Модифицированные алгоритмы Форчуна и Ли скелетизации многоугольной фигуры.

Денис Лагно, Андрей Соболев
Механико-математический факультет МГУ
Москва, Россия

Аннотация

Заметка основана на опыте реализации алгоритмов Форчуна и Ли в библиотеке Open Source Computer Vision Library. Описывается ряд внесенных модификаций, которые оказываются полезными для практической реализации данных алгоритмов, а также ускоряют их работу. Дается сравнительная оценка алгоритмов.

Ключевые слова: Диаграмма Вороного, скелетон.

1. ВВЕДЕНИЕ.

Изначально нашей задачей было написание для практических целей функций скелетизации многоугольников с «дырками» и без «дырок». Среди различных алгоритмов одним из авторов (А.С.) был выбран алгоритм Ли [3], главным достоинством которого является простота – простота идеи и реализации. Параллельно соавтором разрабатывался алгоритм скелетизации многоугольника с «дырками», основанный на алгоритме Форчуна [1]. После реализации обоих алгоритмов было естественно сравнить их быстродействие на многоугольниках без «дырок»: алгоритм Ли постоянно был быстрее алгоритма Форчуна, причем разница в скорости колебалась от 5% до нескольких десятков процентов. Этот результат заставил задуматься над следующим вопросом: можно ли распространить алгоритм Ли на многоугольники с «дырками» без потери его эффективности? Одной из целей этой заметки является попытка ответить на этот вопрос.

2. АЛГОРИТМ ЛИ.

Все понятия, используемые в этой заметке при описании обобщенного алгоритма Ли, заимствованы из статьи Ли [3]. Кратко напомним основные из них.

Многоугольником с «дырками», или обобщенным многоугольником (ОМ), называется ограниченная область в R^2 , граница которой состоит из конечного числа непересекающихся простых многоугольников. Через P_0 обозначим многоугольник, ограничивающий заданный ОМ снаружи, P_0 задается на плоскости последовательностью своих вершин, взятых в таком порядке, что при обходе их против часовой стрелки внутренность P_0 остается слева по направлению обхода. Через $P_1,...,P_k$ обозначим набор из k многоугольников, задающих «дырки», здесь $k+1$ – число компонент связности ОМ. Каждый из них задается последовательностью своих вершин, взятых в таком порядке, что при обходе их по часовой стрелке внутренность ОМ остается слева по направлению обхода (соответственно, внутренность самой дырки остается справа). Таким образом, $k+1$ – связный ОМ однозначно задается набором $P_0 ,P_1,...,P_k$. Далее набор многоугольников $P_0 ,P_1,...,P_k$ мы также будем называть ОМ.

Скелетом ОМ называется множество всех внутренних точек ОМ, имеющих не менее двух ближайших точек границы. *Сайтом ОМ* (ниже просто *сайтом*) будем называть либо ребро многоугольника P_i, $i=0,...,k$, либо вогнутую вершину многоугольника P_i, $i=0,...,k$. Вершину многоугольника P_i, $i=0,...,k$, будем называть вогнутой, если угол при этой вершине, обращенный во внутренность ОМ, больше 180^0. *Цепочкой сайтов*, или *цепочкой*, будем называть любую совокупность сайтов. *Бисектором* двух сайтов будем называть множество точек, равноудаленных от этих сайтов (в частном случае, когда один сайт является отрезком, а второй сайт – концевой точкой этого отрезка, бисектором будем называть прямую, ортогональную отрезку и проходящую через сайт – точку).

Пусть на плоскости задана совокупность S сайтов. *Ячейкой Вороного* (ЯВ) сайта $s \in S$ называется множество точек плоскости, для которых этот сайт является ближайшим (ЯВ – открытое множество). *Диаграммой Вороного* (ДВ) совокупности сайтов S называется множество всех граничных точек ячеек Вороного сайтов из S.

Теорема 1. *Скелет ОМ является подмножеством ДВ совокупности построенных по ОМ сайтов.*

Граница ЯВ произвольного сайта представляет собой последовательность отрезков прямых и парабол, которые мы будем называть ребрами ЯВ. Заметим, что одновременно они являются ребрами ДВ. Ребра ЯВ данного сайта мы будем называть инцидентными этому сайту, соответственно сайт мы будем называть инцидентным ребру ДВ, если это ребро инцидентно сайту. Понятно, что каждое ребро инцидентно двум сайтам. Два сайта, инцидентные одному ребру, будем называть смежными. Граничные точки ребер ДВ будем называть узлами ДВ. Распространяя по транзитивности на узлы ДВ понятие инцидентности, будем говорить, что узел инцидентен сайту, если он является граничной точкой ребра ДВ, инцидентного этому сайту. Каждый узел ДВ инцидентен минимум трем сайтам.

Известно, что алгоритм Ли и алгоритм Форчуна являются $O(n \log n)$ алгоритмами. Главная причина отставания алгоритма Форчуна в реальных приложениях состоит в том, что в этом алгоритме асимптотическая оценка - $O(n \log n)$ операций - верна для любого ОМ, тогда как для алгоритма Ли эта оценка является наихудшей, в среднем же скелетизация простого многоугольника выполняется за $O(n)$ операций. Практически важной представляется задача обобщения алгоритма Ли на ОМ с сохранением этой оценки. Подобное обобщение уже было сделано в статье [4], авторы

получили O(N (log N + k)) алгоритм, где k – количество «дырок». В этой заметке мы предлагаем алгоритм с аналогичной оценкой числа операций, основное отличие от алгоритма [4] состоит в отсутствии необходимости обобщения алгоритма Ли на случай объединения произвольных цепочек сайтов.

Авторы считают своим долгом обратить внимание на критику, которой подвергался алгоритм Ли в литературе, например в работе [5], где указывается на некорректность алгоритма. Действительно, в статье Ли изложена лишь схема алгоритма, которую при реализации надо дорабатывать. Действительно, промежуточные построения для цепочек, которые Ли называет диаграммами Вороного для этих цепочек, на самом деле диаграммами Вороного не являются. Тем не менее, большая практическая работа, проведенная при реализации этого алгоритма, показала, что главная идея с последовательным объединение диаграмм Вороного цепочек допускает корректную реализацию и всегда дает правильный результат для замкнутого многоугольника.

3. ОБОБЩЕННЫЙ АЛГОРИТМ ЛИ.

Авторы не ставят своей целью создание универсального алгоритма скелетизации ОМ, основанного на алгоритме Ли, поскольку это, по-видимому, невозможно в силу того, что алгоритм Ли изначально адаптирован под многоугольные фигуры без «дырок». Описанный ниже алгоритм предназначен, в первую очередь, для построения ДВ ОМ, количество «дырок» в котором сравнимо с $log\ n$, где n – количество вершин ОМ.

Схему алгоритма можно представить в виде трех шагов:

1. Многосвязная область, которую представляет собой ОМ, превращается в односвязную путем соединения несвязных компонент границы дополнительными ребрами.

2. Для построенной на первом шаге односвязной области с помощью алгоритма Ли строится ДВ. Границей этой области является простой вырожденный многоугольник (он имеет сдвоенные ребра), однако классический алгоритм Ли [3] применим к нему без каких-либо изменений.

3. Добавленные ребра удаляются, и в их окрестности восстанавливается ДВ.

Перейдем к более подробному описанию алгоритма.

Построение односвязной области.

1.1. У каждой «дырки» мы находим крайнюю левую вершину (с наименьшей абсциссой), если таких вершин несколько, то выбираем самую нижнюю (с наименьшей ординатой). Такая вершина всегда будет отдельным сайтом.

1.2. Выделенные вершины упорядочиваются по возрастанию: $s_1 < s_2$, если $x_1 < x_2$ или $x_1 = x_2$ и $y_1 < y_2$, тем самым упорядочиваются «дырки».

1.3. Из каждой выделенной вершины проводится горизонтальный луч, направленный влево, его направляющий вектор равен (-1,0). Будем предполагать, что «дырки», заданные $P_1,\dots,P_k$, уже упорядочены, то есть $P_1 < \dots < P_k$. Наша цель - для каждого j –ого луча найти его ближайшую точку пересечения с ребрами многоугольников $P_i,\ i=0,\dots,j-1$. Каждое такое ребро мы будем называть противолежащим выделенной вершине. Если искомая точка пересечения совпадет с вершиной ОМ, которая является сайтом, то её также будем называть противолежащей.

На этом шаге пункт 1.3. является самым трудоемким – в наихудшем случае он требует $O(\ k\ n\ )$ операций. Мы использовали следующий алгоритм, который в среднем требует $O(\ n\ log\ k\ )$ операций.

Упорядочим прямые, содержащие данные лучи, по возрастанию ординаты. В общем случае (когда никакие две прямые не совпадают) они разбивают плоскость на $k+1$ область. Далее мы начинаем перебирать все ребра многоугольников $P_i,\ i=0,\dots,k$. Для каждой из двух граничных точек ребра мы определяем область, в которой она лежит, для этого требуется $O(log\ k)$ операций. Определив области, мы легко находим прямые, которые пересекает данное ребро. Далее мы находим точку пересечения данного ребра с каждой прямой и определяем принадлежность этой точки соответствующему лучу. Понятно, что в большинстве случаев мы будем иметь единственную пересекаемую прямую, в этом случае на обработку одного ребра тратится $O(log\ k)$ операций, в наихудшем же случае ребро пересечет все k прямых, и на обработку одного ребра мы потратим $O(k)$ операций. Таким образом, в среднем мы затратим $O(\ n\ log\ k\ )$, в наихудшем случае остается оценка в $O(\ k\ n\ )$ операций.

1.4. Те из найденных точек пересечения, которые не совпадают с какой-либо вершиной, добавляются в список вершин, при этом содержащие их ребра делятся на два. После этого в список ребер добавляются новые ребра, представляющие собой отрезки, соединяющие выделенные вершины с ближайшими к ним точками пересечений. При этом каждый отрезок добавляется два раза, как проходимый в обоих направлениях.

Полученная область является односвязной, её граница представляет собой простой вырожденный многоугольник, ориентированный так, что при обходе его против часовой стрелки область остается слева по направлению обхода.

Построение диаграммы Вороного односвязной области. Диаграмма Вороного строится с помощью алгоритма Ли [3]. Этот шаг требует $O(\ n\ log\ n\ )$ операций.

Удаление ребер.

На этом шаге удаляются k ребер (точнее k пар ребер), добавленных на первом шаге, и одновременно восстанавливается диаграмма Вороного. Удаление начинается с самого большой (по порядку) «дырки». Поскольку каждая пара удаляется независимо от остальных, далее мы опишем алгоритм удаления одной пары.

1.1. Мы удаляем из ДВ все ребра, инцидентные удаляемым ребрам ОМ. При этом мы запоминаем сайты, смежные с удаляемыми ребрами ОМ (далее - просто смежные) и узлы ДВ, инцидентные удаляемым ребрам.

1.2. К смежным сайтам мы добавляем

1. выделенную вершину «дырки», которая является отдельным сайтом;

2. противолежащее ребро, если на первом шаге оно было разбито на два (при этом две его половины удаляются):

3. противолежащий сайт-точку.

Полученный набор сайтов мы обозначим через S.

Теорема 2. *При восстановлении ДВ изменяются только ЯВ сайтов, принадлежащих S.*

Доказательство. Заметим, что при удалении ребер ни

одна ЯВ не уменьшается. Действительно, пусть s – произвольный сайт. Его ЯВ состоит из всех внутренних точек ОМ, которые ближе к s, чем к любой другой точке границы ОМ (не принадлежащей s). Следовательно, при удалении части границы ОМ ЯВ сайта s не может уменьшиться.

Пусть s – произвольный сайт, не принадлежащий S. Предположим, что существует точка x, которая принадлежала ЯВ удаляемого ребра, а после восстановления ДВ стала принадлежать ЯВ сайта s. Обозначим через B_1 ЯВ сайта s до удаления ребра, через B_2 - ЯВ сайта s после удаления ребра, через B – ЯВ удаляемого ребра. Тогда $B_1 \subseteq B_2$, $B_1 \cap B = \varnothing$, множество $clo(B_1) \cap clo(B)$ либо пусто, либо состоит из одной точки (здесь $clo(\ )$ – замыкание). Так как B_2 – связное множество, то точку x можно соединить непрерывной кривой с B_1, на этой кривой найдется точка y, равноудаленная от s и от удаляемого ребра. Через точку y проходит бисектор сайта s и удаляемого ребра; так как точка y принадлежит B_2 , то пересечение этого бисектора с B_2 (мы обозначим его через Γ) содержит бесконечное множество точек. Множество Γ не может принадлежать ни границе B, ни границе B_1, так как если оно принадлежит одной из них, то оно принадлежит и другой, а значит принадлежит и их пересечению, что невозможно, так как это пересечение состоит не более чем из одной точки. Множество Γ не может также принадлежать ЯВ какого – либо другого сайта ОМ, поскольку ЯВ не уменьшается при удалении ребра (то есть после удаления ребра множество Γ по-прежнему принадлежало бы ЯВ этого сайта, тогда как оно принадлежит B_2). Следовательно, множества Γ не существует, а начит не существует и точки x. Теорема доказана.

1.3. Мы строим ЯВ сайтов, принадлежащих S (из теоремы 2 следует, что ЯВ остальных сайтов не изменяются). Построить эти ЯВ можно, например, с помощью обычного алгоритма Ли. Действительно, набор сайтов S естественным образом упорядочен – сайт s_1 меньше сайта s_2, если при обходе границы ЯВ удаляемых ребер ОМ против часовой стрелки, ребро ЯВ, инцидентное s_1 встречается раньше, чем ребро ЯВ, инцидентное s_2. Мы формируем из сайтов S начальные цепочки, причем на этот раз каждый сайт задает отдельную цепочку, за исключением сайта-точки, являющегося выделенной вершиной «дырки», и, если это так, противолежащего ему сайта-точки. Эти сайт включается в цепочки, состоящие из них и их соседей. После этого мы применяем алгоритм Ли объединения цепочек. Единственное отличие от обычного алгоритма Ли состоит в том, что в классическом алгоритме объединение двух последовательных цепочек начиналось с построения общего бисектора последнего сайта первой цепочки и первого сайта второй цепочки, который начинался в общей вершине этих цепочек. В нашем случае цепочки, вообще говоря, не имеют общих вершин, общий бисектор проводится из узла ДВ, инцидентного одному из удаляемых ребер ОМ, последнему сайту первой цепочки и первому сайту второй цепочки.

Подсчитаем число операций на этом шаге. Так как количество смежных сайтов не превышает $O(\ n\ )$, то применение к ним алгоритма Ли потребует $O(n \log n)$ операций. Таким образом, для удаления k ребер потребуется $O(k\ n \log n)$. Конечно, эта оценка сильно завышена. Известно, что общее число ребер диаграммы Вороного имеет величину порядка $O(\ n\ )$, таким образом количество сайтов, у которых граница ЯВ состоит из $O(\ n\ )$ ребер не зависит от n, а, следовательно, не зависит от k. Таким образом, количество операций на этом шаге можно оценить величиной $O(n \log n)$.

Подсчитав количество операций для всего алгоритма, мы получим величину порядка $O(N\ (\log\ N + k))$. Для большинства же ОМ данный алгоритм требует $O(N\ (\log N + \log k))$ операций, что уже сравнимо с достигнутой сейчас наилучшей оценкой $O(N \log N)$ операций.

4. ИСХОДНЫЙ АЛГОРИТМ ФОРЧУНА.

Пусть дан набор отрезков и точек, которые мы называем сайтами. Концы отрезков считаются принадлежащими множеству сайтов. Алгоритм Форчуна [1] строит ДВ произвольного множества сайтов; ниже мы кратко опишем основные идеи этого алгоритма.

В основе алгоритма лежит следующее преобразование плоскости: каждая точка поднимается вверх на расстояние, равное расстоянию до ближайшего сайта. Можно отметить четыре свойства, которыми обладает это отображение, и которые являются критическими для алгоритма: (1) оно однозначно на ДВ, (2) в образе области Вороного одной из нижних точек является точка соответствующего сайта, (3) для каждого узла преобразованной диаграммы хотя бы два ребра подходят снизу, (4) оно тождественно на сайтах. Образ ДВ можно найти методом заметающей прямой; у нас, как и в оригинальном алгоритме, прямая будет горизонтальной и будет двигаться снизу вверх. Статусом заметающей прямой является список ребер диаграммы пересекаемых ею. При обработке каждого нового сайта алгоритм ищет в этом списке его место. Список должен делать этот поиск эффективно. Координаты сайтов и регистрируемых пересечений хранятся в очереди с приоритетами. Приоритет – лексикографический порядок. В ходе работы алгоритма нужно уметь удалять данное пересечение из очереди с приоритетами, поэтому если она реализуется в виде кучи, то нужно дополнительно отслеживать местоположение элементов.

5. ОПЫТ РЕАЛИЗАЦИИ АЛГОРИТМА ФОРЧУНА.

Наша задача – построить ДВ или, что почти то же самое, скелет для ОМ. По сравнению с общим случаем, при решении этой задачи надо учитывать следующие особенности: (1) нам нужна лишь внутренняя часть ДВ, (2) сайты неким образом связаны (образуют ОМ), (3) некоторые ситуации, являющиеся вырожденными для общего случая, типичны для случая ОМ, необходимо обеспечить их эффективную обработку. Мы не будем строго судить реализацию за локальные ошибки в построенном скелете, но будем требовать, чтобы этот скелет был корректным планарным графом.

Первое, очень естественное, изменение в алгоритме состоит в том, что в список ребер, пересекающих заметающую прямую, добавляются ребра исходного контура; они позволяют определить, с какой стороны находятся внешняя и внутренняя части фигуры. Эти ребра не участвуют в актах определения пересечений между ребрами. Можно отметить два преимущества, которые дает нам это изменение. Во-первых, теперь можно избежать построения внешней части ДВ. Во-вторых, если в списке уже есть ребро, инцидентное вершине, которую нужно обработать, то мы можем избежать поиска в списке. Этот поиск будет нужен

только для вершин, оба инцидентных ребра которых направлены вверх, в приличном многоугольнике таких вершин мало.

Будем считать, что граница ОМ ориентирована так, что при обходе её против часовой стрелки внутренность ОМ остается слева по направлению обхода. Тогда ахиллесова пята алгоритма – направленные влево горизонтальные ребра. Соответствующие им ЯВ при применяемом отображении схлопываются по ординате. Алгоритм Форчуна в идеальной модели точной арифметики решает эту проблему тем, что точки извлекаются из списка в лексикографическом порядке. Но на практике при использовании 32-битных чисел с плавающей точкой IEEE-754 проблемы возникают уже с ребрами, у которых тангенс угла наклона по модулю меньше 0.03. Как можно бороться с этим, не прибегая к помощи точной или адаптивной арифметики? Подход с нормализацией данных (форсированной корректировкой входных данных) из-за большой величины порога плох – может получиться много самопересечений, и алгоритм не выдаст вообще никакого результата (хотя алгоритм терпим к незначительным самопересечениям). Можно извлекать точки из очереди с приоритетами не в лексикографическом порядке, а смотреть на их ординаты и, если они будут близки, то смотреть на абсциссы. В реализации автора выборка из очереди с приоритетами идет только по ординате, а при встрече с направленным влево ребром, близким к горизонтальному происходит временный переход в особый режим.

Еще можно упомянуть о способе вычисления пересечений между ребрами. В области вычислительной математики давно укоренился философский принцип примата комбинаторной информации (КИ) над численной. Алгоритм Форчуна в этом смысле лучше алгоритма Ли, так как он поддерживает достаточно много КИ, в частности, упоминавшийся список ребер. Поэтому для нахождения пересечений реализация не просто находит все корни и выбирает подходящий, а выбор корня или возврат отрицательного ответа старается делать на основе КИ.

6. АНАЛИЗ АЛГОРИТМА ЛИ

Выше уже отмечались слабые стороны алгоритма Форчуна, препятствующие его корректной работе в практических приложениях. Алгоритм Ли лишен подобных недостатков, поэтому при его реализации основные усилия были направлены на увеличение скорости работы программы. Автор (А.С.) исследовал зависимость скорости построения ДВ от вида многоугольника.

Обозначим через N количество сайтов многоугольника, через N_e – Количество ребер ДВ, через N_n – количество узлов ДВ. Верно следующее утверждение.

Лемма 1. *ДВ простого многоугольника, состоящего из N сайтов, обладает следующим свойствами:*

1. $N_e - N_n = N - 1$,
2. *N_n может принимать значения 1,…, N – 2, причем $N_n = 1$ только для правильного N – угольника, в общем случае (когда каждый узел инцидентен ровно трем ребрам ДВ) $N_n = N – 2$.*

Из леммы следует, что в общем случае (можно сказать, что число многоугольников, ДВ которых имеет узлы, инцидентные более чем трем ребрам ДВ, пренебрежимо мало по сравнению с числом всех многоугольников) $N_e = 2\,N – 3$ и $N_n = N – 2$ и не зависит от вида многоугольника. Отсюда напрашивается вывод, что наибольшей скорости на многоугольниках с одинаковым количеством вершин алгоритм Ли достигает на выпуклых многоугольниках (ВМ), когда число сайтов наименьшее и совпадает с количеством вершин. Однако на практике это заключение не подтверждается, и причина этого состоит в одном недостатке алгоритма Ли: значительная часть ребер и узлов, которые он строит для промежуточных цепочек, не входят в окончательную ДВ многоугольника. Лишних ребер алгоритм Ли строит немного, исследования показали, что их число составляет 10 –30 процентов от количества ребер, вошедших в окончательную ДВ. Намного хуже ситуация с построением лишних узлов, их алгоритм строит в 2-4 раза больше (для многоугольника, имеющего 36 вершин), чем их входит в окончательную ДВ, и при этом операция построения узла намного более «дорогостоящая», чем операция построения ребра.

Лемму 1 можно обобщить на случай ДВ произвольных цепочек. Для простоты мы сделаем это только для общего случая.

Лемма 2. *Пусть цепочка состоит из N сайтов и пусть в ДВ цепочки имеется N_h висячих ребер (висячим ребром мы будем называть неограниченное ребро ДВ) и нет узлов, инцидентных более чем трем ребрам. Тогда ДВ цепочки имеет $2\,N + 2 - N_h$ ребер и $N + 1 - N_h$ узлов.*

Заметим, что для цепочек, построенных по ВМ, N_h всегда равно единице, это единственное висячее ребро является бисектором концевых точек цепочки. Таким образом, построение ДВ для таких цепочек требует наибольшего числа операций.

Большое влияние на скорость работы алгоритма оказывает первоначальное количество цепочек, так как чем их больше, тем большее количество лишних узлов считает алгоритм. Первоначальное количество цепочек N_{ch} для произвольного n – угольника можно вычислить по формуле $N_{ch} = 2\,n – N$, где количество сайтов N меняется в пределах $n,\ldots,\ 2\,n – 3$. Таким образом, максимальное количество цепочек $N_{ch} = n$ опять достигается для ВМ.

Полную картину сложности вычисления ДВ для данного многоугольника дает следующая лемма.

Лемма 3. *Алгоритм Ли для многоугольника, имеющего n вершин и N сайтов, имеет сложность $K\,N \log N_{ch} = K\,N \log (2\,n – N)$, здесь K – некоторая константа, определяющая сложность одной операции.*

Следовательно, для ВМ алгоритм имеет сложность $K\,n \log n$. Предполагая, что количество операций определяется количеством всех найденных узлов, получаем, что для ВМ алгоритм Ли вычисляет в $\log n$ узлов больше, чем их входит в конечную ДВ. Этот теоретический результат хорошо согласуется с практическими результатами.

Из леммы 3 следует, что алгоритм Ли наименее эффективен для ВМ, именно для них отношение числа построенных узлов к числу узлов в конечной ДВ максимально. Однако для нас эффективность работы алгоритма является не самым главным параметром, намного важней для нас общая скорость работы алгоритма, которая определяется его сложностью для данного многоугольника. Ниже будет показано, что, несмотря на низкую

эффективность, сложность алгоритма Ли для ВМ остается одной из самых низких в классе многоугольников с заданным количеством вершин.

Исследуем функцию $f(N) = N \log (2n - N)$ на отрезке $[n, 2n - 3]$. Непосредственно проверяется, что функция f монотонно возрастает на отрезке $[n, N_0]$ и убывает на отрезке $[N_0, 2n - 3]$. Максимум достигается при $N = N_0$, можно показать, что $N_0/(2n) \to 1$ и $f(N_0)/(n \log n) \to 2$ при $n \to \infty$. Например, при $n = 100$ $N_0 = 157$, при $n=1000$ $N_0 = 1701$.

Функция f достигает абсолютного минимума на отрезке $[n, 2n - 3]$ при $N = 2n - 3$. В этом случае для построения ДВ требуется $2 K n \log 3$ операций. Пример такого многоугольника (для $n = 6$) приведен на рис. 1. Предположим, что сначала объединяются цепочки C_1 и C_2, потом объединяются $C_{1,2}$ и C_3. При объединении $C_{1,2}$ и C_3 нам нужно построить n – 2 прямые и n – 3 параболы, найти n – 3 пересечения прямой с прямой и n – 3 пересечения прямой с параболой. Необходимость искать пересечения прямых с параболами приводит к тому, что ДВ данного многоугольника строится быстрее ДВ выпуклого многоугольника только для достаточно больших n (когда вычисление $n \log n$ пересечений пар прямых требует большего числа операций, чем вычисление n пересечений пар прямых и парабол). Заметим также, что при другой последовательности объединения цепочек потребуется в два раза больше операций.

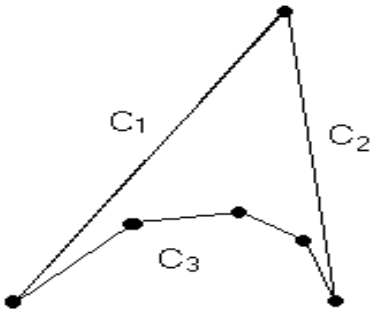

Рисунок 1.

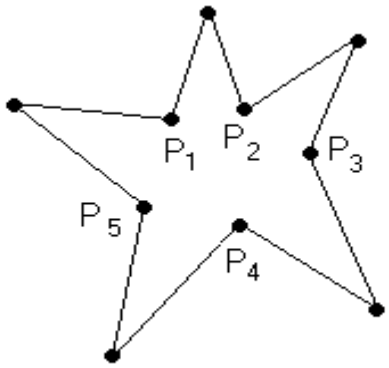

Рисунок 2.

Окончательно получаем, что сложность алгоритма Ли для ВМ больше сложности алгоритма Ли только для тех многоугольников, у которых количество сайтов близко к предельному $2n - 3$. Эти многоугольники принадлежат отрезку $[N_0, 2n - 3]$, относительная длина которого стремится к нулю при $n \to \infty$ (под относительной длиной понимается отношение длины отрезка $[N_0, 2n - 3]$ к длине отрезка $[n, 2n - 3]$). Отсюда следует справедливость нашего утверждения о низкой сложности алгоритма Ли для ВП.

В заключение отметим, что скорость работы алгоритма в большей степени зависит от вида многоугольника, а не от количества сайтов. Практика показывает, что ДВ звездообразного многоугольника (см. рис. 2) строится намного быстрее, чем ДВ ВМ. Из леммы 3 следует, что количество необходимых для этого операций имеет порядок $1.5 K n \log n$, т.е. в 1.5 раза больше числа операций, необходимых для построения ДВ ВМ, что не согласуется с практикой. Более подробный анализ алгоритма показывает, что активными сайтами являются только сайты-точки (сайты – отрезки имеют не более трех смежных сайтов, как следствие, для нахождения их ЯВ требуется $O(n)$ операций – эта величина не входит в асимптотическую оценку общего числа операций). На рисунке 2 активные сайты - это $P_1, \ldots, P_5$. Количество активных сайтов равно $n/2$, отсюда получаем оценку числа операций $0.5 K n \log n$, что в два раза меньше количества операций, необходимых для построения ДВ ВМ. Эта оценка полностью подтверждается на практике.

7. ЗАКЛЮЧЕНИЕ.

Скелетизация простого многоугольника, имеющего 730 вершин, с помощью алгоритма Ли на компьютере с процессором Intel Pentium III – 500 МГц потребовала 0.013 сек., скелетизация ОМ с тем же количеством вершин, имеющего одну дырку, (500 вершин у внешнего многоугольника и 230 у многоугольника, задающего «дырку») потребовала 0.0133 сек., скелетизация ОМ с десятью «дырками» потребовала 0.015 сек. Для сравнения, скелетизация этих же ОМ с помощью алгоритма Форчуна потребовала, соответственно, 0.015 сек. , 0.015 сек. и 0.0153 сек. Таким образом, экспериментальные результаты подтвердили теоретические ожидания, при небольшом количестве "дырок" обобщенный алгоритм Ли уверенно опережает алгоритм Форчуна, однако с увеличением количества "дырок", при неизменном количестве вершин, скорость алгоритма Ли начинает быстро падать, тогда как скорость алгоритма Форчуна падает достаточно медленно. Заметим, что асимптотическая оценка числа операций для алгоритма Форчуна не зависит от количества дырок, падение скорости алгоритма при увеличении их числа связано, очевидно, с увеличением сложности ДВ.

Работа выполнена при поддержке компании Intel в рамках сотрудничества Intel с МГУ. Авторы выражают свою благодарность Нестеренко Людмиле Всеволодовне и Ерухимову Виктору Львовичу за идею написания этой заметки. Отдельную благодарность авторы выражают Храпову Павлу Васильевичу за внимание, проявленное к заметке, и полезные советы.

СПИСОК ЛИТЕРАТУРЫ:

[1] Fortune S. A sweepline algorithm for Voronoi diagrams. – Algorithmica (1987) 2: 153-174.

[2] Goldberg D. What every computer scientist should know about floating-point arithmetic. – ACM Computing Surveys

(1991) 23#1: 5-48.

[3] Lee D. T. Medial axis transformation of a planar shape. – IEEE Transactions on pattern analysis and machine intelligence, Vol. PAM1-4, NO. 4, p. 363-369

[4] V. Srinivasan, L. R. Nackman Voronoi diagram for multiply-connected polygonal domains. – IBM Journal of Research and Development, Vol. 31, NO. 3, 1987

[5] Местецкий Л. М. Скелетизация многоугольной фигуры на основе обобщенной триангуляции Делоне. – Программирование, 1999, NO. 3, с. 16-31

Об авторах

Лагно Денис Владимирович – аспирант МГУ им. М. В. Ломоносова, механико-математический факультет,

Соболев Андрей Сергеевич – аспирант МГУ им. М. В. Ломоносова, механико-математический факультет.