORMATION_BASIS_FROM = "icosphere_80v"
# DEFORMATION_BASIS_TO = "icosphere_deformed_80v"
# DEFORMED_OUTPUT = "pp_thorus_deformed_80v"
DEFORMATION_INPUT = "bunny_1_decimated"
DEFORMATION_BASIS_FROM = "icosphere_80v"
DEFORMATION_BASIS_TO = "icosphere_deformed_80v"
DEFORMED_OUTPUT = "bunny_1_deformed_1"
topology = Topology.Intersect

```

```

#         deform_mesh(DEFORMATION_INPUT,          DEFORMATION_BASIS_FROM,
DEFORMATION_BASIS_TO, DEFORMED_OUTPUT)

```

```

def compare_listcomp(x, y):
    if x is None or y is None:
        return x is None and y is None # Both must be None to return True
    if len(x) != len(y): # Check length first
        return False
    for i, j in zip(x, y):
        if i != j:
            return False
    return True

```

```

last_basis_to_vertices = None

```

```

def update_deformation(scene):
    global last_basis_to_vertices

    basis_to_obj = bpy.data.objects[DEFORMATION_BASIS_TO]
    basis_to_vertices, basis_to_faces = get_mesh_data(basis_to_obj)
    if compare_listcomp(basis_to_vertices, last_basis_to_vertices):
        print("No changes in", DEFORMATION_BASIS_TO, ", skipping update")
        return
    last_basis_to_vertices = deepcopy(basis_to_vertices)

    try:
        deform_mesh(
            DEFORMATION_INPUT,

```

```

    DEFORMATION_BASIS_FROM,
    DEFORMATION_BASIS_TO,
    DEFORMED_OUTPUT,
    topology,
)
print(f"{DEFORMATION_BASIS_TO} updated, recalculating deformation...")
except Exception as e:
    print(
        f"===== Failed (topo={topology}):
{traceback.format_exc()}"
    )
# Register the handler
bpy.app.handlers.depsgraph_update_post.clear()
bpy.app.handlers.depsgraph_update_post.append(update_deformation)

# NOTE: to auto apply, select Edit -> change vertexes -> Unselect -> Save -> Select any vertex.
# NOTE: we can't mix .obj deformed models and models created in Blender because of different
AXIS during export/import.
# NOTE: export IN/OUT model and IN/OUT basises to .obj and then import them back to
Blender to have comparable AXIS.

```