Раздел 1. Алгоритмы и структуры данных

ЭФФЕКТИВНЫЕ АЛГОРИТМЫ ПОСТРОЕНИЯ ТРИАНГУЛЯЦИИ ДЕЛОНЕ*

А.В. Скворцов, Ю.Л. Костюк

1. Введение

Задача построения триангуляции является одной из базовых в вычислительной геометрии на плоскости. К ней сводятся многие практические задачи геоинформатики. Среди всех возможных триангуляций именно триангуляция Делоне обладает рядом оптимальных свойств.

В настоящее время известно достаточно много алгоритмов и их программных реализаций для построения триангуляции.

Трудоёмкость задачи построения триангуляции Делоне составляет $O(N \log N)$ [1,2]. В литературе описаны алгоритмы [1-3], имеющие указанную трудоёмкость в среднем и в худшем случаях, а также ряд алгоритмов с худшими или не проанализированными характеристиками [4-6]. Исключение составляет алгоритм, основанный на клеточном разбиении множества, представленный в работе [7], имеющий в среднем на равномерном распределении точек линейную трудоёмкость, но с большей константой пропорциональности.

В данной работе предлагается ряд новых эффективных алгоритмов, а также модификации известных, позволяющих улучшить работу алгоритмов построения триангуляции Делоне на определённых наборах входных данных.

2. Структуры для представления триангуляции

Как показывает практика, выбор структуры для представления триангуляции оказывает существенное влияние на трудоёмкость алгоритмов, использующих данную структуру, а также на скорость конкретной реализации. В настоящее время наиболее распространённой является следующая структура, представляющая точки и треугольники [2,3,8] (рис. 1):

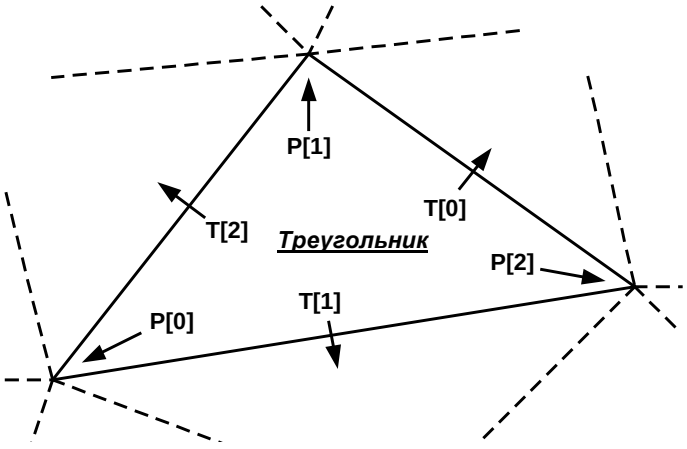

Рис. 1. Структура связей треугольников.

* Работа выполнена при финансовой поддержке РФФИ (грант № 98-05-03194)

```
Точка = record
  X, Y: координата;
end;

Треугольник = record
  P: array [0..2] of Указатель_на_точку;
  T: array [0..2] of Указатель_на_треугольник;
end;
```

В структуре треугольника нумерация точек и соседних треугольников производится в порядке обхода по часовой стрелке, при этом напротив точки с номером $i \in \{0,1,2\}$ располагается ребро, соответствующее соседу с таким же номером.

На приведенных структурах эффективно реализуются оптимальный алгоритм слияния «Разделяй и властвуй», а также итеративный алгоритм [2,3].

3. Алгоритмы построения триангуляции Делоне

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

3.1. Алгоритмы слияния

Концептуально все 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 точек разбивается на две как можно более равные части. Алгоритм триангуляции рекурсивно применяется к подчастям, 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 занимало минимальное время, 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.

В дальнейшем эти алгоритмы рассматриваются детально.

3.2. Итеративные алгоритмы

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, 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. На практике также необходимо учитывать случай попадания точки на уже существующую точку триангуляции (тогда она должна игнорироваться), а также случай попадания точки на некоторое ребро триангуляции, при этом происходит разбиение ребра и смежных треугольников на новые. Принципиально эти варианты не сложнее упомянутых ни по трудоёмкости, ни по реализации, поэтому в дальнейшем они не рассматриваются.

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.

При построении новых треугольников возможны две ситуации, когда добавляемая точка попадает либо внутрь триангуляции, либо вне её. В первом случае строится три новых треугольника и число выполняемых алгоритмом действий фиксировано. Во втором – необходимо построение дополнительных

внешних к текущей триангуляции треугольников, причём их количество может в худшем случае равняться $N-2$. Если же изначально построить суперструктуру (см. разд. 9), то от этой ситуации можно избавиться вообще.

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. Ниже в разд. 10 предлагается гипотеза, утверждающая, что трудоёмкость таких перестроений в среднем составляет $O(1)$, т.е. число перестроений в среднем постоянно. Таким образом, наибольший вклад в трудоёмкость итеративного алгоритма даёт процедура поиска очередного треугольника.

В данной работе предлагается ряд модификаций итеративного алгоритма, которые в той или иной мере решают проблему быстрого поиска, используя специфику последовательного добавления точек.

4. Алгоритм полосового слияния

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

1) разбиения всего исходного множества точек на некоторые полосы;
2) применения специального быстрого алгоритма получения невыпуклых триангуляций полос точек;
3) слияния полученных триангуляций.

Рассмотри эти шаги подробнее.

Шаг 1. Разбиваем множество всех точек на некоторое количество столбцов по принципу их одинаковой ширины по координате X (с помощью цифровой сортировки целых чисел [9-11]) или одинакового количества точек (метод Хоара вычисления квантилей [9,11]). 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.

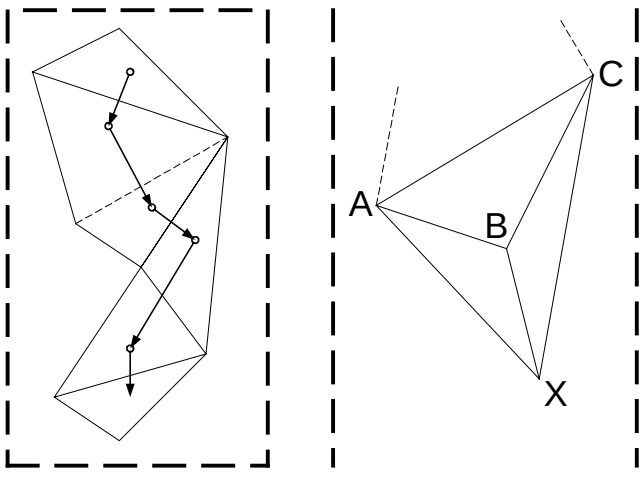

Рис. 2. Триангуляция полосы точек.

Шаг 2. Сортируем все точки внутри столбцов по вертикали (по ко-

ординате *Y*) и триангулируем столбец. Для этого предлагается следующий специальный алгоритм триангуляции (рис. 2). 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, так как тогда с большей вероятностью в будущем не придётся его перестраивать из-за невыполнения условия Делоне.

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 полосы точек. Если предположить верной гипотезу об ограниченном количестве перестроений, рассматриваемую в разд. 10, то трудоёмкость данного шага будет линейной.

В следующих разделах рассматривается вопрос оптимального выбора числа полос для выполнения первого шага алгоритма, а также рассматриваются два варианта выполнения третьего шага.

4.1. Выбор числа полос для полосовых алгоритмов слияния

В данном разделе остановимся на вопросе выбора числа полос для реализации полосовых алгоритмов слияния. Конечно, хорошо было бы найти зависимость среднего числа последующих перестроений от числа полос, но это очень сложно. Поэтому предлагается минимизировать среднюю суммарную длину рёбер всех полученных таким образом невыпуклых триангуляций. Для этого сделаем следующие упрощающие предположения:

1. Координаты точек распределены в прямоугольной области равномерно и независимо по *X* и *Y*.
2. Расстояние между точками определяется по Манхеттену:

$$p(\{x_i,y_i\},\{x_j,y_j\}) = \left|x_i - x_j\right| + \left|y_i - y_j\right|. \qquad (1)$$

Тогда средняя суммарная длина рёбер равна сумме средних длин по *X* плюс сумма средних длин по *Y*. Пусть ширина прямоугольной области есть *a*, высота – *b*, число полос – *m*. Тогда среднее число точек в полосе равно N / m ,

где N – общее число точек в триангуляции. По координате X среднее расстояние между двумя точками в полосе в среднем равно 1/3 от ширины полосы (среднее разности двух равномерно распределённых на интервале величин). Заметим, что в соответствии с приведённым алгоритмом быстрой триангуляции полосы получается $2k-3$ рёбер в каждой полосе (вначале строится один треугольник – 3 ребра; затем остаётся $k-3$ точки, и при каждом добавлении точки строится 2 ребра; итого $3+(k-3)\cdot 2$ рёбер). Средняя ширина полосы равна a/m. Итого, по координате X во всех полосах сумма длин равна $\frac{a}{3m}(2k-3)m$.

Теперь найдём сумму длин рёбер по координате Y. Все построенные рёбра в невыпуклой триангуляции полосы должны принадлежать одной из трёх групп:

1) множеству рёбер, образующих левую границу триангуляции;
2) множеству рёбер, образующих правую границу;
3) множеству рёбер сшивания между границами.

Если пренебречь необходимыми перестроениями треугольников в процессе работы, то получается, что первые две группы оказываются ломаными, протянувшимися «почти» с верха полосы до низа («почти» потому, что с ростом N, а следовательно, и k – среднее расстояние от границы интервала до ближайшей из k равномерно распределённых величин на интервале, равное $1/(k+1)$, стремится к нулю). Средняя длина по координате Y в каждой из этих двух групп составит $b-\frac{2}{k}b$. Третья группа рёбер является ломаной со средней длиной по координате Y $b-\frac{4}{k}b$ и ещё некоторыми дополнительными рёбрами. На рис. 2 одно из таких рёбер показано пунктиром. Пусть средняя общая длина по координате Y таких дополнительных рёбер равна qb. Тогда сумма по координате Y во всех полосах получается равной $(b-\frac{2}{k}b+b-\frac{2}{k}b+b-\frac{4}{k}b+ +qb)\cdot m=(3+q-\frac{8}{k})\cdot bm$. Таким образом, приближённая суммарная длина рёбер в триангуляциях полос составляет $L(m)=\frac{a}{3}(2k-3)+(3+q-\frac{8}{k})\cdot bm$. Если пренебречь членом $\frac{8}{k}$, стремящимся к нулю при больших N, и учесть, что $k=N/m$, то, найдя производную L по m и приравняв её к нулю, получим (в пределе) оптимальное значение для m:

$$L(m)=\frac{a}{3}(2k-3)+(3+q)am=\frac{2aN}{3m}-a+(3+q)bm;$$

$$L'(m) = -\frac{2aN}{3m^2} + (3+q)b;$$

$$L'(m) = 0; \;\Rightarrow\; 2aN = 3(3+q)bm^2; \;\Rightarrow$$

$$m^* = \sqrt{\frac{2aN}{3(3+q)b}}. \qquad (2)$$

Эта оценка позволяет минимизировать сумму длин рёбер полосовых триангуляций, т.е. строить треугольники, которые не будут с большой вероятностью перестраиваться в дальнейшем. Таким образом, выбор числа полос в данном алгоритме влияет на количество последующих перестроений, а следовательно, и на время работы всего алгоритма. Так как оценка (2) включает неизвестную величину q, которую трудно оценить, то было проведено практическое исследование зависимости числа полос от количества исходных точек, минимизирующей время работы всего алгоритма, в виде $\overline{m}^* = \sqrt{s_{сл.пол.} \cdot \frac{a}{b} \cdot N}$, где $s_{сл.пол.}$ – коэффициент разбиения на полосы алгоритма полосового слияния, значение которого необходимо установить. На рис. 3 приведен фрагмент результатов моделирования, в котором отражена зависимость общего времени построения триангуляции Делоне алгоритмом выпуклого полосового слияния от значения $s_{сл.пол.}$ на множестве из 10 000 точек, равномерно и независимо распределённых в квадрате $[0,1]\times[0,1]$.

Как видно из рис. 3, значение константы s можно задавать даже с некоторым разбросом, так как экспериментально установлена большая устойчивость быстродействия полосовых алгоритмов построения триангуляции в зависимо-

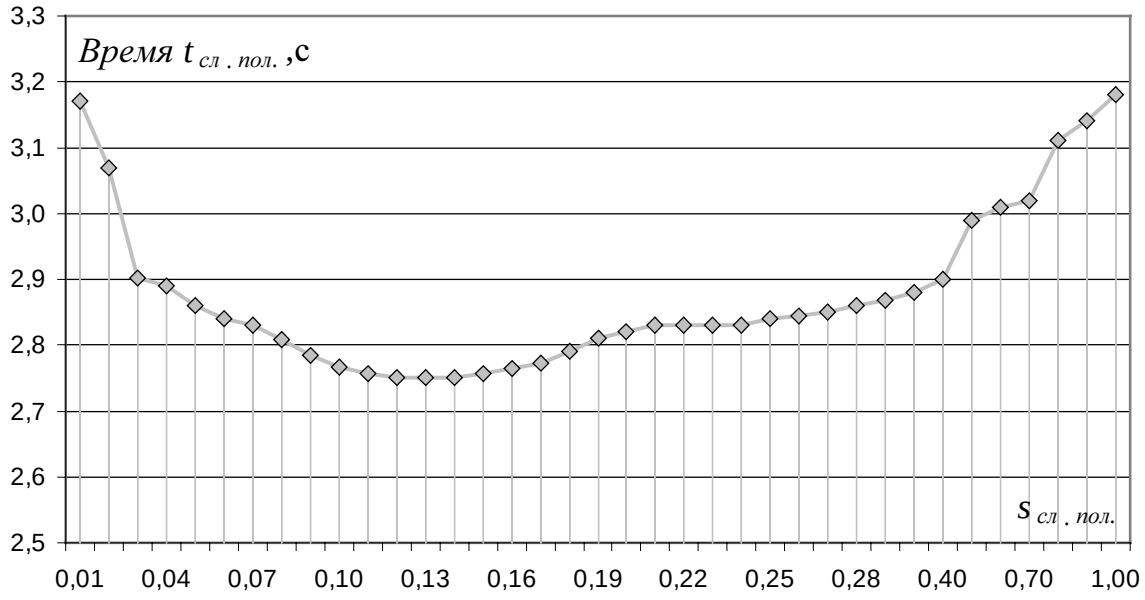

Рис. 3. Зависимость времени построения триангуляции Делоне алгоритмом выпуклого полосового слияния $t_{сл.пол.}$ от значения $s_{сл.пол.}$.

сти от числа задаваемых полос и даже от задаваемого распределения точек.

4.2. Алгоритм выпуклого полосового слияния

Пошаговая схема работы алгоритма выпуклого полосового слияния схематично изображена на рис. 4.

Шаг 3. Вначале необходимо подготовить полученные полосовые триангуляции для алгоритма слияния, применяемого в алгоритме «Разделяй и властвуй», так как 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 (см. рис. 4).

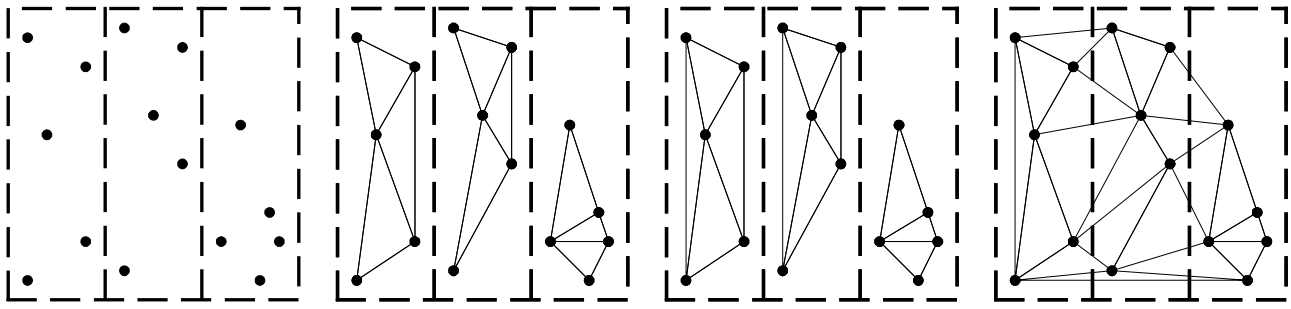

Рис. 4. Шаги работы алгоритма выпуклого полосового слияния.

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, рассматриваемом ниже.

4.3. Алгоритм невыпуклого полосового слияния

Пошаговая схема работы алгоритма невыпуклого полосового слияния схематично изображена на рис. 5.

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. Рассмотрим это подробнее.

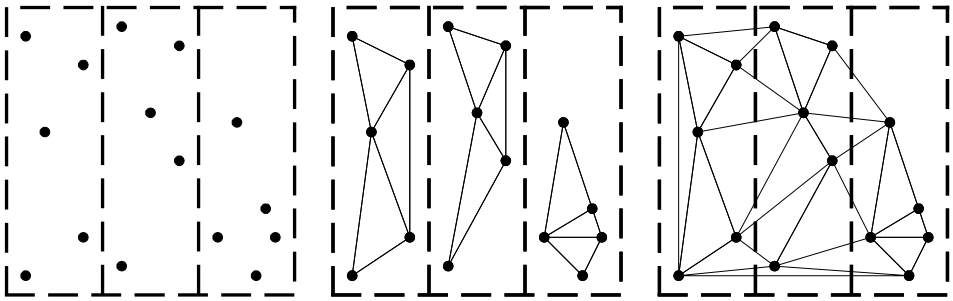

Рис. 5. Шаги работы алгоритма невыпуклого полосового слияния.

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 – $\Delta P_1P_2N_1$ или $\Delta P_1P_2N_2$ (рис. 6). 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.

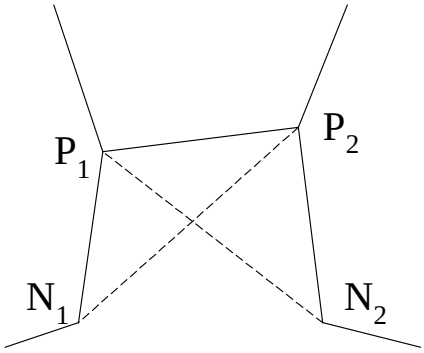

Рис. 6. Сшивание полос.

В алгоритме невыпуклого слияния необходимо проанализировать выпуклость границы 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. 7).

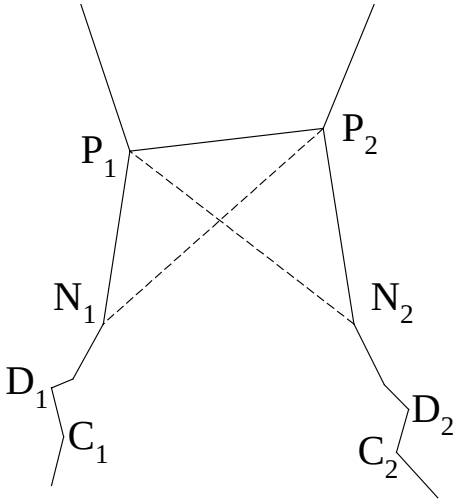

Рис. 7. Локальное достраивание выпуклости.

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.

5. Двухпроходные алгоритмы с откладыванием перестроений

При построении триангуляции Делоне каждый строящийся треугольник после создания должен быть проверен на выполнение условия Делоне для образующих с ним пар 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.

Как показывают эксперименты, довольно большую долю времени отнимают проверки на условие Делоне и перестроения. Для алгоритмов слияния это проявляется наиболее отчётливо. В них приходится тратить довольно много времени для отслеживания корректности указателя на *текущий треугольник* (например, при обходе триангуляции по границе, при слиянии, построении выпуклой оболочки), потому что после того, как треугольник построен, он может тут же в результате неудачной проверки на условие Делоне исчезнуть, а на его месте появятся другие треугольники. Для отслеживания таких явлений приходится заводить специальные структуры типа записи *«кромка»*, которая описывает некоторую вершину на границе триангуляции, и создавать процедуры поддержки таких структур для ситуаций, когда треугольники, смежные с описываемой вершиной, исчезают или появляются.

В данном разделе предлагается упростить логику алгоритмов слияния следующим образом. За первый проход надо 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 авторами 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, так как 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 рассматриваемых в данной работе алгоритмов слияния. За счёт значительного упрощения удалось реализовать алгоритмы, работающие в среднем на 10% быстрее, чем исходные алгоритмы.

Предлагаемый алгоритм с откладыванием перестроений треугольников требует теоретического обоснования ряда возникающих в связи с этим вопросов:

1. Возможно ли преобразовать произвольную триангуляцию перестроениями пар соседних треугольников в триангуляцию Делоне?
2. Возможно ли преобразовать произвольную триангуляцию посредством только улучшающих перестроений в триангуляцию Делоне?
3. Если можно, то можно ли за конечное время?
4. Если можно, то какова трудоёмкость этого преобразования в среднем, например, на равномерном распределении?

Первые три вопроса подробно обсуждаются и получают утвердительный ответ в работах [2,12].

Обсудим четвёртый вопрос трудоёмкости для двухпроходных алгоритмов. В существующих доказательствах трудоёмкости классического алгоритма слияния «Разделяй и властвуй» [2,13] установленная трудоёмкость $O(N \log N)$ не зависит от количества перестроений, поэтому и трудоёмкость первого прохода двухпроходного алгоритма «Разделяй и властвуй» составляет $O(N \log N)$.

На первом проходе полосовых алгоритмов слияния присутствует сортировка чисел внутри столбцов, все остальные части прохода имеют линейную трудоёмкость, что следует из описания алгоритма. Поэтому общая трудоёмкость первого прохода полосового алгоритма равна трудоёмкости сортировки чисел внутри столбцов. Используя цифровую сортировку целых чисел можно получить в общем трудоёмкость $O(N)$, если же используется какой-либо из алгоритмов сортировки, применяющий для своей работы только свойство линейного порядка чисел (сортировка Хоара, пирамидальная и др.), то общая трудоёмкость алгоритма становится не лучше $O(N \log N)$.

Если принять верной гипотезу, предлагаемую в разд. 10, то трудоёмкость второго прохода рассматриваемых алгоритмов будет составлять $O(N)$ в среднем.

Таким образом, общая трудоёмкость алгоритма слияния «Разделяй и властвуй» составляет $O(N \log N)$ в худшем, а полосовых алгоритмов в среднем $O(N \log N)$ или $O(N)$ в зависимости от используемого алгоритма сортировки точек внутри столбцов.

6. Итеративные алгоритмы с ускорением поиска

Поиск треугольника (задача локализации точки) в итеративном алгоритме состоит из двух этапов: выбора некоторого исходного треугольника и собственно целенаправленного поиска – цепочки последовательных переходов по соседним треугольникам к цели с помощью структуры связанных треугольников.

Во всех предлагаемых в данной работе алгоритмах с ускорением поиска строятся некоторые структуры, с помощью которых удаётся выбрать исходный треугольник для поиска достаточно близко к целевому. В работе рассматриваются два варианта ускорения:

1) изменение порядка добавления точек в триангуляцию;

2) кэширование поиска.

Эти алгоритмы позволяют значительно сократить время поиска, уменьшив в некоторых случаях на реальных данных трудоёмкость алгоритма до линейной.

7. Итеративные алгоритмы с изменённым порядком добавления точек в триангуляцию

Интересные результаты можно получить, если разрешить при построении триангуляции производить *предобработку* точек, изменяя порядок добавления точек в частично готовую триангуляцию [2,4].

В работе [4] предлагается изменить порядок добавления точек так, чтобы каждая следующая точка была максимально близка к предыдущей добавленной точке. Тогда, запоминая треугольник, найденный на предыдущем шаге, мы можем использовать его в качестве отправной точки для текущего поиска, применяя алгоритм поиска из простейшего итеративного алгоритма. Удачно перестраивая порядок добавления точек, можно достичь очень неплохих результатов. Но при этом на первый план может выйти трудоёмкость этой самой предобработки.

В данной работе предлагается простой вариант переупорядочения точек в виде разбиения множества точек на полосы и квадраты.

7.1. Итеративный полосовой алгоритм

В этом алгоритме предлагается разбить все точки на полосы по одной координате, а затем отсортировать все точки внутри полос по другой координате. В этом случае, подобрав соответствующее количество полос, можно существенно уменьшить расстояние между последовательностью точек, а следовательно, и время работы алгоритма (рис. 8). В работе [14] теоретически определено оптимальное количество полос $m = \left[\sqrt{\frac{aN}{3b}}\right]$ для разбиения точек на полосы при равномерном независимом распределении точек в прямоугольнике размером $a \times b$, ис-

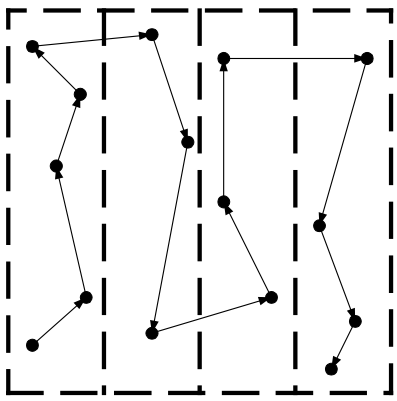

Рис. 8. Полосовой итеративный алгоритм.

ходя из условия минимизации суммарного общего расстояния между последовательными точками разбиения.

Для изучения влияния числа точек на общее время работы полосового итеративного алгоритма было проведено экспериментальное исследование, в котором оценивалась зависимость вида $\overline{m} = [\sqrt{s_{ит.пол.} \cdot \frac{a}{b} \cdot N}]$, где $s_{ит.пол.}$ – константа разбиения на полосы итеративного полосового алгоритма. На рис. 9 представлены результаты моделирования для наборов точек, равномерно распределённых в единичном квадрате. При этом видно, что число полос лучше выбирать примерно в 2 раза меньше, чем в приведённой выше оценке, что объясняется отсутствием в ней учёта времени выполнения других операций алгоритма (не поиска треугольников), в частности предобработки и перестроения треугольников.

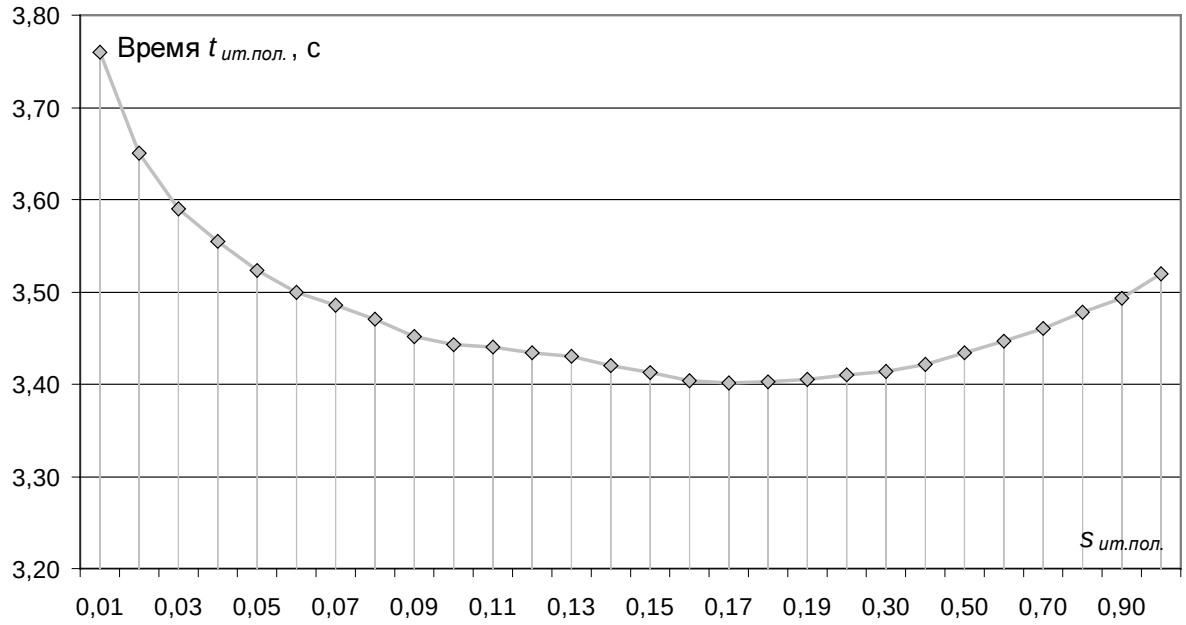

Рис. 9. Зависимость времени построения триангуляции Делоне итеративным полосовым алгоритмом $t_{ит.пол.}$ от значения $s_{ит.пол.}$.

7.2. Итеративный квадратный алгоритм

В работе [2] рассматривается итеративный алгоритм, в котором предлагается разбить плоскость с точками на $\sqrt{N}$ квадратов, а добавление точек производить последовательными группами, соответствующими смежным квадратам. В работе [2] устанавливается трудоёмкость алгоритма, равная $O(N^{3/2})$ в среднем и $O(N^2)$ в худшем случае.

На основе этой идеи в данной работе предлагается итеративный квадратный алгоритм, в котором необходимо разбить плоскость с точками на квадраты

(прямоугольники) по принципу примерного равенства числа точек в квадратах, а добавление точек производить последовательными группами, соответствующими смежным квадратам (рис. 10). При этом если в каждом квадрате есть хотя бы одна точка, то расстояние между точками в последовательности добавляемых точек будет определяться размером квадрата.

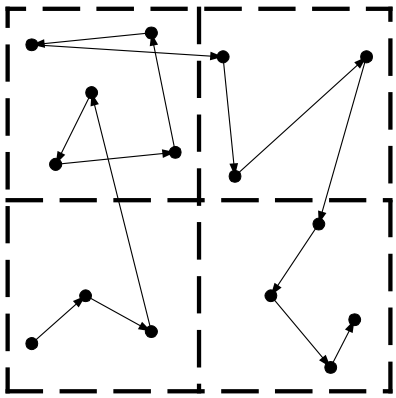

Рис. 10. Квадратный итеративный алгоритм.

Для оценки оптимального числа квадратов предлагается минимизировать расстояние между последовательностью добавляемых в триангуляцию точек (назовём эту последовательность *путём*). Для этого сделаем следующие упрощающие предположения:

1) координаты точек распределены в квадрате $a \times a$ равномерно и независимо по *X* и *Y*;
2) расстояние между точками определяется по Манхеттену (формула (2)).

Тогда средняя длина пути равна сумме средних длин переходов по *X* плюс сумма средних длин по *Y*. Пусть число квадратов разбиения равно m^2. Тогда среднее число точек в квадрате равно N/m^2, где N – общее число точек в триангуляции. Всего в данном алгоритме получается $m^2 - 1$ переходов в соседние квадраты и $N - m^2$ переходов внутри одного квадрата. Среднее расстояние по координатам *X* и *Y* между двумя точками в квадрате равно 1/3 от ширины квадрата a/m (среднее разности двух равномерно распределённых на интервале величин), т.е. $2a/(3m)$. Средние расстояния между двумя точками в соседних квадратах равны $(a/m, a/(3m))$ по двум координатам (это будет пара (X,Y) либо (Y,X) в зависимости от того, как соприкасаются соседние квадраты – вертикальными или горизонтальными сторонами), т.е. $4a/(3m)$. Пренебрегая единицей по сравнению с m^2 при $n \to \infty$, в пределе общую длину пути получим равной $L(m) = (N - m^2) \cdot 2a/(3m) + m^2 \cdot 4a/(3m)$. Теперь, найдя производную L по m, приравняем её к нулю, и тогда из полученного уравнения получим приближённое оптимальное значение для m:

$$L(m) = \frac{2aN}{3m} + \frac{2}{3}a \cdot m;$$

$$L'(m) = -\frac{2aN}{3m^2} + \frac{2}{3}a;$$

$$L'(m) = 0; \quad \Rightarrow \quad aN = am^2; \quad \Rightarrow \quad m^* = \sqrt{N}\,. \qquad (3)$$

Эта оценка минимизирует длину пути, но не общее время работы алгоритма, так как не учитывает время, затрачиваемое на предобработку, которое существенным образом зависит от числа квадратов разбиения. На практике очень многое будет определять соотношение эффективностей реализаций алгоритма предобработки и основного алгоритма. В проводимых автором экспериментах количество квадратов выбиралось автором по формуле $m = [\sqrt{s_{ит.кв} \cdot N}]$, где значение коэффициента разбиения на полосы итеративного квадратного алгоритма $s_{ит.кв} = 0{,}005$, т.е. значительно меньше теоретически установленного. Фрагмент результатов экспериментального исследования зависимости времени построения триангуляции Делоне итеративным квадратным алгоритмом $t_{ит.кв}$ от значения $s_{ит.кв}$ представлен на рис. 11.

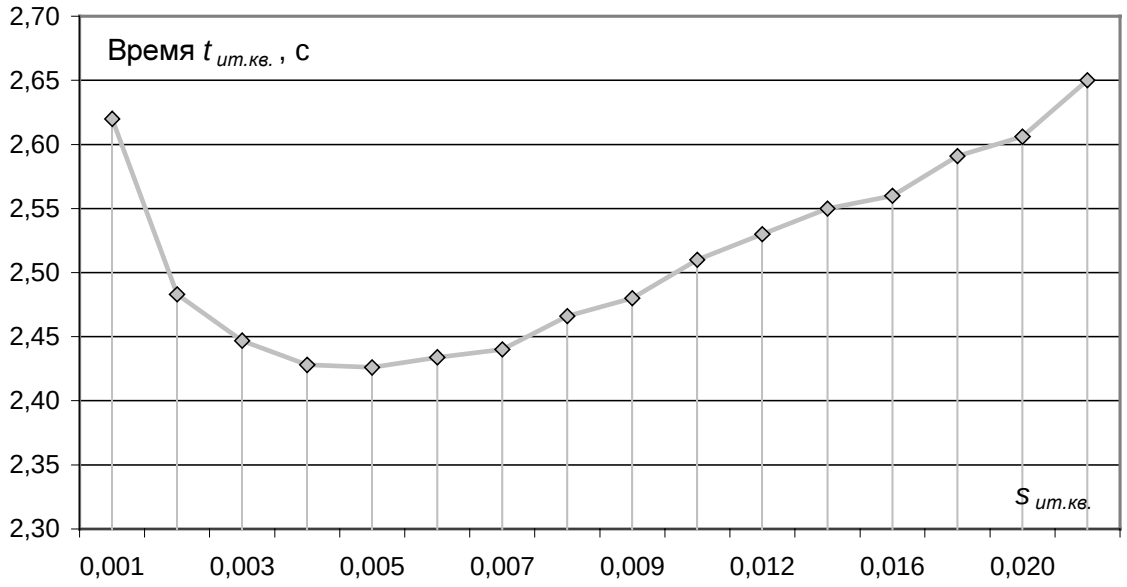

Рис. 11. Зависимость времени построения триангуляции Делоне итеративным квадратным алгоритмом $t_{ит.кв.}$ от значения $s_{ит.кв.}$.

Так как количество выбираемых квадратов в данном алгоритме равно $s_{ит.кв} \cdot N$, а количество точек в одном квадрате в среднем равно $1/s_{ит.кв}$, то средняя трудоёмкость работы алгоритма без учёта предобработки составляет $s_{ит.кв} \cdot N \cdot \left((1/s_{ит.кв})^{3/2} \right) = \theta(N)$. При этом трудоёмкость предобработки (разбиение точек на квадраты) также является линейной в случае использования для этого алгоритма цифровой сортировки. Таким образом, общая средняя трудоёмкость работы итеративного полосового алгоритма является линейной.

8. Итеративные алгоритмы триангуляции с кэшированием

Алгоритм кэширования поиска использует решение отдельной задачи локализации точки на планарном разбиении, не учитывая особенности конкрет-

ной задачи построения триангуляции. В литературе [1,9] описываются несколько оптимальных алгоритмов поиска, основанных на идее бинарного поиска, и тратящих на поиск не более $O(\log N)$ шагов. Рассмотрим алгоритм кэширования поиска в общем случае.

8.1. Локализация точки с кэшированием

Задача локализации точки. Дано планарное разбиение плоскости R_0 (в данном случае триангуляция) и точка (x, y). Надо найти элемент планарного разбиения $r_{0(j)}$ (треугольник), содержащий заданную точку.

Для решения поставленной задачи 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 $r_{1(i)}$ разбиения R_1 элемент $r_{0(j)}$ разбиения R_0 так, чтобы элемент $r_{0(j)}$ находился как можно ближе к $r_{1(i)}$ (например, в смысле расстояний между центрами элементов разбиения). Тогда при поступлении очередного запроса на поиск необходимо найти элемент разбиения $r_{1(i)}$, а по нему $r_{0(j)}$, который располагается обычно довольно близко к цели. После этого за несколько шагов, передвигаясь по цепочке соседних элементов разбиения (треугольников), находится целевой $r_{0(цел)}$. После успешного окончания поиска мы можем изменить соответствие $r_{1(i)} \to r_{0(j)}$ на другое, более уместное для данной задачи, а именно на $r_{1(i)} \to r_{0(цел)}$. Таким образом, после выполнения очередного запроса на поиск алгоритм становится всё более информированным, а поиск с каждым шагом ускоряется.

Теперь рассмотрим, как лучше выбрать разбиение R_1. Для быстрого поиска удобнее всего разбить всю данную область, например, на регулярную сеть квадратов (либо сеть одинаковых треугольников), проведя равноотстоящие горизонтальные и вертикальные прямые по всей области (рис. 12). 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 $i, j \in [0; m-1] \subset \mathrm{N}$: $r_{1(i,j)}$. 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.

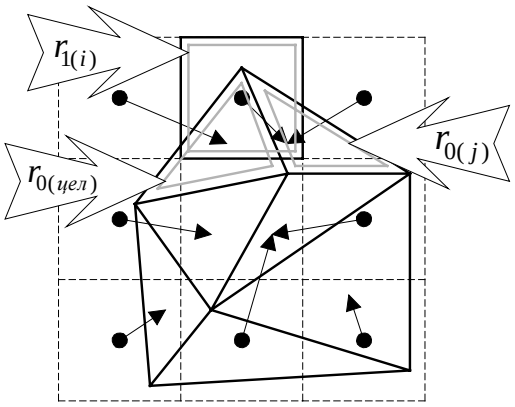

Рис. 12. Локализация точек.

8.2. Алгоритм статического кэширования

В предлагаемом алгоритме построения триангуляции Делоне методом статического кэширования необходимо выбрать 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: $r_{1(i,j)}$:= ссылка на T.

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

8.3. Алгоритм динамического кэширования

В данном алгоритме 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:

$$\forall i, j = \overline{0;\, m_{стар} - 1}:$$
$$r_{1нов(\,2i,2\,j)}, r_{1нов(\,2i,2\,j+1)}, r_{1нов(\,2i+1,2\,j)}, r_{1нов(\,2i+1,2\,j+1)} := r_{1стар(i,j)}\ . \qquad (4)$$

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

8.4. Трудоёмкость алгоритмов кэширования

Для последующих рассуждений нам понадобится следующая теорема, установленная в работах [2,4]:

Теорема 1. Пусть дана триангуляция Делоне на множестве точек, равномерно распределённых в квадрате. Пусть даны два треугольника из этой триангуляции. Тогда для перехода из одного треугольника в другой вдоль прямой, соединяющей некоторые точки этих треугольников, в среднем требуется $\theta(\sqrt{N}) = c_{пер}\sqrt{N}$ операций, где $c_{пер} = \text{const}$.

В следующей теореме устанавливается трудоёмкость алгоритма статического кэширования.

Теорема 2 (трудоёмкость алгоритма статического кэширования). Пусть даны N точек, распределенных в единичном квадрате равномерно и независимо. Пусть размер кэша равен $m\times m$, где $m \sim N^{3/8}$. Тогда трудоёмкость алго-

ритма построения триангуляции Делоне методом статического кэширования в среднем составляет $O(N^{9/8})$.

Доказательство. При использовании кэша, имеющего m^2 ячеек и первоначально пустом, в среднем первые m^2 операций приводят в сумме к $c_{пер} \cdot \sum_{i=1}^{m^2} \sqrt{i}$ переходам (на основании теоремы 1). Будем считать, что последующие операции добавления точек в триангуляцию будут приводить к попаданию в ячейку кэша, в которую уже приходилось попадать, а потому будем считать, что локализацию точки надо проводить только в пределах данной ячейки, а не всего единичного квадрата. Если принять, что в среднем в ячейке на i-м шаге находится i/m^2 точек, то это даёт ещё $c_{пер} \cdot \sum_{i=m^2+1}^{N} \frac{\sqrt{i}}{m}$ переходов. Итого, общая трудоёмкость операций поиска равна

$$
\begin{aligned}
R(m) &= c_{пер.} \cdot \Big(\sum_{i=1}^{m^2} \sqrt{i} + \sum_{i=m^2+1}^{N} \frac{\sqrt{i}}{m}\Big) \approx \\
&\approx c_{пер.} \cdot \Big(\int_{1}^{m^2+1} \sqrt{x}\,dx + \frac{1}{m} \int_{m^2+1}^{N+1} \sqrt{x}\,dx \Big) \approx \\
&\approx \overline{R}(m) = c_{пер.} \cdot \Big(\int_{1}^{m^2} \sqrt{x}\,dx + \frac{1}{m} \int_{m^2}^{N} \sqrt{x}\,dx \Big) = \\
&= c_{пер.} \cdot \frac{2}{3} \left(x^{3/2} \Big|_{1}^{m^2} + \frac{1}{m} x^{3/2} \Big|_{m^2}^{N} \right) = \\
&= c_{пер.} \cdot \frac{2}{3} \Big(m^3 - m^2 - 1 + \frac{N^{3/2}}{m}\Big).
\end{aligned}
\tag{5}
$$

Приравняв производную $\overline{R}(m)$ к нулю и приняв некоторые упрощения, получим оценку оптимального значения для размера кэша и оценку трудоёмкости:

$$
\overline{R}'(m) = c_{пер.} \cdot \frac{2}{3}\Big(3m^2 - 2m - \frac{N^{3/2}}{m^2}\Big);
$$

$$
\overline{R}'(m) = 0; \quad \Rightarrow \quad 3m^4 - 2m^3 = N^{3/2}.
$$

Так как при $m \to \infty$ член $2m^3$ становится пренебрежимо малым по сравнению с $3m^4$, то

$$
3m^4 \approx N^{3/2}; \quad \Rightarrow m^* \sim \mathrm{N}^{3/8}; \; \overline{R}(m^*) \sim \mathrm{N}^{9/8}, \tag{6}
$$

что и требовалось доказать

Для практического применения данного алгоритма немаловажным является вопрос выбора размера кэша *m*. В соответствии с формулой (6), размер кэша следует выбирать по формуле вида $m = s_{стат} \cdot \mathrm{N}^{3/8}$, где $s_{стат}$ – коэффициент статического кэша. На рис. 13 представлен фрагмент результатов экспериментального исследования для оценки значения $s_{стат}$, из которого видно, что наилучшее время достигается при $s_{стат} \approx 0{,}7$.

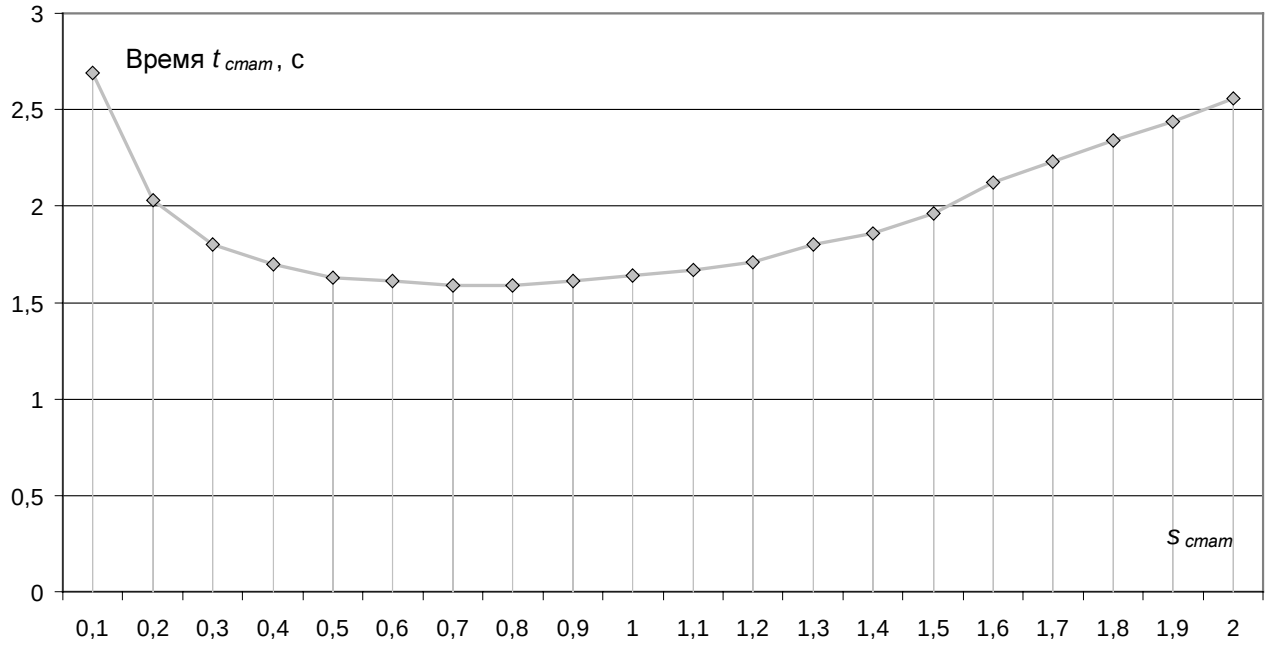

Рис. 13. Зависимость времени построения триангуляции Делоне алгоритмом статического кэширования $t_{стат}$ от значения $s_{стат}$.

Следующая теорема устанавливает трудоёмкость работы алгоритма динамического кэширования в среднем на равномерном распределении.

Теорема 3 (трудоёмкость алгоритма динамического кэширования). Пусть даны *N* точек, распределенных в единичном квадрате равномерно и независимо. Пусть размер кэша увеличивается в 2 раза каждый раз при достижении числа точек в триангуляции $n = r_{кэш} \cdot m^2$, где $r_{кэш}$ – коэффициент роста динамического кэша, а m – текущий размер кэша. Тогда трудоёмкость алгоритма построения триангуляции Делоне методом динамического кэширования в среднем составляет $O(N)$.

Доказательство. Рассмотрим цикл добавления точек при постоянном размере кэша $m = 2^p$. Пусть при последнем увеличении кэша произошло копирование старых ячеек в 4 новых. Тем самым при использовании нового кэша вначале будет возможна локализация точек в среднем в пределах групп по 4

ячейки. Тогда первые m^2 операций вставки точек будут приводить в среднем к попаданию в ячейки, в которые мы ещё не попадали в данном цикле. А поэтому надо будет проводить локализацию в пределах 4 ячеек, т.е. придётся выполнять порядка $c \cdot \sqrt{\frac{i}{m^2}} = c \cdot \frac{\sqrt{i}}{m}$ переходов при поиске, где i – номер текущей добавляемой точки, а c – некоторая константа.

Так как увеличение кэша в 2 раза производится при достижении числа точек в триангуляции, равном $r_{кэш} \cdot m^2$, то тогда верно неравенство $N \le r_{кэш} \cdot M^2$, где M – максимальный размер кэша. Пусть $M = 2^P$. Тогда можно записать суммарное количество операций переходов при поиске:

$$R(N, r_{кэш}) \le R_1(N, r) = \sum_{p=1}^{P-1} \Bigg(\sum_{i=r_{кэш} \cdot 2^{2p}}^{r_{кэш} \cdot 2^{2p} + 2^{p+1} - 1} 2c \cdot \frac{\sqrt{i}}{2^p} + \sum_{i=r_{кэш} \cdot 2^{2p} + 2^{p+1}}^{r_{кэш} \cdot 2^{2(p+1)}} c \cdot \frac{\sqrt{i}}{2^p} \Bigg). \qquad (7)$$

Так же, как и в предыдущем случае, заменим суммы интегралами и найдём оценку числа переходов $\overline{R}(N, r)$:

$$R_1(N, r) \le R_2(N, r) = \int_p^P \Bigg(\int_{r \cdot 2^{2p}}^{r \cdot 2^{2p} + 2^{p+1}} 2c \cdot \frac{\sqrt{x}}{2^p} dx + \int_{r \cdot 2^{2p} + 2^{p+1}}^{r \cdot 2^{2(p+1)}} c \cdot \frac{\sqrt{x}}{2^p} dx \Bigg) dp =$$

$$= \left(\frac{4c}{3 \cdot 2^p} x^{3/2} \Big|_{x = r \cdot 2^{2p}}^{x = r \cdot 2^{2p} + 2^{p+1}} + \frac{2c}{3 \cdot 2^p} x^{3/2} \Big|_{x = r \cdot 2^{2p} + 2^{p+1}}^{x = r \cdot 2^{2(p+1)}} \right) \Bigg|_{p=1}^{p=P} =$$

$$= \left(\frac{2c}{3 \cdot 2^p} \left((r \cdot 2^{2p} + 2^{p+1})^{3/2} - 2(r \cdot 2^{2p})^{3/2} + (r \cdot 2^{2(p+1)})^{3/2} \right) \right) \Bigg|_{p=1}^{p=P} \le$$

$$\le \left(\frac{2c}{3 \cdot 2^p} (r \cdot 2^{2(p+1)})^{3/2} \right) \Bigg|_{p=1}^{p=P} = \left(\frac{cr^{3/2} \cdot 2^{2p+4}}{3} \right) \Bigg|_{p=1}^{p=P} =$$

$$= \frac{cr^{3/2} \cdot 16}{3} \left(\left(2^P\right)^2 - 4 \right) = \frac{cr^{3/2} \cdot 16}{3} \left(M^2 - 4 \right) \approx \overline{R}(N, r) = \frac{16 \cdot c}{3} r^{1/2} N. \qquad (8)$$

Таким образом, количество переходов в среднем линейно зависит от N: $O(R(N)) = O(N)$.

Кроме операций поиска, в алгоритме динамического кэширования производятся локальные перестроения (их количество порядка $3 \cdot N$ в соответствии с гипотезой в разд. 10), а также операции увеличения кэша. Трудоёмкость увеличения кэша получается порядка $O(\sum_{p=1}^{P} 2^p) = O(2^{P+1} - 1) = O(N)$.

Итого, общая трудоёмкость алгоритма динамического кэширования в среднем на равномерном распределении в квадрате равна $O(N)$, что и требовалось доказать.

Для использования алгоритма динамического кэширования на практике необходимо оценить значение коэффициента роста динамического кэша $r_{кэш}$, зависящее от программно-аппаратной реализации алгоритма. Было проведено экспериментальное моделирование работы алгоритма для оценки значения $r_{кэш}$, фрагмент результатов которого представлен на рис. 14, из которого видно, что наилучшие результаты достигаются при $r_{кэш} \approx 5$.

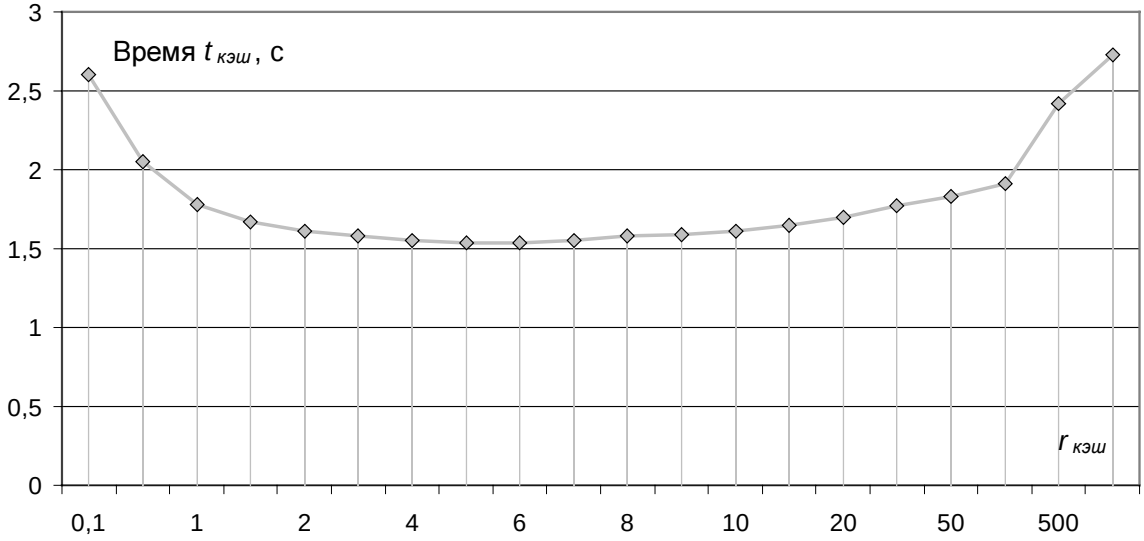

Рис. 14. Зависимость времени построения триангуляции Делоне алгоритмом динамического кэширования $t_{кэш}$ от значения $r_{кэш}$.

9. Суперструктуры для итеративных алгоритмов

Итеративные алгоритмы построения триангуляции Делоне можно несколько упростить за счёт изменения алгоритма добавления точек в частично готовую триангуляцию. При добавлении точки возможно два варианта, когда очередная точка должна быть включена внутрь текущей построенной триангуляции или вне её.

Предположим, что в множество заданных исходных точек разрешено добавить некоторое множество дополнительных точек. Если это множество изначально подобрать так, что построенная на нём триангуляция (*суперструктура*) будет заведомо покрывать все заданные точки, то тогда алгоритм включения текущей точки в триангуляцию будет заметно упрощен за счёт исключения варианта попадания точки вне текущей триангуляции.

Для экспериментальных исследований, вероятно, проще изначально построить треугольник, покрывающий всё множество точек. На практике же выгоднее могут быть, например, несколько точек, распределённых по окружности (в частном случае четырёх точек – квадрат), или же множество точек, равномерно распределённые вдоль выпуклой оболочки входного множества точек (рис. 15).

Были проведены экспериментальные исследования для выяснения влияния суперструктур на производительность итеративных алгоритмов построения триангуляции Делоне.

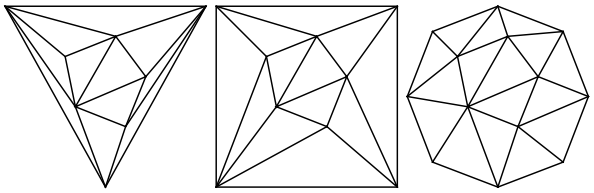

Рис. 15. Варианты суперструктур.

Так, например, для равномерного распределения в единичном квадрате построение суперструктуры в виде единичного квадрата практически на всех видах обсуждаемых в данной работе алгоритмах приводит к некоторому снижению скорости работы. Для такого распределения выгоднее оказалось строить суперструктуру на множестве точек (количество $\sim \sqrt{N}$), равномерно распределённых вдоль периметра квадрата.

Во время исследований выяснилось, что для различных конфигураций точек и различных алгоритмов возможно как увеличение скорости работы алгоритмов, так и снижение. Однако во всех проведённых экспериментах скорость работы алгоритмов с суперструктурой и без неё не отличалась более чем на 10%.

10. Число перестроений в алгоритмах триангуляции

Следующая выдвигаемая гипотеза устанавливает количество перестроений пар соседних треугольников при добавлении в триангуляцию Делоне новой точки итеративным алгоритмом.

Гипотеза. Пусть дан единичный квадрат $[0;1]\times[0;1]$, на котором равномерно распределены N точек. Пусть на этих точках построена триангуляция Делоне. Рассмотрим операцию добавления новой точки в триангуляцию. Математическое ожидание $M_{пер}$ числа последующих перестроений стремится к 3 при $n \to \infty$.

Обоснование. Число перестроений очень сильно зависит от формы границы триангуляции, так как на границе обычно существует очень много длинных вытянутых треугольников. Поэтому рассмотрим более простую задачу о нахождении числа перестроений после добавления точки «вдали от границы», когда перестроения не затронут граничные треугольники.

Исследуем добавление *i*-й и *j*-й точек в триангуляцию ($i < j$). Будем рассматривать достаточно большие области вокруг *j*-й точки, растянутой по обеим координатам в $\sqrt{\frac{i}{j}}$ раз и область вокруг *i*-й точки такого же размера, как и другая после растяжения. Тогда точки в этих областях будут иметь один и тот же закон распределения, условия добавления точек будут совпадать, а следовательно, трудоёмкости перестроений равны. Поэтому количество перестроений $M_{пер}$ вдали от границы не зависит от текущего числа точек в триангуляции, т.е. это константа!

Ранее понятие «вдали от границы» было определено как расстояние, на котором перестроения не затрагивают граничные треугольники. А так как только что было сказано, это расстояние $M_{пер}$ в среднем постоянно, поэтому для области с конечным периметром (в данном случае единичного квадрата) площадь приграничной зоны стремится к нулю, а следовательно, общая трудоёмкость перестроений, включая приграничную зону, стремится к $M_{пер}$ при росте *N*. *Конец обоснования.*

Для вычисления числа $M_{пер}$ были проведены экспериментальные исследования, по результатам которых удалось установить, что число перестроений $M_{пер}$ равно 3 с очень высокой точностью на экспериментах до одного миллиона точек на различных законах распределения (равномерное распределение в областях, нормальное, Лапласа, множество несимметричных распределений). Также были проведены эксперименты по изучению числа перестроений вблизи границы. Это число во всех проведённых экспериментах оказалось ≤ 3.

11. Сравнение алгоритмов триангуляции

В таблице и на рис. 16 приведён фрагмент сравнения рассмотренных в данной работе алгоритмов построения триангуляции Делоне как зависимость времени построения триангуляции Делоне различными алгоритмами от количества тестовых точек, распределённых равномерно и независимо в единичном квадрате. В приведённых данных моделирования с доверительной вероятностью 0,95 относительная точность значений составляет 1%. Все алгоритмы были реализованы и протестированы в одной программной среде и на одной аппаратной платформе (Borland Pascal 7.0, Pentium 166MMX).

В работах [1,2] наиболее эффективным алгоритмом считается алгоритм «Разделяй и властвуй». Однако, как видно из приведённых статистических данных, явным лидером по производительности является алгоритм динамического кэширования. Важным его достоинством представляется гораздо более простая алгоритмическая реализация по сравнению с алгоритмами слияния.

Кроме равномерного распределения, авторами было проведено моделирование работы алгоритмов на множестве неравномерных распределений (рас-

пределения Лапласа, Гаусса, различные варианты комбинированных распределений). При этом была обнаружена высокая устойчивость всех протестированных алгоритмов по отношению к изменению закона распределения точек. Во всех проведённых тестах изменение времени работы алгоритмов не превышало 5% по сравнению с равномерным распределением.

Таблица. Сравнение алгоритмов построения триангуляции Делоне.

Число точек	Время работы алгоритмов *t*, с															
	Алгоритмы слияния						Итеративные алгоритмы									
	«Разделяй и властвуй»		Выпуклое слияние		Невыпуклое слияние		Простейший итеративный		Статическое кэширование		Динамическое кэширование		Полосовой итеративный		Квадратный итеративный	
	1 проход	2 прохода	1 проход	2 прохода	1 проход	2 прохода	С супер.	Без супер.	С супер.	Без супер.	С супер.	Без супер.	С супер.	Без супер.	С супер.	Без супер.
	№1	*№2*	*№3*	*№4*	*№5*	*№6*	*№7*	*№8*	*№9*	*№10*	*№11*	*№12*	*№13*	*№14*	*№15*	*№16*
1 000	0,27	0,27	0,23	0,23	0,22	0,22	0,24	0,25	0,21	0,20	0,13	0,14	0,28	0,29	0,20	0,21
2 000	0,56	0,55	0,45	0,43	0,45	0,43	0,62	0,66	0,39	0,41	0,28	0,30	0,62	0,68	0,44	0,49
3 000	0,87	0,85	0,76	0,72	0,68	0,72	1,13	1,08	0,61	0,58	0,45	0,48	0,94	1,03	0,71	0,72
4 000	1,16	1,12	1,03	0,99	0,94	0,99	1,67	1,69	0,75	0,73	0,61	0,60	1,35	1,46	1,00	1,07
5 000	1,50	1,42	1,35	1,30	1,16	1,21	2,20	2,26	0,90	0,90	0,74	0,77	1,69	1,88	1,24	1,36
6 000	1,80	1,68	1,64	1,57	1,42	1,37	2,80	2,91	1,05	1,04	0,89	0,92	2,07	2,28	1,52	1,64
7 000	2,12	1,98	1,93	1,79	1,68	1,53	3,46	3,59	1,22	1,21	1,02	1,09	2,47	2,70	1,81	1,96
8 000	2,44	2,21	2,25	2,02	1,93	1,76	4,19	4,29	1,35	1,35	1,17	1,21	2,83	3,13	2,09	2,30
9 000	2,76	2,50	2,52	2,35	2,20	2,05	4,93	4,98	1,56	1,55	1,35	1,39	3,20	3,54	2,36	2,52
10 000	3,14	2,79	2,79	2,56	2,54	2,24	5,80	5,90	1,68	1,71	1,49	1,57	3,60	3,93	2,61	2,92
20 000	6,42	5,87	5,77	5,38	5,16	4,86	16,09	15,71	3,29	3,35	3,07	3,07	7,63	8,45	5,55	6,10
30 000	10,05	9,09	9,07	8,20	8,08	7,19	29,00	28,95	4,94	5,22	4,50	4,89	11,59	13,07	8,62	9,44
40 000	13,29	11,94	11,96	10,87	10,58	9,62	44,49	43,94	6,81	7,03	6,16	6,42	16,09	17,85	11,75	12,79
50 000	16,97	15,21	15,17	13,62	13,38	12,03	56,90	56,74	8,57	8,57	7,69	7,91	19,66	22,13	14,56	16,20

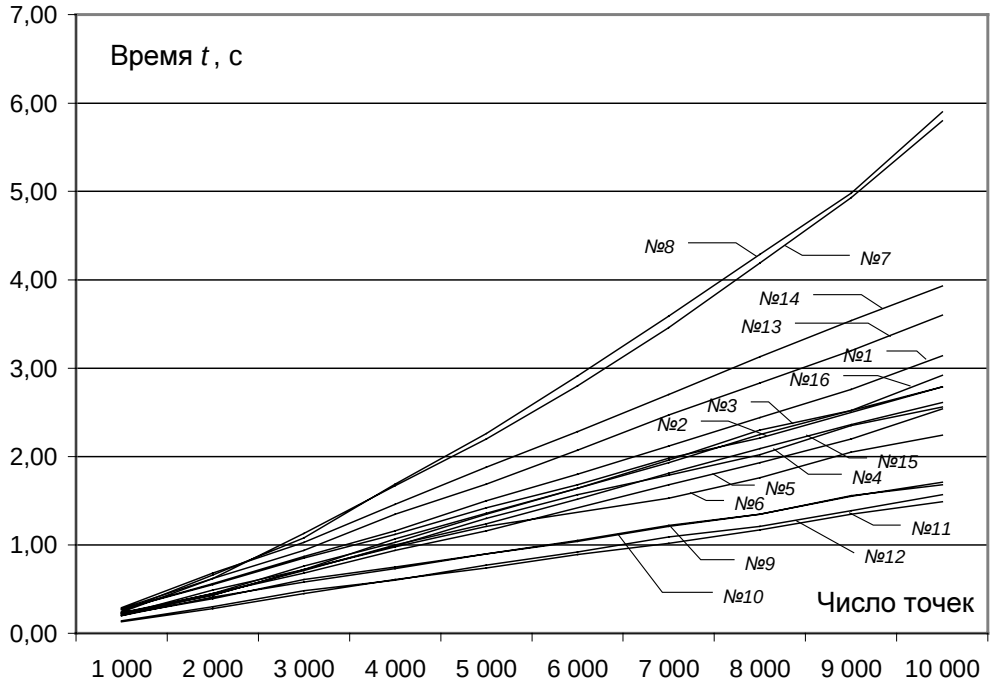

Рис. 16. Сравнение алгоритмов построения триангуляции Делоне (зависимость времени построения триангуляции *t* разными алгоритмами от числа точек).

ЛИТЕРАТУРА

1. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение / Пер. с англ. – М.: Мир, 1989. – 478 с.
2. Lee D., Schachter B. Two algorithms for constructing a Delaunay triangulation // Int. Jour. Comp. and Inf. Sc., 1980, Vol. 9, No. 3, p. 219-242.
3. Shapiro M. A note on Lee and Schachter's algorithm for Delaunay triangulation // Inter. Jour. Of Comp. and Inf. Sciences, 1981, Vol. 10, No. 6, p. 413-418.
4. Lawson C. Software for C^1 surface interpolation // Mathematical Software, 1977, No. 3, p. 161-194.
5. Sloan S., Houlsby G., An implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations // Adv. Eng. Software, 1984, Vol. 6, No. 4, p. 192-197.
6. Toussaint G. Pattern recognition and geometrical complexity // Proc. 5th Int. Conf. of Patt. Recogn. – Miami Beach, 1980, p. 1324-1247.
7. Ильман В.М. Алгоритмы триангуляции плоских областей по нерегулярным сетям точек // Алгоритмы и программы, ВИЭМС. Вып. 10 (88). – М., 1985, с 3-35.

8. Фукс А.Л. Изображение изолиний и разрезов поверхности, заданной нерегулярной системой отсчётов // Программирование. М., 1986, № 4, с. 87-91.
9. Ахо А., Хопкрофт Д., Ульман Д. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 520 с.
10. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 250 с.
11. Кнут Д. Искусство программирования для ЭВМ. Т.3. Сортировка и поиск. – М.: Мир, 1978. – 848 с.
12. Ильман В.М. Экстремальные свойства триангуляции Делоне. // Алгоритмы и программы, ВИЭМС. – М., 1985, Вып. 10 (88), с. 57-66.
13. Lingas A. The Greedy and Delaunay triangulations are not bad… // Lect. Notes Comp. Sc., 1983, Vol. 158, p. 270-284.
14. Костюк Ю.Л., Грибель В.А. Размещение и отображение на карте точечных объектов // Методы и средства обработки сложной графической информации: Тез. докл. Всес. конф. – Горький, 1987, ч. II, с. 60-61.