Математические структуры и моделирование
2010, вып. 21, с. 82–95

УДК 004.92

АЛГОРИТМЫ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ

Д.И. Собинов, В.В. Коробицын

В статье приводится обзор современных алгоритмов и подходов для построения систем обнаружения столкновений объектов. Приводится классификация алгоритмов по нескольким признакам.

1. Введение

В настоящее время, в связи со значительным прогрессом в области разработки аппаратного обеспечения вычислительной техники, стало возможным осуществлять моделирование движения реального мира в соответствии с физическими законами. Результаты такого рода моделирования используются в различных областях человеческой деятельности: при разработке систем виртуальной реальности и тренажеров, компьютерных игр, систем автоматизированного проектирования, в робототехнике и т.д. Базовой частью подобных систем является подсистема обнаружения столкновений. В общем случае, задача данной подсистемы состоит в проверке простого факта: пересекаются ли два объекта в пространстве. В случае, если объекты пересекаются, часто бывает необходима дополнительная информация, такая как нахождение объема пересечения, его аппроксимация в виде набора точек или простого геометрического объекта, глубина взаимопроникновения объектов. Эта информация используется, например, для оценки силы, которая должна быть приложена к моделируемым телам для их расталкивания в соответствии с законами механики.

Усилия исследователей в рассматриваемой области сосредоточены на решении следующих задач:

- создание универсальных алгоритмов и подходов для прозрачной работы с произвольными объектами: выпуклыми и вогнутыми, деформируемыми и абсолютно твердыми телами, статическими и движущимися и т.д.;
- повышение эффективности алгоритмов, получение максимальной производительности с учетом аппаратной платформы, на которой будет эксплуатироваться программная система;
- повышение устойчивости результатов;

Омский государственный университет им. Ф.М. Достоевского.
E-mail: dsobinov@gmail.com

- упрощение алгоритмов при сохранении других качественных параметров, что позволяет уменьшить стоимость реализации подсистем обнаружения столкновений.

За последние годы исследователями было предложено большое количество разнообразных алгоритмов. Были даже изданы книги [1, 2], обобщающие накопленный опыт в построении систем обнаружения столкновений. Целью данной статьи является общая классификация подходов и алгоритмов определения столкновений и обзор тех, что появились за последние несколько лет.

2. Классификация алгоритмов определения столкновений

2.1. По представлению входных данных

Алгоритмы, использующиеся в системах обнаружения столкновений, могут быть классифицированы по способу представления объектов на входе системы. Наиболее часто встречаются входные данные следующих типов:

- сетки из треугольников;
- конструктивная блочная геометрия;
- неявно заданная геометрия.

Использование первого типа обусловлено тем, что треугольник является базовым геометрическим примитивом для отображения на экране дисплея сложных объектов. Координаты точек передаются через API драйверу графического адаптера, который растеризует входные данные и выводит на экран.

Конструктивная блочная геометрия часто используется в САПР. Комбинируя простые базовые трехмерные объекты, такие как сферы, прямоугольные параллелепипеды, конусы с помощью булевых операций, на выходе получаем сложный объект.

Неявное задание объекта предполагает задание формы объекта в виде математического уравнения. Например, область, ограниченную сферой, можно задать в виде уравнения $x^2 + y^2 + z^2 \leq r^2$. Очевидно, что этот способ слишком громоздкий и не годится для сложных объектов.

Далее, если не указано явно, предполагается, что в качестве входных данных для алгоритма обнаружения столкновений используются полигональные сетки из треугольников.

2.2. По связям с системой моделирования

Чаще всего система обнаружения столкновений является подсистемой какой-либо более крупной системы, например моделирования хирургической операции. При этом подсистема определения столкновений может либо являться «черным ящиком» для остальных компонентов, либо быть тесно интегрированной с остальными компонентами.

В первом случае такая подсистема может разрабатываться независимо от других компонентов системы в качестве сторонней библиотеки. Преимуществом подобного подхода является возможность повторного использования наработанного программного кода. Недостатком же можно считать сложность разработки и невозможность учесть все требования в каждом конкретном случае использования.

При разработке подсистемы обнаружения столкновений может учитываться специфика проекта, например необходимость в использовании особого формата задания объектов, или тесная интеграция системы с компонентом моделирования, например обнаружение столкновений и одновременное вычисление расталкивающей силы для двух тел [3].

Для этого случая могут применяться специализированные алгоритмы, которые позволяют добиться лучшей производительности и устойчивости.

2.3. Классификация по фазам

Наиболее простым методом для обнаружения столкновений между двумя объектами A и B является попарная проверка всех элементов, составляющих A, со всеми элементами из B. Учитывая, что на каждом шаге моделирования может участвовать большое количество объектов, состоящих из тысяч многоугольников, можно заключить, что обнаружение столкновений является вычислительно сложной задачей. Если сразу проводить детальную проверку на пересечение с нахождением всей необходимой информации (характеристики объема пересечения, глубина взаимопроникновения и т.д.), то в итоге система будет обладать низкой масштабируемостью и уже при сравнительно небольшом количестве моделируемых объектов скорость работы не будет удовлетворять жестким условиям систем реального времени.

В связи с этим алгоритмы определения столкновений часто делят на последовательность шагов, или фаз. Одно из первых упоминаний такого разделения приводится в работе [4]. Широкая фаза отсеивает те пары объектов (или элементов одного объекта), которые заведомо не пересекаются. Узкая фаза необходима для детального рассмотрения каждой из пар объектов, не отброшенных при выполнении широкой фазы. На данном этапе, если объекты пересекаются, то для них находится контактная информация.

Соответственно, наиболее общим признаком, по которому можно классифицировать алгоритмы определения столкновений, будет принадлежность к широкой или узкой фазе.

2.3.1. Широкая фаза

Для широкой фазы подходят алгоритмы, которые позволяют с минимальными временными затратами отбраковать наибольшее количество пар. При этом преимущество отдается скорости работы алгоритма. Для этого вместо сложного исходного геометрического представления объекта используют ограничивающую геометрию, или ограничивающий объем. Ограничивающий объем — это область пространства, которая содержит данный объект. Для широкой фазы

используют простые по форме объекты, чаще всего сферы или прямоугольные параллелепипеды. Габариты ограничивающего объема должны быть такими, чтобы исходный объект целиком помещался в него, но при этом имел минимально возможный объем.

Таким образом, простая попарная проверка друг с другом объектов с большим количеством элементов заменяется на проверку на пересечение между простыми объектами. Например, для определения, пересекаются ли две сферы, необходимо оценить истинность следующего выражения:

$$R_1 + R_2 \geq |O_1 - O_2|,$$

где R_1 и R_2 – радиусы сфер, O_1 и O_2 – радиус-векторы их центров.

Если два ограничивающих объема пересекаются, то далее производится их проверка на пересечение с использованием исходного геометрического представления (т.е. происходит передача пары объектов на узкую фазу). В реальных моделируемых сценах лишь небольшое количество объектов находится в непосредственной близости, следовательно, получается заметный выигрыш в производительности.

Дальнейшее увеличение производительности можно получить, если вместо наиболее простого случая попарной проверки всех моделируемых объектов применить либо техники пространственного разделения сцены, либо алгоритмы, учитывающие когерентность временных шагов моделирования, т.е. когда положения тел изменяются не слишком сильно по прошествии малого временного шага. К последней группе относится алгоритм «sweep-and-prune», впервые описанный в [5]. Основная идея алгоритма — проецирование объектов сцены на одну или несколько координатных осей, сохранение минимальных и максимальных проекций для объектов в списки и далее на каждом временном шаге поддержании списков в упорядоченном состоянии. Если временная когерентность высокая, то затраты на сортировку будут незначительными при использовании подходящих методов, например сортировки вставками. Запрос на столкновение для объекта может быть выполнен за время, близкое к константе путем проверки соседних значений из списка и получение соответствующих им объектов. В работе [6] приводится целый ряд улучшений данного алгоритма, позволяющих повысить его эффективность. Авторами была проведена оценка, насколько то или иное улучшение влияет на производительность для различных конфигураций моделируемой сцены.

Пространственное разделение предполагает проверку на пересечение только тех объектов, которые попали в один пространственный регион. Один из способов построения разбиения предполагает рекурсивное деление сцены, пока соблюдается некоторое условие, например размер региона, содержащегося в листовом узле, не слишком мал.

Наиболее простой вариант разбиения — равномерное деление плоскости на двумерные ячейки одного размера и далее проецирование всех объектов сцены на эту плоскость. Таким образом, каждый объект ассоциирован с одной или несколькими ячейками. Чтобы найти объекты, потенциально пересекающиеся с заданным, необходимо проверить только те объекты, которые находятся в той

же ячейке, что и заданный. В [7] показано, как подобный алгоритм может быть реализован на CUDA на GPU со значительным выигрышем в производительности по сравнению с версией на CPU. Основным недостатком равномерного разбиения пространства является неэффективность проверок для случая, когда сцена содержит объекты с сильно различающимися размерами. В этом случае сложно подобрать подходящий размер ячейки.

Также в качестве схемы разбиения пространства можно использовать различные иерархические структуры: восьмеричные деревья или квадродеревья, иерархическое разбиение на ячейки [1]. В работе [8] были использованы деревья бинарного разбиения (Binary Space Partitioning) и предложен алгоритм эффективного обновления структуры такого дерева при изменении позиций объектов сцены с течением времени. Авторам удалось найти компромисс между затратами на нахождение потенциально пересекающихся пар объектов и количеством таких пар на выходе широкой фазы.

2.3.2. Узкая фаза

Алгоритм узкой фазы принимает на вход список пар объектов, отобранных широкой фазой как наиболее вероятных кандидатов на столкновение. Используя исходное геометрическое представление объектов, алгоритм на выходе дает ответ на вопрос, пересекаются ли на самом деле объекты, поданные на вход. Если пара объектов пересекается, алгоритм также может вернуть дополнительную информацию:

- глубину взаимопроникновения объектов друг в друга, т.е. минимальное расстояние, на которое нужно переместить объекты, чтобы они перестали пересекаться;
- нормаль, вдоль которой считается глубина взаимопроникновения;
- аппроксимацию объема пересечения.

Эта информация может использоваться для нахождения времени, когда объекты имели одну точку соприкосновения, для вычисления импульса, который необходимо приложить к объектам для их реалистичного расталкивания и т.д.

Алгоритмы узкой фазы можно разделить на две группы по характеру получения результата:

- работающие в пространстве объектов;
- работающие в пространстве изображений.

Работающие в пространстве изображений алгоритмы характеризуются тем, что используют графические операции, такие как растеризация, для обнаружения столкновений. Таким образом, объекты сцены отображаются в пространство с ограниченной размерностью, например в буфер кадра или LDI (Layered Depth Image, [9]), и, следовательно, столкновение находится с погрешностью, зависящей от размерности итогового пространства.

Соответственно, алгоритмы, не имеющие свойств алгоритмов пространства изображений, относятся к группе алгоритмов, работающих в пространстве объектов.

Для последних лет также стало актуально классифицировать алгоритмы обнаружения столкновений на те, что рассчитаны на работу на CPU, и на те, которые разработаны для GPU. Обусловлено это разделение тем, что потоковая модель вычислений на GPU существенно отличается от модели вычисление на CPU, и, следовательно, ранее разработанные алгоритмы для CPU могут быть переработаны для получения выигрыша от большой вычислительной мощности GPU. В то же время для GPU было разработано немало алгоритмов, не имеющих аналогов для CPU.

3. Обзор алгоритмов узкой фазы

3.1. Алгоритмы для CPU

В данную категорию, прежде всего, попадают алгоритмы на основе иерархий ограничивающих объемов. Листовые узлы подобных иерархий могут содержать либо по одному примитиву объекта (чаще всего это треугольник), либо некоторое подмножество примитивов объекта, в котором в дальнейшем необходимо проверить на пересечение каждую пару. Выбор глубины разбиения влияет на соотношение занимаемой памяти и количества вычислительных ресурсов, затрачиваемых при работе алгоритма.

Для узкой фазы было предложено множество типов ограничивающей геометрии:

- сферы [4];
- выровненные по осям прямоугольные параллелепипеды [11];
- ориентированные прямоугольные параллелепипеды [12, 13];
- дискретно-ориентированные многогранники, или k-DOP (Discrete Orientation Polytope; многогранник, у которого нормали к граням принадлежат заранее определенному фиксированному множеству направлений размерности k) [14] и др.

Выбор ограничивающего объема является компромиссом между вычислительной сложностью проверки на пересечение и тем, насколько хорошо выбранный ограничивающий объем аппроксимирует объект. Примеры ограничивающей геометрии приведены на рисунке 1.

Алгоритм обнаружения столкновений на основе иерархий ограничивающих объемов состоит в одновременном прохождении в ширину или в глубину деревьев, соответствующих двум объектам. На каждом шаге проводится проверка на пересечение ограничивающих объемов. Если ограничивающие объемы текущих узлов не пересекаются, то эти узлы и их поддеревья отбрасываются. Иначе, проводится рекурсивная проверка на пересечение у дочерних узлов, пока не будут достигнуты листовые узлы. На выходе получаются пары листовых

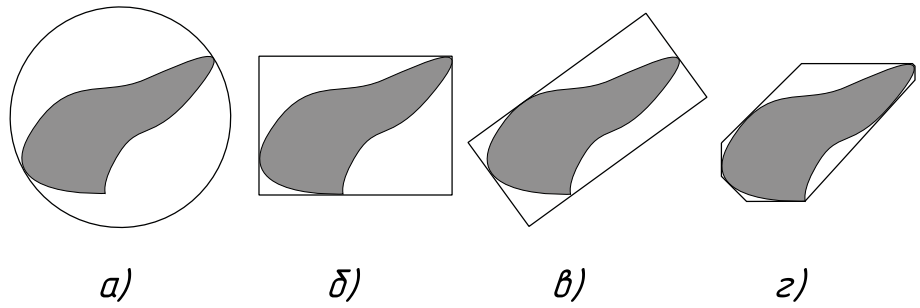

Рис. 1. Примеры ограничивающей геометрии: а) ограничивающая сфера; б) ограничивающий бокс, выровненный по осям (AABB); в) ориентированный ограничивающий бокс (OBB); г) 6-DOP

узлов от каждого из двух объектов, в которых необходимо провести проверку на пересечение на уровне примитивов.

В работе [10] предложено улучшение традиционной схемы путем хранения в листах дерева отдельных элементов (вершин, ребер, граней) так, чтобы каждый элемент присутствовал только в одном листе. Это позволяет значительно сократить количество проверок ребро-ребро, вершина-грань и ребро-грань на последнем шаге алгоритма, когда происходят проверки на уровне отдельных примитивов.

Ввиду того что построение иерархии ограничивающих объемов для объектов сцены является ресурсозатратным процессом, его проводят заранее и ассоциируют с исходными объектами. Таким образом, данный тип алгоритмов в исходном виде хорошо подходит для обнаружения столкновений между твердыми телами. Для деформируемых тел требуется перестроение иерархии или частичное ее обновление, т.к. положение элементов для таких объектов меняется относительно друг друга. Полное перестроение иерархии на каждом временном шаге не может быть использовано для моделирования в реальном времени для даже умеренных по сложности сцен, потому что требует значительных вычислительных затрат. За последние несколько лет в связи с возросшим интересом к моделированию деформируемых тел были разработаны менее ресурсозатратные методы обновления иерархий. Так, в работе [15] был предложен эффективный подход для восьмеричного дерева иерархий описывающих объемов на основе временной когерентности сцены и алгоритма сортировки вставками. Время, затрачиваемое на подобную сортировку, будет тем меньше, чем незначительнее были изменения в положении примитивов объекта. Таким образом, при высокой временной когерентности затраты на обновление иерархии будут малы.

В [16] был предложен подход для ускорения обновления иерархии путем пропуска или упрощения процесса обновления для нескольких временных шагов. Для этой цели описывающий объем расширяют на некоторую величину, и обновление не требуется, пока примитив не сдвинулся более, чем на эту величину. В качестве ограничивающих объемов были использованы 18-DOP'ы.

В качестве базовой структуры данных для представления иерархии ограничивающих объемов для обнаружения столкновений разрушающихся объектов

в работе [17] были использованы АВЛ-деревья. Авторами был представлен алгоритм для динамического изменения структуры дерева в случае изменения позиций, а также при добавлении или удалении отдельных элементов на основе балансировки АВЛ-дерева.

В [18] были использованы динамические деревья ограничивающих объемов. Было предложено проводить обнаружение столкновений в два шага: (1) обновление ограничивающих объемов по всему дереву, (2) запрос на столкновение. Во время первого шага заново просчитываются ограничивающие объемы у существующих узлов, а также помечаются на удаление поддеревья, которые требуют перестройки. Критерием того, следует ли помечать поддерево как негодное для дальнейшего использования, является сравнение отношений объемов старого родительского узла с просчитанным заново. На втором шаге алгоритм обходит деревья объектов в поисках перекрывающихся узлов и перестраивает помеченные на первом шаге поддеревья. Было показано, что такой подход эффективен для обнаружения столкновений между деформируемыми объектами, но для случая самопересечений примитивов внутри одного объекта его эффективность значительно снижается.

Среди алгоритмов узкой фазы можно выделить еще одну группу, в основе которой лежит работа напрямую с вершинами, ребрами и гранями объектов. К этой категории относится алгоритм Лина-Кэнни [19] и V-Clip [20], который устраняет такие недостатки первого, как неспособность работать в случае проникновения объектов друг в друга (а не только касания), неустойчивость в некоторых вырожденных взаимных конфигурациях объектов и сложность реализации.

Популярной группой алгоритмов являются также алгоритмы, основанные на симплексах. Полигональный объект представляется как выпуклая оболочка, и алгоритм работает над симплексами из подмножества множества точек этой оболочки. Наиболее известен алгоритм Гильберта-Джонсона-Кеерти (GJK) [21] и ряд его улучшений. Так, в [22] была добавлена возможность нахождения расстояния между объектами (или глубины проникновения, если они пересекаются). Работа [23] направлена на значительное улучшение стабильности и скорости алгоритма. GJK в явном виде не использует геометрию исходных объектов, а работает над их разностью Минковского. Таким образом, задача нахождения расстояния между двумя объектами сводится к нахождению расстояния от начала координат до разности Минковского объектов.

3.2. Алгоритмы для GPU

Последние 15 лет характеризуются бурным ростом производительности графических адаптеров. Графические адаптеры привлекательны для исследователей в связи с их возможностью работать одновременно с миллионами графических элементов. Также расширение возможностей для программирования с использованием шейдеров приводит к тому, что все большее количество вычислений, не связанных напрямую с выводом графики на экран, переносится с CPU на GPU. Этому способствует и появление унифицированного API OpenCL, обес-

печивающего стандартизированный доступ к видеокарте как к устройству с высокой степенью параллелизма. В настоящее время на рынке доступны потребительские графические адаптеры с низкой ценой и большой производительностью.

3.2.1. Алгоритмы, работающие в пространстве изображений

Для большинства алгоритмов, работающих в пространстве изображений, можно выделить общий ряд достоинств и недостатков. Так, к достоинствам можно отнести то, что часто не требуются шаги для начальной инициализации и обычно не имеет значения, в каком виде поступает геометрия на вход. Недостатком является невозможность получения точного результата из-за ограниченной разрешающей способности пространства изображений и количества бит на пиксель для z-буфера. Также обычным ограничением является требование на замкнутость поверхности у объектов.

Одна из первых работ с использованием методов пространства изображений была представлена в [24]. Метод состоит в отрисовке объектов вдоль заданной оси с сохранением z-значений глубины в списках для каждого пикселя. Отрисовка производится дважды для получения переднего и заднего интервалов объекта. Столкновение может быть обнаружено путем нахождения перекрывающихся интервалов z-значений для объектов. Предложенный метод работает только для выпуклой геометрии.

В [26] было предложено использовать отрисовку объектов в два LDI, по одному на объект, с ограничением видимой области в рамках AABB, соответствующего пересечению двух AABB объектов. Применяя операцию булева пересечения между LDI пары объектов, можно узнать, пересекаются ли объекты. Алгоритм не ограничен только выпуклыми объектами и может быть использован для произвольных объектов с замкнутой поверхностью и для нахождения самопересечений. Авторы показали, что для сложных объектов предложенный метод неэффективен, т.к. требует нескольких проходов рендеринга в LDI и накладных расходов в виде интенсивного чтения из памяти GPU.

Эффективный алгоритм с широким использованием возможностей GPU и OpenGL API, таких как occlusion query, пользовательские буферы кадра и шейдеры, был предложен в [27]. Так же, как и в [26], находится растеризуемый объем в виде AABB, но рендеринг происходит в общий для всех объектов набор текстур с сохранением идентификатора объекта, а не в два LDI. Предполагается, что объекты не самопересекаются и имеют замкнутую поверхность. На выходе алгоритма получаются точки столкновения между объектами, и число объектов, участвующих в столкновении, может быть больше двух.

Схожий с предыдущим подход, но на основе возможностей DirectX 10, был предложен в [28]. Рендеринг производится одновременно в восемь слоев глубины, таким образом значительно сокращая количество вызовов графического API и повышая общую производительность. Уменьшение потока GPU $\rightarrow$ CPU было достигнуто благодаря использованию геометрического шейдера, позволившего отбрасывать не несущие информацию о столкновении элементы. В каче-

стве результата алгоритм возвращает пары треугольников исходных объектов, которые требуется проверить на пересечение на CPU.

Еще один подход на основе LDI представлен в [3]. Область возможного пересечения отрисовывается в три LDI, по одному для каждой из взаимно-перпендикулярных осей видимости. Далее определяется объем, занимаемый пересечением объектов:

$$V = a \sum_{(i,j) \in C} (-1)^d z_{ij},$$

где a — площадь одного пиксела, z — глубина, d — 1 или 2 в зависимости от порядка пиксела в LDI, C — множество индексов фрагментов, покрывающих поверхность объема пересечения. Далее, используя объем пересечения, авторы предлагают получить на его основе величину расталкивающей силы для столкнувшихся объектов. Основной недостаток алгоритма состоит в использовании трех LDI и соответственно трех проходов рендеринга: это снижает производительность из-за большей нагрузки на видеокарту и увеличивает объем занимаемой памяти.

Принцип, лежащий в работе алгоритмов [3, 24, 26–28], достаточно прост. Два луча, выпущенных вдоль оси Z через столкнувшиеся объекты, пересекают их и образуют интервалы $[z_{A0}, z_{A1}], [z_{B0}, z_{B1}]$ и $[z'_{A0}, z'_{A1}], [z'_{B0}, z'_{B1}]$ для первого и второго лучей соответственно, как показано на рисунке 2, а. При этом столкновение можно обнаружить, анализируя на перекрытие полученные интервалы для оси Z, т.е. значения из буфера глубины графического адаптера после рендеринга. Для повышения точности алгоритма и производительности рендеринг выполняется для пересечения двух AABB объектов. Результирующий AABB называется *регион интереса* или *объем интереса*. С помощью графического API выставляется ортографическая проекция с направлением вида по одной из осей координат с ограничением области видимости по полученному AABB (закрашен серым цветом на рисунке 2, б).

Попытка решить проблему низкой пропускной способности шины GPU → CPU, как основной причины низкой производительности алгоритмов на GPU, была предпринята в пакете для обнаружения столкновений CULLIDE [25]. Предложенный алгоритм проверяет видимость объекта O относительно множества объектов S. Множество потенциально пересекающихся примитивов определяется с помощью запроса к видеокарте, загорожен ли O объектами из S. Если не загорожен, то O не пересекается с S и, следовательно, не включается в множество потенциально пересекающихся примитивов. CULLIDE делит исходный объект на множество подобъектов на предварительной стадии и далее проверяет видимость путем прохода по иерархии подобъектов. Недостатком данного алгоритма является дорогостоящая процедура на предварительном шаге, что ограничивает использование этого подхода для разрушаемых тел.

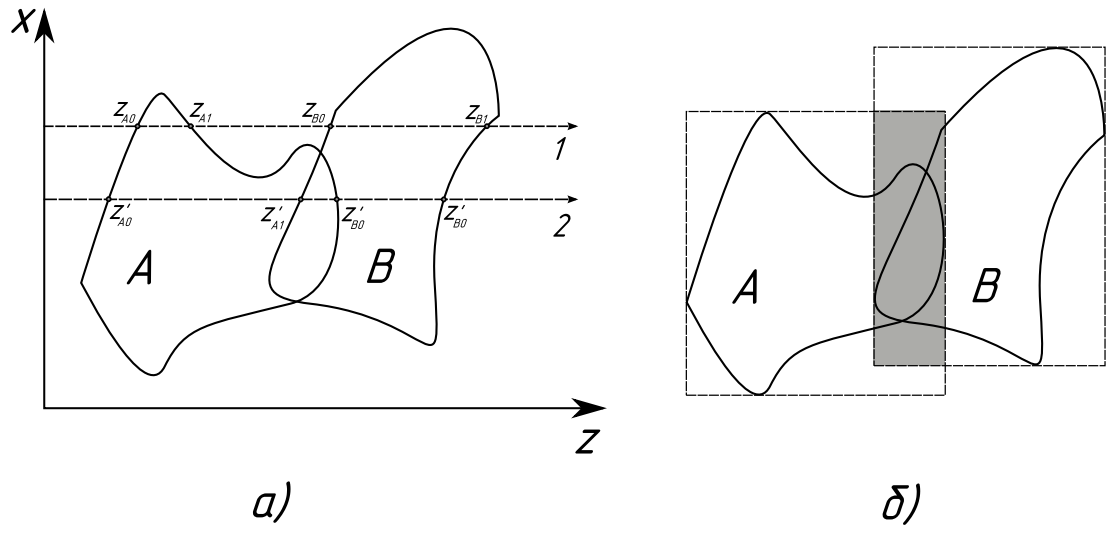

Рис. 2. Принцип работы большинства алгоритмов в пространстве изображений: а) скан-линии 1 и 2 вдоль оси Z для двух пересекающихся объектов; б) нахождения региона интереса

3.2.2. Алгоритмы, работающие в пространстве объектов

Появление программируемости в широко распространенных потребительских графических адаптерах и быстрое наращивание их вычислительной мощности явилось толчком для исследований в направлении разработки алгоритмов обнаружения столкновений не только в рамках пространства изображений, но и в пространстве объектов, что позволило бы избавиться от недостатков алгоритмов пространства изображений, перечисленных в предыдущем подразделе.

Одним из первых алгоритмов в пространстве объектов на GPU является [29]. Подход основан на использовании балансированных иерархий ограничивающих объемов, обход деревьев осуществляется в ширину, вместо обычно используемого обхода в глубину. Все вычисления во время обхода дерева, включая проверки на пересечения между примитивами, реализованы на фрагментных и вершинных шейдерах. Алгоритм не имеет ограничений на форму, топологию или замкнутость поверхности объектов, но может быть использован только для твердых тел. Авторы отмечают, что алгоритм лишь немногим быстрее версии, реализованной для CPU.

В работе [30] предложен подход на основе дерева из AABB для обнаружения столкновений между деформируемыми объектами, заданными NURBS кривыми. Описаны процедуры построения деревьев ограничивающих объемов на GPU в реальном времени, эффективная техника обхода дерева с одновременным сокращением потока, поступающего на CPU (stream reduction) на основе мипмап-уровней текстур, а также дополнение в виде алгоритма для определения самопересечения. Было показано, что для объектов умеренного размера реализация алгоритма работает в реальном времени.

В [31] дерево из AABB для каждого из объектов передается в GPU как

два списка, или потока. В представлении графического API эти списки представлены как 1D текстуры, где каждый элемент такой текстуры в R, G и B компонентах хранит координаты углов AABB, а в альфа-компоненте признак наличия этого узла в дереве, т.к. передаются только листовые узлы самого нижнего уровня дерева, и не все они могут присутствовать в дереве для объекта. Далее на GPU выполняются всевозможные попарные проверки между двумя потоками для двух объектов, а результат для каждого элемента из дерева записывается как ложь/истина и передается для дальнейшей проверки на CPU на предмет пересечения соответствующих узлам треугольников. Для деформируемых тел иерархия каждый раз обновляется. В результате при реализации на шейдерах получается достаточно высокая производительность, авторы отмечают трехкратное превосходство по скорости над CULLIDE [25], который был описан в предыдущем подразделе.

4. Заключение

За последние три десятилетия в литературе было представлено большое количество методов и алгоритмов для обнаружения столкновений. Каждый из них имеет недостатки или может быть использован только при ограничениях на способ задания формы объектов, характер движения объектов или на тип аппаратного обеспечения, используемого для моделирования сцены и т.п. Тем не менее на данный момент наиболее широко используются алгоритмы на основе иерархий ограничивающих объемов, а также постепенно увеличивается доля алгоритмов, созданных для работы на параллельных потоковых программируемых устройствах, к которым относятся современные GPU.

Литература

1. Ericson C. Real-Time Collision Detection. San Franciso: Morgan Kaufmann Publishers, 2005. 593 p.
2. Bergen G. Collision Detection in Interactive 3D Environments. San Franciso: Morgan Kaufmann Publishers, 2004. 278 p.
3. Faure F. Image-based Collision Detection and Response between Arbitrary Volume Objects / F. Faure, S. Barbier, J. Allard, F. Falipou // Proceedings of the 2008 ACM SIGGRAPH Eurographics Symposium on Computer Animation. 2008. P. 155–162.
4. Hubbard P. Collision Detection for Interactive Graphics Applications // IEEE Transactions on Visualization and Computer Graphics. 1995. V. 1, N. 3. P. 218–230.
5. Baraff D. Dynamic Simulation of Non-Penetrating Rigid Bodies: PhD thesis. Cornell University, 1992.
6. Tracy D., Buss S., Woods B. Efficient Large-Scale Sweep and Prune Methods with AABB Insertion and Removal // Proceedings of the 2009 IEEE Virtual Reality Conference. 2009. P. 191–198.
7. Le Grand S. Broad-Phase Collision Detection with CUDA // GPU Gems 3 / edt. by Hubert Nguyen. 2007. P. 697–723.

8. Luque R., Comba J., Freitas C. Broad-Phase Collision Detection Using Semi-Adjusting BSP-Trees // Proceedings of the symposium on Interactive 3D graphics and games. 2005. P. 179–186.
9. Layered Depth Images / J. Shade [and others] // Proceedings of the 25th conference on Computer graphics and interactive techniques. New York, 1998. P. 231–242.
10. Curtis S., Tamstorf R., D. Manocha D. Fast Collision Detection for Deformable Models using Representative-Triangles // Proceedings of the 2008 symposium on Interactive 3D graphics and games. 2008. P. 61–69.
11. van den Bergen G. Efficient Collision Detection of Complex Deformable Models Using AABB Trees // Journal of Graphics Tools. 1997. V. 2, N. 4. P. 1—13.
12. Zachmann G., Felger W. The BoxTree: Enable Real-time and Exact Collision Detection of Arbitrary Polyhedra // Proceedings of SIVE'95. 1995. P. 104–113.
13. Gottschalk S., Lin M.C., Manocha D. OBBTree: a hierarchical structure for rapid interference detection // Proceedings of Computer graphics and interactive techniques. 1996. P. 171–180.
14. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs / J. Klosowski [and others] // IEEE Transactions on Visualization And Computer Graphics. 1998. V. 4, N. 1. P. 21–36.
15. Ganovelli F., Dingliana J., O'Sullivan C. BucketTree: Improving Collision Detection Between Deformable Objects // Proceedings of SCCG2000 Spring Conf. on Comp. Graphics. 2000. P. 156–163.
16. Mezger J., Kimmerle S., Etzmuss O. Hierarchical Techniques in Collision Detection for Cloth Animation // Journal of WSCG. 2003. V. 11, N. 2. P. 322–329.
17. Balanced Hierarchies for Collision Detection between Fracturing Objects / M.A. Otaduy [and others] // Proceedings of 2007 IEEE Virtual Reality Conference. 2007. P. 83–90.
18. Larsson T., Akenin-Moeller T. A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection // Computer and Graphics. 2006. V. 30, N. 3. P. 451–460.
19. Lin M. C. Efficient Collision Detection for Animation and Robotics: PhD thesis. Berkeley: University of California, 1993. 147 p.
20. Mirtich B. V-Clip: Fast and Robust Polyhedral Collision Detection // ACM Transactions on Graphics. 1998. V. 17, N. 3. P. 177–208.
21. Gilbert E.G., Johnson D.W., Keerthi S.S. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space // IEEE Journal of Robotics and Automation. 1998. V. 4, N 2. P. 193–203.
22. Cameron S. Enhancing GJK: Computing Minimum and Penetration Distance between Convex Polyhedra // Proceedings IEEE Int. Conf. on Robotics and Automation. 1997. P. 3112–3117.
23. van den Bergen G. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects // Journal of Graphics Tools. 1999. V. 4, N. 2. P. 7–25.
24. Shinya M., Forgue M.-C. Interference Detection Through Rasterization // The Journal of Visualization and Computer Animation. 1991. V. 2, N. 4. P. 132–134.
25. Govindaraju N., Lin M.C., Manocha D. Quick-CULLIDE: Fast Inter- and Intra-object Collision Culling Using Graphics Hardware // Proceedings of IEEE Virtual Reality. 2005. P. 59–66.
26. Heidelberger B., Teschner M., Gross M. Detection of Collisions and Self-collisions Using

Image-space Techniques // Journal of WSCG. 2004. V. 12, N. 3. P. 145–152.

27. Jang H.-Y., Jeong T.S., Han J.H. GPU-based Image-space Approach to Collision Detection among Closed Objects. 2006. URL: http://drjang.kr/Formal/paper/2006_GPU-based_Imate-space_Approach_to_Collision_Detection_among_Closed_Objects_PG.pdf (01.03.2010).
28. Jang H.-Y., Han J.H. Fast collision detection using the A-buffer // Visual Computer. 2008. V. 24, N. 7-9. P. 659–667.
29. Gress A., Zachmann G. Object-space Interference Detection on Programmable Graphics Hardware // Proceedings of SIAM Conf. Geometric Design and Computing. 2004. P. 311–328.
30. Gress A., Guthe M., Clein R. GPU-based Collision Detection for Deformable Parameterized Surfaces // Proceedings of Eurographics 2006. V. 25, N. 3. P. 497–506.
31. Zhang X., Kim Y.J. Interactive Collision Detection for Deformable Models using Streaming AABBs // IEEE Transactions on Visualization And Computer Graphics. 2007. V. 13, N. 2. P. 31–329.