УДК 519.683

А.В. СКВОРЦОВ

ПОСТРОЕНИЕ ТРИАНГУЛЯЦИИ ДЕЛОНЕ ЗА ЛИНЕЙНОЕ ВРЕМЯ

Предлагается ряд алгоритмов построения триангуляции Делоне с линейной в среднем трудоёмкостью. Предлагается два алгоритма полосового слияния (выпуклого и невыпуклого). Для алгоритмов слияния предлагается стратегия откладывания перестроений. Предлагаются алгоритмы статического и динамического кэширования поиска в итеративном алгоритме.

Введение

Задача построения триангуляции по заданному набору точек широко используется для аппроксимации различного рода физических поверхностей, разбиения плоскости на элементарные части в методах конечных элементов, учёта взаимного влияния смежных тел и в других физических задачах. Данная задача является неоднозначной, поэтому среди всех возможных триангуляций наиболее часто используется именно триангуляция Делоне, так как она обладает рядом оптимальных свойств.

Трудоёмкость (зависимость времени работы алгоритма от размера входных данных) задачи построения триангуляции Делоне составляет $O(N \log N)$, где N – количество исходных точек триангуляции [1,2]. В литературе описаны алгоритмы [1,2,3], имеющие указанную трудоёмкость в среднем и в худшем случаях, а также ряд других с худшими или непроанализированными характеристиками [4,5,6]. Исключение составляет алгоритм, основанный на клеточном разбиении множества, представленный в работе [7], имеющий в среднем на равномерном распределении точек линейную трудоёмкость, но с большой константой пропорциональности.

Практически все существующие алгоритмы построения триангуляции Делоне можно условно разделить на две основные группы: алгоритмы слияния и итеративные алгоритмы.

Концептуально алгоритмы слияния предполагают разбиение исходного множества точек на несколько подмножеств, построение триангуляций на этих подмножествах, а затем объединение (слияние) нескольких триангуляций в одно целое.

В соответствии со стратегией «Разделяй и властвуй» множество исходных точек разбивается с помощью прямой на два подмножества, незначительно отличающиеся по размеру. Алгоритм триангуляции рекурсивно применяется к частям, а затем производится сшивание полученных подтриангуляций. Один из алгоритмов, работающий в соответствии с этой стратегией, описан в [2].

В данной работе предлагается другой вариант алгоритмов слияния, который состоит в разбиении исходного множества точек на такие подмножества, чтобы построение по ним триангуляций имело минимальную трудоёмкость, например, за счёт применения специальных алгоритмов, оптимизированных для конкретных конфигураций точек.

Так, основная предлагаемая идея полосовых алгоритмов слияния предполагает разбиение всего исходного множества точек на полосы и применение быстрого алгоритма получения невыпуклой триангуляции полосы точек. После получения множества полосовых подтриангуляций они сшиваются, при этом, так как полосы невыпуклые, приходится

- либо достраивать полосы до выпуклых и затем использовать обычный алгоритм слияния из алгоритма «Разделяй и властвуй»;

- либо применять более сложный, но более эффективный алгоритм сшивания невыпуклых триангуляций.

Все итеративные алгоритмы имеют в своей основе очень простую идею последовательного добавления точек в частично построенную триангуляцию Делоне. Формально это выглядит так.

Пусть имеется триангуляция Делоне на множестве из (N-1) точек. Очередная N-я точка добавляется в уже построенную структуру триангуляции следующим образом:

1. Вначале производится локализация точки, т.е. находится треугольник (построенный ранее), в который попадает очередная точка; либо, если точка не попадает внутрь триангуляции, то находится треугольник на границе триангуляции, ближайший к очередной точке.

2. Если точка попала внутрь какого-нибудь треугольника, он разбивается на три новых; иначе, при попадании точки вне триангуляции, строится один или более треугольников. Затем проводятся локальные проверки вновь полученных треугольников на соответствие условию Делоне.

Трудоёмкость данного алгоритма складывается из трудоёмкости поиска треугольника, в который на очередном шаге добавляется точка, трудоёмкости построения новых треугольников, а также трудоёмкости соответствующих перестроений структуры триангуляции после проверок пар соседних треугольников полученной триангуляции в случае невыполнения условия Делоне.

Любое добавление новой точки в триангуляцию теоретически может нарушить целостность условия Делоне, поэтому после добавления точки обычно сразу же производится локальная проверка триангуляции на условие Делоне. Эта проверка должна охватить все вновь построенные треугольники и соседние с ними. Известно, что трудоёмкость таких перестроений в среднем составляет $O(1)$, т.е. число возможных перестроений пар соседних треугольников в среднем постоянно.

Таким образом, наибольший вклад в трудоёмкость итеративного алгоритма даёт процедура поиска очередного треугольника.

1. Алгоритм полосового слияния

Предлагаемые в данной работе алгоритмы полосового слияния логически состоят из трёх последовательных шагов.

1. Разбиение всего исходного множества точек на некоторые полосы.
2. Применение специального быстрого алгоритма получения невыпуклых триангуляций полученных полосовых подмножеств точек.
3. Слияние полученных триангуляций.

Рассмотри эти шаги подробнее.

Шаг 1. Разбиваем множество всех точек на некоторое количество столбцов по принципу их одинаковой ширины (с помощью цифровой сортировки целых чисел [8]) или одинакового количества точек (метод Хоара вычисления квантилей [8]). Количество точек в каждом столбце должно получиться не менее трёх (этого требует алгоритм, применяемый на шаге 2). Если это не выполняется для какого-либо столбца, то его необходимо присоединить к соседнему. Трудоёмкость данного шага составляет $O(N)$ в соответствии с трудоёмкостью применяемых алгоритмов разбиения.

Шаг 2. Сортируем все точки внутри столбцов по вертикали (по координате *Y*) и триангулируем столбец. Для этого предлагается следующий специальный алгоритм триангуляции (рис. 1). Вначале на трёх самых верхних точках в столбце строится первый треугольник, и он помечается как текущий. Теперь последовательно перебираем все остальные точки в столбце, начиная с четвёртой, сверху вниз и добавляем их к частичной триангуляции. Пусть ΔABC является текущим, где точка B имеет наименьшую из точек треугольника координату, а X – очередная добавляемая точка из столбца. На очередном шаге необходимо сделать выбор очередного создаваемого треугольника ΔABX или ΔBCX . Если один из этих треугольников построить невозможно ввиду получаемых тогда пересечений различных отрезков триангуляции, то выбор однозначен. Если же можно построить оба треугольника, то естественно выбрать тот, у которого минимальный из углов больше, т.к. тогда с большей вероятностью в будущем не придётся его перестраивать из-за невыполнения условия Делоне.

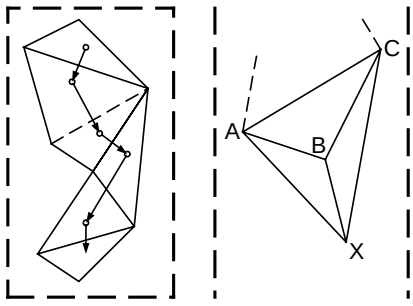

Рис. 1. Триангуляция полосы точек

После построения очередного треугольника надо проверить условие Делоне для вновь образовавшихся пар соседних треугольников и при необходимости перестроить их.

Заметим, что при таком алгоритме будет построена *невыпуклая* триангуляция точек. Если предположить верной гипотезу об ограниченном количестве перестроений, то трудоёмкость данного шага будет линейной.

Для оптимального выбора числа полос, определяющего быстродействие алгоритма, сделаем упрощающее предположение о равномерном распределении точек. Тогда оптимальное число полос составит:

$$m = \sqrt{\frac{2aN}{3(3+q)b}}, \qquad (1)$$

где N – число исходных точек; a, b – ширина и высота триангулируемой области точек, а q – некоторая константа.

Так как в формулу (1) входит неизвестная константа, то для установления точного вида зависимости автором было проведено экспериментальное моделирование работы алгоритма на ряде распространённых распределений и найдено оптимальное значение $q = 2$, имеющее точность 0,01 с доверительной вероятностью 0,95.

2. Алгоритм выпуклого полосового слияния

Шаг 3. Вначале необходимо подготовить полученные полосовые триангуляции для алгоритма слияния, применяемого в алгоритме «Разделяй и властвуй», т.к. в этом алгоритме используется особенность, которая гарантирует правильность слияния только для выпуклых триангуляций. Для этого надо пройтись по границе невыпуклой триангуляции и построить выпуклую оболочку (рис. 2).

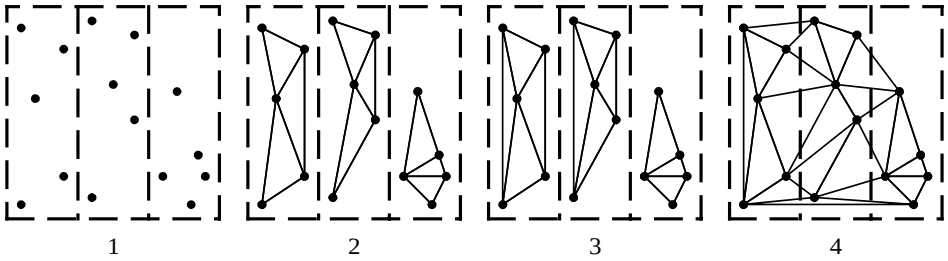

Рис. 2. Шаги работы алгоритма выпуклого полосового слияния

Шаг 4. Пробегаемся в цикле по столбцам и сшиваем их друг с другом, используя алгоритм слияния из алгоритма триангуляции «Разделяй и властвуй».

Данный алгоритм делает много лишней работы, так как при построении выпуклой оболочки узкой полосы обычно получаются длинные узкие треугольники, которые почти всегда приходится перестраивать при слиянии. Этот недостаток исправляется в алгоритме невыпуклого слияния.

3. Алгоритм невыпуклого полосового слияния

Шаг 3. На вход алгоритма невыпуклого слияния подаются невыпуклые полосовые триангуляции, и в ходе работы алгоритма слияния перед очередным построением сшивающего ребра и соответственно треугольника производится достраивание выпуклости на некотором расстоянии от сшивающего ребра (рис. 3). Рассмотрим это подробнее.

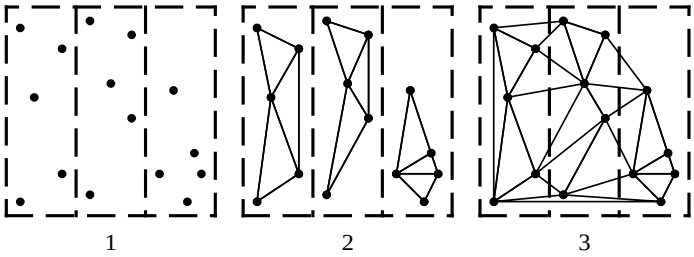

Рис. 3. Шаги работы алгоритма невыпуклого полосового слияния

Пусть на очередном шаге сшивания построен отрезок (P_1, P_2) (P_1 – на левой триангуляции, P_2 – на правой). Пусть N_1, N_2 – соседние точки по границам триангуляции (следующие ниже P_1, P_2). Обычно на очередном шаге мы выбираем, какой треугольник строить – $\Delta P_1 P_2 N_1$ или $\Delta P_1 P_2 N_2$ (рис. 4). В алгоритме выпуклого слияния существенно используется то свойство, что граница первой триангуляции выше N_2 (начиная с P_1 и ниже) и граница второй триангуляции выше N_1 (начиная с P_2 и ниже) выпуклы.

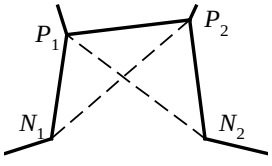

Рис. 4. Сшивание полос

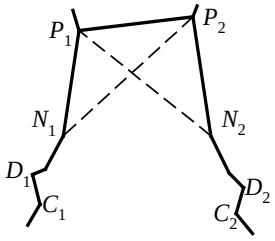

Рис. 5. Локальное достраивание выпуклости

В алгоритме невыпуклого слияния необходимо проанализировать выпуклость грани-

цы и определить первую точку D_1 (D_2), начиная с P_1 (P_2), где нарушается выпуклость, и запомнить следующую за ней точку C_1 (C_2). Тогда перед анализом, какой треугольник сшивания строить, проводится следующая проверка. Если C_1 (C_2) выше N_2 (N_1), то достраивается выпуклая оболочка на границе от C_1 до P_1 (от C_2 до P_2) и ищутся следующие точки *D* и *C*, если они существуют (рис. 5).

При таком подходе удаётся заметно уменьшить число перестроений, а, следовательно, и время работы алгоритма. В проведённом экспериментальном сравнении времени работы алгоритмов полосового слияния алгоритм невыпуклого слияния работал в среднем на 10% быстрее, чем алгоритм выпуклого слияния на различных наборах точек и распределениях.

4. Двухпроходные алгоритмы с откладыванием перестроений

При построении триангуляции Делоне каждый строящийся треугольник после построения должен быть проверен на выполнение условия Делоне для образующих с ним пары соседних треугольников. При этом приходится проводить проверки для трёх пар, соответствующих трём соседним треугольникам к данному. Если для какой-либо пары треугольников условие Делоне не выполняется, то должны последовать перестроения треугольников и новая серия проверок.

Предлагается упростить логику алгоритмов слияния следующим образом. Предлагается за первый проход построить некоторую триангуляцию, игнорируя выполнение условия Делоне. А после этого за второй проход проверить то, что получилось, и провести нужные улучшающие перестроения для приведения триангуляции к триангуляции Делоне.

Как показали проведённые эксперименты, алгоритмы слияния в силу своей специфики даже без проверки условия Делоне строят такие треугольники, на большей части которых выполняется условие Делоне.

Для итеративных алгоритмов двухпроходная стратегия не годится, т.к. сразу же образуются узкие длинные треугольники, которые в дальнейшем делятся на другие, ещё меньшие и ещё более узкие.

По результатам экспериментальных исследований установлено, что такая двухпроходная стратегия действительно даёт выигрыш в производительности для всех известных и новых алгоритмов слияния. За счёт значительного упрощения автору удалось реализовать алгоритмы, работающие в среднем на 10% процентов быстрее, чем исходные алгоритмы.

Теоретическая возможность работы такого алгоритма опирается на известные факты, что произвольную триангуляцию можно преобразовать посредством только улучшающих перестроений за конечное время в триангуляцию Делоне [2,9].

Трудоёмкость работы второго этапа стратегии составляет $O(N)$.

5. Итеративные алгоритмы триангуляции с кэшированием

Алгоритм кэширования поиска использует решение отдельной задачи локализации точки на планарном разбиении, не используя особенности конкретной задачи построения триангуляции. В литературе [1,8] описываются несколько оптимальных алгоритмов поиска, основанных на идее бинарного поиска и тратящих на поиск не менее $O(\log N)$ шагов.

Рассмотрим алгоритм кэширования поиска в общем случае.

Задача локализации точки. Дано планарное разбиение плоскости R_0 (в данном

случае триангуляция) и точка (x, y). Надо найти элемент планарного разбиения r_{0j} (треугольник), содержащий заданную точку.

Для решения этого вопроса предлагается построить некоторое более простое, чем R_0, планарное разбиение R_1 (например, регулярную сеть квадратов), покрывающее R_0, и в котором можно очень быстро производить локализацию точки. Затем сопоставим каждому элементу r_{1i} разбиения R_1 элемент r_{0j} разбиения R_0 так, чтобы элемент r_{0j} находился как можно ближе к r_{1i} (например, в смысле расстояний между центрами элементов разбиения). Тогда, при поступлении очередного запроса на поиск необходимо найти элемент разбиения r_{1i}, а по нему r_{0j}, который находится обычно довольно близко к искомому элементу разбиения. После этого за несколько шагов, передвигаясь по цепочке соседних элементов разбиения (треугольников), находится целевой $r_{0ц}$. После успешного окончания поиска мы можем изменить соответствие $r_{1i} \to r_{0j}$ на другое, более уместное для данной задачи, а именно на $r_{1i} \to r_{0ц}$.

Теперь рассмотрим, как лучше выбрать разбиение R_1. Для быстрого поиска удобнее всего разбить всю данную область, например, на регулярную сеть квадратов, проведя равноотстоящие горизонтальные и вертикальные прямые по всей области (рис. 6). Например, если данное планарное разбиение полностью покрывается квадратом $[0;1]\times[0;1]$, то его можно разбить на m^2 равных частей в виде квадратов. Занумеруем все квадраты естественным образом двумя параметрами $i, j \in [0; m-1] \subset \mathrm{N}$: r_{1ij}. Тогда по данной точке (x, y) мы мгновенно можем найти квадрат $r_{1(\lfloor x/m \rfloor, \lfloor y/m \rfloor)}$, где $\lfloor ... \rfloor$ – знак взятия целой части числа.

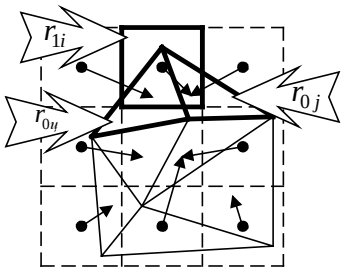

Рис. 6. Локализация точек

В предлагаемом алгоритме построения триангуляции Делоне методом статического кэширования необходимо выбрать некоторое число m и завести 2-мерный массив $r_{1([0;m-1],[0;m-1])}$ ссылок на треугольники. Первоначально этот массив надо заполнить ссылками на самый первый построенный треугольник. Потом, после выполнения очередного поиска, в котором был найден некоторый треугольник *T*, начиная поиск с квадрата (i, j), необходимо обновить информацию в кэше: r_{1ij} := ссылка на *T*.

Первое время, пока кэш не обновится полностью, поиск может идти относительно долго, но потом скорость повышается. Этого недостатка лишён алгоритм динамического кэширования.

В нём вначале предлагается завести кэш минимального размера, например 2×2. По мере роста числа добавленных в триангуляцию точек необходимо последовательно увеличивать его размер, например, в 2 раза, переписывая при этом информацию из старого кэша в новый. Например, для увеличения кэша в 2 раза надо выполнить следующие пересылки:

$$\forall i, j = \overline{0; m_{стар} - 1}: r_{1нов(2i,2j)}, r_{1нов(2i,2j+1)}, r_{1нов(2i+1,2j)}, r_{1нов(2i+1,2j+1)} := r_{1стар(i,j)}. \quad (2)$$

Такая модификация алгоритма кэширования позволяет алгоритму работать одинаково эффективно на маленьком и большом числе точек, заранее не зная их число.

Трудоёмкость полученных алгоритмов построения триангуляции Делоне с использо-

ванием алгоритмов кэширования составляет в худшем $O(N^2)$ (как для любого итеративного алгоритма), но в среднем для алгоритма статического кэширования $O(N^{9/8})$ при выборе $m \sim N^{3/8}$, для динамического кэширования – $O(N)$.

И в заключение можно отметить, что все полученные алгоритмы были экспериментально сравнены в работе на одинаковой программно-аппаратной платформе. При этом установлено, что новые алгоритмы значительно превышают по производительности все известные. Наиболее эффективным алгоритмом оказался алгоритм динамического кэширования, хотя ранее итеративные алгоритмы считались заведомо хуже всех алгоритмов слияния.

СПИСОК ЛИТЕРАТУРЫ

1. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение / Пер. с англ. – М.: Мир, 1989. – 478 с.
2. Lee D., Schachter B. // Int. Jour. Comp. and Inf. Sciences. – 1980. – V.9. – № 3. – P. 219-242.
3. Shapiro M. // Inter. Jour. Of Comp. and Inf. Sciences. – 1981. – V.10. – № 6. – P. 413-418.
4. Lawson C. // Mathematical Software. – 1977. – № 3. – P.161-194.
5. Sloan S., Houlsby G. // Adv. Eng. Software. –1984. – V.6. – № 4. – P. 192-197.
6. Toussaint G. // Proc. 5th Int. Conf. of Patt. Recogn, Miami Beach. – 1980. – P. 1324-1247.
7. Ильман В.М. // Алгоритмы и программы, ВИЭМС. – М., 1985. – Вып. 10 (88). – С. 3-35.
8. Ахо А., Хопкрофт Д., Ульман Д. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 520 с.
9. Ильман В.М. // Алгоритмы и программы, ВИЭМС. – М., 1985. – Вып. 10 (88). – С. 57-66.

Томский госуниверситет

Поступила в редакцию 10.12.98.