УДК 004.925.8, 004.272.2

Терещенко В.М., д. ф. -м. н., проф.,
Осадчий Б.О., студент
Петльована М.В., студентка,
Селезньов А.І., студент,
Тара А.М., студент.

Архітектура моделі єдиного алгоритмічного середовища (МЄАС) для розв'язання задач обчислювальної геометрії

Київський національний університет імені Тараса Шевченка, 83000, м. Київ, Глушкова 4д
e-mail: v_ter@ukr.net
e-mail: bodyaosadchiy@gmail.com
e-mail: mariapetlovana@gmail.com
e-mail: anatoliy.seleznyov@gmail.com
e-mail: derubioo@gmail.com

V.M. Tereshchenko, doctor of ph.-mat. sc., prof.,
B.O. Osadchyi, student,
M.V. Petlovana, student,
A.I. Seleznov, student,
A.M. Tara, student.

Architecture of model of the united algorithmic environment (MUAE) for solving computational geometry problems

Taras Shevchenko National University of Kyiv, 83000, Kyiv, Glushkova st., 4d
e-mail: v_ter@ukr.net
e-mail: bodyaosadchiy@gmail.com
e-mail: mariapetlovana@gmail.com
e-mail: anatoliy.seleznyov@gmail.com
e-mail: derubioo@gmail.com

В статті розглядається архітектура моделі єдиного алгоритмічного середовища (МЄАС) для розв'язання задач обчислювальної геометрії та запропонована її практична реалізація.

Ключові слова: МЄАС, метод "розділяй та пануй".

Architecture of model of the united algorithmic environment (MUAE) for solving computational geometry problems is considered in the article. Implementation of MUAE is proposed. Implementation description includes: problem statement, architecture requirements, description of chosen approach, which satisfies all key features of MUEA. Description of the universal data structure, which is used to maintain all final and intermediate results for all algorithms, is provided. "Divide and conquer" method is a base approach for considered algorithms. In order to algorithms used functions and primitives only from the united base of geometric functions, a separate geometric library was created. Key feature of the implementation is that every algorithm has its own merge method, which connects two already processed parts with possible usage of intermediate results of other algorithms. Algorithm dependency graph that used to verify, if the given algorithm computational sequence is valid, is described.

Key Words: MUAE, "Divide and Conquer" method.

Статтю представив д. ф. -м. н., проф. Анісімов А.В.

1. Вступ

На сьогодні існує безліч окремих ефективних алгоритмів для розв'язання основних класів задач обчислювальної геометрії. В той же час, при розв'язанні задачі візуального моделювання, виникає необхідність у використанні цілої сукупності алгоритмів, які, зазвичай, пов'язані між собою: результати роботи одного алгоритму є вхідними даними для іншого. Якщо кожен алгоритм з цієї сукупності реалізований окремо, то загальна ефективність розв'язання загальної задачі зменшується , а саме:

1) на етапах попередньої обробки даних і виконання алгоритмів дублювання одних і тих же операцій для кожного алгоритму (наприклад, сортування, побудова дерева розбиття, створення необхідних структур даних) суттєво уповільнює загальний час виконання;
2) використання структур даних для кожного з алгоритмів збільшує об'єм загальної пам'яті і додатковий час для доступу до елементів даних;

3) конвертація даних під необхідний формат, яка може призвести до втрати точності обчислень і теж уповільнює час виконання.

Таким чином, постає питання розробки єдиного алгоритмічного середовища для ефективного розв'язання сукупності пов'язаних між собою задач обчислювальної геометрії, яке позбавлено цих проблем.

В роботі [1] була запропонована базова модель єдиного алгоритмічного середовища (МЄАС), яка вирішує поставлену задачу. Перевагою цієї моделі є те, що вона підтримує паралельне виконання алгоритмів, які базуються на методі «розділяй та пануй». В роботах [2,3] основна увага приділяється математичному опису моделі та реалізації паралелізму, в той час як архітектурне рішення самої моделі залишається відкритим питанням.

В цій роботі пропонується можливий варіант архітектури МЄАС, оснований на ідеях моделі [4], яка вирішує такі проблеми:

1) Масштабованість моделі на велику кількість алгоритмів;
2) Автоматизована перевірка середовища на коректність;
3) Забезпечення гнучкості программного коду до змін.

Постановка задачі. Нехай задано множину S з N точок у просторі R^2. Розробити архітектуру єдиного алгоритмічного середовища для ефективної реалізації алгоритмів сукупності задач, що визначені на множині S, методом «розділяй та пануй» з нижньою оцінкою складності $\Omega(N \log N)$, яка буде розв'язувати перераховані вище проблеми.

2. Проектування архітектури

Зважаючи на ті задачі, які має вирішувати реалізація МЄАС, на етапі проектування були сформульовані такі основні вимоги до архітектури середовища:

1. Підтримка сукупності окремих геометричних алгоритмів, які можна виконувати на довільних вхідних даних в довільному порядку. Кожен алгоритм в середовищі має реалізовувати метод «розділяй та пануй». Алгоритми можуть використовувати результати роботи інших алгоритмів середовища, тобто є залежними від них.
2. Масштабованість за кількістю алгоритмів. Додавання або вилучення нового алгоритму з середовища потребує мінімальних та інтуїтивних змін коду.
3. Забезпечення коректності середовища: відсутність циклічних залежностей між алгоритмами, перевірка того, що всі залежні алгоритми дійсно присутні в середовищі.
4. Архітектура має бути побудована таким чином, щоб кількість спільного коду, структур даних, функцій була максимальна. Це забезпечує легкість в підтримці коду середовища, внесенні змін, а також зменшує час виконання алгоритмів.
5. Попередня обробка даних повинна мінімізувати кількість однакових дій для різних алгоритмів (наприклад, сортування, розбиття вхідних даних тощо).
6. Кожен алгоритм має виконуватись тільки в разі необхідності та не більше одного разу на одних і тих самих вхідних даних.
7. Забезпечення візуалізації результатів роботи кожного алгоритму.

3. Реалізація МЄАС

Запропонована реалізація відповідає всім сформульованим вище вимогам. Діаграми, які описують архітектурне рішення наведені на рис.1-4.

3.1. Попередня обробка та дерево розбиття

На кроці попередньої обробки відбувається зчитування вхідних даних, сортування та розбиття для методу "розділяй та пануй". Розбиття вхідних точок зберігається у вигляді дерева, де кожен вузол відповідає деякій множині точок, і будується лише один раз на початку роботи. Дерево розбиття реалізується класом DACTree (Divide and Conquer tree). У вузлах цього дерева, за які відповідає клас DACNode (рис.1), зберігаються результати роботи кожного з алгоритмів на відповідній множині точок в полі data. Клас дерева розбиття DACTree (рис.2) також підтримує список усіх алгоритмів, які можна виконати на цьому розбитті вхідних даних – AlgorithmContainer (див. пункт 3.2).

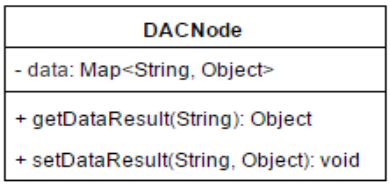

Рис.1. Вузол DACNode дерева розбиття DACTree.

Згідно з вимогами 1, 6 алгоритми можуть виконуватись в довільному порядку, але можуть залежати один від одного. Тому виникає необхідність перевірки, чи є певний алгоритм вже виконаним. Для цього в дереві для кожного алгоритму зберігаються прапорці PROCESSED, NOT_PROCESSED. Відмітимо, що алгоритм не може бути порахованим частково (це порушило б цілісність структур даних), тобто він завжди є або виконаним для всього дерева, або невиконаним для всього дерева.

Виконання алгоритмів відбувається за допомогою функції processAlgorithm, яка спочатку виконує всі алгоритми, від яких залежить поточний алгоритм (якщо вони ще не виконані), і тільки потім виконує поточний.

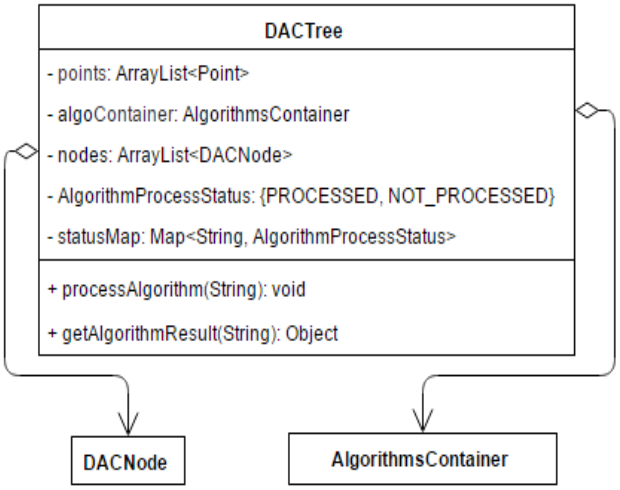

Рис.2. Дерево розбиття DACTree.

3.2. Представлення алгоритмів

Кожен алгоритм в середовищі представлений класом який реалізує сам алгоритм, та класом який представляє собою дані – результат роботи алгоритму (наприклад, діаграма Вороного представлена як реберний список з подвійними зв'язками).

Виходячи з пункту 1 пропонується мати абстрактний клас Algorithm (рис.3), в який винесені спільні для всіх алгоритмів функції. Оскільки усі вони реалізують метод "розділяй та пануй" і використовують спільне розбиття вхідних даних, то для кожного конкретного алгоритму достатньо описати тільки свою функцію merge(), яка реалізує крок злиття результатів, та свої функції обробки тривіальних випадків (зазначимо, що для кожного алгоритму тривіальною може бути різна кількість точок, яка задається константою TrivialCount).

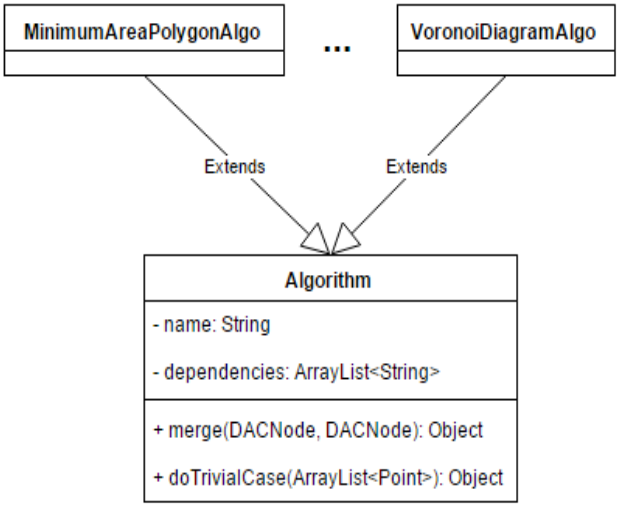

Рис.3. Абстрактний клас Algorithm.

Функція merge() приймає DACNode-и синів в дереві розбиття, що дозволяє отримувати результати роботи алгоритмів, від яких залежить алгоритм, що виконується, і повертає об'єкт даних алгоритму. Окрім цього, клас Algorithm зберігає список тих алгоритмів, від яких він залежить.

Вимога пункту 2 про масштабованість реалізується через клас AlgorithmContainer – контейнер для алгоритмів, через який відбувається додавання та вилучення алгоритмів. Ці операції можуть порушити коректність середовища, тому що алгоритми є залежними один від одного. Для того, щоб автоматизувати перевірку залежностей між алгоритмами (пункт 3 вимог), використовується шаблон проектування Builder. Клас AlgortihmContainerBuilder (рис.4) дозволяє "будувати" об'єкт AlgorithmContainer (який використовується для додавання та видалення алгоритмів), а перед тим, як повернути готовий об'єкт, виконує перевірку середовища (вимога 3).

Перевірка середовища здійснюється шляхом побудови орієнтованого графу залежностей алгоритмів. За допомогою обходу графу в глибину перевіряється, чи не містить граф

невідомих для середовища алгоритмів, а також чи немає циклічних залежностей між алгоритмами.

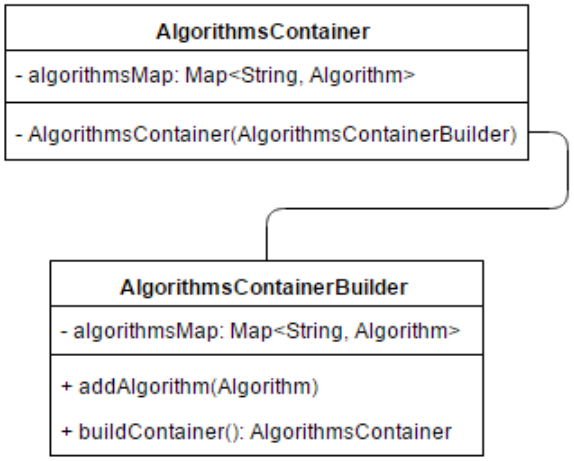

Рис.4. Класи AlgorithmContainer та AlgortihmContainerBuilder.

3.3. Спільна геометрична бібліотека

Для того, щоб алгоритми в середовищі працювали з єдиною базою геометричних примітивів та функцій, було створено окрему геометричну бібліотеку. Це вирішує питання перетворення даних від одного представлення в інше, а також спрощує визначення обчислювальної точності та похибки алгоритмів. Бібліотека побудована таким чином, що алгоритми можна писати на високому рівні абстрактності, оперуючи звичайними геометричними функціями та об'єктами – наприклад, знайти точку перетину між прямою та сегментом. Також в бібліотеці є складені об'єкти, як, наприклад, планарне розбиття у вигляді РСПЗ, яке може використовуватись алгоритмами незалежно один від одного.

4. Список реалізованих алгоритмів

Розроблене за запропонованою архітектурою середовище підтримує ряд алгоритмів, що базуються на методі «розділяй та пануй»:

1) *Побудова опуклої оболонки* [5];
2) *Побудова діаграми Вороного* [6,7];
3) *Пошук усіх найближчих сусідів* [2,3,6];
4) *Побудова Евклідового мінімального кістякового дерева* [8-10];
5) *Пошук простого охоплюючого багатокутника мінімальної площі* [11-16]. Для цього алгоритму була запропонована функція злиття, яка базується на пошуці суміжного простого багатокутника мінімальної площі.

Таким чином, граф залежностей алгоритмів, який підтримується середовищем, буде мати вигляд, що зображено на рис.5.

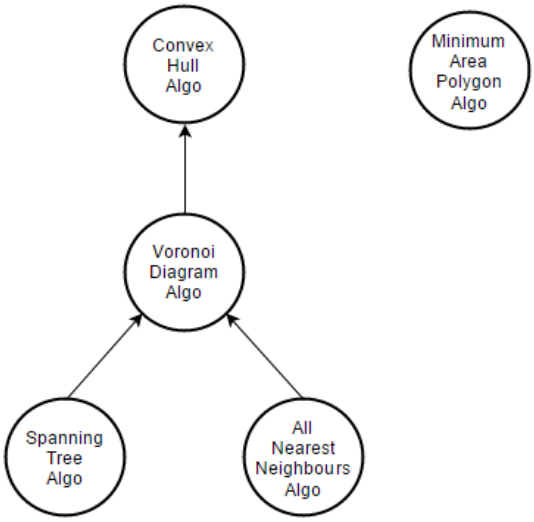

Рис.5. Граф залежностей алгоритмів.

5. Висновок

У статті запропоновано архітектуру та реалізацію моделі єдиного алгоритмічного середовища, яка має єдину універсальну структуру даних для задач обчислювальної геометрії, що розв’язуються методом «розділяй та пануй». Таким чином, можна уникнути окремих структур даних для кожного алгоритму, а отже зайвого споживання пам'яті. Необхідність у процедурах попередньої обробки та рекурсивного спуску для кожного алгоритму зникає, оскільки виконується лише одна процедура, яка готує необхідну структуру вхідних даних для алгоритмів. Отже, це зменшує час виконання алгоритмів.

Також варто зауважити, що наведена в цій роботі реалізація дуже добре масштабується на велику кількість алгоритмів, які можуть бути імплементовані в цьому середовищі. При цьому програмний код залишається гнучким до змін. Відмітимо, що інваріантом даного середовища є те, що дерево залежностей алгоритмів буде коректним після кожної операції вставки або видалення алгоритму. Перевірка на коректність відбувається автоматично.

Список використаних джерел

1. *Терещенко В.М.* Побудова єдиного алгоритмічного середовища для розв'язування комплексу задач обчислювальної геометрії: автореф. дис. канд. техн. наук: 01.05.01/ Терещенко В. М. // – м. Київ, 2011. – 30 с

2. *Терещенко В.Н.* Подход к решению взаимосвязанных задач геометрического моделирования / В.Н. Терещенко // Штучний інтелект, 4'2009. – 2009. – с. 161-167.

3. *Tereshchenko V.N.* Recursion and parallel algorithms in geometric modeling problems / V.N. Tereshchenko, A.V. Anisimov // Cybernetics and Systems Analysis, Springer. Vol. 46(2). – 2010. – pp. 173-184.

4. *Tereshchenko V.N.* The Unified Algorithmic Platform for Solving Complex Problems of Computational Geometry / V. N. Tereshchenko, I. Budjak, A. Fisunenko // 12th International Conference. Lecture Notes in Computer Science, 7979. St. Petersburg, Russia, September 30 – October 4, 2013. Springer Berlin Heidelberg. – 2013. – pp. 424-428.

5. *Терещенко В.Н.* Построение обобщенной стратегии решения некоторых задач вычислительной геометрии / В.Н. Терещенко, В.В. Сапсай, И.С. Статкевич, А. Федоров // Параллельные вычислительные технологии. – Киев, 2009. – с. 736-737**.**

6. *Препарата Ф.* Вычислительная геометрия: введение / Ф. Препарата, М. Шеймос //. – пер. с англ., Москва: Мир. – 1989. – 478 с.

7. *Dobrin A.* A Review of Properties and Variations of Voronoi Diagrams / A. Dobrin // Whitman College, WA. – 2005. – pp. 2-10.

8. *Sedgewick R.* Minimum Spanning Tree lecture notes / R. Sedgewick, K. Wayne // Computer Science: Algorithms & Data Structures, Princeton, Princeton University Press. – 2007. – p. 226.

9. *Eades P.* The realization problem for Euclidean minimum spanning trees is NP-hard / P. Eades, S. Whitesides // Proceeding of the 10th ACM Symposium on Computational Geometry, New York, USA. – 1994. – pp. 49–56.

References

1. TERESCHENKO, V. (2011) Pobudova algoritmychnogo seredovysha dlya rozvyazuvannya compleksu zadach obchisluvalnoi geometrii (Doctoral dissertation). Retrieved from Library of Taras Shevchenko National University of Kyiv.

2. TERESCHENKO, V.N. (2009) Podhod k resheniyu vzaimosvyazannyih zadach geometricheskogo modelirovaniya. *Shtuchniy intelekt.* 4'2009. pp. 161-167.

3. TERESCHENKO, V.N. and ANISIMOV, A.V. (2010) Recursion and parallel algorithms in geometric modeling problems. *Cybernetics and Systems Analysis*. 46(2). pp. 173-184.

4. TERESHCHENKO, V.N., BUDJAK, I. and FISUNENKO, A. (2013) The Unified Algorithmic Platform for Solving Complex Problems of Computational Geometry. In *12th International Conference.* Lecture Notes in Computer Science, 7979. St. Petersburg, Russia, September 30 – October 4, 2013. Springer Berlin Heidelberg. pp. 424-428.

5. TERESHCHENKO, V.N. et al. (2009) Postroenie obobshchennoi strategii reshenia zadach vychislitelnoi geometrii. *Parallel computational technologies*. Kyiv. pp. 736-737.

6. PREPARATA, F. and SHAMOS, M. (1989) Computational geometry: an introduction. – translation, Moscow: Mir.

7. DOBRIN, A. (2005) A Review of Properties and Variations of Voronoi Diagrams. Whitman College, WA. pp. 2-10.

8. SEDGEWICK, R. and WAYNE, K. (2007) Minimum Spanning Tree lecture notes. *Computer Science: Algorithms & Data Structures*. Princeton: Princeton University Press. p. 226.

9. EADES, P. and WHITESIDES, S. (1994) The realization problem for Euclidean minimum spanning trees is NP-hard. In *Proceeding of the 10th ACM Symposium on Computational Geometry*. New York, USA. ACM. pp. 49–56.

10. FORTUNE, S. (1992) Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean geometry*. 1. pp. 193-233.

10. *Fortune S.* Voronoi diagrams and Delaunay triangulations / *S.* Fortune // Computing in Euclidean Geometry. – 1992. – c. 193-233.

11. *Chong Z.* Generating random polygons with given vertices. / Z. Chong, S. Gopalakrishnan, J. Snoeyink, J. S. B. Mitchell // Computational Geometry: Theory and Applications, Elsevier. Vol. 6, Issue 5. – 1996. – pp. 277-290.

12. *Melkemi M.* Computing the shape of a planar points set / M. Melkemi, M. Djebali // Pattern Recognition, Elsevier. Vol. 33(9). – 2000. – pp. 1423–1436.

13. *Garai G.* A split and merge procedure for polygonal border detection of dot pattern / G. Garai, B. Chaudhuri // Image and Vision Computing, Elsevier. Vol. 17(1). – 1999. – pp. 75–82.

14. *Auer T.* Heuristics for the generation of random polygons / T. Auer, M. Held // Proceeding. of the 8th Canadian Conference on Computational Geometry, August 12-15, 1996, Carleton University, Ottawa, Canada. – 1996. – pp. 38-43.

15. *Galton A.* What is the region occupied by a set of points? / A. Galton, M. Duckham // Geographic Information Science, 4th International Conference, Proceeding, ser. Lecture Notes in Computer Science, September 20-23, 2006, Munster, Germany. – 2006. – pp. 81-98.

16. *Tereshchenko V.* Constructing a Simple Polygonalizations / V. Tereshchenko, V. Muravitskiy // World Academy of Science, Engineering and Technology, International Scholarly and Scientific Research & Innovation, Vol. 5(5), June 24-26, 2011, Paris, France. – 2011. – pp. 563-566.

11. CHONG, Z. et al. (1996) Generating random polygons with given vertices. *Computational Geometry: Theory and Applications.* 6(5). pp. 277-290.

12. MELKEMI, M. and DJEBALI, M. (2000) Computing the shape of a planar points set. *Pattern Recognition.* 33(9). pp. 1423–1436.

13. GARAI, G. and CHAUDHURI, B. (1999) A split and merge procedure for polygonal border detection of dot pattern. *Image and Vision Computing.* 17(1). pp. 75-82.

14. AUER, T. and HELD, M. (1996) Heuristics for the generation of random polygons. In *Proceeding. of the 8th Canadian Conference on Computational Geometry*. Ottawa, August 12-15, 1996. Ottawa: Carleton University Press. pp. 38-43.

15. GALTON, A. and DUCKHAM, M. (2006) What is the region occupied by a set of points? In *Geographic Information Science, 4th International Conference, Proceeding, ser. Lecture Notes in Computer Science*. Munster, September 20-23, 2006. Springer. pp. 81-98.

16. TERESCHENKO, V. and MURAVITSKIY, V. (2011) Constructing a Simple Polygonalizations. In *World Academy of Science, Engineering and Technology, International Scholarly and Scientific Research & Innovation.* 5(5). Paris, June 24-26, 2011. pp. 563-566.

Надійшла до редколегії 30.04.2015