Современные методы поиска и индексации многомерных данных в приложениях моделирования больших динамических сцен

Золотов В.А., Семенов В.А.

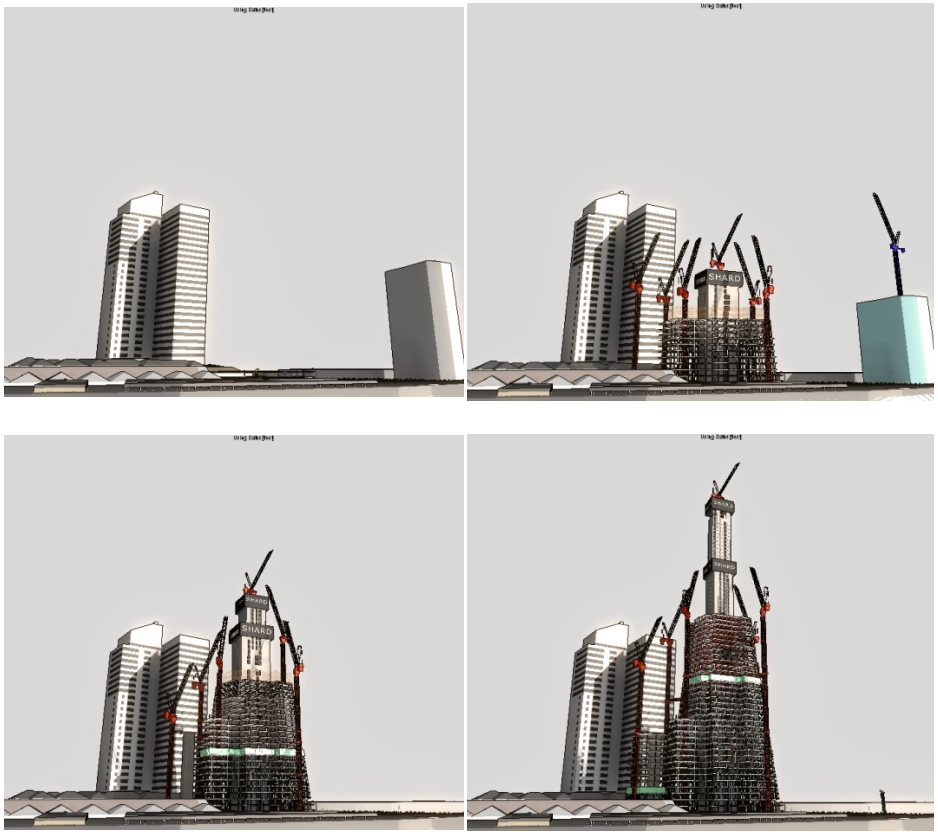
Аннотация. Статья посвящена современным методам поиска и индексации многомерных данных применительно к задачам моделирования больших динамических сцен. Подобные задачи имеют разнообразные приложения в компьютерной графике и анимации, мультимедийных базах данных, системах виртуальной и дополненной реальности, системах автоматизированного проектирования и производства, робототехнике, геоинформационных системах, системах комплексной инженерии и управления проектами, однако их решение часто оказывается невозможным из-за высокой вычислительной сложности алгоритмов пространственно-временной локализации объектов сцен и требует применения специальных схем индексации многомерных данных. В статье обсуждаются два фундаментальных подхода к организации подобных схем, а также проводится их сравнительный анализ в контексте комплексных требований, предъявляемых к приложениям обсуждаемого класса.

1. Введение

В последние годы проблемы управления сложными многомерными данными привлекают все большее внимание как со стороны научного сообщества, так и со стороны производителей прикладного программного обеспечения [1]. Однако, несмотря на обширные исследования в этой области и многочисленные публикации, посвященные, в частности, пространственным и темпоральным СУБД [2], вопросы обеспечения эффективного доступа к многомерным данным с учетом функционала программного приложения и особенностей решаемых прикладных задач остаются открытыми и требуют дальнейшей проработки.

В настоящей работе предпринимается попытка систематизировать современные методы индексации многомерных данных и выделить семейства,

наиболее перспективные для приложений моделирования больших динамических сцен. Рассматриваемый класс приложений чрезвычайно широк и охватывает системы компьютерной графики и анимации, программирование игр, мультимедийные базы данных, системы виртуальной и дополненной реальности, геоинформационные системы (ГИС), системы автоматизированного проектирования и производства (САПР), робототехнику, системы комплексной инженерии, информационные системы управления проектами (ИСУП). На рис. 1 приведен пример большой динамической сцены, возникающей в приложении ИСУП и состоящей из сотен тысяч геометрических объектов. Серия изображений иллюстрирует ход работ по сооружению масштабного, технологически сложного комплекса зданий “Осколок стекла” в центре Лондон-сити.



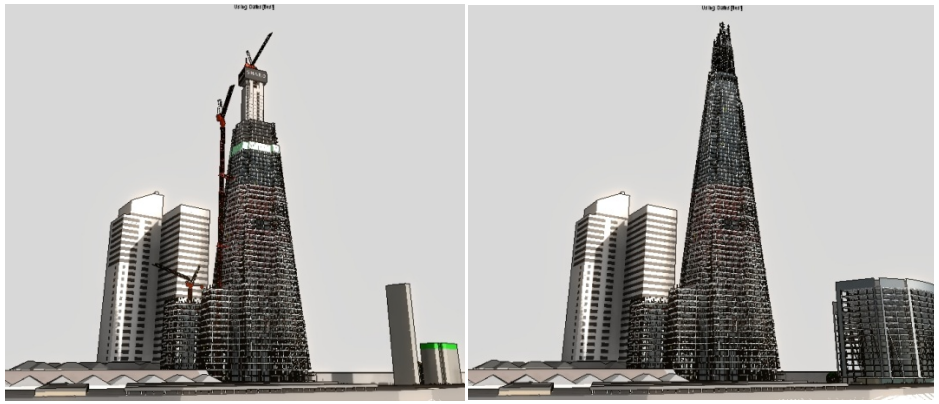


Рис. 1. Пример большой динамической сцены.

Принципиальными особенностями обсуждаемых приложений являются:

— сложность и масштабность сцен, содержащих в себе миллионы объектов с индивидуальными геометрическими и поведенческими характеристиками;

— многообразие геометрических типов данных, представленных в сценах примитивами, сплайнами, неявно заданными алгебраическими кривыми и поверхностями, выпуклыми и невыпуклыми многогранниками, облаками точек, твердотельными конструкциями;

— жесткий характер пространственно-временной когерентности сцен, связанный с сочетанием относительно быстрых и медленных перемещений объектов при их существенно неравномерном пространственном распределении;

— отсутствие строгой детерминированности в динамике сцен и возможная индукция дополнительных случайных событий;

— высокая размерность пространства (более точно, конфигурационного пространства) при моделировании кинематических конструкций с большим числом степеней свободы;

— разнообразие функций визуального моделирования сцен, в частности, поиск объектов, находящихся в заданной области пространства (как правило, имеющей форму параллелепипеда) в заданный промежуток времени; отсечение объектов сцены призмой или конусом видимости; формирование альтернативных геометрических представлений объекта в виде иерархий ограничивающих объемов (bounding volume hierarchies); определение степени детализации геометрического представления объекта (level of details), необходимой для визуально адекватной растеризации сцены; определение пространственных столкновений объектов сцены

(collision detection) на заданном временном интервале; формирование и обновление структур наполненности (occupancy grids) и топологических карт (topological maps) сцены; поиск бесконфликтных путей в псевдо-динамических сценах (motion planning) и т.п.

Перечисленные особенности диктуют довольно жесткие требования к эффективности исполнения типовых запросов и сбалансированности затрат на обновление пространственных индексов при перманентных изменениях в динамической сцене. Примечательно, что вопросы хранения пространственных данных и индексов во внешней памяти часто отходят на второй план, поскольку деградация производительности вычислительных процедур наступает даже в случаях, когда данные целиком помещаются в оперативной памяти. Поэтому основным требованием оказывается эффективность исполнения типовых запросов. В настоящей работе анализируются четыре основных типа запросов, а именно: поиск объектов в заданной области пространства, поиск ближайших соседей в постановках NN–задач (nearest neighbor problem) и *k*NN–задач (*k* nearest neighbor problem), добавление объекта в сцену в заданном положении и удаление объекта из сцены. Примечательно, что многие упомянутые выше функции естественным образом редуцируются к данным запросам при условии, что пространственные индексы обновляются и поддерживаются в согласованном состоянии на каждом временном шаге моделирования динамической сцены. В частности, перемещение объекта в сцене может эмулироваться операцией удаления объекта и последующей его вставкой в новом положении. Определение коллизий объектов может осуществляться путем предварительной локализации соседей и последующего попарного пересечения их граничных представлений [3]. Нужно отметить, что когда динамика сцены полностью предопределена, возникают дополнительные возможности для пространственно-временной индексации событий. В частности, альтернативные схемы подобной индексации, основанные на октальных деревьях, обсуждаются в нашей работе [1].

Естественно, охватить в рамках статьи все известные методы и отобрать наиболее эффективные для обсуждаемого класса приложений не представляется возможным. Поэтому в работе, прежде всего, обсуждаются основополагающие принципы индексации многомерными данных, которые могли бы служить отправной точкой для разработки новых перспективных стратегий управления сложными пространственно-временными данными в приложениях моделирования больших динамических сцен. В разделе 2 мы выделяем два фундаментальных принципа индексирования пространственных данных и на основе них выстраиваем общую классификацию методов. В разделе 3 рассматривается класс методов, в основе которых лежит принцип кластеризации объектов. В разделе 4 анализируются методы индексации, использующие декомпозицию пространства. В заключение проводится их сравнительный анализ и даются рекомендации по их применению.

2. Классификация методов индексации многомерных данных

Хотя арсенал существующих методов поиска и индексации многомерных данных на основе деревьев поиска, хэш-таблиц и пространственных сеток довольно разнообразен, в их основе, как правило, лежат два фундаментальных принципа: кластеризации объектов и декомпозиции пространства. Первый принцип, иногда называемый в иностранной литературе объектными пирамидами (object pyramids), заключается в многоуровневой композиции объектов сцены в кластеры с тем, чтобы обеспечить их плотное покрытие иерархиями ограничивающих объемов и обеспечить быструю пространственную локализацию объектов. Второй принцип (spatial decomposition) предполагает последовательную декомпозицию пространства всей сцены и определение принадлежности каждого объекта той или иной из полученных областей. Каждый из принципов допускает огромное число вариаций структур индексации и алгоритмов заполнения, обновления и поиска, поэтому мы выделяем дополнительные семейства методов.

К методам, реализующим принцип кластеризации объектов, следует отнести

— сбалансированные, хорошо ветвистые деревья, ориентированные на страничный доступ к данным большого объема и хранимым во внешней памяти. Некоторые авторы непосредственно адаптируют для этих целей популярные B [4], B+ [5], B* [6] деревья, устанавливая для объектов отношение линейного порядка, порождаемое их относительным пространственным расположением. Для задания отношения обычно используются кривые построчного, спирального, Z-образного, N-образного, U-образного заполнения пространства, а также известные функции упорядочивания Мортона, Пеано-Гильберта, Грэя и Кантора [7] (см. рис.2). В результате близлежащие объекты помещаются в одну вершину и оказываются на одной странице, что важно при обработке ряда запросов. Тем не менее, выполнение типового запроса — поиска объектов по заданной области остается проблематичным. С этой целью базовые структуры обобщаются до R [8], R+ [9], R* [10] деревьев, которые разбивают пространство на множество иерархически вложенных и, возможно, пересекающихся, прямоугольников

(параллелепипедов в пространственно-трехмерном случае или *n*-мерных политопов в случае более высокой размерности). Каждый прямоугольник является минимально ограничивающим для объектов, ассоциируемых с соответствующей вершиной дерева, что дает возможность ограничить поиск объектов теми вершинами, которые имеют непустое пересечение с областью поиска. Ключевым моментом здесь является выбор техники кластеризации объектов, которая с одной стороны должна обеспечить сбалансированность и наполненность дерева, а с другой стороны — четкую локализацию поиска, обусловленную минимальным перекрытием ограничивающих прямоугольников. Часто это оказывается проблематичным из-за особенностей пространственного заполнения сцены и существенных затрат на кластеризацию и перебалансировку всего дерева. Как обычно, поиск компромисса достигается релаксацией ряда требований и условий;

— метрические деревья, обеспечивающие быстрый поиск соседей в постановках NN– и *k*NN–задач. Линейная сложность наивного поиска часто оказывается неприемлемой для приложений, реализующих вычислительно сложные методы. Особый интерес в связи с этим представляют iDistance-структуры [11], деревья покрытий (cover-tree) [12] и BK-деревья [13]. Например, при формировании iDistance-структур выбирается относительно небольшое количество вспомогательных опорных вершин (обычно в центрах выявленных кластеров объектов), а основные объекты индексируются с использованием в качестве ключа расстояния до ближайшей опорной вершины. При поиске соседей анализу подлежат лишь те объекты, которые лежат в кольце, образованным ближайшей опорной вершиной и имеющим внутренний и внешний радиусы, согласованные с расстоянием до основного объекта на основании правила треугольника. Аналогичным образом реализуются запросы поиска объектов в заданной пространственной области.

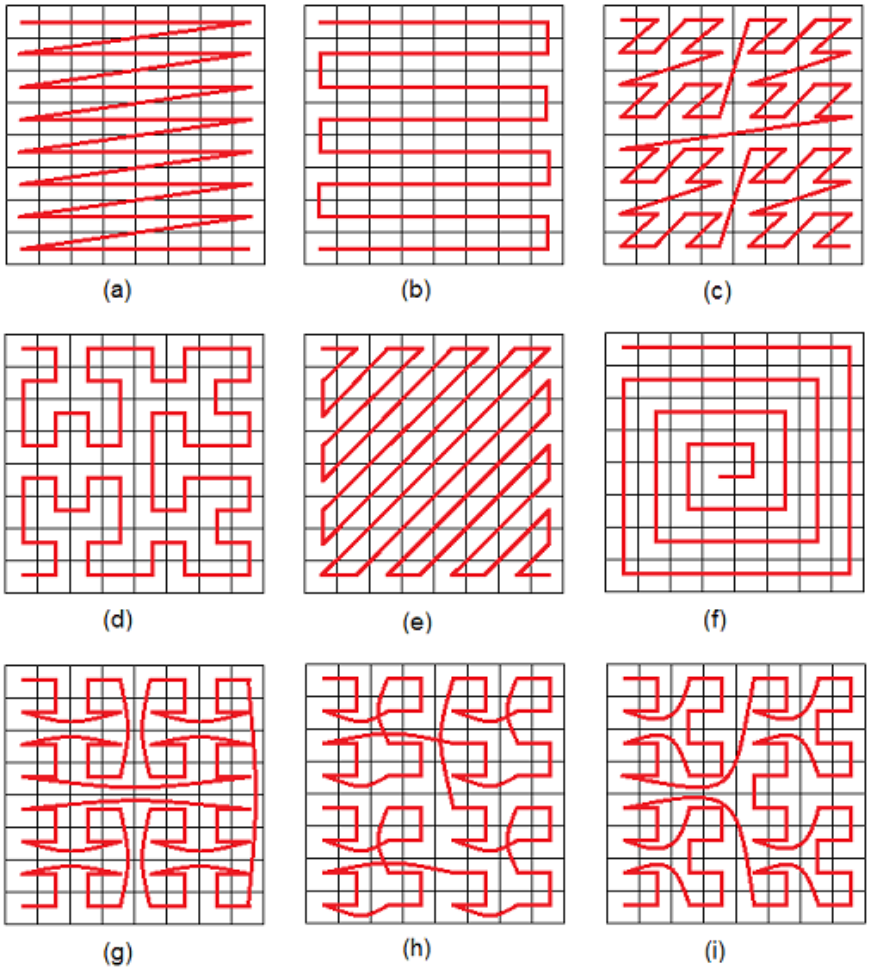


Рис. 2. Кривые упорядочивания пространства. (a) Строчное упорядочивание. (b) Двунаправленное строчное упорядочивание. (c) Z-упорядочивание. (d) Упорядочивание Пиано-Гильберта. (e) Диагональное упорядочивание Кантора. (f) Спиральное упорядочивание. (g) Упорядочивание Грея. (h) Двойное упорядочивание Грея. (i) U-упорядочивание.

Методы, основанные на пространственной декомпозиции, можно условно подразделить на следующие семейства:

— Структуры, обеспечивающие быстрый поиск на интервалах, такие как деревья диапазонов (range trees) [14], сегментов (segment trees) [15],

интервалов (interval trees) [16], деревья приоритетного поиска (priority search trees) [17]. Данные структуры естественным образом обобщаются на случай произвольной размерности, однако редко применяются в приложениях с размерностью выше двух из-за значительных затрат по памяти.

— Бинарные деревья пространственной декомпозиции, к самому известному варианту которых следует отнести BSP-деревья (binary spatial partitioning) [18], [19]. Деревья этого типа предполагают разбиение произвольными одиночными плоскостями, что широко используется в приложениях компьютерной графики для упорядочивания фрагментов сцен и удаления невидимых элементов в зависимости от местоположения камеры. Однако вряд ли они могут использоваться в качестве универсального пространственного индекса. Известные попытки применить полосы, параллелепипеды и даже обобщенные политопы в качестве пространственных элементов разбиения (как это реализуется, например, в LSD (local split decision tree) [20], HB (holey brick tree) [21] и D-деревьях [22] соответственно), оказались не совсем удачными в силу неоправданного усложнения вычислений. Более востребованными представляются kD-деревья [23], допускающие последовательное разбиение ортогональными плоскостями и являющиеся весомой альтернативой интервальным структурам при более высокой размерности пространства. Выбор переменной пространства, относительно которой осуществляется разбиение, и сам способ построения плоскости существенно различаются в многочисленных адаптивных (adaptive kD) [24], гибридных (hybrid kD-B) [25] и обобщенных (generalized k+D) версиях этой структуры. Для обеспечения сбалансированности деревьев успешно применяются техника плавающей медианы в SMP (sliding-midpoint) структурах [26], техника аппроксимации медианы в VAMSplit (varience approximate median split) деревьях [27] и техника выделения кластеров, как это реализуется в BBD (balanced box decomposition) деревьях [28]. Версии DQS (dynamically quantized space) [29], DQP (dynamically quantized pyramids) деревьев [30] учитывают наполненность вершин и в большей степени ориентированы на страничную подгрузку фрагментов сцен.

— Регулярные сетки, обычно применяемые в пространственно-трехмерном случае и имеющие ячейки в виде параллелепипедов, тетраэдров, призм с треугольниками или шестиугольниками в основании. Сопоставляя объекты сцены с ячейками сетки, с которыми они имеют непустое пересечение, можно сформировать необходимый пространственный индекс. Иногда для этого применяют, так называемые, сеточные файлы (grid directory files) [31], которые допускают возможность ассоциирования нескольких ячеек с одним объектом, но исключают привязку нескольких объектов к одной ячейке. Существует ряд работ, в которых для этих целей используют хэш-таблицы на основе известных техник OPLH (order preserving linear hashing),

LHRBI (linear hashing with reversed bit interleaving), MDEH (multidimensional extendible hashing), LHPE (linear hashing with partial expansions), PLOP (piecewise linear order preserving), квантильного (quantile hashing) и спирального (spiral hashing) хэширования [32]. На наш взгляд они предпочтительны для обсуждаемых приложений, поскольку позволяют ассоциировать несколько объектов сцены с каждой ячейкой и, тем самым, непосредственно разрешать запросы пространственной локализации.

— Многоуровневые сетки, в большинстве приложений представленные квадрантными (quadtrees) и октальными (octrees) деревьями, и реже — вложенными тетраэдральными и гексагональными блоками. Как правило, в данных иерархически организованных сетках применяется рекурсивное разбиение пространства на конгруэнтные или подобные ячейки с простыми вычислительными правилами определения пересечений с объектами сцены и областями поиска. Алгоритмы поиска используют эти правила для принятия решения о необходимости распространения анализа в дочерние ячейки. Тем самым, большинство ячеек исключаются из рассмотрения, а время исполнения запроса локализации определяется размером области поиска и глубиной представления многоуровневой сетки в ней. Заметим, что обсуждаемое семейство методов существенно шире и представлено также многочисленными вариантами ортогональных многоуровневых сеток. Например, в обобщенных ATree структурах допускается разбиение D -мерного пространства на переменное число дочерних ячеек 2^d, $1 \le d \le D$, объединяющих смежные D-мерные кубы [33]. Способ разбиения выбирается индивидуально для каждой ячейки с целью наиболее компактно локализовать объекты сцены. В случаях, когда значительная доля объектов пересекается с внутренними вершинами, ребрами или гранями и должны быть приписаны нескольким смежным ячейкам, ATree структуры получают ряд преимуществ. Варианты разбиений, используемые в близких BinTree [34], AHC (adaptive hierarchical coding) [35], X-Y [36], Puzzle деревьях [37] и Treemap структурах [38], предусматривают циклическую и ациклическую смену осей пространственного разбиения на каждом уровне иерархии. Данные структуры лишены существенного недостатка kD деревьев, проявляемого при необходимости разбить ячейку несколькими параллельными гиперплоскостями и приводящего к большей глубине дерева поиска за счет вставки дополнительных вершин. Развертывание перечисленных структур происходит приблизительно одинаково путем последовательной вставки объектов в сцену и инкрементальной коррекции многоуровневого представления сетки. Процедура разбиения пространства применяется рекурсивно до тех пор, пока не достигнуты требуемые условия на листьях дерева. Например, в приложениях моделирования сцен процедура завершается при невозможности более четко локализовать объекты в листовых вершинах или при их относительно небольшом количестве. В

приложениях пространственного анализа (spatial reasoning) с использованием октальных деревьев наполненности (occupancy octrees) обычно используются условия достижения полностью занятых или полностью пустых ячеек, естественно, с заданной точностью представления границ объектов и допустимой ошибкой определения пересечений.

В следующих разделах остановимся более подробно на характерных представителях каждого выделенного класса методов и особенностях реализации типовых запросов в них.

3. Методы, основанные на кластеризации объектов

Принцип кластеризации предполагает многоуровневую композицию объектов в виде сбалансированного дерева, листья которого соответствуют отдельным объектам, нелистовые вершины — выделенным кластерам, а корень дерева — всей сцене. С каждой нелистовой вершиной ассоциируется $[m, M]$ дочерних вершин, где фиксированные параметры $m \ge 1$, $M \ge m$ ограничивают их число снизу и сверху. Структуры данного семейства являются развитием B-деревьев, широко применяемых в СУБД общего назначения для хранения во внешней памяти со страничным доступом, однако предполагают особые способы пространственной кластеризации. Так, например, гильбертово R-дерево [39] является B+-деревом, в котором для кластеризации объектов применено упорядочивание Пиано-Гильберта (см. рис. 2).

Параметр M обычно выбирается, исходя из размера страницы файловой системы. Чем больше его значение, тем более кустистым и менее глубоким будет полученное дерево, что повышает эффективность пространственной локализации объектов. Этими же соображениями объясняется интерес к заполненным (packed) структурам. Формирование подобных структур может происходить как снизу вверх, так и сверху вниз. В первом случае объекты рекурсивно объединяются на каждом уровне дерева таким образом, что каждая нелистовая вершина содержит ровно M дочерних. Такая схема используется, в частности, в упакованных гильбертовых R-деревьях (packed Hilbert R-tree) [40]. Во втором случае объекты сцены рекурсивно разбиваются на M групп с одинаковым количеством объектов в каждой до тех пор, пока в группе не окажется менее M объектов. Такая схема иногда применяется в STR-методе (sort tile recurse method) [41]. Важно отметить, обе схемы предполагают использование дополнительных пространственных соображений при кластеризации объектов и обеспечении надлежащей сбалансированности деревьев.

Тем не менее, практическое применение заполненных структур довольно ограничено при наличии динамики в сценах и необходимости обновлений,

часто сводящихся к полному перестроению дерева. Для приложений, оперирующими статическими и квазистатическими сценами, как, например, ГИС, использование заполненных структур вполне оправданно. Пример подобной структуры с параметром заполнения $M = 3$ для сцены, состоящей из 9 объектов, приведен на рис. 3.

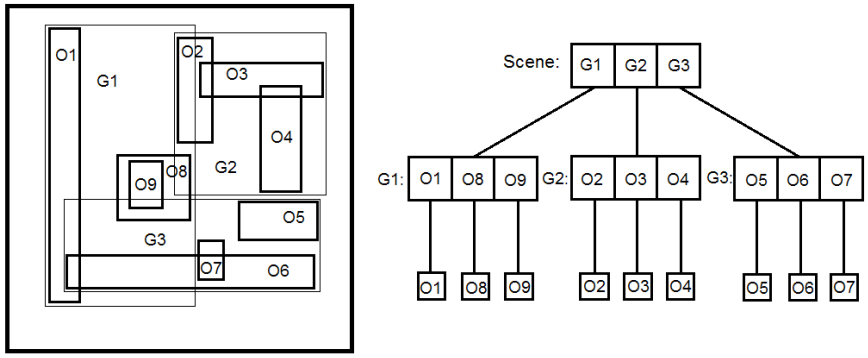


Рис. 3. Пример заполненной структуры параметром заполнения $M = 3$.

Для более динамичных сцен на практике применяются структуры с нижней границей заполнения $m = 0.3 - 0.4M$. Такое значение порога, установленное эмпирически в ряде экспериментов, несколько ниже, чем величина $m = 0.5M$, обычно применяемая для B-деревьев, и величина $m = 0.66M$, устанавливаемая для B*-деревьев. Уменьшение нижнего порога объясняется значительными вычислительными затратами на пространственную кластеризацию объектов, которые, тем не менее, могут быть частично скомпенсированы релаксацией требований наполненности и отложенными операциями перебалансировки дерева.

Для непосредственного снижения затрат на пространственную кластеризацию и локализацию объектов обычно привлекают методы ограничивающих объемов, которые составляют традиционные главы вычислительной геометрии. Данные методы предусматривают использование альтернативных геометрических представлений объектов, обычно основанных на упрощенной аппроксимации их границ. С их использованием удается заменить вычислительно сложные процедуры определения взаимных пересечений объектов аналогичными тестами для ограничивающих объемов и, тем самым, выносить быстрый отрицательный вердикт в тех случаях, когда объемы действительно не пересекаются.

Чаще всего для этой цели применяют описанные вокруг объекта прямоугольные параллелепипеды с ребрами параллельными координатным осям (AABB от английского axis-aligned bounding box). Известны попытки использования описанных сфер, цилиндров, выпуклых оболочек,

многогранников с заданным количеством возможных ориентаций граней k-DOP (k-discrete orientation polyhedron) [42] (используемый, в частности в cell-tree [43]), произвольно ориентированных параллелепипедов (OBB от oriented bounding box) [44] . Однако использование более точной аппроксимации неизбежно приводит к более высоким вычислительным затратам. Так пересечение двух AABB параллелепипедов требует всего 6 операций сравнения, в то время как пересечение двух OBB параллелепипедов требует от 89 до 252 операций в лучшем и худшем случае соответственно [45]. Поэтому применение вычислительно простых методов представляется наиболее оправданным для обсуждаемого класса приложений. В дальнейшем мы будем полагать, что объекты сцены и сформированные на их основе кластеры представляются своими ограничивающими объемами так, что с каждой вершиной дерева композиции объектов ассоциирован соответствующий AABB параллелепипед.

Способ пространственной кластеризации является ключевым алгоритмическим элементом, определяющим эффективность поиска на основе индексированных иерархических структур. Чтобы проиллюстрировать этот факт, рассмотрим типовой запрос пространственной локализации — поиск объектов, имеющих непустое пересечение с заданной областью пространства. Исполнение такого запроса сводится к обходу вершин дерева, чьи ограничивающие объемы пересекают заданную область, сверху вниз. Если выявлено пересечение для некоторой вершины дерева, то обход распространяется на всех ее детей. В наихудшем случае, когда область поиска охватывает всю сцену, неизбежному анализу подлежат все вершины дерева. Естественно, что кластеры объектов могут формироваться на основе условий естественной пространственной декомпозиции сцены, а могут следовать и иным требованиям. В первом случае удается существенно сократить количество анализируемых вершин и время исполнения типовых запросов. В качестве таких условий обычно используют критерий минимального покрытия, предусматривающий минимизацию суммарного ограничивающего объема для формируемого кластера объектов, и критерий минимального перекрытия, препятствующий значительному взаимному пересечению выделенных кластеров. На первый взгляд, оба критерия обеспечивают компактное пространственное представление для каждого кластера объектов. Тем не менее, критерии не эквивалентны, что демонстрирует пример, представленный на рис. 4. Результат кластеризации на основе критерия минимального перекрытия (см. рис. 4а) существенно отличается от аналогичного результата на основе критерия минимального покрытия (см. рис. 4б). Хотя первый критерий приводит к более рациональной пространственной кластеризации, спекулятивный комбинаторный анализ перекрытий оказывается довольно затратным и при практической реализации R, R+, R*-деревьев чаще применяется второй критерий, основанный на минимизации покрытия. Иногда используются комбинированные критерии, основанные на взвешенной оценке объемов покрытия и перекрытия [46], [47].

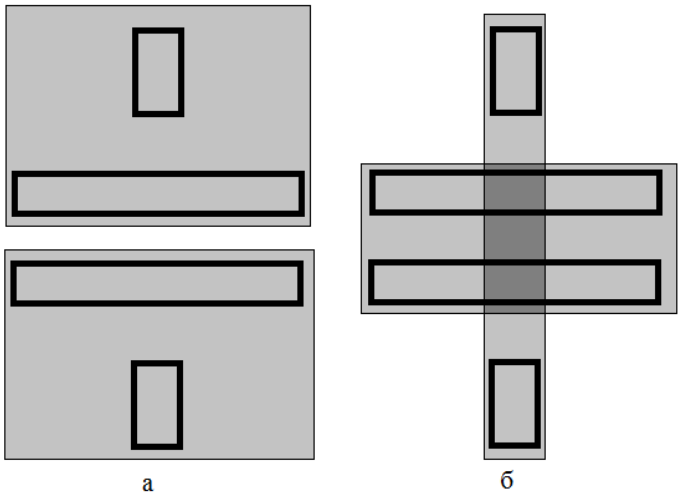


Рис. 4. а) Результат кластеризации объектов на основе минимального перекрытия. б) Результат кластеризации объектов на основе минимального покрытия.

Задача о минимальном покрытии носит комбинаторный характер, поэтому вычислительная сложность наивного алгоритма – $O(2^M)$, где M – количество объектов, которые необходимо сгруппировать. Однако разработаны эффективные методы, решающие задачу о минимальном покрытии за существенно меньшее время. В работе [48] описан метод, который в пространственно двумерном и трехмерном случае приводит к более оптимистическим оценкам сложности $O(M^3)$ и $O(M^5)$ соответственно. В общем случае асимптотическая оценка сложности этого метода составляет $O(dM \log M + d^2 M^{2d-1})$, где d — размерность пространства. В работе [49] предложен метод с лучшей оценкой сложности $O(M^2)$ для двумерного случая, однако его применение не гарантирует балансировки структуры пространственного индекса. Оба упомянутых метода допускают параметризацию с помощью монотонной функции оценки группового покрытия и обеспечивают поиск точного решения задачи. Ряд авторов констатирует, что вычислительные затраты на точный поиск минимального покрытия не всегда оправданы в контексте общей проблемы пространственной кластеризации и эвристические подходы, в частности, метод разбиения пространства гиперплоскостями [50] и метод группирования по ключевым объектам, приводят к вполне удовлетворительным результатам [49].

Рассмотрим более подробно метод разбиения пространства гиперплоскостями. Идея метода состоит в делении пространства сцены плоскостями, перпендикулярными одной из осей. Принадлежность объекта тому или иному кластеру определяется его положением относительно секущих плоскостей, а именно, максимальной координатой ограничивающего параллелепипеда вдоль выбранной оси разбиения. Плоскости проводятся таким образом, чтобы обеспечить наиболее равномерное распределение объектов по кластерам. Для этой цели объекты сортируются в порядке возрастания максимальной координаты ограничивающего параллелепипеда и последовательно объединяются в необходимое число кластеров. Аналогичные попытки предпринимаются для каждой из осей, а окончательное разбиение выбирается, исходя из критериев равномерности распределения объектов по кластерам, минимизации покрытия или минимизации перекрытия в соответствии с назначенными приоритетами. Анализ сложности метода приводит к оценке $O(dM)$ при размерности пространства d. Результаты экспериментов показывают [50], что метод демонстрирует хорошие результаты для сцен с равномерно распределенными объектами.

Следует отметить, что близкий метод используется для минимизации перекрытия в R*-деревьях. Однако выбор положения плоскости разбиения происходит, исходя из условия минимальности среднего периметра ограничивающих параллелепипедов сформированных кластеров. Такой метод требует более высоких вычислительных затрат с оценкой $O(M \log M)$, однако несколько уменьшает расходы на хранение структур индексации [10], [51].

Интересную альтернативу составляют эвристики, которые реализуют инкрементальную технику формирования кластеров объектов. Вначале выбирается несколько характерных объектов, каждый из которых образует свой собственный кластер. Затем остальные объекты приписываются к одному из них в соответствии с критерием минимизации покрытия. При образовании кластеров следует выбирать наиболее удаленные и протяженные объекты. Для этого обычно применяются условие максимальности объема параллелепипеда, описанного вокруг пары образующих вершин, и условие максимальной разницы в одной из координат объектов, нормированной на ширину ограничивающего параллелепипеда. При использовании первого условия пара образующих вершин может быть выбрана за $O(M^2)$ операций, при применении второго — за $O(M)$ операций, что существенно улучшает ситуацию с общими затратами на кластеризацию. Вычислительные эксперименты показывают, что приведенные условия выбора образующих объектов незначительно влияют на итоговое время исполнения типовых запросов [8], поэтому применение второго условия, безусловно, выглядит предпочтительнее для практических реализаций. На рис. 5 приведен пример сцены из 9 объектов и R-дерево с параметрами наполненности $m = 2$,

$M = 3$, построенное с помощью инкрементальной кластеризации на основе данного условия.

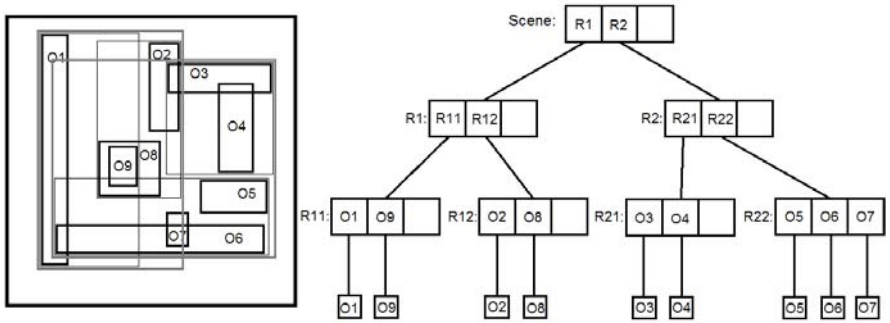


Рис. 5. Пример сцены из 9 объектов и R-дерево с параметрами наполненности $m = 2$, $M = 3$, построенное с помощью инкрементальной кластеризации.

Наличие динамики в сцене может существенно поменять оценку рассмотренных методов кластеризации, поскольку появление в сцене новых объектов, удаление или перемещение существующих объектов вызывает необходимость перманентных обновлений структур индексации. Поскольку подобные события неизбежно влекут изменения в площади покрытия и перекрытия ограничивающих параллелепипедов (причем эти изменения могут зависеть как от очередности событий в сцене, так и от их пространственного распределения), эффективность исполнения запросов и производительность программного приложения могут существенно деградировать в сценах с интенсивной динамикой.

Один из способов решения этой проблемы состоит в сохранении сбалансированности дерева, даже если события происходят в одной и той же области пространства и влекут переполнение или исчерпание соответствующих вершин дерева. При переполнении вершины проводится ее пространственное расщепление и часть объектов переносится в соседнюю свободную и неизмененную на данной транзакции вершину того же уровня. Если соседняя вершина переполнена или изменена, то предпринимается попытка перенести их в следующую соседнюю вершину, а если переполнены все соседи — то она вставляется заново в верхнюю родительскую вершину.

Данный способ, называемый повторной вставкой (forced reinsertion), широко применяется при реализации R*-деревьев. При переполнении одной из вершин часть ее объектов (обычно 30% от максимально допустимого заполнения [10]) поочередно вставляются в дерево. Для повторной вставки выбираются объекты наиболее удаленные от центра ограничивающего объема

расщепляемой вершины. Как было отмечено выше, очередность событий может существенно влиять на организацию дерева и эффективность исполнения запросов. Поэтому на практике, чтобы придать некоторый детерминизм данной процедуре, следуют одной из двух дисциплин, предусматривающий порядок повторной вставки объектов, начиная от самых дальних к ближним (far-reinsert) или от самых ближних к дальним (close-reinsert). Чтобы избежать попыток повторного включения объекта в саму расщепляемую вершину или в соседние переполненные вершины, применение процедуры ограничивается однократной попыткой на каждом уровне дерева.

Несмотря на многочисленные подходы к пространственной кластеризации и обеспечению сбалансированности структур индексации, область их практического применения в случае динамических сцен довольно ограниченна из-за значительных вычислительных затрат. Для их содержательной адаптации к динамическим сценам были предложены такие специальные структуры, как параметрические R-деревья [52], TPR-tree [53] и TPR*-tree [54]. Основная идея, лежащая в их основе, заключается в использовании описываемых объемов движущихся объектов. Такие объемы могут быть насчитаны на любом заданном временном интервале с учетом всех промежуточных положений объекта и применяться при пространственной кластеризации сцены и разрешении запросов на данном интервале. Естественно, что использование подобной аппроксимации динамического объекта существенно расширяет границы его локализации в любой фиксированный момент времени и неизбежно приводит к дополнительным расходам. По-видимому, разумный компромисс достигается в случае, если временной интервал моделирования динамической сцены разбит на участки, на каждом из которых применяется своя структура индексации с учетом объемов, описываемых движущимися объектами локально на каждом отдельном участке. К сожалению, применимость данных методов ограничена сценами с предопределенным характером динамики, когда положения всех объектов в любой временной точке моделирования известны заранее.

Обсудим вопросы реализации типовых операций с использованием структур объектной кластеризации. Псевдокод операции поиска объектов в заданной пространственной области представлен на рис. 6а.

```
procedure FIND_OBJECTS_IN_VOLUME(T,V,R)
pointer node T
pointer volume V
pointer collection R

if(IS_INTERSECTING(VOLUME(T), V ) )
        {
        if(IS_LEAF(T))
                {
                ADD(R,T)
```

```
        return
        }

    for_each(node child in CHILDREN(T))
        FIND_OBJECTS_IN_VOLUME(child, V, R)
    }
```

Рис. 6а. Псевдокод операции поиска объектов в заданной области.

Приведенный алгоритм осуществляет рекурсивный обход дерева сверху вниз, начиная с корневой вершины. Функция IS_INTERSECTING возвращает статус принадлежности вершины заданному объему. В случае, когда вершина не принадлежит заданному объему, обход приостанавливается, поскольку все дочерние вершины априори не попадают в заданную область, иначе статус ее принадлежность листовым вершинам устанавливается при помощи процедуры IS_LEAF. Если вершина является листовой, то она ассоциируется с объектом сцены и этот объект добавляется к результатам запроса, в противном случае процедура вызывается для всех ее дочерних вершин. Вычислительная сложность этого алгоритма составляет $O(N)$ в худшем случае, поскольку каждая вершина дерева пересекается с заданным объемом и для выполнения запроса необходимо проанализировать все объекты.

На рис. 6б приведен набор функций, осуществляющий поиск ближайшего соседнего объекта.

```
procedure FIND_NEAREST_NEIGHBOUR(T,R)
pointer node T
pointer collection R

pointer node P=PARENT(T)
float D = ∞
return FIND_NEAREST_NEIGHBOUR_IN_ANCESTORS(P,T,T,D,R)


procedure FIND_NEAREST_NEIGHBOUR_ANCESTORS(T,C,N,D,R)
pointer node T
pointer node C
pointer node N
pointer float D
pointer collection R

for_each(node child in CHILDREN(T))
        {
        if(child != C)
                {
                FIND_NEAREST_LEAF(child,N,D,R)
```



```
                }
        }
    if(IS_ROOT(T))
        return

    return
FIND_NEAREST_NEIGHBOUR_ANCESTORS(PARENT(T),T,N,D,R))

    procedure FIND_NEAREST_LEAF(T,N,DST,R)
    pointer node T
    pointer node N
    pointer float D
    pointer collection R

    float distance = DISTANCE(T,N)
    if(distance>DST)
        return

    if(IS_LEAF(T))
        {
        if(distance<DST)
                {
                CLEAR(R)
                DST = distance;
                }
        ADD(R,T)
        return
        }

    for_each(node child in CHILDREN(T))
        {
        FIND_NEAREST_LEAF(child,N,DST,R)
        }
```

Рисунок 6б. Псевдокод алгоритма поиска ближайших объектов.

Функция FIND_NEAREST_NEIGHBOUR принимает в качестве аргумента объект и возвращает коллекцию ближайших соседей. Для этого используется вспомогательная функция FIND_NEAREST_NEIGHBOUR_IN_ANCESTORS, принимающая в качестве аргументов текущую вершину дерева T, предыдущую вершину дерева C, вершину объекта N, а также найденное минимальное расстояние DST и соответствующий ему результат в виде коллекции соседних объектов R. Она осуществляет рекурсивный поиск

ближайших объектов в текущей вершине, всех ее дочерних вершинах и затем распространяет поиск на родителей. Для корректной работ алгоритма переменные устанавливаются следующим образом: DST=∞, R=Ø. Поиск ближайших объектов среди дочерних вершин осуществляется с помощью вспомогательной процедуры FIND_NEAREST_LEAF. Существенно, что рекурсивный обход в ней прекращается в случае, если расстояние до текущей вершины больше, чем найденное ранее расстояние до ближайшего объекта. В наихудшем случае для выполнения запроса придется пройти все вершины дерева, поэтому вычислительная сложность алгоритма составляет $O(N)$.

Несмотря на многообразие подходов к кластеризации объектов, ограничимся при дальнейшем рассмотрении классическим R-деревом. Оценки для других подобных структур этого семейства, таких как R*-дерево, оказываются значительно выше.

```
procedure INSERT(T,O,M)
pointer node T
pointer object O
integer M

if(STORE_OBJECTS(T))
        {
        ADD(T,O)
        if(NUM_STORED(T) > M)
                SPLIT(T,M)
        return
        }

return INSERT(FIND_BEST_CHILD(T,O),O,M)


pointer node procedure FIND_BEST_CHILD(T,O)
pointer node T
pointer object O

pointer node result;
float delta = ∞
for_each(node child in CHILDREN(T))
        {
        volume selfVolume = VOLUME(child)
        volume volumeWithObject = EXPAND(selfVolume,
VOLUME(O))

        if(volumeWithObject – selfVolume < delta)
                result=child
```



```
        }
return result


procedure SPLIT(T,M)
pointer node T
integer M

pointer node P = PARENT(T)
if(P==NULL)
        P=CREATE_NEW_ROOT()
else
        REMOVE_FROM_PARENT(P,T)
FIND_GROUPS(T,L,R)
ADD(P,L)
ADD(P,R)
if(NUM_STORED(P)>M)
        return SPLIT(P,M)

return
```

Рисунок 6в. Псевдокод операции вставки объекта в R-дерево.

На рис. 6в. приведен псевдокод процедуры INSERT, которая вставляет объект O в R-дерево с параметром заполнения M. Для этого статус текущей вершины проверяется при помощи функции STORE_OBJECTS, которая возвращает логическое значение TRUE, если в вершине хранятся ссылки на объекты, и FALSE в противоположном случае. Если функция STORE_OBJECTS возвращает TRUE, выполняется вставка нового объекта и, если не нарушается верхняя граница наполненности вершин дерева, то процедура завершается. В противном случае объекты, содержащиеся в вершине, разбиваются с помощью вспомогательной процедуры FIND_GROUPS на две группы, которые и формируют новую пару вершин. Существенно, что при этом родительская вершина может оказаться переполненной и эту процедуру следует повторить на верхнем уровне иерархии. Если функция STORE_OBJECTS возвращает FALSE, то с помощью вспомогательной функции FIND_BEST_CHILD ищется дочерняя вершина, в которую будет вставлен объект. Приведенный псевдокод реализует критерий минимального покрытия, предполагающий поиск дочерней вершины, вставка в которую приведет к минимальному увеличению ее объема. В наихудшем случае каждая из пройденных вершин окажется переполненной и подлежит последующей перегруппировке, тогда сложность операции вставки будет составлять $O(M \log N)$ в предположении, что для кластеризации объектов

использовалась рассмотренная выше процедура с вычислительной трудоемкостью $O(M)$.

```
procedure REMOVE(T,m,M)
pointer node T
integer m
integer M

pointer node P = PARENT(T)
REMOVE_FROM_PARENT(P,T)

if(IS_ROOT(P))
        return

if(NUM_STORED(P)>=m)
        return

pointer node PP = PARENT(P)
REMOVE_FROM_PARENT(PP,P)
for_each(node child in P)
        {
        pointer node NP = FIND_BEST_CHILD(PP,child)
        ADD(NP,node)
        if(NUM_STORED(NP) > M)
                SPLIT(NP,M)
        }

if(NUM_STORED(PP)<m)
        REMOVE(PP,m)

return
```

Рисунок 6д. Псевдокод операции удаления объекта из R-дерева.

Рассмотрим процедуру удаления объекта из R-дерева, псевдокод которой приведен на рис. 6д. В качестве параметра она принимает удаляемую вершину T, а также параметры заполнения дерева m и M. В теле процедуры выполняется удаление ссылки на удаляемую вершину из ее родителя. В случае, когда число объектов, оставшихся в родительской вершине, лежит в интервале $[m, M]$, процедура завершается. В противном случае родительская вершина удаляется, а ее объекты перемещаются в смежные вершины. В случае их переполнения осуществляется рекурсивное обновление дерева, как при вставке объекта. В наихудшем случае вычислительная сложность операции удаления составит $O(M \log N)$.

4. Методы, основанные на декомпозиции пространства

Структуры индексации, обсуждаемые в настоящем разделе, реализуют фундаментальный принцип декомпозиции пространства. Следуя данному принципу, каждый объект сцены ассоциируется с одной или несколькими ячейками пространства в зависимости от его положения. Способ ассоциирования, при котором объект содержит ссылки на ячейки пространства, называется в англоязычной литературе явным (explicit). Данный способ позволяет быстро установить ячейки пространства, в которых находится заданный объект, и эффективно выполнить ряд операций, например, удалить объект из сцены без каких-либо обновлений структур индексации. Однако выполнение других запросов, таких как, поиск объектов в заданной области пространства или поиск соседей, становится трудоемким, поскольку требуется проанализировать положение всех объектов в сцене перед тем, как сформировать окончательный результат. Поэтому для обсуждаемого класса приложений более перспективным представляется неявный (implicit) способ ассоциирования, предполагающий, что ячейки пространственного разбиения хранят ссылки на объекты, которые в них содержатся. В этом случае пространственные запросы разрешаются более естественным образом на основе анализа относительного положения ячеек разбиения. Проблема установления ячеек, принадлежащих заданному объекту, также не является критичной, если объекты сцены представлены своими ограничивающими объемами (AABB, OBB, k-DOP) и существуют быстрые вычислительные процедуры локализации этих объемов в ячейках разбиения.

В зависимости от способа задания формы ячеек пространства выделяют полигональные и аналитические разбиения. При полигональном разбиении ячейки пространства представляются многогранниками, в то время как аналитическое разбиение предполагает использование алгебраических уравнений для описания их формы. На практике получили распространение полигональные, топологически регулярные разбиения, ячейки которых геометрически подобны или даже конгруэнтны друг другу.

Чтобы контролировать глубину иерархических разбиений, допустимый размер ячеек обычно ограничивается заданным параметром. Ячейки минимального размера называются единичными. В зависимости от формы единичных ячеек пространственные разбиения могут обладать различными свойствами [55]. Определенное распространение получила сотовая структура (septree) [56], единичные ячейки которой имеют форму правильных шестиугольников. Для ландшафтных сцен, двумерные объекты которых располагаются на некоторой сфере, известны случаи применения в качестве единичных ячеек треугольников. Поверхность сферы в этом случае аппроксимируется икосаэдром, грани которого иерархически разбиваются на треугольные ячейки, как показано на рис. 8.

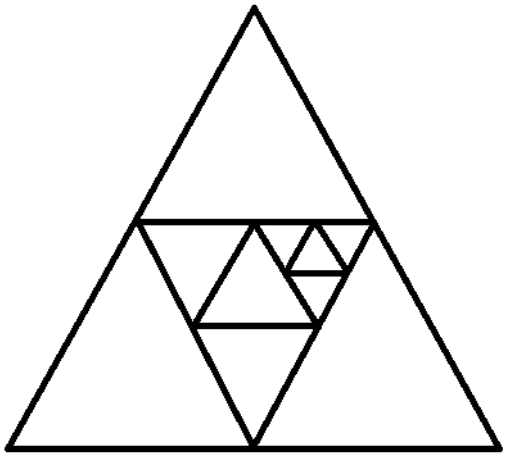


Рис. 8. Иерархическая декомпозиция двумерной области пространства треугольными ячейками.

Пространственные разбиения, использующие одни и те же геометрические построения, могут допускать различные структуры доступа к своим ячейкам (data access structures). В качестве таких структур могут применяться многомерные массивы ссылок, хэш-таблицы с соответствующими функциями пространственного хэширования и деревья. Остановимся более подробно на их особенностях.

Преимущества многомерных массивов проявляются в тех случаях, когда может быть определена функция явного преобразования пространственных координат в соответствующие индексы массива ссылок. При наличии подобной функции быстро осуществляется локализация объектов в заданной области, однако поиск соседей затруднен из-за необходимости организации процедуры распространения по смежным ячейкам, что может быть затратным с вычислительной точки зрения. Другой недостаток массивов связан с необходимостью хранения в памяти записей для всех ячеек пространства, полученных при разбиении, независимо от наличия в них объектов сцены. Для разреженных сцен, в которых объекты распределены неравномерно, это может приводить к существенным накладным расходам по памяти.

Для устранения данного недостатка структура доступа должна быть организована таким образом, чтобы исключать использование пустых ячеек. Определяется порядок обхода ячеек, например, с использованием кривых заполнения пространства, и непустые ячейки нумеруются в соответствии с ним. Установленный порядок позволяет использовать традиционные структуры поиска, такие как AVL-деревья [57], красно-черные деревья [58], B-деревья, хэш-таблицы [59], а присвоенный номер служит ключем.

Поиск объектов в заданной области сводится к поиску экстремальных ячеек, принадлежащих области и имеющих минимальный и максимальный порядковые номера. Ячейки с номерами, лежащими в данном диапазоне, находятся в результате запроса к соответствующей структуре поиска, и далее индивидуально подвергаются точной проверке принадлежности заданной области. Поиск ближайших соседей также реализуем на основе рассмотренных структур доступа [60].

Однако более перспективным для решения обсуждаемого круга проблем нам представляется использование деревьев, вершины которых ассоциированы с ячейками соответствующих уровней пространственного иерархического разбиения. Терминальным условием для роста дерева в глубину выступает ограничение минимального количества объектов, содержащихся в каждом листе. Так в случае применения иерархической декомпозиции пространства размерности d на 2^d равных частей плоскостями, перпендикулярными каждой из координатных осей, получается обобщенное октальное дерево. В трехмерном случае каждая нелистовая вершина октального дерева содержит восемь дочерних вершин, ассоциированных с соответствующими октантами пространственного разбиения. Верхняя вершина дерева соответствует AABB параллелепипеду всей сцены. Пример пространственной декомпозиции и полученного октального дерева приведен на рис. 8.

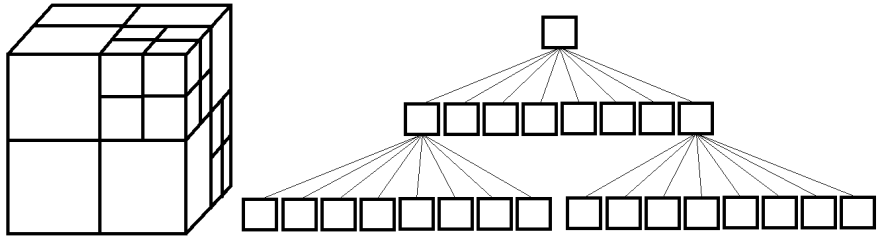


Рис. 8. Пример пространственной декомпозиции и соответствующего ей октального дерева.

Главным достоинством октальных структур является их простота, обусловленная априори известным положением секущих плоскостей и, как следствие, исключением дополнительного пространственного анализа при проведении разбиения. Основным недостатком октальных структур является несбалансированность дерева в случае неравномерного распределения объектов по сцене. Поэтому время исполнения типовых запросов может существенно варьироваться в зависимости от области сцены, подлежащей анализу. Другим недостатком является обязательное хранение восьми дочерних вершин даже в случае, если объекты локализуются только в одной их них.

Проблема избыточности представления отчасти решена в ATree структурах, в которых предполагается выбор приоритетных осей разбиения в каждой вершине дерева. Правильный выбор осей может значительно уменьшить общее количество вершин дерева. В деревьях Bintree данная проблема решается путем бинарной декомпозиции, основанной на циклическом выборе осей разбиения. В случае произвольного выбора данное дерево обобщается до AHC структуры (adaptive hierarchical coding). Если в Bintree деревьях сохранить цикличность выбора осей, но допустить выбор положения секущих плоскостей, то достигается обобщение до kD-деревьев. Существенной чертой данных структур является возможность обеспечить сбалансированность дерева. В обобщенных kD-деревьях снимается условие цикличности, а в X-Y деревьях, treemap структурах и puzzle деревьях допускается разбиение несколькими параллельными секущими плоскостями в каждой вершине. Тем самым, достигается сбалансированность дерева и наполненность вершин. Однако это может быть достигнуто в результате трудоемкого анализа расположения объектов и релевантного выбора плоскостей сечений. Попытки обеспечить сбалансированность данных структур в случае динамических сцен оказываются безнадежными вследствие того, что незначительное ускорение запросов локализации не компенсируется существенными затратами на обновление структур индексации.

Вернемся к ключевым вопросам реализации октальных деревьев. Одним из интересных аспектов является способ разрешения коллизий, связанных с неоднозначностью приписывания объектов, лежащих на секущих плоскостях, тому или иному дочернему октанту. Существует несколько альтернатив. В случае приписывания объекта нескольким октантам возрастают расходы на хранение избыточных ссылок, а также дополнительные вершины подвергаются анализу в ходе выполнения типовых запросов.

Другой альтернативой является возможность ассоциировать такие объекты непосредственно с разбиваемой вершиной. В этом случае каждый объект оказывается ассоциированным только с одним октантом, что упрощает реализацию операций добавления и удаления объектов. Данный способ применяется, в частности, в структуре со строгой многоуровневой локализацией, известной в литературе как MX-CIF octree [61]. Однако при ее использовании существует вероятность, что большая часть объектов сцены попадет на секущие плоскости верхних октантов и окажется приписанной им. В этом случае дерево вырождается и его дальнейшее использование для разрешения пространственных запросов теряет смысл. Проблема особенно усугубляется для объектов, соразмерных габаритам единичных октантов, но локализующихся на верхних уровнях дерева.

В работах [62], [63], [64] данная проблема решается путем пропорционального увеличения габаритов каждого октанта в $p > 0$ раз вдоль каждой из осей. Параметр p выбирается, исходя из размеров объектов, подлежащих более строгой локализации. Недостатком данного метода является перекрытие

октантов и необходимость анализа большего числа вершин при типовых запросах (этот аспект довольно близок проблеме пространственного перекрытия в методах кластеризации объектов). Для преодоления этого недостатка в работе [64] было предложено смещать положение плоскостей разбиения относительно центра разбиваемого октанта на четверть его размера вдоль каждой из осей. В этом случае исключается пересечение дочерних октантов, однако увеличивается число дочерних ячеек. Можно показать, что при выборе параметра $p = 1/2$ любой объект сцены локализуется в ячейке, габариты которой не превышают размеры объекта более чем в 4 раза. Несмотря на увеличение числа вершин и усложненный пространственный анализ, данный метод хорошо компенсирует недостатки классических октальных деревьев поскольку обеспечивает точную локализацию объектов и позволяет установить однозначное соответствие объектов вершинам октального дерева.

На рисунках 9а-9д приведены псевдокоды алгоритмов для выполнения основных пространственных запросов к рассмотренным в этом разделе MX-CIF октальным деревьям со строгой многоуровневой локализацией объектов, где в качестве критерия разбиения используется ограничение максимально допустимого количества объектов M, хранимого в вершине.

```
procedure FIND_OBJECTS_IN_VOLUME(T,V,R)
pointer node T
pointer volume V
pointer collection R

if(IS_INTERSECTING(T,V) )
        {
        for_each(object o in OBJECTS(T))
                {
                if(IS_INTERSECTING(o,V)
                        ADD(R,o)
                }

        for_each(node child in CHILDREN(T))
                FIND_OBJECTS_IN_VOLUME(child, V, R)
        }
```

Рис. 9а. Псевдокод операции поиска объектов в заданной области.

Рассмотрим псевдокод алгоритм поиска объектов в заданной области сцены, изображенный на рис. 9а. Он незначительно отличается от приведенного для структур, использующих композицию объектов. Отличие связано с тем, что нелистовые вершины так же содержат ссылки на объекты подлежащие анализу. Как и в случае R-дерева, в наихудшем случае анализу подвергнутся

все объекты, входящие в сцену, поэтому вычислительная сложность запроса составляет $O(N)$.

Псевдокод алгоритма поиска ближайших соседних объектов приведен на рис. 9б. Основная процедура FIND_NEAREST_NEIGHBOURS принимает в качестве аргумента объект и возвращает множество его ближайших соседей. Используемая в ней вспомогательная функция NODE по данному объекту находит ассоциированную с ним вершину V октального дерева. Для этого рекурсивный обход октального дерева сверху вниз продолжается до тех пор, пока данный объект целиком содержится в рассматриваемой вершине дерева. Последняя вершина, удовлетворяющая этому условию, является искомой. Вычислительная сложность такой процедуры составляет $O(D)$, где D — глубина октального дерева. Также существенной деталью является возможность задать начальную аппроксимацию расстояния до ближайших объектов. Действительно, если при построении октального дерева вершина была разбита, то в ней должен содержаться, по меньшей мере, $M+1$ объект. Следовательно, родитель вершины V содержит еще, как минимум, один объект, расстояние до которого не превышает удвоенной длины диагонали V. Это значение и выбирается в качестве начальной аппроксимации. В остальном алгоритм поиска ближайших соседей в октальных деревьях концептуально повторяет описанный алгоритм поиска ближайших соседей в R-деревьях. В наихудшем случае трудоемкость выполнения этого запроса составляет $O(N)$.Необходимо отметить схожесть приведенного алгоритма с алгоритмом поиска октантов, смежных данному. В работе [65] было показано, что для поиска соседних вершин заданной, лежащих на том же уровне иерархии, в среднем необходимо 4 раза проследовать по ссылке, связывающей родительскую и дочернюю вершины. Это дает основание полагать, что в среднем трудоемкость поиска ближайших соседних объектов будет значительно ниже.

```
procedure FIND_NEAREST_NEIGHBOURS(O, R)
pointer object O
pointer collection R

pointer node V=NODE(O)
float D = 2*DIAG(V)
FIND_NEAREST_LEAF(O,V,D,R)
FIND_NEAREST_NEIGHBOUR_ANCESTORS(PARENT(V),O,V,D,R)

procedure FIND_NEAREST_LEAF (O,N,D,R)
pointer object O
pointer node N
pointer float D
pointer collection R
```



```
if(DISTANCE(O,N)>D)
        return

for_each(object o != O  in OBJECTS(N))
        {
        float distance = DISTANCE(o, O)
        if(D< distance)
                continue
        else
                {
                if(D > distance)
                        {
                        CLEAR(R)
                        D = distance
                        }
                ADD(R,o)
                }
        }
for_each(node child in CHILDREN(N))
        FIND_NEAREST_LEAF(O,child,D,R)

procedure FIND_NEAREST_NEIGHBOUR_ANCESTORS (N,O,E,D,R)
pointer node N
pointer object O
pointer node E
pointer float D
pointer collection R

for_each(object o in OBJECTS(N))
        {
        float distance = DISTANCE(o, O)
        if(D< distance)
                continue
        else
                {
                if(D > distance)
                        {
                        CLEAR(R)
                        D = distance
                        }
                ADD(R,o)
                }
        }
```

```
for_each(node n != E in CHILDREN(N))
        FIND_NEAREST_LEAF (O,n,D,R)

if(IS_ROOT(N))
        return

return                  FIND_NEAREST_NEIGHBOUR_ANCESTORS
(PARENT(N),O,N,D,R)
```

Рис. 9б. Псевдокод алгоритма поиска ближайших объектов.

Вставка объекта в октальное дерево осуществляется процедурой INSERT, псевдокод которой приведен на рис. 9в. В качестве параметров она объект и вершину октального дерева, в которую он вставляется. Используемая в методе вспомогательная функция COUNT_OVERLAPPED_CHILDREN возвращает число дочерних октантов заданного, пересекающихся с данным объектом. Если их более одного, то объект вставляется в текущую вершину дерева при помощи процедуры ADD, иначе вставка рекурсивно повторяется для дочерних вершин. Вычислительная трудоемкость процедуры ADD составляет $O(\log N_o)$ где N_o — количество объектов ассоциированных с данной вершиной (мы полагаем, что объекты, ассоциированные с вершинами октального дерева, проиндексированы для быстрого поиска). В наихудшем случае, когда все объекты сцены оказываются ассоциированными с единственной ячейкой, $N_o = N$, где N — число объектов в сцене. Функция SHOULD_SPLIT принимает в качестве параметра вершину октального дерева и возвращает TRUE, если количество объектов, ассоциированных с ней, превышает допустимое, и ее разбиение приведет к тому, что как минимум один из дочерних узлов будет непустым. В противном случае возвращается значение FALSE. Процедура SPLIT выполняет разбиение октанта и переносит часть ассоциированных с ним объектов в дочерние вершины. При условии, что объекты, которые могут быть помещены в дочерние блоки, помечаются заранее, затрачиваемое время составит $O(M)$. Таким образом, трудоемкость вставки объекта в октальное дерево в наихудшем случае составляет $O(\log N + D + M)$.

```
procedure INSERT(T,O)
pointer node T
pointer object O

if(!IS_OVERLAPPING(T, O)
        return

if(!IS_LEAF(T))
```



```
{
if( COUNT_OVERLAPPED_CHILDREN(T,O)>1 )
        ADD(T,O)
else
        for_each(t in CHILDREN(T))
                INSERT(t,O)
return
}

ADD(T,O)
if(SHOULD_SPLIT(T))
        SPLIT(T)
return

procedure SPLIT(T)
pointer node T

collection B = SPLIT_BOUNDS(BOUNDS(T))
for_each(bounds b in B)
        ADD_CHILD(T, NODE(b))

for_each(object o in OBJECTS(T))
        INSERT(T,o)
```

Рис. 9в. Псевдокод операции вставки объекта в октальное дерево.

Ниже на рис. 9д приведен псевдокод процедуры REMOVE, удаляющей объект из октального дерева. В качестве параметра она принимает объект, который при помощи процедуры REMOVE_NODE удаляется из вершины, которая с ним ассоциирована. Для ее нахождения применяется функция NODE, описанная выше. Вычислительная сложность процедуры REMOVE_NODE составляет $O(\log N_o)$. Как уже было сказано ранее, в наихудшем случае, когда все объекты попадают в единственную ячейку пространства $N_o = N$. Функция SHOULD_MERGE возвращает TRUE, если для данной вершины не выполняются условия разбиения. Процедура MERGE принимает в качестве параметра вершину и удаляет все дочерние вершины. При этом все ассоциированные с ними объекты переносятся в родителя. Процедура рекурсивно выполняется до тех пор, пока в рассматриваемой вершине не выполняются условия разбиения. Вычислительная сложность этой процедуры не превышает $O(M + D)$. Таким образом, вычислительные затраты на удаление объекта составляют $O(\log N + D + M)$.

```
procedure REMOVE(O)
```

```
pointer object O

pointer node N = NODE(O)
REMOVE_NODE(N,O)

for_each(node p in PARENTS(N))
        if(SHOULD_MERGE(p))
                MERGE(p)

procedure MERGE(T)
pointer node T

for_each(node child in CHILDREN(T))
        for_each(object o in OBJECTS(child))
                ADD(T, o)
        REMOVE_CHILD(T, child)

if(SHOULD_MERGE(PARENT(T))
        return MERGE(PARENT(T))
```

Рис. 9д. Псевдокод операции удаления объекта из октального дерева.

5. Заключение

Таким образом, проведен сравнительный анализ современных методов поиска и индексации многомерных данных в приложениях моделирования больших динамических сцен. В рамках двух фундаментальных подходов, реализующих принципы объектной кластеризации и пространственной декомпозиции, выделены основные семейства методов и обсуждены их алгоритмические особенности. Анализ методов проведен в контексте комплексных требований, предъявляемых к приложениям обсуждаемого класса, и прежде всего, требований эффективности исполнения типовых пространственных запросов в больших сценах с недетерминированной динамикой и жестким характером пространственно-временной когерентности. Проведенный сравнительный анализ вычислительной сложности позволяет сделать вывод о перспективности методов, основанных на декомпозиции пространства, и, в частности, октальных деревьев со строгой многоуровневой локализацией объектов (MX-CIF) и с нижней границей кардинальности ячеек. Данный вывод основывается на следующих фактах:

— Методы, основанные на пространственной декомпозиции, и методы, использующие объектную кластеризацию, имеют одинаковую асимптотическую сложность операций поиска объектов в заданной области и поиска соседей объектов, составляющую $O(N)$. Вместе с тем,

сложность операций добавления объекта в сцену и его удаления для методов декомпозиции выражается оценкой $O(\log N + D + M)$, что более оптимистично, чем затраты $O(M \log N)$, необходимые для методов кластеризации.

— Методы кластеризации в большей степени ориентированы на особенности страничной работы с внешней памятью, что приводит к требованию высокой наполненности вершин дерева и его сбалансированности. Однако эти требования не являются абсолютно критичными для скорости разрешения пространственных запросов. Более гибкие условия наполненности ячеек в методах декомпозиции могут приводить к некоторой разбалансировке дерева, однако позволяют более точно локализовать объекты уже на верхних уровнях и избежать дополнительные проверки на нижних уровнях.

— Кластеризация объектов приводит к чрезмерно высоким затратам в динамических сценах, связанным с необходимостью перманентной перебалансировки дерева и сложным пространственным анализом. Для обновления структур индексации в методах декомпозиции достаточно идентифицировать ячейку сцены, в которой произошли события, и соответствующим образом модифицировать ее и, возможно, ее локальных соседей.

— Наконец, структуры пространственной декомпозиции инварианты по отношению к порядку событий в сцене и определяются лишь ее текущим представлением. В методах кластеризации необходим дополнительный анализ для обработки потока событий, чтобы обеспечить требуемую инвариантность структур индексации и избежать их деградации при масштабных изменениях в сцене.

Список литературы

[1]. V. Semenov, K. Kazakov, S. Morozov, O. Tarlapan, V. Zolotov and T. Dengenis, "4D modeling of large industrial projects using spatio-temporal decomposition," in *eWork and eBusiness in Architecture, Engineering and Construction*, London, UK, 2010.

[2]. С. Кузнецов и Б. Костенко, «История и актуальные проблемы темпоральных баз данных,» *Труды Института системного программирования,* т. 2, № 13, стр. 77-114, 2007.

[3]. V. Semenov, K. Kazakov, V. Zolotov, H. Jones and S. Jones, "Combined strategy for efficient collision detection in 4D planning applications," in *Computing in Civil and Building Engineering*, Nottingham, UK, 2010.

[4]. R. Bayer and E. M. McCreight, "Organization and maintanence of large ordered indexes," *Acta Informatica,* vol. 3, no. 1, pp. 173-189, 1972.

[5]. D. J. Abel, "A B+-tree structure for large quadtrees," *Computer Vision, Graphics, and Image Processing,* vol. 1, no. 27, pp. 19-31, July 1984.

[6]. [6] D. Comer, "The ubiquitos B-tree," *ACM Computing Surveys,* vol. 2, no. 11, pp. 121-137, June 1979.

[7]. H. Sagan, Space-Filling curves, New York: Springer-Verlag, 1994.
[8]. A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the ACM SIGMOD Conference*, Boston, USA, 1984.
[9]. T. Sellis, N. Roussopoulos and C. Faloutsos, "The R+-tree: a dynamic index for multi-dimentional objects," in *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, Brighton, UK, 1987.
[10]. N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Proceedings of the ACM SIGMOD Conference*, Atlantic City, USA, 1990.
[11]. C. Yu, B. C. Ooi, K.-L. Tan and H. V. Jagadish, "Indexing the distance: an efficient method to KNN processing," in *Proceedings of the 27st international Conference on Very Large Databases (VLDB)*, Roma, Italy, 2001.
[12]. A. Beygelzimer, S. Kakade and J. Langford, "Cover Trees for Nearest Neighbor," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.
[13]. W. A. Burkhard and R. Keller, "Some approaches to best-match file searching," *Communications of the ACM,* vol. 4, no. 16, pp. 230-236, April 1973.
[14]. J. L. Bentley, "Decomposable searching problems," *Information Processing Letters,* vol. 5, no. 8, pp. 244-251, June 1979.
[15]. J. L. Bentley and D. Wood, "An optimal worst-case algorithm for reporting intersections of rectangles," *IEEE Transactions on Computers,* vol. 7, no. 29, pp. 571-577, July 1980.
[16]. H. Edelsbrunner, "A new approach to rectangle intersections: part II," *International Journal of Computer Mathematics,* Vols. 3-4, no. 13, pp. 221-229, 1983.
[17]. E. M. McCreight, "Priority search trees," *SIAM Journal on Computing,* vol. 2, no. 14, pp. 257-276, May 1985.
[18]. H. Fuchs, G. Abram and E. Grant, "Near real-time shaded display of rigid objects," *Computer Graphics,* vol. 3, no. 17, pp. 65-72, 1983.
[19]. H. Fuchs, Z. Kedem and B. Naylor, "On visible surface generation by a priori tree structures," *Computer Graphics,* vol. 3, no. 14, pp. 124-133, 1980.
[20]. A. Henrich, H. W. Six and P. Widmayer, "The LSD-tree: spatial access to multidimentional point an non-point data," in *Proceedings of the 15th International Conference on Very Large Databases (VLDB)*, Amsterdam, Netherlands, 1989.
[21]. D. Lomet and B. Salzberg, "The hB-tree: a multi-attribute indexing method with good guaranteed performance," *ACM Transactions on Database Systems,* vol. 4, no. 15, pp. 625-658, December 1990.
[22]. J. Xu, B. Zheng, W. C. Lee and D. L. Lee, "The D-tree: an index structure for planar point queries in location-based wireless services," *IEEE Transactions on Knowledge and Data Engineering,* vol. 12, no. 16, pp. 1526-1542, December 2004.
[23]. J. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM,* vol. 9, no. 18, pp. 509-517, September 1975.
[24]. J. H. Friedman, J. L. Bentley and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software,* vol. 3, no. 3, pp. 209-226, September 1977.
[25]. K. Chakrabarti and S. Mehrotra, "The hybrid tree: an index structure for high dimentional feature spaces," in *Proceedings of the 15th IEEE International Conference on Data Engineering*, Sydney, Australia, 1999.
[26]. D. M. Mount and S. Arya, "ANN: a library for aproximate nearest neighbour searching," in *Proceedings of the 2nd Annual Center for Geometric Computing Workshop on Computational Geometry*, Durham, 1997.

[27]. D. A. White and R. Jain, "Similarity indexing with the SS-tree," in *Proceedings of the 12th IEEE International Conference on Data Engineering*, New Orleans, USA, 1996.
[28]. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman and A. Y. Wu, "An optimal algorithm for approximate nearest neighbour searching in fixed dimentions," *Journal of the ACM,* vol. 6, no. 45, pp. 891-923, November 1998.
[29]. J. O'Rourke, "Dynamically quantized spaces for focusing Hough transform," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, 1981.
[30]. K. R. S. Jr., "Dynamicaly quantized pyramids," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, 1981.
[31]. L. Becker, K. Hinrichs and U. Finke, "A new algorithm for computing joins with grid files," in *Proceedings of the 9th IEEE Conference on Data Engineering*, Vienna, Austria, 1993.
[32]. H. Samet, Foundations of Multidimentional and Metric Data Structures, San Francisco: Morgan Kaufmann, 2006.
[33]. P. Bogdanovich and H. Samet, "The ATree: a data structure to support a very large scientific databases," in *Springer-Verlag Lecture Notes in Computer Science*, vol. 1737, P. Agouris and A. Stefanidis, Eds., Springer-Verlag, 1990, pp. 235-248.
[34]. K. Knowlton, "Progressive transmission of gray-scale and binary pictures by simple efficient and lossless encoding schemes," *Proceedings of IEEE,* vol. 7, no. 68, pp. 885-896, 1980.
[35]. Y. Cohen, M. Landy and M. Pavel, "Hierarchical coding of binary images," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 3, no. 7, pp. 284-298, 1985.
[36]. G. Nagy and S. Wagle, "Hierarchical representation of optically scanned documents," in *Proceedings of 7th International Conference on Pattern Recognition*, 1984.
[37]. A. Dengel, "Object-oriented representation of image space by puzzle-trees," *SPIE Visual Communications and Image Processing,* pp. 20-30, 1991.
[38]. B. Shneiderman, "Tree visualization with tree maps: 2-d space-filling approach," *ACM Transactions on Graphics,* vol. 1, no. 11, pp. 92-99, 1992.
[39]. I. Kamel and C. Faloutsos, "Hilbert R-tree: an improved R-tree using fractals," *Proceedings of 20th International Conference on Very Large Data Bases,* vol. 3, no. 3, 1994.
[40]. I. Kamel and C. Faloutsos, "On packing R-trees," in *Proceedings of 2nd International Conference on Information and Knowledge Management*, 1993.
[41]. S. T. Leutenegger, M. A. Lopez and J. Edgington, "STR: a simple and efficient algorithm for R-tree packing," in *Proceedings of the 13th IEEE International conference on Data Engineering*, 1997.
[42]. J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE Transactions on Visualization and Computer Graphics,* vol. 1, no. 4, pp. 21-36, Januray 1998.
[43]. O. Gunther and J. Bilmes, "Tree-based access methods for spatial databases: implementation and performance evaluation," *IEEE Transactions on Knowledge and Data Engineering,* vol. 3, no. 3, pp. 342-356, 1991.
[44]. S. Gottschalk, M. C. Lin and D. Manocha, "OBB Tree: a hierarchical structure for rapid interference detection," in *Proceedings of the SIGGRAPH'96 Conference*, New Orleans, USA, 1996.
[45]. S. Gottschalk, Collision queries using oriented bounding boxes, Chapel Hill: The University of North Carolina, 2000.

[46]. Y. Theodoris and T. Sellis, "Optimization issues in R-tree construction," in *IGIS'94: Geographic Information Systems, International Workshop on Advanced Research in Geographic Information Systems*, 1994.
[47]. Y. Garcia, M. Lopez and S. Leutenneger, "A greedy algorithm for bulk loading R-trees," in *Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems*, 1997.
[48]. B. Becker, P. Franciosa, S. Gschwind, T. Ohler, G. Thiemt and P. Widmayer, "Enclosing many boxes by an optimal pair of boxes," in *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, 1992.
[49]. Y. Garcia, M. Lopez and S. Leutenneger, "An optimal node splitting for R-trees," in *Proceedings of the 24th International Conference on Very Large Data Bases*, 1998.
[50]. C. Ang and T. Tan, "New linear node splitting algorithm for R-trees," in *Advances in Spatial Databases – 5th International Symposium, SSD'97*, 1997.
[51]. E. G. Hoel and H. Samet, "Benchmarking spatial join operations with spatial output," in *Proceedings of the 21th International Conference on Very Large Databases (VLDB)*, Zurich, Switzerland, 1995.
[52]. M. Cai and P. Revesz, "Parametric rectangles: an index structure for moving objects," in *Proceedings of the 10th COMAD International conference on management of data*, 2000.
[53]. S. Saltenis, C. S. Jensen, S. T. Leuteenegger and M. A. Lopez, "Indexing the positions of continuously moving objects," in *Proceedings of the ACM SIGMOD Conference*, 2000.
[54]. Y. Tao, D. Papadias and J. Sun, "The TRP*-tree: an optimized spatio-temporal access for predictive queries," in *Proceedings of the 29th international conference on Very Large Data Bases (VLDB)*, 2003.
[55]. S. B. M. Bell, B. M. Diaz, F. Holroyd and M. J. Jackson, "Spatially referenced method of processing raster and vector data," *Image and Vision Computing,* vol. 4, no. 1, pp. 211-220, 1983.
[56]. L. Gibson and D. Lucas, "Vectorization of raster images using hierarchical methods," *Computer Graphics and Image Processing,* vol. 1, no. 20, pp. 82-29, 1982.
[57]. Г. М. Адельсон-Вельский и Е. М. Ландис, «Один алгоритм организации информации,» *Доклады Академии Наук СССР,* № 146, стр. 263-266, 1962.
[58]. T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Red–Black Trees," in *Introduction to Algorithms*, MIT Press and McGraw-Hill, 2001, p. 273–301.
[59]. A. Amir, A. Efrat, P. Indyk and H. Samet, "Efficient algorithms and regular data structures for dilation, location and proximity problems," *Algorithmica,* vol. 2, no. 30, pp. 164-187, 2001.
[60]. G. Schrack, "Finding neighbors of equal size in linear quadtrees and octrees in constant time," *CVGIP: Image Understanding,* vol. 3, no. 55, pp. 221-230, 1992.
[61]. G. Kedem, "The quad-CIF tree: a data structure for hierarchical on-line algorithms," in *Proceedings of the 19th Design Automation Conference*, 1992.
[62]. A. U. Frank, "Problems of realizing LIS: storage methods for space related data: the fieldtree," ETH, 1983.
[63]. A. U. Frank and R.Barrera, "The Fieldtree: a data structure for geographic information systems," in *Springer-Verlag Lecture Notes in Computer Science*, vol. 409, A. Buchmann, O. Gunter, T. Smith and Y. Wang, Eds., Springer-Verlag, 1989, pp. 29-44.
[64]. T. Ulrich, "Loose octrees," in *Game programming gems*, Rockland, Charles river media, 2000, pp. 444-453.

[65]. H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer Graphics and Image processing,* vol. 1, no. 18, pp. 37-57, 1982.

Advanced indexing methods for large spatial data in complex dynamic scenes

V.A. Zolotov, V.A. Semenov(ISP RAS, Moscow, Russia)

Annotation. This paper is dedicated to review of recent methods for indexing of multidimensional data and their use for modeling of large-scale dynamic scenes. The problem arises in many application domains such as computer graphics systems, virtual and augmented reality systems, CAD systems, robotics, geographical information systems, project management systems, etc.

Two fundamental approaches to the indexing of multidimensional data, namely object aggregation and spatial decomposition, have been outlined. In the context of the former approach balanced search trees, also referred as object pyramids, have been discussed. In particular, generalizations of B-trees such as R-trees, R*-trees, R+-trees have been discussed in conformity to indexing of large data located in external memory and accessible by pages. The conducted analysis shows that object aggregation methods are well suited for static scenes but very limited for dynamic environments.

The latter approach assumes the recursive decomposition of scene volume in accordance with the cutting rules. Space decomposition methods are octrees, A-trees, bintrees, K-D-trees, X-Y-trees, treemaps and puzzle-trees. They support more reasonable compromise between performance of spatial queries and expenses required to update indexing structures and to keep their in concordant state under permanent changes in the scenes. Compared to object pyramids, these methods look more promising for dramatically changed environments. It is concluded that regular dynamic octrees are most effective for the considered applications of modeling of large-scale dynamic scenes.

Keywords: spatial indexing, multidimensional data, dynamic scene modeling

References

[1]. Semenov V.A., Kazakov K.A., Morozov S.V., Tarlapan O.A., Zolotov V.A., Dengenis T., "4D modeling of large industrial projects using spatio-temporal decomposition" in eWork and eBusiness in Architecture, Engineering and Construction, London, UK, pp. 89-95, 2010.

[2]. Kuznecov S.D., Kostenko B.B. "Istoriya i aktual'nye problemy temporal'nykh baz dannykh" [History and actual state of temporal databases]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 2, № 13, pp. 77-114, 2007. (in Russian)

[3]. Semenov V.A., Kazakov K.A., Zolotov V.A., Jones H., Jones S. "Combined strategy for efficient collision detection in 4D planning applications" in Computing in Civil and Building Engineering, Nottingham, UK, pp. 31-39, 2010.

[4]. Bayer R., McCreight E. M., "Organization and maintanence of large ordered indexes" Acta Informatica, vol. 3, no. 1, pp. 173-189, 1972.

[5]. Abel D. J. "A B+-tree structure for large quadtrees" Computer Vision, Graphics, and Image Processing, vol. 1, no. 27, pp. 19-31, July 1984.

[6]. Comer D. "The ubiquitos B-tree" ACM Computing Surveys, vol. 2, no. 11, pp. 121-137, June 1979.

[7]. Sagan H. "Space-Filling curves", New York: Springer-Verlag, 1994.

[8]. Guttman A. "R-trees: a dynamic index structure for spatial searching" in Proceedings of the ACM SIGMOD Conference, Boston, USA, 1984.

[9]. Sellis T., Roussopoulos N., Faloutsos C. "The R+-tree: a dynamic index for multi-dimentional objects" in Proceedings of the 13th International Conference on Very Large Databases (VLDB), Brighton, UK, 1987.

[10]. Beckmann N., Kriegel H. P., Schneider R., Seeger B. "The R*-tree: an efficient and robust access method for points and rectangles" in Proceedings of the ACM SIGMOD Conference, Atlantic City, USA, 1990.

[11]. Yu C., Ooi B. C., Tan K.-L., Jagadish H. V. "Indexing the distance: an efficient method to KNN processing" in Proceedings of the 27st international Conference on Very Large Databases (VLDB), Roma, Italy, 2001.

[12]. Beygelzimer A., Kakade S., Langford J., "Cover Trees for Nearest Neighbor" in Proceedings of the International Conference on Machine Learning (ICML), 2006.

[13]. Burkhard W. A., Keller R. "Some approaches to best-match file searching" Communications of the ACM, vol. 4, no. 16, pp. 230-236, April 1973.

[14]. Bentley J. L. "Decomposable searching problems" Information Processing Letters, vol. 5, no. 8, pp. 244-251, June 1979.

[15]. Bentley J. L., Wood D. "An optimal worst-case algorithm for reporting intersections of rectangles" IEEE Transactions on Computers, vol. 7, no. 29, pp. 571-577, July 1980.

[16]. Edelsbrunner H. "A new approach to rectangle intersections: part II" International Journal of Computer Mathematics, Vols. 3-4, no. 13, pp. 221-229, 1983.

[17]. McCreight E. M. "Priority search trees" SIAM Journal on Computing, vol. 2, no. 14, pp. 257-276, May 1985.

[18]. Fuchs H., Abram G., Grant E. "Near real-time shaded display of rigid objects" Computer Graphics, vol. 3, no. 17, pp. 65-72, 1983.

[19]. Fuchs H., Kedem Z., Naylor B. "On visible surface generation by a priori tree structures" Computer Graphics, vol. 3, no. 14, pp. 124-133, 1980.

[20]. Henrich A., Six H. W., Widmayer P. "The LSD-tree: spatial access to multidimentional point an non-point data" in Proceedings of the 15th International Conference on Very Large Databases (VLDB), Amsterdam, Netherlands, 1989.

[21]. Lomet D., Salzberg B. "The hB-tree: a multi-attribute indexing method with good guaranteed performance" ACM Transactions on Database Systems, vol. 4, no. 15, pp. 625-658, December 1990.

[22]. Xu J., Zheng B., Lee W. C., Lee D. L. "The D-tree: an index structure for planar point queries in location-based wireless services" IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 16, pp. 1526-1542, December 2004.

[23]. Bentley J.L. "Multidimensional binary search trees used for associative searching" Communications of the ACM, vol. 9, no. 18, pp. 509-517, September 1975.

[24]. Friedman J. H., Bentley J. L., Finkel R. A. "An algorithm for finding best matches in logarithmic expected time" ACM Transactions on Mathematical Software, vol. 3, no. 3, pp. 209-226, September 1977.

[25]. Chakrabarti K., Mehrotra S. "The hybrid tree: an index structure for high dimentional feature spaces" in Proceedings of the 15th IEEE International Conference on Data Engineering, Sydney, Australia, 1999.

[26]. Mount D. M., Arya S. "ANN: a library for aproximate nearest neighbour searching" in Proceedings of the 2nd Annual Center for Geometric Computing Workshop on Computational Geometry, Durham, 1997.

[27]. White D. A., Jain R. "Similarity indexing with the SS-tree" in Proceedings of the 12th IEEE International Conference on Data Engineering, New Orleans, USA, 1996.

[28]. Arya S., Mount D. M., Netanyahu N. S., Silverman R., Wu A. Y. "An optimal algorithm for approximate nearest neighbour searching in fixed dimentions" Journal of the ACM, vol. 6, no. 45, pp. 891-923, November 1998.
[29]. O'Rourke J. "Dynamically quantized spaces for focusing Hough transform" in Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, Canada, 1981.
[30]. Kenneth R. Sloan Jr., "Dynamicaly quantized pyramids" in Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, Canada, 1981.
[31]. Becker L., Hinrichs K., Finke U. "A new algorithm for computing joins with grid files" in Proceedings of the 9th IEEE Conference on Data Engineering, Vienna, Austria, 1993.
[32]. Samet H. "Foundations of Multidimentional and Metric Data Structures", San Francisco: Morgan Kaufmann, 2006.
[33]. Bogdanovich P., Samet H. "The ATree: a data structure to support a very large scientific databases" in Springer-Verlag Lecture Notes in Computer Science, vol. 1737, P. Agouris and A. Stefanidis, Eds., Springer-Verlag, 1990, pp. 235-248.
[34]. Knowlton K. "Progressive transmission of gray-scale and binary pictures by simple efficient and lossless encoding schemes" Proceedings of IEEE, vol. 7, no. 68, pp. 885-896, 1980.
[35]. Cohen Y., Landy M., Pavel M. "Hierarchical coding of binary images" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 3, no. 7, pp. 284-298, 1985.
[36]. Nagy G., Wagle S. "Hierarchical representation of optically scanned documents" in Proceedings of 7th International Conference on Pattern Recognition, 1984.
[37]. Dengel A. "Object-oriented representation of image space by puzzle-trees" SPIE Visual Communications and Image Processing, pp. 20-30, 1991.
[38]. Shneiderman B. "Tree visualization with tree maps: 2-d space-filling approach" ACM Transactions on Graphics, vol. 1, no. 11, pp. 92-99, 1992.
[39]. Kamel I., Faloutsos C. "Hilbert R-tree: an improved R-tree using fractals" Proceedings of 20th International Conference on Very Large Data Bases, vol. 3, no. 3, 1994.
[40]. Kamel I., Faloutsos C. "On packing R-trees" in Proceedings of 2nd International Conference on Information and Knowledge Management, 1993.
[41]. Leutenegger S. T., Lopez M. A., Edgington J. "STR: a simple and efficient algorithm for R-tree packing" in Proceedings of the 13th IEEE International conference on Data Engineering, 1997.
[42]. Klosowski J. T., Held M., Mitchell J. S. B., Sowizral H., Zikan K. "Efficient collision detection using bounding volume hierarchies of k-DOPs" IEEE Transactions on Visualization and Computer Graphics, vol. 1, no. 4, pp. 21-36, Januray 1998.
[43]. Gunther O., Bilmes J. "Tree-based access methods for spatial databases: implementation and performance evaluation" IEEE Transactions on Knowledge and Data Engineering, vol. 3, no. 3, pp. 342-356, 1991.
[44]. Gottschalk S., Lin M. C., Manocha D. "OBB Tree: a hierarchical structure for rapid interference detection" in Proceedings of the SIGGRAPH'96 Conference, New Orleans, USA, 1996.
[45]. Gottschalk S. "Collision queries using oriented bounding boxes", Chapel Hill: The University of North Carolina, 2000.
[46]. Theodoris Y., Sellis T. "Optimization issues in R-tree construction" in IGIS'94: Geographic Information Systems, International Workshop on Advanced Research in Geographic Information Systems, 1994.

[47]. Garcia Y., Lopez M., Leutenneger S. "A greedy algorithm for bulk loading R-trees" in Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems, 1997.
[48]. Becker B., Franciosa P., Gschwind S., Ohler T., Thiemt G., Widmayer P., "Enclosing many boxes by an optimal pair of boxes" in Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science, 1992.
[49]. Garcia Y., Lopez M., Leutenneger S. "An optimal node splitting for R-trees" in Proceedings of the 24th International Conference on Very Large Data Bases, 1998.
[50]. Ang C., Tan T. "New linear node splitting algorithm for R-trees" in Advances in Spatial Databases – 5th International Symposium, SSD'97, 1997.
[51]. Hoel E. G., Samet H. "Benchmarking spatial join operations with spatial output" in Proceedings of the 21th International Conference on Very Large Databases (VLDB), Zurich, Switzerland, 1995.
[52]. Cai M., Revesz P. "Parametric rectangles: an index structure for moving objects" in Proceedings of the 10th COMAD International conference on management of data, 2000.
[53]. Saltenis S., Jensen C. S., Leuteenegger S. T., Lopez M. A., "Indexing the positions of continuously moving objects" in Proceedings of the ACM SIGMOD Conference, 2000.
[54]. Tao Y., Papadias D., Sun J. "The TRP*-tree: an optimized spatio-temporal access for predictive queries" in Proceedings of the 29th international conference on Very Large Data Bases (VLDB), 2003.
[55]. Bell S. B. M., Diaz B. M., Holroyd F., Jackson M. J. "Spatially referenced method of processing raster and vector data" Image and Vision Computing, vol. 4, no. 1, pp. 211-220, 1983.
[56]. Gibson L., Lucas D. "Vectorization of raster images using hierarchical methods" Computer Graphics and Image Processing, vol. 1, no. 20, pp. 82-29, 1982.
[57]. Adel'son-Vel'sky G.M, Landis E.M. "Odin algoritm organizatsii informatsii"[An algorithm for information organization] Proceedings of RAS USSR, № 146, pp. 263-266, 1962.
[58]. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. "Red–Black Trees" in Introduction to Algorithms, MIT Press and McGraw-Hill, 2001, p. 273–301.
[59]. Amir A., Efrat A., Indyk P., Samet H. "Efficient algorithms and regular data structures for dilation, location and proximity problems" Algorithmica, vol. 2, no. 30, pp. 164-187, 2001.
[60]. Schrack G. "Finding neighbors of equal size in linear quadtrees and octrees in constant time" CVGIP: Image Understanding, vol. 3, no. 55, pp. 221-230, 1992.
[61]. Kedem G. "The quad-CIF tree: a data structure for hierarchical on-line algorithms" in Proceedings of the 19th Design Automation Conference, 1992.
[62]. Frank A. U., "Problems of realizing LIS: storage methods for space related data: the fieldtree" ETH, 1983.
[63]. Frank A. U., Barrera R. "The Fieldtree: a data structure for geographic information systems" in Springer-Verlag Lecture Notes in Computer Science, vol. 409, [ed] Buchmann A., Gunter O., Smith T., Wang Y., Springer-Verlag, 1989, pp. 29-44.
[64]. Ulrich T., "Loose octrees" in Game programming gems, Rockland, Charles river media, 2000, pp. 444-453.
[65]. Samet H. "Neighbor finding techniques for images represented by quadtrees" Computer Graphics and Image processing, vol. 1, no. 18, pp. 37-57, 1982.