Оптимизация трассировки лучей в октантных деревьях

В.А. Бобков, С.В. Мельман, Ю.И. Роньшин
Институт автоматики и процессов управления ДВО РАН
Владивосток, Россия
bobkov@iacp.dvo.ru

Аннотация

Описан алгоритм дискретной трассировки лучей в октантных деревьях с использованием целочисленной арифметики и предварительно вычисленных таблиц геометрических решений. Алгоритм основывается на декомпозиции решения 3D задачи поиска пересекаемых подбоксов в родительском боксе на 2D задачи в координатных плоскостях. Предложена оптимизация алгоритма за счет реализации когерентности траекторий лучей в октантном дереве сцены. Вводится понятие опорных лучей, октантные траектории которых запоминаются, и понятие общего пути в октантном пространстве для двух близких лучей. Представлен алгоритм оптимизированной трассировки промежуточных лучей, основанный на использовании общих путей. Рассмотрены два варианта реализации когерентности октантных траекторий. Получены экспериментальные оценки эффективности рассмотренной оптимизации на разных сценах. Также получены сравнительные оценки эффективности оптимизированного алгоритма дискретной трассировки и алгоритма прямого рендеринга на октантных сценах. Работа системы иллюстрируется примерами применения ее в океанологии и в геоинформатике.

***Ключевые слова**: октантное дерево, дискретная трассировка лучей, когерентность октантных траекторий*

1. ВВЕДЕНИЕ

Вопрос повышения эффективности рендеринга воксельных моделей продолжает сегодня оставаться актуальным, несмотря на достигнутый прогресс в последнее время в этой области. Поиски путей ускорения обработки графических данных ведутся как в направлении совершенствования пространственных структур данных и оптимизации алгоритмических решений, так и в направлении использования существующих аппаратных возможностей и многопроцессорных вычислительных архитектур. Много внимания разными исследователями было уделено вопросам эффективной трассировки октантных деревьев, которые широко применяются в компьютерной графике. В известных алгоритмах этого типа трассировка выполняется сверху вниз (от корневого узла дерева) [1-5] с реализацией рекурсивности, или снизу вверх [6,7] с поиском решения в окрестности последнего найденного граничного воксела. При этом наибольший эффект достигается при реализации дискретной трассировки [8,10]. В алгоритме отслеживания лучей в октантных деревьях, реализованном авторами в рамках системы воксельной графики, алгоритмическая оптимизация обеспечивается, как за счет реализации дискретной трассировки, целочисленной арифметики и использования табличных геометрических решений, так и за счет реализации когерентности октантных траекторий пространственно близких лучей. Оптимизация последнего типа существенно отличается от известного авторам аналога [11], где предлагается использование когерентности траекторий в BSP структуре. В работе получены оценки эффективности предлагаемой оптимизации алгоритма. Также приведены сравнительные оценки алгоритмов трассировки и прямого рендеринга октантных сцен.

2. СИСТЕМА ВОКСЕЛЬНОЙ ГРАФИКИ

Рассматриваемые в статье алгоритмы работают в составе разработанной авторами системы воксельной графики, основными характеристиками которой являются:

- визуализация изоповерхностей и поверхностей объемов, представленных октантными деревьями;
- поддержка операций конструктивной геометрии;
- применение ray casting-а и алгоритма послойной визуализации (прямой рендеринг);
- реализация мультиразрешения;
- применение surfel-ой модели с использованием аппаратной поддержки;
- реализация диффузной освещенности и возможность view dependent rendering;
- параллельные вычисления на МВС-1000;
- применение для визуализации скалярных полей, данных медицинских измерений, пространственной модели карты города (см. рис. 7-13).

3. ОКТАНТНАЯ СТРУКТУРА ДАННЫХ

Поддерживаются две формы структуры. Основная, назовем ее структура «объем», содержит: промежуточные узлы, внутренние вокселы, внутренние термы, граничные вокселы, граничные термы. Она обеспечивает выполнение булевых операций и визуализацию. Структура «оболочка» - без внутренних термов и внутренних вокселов – только для визуализации. Термы обозначают октанты, полностью принадлежащие объекту. У граничных термов одна грань или несколько принадлежат граням бокса сцены. Структура «оболочка» требует существенно меньше памяти, чем основная. Во внешней памяти данные хранятся в компрессированном виде. Например, для октантного дерева глубиной 8 сцены «изоповерхность температуры» основная структура в оперативной памяти занимает 22 Мб, в сжатом виде во внешней памяти 1.42 Мб, структура «оболочка», соответственно, 12 Мб и 1.16 Мб. А для сцены «город» с глубиной дерева 9 структура «оболочка» в оперативной памяти занимает 22 Мб и в сжатом виде – 3.59 Мб. Значения нормалей для граничных вокселов при этом заданы плавающей арифметикой. Возможно хранение нормалей в

целочисленном виде, тогда объем требуемой памяти уменьшается примерно на 30%.

Поддерживаются два режима визуализации: с предварительно вычисленной диффузной освещенностью (используется, когда положение источников освещения не меняется при просмотре с разных точек наблюдения) и с вычислением закраски в процессе рендеринга (по хранящимся в узлах-вокселах нормалям к поверхности объекта).

4. ДИСКРЕТНАЯ ТРАССИРОВКА ЛУЧЕЙ

В реализованном в данной работе алгоритме трассировки лучей в октантных деревьях существенно используются уже известные решения, к которым можно отнести: рекурсивную обработку дерева top-down с горизонтальным движением по дереву (анализ пересекаемых подбоксов в рамках родительского бокса) и вертикальным (спуск от текущего бокса к пересекаемым подбоксам); выбор «главных направлений» при анализе пересечений подбоксов лучом и рекурсивное упрощенное вычисление «средних точек» по точкам входа, выхода луча в боксе; реализация пространственной когерентности с использованием предварительно вычисленных таблиц геометрических решений; переход к целочисленной арифметике для ускорения вычислений. Как и в некоторых аналогах, в данном случае рассматривается режим ray casting. Вместе с тем описываемый ниже алгоритм поиска упорядоченного списка пересекаемых подбоксов, построенный на декомпозиции пространственной задачи в «плоские», отличается от известных аналогов. С другой стороны, в работе делается акцент на оптимизации трассировки октантного дерева в целом за счет реализации когерентности октантных траекторий близких лучей.
Оценивается также дополнительный оптимизационный эффект при рендеринге сцены за счет реализации когерентности цветов соседних пикселов экрана.

4.1 Переход к целочисленной арифметике

Для реализации алгоритма была выбрана целочисленная арифметика, где любое число представлено четырехбайтовым целым числом со знаком (32 бита). Любой бокс сцены операцией сдвига и растяжения приводится к кубу с гранью 2^14 = 16384. Также к новой системе координат приводятся и все лучи трассировки. Также как и, например, в [10] используется понятие «главного направления». Это координатная ось (X, Y или Z), проекция луча на которую имеет наибольшее значение по модулю. Главное направление

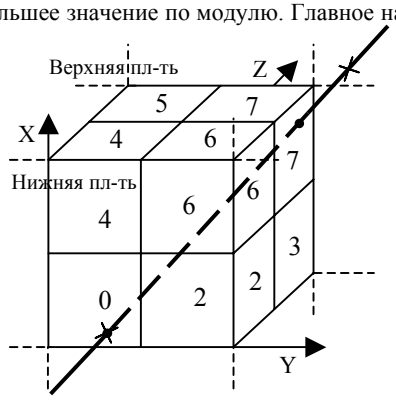


Рис.1 Нумерация подбоксов в боксе.

позволяет выбрать нижнюю и верхнюю плоскости (на рис.1 главное направление – ось Z), проходящие через грани бокса сцены и перпендикулярные главному направлению. В каждой из этих плоскостей определены плоские целочисленные координаты. Отметим, что в рассматриваемом алгоритме не имеет значения направление луча, и, соответственно, не имеет значения, какая из точек является точкой «входа», а какая точкой «выхода» луча.

4.2 Пересечение подбоксов в боксе. «Байт состояния».

Если не принимать во внимание порядок следования подбоксов в боксе, то сам набор пересеченных подбоксов можно представить как «байт состояния». В этом байте состояния каждому пересеченному подбоксу соответствует бит, порядковый номер которого в байте соответствует номеру подбокса. Если, например, пересеченные подбоксы в боксе имеют порядковые номера 1, 0, 4, 6, тогда байт состояния будет 0b01010011 = 83.

4.3 Вычисление «байта состояния».

Предположим, что главное направление - ось Z и луч имеет положительные проекции на все оси координат (другие случаи могут быть сведены к этому). Рассмотрим три проекции луча и проекции подбоксов на координатные плоскости XY, XZ, YZ. Проекция куба на координатную плоскость даст квадрат, луча – прямую, а так как луч задан двумя точками, лежащими на плоскостях граней бокса сцены, то в проекции получаем отрезок. Таким образом, имеем вариант пересечения квадрата лучом (рис.2). Каждая из

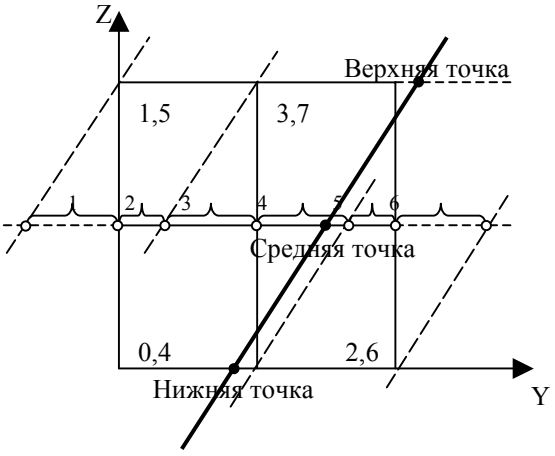


Рис.2 Проекция бокса на пл-ть YZ

четвертей квадрата является проекцией двух подбоксов. Будем их называть подбоксы-кандидаты. Если в каждой из проекций найти байт состояния подбоксов-кандидатов, то логическое «и» для всех этих байтов, даст искомый байт состояния бокса.

Рассмотрим, как осуществляется поиск байта состояния подбоксов-кандидатов (рис. 2). Точка пересечения средней линии квадрата и луча - «средняя точка», - может принадлежать одному из 6 отрезков. В зависимости от того, какому из отрезков принадлежит эта точка, можно однозначно определить байт состояния подбоксов-кандидатов для этой проекции. В примере, приведенном на

рисунке, это подбоксы с номерами: 0, 2, 4, 5, 6, 7. Поиск точки пересечения средней линии и луча в проекциях XZ и YZ сводится к нахождению среднего арифметического между «нижней» и «верхней» точками (рис.2). Для плоскости XY эта задача решается непосредственным вычислением точки пересечения двух прямых. С учетом того, что средняя линия квадрата параллельна одной из осей координат и ряда других упрощающих факторов, решение сводится к минимуму арифметических операций (всего 2 «*», 2 «+» и 1 «<<1»). Получив «среднюю точку» можно по предварительно составленной таблице получить байт состояния подбоксов-кандидатов. Для определения конечного байта состояния бокса в реализованном алгоритме требуется:

5 «+», 2 «*», 2 «&» и не больше 21 сравнения (в среднем около 10).

4.4 Нумерация подбоксов. Табличный выбор порядка следования подбоксов

Определим «базовую» нумерацию подбоксов, которая используется для организации структуры дерева и хранения в памяти (рис.1). Нумерация подбоксов такова, что продвижение подбокса вдоль оси устанавливает соответствующий бит в 1. Вдоль оси Z устанавливается первый бит, Вдоль оси Y – второй, вдоль оси X – третий

(0b00000xyz).

Выше был рассмотрен алгоритм поиска «байта состояния» бокса, если луч имеет только положительные проекции на оси координат. Теперь рассмотрим порядок следования пересеченных подбоксов в боксе. При таких проекциях луча, последовательность следования подбоксов строго определена. Например:

«байт состояния» = 0b01010011, тогда их последовательность 0, 1, 4, 6;

«байт состояния» = 0b00110001, то их последовательность 0, 4, 5;

Если все проекции луча на оси координат отрицательные, то эта последовательность меняется на строго обратную:

«байт состояния» = 0b01010011, то, их последовательность 6, 4, 1, 0;

«байт состояния» = 0b00110001, то, их последовательность 5, 4, 0.

Для всех возможных состояний проекций луча (их 8, 2 состояния для каждой проекции и 3 проекции) можно построить таблицу последовательностей подбоксов. Она будет содержать 256 строк и 4 столбца для каждых из 8 состояний проекций. Использование таблицы существенно экономит вычислительные затраты.

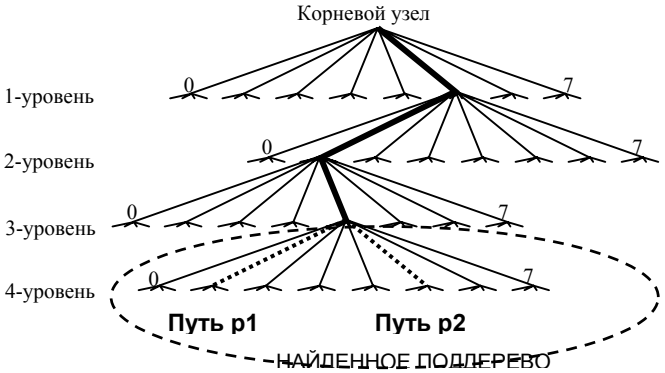
5. КОГЕРЕНТНОСТЬ ТРАЕКТОРИЙ В ОКТАНТНЫХ ДЕРЕВЬЯХ.

В октантном пространстве каждому лучу трассировки соответствует своя ветвь октантного дерева, которая начинается корневым узлом и завершается своим граничным вокселом (первое пересечение луча с объектом сцены). Далее по тексту будем называть ее октантной траекторией луча. Количество промежуточных узлов на траектории равно глубине дерева. Рассмотрим два соседних или близких пиксела на экране. Лучи, проходящие через эти пикселы из точки наблюдения, имеют небольшое пространственное расхождение. Для них велика вероятность того, что граничные вокселы, найденные в процессе трассировки, окажутся пространственно близкими друг к другу или даже совпадающими. В октантном пространстве (дереве) эта близость проявляется в наличии общего начального участка траектории (общий путь). При этом бокс узла разветвления близких траекторий является наименьшим из боксов, содержащим оба граничных воксела. Таким образом, чем ближе друг к другу расположены граничные вокселы двух близких лучей, тем больше у этих лучей будет общий путь в октантном пространстве. И, соответственно, чем больше расстояние между двумя лучами, тем меньше у них будет общий путь. Логично предположить, что и траектория любого промежуточного луча будет включать в себя этот общий путь. Это соответствует предположению, что бокс, содержащий пересекаемые граничные вокселы двух близких лучей, будет содержать и граничные вокселы, пересекаемые промежуточными лучами. Простой геометрический анализ показывает, что, как правило, это справедливо (но не всегда). В принципе, этого достаточно, чтобы построить алгоритм, реализующий когерентность октантных траекторий близких лучей.

Будем называть «опорными» лучами два близких луча, для которых выполняется полная трассировка. Идея алгоритма заключается в том, чтобы, запоминая вычисленные траектории опорных лучей, выполнять только частичную трассировку для промежуточных лучей. Для этого необходимо вычислить общий путь и, спустившись по нему, начинать трассировку с узла разветвления. Чем больше суммарный общий путь по всему множеству лучей, тем большая часть трассировки исключается из полного объема трассировки. Поэтому эффективность такого алгоритма определяется тем, насколько он максимизирует общий путь по всем лучам экрана. Ниже рассмотрены два варианта реализации когерентности указанного типа.

5.1 Использование когерентности на линейных участках.

Рассмотрим вариант построчного обхода пикселов экрана при трассировке лучей. Как было сказано выше, для использования когерентности требуется два опорных луча, по которым строятся два опорных пути, назовем их p1 и p2.



Путь p1 – (5,1,4,1…)

Путь p2 – (5,1,4,5…) Общий путь cp12 – (5,1,4)

Рис.3 Пути к граничным вокселам по опорным лучам. Общий путь.

Общий путь для них обозначим cp12 (рис.3). Предположительно существует оптимальное расстояние (в пикселах) между опорными лучами, поскольку и увеличение до максимума и уменьшение до минимума этого расстояния очевидным образом приводит к уменьшению общего пути в целом. С учетом этого соображения и экспериментальной проверки было выбрано расстояние в 4 пиксела.

Используя метод деления отрезка пополам, вычисляем граничный воксел для среднего пиксела (рис.4), используя общий путь cp12. Полученный путь назовем p3. Для двух оставшихся пикселов используются общие пути cp13 и cp23. Таким образом, для среднего пиксела расстояние между опорными лучами 4 пиксела, а для каждого из остальных двух - 2 пиксела.

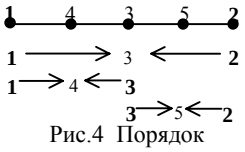


Рис.4 Порядок построения промежуточных лучей

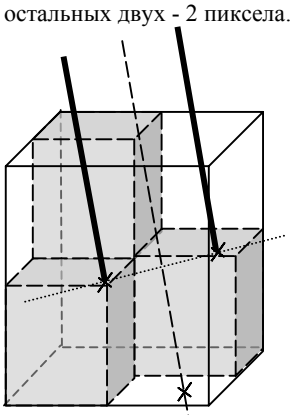


Рис.5 Пример отсутствия решения в найденном поддереве.

Заметим, что алгоритм не даст решения, если поддерево к которому приводит общий путь, не содержит искомого граничного воксела для промежуточного луча. Такие ситуации маловероятны, но могут возникать. Например, на рис.5 показано, что два «базовых» луча приводят к выделенному поддереву, но для промежуточного луча в выделенном поддереве решения нет, т.к. оно находится в другом поддереве. В этих случаях выполняется полный проход октодерева от корня. Формально алгоритм может быть описан следующим образом:

ЦИКЛ по пикселам одной строки

разбить строку на интервалы

на границах интервалов по «опорным» лучам получить «пути» для граничных вокселов

по общему пути для каждого интервала получить поддерево

для пикселов внутри интервалов найти граничные вокселы используя найденные поддеревья,

если в поддереве граничный воксел не найден, повторить поиск, используя полное октодерево

для промежуточных пикселов запоминать пути

КОНЕЦ ЦИКЛА

5.2 Использование когерентности на площадных участках

Поскольку линейная развертка не учитывает близости лучей в 2D, был реализован вариант «площадной» трассировки пикселов с целью получить дополнительный эффект оптимизации. Так же как и в случае построчной развертки берется расстояние между двумя строками в 4 пиксела. Назовем крайние строки «базовыми строками». При построении базовых строк применяется описанный выше алгоритм. При сканировании промежуточных пикселов так же запоминаются пути. Построение промежуточных строк происходит, как и в линейном случае. Между двумя строками выбираем два соседних пиксела, принадлежащие одному столбцу и делаем их опорными. Используя уже построенные для них пути, находим граничный воксел для пиксела расположенного между опорными. Сохраняем путь к этому вокселу, для того чтобы использовать его при нахождении граничных вокселов для двух оставшихся пикселов в столбце между опорными строками. Описанный порядок построения лучей и использование когерентности на площадных участках показан на рис. 6.

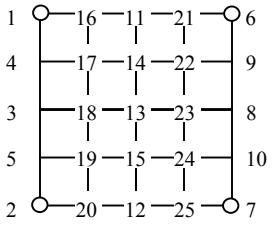


Рис. 6 Порядок построения лучей на площадном участке.

1 – (***Б***); **2** – (***Б***); **3** – (1, 2) r=4;
4 – (1, 3) r=2; **5** – (2, 3) r=2;
6 – (***Б***); **7** – (***Б***); **8** – (2, 3) r=4;
9 – (6, 8) r=2; **10** – (8, 7) r=2;
11 – (1, 8) r=4; **12** – (2, 7) r=4;
13 – (11,12) r=4; **14**–(11,13) r=2;
15 – (12,13) r=2; **16** – (1,11) r=2;
17 – (4, 14) r=2; **18** – (3, 13) r=2;
19 – (5, 15) r=2; **20** – (2, 12) r=2;
21 – (6, 11) r=2; **22** – (9, 14) r=2;
23 – (8, 13) r=2; **24** – (10,15) r=2;
25 – (7, 12) r=2

(Б)-Базовый, полный проход.

Пример промежуточного: **4** – (1, 3) r=2; - означает, что для построения 4-го луча, используются ранее найденные пути к вокселам 1, 3. Расстояние между лучами 1, 3 равно 2 пиксела.

Отметим что рассмотренный алгоритм отличен от предложенного в [11] аналогичного алгоритма использования когерентности траекторий применительно к BSP-деревьям. В указанном алгоритме строится выпуклый frustum, для которого все принадлежащие ему лучи попадают в один граничный лист или промежуточный узел. Использование такой пирамиды в нашем случае было бы менее эффективным, поскольку общий путь для нескольких лучей,

как правило, меньше общего пути для двух лучей. Кроме того, в нашем алгоритме не требуется дополнительных затрат на построение упомянутых пирамид.

5.3 Когерентность цветов

Независимую экономию вычислений при трассировке лучей можно получить за счет использования когерентности цветов. Если для трех последовательных пикселов (лучей) вычисленные закраски (цвета) различаются мало, то целесообразно выполнять трассировку только для крайних пикселов, закрашивая средний пиксел усредненным цветом. Такая оптимизация может давать существенный эффект (до 35 % на типовых сценах). Учитывая независимость этой оптимизации и оптимизации на основе когерентности октантных траекторий, было выполнено их объединение. В рассмотренной выше схеме обработки пикселов при вычислении закраски промежуточного пиксела на участке из 3-х пикселов вначале выполняется проверка на цвет, и если она не дает результата, то работа алгоритма продолжается. В итоге реализуется когерентность обоих типов.

6. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ

Результаты измерения эффективности использования когерентности траекторий в алгоритме трассировки октантных деревьев приведены в таблице 1. Эксперименты проводились на компьютере с процессором Athlon-1800 с видеоадаптером GeForce4 FX 5700 на сценах: «изоповерхности океанологических полей» (рис. 7-9) с глубинами октодеревьев 7-9; «пространственная карта города» (рис.10) с деревьями переменной глубины от 7 до 12; «молекула» (рис.11), состоящая из 4700 разноцветных сфер. На сценах первого типа показаны изоповерхности полей температуры, солености, скорости звука в заданном районе Японского моря. Сцена города состоит из рельефа с нанесенной текстурной картой и смоделированных по кадастровому плану зданий. При использовании разрешения экрана 1600x1200x32 для сцен первого типа эффективная площадь изображения ≈ 685729 пикселов (лучей трассировки), для второй сцены площадь ≈ 1639389 пикселов и для сцены «молекула» -1477514 пикселов. Для каждой сцены указано количество граничных вокселов, характеризующее потенциальную трудоемкость ее обработки. В таблице показаны времена визуализации разных сцен с разными глубинами октантных деревьев для исходного алгоритма дискретной трассировки и двух вариантов описанной выше оптимизации с использованием когерентности октантных траекторий соседних лучей. Визуализация выполнялась для центральной проекции. Чтобы исключить зависимость измеренных величин скорости рендеринга от площади получаемых изображений и используемого разрешения экрана, одновременно приведены и усредненные значения времен обработки для одного луча, которые, собственно, и отражают реальную картину сравнительной эффективности рендеринга рассматриваемых вариантов.

Из таблицы видно, что реализация когерентности октантных траекторий дает значительный прирост скорости рендеринга, причем «площадная интерполяция» несколько эффективнее «линейной». В среднем для указанных сцен она повышает скорость визуализации по отношению к полной трассировке на 35-40%. Поскольку когерентность по цветам дает дополнительную оптимизацию, в таблице приведены и результаты измерений с реализацией суммарной оптимизации по когерентности траекторий и цветов. Суммарная оптимизация дает увеличение скорости в 2 раза по отношению к исходной. Ухудшения качества изображения при этом не наблюдается. Как уже было отмечено выше, дополнительную экономию вычислений (5-10% по отношению к приведенным в табл.1 значениям времени рендеринга) дает режим с вычислением цветов граничных вокселов (для фиксированного положения источников освещения) на предварительном этапе.

7. ЗАКЛЮЧЕНИЕ

В статье представлен эффективный алгоритм дискретной трассировки лучей в октантных деревьях с решением, отличным от известных аналогов. Предложена оптимизация рендеринга за счет использования когерентности октантных траекторий близких лучей. Эксперименты, проведенные для различных тематических сцен, показали, что предложенная оптимизация в сочетании с использованием когерентности цветов соседних пикселов повышает скорость рендеринга в два раза без ущерба качеству изображения.

Работа выполнена при поддержке гранта РФФИ № 04-07-90287 и Программы №17 Президиума РАН

8. БИБЛИОГРАФИЯ

[1] M. Agate, R.L. Grimsdale, P.F. Lister/. The HERO Algorithm for Tracing Octrees. Advances in Computer Graphics Hardware IV R.L. Grimsdale, W. Strasser (eds) Spring-Verlag, New York, 1991.

[2] R. Endl, M. Sommer. Classification of Ray-Generators in Uniform Subdivisions an Octrees for Ray Tracing. Computer Graphics Forum Vol. 13(1), 1994.

[3] I. Gargantini, H.H. Atkinson. Ray tracing on Octree: Numerical Evaluation of the First Intersection. Computer Graphics Forum Vol. 12(4), 1993.

[4] Y. P. Kuzmin. Ray Traversal of Spatial Structures. Computer Graphics Forum Vol. 13(4), 1994, pp.223-227.

[5] J. Revelles, C. Urena, and M. Lastra. An Efficient Parametric Algorithm for Octree Traversal. Journal of WsCCG 8(2), pp. 212-219.

[6] A.S. Glassner. Space Subdivision for Fast Ray Tracing. IEEE Computer Graphics & Applications Vol. 4(10), 1984, pp.15-22.

[7] H. Samet. Implementing Ray Tracing with Octrees and Neighbor Finding. Computer & Graphics Vol. 13(4), 1989, pp.445-460.

[8] J. Spackman, P.J. Willis. The Smart Navigation of a Ray Through an Octree. Computers & Graphics Vol. 15(2), 1991, pp. 185-194.

[9] R. Yagel, D. Cohen, A. Kaufman. Discrete Ray Tracing. IEEE Computer Graphics & Applications Vol. 12(9), 1992, pp.19-28.

[10] М. Цыганков. Иерархическая трассировка лучей в октантных деревьях // GraphiCon 98. Conference Proceedings. Москва, сентябрь, 1998.

[11] V. Havran and J. Bittner. LCTS: ray shooting using longest common traversal sequences // EUROGRAPHICS 2000 / Volume 19 (2000), Number 3.

Об авторах

Бобков Валерий Александрович – д.т.н, зав. лабораторией машинной графики Института автоматики и процессов управления ДВО РАН.

Адрес: г. Владивосток, 690090, ул. Радио,5, ИАПУ.

Телефон: (4232)313776

E-mail: bobkov@iacp.dvo.ru

Мельман Сергей Владимирович – аспирант (м.н.с) лаборатории машинной графики Института автоматики и процессов управления ДВО РАН.

Адрес: г. Владивосток, 690090, ул. Радио,5, ИАПУ.

Телефон: (4232)313776

E-mail: gruzd@dvo.ru

Роньшин Юрий Иванович – вед. инж.-программист лаборатории машинной графики Института автоматики и процессов управления ДВО РАН.

Адрес: г. Владивосток, 690090, ул. Радио,5, ИАПУ.

Телефон: (4232)313776

E-mail: ronshin@iacp.dvo.ru

Optimization of Ray Tracing in Octrees

Abstract

Ray tracing algorithm on octrees is described. The algorithm is based on decomposition of 3D search child sub-voxel to 2D task in coordinate planes. Optimization of algorithm at the path coherency in octree is suggested. The notion of supporting rays with storing and common path of close rays is introduced. The algorithm of optimized traversal for intermediate rays, based on common paths, is presented. Two version of path coherency in octree realization is considered. Experimental efficiency values of considered optimization are received for different scenes. Comparative efficiency values of optimized discrete ray tracing algorithm and forward rendering algorithm are received also.

Keywords: *octree, ray tracing, path coherency in octree*

About the author(s)

Valery Bobkov is a professor at Institute for Automation and Control Processes, Department of Computer Graphics. His contact email is: bobkov@iacp.dvo.ru

Sergey Melman is a Ph.D. student at IACP. His contact email is: gruzd@dvo.ru

Yurij Ronshin is an engineer at Institute for Automation and Control Processes, Department of Computer Graphics. His contact email is:ronshin@iacp.dvo.ru

Сцена	Глубина октодерева	Количество вокселов	Полная трассировка (время в сек.) (1 луч мкс)	Оптимизированная трассировка время в сек (время 1 луча в мкс)			Прямой рендеринг (back-to-front)
				Когерентность октант-х траекторий		Когерентность траекторий + когерентность цветов	
				Линейная интерполяция (по строкам экрана)	«Площадная» интерполяция		
изопов-сть T	7	107632	1.73 (2.52)	1.34 (1.95)	1.14 (1.66)	0.88 (1.28)	0.19
изопов-сть V	7	134747	2.06 (3.00)	1.44 (2.10)	1.20 (1.74)	1.00 (1.46)	0.25
изопов-сть S	7	148687	1.94 (2.83)	1.32 (1.92)	1.11 (1.62)	0.95 (1.39)	0.50
изопов-сть T	8	434982	1.95 (2.84)	1.53 (2.23)	1.34 (1.95)	1.00 (1.46)	0.34
изопов-сть V	8	560410	2.36 (3.43)	1.67 (2.44)	1.45 (2.11)	1.13 (1.65)	0.45
изопов-сть S	8	614339	2.20 (3.21)	1.49 (2.17)	1.31 (1.91)	1.04 (1.52)	0.72
изопов-сть T	9	1751387	2.25 (3.21)	1.76 (2.57)	1.66 (2.42)	1.14 (1.66)	0.95
изопов-сть V	9	1280668	2.74 (4.00)	1.95 (2.84)	1.81 (2.64)	1.32 (1.92)	1.25
изопов-сть S	9	1489524	2.52 (3.67)	1.73 (2.52)	1.61 (2.35)	1.23 (1.79)	1.60
город	7,9	117519	4.31 (2.63)	2.83 (1.73)	2.41 (1.47)	2.06 (1.26)	-
город	9,9	335574	5.20 (3.17)	3.58 (2.18)	3.05 (1.86)	2.63 (1.60)	-
город	10,11	2969932	5.89 (3.59)	4.22 (2.57)	3.72 (2.27)	3.08 (1.88)	-
город	10,12	8773104	6.48 (3.95)	4.66 (2.84)	4.14 (2.53)	3.38 (2.06)	-
молекула	9	675672	4.69 (3.17)	2.36 (1.60)	1.97 (1.33)	2.14 (1.45)	-

Табл. 1. Сравнительная эффективность оптимизаций трассировки для разных сцен
(площадь изображения для изоповерхности – 685729 пикселов (лучей),
площадь изображения для города – 1639389, площадь изображения для сцены «молекула» - 1477514)

Рис.7. Изотермическая поверхность (район Японского моря)

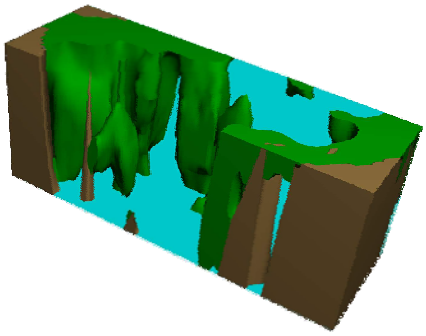


Рис.8. Изоповерхность солености воды

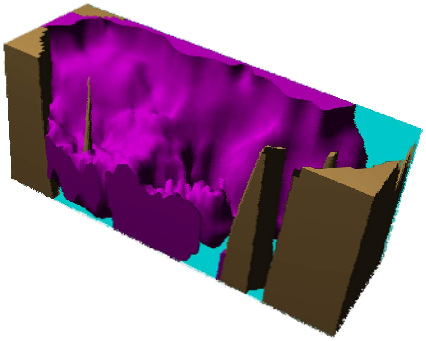


Рис.9. Изоповерхность скорости звука в океане

Рис.10. Город

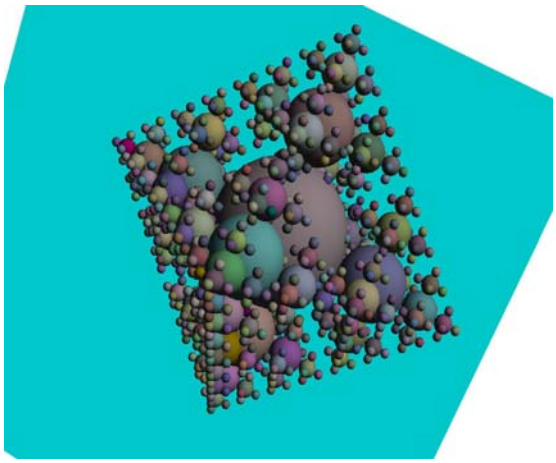


Рис.11. Молекула

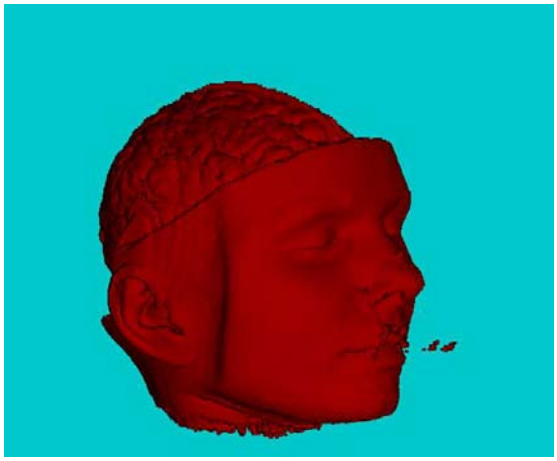


Рис.12. Визуализация данных магнито-резонансной томограммы

Рис.13. Визуализация данных компьютерной томограммы