УДК 004.02 DOI 10.31114/2078-7707-2018-1-167-172

Метод многоуровневых регулярных сеток для индексации геометрических данных

В.В. Давыдов

Silvaco Inc., vitaly.davydov@silvaco.com

***Аннотация* — В работе представлен метод индексации фиксированного набора геометрических данных, основанный на пространственной декомпозиции с помощью многоуровневых регулярных сеток в применении к задачам САПР. Предложенная реализация метода демонстрирует его применимость на практике, сочетает в себе низкое потребление памяти с высокой скоростью выполнения поисковых запросов для не сильно кластеризованных данных. Метод был успешно реализован и применен для восстановления связности геометрий цепей питания в аналоговых или смешанных дизайнах. Произведено сравнение данного подхода с реализацией R-tree библиотеки BOOST.**

***Ключевые слова* — метод сетки, многоуровневая регулярная сетка, индексация геометрических данных.**

I. Введение

Алгоритмы быстрого поиска геометрий имеют важное практическое значение в задачах, решаемых инструментами проектирования микросхем (топологическое проектирование, физическая верификация, задачи визуализации данных). Развитие технологического процесса приводит к существенному возрастанию количества элементов схемы и увеличению плотности упаковки. Как следствие, возрастают требования к алгоритмам поиска (индексации) геометрий в плане производительности и использования памяти.

Существует множество подходов, которые условно можно разделить на два типа: подходы, основанные на принципе кластеризации (R-деревья, адаптированные B-деревья); подходы, основанные на принципе пространственной декомпозиции (сетки, дерево квадрантов, BSP-деревья) [1]. На практике нередко применяется классический метод сетки ([2], [3]) для поиска геометрий в прямоугольной области в силу простоты реализации и хорошей производительности. Метод сетки заключается в декомпозиции всей области пространства, содержащей геометрии, на ячейки регулярной сеткой. Каждая ячейка содержит список ссылок на геометрии, пересекающиеся с данной ячейкой. На реальных данных метод сетки нередко показывает более высокую производительность, чем альтернативные подходы, основанные на деревьях различных типов. Это объясняется тем фактом, что в силу специфики предметной области большинство схем часто имеют структуру, близкую к регулярной, а набор геометрий (например, металлических проводников или окаймляющих прямоугольников ячеек) не является сильно кластеризованным.

Одним из недостатков метода сетки является существенная деградация производительности операций поиска в случае сильно кластеризованных данных. Другой недостаток заключается в том, что выбор размера ячеек сетки существенно влияет на производительность операций поиска и объем используемой памяти. На сетке с ячейками малых размеров существенная часть геометрий может пересекать более чем одну ячейку. В реализации операций поиска потребуется устранить дублирование результата. При задании ячеек слишком больших размеров произойдет деградация производительности поисковых запросов. Существуют методы адаптивных регулярных сеток, в которых размеры ячеек вычисляются по входящим данным (например, [4]). Но они не решают проблему длинных и узких геометрий, которые часто встречаются в реальных схемах.

Предложенный в статье метод решает проблемы выбора ячеек сетки и устранения дублирования результата в случае фиксированного набора геометрий, т.е. набора, который не предполагает добавления новых геометрий, изменения или удаления существующих из структуры поиска. Метод позволяет применить быстрые параллельные алгоритмы для построения структуры поиска. В качестве геометрий рассматриваются двумерные прямоугольники с целочисленными координатами (32 бита). В случае сложных геометрий, таких как многоугольники и полигоны (в том числе с дырками), для поиска можно использовать окаймляющие прямоугольники геометрий. Метод допускает обобщение на многомерное пространство, но для двумерного пространства существует ряд оптимизационных решений, позволяющих достичь хороших показателей по производительности запросов и по объему используемой памяти.

Основная идея метода заключается в классификации всех геометрий по размерным группам и создании для каждой размерной группы пространственной регулярной сетки с ячейками, размеры которых превышают размеры любой геометрии из этой группы. В научной литературе автором найдено не много статей, посвященных развитию классического метода сетки. Нередко метод сетки сочетается с иерархическими структурами ([5], [6]). Предложенный в данной работе метод отличается от подходов с использованием иерархических структур тем, что набор пространственных сеток размерных групп не образует иерархию.

II. ОСНОВНОЙ ПОДХОД

Назовем областью поиска прямоугольную область, содержащую все окаймляющие прямоугольники геометрий, а окном запроса прямоугольную область, в которой выполняется операция поиска. Рассмотрим случай двумерных геометрий на плоскости. Разобьем весь диапазон размеров геометрий регулярной логарифмической сеткой. Каждой геометрии сопоставим координаты ячейки размерной сетки по формулам:

$$w_g = \text{FLOOR}(\log_K W)$$

$$h_g = \text{FLOOR}(\log_K H)$$

где W, H – ширина и высота геометрии (ее окаймляющего прямоугольника), w_g, h_g– координаты ячейки размерной сетки геометрии, K – эвристически задаваемое основание логарифма, FLOOR – обозначение операции округления вниз до целого. Ячейку размерной сетки назовем размерной группой. Основание логарифма выбирается с учетом необходимости уменьшения количества ячеек размерной сетки (размерных групп). Определенным таким образом координатам размерной сетки можно взаимно-однозначно сопоставить номер ячейки сетки. В случае целочисленных 32 битных координат и основанием логарифма 8 размер сетки будет 11x11 (121 ячейка).

Для каждой размерной группы формируется пространственная сетка, ограниченная областью поиска. Размеры ячеек пространственной сетки определяются как максимальная ширина и высота геометрий данной размерной группы. Несложно показать, что следует выбрать следующие размеры ячеек пространственной сетки для определенных w_g, h_g:

$$W_{cell} = K^{w_g+1}$$

$$H_{cell} = K^{h_g+1}$$

Таким образом, каждой геометрии ставится в соответствие одна и только одна ячейка пространственной сетки. Алгоритмы поиска основаны на переборе ячеек, попадающих в окно запроса, для геометрий из каждой размерной группы независимо.

Для практического применения метода необходимо редуцировать количество размерных групп. Вначале определим максимально допустимое количество столбцов и строк размерных сеток (минимальные и максимальные размеры ячеек сетки). Это можно сделать как исходя из свойств геометрий, так и заранее зафиксировав максимальные значения (например, 5000x5000). На размерной сетке такой выбор аналогичен выбору левой нижней ячейки. Минимальное количество столбцов и строк – 1 (ячейка соответствует всей области поиска).

Введем следующий порядок обхода ячеек размерной сетки (рис. 1):

- Начнем обход с левой нижней ячейки размерной сетки с координатами (w_g, h_g).
- Зафиксируем h_g и обойдем все ячейки в порядке возрастания w_g.
- Для той же левой нижней ячейки теперь зафиксируем w_g и обойдем все ячейки размерной сетки в порядке возрастания h_g.
- Перейдем от левой нижней к следующей ячейке «по диагонали», увеличив одновременно две координаты.
- Повторим шаги алгоритма до тех пор, пока все ячейки размерной сетки не будут пройдены.

h_g (высота) ⇨	7	12	15	16
	6	11	13	14
	5	8	9	10
	1	2	3	4
	w_g (ширина) ⇨			

Рис. 1. Порядок обхода ячеек размерной сетки

Установленный порядок обхода ячеек размерной сетки позволяет последовательно выбрать те размерные группы, для которых соблюдается определенный эвристический критерий. Например, критерий заполнения пространственной сетки размерной группы. Если для размерной группы не соблюдается выбранный критерий, то эта размерная группа игнорируется, и выбирается следующая размерная группа в порядке обхода (для которой также проверяется соблюдение выбранного критерия).

Представим набор всех геометрий в виде «плоского» массива. Отсортируем геометрии по размерным группам (по координатам ячеек размерной сетки). Линейным проходом по массиву определим начальное и конечное смещения диапазонов элементов, соответствующих каждой размерной группе. Зная границы диапазонов ячеек размерной сетки можно сопоставить количество элементов в каждой группе определенному критерию и принять решение о сохранении данной размерной группы или переназначении геометрий в следующую размерную группу. Пример критерия – количество геометрий в размерной группе превышает количество ячеек соответствующей пространственной сетки. Альтернативный подход заключается в том, чтобы заранее зафиксировать подмножество всех размерных групп, классифицировать по ним все геометрии, а уже потом выполнять редукцию. Именно такой подход будет использован в нашей реализации. Метод допускает обобщение на случай многомерных пространственных данных.

III. РЕАЛИЗАЦИЯ

В данном разделе представлена реализация метода для случая фиксированного набора двумерных геометрий. В качестве базовой структуры хранения геометрических данных (ссылок) предлагается использовать C++ контейнер vector из библиотеки STL или аналогичный. Каждый элемент контейнера может содержать как индексируемые геометрические данные, так и ссылки на них. Элементы контейнера должны иметь небольшой

размер, а операции копирования или перемещения (move constructor) быть быстрыми. В противном случае подготовка контейнера к выполнению операций поиска (предобработка) займет большое время, так как в процессе предобработки будет выполняться сортировка элементов контейнера. Решение хранить ссылки или непосредственно геометрические данные определяется характером решаемой задачи. Хранение геометрических данных в контейнере позволяет решить как задачу хранения данных, так и задачу их индексации на том же объеме памяти.

A. Предобработка

На этом этапе по определенному эвристическому принципу происходит выбор пространственных сеток. В нашей реализации заранее зафиксирован следующий максимальный набор сеток, показанный в табл. 1. Используются только сетки, соответствующие закрашенным ячейкам табл. 1. Количество ячеек по X и Y определяет на сколько частей будет изначально разрезана область поиска по соответствующим осям. Ширины и высоты ячеек пространственных сеток округлялись вверх до ближайшего значения, соответствующего степени двойки. Это позволило заменить операции деления или умножения на более быстрые битовые операции (логический сдвиг, побитовое «И»).

Размер области поиска, покрывающей все геометрии, вычисляется линейным проходом по всему массиву геометрий, либо с использованием параллельных алгоритмов, таких, например, как reduce из библиотеки Intel Threading Building Blocks.

Таблица 1

Фиксированный набор пространственных сеток

		Количество ячеек по Y					
		8192	4096	512	64	8	1
Количество ячеек по X	8192	■		■			
	4096		■	■	■	■	■
	512	■	■	■	■	■	■
	64		■	■	■	■	■
	8		■	■	■	■	■
	1		■	■	■	■	■

Каждая ячейка регулярной сетки идентифицируется двумя сеточными координатами X_g и Y_g, определяющими позицию ячейки по осям x и y:

$$X_g = FLOOR\frac{(X - X_0)}{W_{cell}}$$

$$Y_g = FLOOR\frac{(Y - Y_0)}{H_{cell}}$$

где W_{cell}, H_{cell} – ширина и высота ячейки сетки, соответственно; X_0, Y_0 – минимальные координаты области поиска. Сеточные координаты округляются «вниз» до целого. Если сеточная координата больше максимальной, то используется максимальная координата. Наравне с сеточными координатами каждой ячейке сетки ставится в соответствие индекс по формуле:

$$I_{cell} = X_g + Y_g * N_x$$

где I_{cell} - индекс (номер) ячейки, N_x – количество ячеек сетки по оси X. Операция деления в представленных выше формулах заменяется операцией сдвига, так как W_{cell}, H_{cell} по принципу построения являются степенями двойки.

Формирование разделов вектора, каждый из которых соответствует одной размерной группе, происходит в установленном порядке обхода размерных групп. К вектору применяется стандартный алгоритм STL partition, который перемещает элементы, размеры которых соответствуют текущей размерной группе, в начало вектора. Если количество выбранных для данной сетки элементов не соответствует заранее установленному критерию, то данная сетка игнорируется и выбирается более крупная сетка для построения. В качестве критерия может выступать следующее условие: количество геометрий больше общего количества ячеек в сетке. Начальная и конечная позиции данного раздела в векторе сохраняются во вспомогательных структурах для дальнейшего использования.

После формирования разделов вектора набор элементов в каждом разделе сортируется по сеточным координатам (или индексам ячеек). Элементы с одинаковыми сеточными координатами дополнительно сортируются по X или Y в зависимости от соотношения ширины и высоты ячеек. Таким образом, все элементы данного раздела упорядочиваются по строкам сетки, а потом по колонкам. Важным свойством такой сортировки является то, что элементы вектора, которые лежат в одной строке сетки, близко расположены в памяти. Этап сортировки занимает основную часть времени предобработки. Существуют параллельные алгоритмы сортировки, позволяющие существенно сократить время на современных многопроцессорных, многоядерных системах. Например, алгоритм параллельной сортировки из библиотеки Intel Threading Building Blocks.

Заключительным шагом предобработки является построение для каждой пространственной сетки массива смещений ячеек в векторе для ускорения операции локализации элементов ячеек по сеточным координатам. Массив смещений может занимать существенную часть памяти, в зависимости от количества ячеек в сетке. Для сетки размером 5000x5000 для хранения смещений ячеек требуется выделить ~200M памяти (смещение представлено типом size_t, занимающего 8 байт в случае 64-х битных систем). Хранение смещений ячеек позволяет существенно ускорить операции поиска, но не является обязательным. В качестве альтернативы можно использовать бинарный поиск или,

например, хранить смещения для строк сетки, а элементы столбцов локализовывать используя бинарный поиск внутри строки сетки.

В итоге, все элементы вектора разбиваются на разделы, каждый из которых соответствует своей размерной группе. В каждом разделе элементы разбиваются по ячейкам пространственных сеток, как показано на рис. 2. Элементы, расположенные в одной ячейке пространственной сетки, отсортированы по их пространственным координатам.

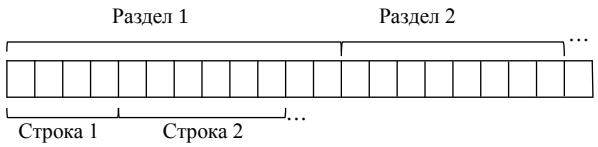

Рис. 2. Порядок расположения ячеек размерных сеток в массиве геометрий

По сути, вся память используется для хранения данных геометрий (или ссылок на данные), а также для хранения смещений ячеек в массиве геометрий. Память, используемая для хранения смещений ячеек, может занимать существенную долю всей используемой памяти. Размер памяти, используемой для хранения вспомогательной информации (дескрипторы уровней сетки и др.) пренебрежимо мал. Отказ от использования смещений ячеек сеток позволит уменьшить используемую память, но приведет к существенной деградации операций поиска, так как для определения смещений ячеек потребуется применить бинарный поиск ко всему массиву геометрий или его части. В качестве приемлемой альтернативы можно хранить только смещения строк пространственной сетки, а для локализации элементов в строке сетки использовать бинарный поиск по элементам строки.

B. Поиск в окне запроса

Поиск в окне запроса осуществляется независимо для каждой размерной группы (пространственной сетки). Результаты поиска объединяются. Для каждой пространственной сетки выполняются следующие шаги:

1) Определяется диапазон сеточных координат по каждой из осей по минимальным и максимальным координатам окна запроса. Если для привязки геометрии к ячейке пространственной сетки используется нижняя левая точка, то сеточные координаты левой нижней точки окна запроса необходимо уменьшить на 1, так как возможно пересечение геометрий с окном запроса, находящихся левее и ниже на одну ячейку.

2) Поиск выполняется для каждой строки сетки, попадающей в диапазон поиска. Для ячеек строки определяется начальное и конечное смещение элементов в массиве геометрий, используя массив смещений. Каждый элемент диапазона проверяется на пересечение с окном запроса и добавляется в результирующий список в случае пересечения.

Производительность операции поиска зависит от множества факторов: от характера набора размерных групп и соответствующих им пространственных сеток, от размера окна запроса, от характера заполнения ячеек сетки. На общее время поиска основное влияние оказывает поиск в сетках с ячейками наименьших размеров, так как количество ячеек, попадающих в окно запроса, существенно превышает количество ячеек для крупных сеток.

Время поиска в пространственной сетке условно можно разделить на две временные компоненты: время прохода по строкам сетки и время прохода по ячейкам строк сетки, попадающих в окно запроса. В силу близости расположения элементов строки в векторе проход по ячейкам (столбцам) строки, попадающим в окно запроса, заменяется проходом по соответствующему диапазону элементов вектора. Границы диапазона элементов строки вычисляются за константное время с использованием массива смещений или за время, соответствующее времени бинарного поиска элементов в строке, в зависимости от реализации.

Первый временной компонент зависит только от размеров области поиска и не зависит от наличия или отсутствия геометрий. Второй компонент зависит от наличия геометрий – кандидатов на попадание в результат, т.е. если в строке сетки отсутствуют геометрии, то время на итерацию по ячейкам строки потрачено не будет. Если второй компонент фактически определяется объемом результата (количеством геометрий – кандидатов на попадание в результат) и является естественным, то первый компонент является нежелательным, так как даже при отсутствии геометрий будет потрачено время на проход по строкам сетки.

Введем следующие величины: N_{gx}^{q}, N_{gy}^{q} - количество ячеек, частично или полностью попадающих в окно запроса, N_{cell} – среднее количество геометрий в ячейке. Тогда сложность алгоритма поиска в одной сетке можно представить формулой $O\left(N_{gx}^{q}\, N_{gy}^{q}\, N_{cell}\right)$ с учетом того, что сложность операции вычисления диапазона сеточных координат будет O(1), которая достигается при использовании массива смещений элементов ячеек сетки.

IV. Тестирование

Для проверки применимости реализации метода на практике было выполнено тестирование с использованием данных из реального дизайна. Тестирование производилось на компьютере с 512Гб оперативной памяти и 2-мя процессорами Intel Xeon E5-2699 v3 (18 ядер, 36 рабочих потоков). В сумме - 72 рабочих потока. В качестве альтернативы для сравнения была выбрана реализация R-дерева из библиотеки BOOST версии 1.64.0 ([7], [8]). В качестве тестовых прямоугольников были взяты окаймляющие прямоугольники компонент дизайна в количестве ~ 1 млрд. Особенностью дизайна, выбранного для тестирования, является отсутствие компонент (прямоугольников) в большой области в центральной части плана кристалла (~25% общей площади), что позволяет протестировать случай, когда существенная часть ячеек пространственных сеток не со-

держит геометрий. Тест состоял в наполнении структуры поиска прямоугольниками и последующем выполнении операций поиска в областях, определяемых каждым прямоугольником из тестового массива. По мнению автора такой тест наиболее приближен к реальным задачам экстракции связанных геометрий дизайна.

В табл. 2 представлена структура пространственных сеток для тестового дизайна. Видно, что подавляющее количество прямоугольников были добавлены в самую мелкую сетку с количеством ячеек 5500x7063.

Таблица 2

Структура пространственных сеток

№	NX	NY	К-во элементов	Среднее к-во элементов в ячейке	Макс. к-во элементов в ячейке
1	5500	7063	1091292530	28.1	369
2	5500	442	3221776	1.3	187
3	344	7063	21865478	9.0	370
4	2750	442	5485955	4.5	54
5	2750	56	448955	2.9	21
6	2750	7	158760	8.2	26
7	43	3532	181687	1.2	114
8	344	56	126727	6.6	330
9	344	7	9393	3.9	15
10	43	442	22913	1.2	54
11	43	56	6147	2.6	58
12	43	7	584	1.9	11
13	43	1	67	1.6	28
14	6	56	444	1.3	47
15	1	56	134	2.4	53
16	6	7	74	1.8	9
17	6	1	15	2.5	6
18	1	7	50	7.1	19
19	1	1	28	28.0	28

Результаты тестирования представлены в табл. 3. Для наполнения R-деревьев использовалась наиболее быстрая операция группового добавления элементов, подразумевающая использование алгоритмов упаковки [9]. Использование данной операции подразумевает предварительную подготовку данных в определенном формате, что может потребовать дополнительной памяти и времени.

Подготовка сетки поиска для выполнения поисковых запросов состоит из двух этапов: заполнение и предобработка. Заполнение подразумевает выделение памяти и добавление элементов в массив. Этап предобработки заключается в непосредственной подготовке (сортировке) данных для выполнения операций поиска. Для сравнения использовались три типа R-дерева, реализованные в библиотеке BOOST. Автором было выполнено тестирование с разными значениями параметра, определяющего максимальное количество потомков в узле: 4, 16, 64 потомка, из которых было выбрано значение 16, приводящее к наилучшим показателям. Тестирование проводилось несколько раз, показатели памяти и времени были усреднены. Для метода многоуровневых сеток в таблице отдельно показаны шаги заполнения и предобработки. Шаг предобработки представлен в однопоточном и многопоточном вариантах. Для R-дерева указан только шаг заполнения, так как подготовка структуры поиска происходит непосредственно при добавлении элементов, а отдельного шага предобработки не предусмотрено. Видно, что время поиска в случае сетки поиска меньше примерно в 3 раза по сравнению с временем поиска в случае R-деревьев. Выигрыш в использовании памяти составляет около 2 раз. Время, затрачиваемое на заполнение R-деревьев, ненамного меньше времени, затрачиваемом на предобработку для метода сетки. В случае распараллеливания этапа предобработки достигается существенный выигрыш во времени: 30 сек вместо 327 сек при распараллеливании на ~72 потоках.

Таблица 3

Результаты тестирования

Шаг	Память (Гб)	Время (сек)
Метод многоуровневых сеток		
Заполнение	17.5	19
Предобработка (1 поток)	0.4	327
Предобработка (~72 потока)	0.4	30
Поиск	-	2300
R-дерево (linear split, 16 узлов)		
Заполнение	31.5	273
Поиск	-	6981
R-дерево (quadratic split, 16 узлов)		
Заполнение	31.5	272
Поиск	-	6873
R*-дерево (16 узлов)		
Заполнение	31.5	268
Поиск	-	6895

V. Заключение

Описанный метод многоуровневых регулярных сеток демонстрирует существенно меньшее время поиска на меньшем объеме памяти. Метод был успешно реализован и применен на практике на дизайнах с большим количеством геометрий. Несмотря на фундаментальный недостаток метода сетки – существенная деградация показателей при высокой кластеризации данных – на реальных дизайнах метод часто показывает лучшие сравнительные показатели, чем R-дерево (его разновидности) и другие структуры поиска. Представленная реализация метода применима к фиксированному набору двумерных геометрий, но сам подход допускает обобщение на многомерные данные.

Благодарности

Автор признателен разработчикам САПР компании Silvaco К.К. Малинаускасу и А.Ф. Самойлову за консультации по математическим и инженерным темам, а также компании Silvaco за возможность подготовить статью.

Литература

[1] Золотов В.А., Семенов В.А. Современные методы поиска и индексации многомерных данных в приложениях моделирования больших динамических сцен // Труды Института системного программирования РАН. 2013. Том 24. С. 381-416.

[2] Bentley J.L., Friedman J. H. Data structures for range searching // ACM Comput. Surv. 1979. 11(4). P. 397–409.

[3] Ласло М. Вычислительная геометрия и компьютерная графика на C++. — М.: БИНОМ, 1997.
[4] Akman V., Franklin W.R., Kankanhalli M., Narayanaswami C. Geometric computing and uniform grid technique // Comput. Aided Des. 21, 7 (August 1989), P. 410-420
[5] Cazals F., Puech C. Bucket-like space partitioning data structures with applications to ray-tracing // In Proceedings of the thirteenth annual symposium on Computational geometry (SCG '97). ACM, New York, NY, USA, P. 11-20.
[6] Nievergelt J., Hinterberger H., Sevcik K. C. The grid file: An adaptable, symmetric multikey file structure // ACM Trans. Database Syst. 1984. 9(1). P. 38–71.
[7] Guttman A. R-trees: a dynamic index structure for spatial searching // SIGMOD Rec. June 1984. V. 14. Issue 2. P. 47-57.
[8] Beckmann N., Kriegel H.P., Schneider R., Seeger B. The R*-tree: an Efficient and robust Access Method for Points and Rectangles // Proc. ACM SIGMOD. May 1990. P. 323-331.
[9] Leutenegger S.T., Lopez M.A., Edgington J. STR: a simple and efficient algorithm for R-tree packing // Proceedings 13th International Conference on Data Engineering. 7-11 April. 1997.

A Method of Multi-level Uniform Grids for Spatial Searching

V.V. Davydov

Silvaco Inc., vitaly.davydov@silvaco.com

Abstract — **Fast geometry search techniques are of high importance in electronic design automation area (topological design, physical verification, data visualization and others). The evolution of the technology leads to the increase of the number of elements on a chip. Consequently, the requirements to geometry indexing algorithms become stricter regarding memory consumption and query performance. In practice, a simple uniform grid method ([2], [3]) is often a good alternative to tree-based search structures such as R-tree, BSP-tree, and quad tree. Despite the decent query performance in comparison with tree-based structures, it has such weakness as significant performance degradation with highly clustered data, higher memory consumption when geometries cover more than one grid cell, that can result in duplication of found geometries (some geometries may cover several adjacent grid cells).**

To get over some of the disadvantages of the classical-grid method a new uniform grid-based method is presented in this article. The implementation of the method for the fixed set of 2D geometries demonstrates lower memory consumption with higher query performance in comparison with BOOST R-tree implementations. The method has been successfully applied in practice for restoring interconnectivity of net power shapes in analog and mixed designs.

The idea of the method is to group all the geometries by their sizes and to create a separate spatial uniform grid for each size group. The uniform grid cells sizes are chosen to be larger than any geometry sizes of the group. Each geometry is assigned on a single grid cell only by its origin point (bounding box lower left point, center). The range search algorithm is applied independently to each spatial grid. Because the number of size groups can be high enough, a reduction technique is proposed to decrease its number.

The implementation was successfully tested on a real design with the huge amount of geometries. About 109 device boundary boxes were used as the input. The test was to construct a search structure with rectangles and to perform a query in the areas corresponding to each input rectangle. The test demonstrates higher query rates in comparison with BOOST R-tree implementation - more than 3x faster with the lower amount of memory (2x lower). Considering that the real design has a significant empty area in the center of the layout (about 25% of full layout size) the method demonstrates decent performance on non-regular data.

Despite some limitations related to clustered data, the method can be applied to the most tasks in electronic design automation area. The introduced implementation is applicable to the fixed set of 2D geometric data, but it can be enhanced to support spaces of higher dimensions.

Keywords —multi-level uniform grid, spatial search, geometry search

REFERENCES

[1] Zolotov V.A., Semenov V.A. Sovremenie metodi poiska I indekstsii mnogomernih dannih v prilosheniyah modelirovaniya bolshih dinamitcheskih tcen. // Trudi Instituta sistemnogo programmirovaniya RAN. 2013. Том 24. С. 381-416.
[2] Bentley J.L., Friedman J. H. Data structures for range searching // ACM Comput. Surv. 1979. 11(4). P. 397–409.
[3] Lazslo M. Vytchitlitelnaya geometriya i komputernaya grafika na C++. — M.: BINOM, 1997.
[4] Akman V., Franklin W.R., Kankanhalli M., Narayanaswami C. Geometric computing and uniform grid technique // Comput. Aided Des. 21, 7 (August 1989), P. 410-420.
[5] Cazals F., Puech C. Bucket-like space partitioning data structures with applications to ray-tracing // In Proceedings of the thirteenth annual symposium on Computational geometry (SCG '97). ACM, New York, NY, USA, P. 11-20.
[6] Nievergelt J., Hinterberger H., Sevcik K. C. The grid file: An adaptable, symmetric multikey file structure // ACM Trans. Database Syst. 1984. 9(1). P. 38–71.
[7] Guttman A. R-trees: a dynamic index structure for spatial searching // SIGMOD Rec. June 1984. V. 14. Issue 2. P. 47-57.
[8] Beckmann N., Kriegel H.P., Schneider R., Seeger B. The R*-tree: an Efficient and robust Access Method for Points and Rectangles // Proc. ACM SIGMOD. May 1990. P. 323-331.
[9] Leutenegger S.T., Lopez M.A., Edgington J. STR: a simple and efficient algorithm for R-tree packing // Proceedings 13th International Conference on Data Engineering. 7-11 April. 1997.