**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра дифференциальных уравнений и системного анализа**

**АЛГОРИТМЫ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ В КОМПЬЮТЕРНОЙ ГРАФИКЕ**

Курсовая работа

Глухотко Владислава Владимировича

студента 3 курса,

специальность 1-31 03 09

Компьютерная математика   
и системный анализ

Научный руководитель:  
доцент Н. Л. Щеглова

Минск, 2021

**ОГЛАВЛЕНИЕ**

**ВВЕДЕНИЕ ……………………………………………………………………………………………………….** 2

**ГЛАВА 1 ОСНОВНЫЕ ЗАДАЧИ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ В КОМПЬЮТЕРНОЙ ГРАФИКЕ ………………………………………………………………………………………………………………………** 3

1.1 ИСТОРИЧЕСКАЯ СПРАВКА О ПРЕДМЕТЕ И КЛЮЧЕВЫЕ РЕЗУЛЬТАТЫ …………………………… 3

1.2 ОСНОВНЫЕ ПОНЯТИЯ, СВЯЗАННЫЕ С ПРЕДМЕТОМ, И ПОЯСНЕНИЯ …………………………… 5

*1.2.1* *Обнаружение и обработка коллизий ..……………………………………………………* 5

*1.2.2* *Апостериорный и априорный подходы ……………………………………… …………..*  6

1.3 ОБЗОР НЕКОТОРЫХ ЗАДАЧ И АЛГОРИТМОВ ИХ РЕШЕНИЯ ..…………………………………….. 7

1.4 МЕТОДЫ, ПРИМЕНЯЕМЫЕ ДЛЯ ОПТИМИЗАЦИИ ВЫЧИСЛЕНИЙ …………………………….… 11

**ГЛАВА 2 РЕАЛИЗАЦИЯ В ПАКЕТЕ MATHEMATICA НЕКОТОРЫХ ПРОСТРАНСТВЕННЫХ ЗАДАЧ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ ................................................................................................................** 14

2.1 ПРЕДСТАВЛЕНИЕ НАЧАЛЬНЫХ ДАННЫХ ……………………………………………………... 14

2.2 ПЕРЕСЕЧЕНИЕ ЛУЧА И ВЫПУКЛОГО ПОЛИЭДРА ...……………………………………………… 16

*2.2.1 Постановка задачи …………………………………………………………………………....* 16

*2.2.2 Подход к решению и реализация………………………………………………………….…* 16

2.3 СТОЛКНОВЕНИЕ ДВИЖУЩЕЙСЯ СФЕРЫ И ВЫПУКЛОГО ПОЛИЭДРА ……………………… 18

*2.3.1 Постановка задачи …………………………………………………………………………...* 18

*2.3.2 Подход к решению и реализация………………………………………………………….…* 18

2.4 ТЕСТ НА ПЕРЕСЕЧЕНИЕ ДВУХ СТАТИЧНЫХ ВЫПУКЛЫХ ПОЛИЭДРОВ .…………………… 21

*2.3.1 Постановка задачи …………………………………………………………………………...* 21

*2.3.2 Подход к решению и реализация………………………………………………………….…* 21

**ЗАКЛЮЧЕНИЕ ………………………………………………………………………………………………………………** 24

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ ………………………………………………………………….** 25

**ПРИЛОЖЕНИЕ А.** **КОД ПРОГРАММЫ** **…………………………………………………………………………** 26

**ВВЕДЕНИЕ**

При разработке компьютерных игр, систем автоматизированного проектирования, систем виртуальной реальности, в физических симуляциях и других областях деятельности, тесно связанных с компьютером, часто возникает потребность в моделировании движений объектов в двумерном или трехмерном пространствах. При этом требуется определять, пересекаются ли два или более объекта между собой, поскольку наличие или отсутствие столкновения в реалистичных моделях напрямую влияет на траекторию или параметры движения тел. При этом во многих моделях обнаружение столкновений проводится в реальном времени, что накладывает ограничения на время работы соответствующих алгоритмов. В этом заключается важность данной задачи. Помимо самого факта столкновения зачастую требуется, например, определить глубину взаимопроникновения объектов или найти положения объектов, при котором они лишь касаются друг друга. Зная те или иные данные, исследователь может обработать столкновение, в соответствие с законами физики или нужным ему способом.

В рассматриваемой области ставятся следующие основные задачи[1]:

• создание универсальных алгоритмов и подходов для прозрачной работы с произвольными объектами: выпуклыми и вогнутыми, деформируемыми и абсолютно твердыми, статическими и движущимися.

• повышение эффективности алгоритмов и получение максимально возможной производительности, что позволит моделировать больше объектов и увеличивать их сложность.

• повышение надежности и точности алгоритмов

В данной работе я изучу некоторые из задач компьютерной графики, связанные с обнаружением столкновений. Я рассмотрю их с точки зрения математики, в частности геометрии, и приведу реализацию некоторых из наиболее важных алгоритмов, оставив в стороне вопрос о физической обработке столкновений. Сформулирую цель и задачу своего исследования.

**Цель исследования** – изучить и реализовать основные алгоритмы обнаружения столкновений в компьютерной графике.

**Задачи исследования** – задать представление объектов, реализовать и пояснить основные алгоритмы в пакете Mathematica.

**ГЛАВА 1**

**ОСНОВНЫЕ ЗАДАЧИ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ В КОМПЬЮТЕРНОЙ ГРАФИКЕ**

**1.1 Историческая справка о предмете и ключевые результаты**

В книге[2] приводится объемная историческая справка по предмету моего исследования. Самые первые применения задач пространственного обнаружения столкновений встречаются в работах посвященных робототехнике и автоматизации[3,4] за 1979 и 1985 годы. Здесь роботизированное оборудование для сборки или тестирования товаров предприятия моделировалось на компьютере. В них объекты, которые необходимо было проверить на пересечение, обычно представлялись в виде полиэдров. Поиск столкновений проводился затратным способом с учетом времени как четвертого измерения в модели[4].

Множество приемов используемых для обнаружения столкновений были заимствованы из задач 3D – визуализации. Так, в годы становления компьютерной графики, актуальность имела необходимость рендеринга (отрисовки) трехмерных объектов. Трудность заключалась в том, что требовалось определить и отрисовать лишь видимые наблюдателю части сцены из данного положения. Оказалось, что данная задача имела много общего с задачами обнаружения столкновений, что послужило толчком к появлению алгоритмов, находящим применение в обеих областях.

Почти в это же время возникает новая ветвь исследований, которая позднее получит название вычислительная геометрия. Исследования были посвящены представлениям структур данных и таким задачам, как построение выпуклой оболочки, определение пересечения, нахождение расстояния между объектами и пр.

Затем в 1984 году выходит компьютерная игра Elite, в которой игровые объекты изображены полиэдрами, однако для обнаружения столкновений и определения расстояния между ними использовались более простые геометрии, такие как сферы и параллелепипеды. Практика использования упрощенных моделей была весьма популярна в то время, поскольку вычисление точного пересечения было слишком трудным для современных вычислительных машин. Ранние попытки решения, в основном, не были направлены на определение столкновений в реальном времени. Было замечено, что данная задача является достаточно трудоемкой с точки зрения вычислений и необходимы новые подходы для ее оптимизации.

Алгоритмы обнаружения столкновений вскоре находят применение в физических симуляциях. Так, в 1989 г. в исследовании[5]  Д. Барафф в применение к своим моделям предлагает улучшение существующих алгоритмов, опираясь на свойство о том, что между соседними кадрами положение тел значительно не изменяется.

Со временем вычислительные мощности компьютеров стремительно увеличивались и в 1995 году выходит первая публичная библиотека столкновений I-COLLIDE для систем с большим числом объектов на сцене. Создатели, используя наработки Д. Бараффа смогли достичь высокой производительности. На сцене присутствовало более 1000 двигающихся полиэдров, каждый состоящий из более, чем 50 граней. Причем алгоритм работал для произвольно заданных траекторий, которые могли меняться во времени. Системе требовалось менее 1/20 секунды для определения всех столкновений[6].

**1.2 Основные понятия, связанные с предметом, и пояснения**

**1.2.1 Обнаружение и обработка коллизий**

*Коллизия (столкновение)* – это состояние при котором два или более объекта занимают одну и ту же точку пространства. *Обнаружение столкновений (collision detection)* - это алгоритм, позволяющий определять коллизии в той или иной модели. Положения объектов могу изменяться во времени, тогда они называются *движущимися*. Обнаружение столкновений – это базовая задача, производительность и точность при решении которой имеют первостепенное значение в множестве программ.

Компьютерные модели симулируют движение объектов постоянным обновлением положения объектов через фиксированные (дискретные) промежутки времени. Человеческий глаз видит передвижение предметов гладким, если их положение обновляется с частотой порядка 60 кадров в секунду. Если модель представляет из себя физическую симуляцию или анимацию в реальном времени, то все объекты сцены должны проходить проверку на столкновение с этой частотой. При этом модель может состоять из большого числа двигающихся объектов со сложной геометрией, например, многогранников. Для обработки такого числа объектов необходимо применять различные оптимизационные методы, о которых пойдет речь в следующих главах.

В случае, если коллизия между двумя или более объектами была обнаружена, то, поскольку в большинстве моделей не допускается взаимопроникновение объектов, то для следующего шага модели их необходимо “развести” так, чтобы они либо продолжали касаться друг друга, либо продолжали движение без касания. Эта процедура называется *обработкой столкновений (collision handling).*

Обработка столкновений проводится в зависимости от таких параметров, как скорость, масса, точка касания и пр. Впрочем, реакция на столкновение зависит от требования модели и конкретного моделируемого процесса и не обязательно основана на законах физики.

**1.2.2 Апостериорный и априорный подходы**

Существует два общеизвестных подхода при определении / обработке столкновений в физических симуляциях.

*Апостериорный (a posteriori)* подразумевает статичность объектов на данном шаге модели. Затем происходит проверка, есть ли на сцене пересекающиеся объекты. Если такие объекты обнаружены, то исходя из их предыдущего положения, их параметры пересчитываются и передаются на следующий шаг. Название объясняется тем, что столкновение определяется уже после того, как оно произошло. Так если в действительности момент столкновения произошел между предыдущим и текущим состояниями модели.

*Априорный (a priori)* подход предполагает создание такого алгоритма обнаружения столкновений, который будет способен предсказывать момент столкновения и положение объектов в этот момент времени по имеющимся траекториям движений тел. Столкновения определяются этой моделью так, что физические тела в сущности никогда не оказываются в состоянии взаимного проникновения. Этот подход называется априорным, поскольку моменты столкновений тел определяются до перехода к следующему шагу модели.

Основными преимуществами апостериорного подхода являются простота его реализации и быстродействие. Определить, пересекаются ли два статических объекта значительно проще, чем по уравнениям движения тел определить столкнутся ли они и будет ли момент столкновения раньше, чем следующий шаг модели. С другой стороны, апостериорный подход имеет два существенных недостатка, которые могут сделать данный подход неприемлимым в реалистичных моделях, а именно неточность и ненадежность. Из-за того, что момент столкновения и в частности, точка касания, не могут быть точно определены, возникают трудности с обработкой этого столкновения с точки зрения физики или иных правил заложенных в модели. Второй недостаток апостериорного подхода заключается в том, что если шаг модели выбран слишком большим, или скорость передвижения объекта слишком высока, то алгоритм может вовсе не обнаружить столкновения т.к. объекты могут пройти сквозь друг друга между предыдущим и текущим шагом модели.

Преимуществами априорного подхода являются точность и стабильность модели. Однако для определения столкновений требуется искать решения уравнений, в которых время будет выступать дополнительной переменной. Для оптимизации вычислений может применяется численные методы поиска корней. Трудоемкость этого подхода значительно выше, чем у апостериорного, и поэтому в некоторых ситуациях выгоднее применять последний.

**1.3 Обзор некоторых задач и алгоритмов их решения**

Очень часто для проверки на столкновения объектов в модели, используют упрощенную геометрию, называемую коллижн-моделью. Часто такой геометрией является сфера либо прямоугольный параллелепипед или куб. В самом простом случае, если два объекта имеют сферу в качестве коллижн-модели, факт пересечения определяется следующим образом: вычисляется расстояние между центрами сфер и если оно не превышает сумму радиусов этих сфер, то такие объекты пересекаются. Определение факта пересечения двух прямоугольных параллелепипедов (box) также является очень важной задачей в рассматриваемой области. Существует более простой случай этой задачи, когда стороны обоих боксов параллельны координатным осям. Такие боксы имеют название AABB (axis-aligned bounding boxes). Они очень широко используются в целях оптимизации вычислений, о чем будет рассказано в следующем пункте 1.4.

Рассмотрим данную задачу для двух объектов, зафиксированных в некоторый момент времени и приведем алгоритм ее решения[2]. Пусть даны два прямоугольных параллелепипеда с известными координатами центра и вершин. Необходимо построить разделяющую плоскость (такую плоскость, что параллелепипеды будут лежать по разные стороны от нее), что будет соответствовать тому, что пересечения нет. Или показать, что такой плоскости не существует, что означает, что тела пересекаются и коллизия обнаружена.

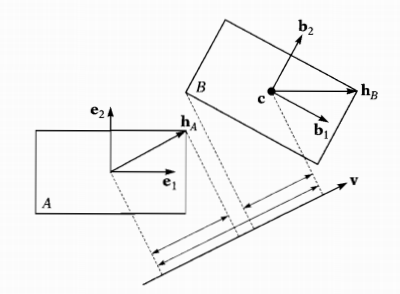


Рисунок. 1. Проверка на пересечение двух боксов. Источник – [2]

Идея алгоритма заключается в следующем: первый и второй параллелепипеды временно переносятся к началу отсчета, так, чтобы центр первого из них лежал в начале координат. Тогда координата центра второго будет с. Далее, используя значения координат вершин необходимо построить два базиса A и B, пустив векторы вдоль соответствующих ребер. Полученный базис будет, очевидно, ортогональным и данное свойство используется следующим образом: если два базиса ортогональны, то результат скалярного произведения двух произвольных векторов не зависит от того, в каком из двух базисов они заданы, так если для вычисления скалярного произведения можно использовать любой из этих двух базисов. Пустим векторы ha , hb относительно соответствующих им базисов к вершинам определенным направлениями базисных векторов, как показано на рис. 1.

Пусть вектор v задан в базисе A, тогда в базисе B он будет иметь вид BTv. Следовательно, разделяющая плоскость будет существовать тогда и только тогда, когда выполняется неравенство

Рассмотрим теперь случай, когда один объект представлен боксом, а другой – сферой. Как известно, сфера пересекает точку, если и только если расстояние от нее до центра сферы не превосходит радиус.

Идея алгоритма[2] заключается в нахождении ближайшей к центру сферы точки параллелепипеда. Параллельным переносом преобразуем координаты центра сферы и параллелепипеда так, чтобы центр последнего находился в начале координат. Пусть точка c - центр сферы после переноса. Пустим векторы e1 , e2 , e3 в направлении вдоль ребер параллелепипеда, а вектор h пустим к вершине находящейся в направлении этих векторов (рис. 2).

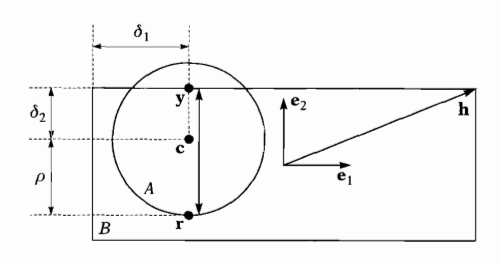


Рисунок. 2. Проверка на пересечение сферы и параллелепипеда. Источник – [2]

Пусть при этом (γ1, γ2, γ3) – координаты с, а (η1, η2, η3) – координаты h. Тогда искомая точка x может быть задана следующим образом:

x = (clamp(γ1, - η1, η1), clamp(γ2, - η2, η2), clamp(γ3, - η3, η3)), где

В случае, если c находится внутри бокса, то точка x совпадает с c. Затем, очевидно производится проверка, принадлежит ли точка x сфере или нет. В данной задаче также довольно легко найти величину, называемой глубиной взаимопроникновения, а именно минимальное расстояние, на которое необходимо развести объекты для того, чтобы они лишь касались друг друга.

В случае, если объекты, участвующие в столкновении имеют сложную геометрию в виде произвольных выпуклых полиэдров, используются более сложные алгоритмы. В главе 2 я представлю решение некоторых из таких задач, а именно предложу свой собственный алгоритм определения пересечение сферы и выпуклого полиэдра с априорным подходом к решению и апостериорное пересечение двух выпуклых полиэдров посредством алгоритма Гилберта-Джонсона-Кирти.

В случае, если в модели участвуют объекты с геометрией в виде невыпуклых полиэдров, то одним из основных подходов может быть разбиение этих оъектов на несколько выпуклых (что всегда возможно) и затем проверять пересечение с каждым из полученных компонентов в отдельности в соответствие со способами, указанными выше.

**1.4 Методы, применяемые для оптимизации вычислений**

В моделях, использующих большое количество объектов для оптимизации вычислений используются многчисленные подходы. Большинство объектов сцены, как правило, не пересекаются друг с другом, причем положение объектов на следующем шаге модели значительно не изменяется (frame coherence). Поэтому для того, чтобы модель была масштабируемой и могла обрабатывать столкновения в реальном времени, используются специализированные алгоритмы, по принципу работы классифицируемые по принадлежности к широкой или узкой фазе[1]:

Для широкой фазы подходят алгоритмы, которые позволяют с минимальными временными затратами отбраковывать наибольшее число пар, заведомо не пересекающихся. При этом преимущество отдается скорости работы алгоритма. Для этого вместо сложных исходных представлений объекта используют упрощенную геометрию, или ограничивающий объем. Ограничивающий объем – это область пространства, которая содержит данный объект. Для широкой фазы, как уже было отмечено ранее, используют простые по форме объекты, чаще всего сферы и прямоугольные параллелепипеды. Габариты ограничивающего объема должны быть такими, чтобы исходный объект целиком помещался в него, но при этом имел минимальный объем. К слову, построение минимально возможной объемлющей сферы за оптимальное время является важным алгоритмом в оптимизации.

Таким образом, простая попарная проверка друг с другом объектов с большим количеством элементов заменяется на проверку на пересечение между простыми объектами. Если оказалось, что два ограничивающих объема пересекаются, то далее производится точная проверка на пересечение, использующая исходную геометрию объекта без упрощений (т.е происходит передача пары объектов с широкой фазы на узкую).

Алгоритмы 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 быть настроен для возврата дополнительных сведений:

• глубину взаимопроникновения объектов, так если минимальное расстояние, на которое нужно переместить объекты, чтобы они лишь касались друг друга.

• направление, вдоль которой считается глубина взаимопроникновения.

• точное или численное значение объема пересечения.

Эта информация может использоваться для нахождения момента столкновения, для вычисления испульсов объектов для реалистичной симуляции их взаимодействия.

Алгоритмы узкой фазы могут быть разделены на две группы по характеру получения результата[1]:

• 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) и, следовательно, столкновение находится с погршностью, зависящей от размерности итогового пространства.

Соответственно, алгоритмы не имеющие свойств алгоритмов пространства изображений, относятся ко второй группе. Существует другая классификация алгоритмов обнаружения столкновений, касательно того, рассчитаны они на работу на CPU или на GPU. Обусловлено это тем, что потоковая модель вычислений GPU существенно отличается от модели вычисления на CPU, и, следовательно, ранее разработанные для CPU алгоритмы могут быть переработаны для получения выигрыша от большей вычислительной мощности GPU. Также для GPU разработано немало алгоритмов, не имеющих аналогов для CPU.

Как было указано выше, для оптимизации вычислений в широкой фазе используются ограничивающие объемы. Чаще всего для этой цели используется объемлющий прямоугольный параллелепипед, расположенный параллельно координатным осям, называемый AABB (axis-aligned bounding box).

Преимуществом использования AABB, очевидно, является возможность быстрой проверки двух боксов на предмет пересечения. В отличие от алгоритма, представленного в предыдущем пункте, боксы AABB всегда параллельны координатным осям, поэтому тест сводится лишь к последовательной проверке их координат вдоль трех (или двух в случае плоской задачи) координат. Недостатком AABB является то, что чаще всего бокс, параллельный осям не является наименьшим возможным объемлющим боксом. Более того, при повороте объекта бокс AABB обязан пересчитываться т.к. коллижн-модель этого объекта при повороте может изменится в высоте, ширине и длине вдоль координатных осей. Двух таких недостатков лишен другой вариант объемлющего бокса – OBB (oriented bounding box). Подход OBB не предполагает построение параллельного осям параллелепипеда, т.е. его положение никак не ограничено. Объем и размеры OBB оптимальны и меньше, чем у AABB. Однако с точки зрения алгоритма в широкой фазе проверка двух OBB на пересечение по трудоемкости аналогично задаче, рассмотренной в предыдущем пункте 1.3.

На основе структур AABB и OBB могут быть построены бинарные деревья для дополнительной оптимизации вычислений. Они строятся сверху вниз для всех объектов сцены при помощи рекурсии, на каждом шаге множество этих объектов разделяется некоторой гиперплоскостью на две части. На следующем шаге происходит то же самое для новых полупространств и так далее. Процесс разбиение продолжается до тех пор, пока каждый лист дерева будет содержать по одному объекту. Подробное описание построения и использование дерева можно встретить в работе [7].

Аналогичную идею использует подход двоичного (BSP – binary space partiotioning) или k-ричного (k-d tree) разбиения пространства сцены. Эти методы для уменьшения числа проверяемых объектов становятся очень эффективными при большом числе объектов, распределенных более-менее равномерно по сцене, поэтому лежат в основе многих алгоритмов широкой фазы.

**ГЛАВА 2**

**РЕАЛИЗАЦИЯ В ПАКЕТЕ MATHEMATICA НЕКОТОРЫХ ПРОСТРАНСТВЕННЫХ ЗАДАЧ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ**

**2.1 Представление начальных данных**

Существуют различные способы представления начальных данных и они зависят от постановки задачи. Например, могут использоваться сетки из треугольников, или конструктивная блочная геометрия, используемся в САПР, и пр.

В моем случае рассматриваемые объекты представлены в виде выпуклых полиэдров, причем грани могут, в общем случае, быть произвольными выпуклыми многоугольниками. Идею для задания геометрии полиэдра я взял из книги[8]. Выпуклый полиэдр P я определяю списком P ={V, G}, где V – список вершин полиэдра, а G – список граней полиэдра. Вершина представляется списком своих координат. На G накладывается следующее ограничение: порядок обхода вершин такой, что первые три вершины грани, не лежащие на одной прямой образуют обход, положительный относительно внешней нормали.

Для оптимизации вычислений, целесообразно проводить предобработку полиэдров, вычисляя некоторые важные свойства, такие как список нормалей, количество вершин, граней, центр и радиус объемлющей сферы и пр. Для того, чтобы вычислять эти параметры при создании нового полиэдра и хранить их связанно с объектом я использовал встроенный инструмент DownValues. Конструктор объекта выпуклый полиэдр принимает на вход списки V и G. Затем вызывается функция внутри конструктора, проверяющая, выполняется ли сформулированное выше ограничение для G.

Проверка осуществляется следующим образом: по имеющемуся набору координат V мы можем найти точку, гарантированно лежащую внутри этого полиэдра. В силу того, что полиэдр выпуклый, его вершины могут выступать в качестве барицентрических координат, так если любая их линейная комбинация с коэффициентами от 0 до 1, сумма которых равна 1, будет внутренней точкой полиэдра. Вычислив арифметическое среднее по всем вершинам при помощи функции Mean получим внутреннюю точку. Далее необходимо, чтобы для каждой грани, данная точка имела отрицательное ориентированное расстояние до плоскости, построенной на этой грани. Если же для некоторой грани это расстояние оказалось положительным, то это означает, что нормаль не является внешней и ее необходимо умножить на -1, а также поменять обход вершин. Для этого просто нужно развернуть наоборот список, представляющий номера вершин соотв. грани. Проделав данную процедуру для всех граней, получим полиэдр с положительным относительно внешних нормалей обходом вершин. На каждом этапе сохраняем вычисленную нормаль. Для удобства вычисления расстояний в последующем, каждую из них нормируем n = n / | n |. Вычислим и прочие полезные параметры полиэдра, которые описаны выше.

**2.2 Пересечение луча и выпуклого полиэдра.**

Данная задача имеет важное значение для компьютерной графики, и как я уже отмечал в первой главе, имеет много общего с задачей обнаружения столкновений. Техника бросания лучей (ray casting) является методом рендеринга, при котором сцена строится на основе замеров пересечения лучей с визуализируемой поверхностью. Пересечение луча с объектами сцены позволяет определить, какие части объекта видны наблюдателю, а какие – нет.

**2.2.1 Постановка задачи**

В пространстве дана точка p0, из которой пускается бесконечный луч в направлении вектора dir. Требуется определить точки пересечения этого луча с выпуклым полиэдром P.

**2.2.2 Подход к решению и реализация**

За основу я взял один из алгоритмов, предложенных в работе[9]. Здесь рассматривается схожая задача о пересечении прямой и выпуклого полиэдра. Выбрав прямую, содержащую луч, найдя точки пересечения и проверив, находятся ли они на луче, мы решим поставленную задачу. Алгоритм в узкой фазе заключается в следующем:

Точка, из которой пускается луч переносится к началу координат. Затем вычисляется угол, на который необходимо повернуть луч вдоль оси x и затем y и при помощи матриц поворота в пространстве вектор dir преобразуется к вектору, лежащему на оси z. Пусть матрицы поворота Mx и My. Те же самые преобразования проводятся и с полиэдром. Таким образом, ось z становится прямой, содержащей луч. Ее можно представить в виде пересечения двух плоскостей x = 0, y = 0. Разбиваем вершины полиэдра на две группы, лежащие по разные стороны от плоскости y = 0. Обозначим их V1 и V2. Теперь соединим все точки из V1 со всеми точками из V2 и найдем пересечение этих отрезков с плоскостью x = 0. Если одно из множеств оказалось пустым, то прямая, а следовательно, и луч не пересекают полиэдр. Теперь наша задача сведена к плоскости x = 0. Снова разделим все полученные точки пересечения на две группы V3 и V4, лежащие по разные стороны от плоскости x = 0, или для которых x < 0 и x ≥ 0 соответственно. Если одно из множеств оказалось пустым, то прямая и луч не пересекают полиэдр. Соединим все точки из V3 со всеми точками из V4 и найдем точки пересечения с осью z по формуле:

для всех точек x3i, z3i из V3 и x4j, z4j из V4. Теперь задача сведена к одномерной. Определим точки zmin , zmax из Z с наименьшей и наибольшей координатой. Тогда точки p1 = (0, 0, zmin) и p2 = (0, 0, zmax) будут точками пересечения прямой с выпуклым полиэдром. Теперь выполним обратное преобразование координат для этих двух точек по формуле (i = 1,2):

Осталось определить, находятся ли эти точки на луче. Для этого посчитаем скалярное произведение векторов (ui – p0).dir и если значение неотрицательно, то значит точка ui – принадлежит лучу .

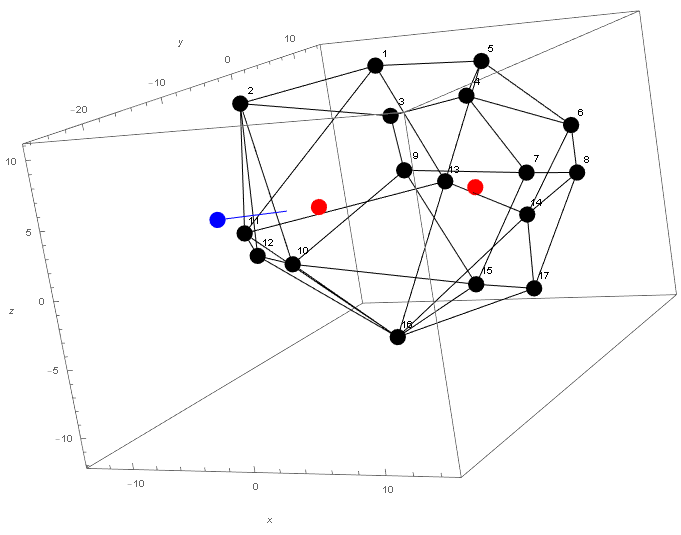


Рисунок 3. Бросание луча на выпуклый полиэдр.

**2.3 Столкновение движущейся сферы и выпуклого полиэдра**

В физических симуляциях и компьютерных играх часто требуется определить факт столкновения небольшого по размерам тела с объектом, имеющим сложную геометрию. Причем геометрия первого несущественна в связи с его малыми размерами. В таких случаях это тело можно аппроксимировать сферой. Также эта сфера в зависимости от того, какой объект она представляет, может двигаться с разной скоростью. Так, в точных моделях полета пули или иного быстро передвигающегося тела важно своевременно обнаружить столкновение. В случае апостериорного подхода, о котором шла речь в главе 1 для достижения точности результатов, шаг модели должен быть выбран достаточно малым, и даже это не может гарантировать абсолютную точность, поскольку как было отмечено, тело на слеующем шаге модели может пройти сквозь рассматриваемый в узкой фазе объект. Таким образом, вместо апостериорного я применил априорный подход к решению этой задачи и предлагаю свой алгоритм решения.

**2.3.1 Постановка задачи**

Дана сфера радиуса r и выпуклый полиэдр P. На предыдущем шаге модели центр сферы находился в точке p0prev, а на текущем шаге, в случае, если столкновения не было, окажется в точке p0cur. Траекторию движения сферы между состояниями модели считаем прямолинейной. Полиэдр P считаем статичным (в случае, если полиэдр также движется по некоторой заданной траектории, то с помощью сложения скоростей мы всегда можем свести такую задачу к данной). В случае, если столкновение было обнаружено, определить положение центра сферы на отрезке p0prev p0cur, при котором сфера касается полиэдра и передать его на текущее состояние модели, иначе передать p0cur.

**2.3.2 Подход к решению и реализация**

Идея алгоритма основана на том, что требуется проверить пересечение сферы только с теми плоскостями, которые либо пересекают отрезок p0prev p0cur, либо те, относительно которых точка p0cur находится с внешней стороны (ориентированное расстояние неотрицательно). Для этого мы вычисляем точки пересечения прямой, содержащей отрезок p0prev p0cur и всех плоскостей по формуле:

где ni – нормаль к i-ой грани, vi – ее носитель, а dir = p0cur - p0prev. Затем, по знакам скалярного произведения определяем, какие из точек p лежат на отрезке p0prev p0cur. Затем рассматриваем соответствующие грани Gj на предмет столкновения со сферой. Сфера может касаться либо грани полиэдра, либо ребра, либо вершины. Самой простой проверкой является касание первого типа. От точки pj пересечения j-ой плоскости с отрезком траектории в направлении –dir откладываем такой вектор, чтобы его проекция на вектор нормали этой плоскости nj по модулю была равна |nj r|, где r-радиус сферы (нормаль nj -единичная). Тогда другая проекция такого вектора укажет на точку плоскости j, в которой сфера ее касается. Эта точка может быть проверена на принадлежность к грани следующим образом: если ориентированное расстояние точки до всех остальных граней окажется меньшим либо равным нулю, то в этой точка сфера касается грани полиэдра и в таком случае соответствующее ей положение сферы добавляется в список для финальной проверки.

Следующим возможным случаем столкновения является касание сферы и вершины полиэдра. Задав движения сферы u(t) = p0prev + t\*dir и задав критерий столкновения: (u(t) - vk) • (u(t) - vk) = r2  мы можем получить в явном виде выражение для вычисления t. В решении будет присутствовать подкоренное выражение. В случае, если сфера не касается вершины, то подкоренное выражение станет отрицательным, а значения t – комплексными. Так если при вычислении t необходимо сначала вычислить подкоренное выражение, которое выступает в качестве критерия пересечения вершины и сферы. Из полученных значений t выбираем наименьшее неотрицательное tmin и заносим точку u(tmin) в список для финальной проверки, а если оба значения отрицательны, то переходим к рассмотрению следующей вершины.

Последним случаем возможного пересечения является касание ребра полиэдра. Эта задача равносильна задаче о пересечении прямой и цилиндра радиуса r на ребре. С помощью параллельного переноса и поворотов вокруг осей x и y можем привести цилиндр к началу координат и так, что его ось совпадает с осью z. Проведем те же преобразования с точкой p0prev и вектором dir. Сначала найдем пересечения этой прямой с бесконечным цилиндром. Эту задачу можно решить абсолютно аналогично предыдущему случаю. Критерием наличия пересечения будет служить неотрицательность подкоренного выражения. Получив точки пересечения, отбросим те, которые лежат лишь на продолжении цилиндра, но не на исходном. Вычислив подходящие значения параметра t и вернувшись к исходным координатам, получим точки пересечения сферы с ребрами рассматриваемых граней, которые заносим в список для проверки. Тот факт, что нас не интересуют точки пересечения прямой с основаниями цилиндра объясняется тем, что в таком случае, движущаяся сфера сначала коснется соответствующей вершины.

В финальном списке точек для проверки все найденные точки сортируются по квадрату расстояния до p0prev и наименьшая из них и есть требуемое значение, которое передается на текущий шаг модели. В случае, если список оказался пуст, то сфера не касается полиэдра и передается значение p0cur.

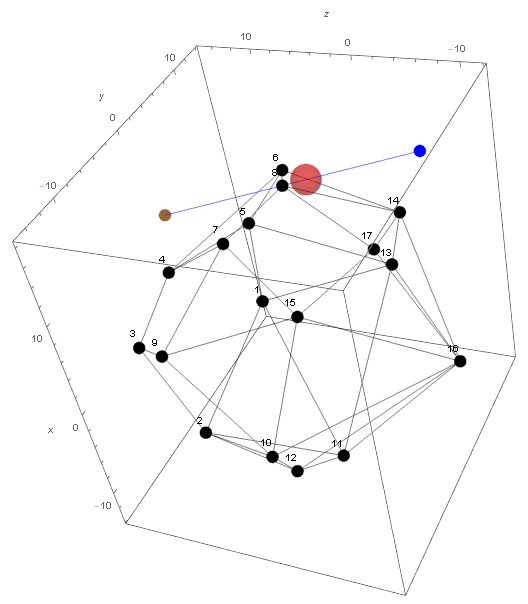


Рисунок 4. Априорное обнаружение столкновения сферы и вып. полиэдра.

**2.4 Тест на пересечение двух статичных выпулкых полиэдров**

Следующая задача требует алгоритма узкой фазы обнаружения столкновений двух объектов со сложной геометрией в виде выпуклых полиэдров. В виду трудоемкости данной задачи, используется апостериорный подход, так если два объекта считаются статическими в данный момент времени.

**2.4.1 Постановка задачи**

Даны два статических выпуклых полиэдра P1 и P2.

Определить, пересекаются ли они.

**2.4.1 Подход к решению и реализация**

Для решения данной задачи я использую классический вариант алгоритма Гильберта-Джонсона-Кирти[2]. Это итеративный алгоритм для определения пересечения выпуклых полиэдров. Его преимуществом является наглядность и простота в реализации, а также масштабируемость на выпуклые полиэдры с большим числом граней. Различные модификации этого алгоритма по-прежнему применяются в задачах обнаружения столкновения.

Алгоритм опирается на понятие суммы Минковского. Для объектов A и B сумма Минковского определяется как множество

Тогда разность A – B определяется как

Существуют теоремы, что если A и B – выпуклые множества, то множество A-B также будет выпуклым, причем для выпуклых полиэдров оно совпадает с множеством conv(Va – Vb), где Va  и Vb –вершины этих полиэдров. Эти свойства дают представление о том, что множество A – B можно построить простым вычитанием каждой вершины второго полиэдра из каждой вершины первого.

Пусть точка v определяется как точка некоторого множества, ближайшая к началу отсчета:

Для работы алгоритма также используется вспомогательная функция, называемая suport mapping. Для заданного вектора u она ставит в соответствие вектор (support point) из некоторого множества A, который имеет с данным наибольшее значение скалярного произведения.

Также определяется понятие симплекса: симплекс это полигональный объект, представленный в виде выпуклой оболочки множества точек и хранящийся как список этих точек. Соответственно, подсимплекс – это любое подмножество точек симплекса. Наименьшим подсиплексом далее называется подсимплекс, составленный из минимального числа точек. Для данной версии алгоритма симплекс либо представляет из себя треугольник, либо отрезок, либо точку. Если симплекс оказался тетраэдром (|W| = 4), то программа останавливается. В других версиях алгоритма, например, где определяется глубина взаимопроникновения, используются и тетраэдрические симплексы. Поиск минимального подсимплекса, а также точки v(conv(Y)) производится при помощи вспомогательного алгоритма Джонсона для вычисления расстояния (Johnson’s distance algorithm), использующий идеи и свойства аффинных оболочек[2, стр. 126].

В классическом виде алгоритм GJK имеет следующий вид[2] (псевдокод):

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11.

12. break;

13. }

14. return true

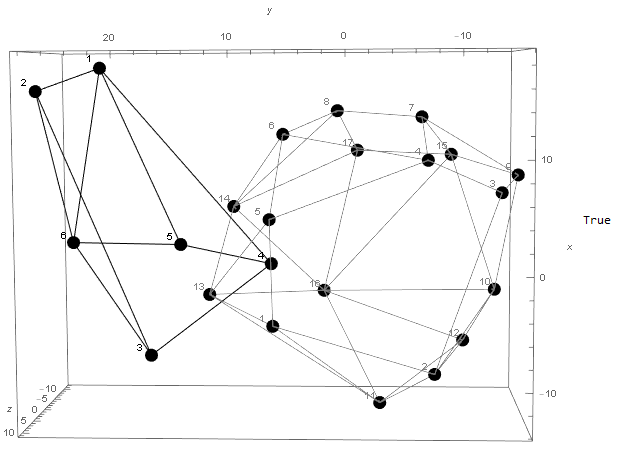


Рисунок 5. Тест на обнаружение столкновения двух вып. полиэдров

**ЗаКЛЮЧЕНИЕ**

С момента становления компьютерной графики и по сей день задачи обнаружения столкновения возникают во многих моделях и приложениях. Соответствующие алгоритмы начали разрабатываться в 80-е годы вместе с появлением потребности в решении новых на тот момент задач в области компьютерной графики. Как мы видим, эти алгоритмы весьма разнообразны и узко специализированны. Трудность задач обнаружения столкновений напрямую зависит от того, представлены объекты сложной или упрощенной геометрией, от того, в широкой или узкой фазе определяется пересечение, а также от того, применяется апостериорный или априорный подход. Для различных типов рассматриваемых объектов для оптимизации вычисление целесообразно применять отдельные алгоритмы к обнаружению столкновений. Применение одного и того же алгоритма в иных случаях может приводить к излишним вычислениям и даже к некорректному результату (например в 2.3, использование апостериорных алгоритмов для быстро передвигающихся объектов).

Скорость и точность являются ключевыми требованиями к этим алгоритмам. Первое необходимо для того, что многие модели воспроизводят симуляцию в реальном времени и поэтому для всех объектов на сцене должны быть обнаружены и обработаны столкновения порядка 60 раз в секунду. Точность необходима, очевидно, для реалистичности моделей. Со времен появляния первых алгоритмов исследователи в области компьютерной графики и разработчики компьютерных игр совершенствовали алгоритмы, повышая их эффективность. Так, некоторые из старых алгоритмов, например, GJK в различных модификациях и доработках используются и сейчас во многих подсистемах и программах. В 2003 и 2005 годах вышли основные книги[2,10] собравшие в себе накопленный опыт в области обнаружения столкновений.

В своей работе я прикоснулся к проблеме обнаружения столкновений в компьютерной графике и разобрался с основными ее задачами. Мной в пунктах 2.2, 2.4 были реализованы два алгоритма из работ[9,2] и приведены пояснения на русском языке. Также в пункте 2.3 я предложил собственный алгоритм для решения одной из популярных задач поиска коллизий с использование априорного подхода.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Алгоритмы обнаружения столкновений. Д. И. Собинов, В. В. Коробицын. Математические структуры и моделирование вып. 21 стр. 82-95, 2010 г.

2. Collision detection in interactive 3D environments. Gino van den Bergen 2003.

3. J. W. Boyse. Interference detection among solids and surfaces. Communication of the Acm, 22:3-9, 1979

4. S. Cameron. A study of the clash detection problem in robotics. In Proc. IEEE p. 488-493, 1985

5. Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies D.Baraff, 1989

6. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. D.Cohen, C. Lin, D. Manocha, K.Ponamgi

7. Efficient Collision Detection of Complex Deformable Models using AABB Trees. Gino van den Bergen. 1998

8. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики, 2003.

9. Line – Convex Polyhedron Intersection Using Vertex Connections Table. A. M. Konashkova, 2014

10. Real Time Collision Detection. Christer Ericson 2005.

**ПРИЛОЖЕНИЕ: КОД ПРОГРАММЫ**

* GJK

StaticCollisionQ3DConvPolyhedronConvPolyhedron[{P1\_, P2\_}] :=

Module[{v, w, W, Y, AB, iter = 1, ε = 2^(-53), εtol},

(\* GJK-алгоритм для построения разд. оси

(GJK separating- axis alg.) \*)

εtol = N[100 ε];

AB = Flatten[Outer[#1 - #2 &, P1["coord"], P2["coord"], 1], 1];

v = P1["coord", 1];

W = {};

Y = {};

While[True,

w = SupportVector[-v, AB];

If[MemberQ[Y, w] || v.w > 0,

(\* v - разделяющая ось \*)

Return[False];

];

Y = Append[W, w];

{v, W} = NextVertexAndMinSubsimplex[Y εtol];

If[(Length@W == 4) || (v.v <= εtol\*Max@MapThread[Dot, {W, W}]),

Break[]];

If[iter >= 20, Break[]];

iter = iter + 1;

];

Return[True];

]