УДК 519.68

Терещенко В.М.[1], д.ф.-м.н., доц.

Оптимальний алгоритм локалізації точки для тривимірного простору

Робота присвячена дослідженню задачі локалізації точки у R^3. Запропонований алгоритм розв'язання задачі базується на модифікованому методі смуг і дозволяє виділити грані що локалізують задану точку за час O(logN) з використанням $O(N^3)$ пам'яті.
Ключові слова: локалізація точки, запитна точка, 3-вимірний простір, смуга, дерево, структура даних.

[1] Київський національний університет імені Тараса Шевченка,03680, м. Київ, пр-т. Глушкова 4д, e-mail: vtereshch@gmail.com

V.N. Tereshchenko[1] , Dr. Sci.(Phis.-Math.), As. Prof.

The point-location problem for three-dimensional space

The paper is devoted research the point-location problem in Euclidean 3-dimensional space. We propose an algorithm for solving the problem, which is based on a modified slab method and allows to find facets that localize the point in O (logN) time using O (N^3) memory.

Key Words: point-location, 3-dimensional space, query point, a 3-dimensional space, slab, tree, data structures.

[1] Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova av., 4d, e-mail: vtereshch@gmail.com

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

1. Вступ

Постановка проблеми. Проблема локалізації точки є однією з основних тем обчислювальної геометрії. Локалізація точки застосовується в сферах, які пов'язані з обробкою геометричних даних, для розрахунку фізичних процесів: механіка, термодинаміка тощо. Швидка локалізація точки може пришвидшити роботу існуючих алгоритмів зменшивши кількість даних та об'єктів які беруть участь у обрахунках. Це можливо завдяки відсіканню тих об'єктів які знаходяться за межами області інтересів. Також локалізація точки може використовуватись у картографічних алгоритмах. Багатовимірна локалізація точки може використовуватись у алгоритмах штучного інтелекту.

Основне питання для комп'ютерних додатків, які використовують геометричні структури (наприклад, для комп'ютерної графіки, географічні інформаційні системи, робототехніка, і бази даних) звучить так: "Де я?". Враховуючи безліч незалежних геометричних об'єктів, ставиться питання знайти об'єкт, що містить точки запиту. Проблеми локалізації розрізняються по розмірності і типу об'єкта, чи є набір початкових даних статичний або динамічний. Рішення розрізняються по часу попередньої обробки, об'єму пам'яті, і часу виконання запиту [3].

Аналіз останніх досліджень. На сьогоднішній день існує багато алгоритмів розв'язання задачі локалізації в просторі з розмірністю менше 3 [2]. Одним із найбільш відомим є "метод деталізації тріангуляції" Кіркпатріка [6]. Ця задача може вважатися повністю розв'язаною, але задача для $d > 2$ викликає багато питань і не існує єдиного відомого чіткого алгоритму розв'язання [7]. Для простору розмірності 2 ефективними є наступні алгоритми: метод ланцюгів [2], метод смуг [2], *persistent search trees* [4]. В таблиці 1 подано основні результати розв'язання задачі локалізації точки на сьогодні.

D.P. Dobkin та R.J. Lipton [1] розв'язують проблему локалізації точки в багатовимірному просторі шляхом рекурсивного проектування d вимірної задачі на d-2 або d-1 вимірну. Час пошуку $O(d\log N)$, а пам'ять $O(N^{2d})$.

Таблиця 1

Основні результати задачі локалізації точки

Назва методу	Розмірність простору	Запит	Перед-обробка	Пам'ять
Метод ланцюгів	2	$O(log^2 N)$	$O(N \log N)$	$O(N)$
Метод смуг	2	O(logN)	$O(N^2)$	$O(N^2)$
persistent search trees [4]	2	$O(\log n)$	$O(n \log n)$	$O(n)$
D.P. Dobkin та R.J. Lipton [1]	d	O(d logN)		$O(N^{2\wedge d})$

Новизна та ідея. Запропонована реконструкція методу смуг для тривимірного простору, яка дозволяє одержати $O(\log N)$ час пошуку та $O(N^3)$ пам'яті, що покращує існуючий результат [1].

2 Постановка задачі та метод розв'язання

Постановка задачі. *Задано розбиття геометричного простору N вершинним прямолінійним графом на області, а точка запиту K. Необхідно локалізувати точку на заданому розбитті.*

В даному випадку простором є тривимірний евклідовий простір, в якому задано N точок і множина граней $O(N)$, що побудовані на цих точках. Необхідно для заданої точки знайти грані що її обмежують.

Розглянемо наступну схему побудови алгоритму локалізації точки для масових запитів:

1. *Попередня обробка даних.* Для попередньої обробки важливо побудувати такі структури даних, що дозволять значно скоротити час запиту на локалізацію точки. Будь-яка обробка даних, що виконуються щоразу під час запиту і конкретно не залежать від неї мають бути винесені в передобробку.
2. *Структура даних.* Необхідно побудувати такі структури даних, в яких будуть зберігатися результати попередньої обробки. Враховуючи те що метою є логарифмічний час пошуку, нам необхідно намагатися будувати деревовидні структури. Також при цьому необхідно звертати увагу на об'єм пам'яті що використовується.
3. *Пошук.* Основним критерієм є час пошуку – локалізації точки. Тобто необхідний алгоритм який використовуючи результат попередньої обробки за мінімальний час знаходить усі грані що локалізують точку.

2.1 Алгоритм попередньої обробки

Основна ідея алгоритму полягає в тому щоб розбити весь простір на обмежені підпростори в яких є структурована інформація про грані графа.

1. На першому кроці розбиваємо простір на підпростори паралельними площинами (рис.1).

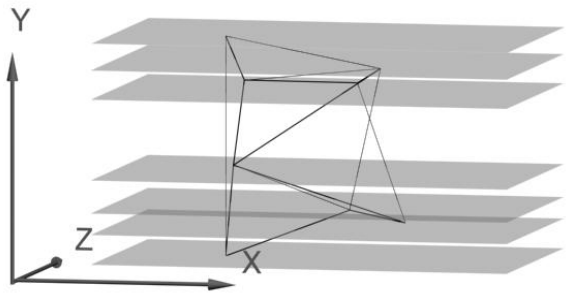

Рис1. Площини, що розрізають граф на шари.

Таким чином ми маємо підрозбиття на N-1 шарів. Впорядкувавши вершини розбиття, ми можемо знайти шар який містить запитну точку за час $O(\log N)$.

2. На другому кроці розбиваємо площини на смуги і визначаємо грані що їх перетинають. В результаті перетину граней одержимо області, які нагадують призми (рис.2).

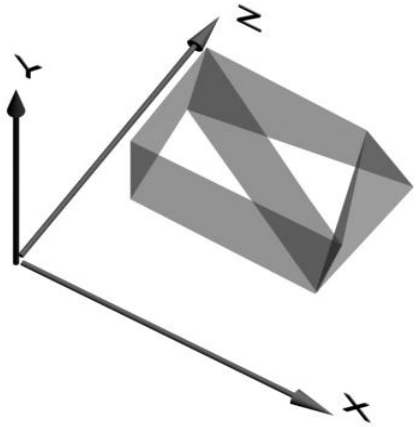

Рис.2. Вигляд одного з шарів.

Таким чином можна провести площини паралельні кожній з осей і через кожну точку розбиття (див. рис. 3).

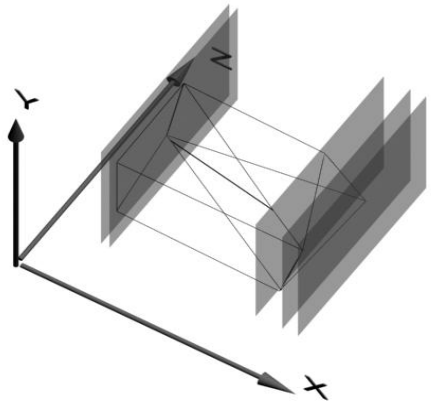

Рис.3 Побудова стержнів для шару

В результаті ми розіб'ємо шари на стержні (див рис. 4). Впорядкувавши стержні можемо знайти стержень, який містить запитну точку.

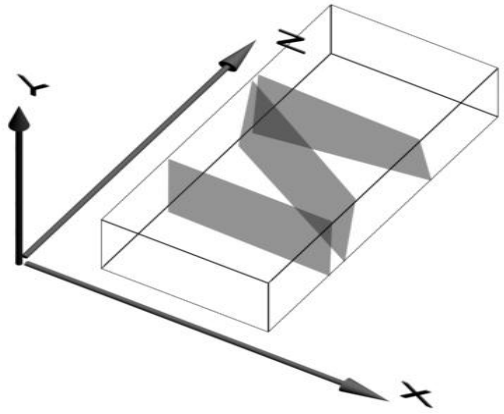

Рис.4. Остаточне розбиття кожного шару стержнями.

3. На третьому кроці будуємо дерево граней в кожній смузі. Нижче приведений детальний алгоритм.

Алгоритм:

1. Сортуємо усі задані точки простору по *y* координаті. І через кожну точку простору будуємо площину паралельну OXZ. Будуємо дерево площин.
2. Шукаємо перетин побудованих в п.2 площин з усіма гранями, які її перетинають. В перетині буде або точка або пряма. Нехай пряма на площині задається двома точками.
3. В кожній побудованій площині сортуємо точки по *z* координаті і через кожну точку на ній проводимо лінії паралельні осі OX. Отримуємо смуги. Будуємо дерево смуг для кожної площини.
4. В кожній смузі визначаємо ребра які її перетинають. Для кожного ребра відома початкова грань до якої воно належить. Сортуємо ці грані по *x* координаті середньої точки *O* і будуємо дерево граней. Крок перетину площин очевидний.

2.2 Структура даних

Данні про точки і грані зберігаються у двох двозв'язних списках. Всі інші надбудови є вказівниками на елементи цього списку. Дерево зберігається як впорядкований масив. Кожна площина містить смуги, що також зберігаються у вигляді бінарного дерева, а для кожної смуги є бінарне дерево граней. Тобто три вкладених бінарних дерева. Дерево площин впорядковане за зростанням координати *y*, дерево смуг впорядковане за координатою *z*, а дерево граней за *x* координатою середньої точки грані.

2.3 Алгоритм пошуку

Оскільки в результаті передобробки ми одержали структуру даних, яка складається з трьох дерев, то алгоритм пошуку не викликає труднощів.

Алгоритм:

1. Дана точка *K*(x,y,z). По дереву площин шукаємо дві площини що

обмежують дану точку по *y* координаті. Нехай ці площини *S*1, *S*2.

2. За допомогою дерева смуг шукаємо в площині *S*1 смугу *P* в яку проектується точка *K* по координаті *z*.

3. За допомогою дерева граней в смузі *P* шукаємо грані що обмежують задану точку. Отримаємо множину граней *bound*(*S*1).

4. Кроки 2 і 3 повторюються для площини *S*2. Отримуємо множину граней *bound*(*S*2).

5. Результатом буде множина *bound* = *bound*(*S*1) ∩ *bound*(*S*2).

3 Обґрунтування складності

Теорема 1. Задачу локалізації точки у тривимірному прямолінійному графі можна розв'язати за час $O(\log N)$ з використанням попередньої обробки $O(N^3 \log N)$.

Доведення. Для локалізації точки за зазначеним вище алгоритмом необхідно послідовно зробити пошуки в п'ятьох бінарних деревах. Перевірка вузла дерева O(1). Час пошуку в дереві $O(\log N)$, якщо дерев 5 то $O(5\log N)$. Тобто час пошуку дійсно $O(\log N)$. Час попередньої обробки $O(N^3 \log N)$ ґрунтується на алгоритмі попередньої обробки. Так як для даного алгоритму треба зберігати 3 вкладені дерева то пам'ять яка використовується $O(N^3)$.

4 Практична частина

Для реалізації алгоритму розроблена програма мові C# для WPF (Windows Presentation Foundation). WPF – графічний фреймворк для рендерінгу користувацьких інтерфейсів під Windows. Користувачу програми надано можливість задавати довільно точку, під час виконання програми, а також редагувати вхідний файл з трьохвимірним графом. На рис. 5. Подано один із прикладів реалізації алгоритму.

4. Висновки

У роботі запропоновано новий підхід і алгоритм до вирішення задачі локалізації точки в тривимірному просторі, обґрунтована складність і описані структури даних, що використовувалися для вирішення даної задачі.

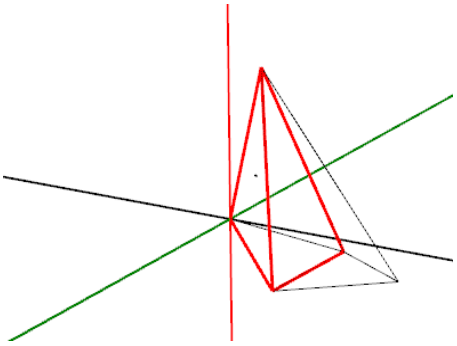

Рис.5. Приклад реалізації.

Описаний алгоритм є модифікацією методу смуг для двовимірного випадку. Досягнуто оптимальний час пошуку $O(\log N)$ з використанням $O(N^3)$ пам'яті.

Список використаних джерел

1. Tereshchenko, Vasyl' M.; Kas'yanov, Andrij A. On an approach to the solution of a local search problem // Visn., Ser. Fiz.-Mat. Nauky, Ky¨ıv. Univ. Im. Tarasa Shevchenka 2006, No. 3, 265-269.
2. D.P. Dobkin and R.J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5:181–186, 1976.
3. *Шеймос М., Препарата Ф.* Вычислительная геометрия: Введение. Пер. с англ. – М: Мир, 1989. – 478 с.
4. Handbook of discrete and computational geometry / edited by Jacob E. Goodman and Joseph O'Rourke. ISBN 1-58488-301-4
5. N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.
6. K. Mulmuley and S. Sen. Dynamic point location in arrangements ofh yperplanes. In p*roc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 132–141, 1991.
7. Kirkpatrik D.G. Optimal search in planar subdivisions // SIAM J. Comput. Б 12(1), 1983, pp. 28-35.
8. http://en.wikipedia.org/wiki/Point_location

Надійшла до редакції 28.12.12