Даниил Золотарев

A Dynamic Data Structure for 3-d Convex Hulls and 2-d Nearest Neighbor Queries

Содержание

1. Основные результаты

В статье предложена рандомизированная структура данных, поддерживающая динамическое множество точек в $\mathbb{R}^3$, с возможностью:

- вставки за $O(\log^3 n)$ (амортизированное время),
- удаления за $O(\log^6 n)$ (амортизированное время),
- запросов на экстремальную точку за $O(\log^2 n)$ (в худшем случае).

Это первая структура с полилогарифмическими временными оценками как на вставку, так и на удаление.

2. Ключевые идеи

Используется двойственность: точки $\leftrightarrow$ плоскости. Структура строится для задачи *vertical ray shooting* по нижнему контуру множества плоскости, то есть будет поддерживаться динамическое множество H, состоящее из n плоскостей в трёхмерном пространстве, так что для заданной вертикальной прямой q можно быстро определить нижнюю плоскость, проходящую через q, обозначаемую как $ans(H, q)$.

3. Определения

Пусть задано множество H, состоящее из n плоскостей, и область γ. $(1/r)$-*разбиением* области γ называется набор непересекающихся открытых ячеек (тетраэдров) такой, что объединение замыканий этих ячеек содержит γ, и каждая ячейка пересекается не более чем с $\lceil n/r \rceil$ плоскостями из H.

Списком конфликтов ячейки Δ называется подмножество всех плоскостей из H, пересекающих Δ. Он обозначается H_Δ.

Уровнем $\leq k$ называют замыкание множества всех точек q, таких что строго ниже q находятся не более k плоскостей из H.

Нам понадобится следующая лемма.

Лемма 1. Пусть H — множество из n плоскостей и $r > 0$. Существует $O(1/r)$-разбиение уровня $\leq n/r$ множества H на $O(r)$ ячеек. Это разбиение, а также списки конфликтов всех ячеек, можно построить за $O(n \log n)$ в среднем.

Ячейки являются (неограниченными) тетраэдрами и называются вертикальными в том смысле, что каждая из них содержит точку $(0{,}0, -\infty)$.

Данный результат из вычислительной геометрии считаем доказанным.

Сначала бы построим подструктуру и определим операции с ней, а затем уже из подструктур сложим основную нашу динамическую структуру.

4. Подструктура

Каждая подструктура включает множество S из $\log n$ слоёв, состоящих их разрезов, которые строятся с помощью применения Леммы 1. Вводятся так же две категории плоскостей:

- **Живые плоскости** S_{live} — кандидаты на ответ.
- **Статичные плоскости** S_{static} — служат для выполнения запросов.

4.1 Построение

Ниже приведён псевдокод построения подструктуры. Главная идея — как и сказано выше, всего изначально есть $\log n$ слоёв, каждый из которых получается фильтрацией предыдущего по новому "хорошему" разрезу соответсвующего уровня, то есть удалением плоскостей предыдущего слоя, которые создают слишком большие конфликты (в смысле списков конфликтов ячеек). Так же вычисляются списки конфликтов для всех уровней. В конце определяются множества S_{live} и S_{static}.

construct(S):
  $S_0 := S$
  for $i = 1 \dots \log n$:
    $T_i :=$ $(1/2^i)$-разрез уровня $\leq \frac{|S_{i-1}|}{c2^i}$ на $c'2^i$ вертикальных клеток
    $S_i := S_{i-1} \setminus \{$плоскости, пересекающие $> 4c' \log n$ клеток из $T_i\}$
    for each ячейка Δ in T_i:
      вычислить список конфликтов $(S_i)_\Delta$
      задать $j_\Delta := 0$
  $S_{live} := S_{static} := S_{\log n}$

4.2 Удаление

При удалении какой-то плоскости те плоскости, который создают слишком много конфликтов, будут удаляться из живых.

delete(S, h):
  удалить h из S и S_{live} (если h был живым)
  for each i и Δ в T_i с $h \in (S_i)_\Delta$:
    $j_\Delta := j_\Delta + 1$
    if $j_\Delta = \lceil |(S_i)_\Delta| / c'' \rceil$:
      удалить все плоскости из $(S_i)_\Delta$ из S_{live}

4.3 Dead planes tell no tales

Приведём результат, обосновывающий введение множеств S_{live} и S_{static}.

Лемма 2. (a) Вызов construct() занимает $O(n \log^2 n)$ ожидаемого времени и помещает по меньшей мере $n/2$ плоскостей в S_{live}.
(b) Последующие n_D операций delete() требуют $O(n_D \log^2 n)$ времени и удаляют не более $O(n_D \log^2 n)$ плоскостей из S_{live}.
(c) Для любой вертикальной прямой q, если $ans(S, q) \in S_{live}$, то $ans(S, q) = ans(S_{static}, q)$.

Пункт (c) говорит, что если искомая плоскость живая, то ответ можно искать в множестве S_{static}, что мы и будем делать. S_{static} – это не меняющееся множество, поэтому для него применимо использования структур, про которые рассказывали мои коллеги (Planar Point Location Problem).

5. Основная структура

Теперь мы готовы представить нашу структуру данных, поддерживающую как вставки, так и удаления для динамического множества H плоскостей.

Мы поддерживаем набор из $O(\log n)$ подмножеств S, каждое из которых хранится в построенной подструктуре. В любой момент времени поддерживается инвариант: подмножества

S_{live} попарно непересекающиеся и их объединение равно H, то есть $\bigcup S_{live} = H$. Однако сами подмножества S (и, соответственно, S_{static}) не обязаны быть непересекающимися из-за наличия "мёртвых" плоскостей.

Для каждого подмножества S в коллекции мы храним множества S_{static}.

Начальная структура данных может быть построена путём многократного (логарифмическое число раз) вызова процедуры построения до тех пор, пока каждая плоскость не станет "живой" хотя бы в одном из подмножеств, как того требует инвариант.

5.1 Построение

```
preprocess(S):
    if S ≠ ∅:
      добавить подмножество S
      construct(S)
      preprocess(S \ S_live)
```

5.2 Вставка

Чтобы вставить новую плоскость в множество H, мы просто добавляем в коллекцию новое подмножество размера 1.

Чтобы гарантировать, что общее число подмножеств остаётся логарифмическим, мы выполняем "очистку" — многократно объединяя подмножества схожего размера (и удаляя по пути все "мёртвые" плоскости). Для подмножества S определим его *глубину* как $\lfloor \log |S_{live}| \rfloor$. Положим $b = 16$.

```
insert(h):
  добавить новое подмножество S := {h} с S_live := S_static := S
  while существуют b подмножеств S^1, ..., S^b на одной глубине:
    удалить из структуры S^1, ..., S^b
    preprocess(S^1_live ∪ ... ∪ S^b_live)
```

После этапа очистки на строках 2–3 выше, на каждой глубине может остаться не более чем $b-1$ подмножеств. Следовательно, общее число подмножеств составляет $O(\log n)$, как и утверждалось.

5.3 Удаление

Чтобы удалить плоскость h из множества H, мы вызываем процедуру удаления из Раздела 3 для каждого подмножества S , содержащего h (таких подмножеств может быть несколько, поскольку могут существовать мёртвые копии h). В результате некоторые плоскости в S становятся мёртвыми. Ключевая идея заключается в том, чтобы повторно вставить эти новые мёртвые плоскости обратно в структуру данных (в другие подмножества коллекции), тем самым восстанавливая инвариант.

```
delete(h):
  A := ∅
  for each S, содержащий h:
    delete(S, h)
    добавить плоскости, только что удалённые из S_live, в A
  for each g ∈ A \ {h}:
    insert(g)
```

5.4 Запрос

Алгоритм обработки запроса прост, поскольку задача вертикального лучевого поиска *разложима*: ответ можно получить, выполнив запрос по каждому подмножеству коллекции отдельно.

query(q):
  $B := \varnothing$
  for each подмножество S, такое что $ans(S_{static}, q) \in S_{live}$:
    $B.add(ans(S_{static}, q))$
  return $ans(B, q)$

6. Главная теорема

Теперь мы готовы сформулировать основной результат про построенную структуру.

Теорема 1. *(a) Любая последовательность из n_I операций insert() и n_D операций delete() над начальным множеством размера n_0 требует в сумме ожидаемого времени обновления $O(n_0 \log^2 n + n_I \log^3 n + n_D \log^6 n)$, где n — максимальный размер множества в любой момент времени.*

(b) Вызов query() (поиск по вертикальному лучу) возвращает корректный ответ за $O(\log^2 n)$ времени.

7. Применение и выводы

Структура применима к:

- 2D-запросам ближайших соседей при помощи *отображения поднятия (lifting transformation)* точек на плоскости в пространство $\mathbb{R}^3$
- динамическим Евклидово минимальным остовным деревьям
- выпуклым оболочкам

и прочим задачам. Её можно адаптировать к неперпендикулярным запросам, линейному программированию и использованию с ограниченными ресурсами памяти.