УДК 519.688:515.142.33

А.В. Скворцов

ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ ПОСТРОЕНИЯ ТРИАНГУЛЯЦИИ ДЕЛОНЕ С ОГРАНИЧЕНИЯМИ

Анализируются итеративные алгоритмы построения триангуляции с ограничениями и без них. Выявляются места, в которых возможна потеря точности вычислений, зачастую приводящие к неверной работе программ. Предлагается явное использование целочисленной арифметики для представления данных и проведения промежуточных вычислений. Приводятся детальные подробности реализации.

Задачи построения триангуляции Делоне и триангуляции Делоне с ограничениями являются одними из базовых в вычислительной геометрии и машинной графике [1]. Триангуляция широко применяется в геоинформатике для моделирования поверхностей и анализа геометрической близости, в машинной графике для построения моделей трехмерных объектов, в различных методах конечных элементов.

В предыдущих работах автором проводились исследования различных алгоритмов построения триангуляции Делоне и триангуляции с ограничениями. В работе [2] обосновывается выбор итеративного алгоритма построения триангуляции Делоне с динамическим кэшированием как наиболее эффективного и простого в реализации среди всех известных автору алгоритмов. Этот алгоритм имеет в среднем линейную относительно количества исходных точек трудоемкость. В работе [3] приводится алгоритм построения триангуляции Делоне с ограничениями, построенный на основе простого итеративного алгоритма.

В приводимых ранее автором описаниях алгоритмов предполагалось, что координаты исходных для построения триангуляции точек и структурных линий представлены в виде вещественных чисел, и так же определена сама структура триангуляции. Никаких специальных оценок потенциальных потерь точности при вычислениях не проводилось [2–3]. Это же неявное допущение о бесконечной точности вещественных вычислений свойственно и всем другим работам, известным автору по данной тематике.

В данной работе в явном виде оцениваются возможные потери точности вычислений, предлагается явный переход от арифметики с плавающей точкой к арифметике с фиксированной точкой (в частности, к целочисленным вычислениям). Также рассматривается задача построения триангуляции Делоне с ограничениями в условиях вставки пересекающихся структурных линий и показывается, что прямая реализация может привести к значительным затратам по времени и даже к зацикливанию алгоритма. Для устранения таких эффектов предлагается явно контролировать точность вычислений, избегая построения слишком близких точек и проведения структурных ребер вблизи узлов триангуляции.

Анализ алгоритмов

В настоящее время наиболее распространенной структурой для представления триангуляции без ограничений является структура, в которой в явном виде представлены точки и треугольники [2, 4, 5]:

```
Точка = record
  X, Y: Координата;
end;
Треугольник = record
  P: array [1..3] of Указатель_на_точку;
  T: array [1..3] of Указатель_на_треугольник;
end;
```

Обычно координаты представляются в виде вещественных чисел.

Рассмотрим основные шаги работы итеративного алгоритма построения триангуляции Делоне [2].

Итак, пусть имеется N исходных входных двумерных вещественных точек (x_i, y_i). Вначале на базе некоторых трех неколлинеарных исходных точек строится первый треугольник, который и образует текущую триангуляцию. Затем все оставшиеся точки поочередно добавляются в триангуляцию. Вставка очередной точки разбивается на два этапа:

1. *Локализация точки в триангуляции*. Выполняется поиск ранее построенного треугольника, в который попадает очередная точка, либо, если точка не попадает в триангуляцию, то определяется ближайший к точке треугольник.

2. *Вставка точки*. Если точка попала строго на ранее вставленную точку, то эта точка отбрасывается. Если точка попала на ребро триангуляции, то треугольники по разную сторону от ребра разбиваются на два новых треугольника. Если точка попала внутрь некоторого треугольника, то он разбивается на 3 новых треугольника. Если точка не попала в триангуляцию, то строится 1 или более новых треугольников. После этого производятся локальные проверки выполнения условия Делоне для всех вновь построенных треугольников.

В задаче построения триангуляции Делоне с ограничениями помимо исходных точек дополнительно задается множество ломаных (структурных линий), при этом требуется, чтобы все эти ломаные проходили строго по ребрам триангуляции, не пересекая треугольники.

Эта задача обычно решается в два этапа. Вначале строится обычная триангуляция Делоне без ограничений на основе всех исходных точек и узловых точек структурных линий. Затем выполняется последовательная вставка отрезков ломаных структурных линий. При этом возможна ситуация, когда очередное вставляемое структурное ребро будет конфликтовать с уже ранее вставленным. В таком случае либо новое ребро отбрасывается, либо уже вставленное ребро и вновь вставляемое разбиваются на две части точкой их пересечения.

Для поддержки понятия структурных линий в структуру триангуляции вводятся дополнительные изменения обычно следующего вида:

```
Точка = record
  X, Y: Координата;
end;
Треугольник = record
  P: array [1..3] of Указатель_на_точку;
  T: array [1..3] of Указатель_на_треугольник;
  R: array [1..3] of Указатель_на_ребро;
end;
```

```
Ребро = record
  P: array [1..2] of Указатель_на_точку;
  T: array [1..2] of Указатель_на_треугольник;
end;
```

При этом ребро в явном виде хранится только для структурных ребер. Если ребра треугольника не являются структурными, то соответствующие ссылки `R[1] = R[2] = R[3] = 0` являются пустыми.

Однако за такой внешней простотой описания алгоритмов скрываются многочисленные детали реализации. Перечислим основные подзадачи:

Задача 1. *Проверка совпадения двух заданных точек*. Эта проблема является особенно актуальной в случае использования вещественной арифметики с плавающей точкой. Как известно, сравнение плавающих вещественных чисел на равенство производится всегда с заданной точностью ε. Здесь определяющим является выбор значения ε.

Несмотря на кажущуюся простоту, данная проблема имеет далеко идущие последствия. Как известно, в силу своей ограниченной точности обычные вещественные вычисления на компьютерах не обладают многими свойствами истинно вещественных чисел. Например, если мы используем числа, хранящиеся в памяти компьютера с помощью 3 значимых цифр, то результат вычисления следующих выражений может не совпадать, т.е. нарушается свойство ассоциативности:

$$(100 + 0{,}5) + 0{,}5 \cong 100 \neq 101 \cong 100 + (0{,}5 + 0{,}5).$$

Задача 2. *Проверка взаимного расположения двух точек относительно прямой*, проходящей через две заданные точки. Данная задача обычно очень просто решается методами аналитической геометрии. Записываем уравнение прямой, проходящей через две заданные точки – (x_1, y_1) и (x_2, y_2):

$$(x_1 - x)(y_2 - y) - (x_2 - x)(y_1 - y) = 0.$$

Затем подставляем в это уравнение вместо x и y координаты тестовых точек (x_3, y_3) и (x_4, y_4). Если значения выражений будут иметь одинаковый знак, то точки находятся по одну сторону от прямой, иначе – по разную. Результат выражения, равный нулю, будет означать попадание точки строго на прямую.

Здесь проблема заключается в потере точности промежуточных вычислений. Перемножая два n-значных числа, вообще говоря, получаем $2n$-значное число. На практике это обычно не учитывается, и младшие n разрядов попросту отбрасываются. В итоге результат вычислений может показать, что тестовая точка лежит на прямой, хотя это не так. Для избавления от этого эффекта сравнение с нулем проводят с некоторой точностью ε. Несмотря на это, реальная точность вычислений все равно уменьшается в 2 раза, составляя не более $n/2$ исходных значащих цифр.

Задача 3. *Проверка коллинеарности трех заданных точек*. Эта задача является частным случаем предыдущей, и ей свойственны те же проблемы с переполнением промежуточных вычислений.

Задача 4. *Проверка взаимного расположения точки и треугольника*. Здесь требуется определить: 1) не совпадает ли точка с одной из вершин треугольника; 2) не попадает ли точка на одно из его ребер; 3) не попадает ли точка строго внутрь треугольника. Новым здесь является проверка попадания точки строго внутрь треугольника. Это решается путем трехкратной проверки взаимного расположения заданной точки относительно различных ребер треугольника, т.е. также сводится к предыдущим задачам.

Задача 5. *Проверка порядка обхода трех заданных точек*. Здесь требуется определить, обходятся ли точки в заданном порядке по часовой стрелке или против. Эту задачу также решаем, записывая уравнение прямой, проходящей через две заданные точки, и подставляя в уравнение координаты третьей точки. После чего анализируем знак выражения. Т.е. если

$$(x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3) < 0,$$

то точки обходятся по часовой стрелке, а если > 0, то против (это верно для левосторонней системы координат, для правосторонней системы все будет наоборот).

В данном алгоритме возникает та же самая проблема переполнения, что и при решении предыдущих задач.

Задача 6. *Проверка выполнения условия Делоне для двух заданных смежных треугольников*. Данная задача может быть решена через классическое определение условия Делоне: внутрь окружности, описанной вокруг любого треугольника, не должны попадать никакие точки триангуляции (рис. 1).

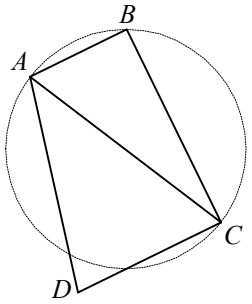

Рис. 1. Проверка условия Делоне

В этом случае записывается уравнение окружности, проходящей через вершины одного из треугольников, и затем подставляется в уравнение противолежащая точка второго треугольника:

$$\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0.$$

На основе этого уравнения можно получить следующий способ определения попадания заданной точки (x_0, y_0) внутрь окружности, описанной вокруг точек (x_1, y_1), (x_2, y_2) и (x_3, y_3). Положим:

$$y_{2-3} = y_2 - y_3,\ y_{3-1} = y_3 - y_1,\ y_{1-2} = y_1 - y_2,$$
$$a = x_1 y_{2-3} + x_2 y_{3-1} + x_3 y_{1-2},$$
$$k = x_1^2 + y_1^2,\ m = x_2^2 + y_2^2,\ n = x_3^2 + y_3^2,$$
$$b = k y_{2-3} + m y_{3-1} + n y_{1-2},$$
$$c = k(x_2 - x_3) + m(x_3 - x_1) + n(x_1 - x_2),$$
$$d = k(x_2 y_3 - x_3 y_2) + m(x_3 y_1 - x_1 y_3) + n(x_1 y_2 - x_2 y_1),$$

$r = a(x_0^2 + y_0^2) - bx_0 + cy_0 - d.$

Тогда если $r \le 0$, то условие Делоне выполняется, иначе – нет. Если исходные координаты точек имели n точных знаков, то для выполнения вычислений указанным способом нам требуется использовать $4n$-значную арифметику, в частности, требуется уметь перемножать $2n$-значные числа для получения $4n$-значных.

Другой способ проверки условия Делоне на основе анализа углов треугольников приведен в [6]:

$x_{2-0} = x_2 - x_0,\ \ y_{2-0} = y_2 - y_0,$

$x_{3-0} = x_3 - x_0,\ \ y_{3-0} = y_3 - y_0,$

$x_{3-1} = x_3 - x_1,\ \ y_{3-1} = y_3 - y_1,$

$x_{2-1} = x_2 - x_1,\ \ y_{2-1} = y_2 - y_1,$

$s = (x_{2-0} + x_{3-0})(y_{2-0} + y_{3-0}),$

$t = (x_{3-1} + x_{2-1})(y_{3-1} + y_{2-1}).$

Если $s < 0$ и $t < 0$, то условие Делоне не выполняется, иначе если $s \ge 0$ и $t \ge 0$, то условие Делоне выполняется, иначе требуются еще дополнительные проверки:

$u = (x_{2-0} - x_{3-0})(y_{2-0} - y_{3-0}),$

$v = (x_{3-1} - x_{2-1})(y_{3-1} - y_{2-1}),\ r = au + bv.$

Если $r \ge 0$, то условие Делоне выполняется, иначе – нет.

Второй способ требует значительно меньшего количества элементарных операций, однако также требует использования $4n$-значной арифметики для вычисления значения r.

В обоих способах если не используется специальная арифметика, то реальная точность вычислений уменьшается в 4 раза, составляя не более $n/4$ исходных значащих цифр.

Задача 7. *Локализация точки в триангуляции*. Локализация точки в триангуляции состоит из выбора некоторого начального треугольника в триангуляции и последовательного перехода по треугольникам к цели. Автору известны два варианта решения данной задачи. Пусть (x_0, y_0) – заданная точка, T – начальный треугольник для поиска.

Вариант 1. Пусть (x, y) – центроид текущего треугольника T (среднее арифметическое координат вершин). Тогда в качестве следующего треугольника T для перехода к цели будем брать тот из трех возможных соседей текущего треугольника, центр которого находится ближе к точке (x_0, y_0), чем текущее значение (x, y).

В этом варианте реальна ситуация, когда определяемый центр треугольника из-за потери точности вычислений оказывается вне самого треугольника, и тогда последующие вычисления теряют смысл и возможно зацикливание. Это вероятно при наличии длинных и узких треугольников, часто образующихся на границе триангуляции, а также вдоль структурных линий. Приведем пример. Пусть точность вычислений составляет 3 знака, тогда центром треугольника T с координатами вершин (100;100), (110;101), (111;101) будет точка (107;101), лежащая вне самого треугольника. Более того, эта точка может лежать даже не внутри соседнего к T треугольника, а гораздо дальше.

Вариант 2. Вычисляется начальная точка (x_1, y_1) как центроид начального треугольника T. Затем последовательно производятся переходы к цели строго вдоль прямой $(x_1, y_1) - (x_0, y_0)$.

В этом варианте возможна та же самая проблема с вычислением центроида на первом шаге, что и в предыдущем, и алгоритм может зациклиться. Также здесь требуется отдельно учесть возможность попадания на отрезок $(x_1, y_1) - (x_0, y_0)$ каких-либо существующих узлов триангуляции.

Задача 8. *Поиск точки пересечения двух прямых*. Данная задача возникает при построении триангуляции Делоне с ограничениями, когда обнаруживается, что очередной вставляемый отрезок пересекается с ранее вставленным структурным ребром триангуляции. Обычно при этом производится вставка новой точки в триангуляцию и разбиение существующего ребра этой точкой на две части. Только после этого вставляется новое ребро, также разбитое на две части.

В данном алгоритме первой проблемой является то, что определяемая точка пересечения в силу ограниченности точности вычислений в большинстве случаев не лежит на ранее существовавшем ребре и не лежит на новом. Возможно, что эта точка лежит даже не в смежных с ребром треугольниках, поэтому в результате разбиения старого ребра на части образуются новые «вывернутые» треугольники, разрушая структуру триангуляции. На рис. 2 приведен пример вставки ребра AB в триангуляцию, приводящий к пересечению с существующим ребром в точке S (пунктирными линиями размечена дискретная координатная сетка). В результате округления точка пересечения окажется немного выше реальной – в узле дискретной координатной сетки.

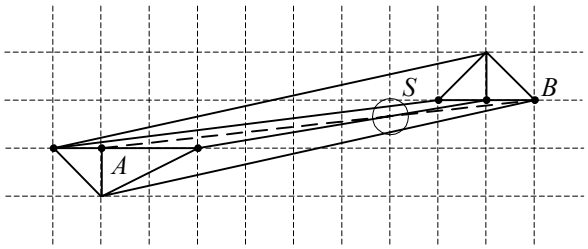

Рис. 2. Пример возможного «выворачивания» треугольников

Несмотря на кажуюся экзотичность приведенного сценария возникновения ошибки вероятность его велика уже при попытке вставить в триангуляцию порядка нескольких десятков взаимно пересекающихся структурных ребер. Вдоль структурных ребер образуются многочисленные узкие вытянутые треугольники, которые и создают указанную критическую ситуацию.

Вторая проблема в задаче поиска точек пересечения связана непосредственно с самим используемым способом вычислений. Обычно точка пересечения (x, y) находится следующим образом:

$a = (x_1 - x_3)(y_4 - y_3) - (y_1 - y_3)(x_4 - x_3),$

$b = (x_4 - x_3)(y_2 - y_1) - (y_4 - y_3)(x_2 - x_1),$

$$x = x_1 + a(x_2 - x_1)/b, \quad y = y_1 + a(y_2 - y_1)/b.$$

Здесь сложности возникают при нахождении точки пересечения двух «почти» коллинеарных отрезков. В результате потери точности возможно значительное смещение найденных координат от реального значения. Кроме того, из-за потерь точности мы можем предполагать, что отрезки пересекаются, хотя это не так. Тогда попытка вычисления их пересечения может привести к значительному удалению найденной точки от самих отрезков (для «почти» коллинеарных отрезков).

Предлагаемые модификации

Таким образом, большинство поставленных проблем связано с потерей точности внутренних вычислений.

Контролировать точность, используя стандартные вещественные типы данных, предлагаемые большинством распространенных языков программирования, весьма сложно. Большинство современных компьютеров поддерживают стандарты ANSI представления вещественных чисел, однако даже 10-байтовый тип extended позволяет хранить не более 20 значащих цифр. В то же время при описании 6-й задачи показано, что для корректных вычислений требуется $4n$-значная арифметика. Это означает, что реальная достижимая точность построения триангуляции Делоне составляет не более $20/4 = 5$ знаков в задании координат исходных данных. Т.е. значение точности ε для проверки совпадения двух точек, возникшее при описании 1-й задачи, следует установить не менее чем 10^{-5}, что не всегда приемлемо на практике.

Другой способ заключается в использовании в явном виде вычислений с фиксированной точкой. В этом случае можно точно контролировать все потери точности.

Еще проще является переход к целочисленному представлению координат исходных точек. Например, используя обычные 32-битные целые числа, можно обеспечить точность представления в 9 значащих цифр, что является уже приемлемым в большинстве ситуаций.

Тогда для реализации алгоритма построения триангуляции понадобится реализовать несколько дополнительных функций, оперирующих с 32-, 64- и 128-битными числами. Выполненная на платформе IA-32 автором реализация алгоритма включала в себя следующие функции:

1. **`Mul64(A,B).`** Умножение 32-разрядных чисел. Результат возвращается 64-разрядным. Функция реализована как одна команда ассемблера.

2. **`Sqr64(A).`** Возведение 32-разрядного числа в квадрат. Результат возвращается 64-разрядным. Функция реализована также как одна команда ассемблера.

3. **`MulSum64(A,B,C,D).`** Сумма двух произведений 32-разрядных чисел A·B и C·D. Результат возвращается 64-разрядным. Функция реализована с помощью 9 команд ассемблера.

4. **`MulDif64(A,B,C,D).`** Разность произведений 32-разрядных чисел A·B и C·D. Результат возвращается 64-разрядным. Функция реализована с помощью 11 команд ассемблера.

5. **`Mul128(A,B).`** Умножение 64-разрядных чисел. Результат возвращается 128-разрядным. Функция реализована с помощью 40 команд ассемблера.

6. **`Compare128(A,B,C,D).`** Вначале вычисляется сумма двух произведений 64-разрядных чисел A·B и C·D. Затем получаемое 128-разрядное число сравнивается с нулем. Если оно меньше нуля, то возвращается false, иначе – true. Функция реализована с помощью двух вызовов функции **`Mul128`** и дополнительных 13 команд ассемблера.

(Безусловно, использование 64-битных микропроцессоров могло бы существенно сократить реализацию до 1–4 команд ассемблера для каждой функции).

Таким образом, используя целочисленный способ представления исходных данных совместно с дополнительными операциями над 32-, 64- и 128-битными целыми числами, можно чётко решить поставленные задачи.

Если используются целочисленные вычисления, то отпадает необходимость использования величин ε, возникающих в задачах 1 и 3, и можно выполнять проверки на равенство непосредственно.

Теперь рассмотрим описанную в 8-й задаче проблему поиска точек пересечения и разбиения вставляемых структурных ребер на части.

Несмотря на то, что мы можем выполнять вычисления с помощью дополнительных функций практически без потери точности, все равно точка пересечения двух прямых в общем случае будет иметь нецелые координаты, которые мы будем вынуждены округлить, т.е. в общем случае точка пересечения двух прямых не будет лежать на этих прямых.

Таким образом, встает необходимость модификации алгоритма вставки структурных линий так, чтобы учесть возможные нарушения структуры триангуляции и избавиться от них. Автором предлагается следующий алгоритм вставки структурных линий.

Алгоритм вставки структурных линий в триангуляцию с ограничениями. Пусть L – сортированный по длине список еще не вставленных отрезков структурных линий. Вначале в L заносим все отрезки исходных структурных линий. Пока список L не пуст, извлекаем (с удалением) из этого множества самый длинный отрезок AB. Далее пытаемся вставить этот отрезок в триангуляцию методом, описанным в [3]. Если обнаруживается, что вставляемый отрезок пересекает некоторые ранее вставленные структурные ребра, то их необходимо удалить из триангуляции. При этом надо найти все точки пересечения C_i, где $i = \overline{1,n}$, нового отрезка со вставленными ребрами E_iF_i. Затем надо в список L поместить все отрезки C_iC_{i+1}, где $i = \overline{0,n}$, $C_0 = A$, $C_{n+1} = B$. Также необходимо туда поместить все отрезки E_iC_i и C_iF_i, где $i = \overline{1,n}$. Если вставляемый в список отрезок имеет нулевую длину, то не вставляем его. *Конец алгоритма*.

В проводимом автором исследовании данного алгоритма достаточно часто возникала ситуация,

когда небольшое количество исходных структурных линий приводит к значительному разрастанию списка L в процессе работы. Например задав 5 «почти» коллинеарных отрезков в качестве исходных структурных линий, алгоритм в конце концов выдавал более 15 тысяч структурных ребер. Происходит это вследствие того, что каждая пара отрезков после нахождения их пересечений образует 4 новых отрезка, также являющихся «почти» коллинеарными остальным отрезкам. Такое дробление идет до тех пор, пока размеры отрезков не станут сравнимыми с размерами единицы координатной сетки, когда маленькие отрезки становятся «совсем не» коллинеарными или размер отрезков становится настолько малым, что его дальнейшее деление невозможно. Но и на этом микроуровне возможны проблемы. Например, пусть построена некоторая «плотная» триангуляция, когда в каждом узле координатной сетки имеется по узлу триангуляции и требуется вставить отрезок AB, который пересекается с ранее вставленным структурным ребром CD (рис. 3). В результате найденная точка пересечения AB и CD будет округлена, например, до точки A. В итоге в список L попадут отрезки AC, AD и опять AB. Так как мы извлекаем на каждом шаге из списка самое большое ребро, то он будет бесконечно разрастаться за счет постоянной вставки ребер AC и AD.

Таким образом, в этом варианте алгоритм все же не годится для практической работы.

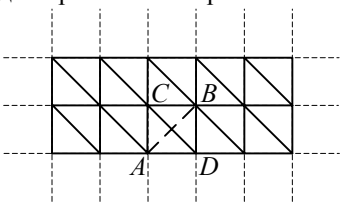

Рис. 3. Попытка вставки микроребра

Во избежание пересечения пар «почти» коллинеарных отрезков и проблем на микроуровне автором предлагается две модификации алгоритма.

Во-первых, при вставке очередного отрезка необходимо найти все узлы триангуляции, лежащие вблизи от отрезка на расстоянии не более некоторого ε_1. Тогда, если такие точки найдены, вставляемый отрезок разобьем этими точками на части, которые поместим в список L.

Во-вторых, найдя точку пересечения структурных ребер, прежде чем вставлять новый узел попробуем найти в окрестности радиуса ε_2 другой, ранее уже вставленный узел триангуляции.

В проведенном автором экспериментальном моделировании работы алгоритма значительное улучшение наблюдалось уже при значениях $\varepsilon \geq 3$. Увеличение ε приводит к сокращению размера списка L, но несколько увеличивает время выполнения дополнительного поиска точек в окрестностях. Экспериментально были выбраны наиболее приемлемые с точки зрения быстродействия и качества значения $\varepsilon_1 = \varepsilon_2 = 10$.

Заключение

Использование целочисленного представления исходных чисел позволяет, с одной стороны, явно контролировать точность вычислений, с другой – повысить скорость работы алгоритма построения триангуляции за счет отказа от вещественных операций.

Тем не менее простой переход от вещественных вычислений к целочисленным приводит к другому неприятному эффекту – значительному росту количества структурных ребер в триангуляции. Во избавление этого приходится несколько усложнять алгоритм, вводя дополнительные проверки на наличие совпадающих узлов триангуляции с заданной точностью ε.

ЛИТЕРАТУРА

1. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение / Пер. с англ. М.: Мир, 1989. 478 с.
2. *Скворцов А.В., Костюк Ю.Л.* Эффективные алгоритмы построения триангуляции Делоне // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Том. ун-та, 1998. С. 22–47.
3. *Скворцов А.В., Костюк Ю.Л.* Применение триангуляции для решения задач вычислительной геометрии // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Том. ун-та, 1998. С. 127–138.
4. *Lee D., Schachter B.* Two algorithms for constructing a Delaunay triangulation // Int. Jour. Comp. and Inf. Sciences. 1980. V.9. № 3. P. 219–242.
5. *Shapiro M.* A note on Lee and Schachter's algorithm for Delaunay triangulation // Inter. Jour. Of Comp. and Inf. Sciences. 1981. V.10. № 6. P. 413–418.
6. *Фукс А.Л.* Предварительная обработка набора точек при построении триангуляции Делоне // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Том. ун-та, 1998. С. 48–60.

Статья представлена кафедрой теоретических основ информатики факультета информатики Томского государственного университета, поступила в научную редакцию 3 декабря 2001 г.