УДК 519.688

А.В. Скворцов

ЛИНЕЙНО-УЗЛОВОЙ АЛГОРИТМ ПОСТРОЕНИЯ ОВЕРЛЕЕВ ДВУХ ПОЛИГОНОВ

Предлагаются универсальный алгоритм решения задачи нахождения объединения, пересечения и разности двух заданных полигонов на основе линейно-узловой модели и алгоритм упрощения сложных самопересекающихся полигонов. Обсуждаются особенности реализации.

Задача построения оверлеев двух полигонов (объединение, пересечение и разность) является одной из базовых задач вычислительной геометрии. В том или ином виде она встает в системах автоматизированного проектирования, геоинформационных системах, машинной графике.

Существуют многочисленные алгоритмы для решения данной задачи. Однако большинство из них обладает различными ограничениями. В первую очередь это связано с типом исходных данных. Наиболее легко решается случай оверлея двух выпуклых полигонов [1] или когда хотя бы один из полигонов является выпуклым. При этом дополнительно предполагается отсутствие различных вырожденных случаев (отсутствие совпадений последовательных точек, ненулевая площадь полигонов и т.д.). Один из первых алгоритмов, работающих для невыпуклых и многоконтурных полигонов, был представлен Вейлером и Азертоном [2–4]. К сожалению, в этом алгоритме приходится отслеживать многочисленные частные случаи, приводящие к существенному усложнению реализации. В работе Маргалита и Нота представлен значительно более проработанный алгоритм, позволяющий обрабатывать невыпуклые многоконтурные самонепересекающиеся полигоны [5].

Ранее автором совместно с Костюком предлагался еще более универсальный алгоритм построения оверлеев на основе алгоритма построения триангуляции с ограничениями [6]. Алгоритм обладает очень простой логикой, легок в реализации, но, к сожалению, работает значительно дольше других известных алгоритмов.

В настоящей работе представлен также универсальный алгоритм, позволяющий работать с произвольными полигонами, в т.ч. многоконтурными и самопересекающимися, но значительно более быстрый, чем основанный на триангуляции.

Постановка задачи

Определение. Границей полигона G будем называть множество контуров $\{K_1,\ldots, K_m\}$, $m\geq 1$, каждый из которых является замкнутой ломаной линией, соединяющей набор точек на плоскости (узлов границы): $K_i = \left((x_1^i, y_1^i),\ldots,(x_{n_i}^i, y_{n_i}^i)\right)$, $n_i \geq 3$.

Определение. Пусть даны некоторая граница G и некоторая точка (x,y), не лежащая на границе G. Проведем через точку (x,y) любой луч, не проходящий через узлы границы G. Будем считать, что точка (x,y) находится «внутри» границы G, если число пересечений луча с отрезками ломаных границы нечетно.

Данное определение является конструктивным, и его внутренняя непротиворечивость показывается, например, в [1].

Определение. Полигоном P будем называть геометрическое место точек, лежащих на заданной границе G либо находящихся «внутри» нее.

В литературе так иногда определяют общие полигоны, чтобы отличить их от более простых частных случаев (выпуклых, простых, ориентируемых, регулярных и др.) [1, 5]. В данной работе мы не будем вводить различные классы полигонов и поэтому будем использовать только такое общее определение.

Задача построения оверлеев. Пусть заданы два полигона P_1 и P_2 в виде границ G_1 и G_2. Требуется найти границы полигонов I, U, D, определенных как соответствующие булевы операции над множеством всех точек исходных полигонов:

$$I = \overline{\breve{P}_1 \cap \breve{P}_2}\,,\; U = \overline{\breve{P}_1 \cup \breve{P}_2}\,,\; D = \overline{\breve{P}_1 - \breve{P}_2}\,,$$

где знаком $\breve{P}$ обозначена операция размыкания (вычитание из множества его границы); $\overline{P}$ – замыкание множества.

Использование операций замыкания и размыкания в постановке задачи позволяет избежать вырожденных ситуаций при построении оверлеев. Иначе, например, в случае нахождения пересечения двух касающихся полигонов может образоваться множество в виде линии или даже одной точки, которое нельзя представить в виде невырожденного многоугольника.

Задача упрощения полигонов. Пусть задан полигон P в виде границы G. Требуется получить самонепересекающуюся границу S, определяющую полигон $S = \overline{\breve{P}}$ и не имеющую последовательных повторяющихся точек.

Дополнительно эта операция также используется для борьбы с вырожденными многоугольниками.

Линейно-узловая модель

Основная идея предлагаемых алгоритмов заключается в предварительном построении линейно-узловой модели – геометрического планарного графа специального вида, ребра которого должны соответствовать отрезкам исходных границ полигонов. Затем после построения графа производится классификация ребер графа по признаку вхождения в результирующий полигон. И в конце выполняется сборка отрезков графа в последовательность отрезков границы требуемого полигона.

Происхождение вводимого автором термина «линейно-узловой» связано с используемыми в геоинформатике похожими топологическими моделями данных – покрытиями, называемыми также линейно-узловыми моделями.

Для решения ранее поставленных задач дадим некоторые определения и поставим некоторые более простые задачи.

Определение. Линейно-узловой моделью M будем называть геометрический граф $\{N, R\}$, где N – множество вершин графа – точек (x_i, y_i) на плоскости; R – множество непересекающихся ребер – отрезков $((x_j^1, y_j^1),(x_j^2, y_j^2))$, соединяющих вершины графа.

Задача построения линейно-узловой модели. Пусть задано произвольное множество отрезков $((x_j^1, y_j^1), (x_j^2, y_j^2))$ на плоскости. Надо построить линейно-узловую модель, в которую войдут в качестве вершин все точки $(x_j^1, y_j^1), (x_j^2, y_j^2)$ и все точки пересечения исходных отрезков, а также в качестве ребер должно быть взято минимальное множество непересекающихся отрезков (возможно, касающихся концами), совпадающее как геометрическое место точек с исходным множеством отрезков.

По сути, требуется взять в качестве ребер все исходные отрезки, причем если эти отрезки пересекаются между собой, то они должны быть разбиты на части точками пересечений.

Данная задача легко решается в терминах классической геометрии. К сожалению, как правило, реальные компьютерные вычисления выполняются только с ограниченной точностью, что приводит в рамках данной задачи к интересным эффектам. Например, найденная точка пересечения отрезков в силу ограниченности точности вычислений оказывается немного в стороне от реального вещественного значения. В итоге два исходных пересекающихся отрезка после разбиения на две части точкой пересечения оказываются лежащими не на прямых, проходящих через исходные отрезки. В итоге это может привести к пересечению этих новых отрезков с какими-то другими ранее непересекаемыми отрезками.

Данная ситуация иллюстрируется на рис. 1. Пунктирной сеткой размечена система координат, в которой каждый квадрат соответствует минимальной возможной точности представления координат точек (в частном случае использования целых чисел шаг сетки будет равен единице).

На рис. 1, *а* изображены отрезки A_iB_i, i=1, 2, 3, причем второй и третий отрезки пересекаются в точке S_1.

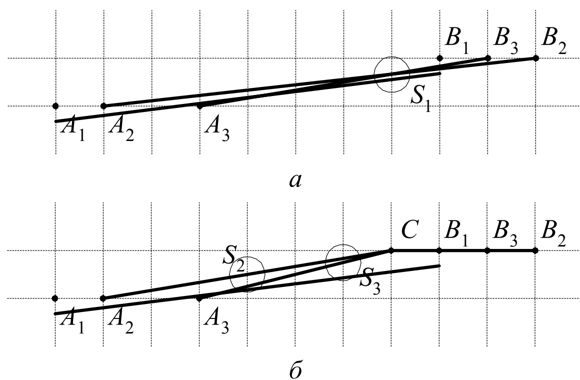

Рис. 1. Проблема поиска пересечений

После разбиения второго и третьего отрезков на части точкой их пересечения C=округление (S_1) образуются четыре новых отрезка A_2C, A_3C, B_2C, B_3C, из которых первые два пересекаются с отрезком A_1B_1 в точках S_2 и S_3 соответственно (рис. 1, *б*).

Данная проблема обычно возникает при попытке построения оверлеев соприкасающихся полигонов. При этом касающиеся стороны полигонов в силу ограниченной точности представления координат могут, строго говоря, пересекаться, что и приводит в дальнейшем к появлению ошибок.

Таким образом, при разработке алгоритма построения линейно-узловой модели необходимо учесть возможность появления новых пересечений и при необходимости разбивать в том числе и вновь образуемые отрезки.

Алгоритм построения линейно-узловой модели. Предполагается, что дано множество исходных отрезков $((x_j^1, y_j^1), (x_j^2, y_j^2))$.

Шаг 1. Формируем множество вершин $N=\{n_i\}$, по очереди добавляя в него вершины исходных отрезков; совпадающие вершины отбрасываем. Множество ребер делаем пустым: $R = \varnothing$.

Шаг 2. Заносим в список L еще не обработанных отрезков все исходные отрезки в виде пар (n_j^1, n_j^2).

Шаг 3. Пока список L не пуст, извлекаем из него самый длинный отрезок (n^1, n^2) и пытаемся вставить его в текущую линейно-узловую модель. Если такой отрезок уже вставлен в качестве ребра, то ничего не делаем. Если вставляемый отрезок не пересекает ни одно ребро из R и не проходит через вершины из N, кроме n^1 и n^2, то вставляем его. В противном случае, если имеет место последовательное прохождение через вершины $\tilde{n}^1,...,\tilde{n}^m \in N$, заносим в список L новые отрезки $(n^1, \tilde{n}^1)$, $(\tilde{n}^1, \tilde{n}^2), \ldots, (\tilde{n}^m, n^2)$. Иначе, если обнаружено последовательное пересечение с некоторыми ребрами $r^1,...,r^m, r^i = (n_i^1, n_i^2)$ в точках $(\hat{x}^i, \hat{y}^i)$, удаляем эти ребра из R и заносим в список L новые отрезки $(n^1, \hat{n}^1), (\hat{n}^1, \hat{n}^2), ..., (\hat{n}^m, n^2)$ и $(n_i^1, \tilde{n}^i)$, $(\hat{n}^i, n_i^2)$ $i = \overline{1,m}$, где $\hat{n}^i$ – старые или вновь добавленные вершины в точке $(\hat{x}^i, \hat{y}^i)$. **Конец алгоритма.**

Данный алгоритм очень похож на алгоритм вставки структурных линий в триангуляцию Делоне с ограничениями, приведенный в [7]. Именно поэтому ему свойственны те же самые проблемы.

При моделировании автором работы данного алгоритма часто возникала ситуация, когда небольшое количество исходных отрезков приводит к значительному разрастанию списка L в процессе работы. Например, была ситуация, когда на 5 исходных «почти» коллинеарных отрезков алгоритм в конце концов выдавал более 15 тысяч ребер. Происходит это вследствие того, что каждая пара отрезков после расчета их пересечения образует 4 новых отрезка, также являющихся «почти» коллинеарными другим 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 в каждом узле координатной сетки имеется по вершине линейно-узловой модели и требуется вставить отрезок AB, который пересекается с ранее вставленным ребром CD (рис. 2). В результате найденная точка пересечения AB и CD будет округлена, например, до точки A. В итоге в список L попадут от-

резки AC, AD и опять AB. Так как мы извлекаем на каждом шаге из списка самый большой отрезок, то он будет бесконечно разрастаться за счет постоянной вставки отрезков AC и AD.

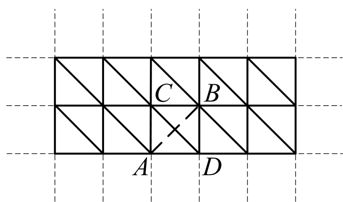

Рис. 2. Попытка вставки микроотрезков

В таком варианте алгоритм все же не годится для практической работы.

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 вершины линейно-узловой модели, 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 отрезков, прежде чем вставлять новую, попробуем найти в окрестности радиуса ε_2 уже ранее вставленную вершину линейно-узловой модели.

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 немного увеличивает время выполнения дополнительного поиска точек в окрестностях. Экспериментально были выбраны наиболее приемлемые с точки зрения быстродействия и качества значения $\varepsilon_1=\varepsilon_2=10$.

В завершение отметим, что для более эффективного поиска вершин в заданной окрестности необходимо поместить все вершины в бинарное дерево поиска. Тогда поиск в указанной окрестности будет выполняться в среднем за логарифмическое время. Точно так же все ребра следует поместить в структуру R-дерева [8] для ускорения поиска возможных пересечений выставляемых отрезков с ребрами модели.

Построение топологии

Теперь перейдем к вопросу применения линейно-узловой модели для построения оверлеев и упрощения полигонов.

Основная идея предлагаемых ниже алгоритмов построения оверлеев и упрощения полигонов заключается в передаче на вход алгоритма построения линейно-узловой модели всех отрезков исходных полигонов, последующего построения топологии, затем классификации ребер по некоторому критерию и, в заключение, объединения классифицированных отрезков в границу результирующего полигонов.

Вначале обратим внимание, что если в границу некоторого заданного полигона входят $2n$ одинаковых отрезков, то в силу введенного определения полигона их можно не учитывать при определении попадания точек внутрь полигона. При этом можно во время вставки второго совпадающего отрезка сразу же их удалять. Однако этого лучше сразу не делать, т.к. в случае более чем двукратной вставки одинаковых ребер несколько возрастает время работы алгоритма за счет удаления и повторной вставки ребер. Аналогично если в границу некоторого заданного полигона входят $2n+1$ одинаковых отрезков, то это эквивалентно их однократному вхождению.

Таким образом, при построении линейно-узловой модели необходимо дополнительно для каждого ребра модели хранить параметр количества прохождений через ребро различных исходных отрезков первого и второго полигона (в задаче упрощения второго полигона нет). Для нас будет важно, что этот параметр принимает одно из трех значений: 0, нечетное или четное число. При этом алгоритм построения линейно-узловой модели несколько модифицируется. В списке L надо хранить тройки (n^1, n^2, p), где p – номер полигона, а для каждого вставленного ребра нужно иметь счетчики c_1 и c_2 его использования первым и вторым полигоном, которые увеличиваются при вставке новых ребер или прохождении по уже существующим. Если обнаруживается пересечение, то в список L нужно поместить все разрезанные отрезки от одного до четырех раз. А именно, если отрезок использовался первым полигоном нечетное число раз, то в список L помещается одна тройка (n^1, n^2, 1), если четное – то две таких тройки. Точно также для второго полигона в список L может быть помещена одна или две тройки (n^1, n^2, 2).

В итоге построенная линейно-узловая модель будет содержать узлы и ребра, представляемые в следующем виде:

```
Узел = record
  X, Y: координата;
end;
Ребро = record
  N: array [1..2] of Указатель_на_узел;
  C: array [1..2] of integer;
end;
```

В дальнейших алгоритмах нам понадобится узнавать, какие ребра инцидентны данному узлу. Поэтому нам требуется несколько дополнить имеющуюся структуру следующим образом:

```
Узел = record
  X, Y: координата;
  RibCount: integer;
  Ribs: array [1..RibCount] of Указатель_на_ребро;
end;
```

В геоинформатике такой шаг обычно называют «построение топологии», т.е. полный расчет прямых и обратных связей между объектами. Рассмотрим его.

Алгоритм построения топологии линейно-узловой модели. Пусть даны списки ребер и узлов. Для каждого узла пусть заданы пары (x,y), для ребер – четверки (n^1,n^2,c^1,c^2). Требуется для каждого узла установить число d_i и список инцидентных ребер $(r_1^i, \ldots, r_{d_i}^i)$.

Шаг 1. Устанавливаем для каждого узла число инцидентных ребер d_i:=0. Далее в цикле для каждого ребра (n^1,n^2,c^1,c^2) увеличиваем на единицу счетчики d в вершинах n_i^1 и n_i^2.

Шаг 2. В соответствии с найденным значением d_i в каждой вершине выделяем память под массивы $(r_1^i,...,r_{d_i}^i)$. Затем опять устанавливаем для каждого узла d_i=0.

Шаг 3. В цикле для каждого ребра $r_i = (n_i^1, n_i^2, c_i^1, c_i^2)$ рассматриваем вершины n_i^1 и n_i^2. Заносим в список инцидентных ребер для вершины n_i^1 и увеличиваем счетчик d_i на единицу: $r_{d_i}^i := r_i, \quad d_i := d_i + 1$. **Конец алгоритма.**

Легко видеть, что трудоемкость данного алгоритма линейна.

Классификация ребер модели и конструирование полигонов

Этап классификации ребер линейно-узловой модели при построении оверлеев предназначен для выделения тех ребер, которые должны будут войти в результирующий полигон. Рассмотрим его для разных типов оверлеев.

Вначале нужно для каждого ребра определить, по какую сторону от него находится первый и второй полигон. Далее же используем следующие логические условия в зависимости от типа операции (подобная, но несколько более сложная методика используется в [5], однако там отдельно рассматриваются случаи «дырок» и «островов» в полигоне, а самопересечения вообще невозможны):

1. Объединение полигонов. В результирующий полигон должны войти только два типа рёбер (рис. 3, *а*, *б*):

– если ребро используется в обоих полигонах и оба они лежат по одну сторону от ребра;

– если ребро используется только одним из полигонов и оно лежит вне другого.

2. Пересечение полигонов. В результирующий полигон должны войти только два типа рёбер (рис. 3, *а*, *в*):

– если ребро используется в обоих полигонах и оба они лежат по одну сторону от ребра;

– если ребро используется только одним из полигонов и оно лежит внутри другого.

3. Разность полигонов. В результирующий полигон должны войти только три типа рёбер (рис. 3, *а*, *г*):

– если ребро используется в обоих полигонах и оба они лежат по разную сторону от ребра;

– если ребро используется только первым полигонов и оно лежит вне второго;

– если ребро используется только вторым полигонов и оно лежит внутри первого.

Таким образом, для классификации всех ребер требуется умение выполнять две операции:

1) определять расположение полигона относительно ребра линейно-узловой модели;

2) определять, попадает ли некоторое ребро на границу, внутрь или вне полигона.

Для выполнения первой операции необходимо предварительно рассчитать для каждого ребра расположение полигона относительно ребра. Для этого предлагается пройти по части линейно-узловой модели, используемой данным полигоном, по принципу поворачивания в каждом узле направо на еще не пройденное ребро. Однако если мы начали обходить

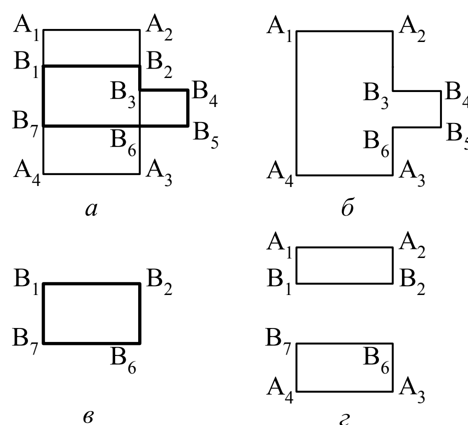

Рис. 3. Примеры оверлеев

контур, являющийся «дыркой», то нужно поворачивать всегда налево. Тем самым будут выделены все контуры, используемые полигоном, а при обходе ребер полигон будет находиться всегда справа (рис. 4).

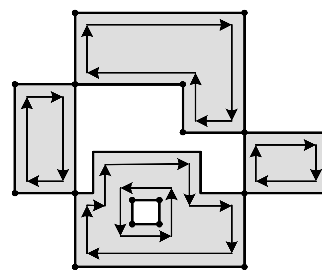

Рис. 4. Выделение контуров линейно-узловой модели

Для этого используем следующий алгоритм.

Алгоритм выделения контуров линейно-узловой модели.

Шаг 0. Помечаем как непройденные только те ребра, которые используются заданным полигоном нечетное число раз.

Шаг 1. Сортируем все вершины линейно-узловой модели по горизонтали и помещаем в список *T*.

Шаг 2. Последовательно идем по списку *T* и извлекаем из него вершины *N*. Для каждой вершины *N* выполняем шаг 3.

Шаг 3. Вначале определяем, является ли данная вершина *N* частью контура дырки. Для этого определяем количество ранее сформированных контуров, строго внутрь которых попадает данная вершина. Если это число нечетно, то мы имеем дело с дыркой. Далее, пока в вершине *N* имеются инцидентные непройденные ребра, выполняем шаг 4.

Шаг 4. Идем из вершины *N* по непройденному ребру, составляющему минимальный угол от вертикали вверх (вниз для дырок) среди всех еще не пройденных ребер. Начинаем идти по найденному ребру. При переходе по ребру на некоторую вершину N_i надо выбрать следующее необработанное ребро, инцидентное N_i, составляющее минимальный угол при повороте по (против для дырок) часовой стрелке от этого ребра на только что пройденное по принципу поворачивания всегда направо (налево для дырок). При переходах по ребрам необходимо помечать все эти ребра как пройденные. Переходы

ведем до тех пор, пока есть возможность куда-либо идти. Иначе мы оказываемся в исходной вершине *N*. **Конец алгоритма.**

Вторая операция определения попадания ребра внутрь полигона может быть выполнена путем проверки попадания средней точки тестируемого ребра внутрь заданного полигона. Это обычно делается путем проведения некоторого луча из тестируемой точки и последующего подсчета числа точек пересечения луча со сторонами полигона.

Однако, как мы ранее заметили, в результате построения линейно-узловой модели её ребра из-за округлений могут не точно проходить по исходным отрезкам полигона. Поэтому нужно обратно сформировать по линейно-узловой модели список отрезков, образующих исходный полигон. И уже по этому списку отрезков определять попадание внутрь или вне полигона.

После того как мы проклассифицировали все ребра линейно-узловой модели, остается только собрать эти ребра в результирующий полигон. Для этого достаточно после классификации пометить входящие в результат ребра как непройденные, а остальные – как пройденные и запустить алгоритм выделения контуров с шага 1.

Обратим внимание, что в результате работы предложенных алгоритмов будет построен полигон, в котором контуры не будут взаимно- и самопересекаться, и каждый контур будет задан в таком порядке обхода точек, что полигон всегда находится справа по ходу обхода.

В данном разделе мы рассмотрели задачу построения полигонов. Задача об упрощении полигонов решается теми же самыми алгоритмами, только этап классификации здесь отсутствует. При этом достаточно сформировать линейно-узловую модель по исходным отрезкам полигона и затем вызвать алгоритм выделения контуров.

Заключение

Предложенные автором линейно-узловой алгоритм построения оверлеев и алгоритм упрощения полигонов позволяют явно учесть многочисленные тонкие эффекты, возникающие на пороге точности вычислений. Проведенное автором имитационное моделирование работы алгоритма на различных вариантах исходных данных показало, что данный алгоритм имеет такой же порядок трудоемкости, что и лучший из известных алгоритм Маргалита–Нотта [5], но в среднем в 1,5 раза медленнее последнего. Тем не менее предлагаемый алгоритм имеет несколько большую область определения и достаточно простую структуру для программной реализации.

ЛИТЕРАТУРА

1. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение: Пер. с англ. М.: Мир, 1989. 478 с.
2. *Роджерс Д.* Алгоритмические основы машинной графики: Пер. с англ. М.: Мир, 1989. 512 с.
3. *Weiler K., Atherton P.* Hidden surface removing using polygon area sorting // Computer Graphics. 1977. Vol. 11. P. 214–222.
4. *Weiler K.* Polygon comparison using graph representation // Computer Graphics. 1980. Vol. 14. P. 10–18.
5. *Margalit A., Knott G.D.* An algorithm for computing the union, intersection of difference of two polygons // Computers & Graphics. 1989. Vol. 13. № 2. P. 167–183.
6. *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. 22–47.
7. *Скворцов А.В.* Особенности реализации алгоритмов построения триангуляции Делоне с ограничениями // Наст. журн.
8. *Guttmann A.* R-trees: A dynamic index structure for spatial searching // Proc. ACM SIGMOD Int. Conf. on Management of Data. 1984. P. 47–57.

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