Инкрементальный алгоритм построения триангуляции Делоне

По книге Ласло,
"Вычислительная геометрия и компьютерная графика на C++"

Определение триангуляции Делоне

Триангуляция для конечного набора точек S является задачей триангуляции выпуклой оболочки CH(S), охватывающей все точки набора S. Отрезки прямых линий при триангуляции не могут пересекаться — они могут только встречаться в общих точках, принадлежащих набору S. Поскольку отрезки прямых линий замыкают треугольники, мы будем считать их ребрами. На рис. 1 показаны два различных варианта триангуляции для одного и того же набора точек (временно проигнорируем окружности, проведенные на этих рисунках).

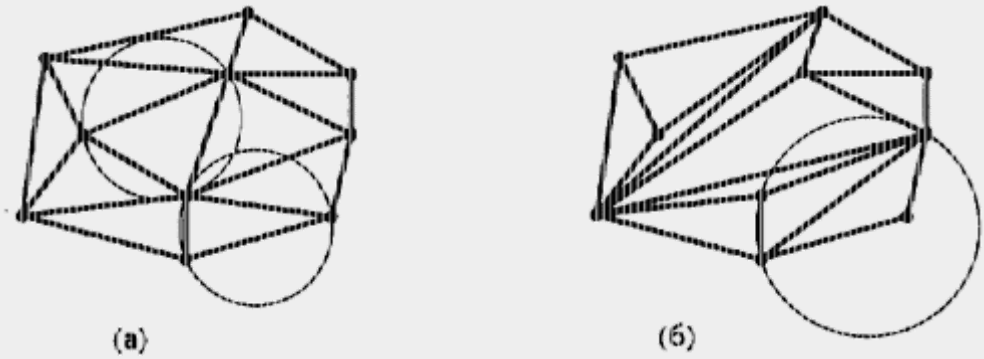

Рис. 1: Два различных варианта триангуляции мя одного набора точек

Для данного набора точек S мы можем видеть, что все точки из набора S могут быть подразделены на граничные точки — те точки, которые лежат на границе выпуклой оболочки CH(S), и внутренние точки — лежащие внутри выпуклой оболочки CH(S). Также можно классифицировать и ребра, полученные в результате триангуляции S, как *ребра оболочки* и *внутренние ребра*. К ребрам оболочки относятся ребра, расположенные вдоль границы выпуклой оболочки CH(S), а к внутренним ребрам — все остальные ребра, образующие сеть треугольников внутри выпуклой оболочки. Отметим, что каждое ребро оболочки соединяет две соседние граничные точки, тогда как внутренние ребра могут соединять две точки любого типа. В частности, если внутреннее ребро соединяет две граничные точки, то оно является хордой выпуклой оболочки CH(S). Заметим также, что каждое ребро триангуляции является границей двух областей: каждое внутреннее ребро находится между двумя треугольниками, а каждое ребро оболочки — между треугольником и бесконечной плоскостью.

Любой набор точек, за исключением некоторых тривиальных случаев, допускает более одного способа триангуляции. Но при этом существует замечательное свойство: любой способ триангуляции для данного набора определяет одинаковое число треугольников, что следует из теоремы:

Теорема о триангуляции набора точек. Предположим, что набор точек S содержит n>3 точек и не все из них коллинеарны. Кроме того, i точек из них

являются внутренними (т. е. лежащими внутри выпуклой оболочки CH(S). Тогда при любом способе триангуляции набора S будет получено точно n + i - 2 треугольников.

Для доказательства теоремы рассмотрим сначала триангуляцию n-i граничных точек. Поскольку все они являются вершинами выпуклого полигона, то при такой триангуляции будет получено (n - i) - 2 треугольников. (В этом нетрудно удостовериться и, более того, можно показать, что любая триангуляция *произвольного* m-стороннего полигона - выпуклого или невыпуклого — содержит m - 2 треугольника). Теперь проверим, что будет происходить с триангуляцией при добавлении оставшихся i внутренних точек, каждый раз по одной. Мы утверждаем, что добавление каждой такой точки приводит к увеличению числа треугольников на два. При добавлении внутренней точки могут возникнуть две ситуации, показанные на рис. 2. Во-первых, точка может оказаться внутри некоторого треугольника и тогда такой треугольник заменяется тремя новыми треугольниками. Во-вторых, если точка совпадает с одним из ребер триангуляции, то каждый из двух треугольников, примыкающих к этому ребру, заменяется двумя новыми треугольниками. Из этого следует, что после добавления всех г точек, общее число треугольников составит (n - i - 2) + (2i), или просто n + i - 2.

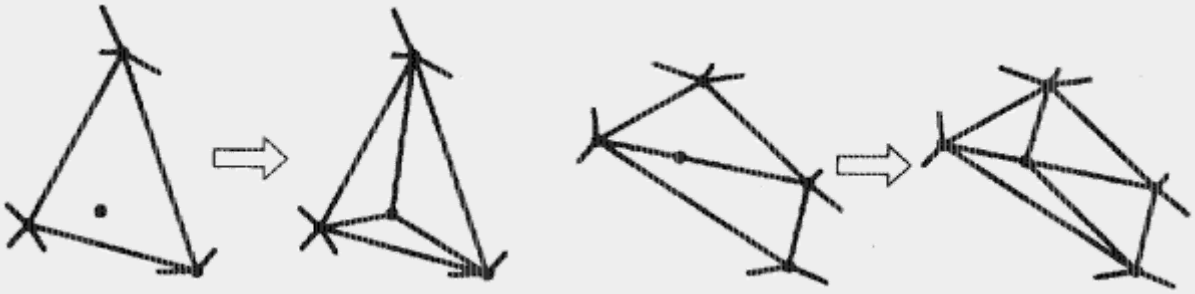

Рис. 2: Две ситуации, возникающие при триангуляции после добавления новой внутренней точки

В этом разделе мы представим алгоритм формирования специального вида триангуляции, известный как триангуляция Делоне. Эта триангуляция хорошо сбалансирована в том смысле, что формируемые треугольники стремятся к равноугольности. Так например, триангуляцию, изображенную на рис. 1а, можно отнести к типу триангуляции Делоне, а на рис. 1б триангуляция содержит несколько сильно вытянутых треугольников и ее нельзя отнести к типу Делоне. На рис. 3 показан пример триангуляции Делоне для набора большого числа точек.

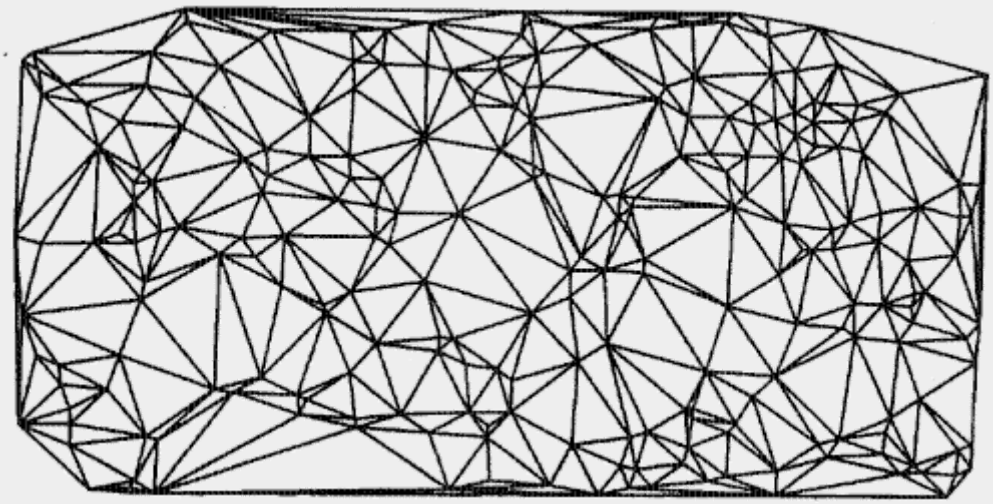

Рис. 3: Триангуляция Делоне для 250 точек, выбранных случайным образом в пределах прямоугольника. Всего образовано 484 треугольника

Для формирования триангуляции Делоне нам потребуется несколько новых определений. Набор точек считается круговым, если существует некоторая

окружность, на которой лежат все точки набора. Такая окружность будет описанной для данного набора точек. Описанная окружность для треугольника проходит через все три ее (не коллинеарные) вершины. Говорят, что окружность будет свободной от точек в отношении к заданному набору гочек S, если внутри окружности нет ни одной точки из набора S. Но, однако, точки из набора S могут располагаться на самой свободной от точек окружности.

Триангуляция набора точек S будет триангуляцией Делоне, если описанная окружность для каждого треугольника будет свободна от точек. На схеме триангуляции рис. 1а показаны две окружности, которые явно не содержат внутри себя других точек (можно провести окружности и для других треугольников, чтобы убедиться, что они также свободны от точек набора). Это правило не соблюдается на схеме рис. 16 — внутрь проведеной окружности попала одна точка другого треугольника, следовательно, эта гриангуляция не относится к типу Делоне.

Можно сделать два предположения относительно точек в наборе S, чтобы упростить алгоритм триангуляции. Во-первых, чтобы вообще существовала триангуляция, мы должны полагать, что набор S содержит по крайней мере три точки и они не коллинеарны. Во-вторых, для уникальности триангуляции Делоне необходимо, чтобы никакие четыре точки из набора S не лежали на одной описанной окружности. Легко видеть, что без такого предположения гриангуляция Делоне не будет уникальной, ибо 4 точки на одной описанной окружности позволяют реализовать две различные триангуляции Делоне.

Наш алгоритм работает путем постоянного наращивания текущей триангуляции по одному треугольнику за один шаг. Вначале текущая триангуляция состоит из единственного ребра оболочки, по окончании работы алгоритма текущая триангуляция становится триангуляцией Делоне. На каждой итерации алгоритм ищет новый треугольник, который подключается к *границе* текущей триангуляции.

Определение границы зависит от следующей схемы классификации ребер триангуляции Делоне относительно текущей триангуляции. Каждое ребро может быть *спящим*, *живым* или *мертвым*:

- *спящие ребра*: ребро триангуляции Делоне является спящим, если она еще не было обнаружено алгоритмом;
- *живые ребра*: ребро живое, если оно обнаружено, но известна только одна примыкающая к нему область;
- *мертвые ребра*: ребро считается мертвым, если оно обнаружено и известны обе примыкающие к нему области.

Вначале живым является единственное ребро, принадлежащее выпуклой i лочке — к нему примыкает неограниченная плоскость, а все остальные ребра спящие. По мере работы алгоритма ребра из спящих становятся живыми, затем мертвыми. Граница на каждом этапе состоит из набора живых ребер.

На каждой итерации выбирается любое одно из ребер е границы и оно подвергается обработке, заключающейся в поиске неизвестной области, ко торой принадлежит ребро е. Если эта область окажется треугольником f, определяемым концевыми точками ребра е и некоторой третьей вершинов v, то ребро е становится мертвым, поскольку теперь известны обе примыкающие к нему области. Каждое из двух других ребер треугольника t переводятся в следующее состояние: из спящего в живое или из живого в мертвое. Здесь вершина v будет называться *сопряженной* с ребром е. противном случае, если неизвестная область

оказывается бесконечной плоскостью, то ребро e просто умирает. В этом случае ребро e не имеет сопряженной вершины.

На рис. 4 показана работа алгоритма, где действие происходит сверху вниз и слава направо. Граница на каждом этапе выделена толстой линией.

Алгоритм реализован в программе delaunayTriangulate. Программе задается массив s из n точек и она возвращает список треугольников, представляющих триангуляцию Делоне. Реализация использует класс кольцевого списка и классы из раздела структуры геометрических данных. В качестве класса Dictionary можно использовать любой словарь, поддерживающий требуемые операции. Например, можно переопределить #define Dictionary RandomizedSearchTree.

```
List<Polygon*> * (Point s[], int n)
{
  Point p;
  List<Polygon*> *triangles = new List<Polygon*>;
  Dictionary<Edge*> frontier(edgeCmp);
  Edge *e = hullEdge(s, n);
  frontier.insert(e);
  while (!frontier.isEmpty()) {
    e = frontier.removeMin();
    if (mate(*e, s, n, p)) {
      updateFrontier(frontier, p, e->org);
      updateFrontier(frontier, e->dest, p);
      triangles->insert(triangle(e->org, e->dest, p));
    }
    delete e;
  }
  return triangles;
}
```

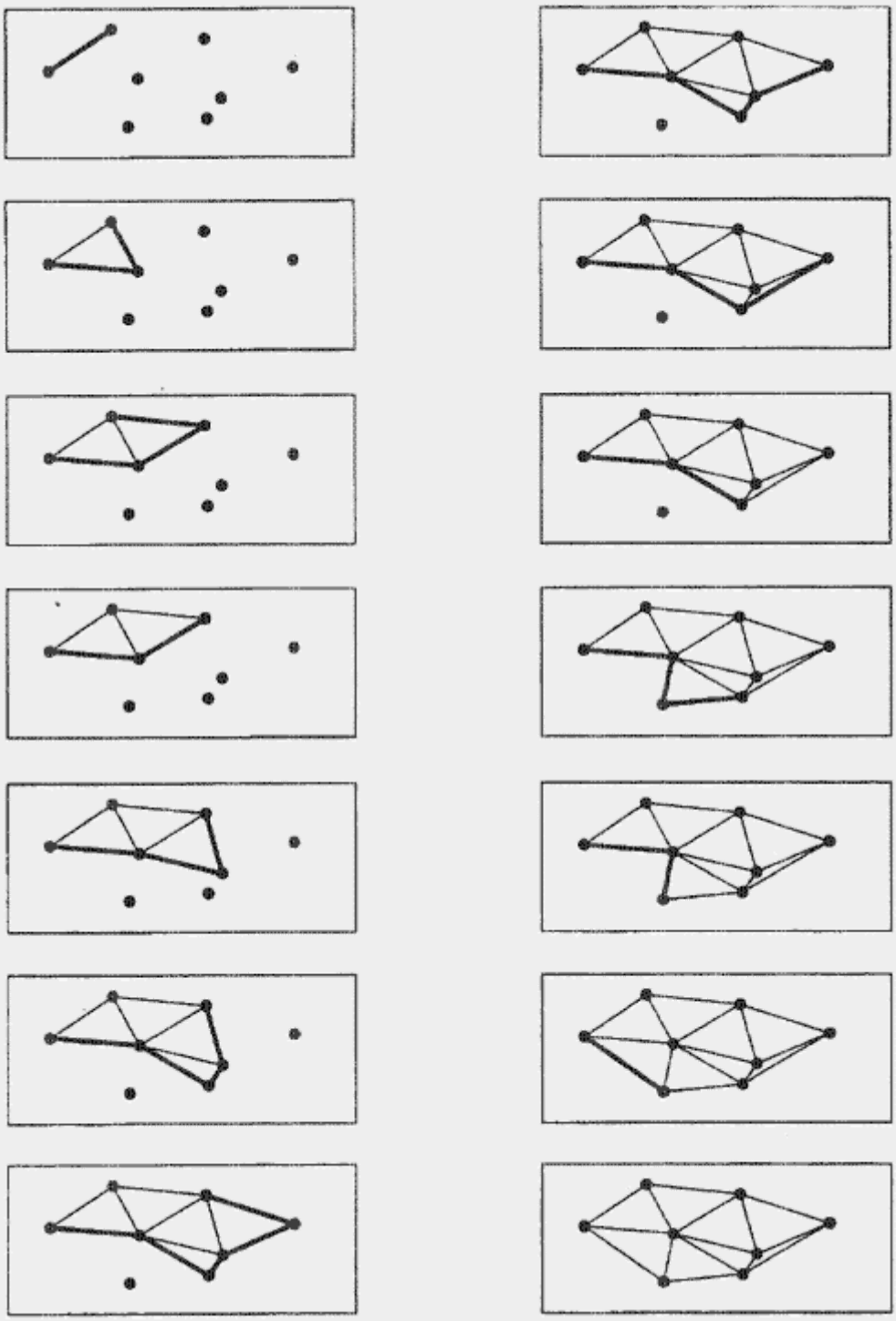

Рис. 4: Наращивание триангуляции Делоне. Ребра, входящие в состав границы, выделены толстой линией

Треугольники, образующие триангуляцию, записываются в список triangles. Граница представлена словарем frontier живых ребер. Каждое ребро направлено, так что неизвестная область для него (подлежащая определению) лежит справа от ребра. Функция сравнения edgeCmp используется для просмотра словаря. В ней сравниваются начальные точки двух ребер, если они оказываются равными, то потом сравниваются их концевые точки:

```
int edgeCmp (Edge *a, Edge *b)
{
  if (a->org < b->org) return 1;
  if (a->org > b->org) return 1;
  if (a->dest < b->dest) return -1;
  if (a->dest > b->dest) return 1;
  return 0;
}
```

Как же изменяется граница от одного шага к другому и как функция updateFrontier изменяет словарь ребер границы для отражения этих изменений? При подсоединении к границе нового треугольника t изменяются состояния трех ребер треугольника. Ребро треугольника t, примыкающее к границе, из живого становится мертвым. Функция updateFrontier может игнорировать это ребро, поскольку оно уже должно быть удалено из словаря при обращении к функции removeMin. Каждое из двух оставшихся ребер треугольника t изменяют свое состояние из спящего на живое, если они уже ранее не были записаны в словарь, или из живого в мертвое, если ребро уже находится в словаре. На рис. 5 показаны оба случая. В соответствии с рисунком мы обрабатываем живое ребро af и, после обнаружения, что точка b является сопряженной ему, добавляем треугольник afb к текущей триангуляции. Затем ищем ребро fb в словаре и, поскольку его там еще нет и оно обнаружено впервые, его состояние изменяется от спящего к живому. Для редактирования словаря мы повернем ребро fb так, чтобы примыкающая к нему неизвестная область лежала справа от него и запишем это ребро в словарь. Затем отыщем в словаре ребро ba — поскольку оно есть в нем, то оно уже живое (известная примыкающая к нему область — треугольник abc). Так как неизвестная для него область, треугольник afb, только что была обнаружена, это ребро удаляется из словаря.

Функция updateFrontier редактирует словарь frontier, в котором изменяется состояние ребра из точки a в точку b:

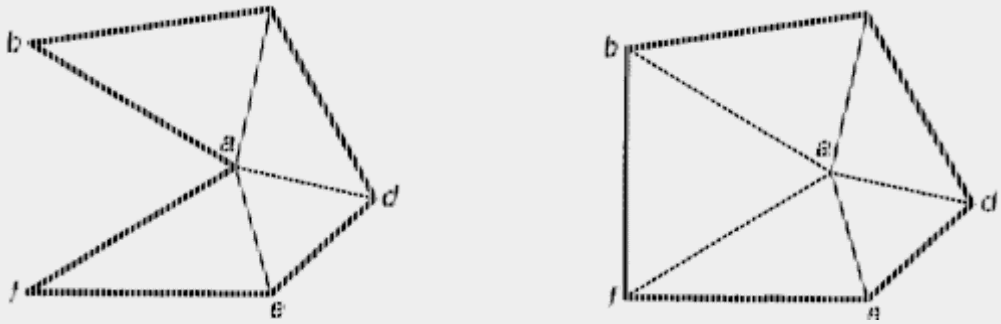

Рис. 5: Подключение треугольника afb к живому ребру at

```
void updateFrontier (Dictionary<Edge*> &frontier, Point &a, Point &b)
{
  Edge *e = new Edge (a, b);
  if (frontier.find (e))
    frontier.remove(e);
  else {
    e->flip();
    frontier.insert(e);
  }
}
```

Функция hullEdge обнаруживает ребро оболочки среди n точек массива s. В этой функции фактически применяется этап инициализации и первой итерации метода заворачивания подарка:

```
Edge *hullEdge (Point s[], int n)
{
  int m = 0;
  for (int i = 1; i < n; i++)
    if (s[i] < s[m])
      m = i;
  swap(s[0], s[m]);
  for (m = 1, i = 2; i < n; i++) {
    int c =  s[i].classify (s[0], s[m]);
```

```
    if ( (c  ==  LEFT) || (C  == BETWEEN) )
      m = i;
  }
  return new Edge(s[0], s[m] );
}
```

Функция triangle просто формирует и возвращает полигон для трех точек, передаваемых ей в качестве параметров:

```
Polygon  *triangle (Point  &a,   Point  &b,   Point  &c)
{
  Polygon *t = new Polygon;
  t->insert (a);
  t->insert (b);
  t->insert (c);
  return t;
}
```

Поиск сопряженной точки для ребра

Обратим внимание на задачу, решаемую функцией mate, которая определяет, существует ли для данного живого ребра сопряженная точка и, если она есть, находит ее. Рассмотрим определение: любое ребро ab определяет бесконечное семейство окружностей, проходящих через его концевые точки a и b. Обозначим это семейство окружностей через C (a, b) (рис. 6).

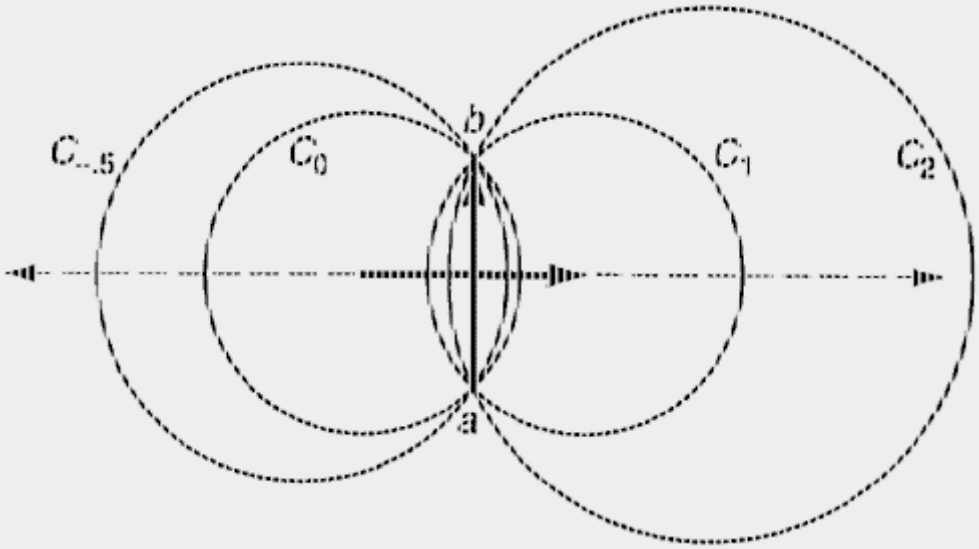

Рис. 6: Четыре окружности из семейства C(a,b), определяемого ребром ab и их параметрические значения

Центры окружностей семейства C (a, b) лежат на прямой линии, перпендикулярной отрезку ab и проходящей через его центр, и для них можно установить однозначное соотношение с точками на этом перпендикуляре. Для спецификации окружностей семейства параметризуем перпендикуляр — припишем каждой окружности параметрическое значение положения ее центра. Средства, описанные в разделе о структурах геометрических данных, позволяют определить естественную параметризацию: ребро ab поворачивается вокруг своей средней точки на 90 градусов до совпадения с перпендикуляром и затем можно использовать параметрические значения точек вдоль этого ребра. На рис. 6 используется запись C_r для обозначения окружности, соответствующей параметрическому значению r.

Как же найти сопряженную точку для некоторого живого ребра ab среди множества точек набора S? Предположим, что окружность C_r является описанной окружностью для известной области ребра ab (на рис. 7 такой известной областью будет треугольник abc). Если известная область для ребра ab является неограниченной, то r = -oo и C_r представляет собой полуплоскость, лежащую слева от ab. Нам нужно найти такое наименьшее значение t > r, чтобы некоторая точка из набора S (отличная от точек a и b), принадлежала окружности C_t. Если не существует такого значения t, то ребро ab не имеет сопряженной точки. Более образно это соответствует надуванию двухмерного пузыря, привязанного к отрезку ab. Если такой пузырь достигает некоторой точки из набора S, то эта точка является сопряженной отрезку ab (точка d на рис. 7). В противном случае, если не встретится ни одной такой точки из S, то пузырь разрастается до заполнения всей бесконечной полуплоскости справа от отрезка ab и тогда отрезок ab не имеет сопряженной точки.

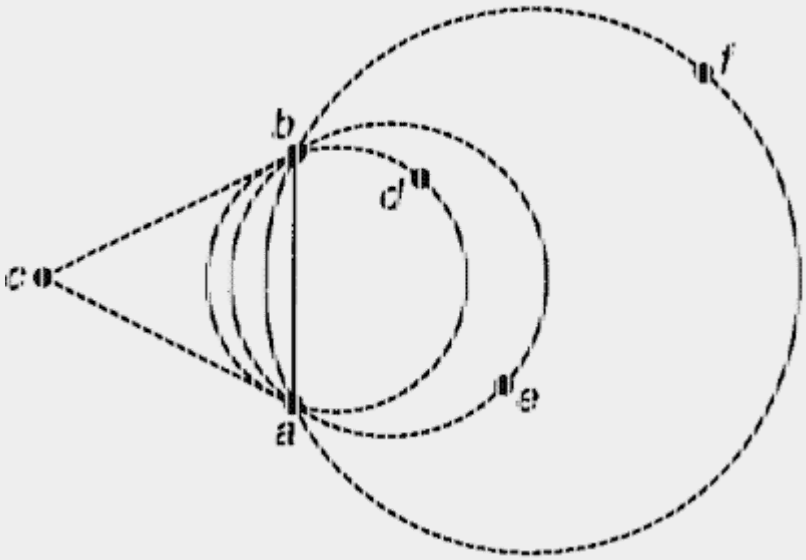

Рис. 7: Определение сопряженной точки (d) для ребра ab

Почему же работает такой алгоритм? Пусть C_r обозначает описанную окружность известной области отрезка ab и C_t — описанную окружность неизвестной области отрезка ab. Здесь t > r и t = oo, если отрезок ab не имеет сопряженной точки. Будет ли окружность C_t свободной от точек, что нам нужно? Слева от отрезка ab окружность C_t должна быть свободной от точек, поскольку C_r свободна от точек, а часть C_t, лежащая слева от ребра ab, юходится внутри C_r. Справа от ребра ab C_t также должна быть свободна от точек, поскольку если бы некоторая точка q попала бы внутрь этой окружности, то она бы принадлежала окружности C_s из C(a,b), где r < s < t, что противоречило бы нашему выбору t. В нашей аналогии с пузырем расширяющийся пузырь достиг бы точки q до того, как он достиг сопряженной гечки ребра ab.

При поиске сопряженной точки для ребра ab мы будем рассматривать только те точки p из S, которые лежат справа от ab. Центр окружности, описанной вокруг любых трех точек a, b, и c, лежит на пересечении перпендикуляров, проведенных через середины отрезков ab и bp. (Здесь используется тот факт, что перпендикуляры в серединах ребер треугольников пересекаются в центре описанной окружности треугольника.) Вместо вычисления положения центра окружности мы будем вычислять его параметрическое значение вдоль перпендикуляра к середине ребра ab. Таким образом мы можем осуществить поиск наименьшего параметрического значения.

Эта методика реализована в функции mate, которая возвращает значение TRUE, если ребро e имеет сопряженную точку, и FALSE, если такой точки нет. Если сопряженная точка существует, то она возвращается через ссылочный параметр p:

```
bool mate (Edge &e, Point s[], int n, Point  &p)
{
  Point *bestp = NULL;
  double  t, bestt = FLT_MAX;
  Edge  f = e;
  f.rot();      // f - перпендикуляр в середине отрезка e
  for (int i = 0; i < n; i++)
    if (s[i].classify(e) == RIGHT) {
      Edge g(e.dest, s[i]);
      g.rot();
      f.intersect (g, t);
      if (t < bestt) {
        bestp = &s[i];
        bestt = t;
      }
    }
  if (bestp) {
    p = *bestp;
    return TRUE;
  }
return FALSE;
}
```

В функции mate переменная bestp указывает на самую лучшую точку, найденную к данному моменту, а в переменной bestt хранится параметрическое значение для окружности, которое проходит через эту точку. Напомним, что при этом анализируются только те точки, что лежат справа от ребра e.

Этот алгоритм для вычисления триангуляции Делоне по набору из n точек выполняется за время $O(n^2)$, поскольку при каждой итерации из границы исключается одно ребро. Поскольку каждое ребро исключается из границы точно однажды — каждое ребро относится к границе однажды и затем исключается из нее, никогда не возвращаясь — число итераций равно числу ребер в триангуляции Делоне. Согласно теореме о триангуляции набор точек любая триангуляция содержит не более, чем O(n) ребер, поэтому алгоритм выполняет O(n) итераций. Поскольку на каждую итерацию тратится время O(n), то полностью алгоритм выполняется за время $O(n^2)$.