## Моделирование футбола роботов лиги MiroSot

Синявский О.Ю.

Московский Энергетический Институт (ТУ), Москва, Россия, otognan@mail.ru

*В статье ставится задача разработки симулятора футбола мобильных роботов (лига MiroSot). Рассматриваются конкретные задачи моделирования для облегчения условий разработки управляющих роботами алгоритмов. Также в статье выводятся основные соотношения физики контакта твердых тел, определяется значение ударного импульса, приводится алгоритм расчета сил в контактах покоя. Описывается полный алгоритм роботов на такте расчета, который соответствует требованиям к моделированию игры в футбол роботов.*

## Введение

Роботы класса Mirosot предназначены для командной игры в робофутбол (международный термин Robosoccer) по правилам федерации FIRA [1]. Согласно этим правилам в робофутбол играют маленькие роботы (рис. 1), размерами не более 7.5 на 7.5 на 7.5 см, в командах по 5-11 участников. Для навигации роботов каждая команда использует собственную камеру, которую располагает на высоте 2.5 метра над полем, эта камера соединяется с компьютером, на котором происходит обработка видеоинформации, принятие командных решений, согласование действий роботов и связь с роботами по радиоканалу. Описание роботов, применявшихся российской командой по робофутболу в соревнования FIRA RoboWorld Cup 2006 можно найти в [2].

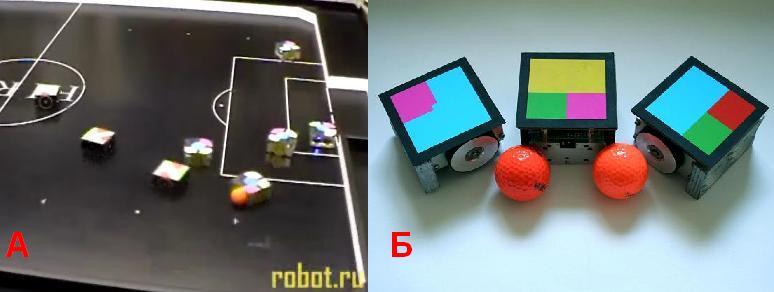


Рис. 1 (А) Кадр из финальной игры FIRA 2006 между командами Сингапура и Кореи. (Б) Роботы российской команды Moscow Pioneers

Комплект программного обеспечения, необходимого для работоспособности команды роботов состоит из модуля технического зрения (модуль распознавания), модуля управления и методов обработки данных, обеспечивающих передачу информации между этими модулями и объектами системы, такими как роботы и камера. Упрощенно модульная структура программного обеспечения изображена на рис. 2. Более подробно структура программного обеспечения описывается в работе [3]. Модуль распознавания обрабатывает информацию с камеры и передает информацию модулю управления в виде массивов координат роботов, мяча и информацию о границах поля. На основе этих данных модуль управления вычисляет желаемые скорости колес роботов, после чего они передаются по радиоканалу роботу, где происходит их обработка и выполнение роботом поставленной задачи – наиболее быстрое достижение и поддержание желаемых скоростей вращения колес.



**Рис. 2 Структура программного обеспечения робофутбола**

## Постановка задачи моделирования футбола роботов

Способность моделировать поведение роботов и мяча на поле является очень важным элементом успеха команды на соревнованиях по футболу роботов. Моделирование позволяет производить отладку управления роботов в пошаговом режиме, позволяет изучать сложные ситуации для алгоритмов управления, многократно обрабатывая такие ситуации, чего очень сложно добиться в реальности. В модели координаты и параметры объектов известны разработчику с любой точностью. Это позволяет отделить ошибки этапа распознавания от непосредственно ошибок алгоритмов управления. Появляется возможность моделировать различные погрешности этапа распознавания и этапа обработки управления, передаваемого из сервера, низкоуровневыми алгоритмами робота, искусственно внося шум в параметры моделируемых объектов, например, в координаты или в управляющее моменты на колесах роботов.

Моделирование призвано также решать и ряд других прикладных задач, возникающих в процессе игры и не связанных с отладкой алгоритмов управления. Между моментом времени получения видеокамерой текущего кадра с игрового поля и моментом, когда радиомодуль робота принял команды управления скоростями колес, существует задержка, связанная с аппаратными ограничениями, а также с вычислительной трудоемкостью алгоритмов распознавания. Очевидно, что задержка может сильно влиять на процесс игры, т.к. робот будет получать устаревшие команды управления, не соответствующие текущей игровой ситуации. С помощью корректного моделирования можно свести на нет негативные эффекты задержки. После получения координат роботов и мяча с модуля распознавания, модуль управления вырабатывает требуемые скорости колес роботов. Эти команды и текущие координаты передаются модулю моделирования, который предсказывает игровую ситуацию в следующий такт игры. Предсказанная игровая ситуация опять передается в модуль управления и все повторяется заново. Выполнив несколько тактов выработки управления и предсказания, можно избавиться от задержки, причем эффективность такого подхода напрямую зависит от физической точности и корректности алгоритмов моделирования.

Из-за наличия внешнего шума, а также непредвиденных ситуаций, например, когда между камерой и объектами на поле находится посторонний предмет, модуль распознавания иногда не может корректно определить параметры роботов, выдавая сигнал, что какой-либо объект не видим. В таких ситуациях положение невидимых и нераспознаваемых так же приходится моделировать для корректной выработки требуемого управления.

Требования к процессу моделирования несколько отличаются от обычно требующегося компромисса между точностью моделирования и быстродействием алгоритма моделирования. В данной задаче не требуется полного трехмерного моделирования, т.к. игра ведется в плоскости поля и выходит за рамки двух измерений в исключительных случаях, при которых игра обычно останавливается (например, переворот робота, вылет мяча за пределы поля). Количество тел фиксировано максимальным пределом в 22 робота и один мяч (в игре Large Mirosot League), а чаще игра ведется с 10 роботами (Middle Mirosot League). Из-за такого малого количества моделируемых тел практически не возникает проблемы быстродействия алгоритмов моделирования на современных компьютерах. Более того, наоборот возникает задача симуляции потребления вычислительного времени. Будет лучше, если симулятор будет потреблять некоторое значительное количество вычислительных ресурсов, ведь при реальной игре эти же ресурсы потребляет блок распознавания, тем самым заставляя разработчика оптимизировать управляющие алгоритмы. Используя возможность точного определения максимального времени работы блока распознавания, требуется лишь, чтобы симулятор не превосходил этот предел. Чтобы сделать время работы симуляции постоянной, следует добавлять в такты работы, ненагруженные сложными вычислениями, программные задержки. В связи с такими требованиями, в данной работе мы сделаем акцент на реализации алгоритма модели футбола роботов, обладающего наибольшей физической точностью и правдоподобностью, не уделяя большого внимания его быстродействию. Схема программного обеспечения в режиме моделирования представлена на рис. 3.



**Рис. 3 Симуляция структуры программного обеспечения**

## Моделирование объектов робофутбола

С точки зрения моделирования, игра в футбол осуществляется совокупностью твердых двухмерных тел, на которые действуют внешние силы и моменты и которые взаимодействуют только при столкновениях. Каждое тело геометрически протяженно (имеет конечную форму), имеет массу, момент инерции, может быть охарактеризовано двухмерным вектором координат центра масс и ориентацией (угол поворота).

Робот-футболист моделируется с помощью уравнений движения (1) неголономной механики в форме уравнений Аппеля [4]. Линия симметрии робота перпендикулярна линии, соединяющей центры колес. Обозначим точкой *A* пересечение этих линий. Центр масс робота ** расположен на линии симметрии, причем *AC = a*. Пусть  - радиус колес,  - момент инерции относительно центра масс,  - расстояние между центрами колес,  - моменты на колесах, развиваемые двигателями. Обобщенными координатами в уравнениях служат координаты робота, его угол поворота, величины линейной и угловой скорости. Управляющие воздействия задаются в виде моментов на соответствующих колесах робота.



где - единичный вектор ориентации робота.

Модель двигателей и низкоуровневые алгоритмы стабилизации скоростей колес, выполняющиеся внутри робота, будем моделировать упрощенно. Момент на колесе робота зависит от желаемой скорости, вырабатываемой алгоритмом управления, и от ее текущего значения:

,

где – желаемая линейная скорость колеса,  - текущая скорость вращения.

Дифференциальные уравнения движения мяча представлены в виде второго закона Ньютона для материальной точки (3). В процессе игры на мяч действуют только ударные силы взаимодействия, реакции соприкасающихся тел и сила трения, пропорциональная скорости мяча.



где - коэффициент трения мяча.

Стенки поля моделируются набором прямоугольников, имеющих бесконечную массу, которые соприкасаются сторонами.

## Моделирование столкновений

Основной сложностью является моделирование корректного столкновения между объектами. В реальной игре столкновения между роботами происходят достаточно часто, особенно если борьба за мяч ведется около стенок, углов поля или ворот. В данной работе рассматривается алгоритм, который пытается наиболее точно смоделировать удары между телами, прибегая к минимуму упрощений, обычно применяющихся для убыстрения его работы. Представленный алгоритм основан на практически полном отсутствии взаимопроникновения объектов. Взаимопроникновение разрешается только в исключительно редких случаях, рассмотренных далее, и обрабатывается отдельно.

Интегрирование уравнений движения тел ведется методом Эйлера с постоянным шагом . Рассмотрим отрезок времени , на котором произошло взаимопроникновение объектов в результате прямого интегрирования уравнений движения. Пусть в момент времени  объекты не пересекаются. В момент времени объекты уже проникли друг в друга. Это значит, что в этот момент времени  существует такая точка объекта 1, лежащая внутри объекта 2 и наоборот.

Для того чтобы избежать такого взаимопроникновения, нам нужен детектор столкновений[5]. Детектор столкновений – это алгоритм, который получает на входе текущие параметры состояния всех тел системы в момент времени и определяет, пересекутся ли какие либо тела в следующий момент времени при заданных уравнениях движения и внешних силах. Также детектор столкновений должен определить по возможности наиболее точные времена столкновения между отдельными телами, а также точку столкновения. В данной работе применялся алгоритм определения столкновений на основе разделяющих плоскостей. Два выпуклых многогранника не проникают друг в друга тогда и только тогда, когда между ними существует разделяющая плоскость. В двухмерном случае плоскость превращается в прямую. Чтобы проверить, является ли прямая разделяющей для двух многогранников A и B, достаточно чтобы все вершины A и B лежали на противоположных сторонах от этой прямой. В двухмерном случае, если многоугольники не пересекаются, разделяющая прямая содержит одну из сторон многоугольника, что упрощает ее поиск. Таким образом, если в момент времени t0 между двумя многогранниками существует разделяющая прямая, а в – не существует, то многогранники пересеклись на данном интервале . Применяя простой метод бисекции (деления отрезка пополам), можно вычислить время пересечения  с любой заданной точностью, т.к. положения фигур в отсутствии ударов являются гладкими функциями времени. Зная, какая из вершин проникнет в один многоугольник сразу после , а также зная, какая сторона второго многоугольника содержалась в разделяющей линии непосредственно до времени , мы получим точку столкновения и нормаль к поверхности, на которой произойдет удар. Причем точкой столкновения будет служить вершина одного выпуклого многогранника, а поверхностью столкновения будет сторона другого.



Рис. 4 Столкновение многоугольников (роботов)

Очевидно, что этим же методом можно определить столкновения круглого тела с многогранником (мяча с роботом). Для того, чтобы круг целиком лежал с одной стороны от разделяющей прямой, необходимо и достаточно, чтобы расстояние от центра круга до линии по нормали к ней было больше, чем радиус круга. Немного будет отличатся определение точки и поверхности столкновения: при пересечении круга со стороной многогранника точкой прикосновения будет служить точка касания круга к стороне, при пересечении круга с вершиной точкой будет вершина, а нормалью к поверхности – нормаль к окружности в этой точке.



Рис. 5 Варианты столкновения многоугольника и круга (робота и мяча)

Точку столкновения будем называть контактом. В симуляторе контакт представляется в виде структуры, содержащей ссылки на сталкивающиеся объекты A и В, точку столкновения, нормаль к поверхности столкновения, а также время , прошедшее от начала такта интегрирования до предполагаемого контакта. Для определенности будем предполагать, что точку содержит тело A, а поверхность, нормаль к которой была определена, – тело B, причем нормаль направлена от тела B к телу A. Детектор столкновений должен заполнять массив таких структур и передавать его блоку решения контактов. Основной величиной, определяющей дальнейший путь решения взаимодействия двух тел при контакте, служит проекция относительной скорости двух тел в точке контакта на нормаль к поверхности контакта.

Если скорость положительна, значит, тела расходятся после контакта, и такой контакт отбрасывается, т.к. не влияет на взаимное проникновение тел. Если относительная скорость отрицательна, то такой контакт называется ударом. При нулевой скорости контакт называется контактом покоя. В зависимости от типа контакта применяются различные алгоритмы их обработки. Более подробно алгоритмы обработки ударов, а также выводы уравнений теории взаимодействия твердых тел представлены в работе[6].

### Удар

Относительная скорость двух тел в точке контакта отрицательна, и если не изменить скорости этих тел, то произойдет взаимопроникновение. Во время удара на тела действуют значительные силы по сравнению с остальными внешними силами. Однако, удар происходит на очень малом промежутке времени. Чтобы охарактеризовать взаимодействие тел в таких условиях в теории удара вводят величину, называемую ударным импульсом:



Если время удара стремится к нулю, то при ударе импульс мгновенно изменяет линейные и угловые скорости тел:



где - радиус-вектор из центра масс тела в точку удара P, - скорость центра масс и угловая скорость тела А до и после удара.

Импульс – векторная величина. Его направление совпадает с нормалью контакта. Примем, что импульс положительно действует на тело A и отрицательно на тело B. Величину импульса вычислим на основе эмпирического закона столкновения:



,- проекции на нормаль к поверхности удара относительных скоростей тел в точке удара P соответственно после и до удара:



где -скорости точки контакта *P* после и до удара тел A и B соответственно.

Коэффициент  называется коэффициентом упругого восстановления. С помощью этого коэффициента можно регулировать упругость удара: от абсолютно не упругого (=0), когда тела слипаются в точке контакта при соударении, до абсолютно упругого удара (=1), когда не происходит потери кинетической энергии в результате удара.

Выясним, как изменятся скорости под воздействием импульса после удара, через величину импульса. Так как импульс действует по нормали , представим импульс в виде:



Найдем скалярную величину амплитуды *j.*Скорость точки P тела A после удара с применением формулы (5) равна:



Так как импульс действует отрицательно на тело B, то:



Подставим (10) в формулу (7) для относительной скорости тел после удара:



Заметим, что:



Решив систему из уравнений (6) и (11), относительно *j* , используя выражение (12), получим:



Заметим, что для того чтобы моделировать соударения с неподвижными стенками поля требуется положить массы и моменты инерции образующих их тел бесконечности (или нулю величины, обратно пропорциональные их массам и моментам инерции).

### 4.2. Контакт покоя

Относительная скорость двух тел в точке контакта равна нулю. Взаимопроникновения произойти не должно, однако возможно на тела действуют внешние силы, способные привести к взаимопроникновению в дальнейшем. Если такие силы существуют, то чтобы этого не произошло, должны возникнуть силы взаимодействия, препятствующие росту относительной скорости в отрицательном направлении. Будем рассматривать контакты покоя при отсутствии трения в контакте, поэтому силы будут направлены по нормали к поверхности. Так же как и в случае с импульсом, представим силу в контакте покоя в виде:



где - скалярная величина силы, - единичный вектор нормали к поверхности контакта.

Так же как и для удара, положим, что сила в контакте покоя действует положительно на тело A и отрицательно на тело B, т.е. нормаль направлена от B к A. Силы в контактах покоя должны удовлетворять трем условиям. Во-первых, ускорение тел в точке контакта должно не отрицательно, чтобы предотвратить увеличение относительной скорости, приводящей к взаимопроникновению. Для этого введем величину :



где  - абсолютные радиус-вектора до точек контакта тел A и B.Заметим, что, если тела контактируют в момент *t* (точки контакта совпадают),  - тела разделены, - произошло взаимопроникновение.

Во-вторых, сила контакта покоя должна исчезнуть, как только тела начнут расходиться. В-третьих, эта сила должна быть положительна, т.е. не приводить к взаимному притяжению тел в точке контакта. Данные условия можно выразить следующими формулами:



где - ускорение сближения тел в *i*-той точке контакта покоя во время контакта . Заметим, что если , то второе условие должно выполняется за счет и наоборот, обеспечивая тем самым, что сила контакта покоя должна исчезнуть, как только тела начнут расходиться.

Рассмотрим более подробно, каким образом будет выполняться первое условие. Пускай в момент времени  на тела находятся в нескольких контактах покоя друг с другом, причем для каждой точки контакта должны выполняться все три условия. Относительное ускорение в точке контакта можно представить как функцию внешних сил, действующих на тела (коэффициент *b*) и сил в контактах покоя (коэффициенты *a*):



Найдем коэффициенты этого уравнения через параметры контакта и контактирующих тел. Ускорение сближения тел в точке контакта можно представить в виде:



где  - скорости и ускорения точек контакта. В (18) также применялось условие .

Абсолютные ускорения точек тел в точке контакта для двумерного тела равны:



где - радиус вектор из центра масс тела до точки контакта.

Чтобы найти коэффициенты в (17), следует заметить, что из второго закона ньютона от в выражениях (18) и (19) зависят только  и . Следовательно:



Рассмотрим, как вычислить свободный коэффициент  используя (18), (19) и второй закон Ньютона для внешних сил и моментов:



где - внешние силы и моменты, действующие на тело. Нормаль принадлежит телу B, следовательно .

Вычислив коэффициенты в выражении (17) мы получим систему уравнений и неравенств, решаемую с помощью так называемого метода квадратичного программирования. По сути задача состоит в нахождении вектора , удовлетворяющего следующим условиям:



где A и b – матрица и вектор, составленные из коэффициентов, найденных в (20) и (21).

В данной работе эта задача решается с помощью алгоритма всплытия решения Данцига. Опишем кратко работу данного алгоритма. Более подробно постановка задачи и алгоритмы решения представлены в работе [7]. В начале работы алгоритма все контактные силы, кроме первой игнорируются и приравниваются к нулю. Алгоритм ищет значение , удовлетворяющее условиям силы в контакте покоя (16). Далее алгоритм вычисляет значение  с условиями (16), поддерживая выполнение этих условий для уже найденного значения , возможно, изменяя его. Далее алгоритм действует похожим образом, пока не будут найдены все силы, удовлетворяющие этим условиям. Поиск ведется таким образом, чтобы относительное ускорение в точке контакта, в которой действуют силы, было не отрицательно. Если при текущей силе  ускорение  меньше нуля, то очевидно, что следует увеличить силу в данной точке, если это нарушит неотрицательность ускорений в других точках, следует изменить силы в этих точках похожим образом. Чтобы вычислить величину, на которую следует увеличивать силу , не испортив выполнение условий (14) в других точках, следует ввести два множества *C* и *NC* на множестве точек контактов покоя. В первом множестве *C* (constrained – ограниченный) будут находиться индексы точек, в которых . В множестве *NC* (non constrained – не ограниченный) – индексы точек, в которых . Мы не должны менять , если эта точка принадлежит множеству *NC*, т.к. для выполнения условия ,  должно поддерживаться равным нулю. Если же точка принадлежит множеству *С*, изменение  не должно нарушить того факта, что . Приращения  в зависимости от приращений  могут быть из следующего выражения:



Следовательно, чтобы найти приращения в точках принадлежащих *C*, требуется решать систему линейных уравнений порядка, равного мощности множества *C*. Если точка не принадлежит не одному из множеств, что означает, что , положим  равным единице. Во время приращения *f* может возникнуть ситуация, при которой  станет равным или большим нуля. В этом случае требуется изменить множества *C* и *NC*, чтобы они соответствовали текущей ситуации. Изменение  ведется на величину , где *s*>0 – скаляр, определяющийся из условия обнуления отрицательного ускорения в текущей точке или перемещения элементов из множеств *C* и *NC*. Код на языке С++ представлен в Приложении .

В работе [7] доказано, что алгоритм Данцига всегда сходится при отсутствии трения в контактах покоя. После нахождения сил в контактах, следует прибавить их в уравнения движения тела на данном такте интегрирования:



Таким образом, мы располагаем всеми инструментами для корректной обработки столкновения тел в симуляции футбола роботов, почти полностью предотвращающих взаимопроникновение тел.

## Полный алгоритм симуляции

Рассмотрим полный алгоритм симуляции игры на такте интегрирования. На вход алгоритма поступают входные данные о координатах и скоростях роботов и мяча в момент времени , а также желаемые скорости на колесах каждого робота, вырабатываемые модулем управления. Задачей алгоритма является расчет координат и скоростей объектов с учетом столкновений и динамики объектов через время , где – известный шаг интегрирования, остающийся практически постоянным на всем протяжении симуляции. Входные данные поступают в модуль детектора столкновений, который создает массив структур контактов. Каждый контакт имеет ссылки на сталкивающиеся тела, описывает точку касания и нормаль к поверхности касания, а так же содержит значение времени , прошедшее от  до момента удара. При нахождении столкновений определяется контакт с минимальным временем до столкновения , а затем составляется массив контактов, минимально-близких по времени к  с некоторым порогом. Текущему частичному шагу интегрирования присваивается значение . Если массив контактов пуст, т.е. столкновений на данном шаге не найдено, частичным шагом для модуля интегрирования является весь временной отрезок . Управление передается модулю интегрирования, который изменяет координаты и скорости всех объектов в соответствии с их уравнениями движения на частичном шаге интегрирования. Запоминается уже количество проинтегрированного времени, если это время равно полному шагу , то вычисления на данном шаге считаются законченными и происходит выход из алгоритма. Если проинтегрирован промежуток времени, меньший , это значит, что на данном шаге существуют контакты, и массив минимальных контактов не пуст. Этот массив передается в модуль решения контактов. Модуль решения контактов в первую очередь определяет относительную скорость тел в точке контакта и сортирует контакты по типу. Он отбрасывает контакты с положительной больше некоторого порога скоростью, создает массив контактов покоя, в которых относительная скорость равная нулю с тем же порогом. Все остальные контакты, в которых относительная скорость отрицательна, являются ударами. Для каждого контакта из массива ударов вычисляется импульс удара по формуле (13), затем скорости объектов меняются в соответствии с формулами (5). Далее решается массив контактов покоя: находятся матрицы A и B по формулам (20),(21), решается задача квадратичного программирования по описанному алгоритму и находятся силы в контактах покоя, которые изменяют величины внешних сил, действующих на тело по формулам (24). Данные с уже измененными координатами и скоростями тел передаются снова в блок детектора столкновений. Таким образом, может произойти несколько итераций, прежде чем будут решены все контакты, происходящие на основном промежутке интегрирования . Блок-схема алгоритма представлена на рис. 6.



Рис. 6 Полный алгоритм моделирования робофутбола на такте интегрирования

## Выводы

Внешний вид симулятора футбола роботов с выведенной графической информацией для отладки алгоритмов управления представлен на рис. 7.

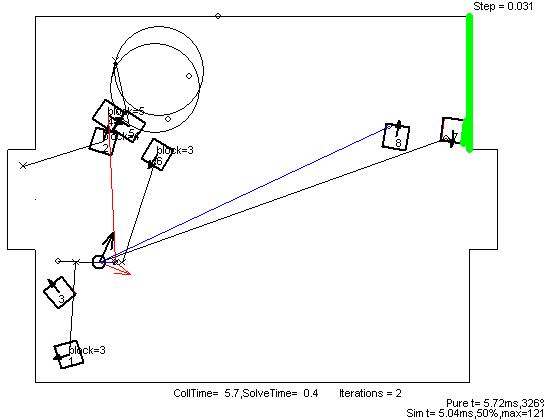


Рис. 7 Симулятор робофутбола

Достоинством алгоритма моделирования (рис. 6) является высокое физическое правдоподобие моделирования. Главным недостатком такого алгоритма, очевидно, является сильная зависимость количества итераций, а соответственно и времени выполнения от ситуации на поле. Наибольшую трудность составляют ситуации, когда, например, несколько роботов группируются около мяча, зажатого в угол. В таких случаях количество главных итераций алгоритма может составлять несколько десятков, однако, т.к. количество тел мало и практически постоянно, это не вызывает вычислительных проблем при моделировании. Заметим, что свойства алгоритма полностью соответствуют требованиям, представленным в разделе 2.

**Литература:**

1. **Официальный сайт организаторов турниров по футболу роботов http://www.fira.net**
2. **Victor Pakhomov, Evgeny Yelkin “Introducing an Another One MiroSot Robot Soccer System”, FIRA RoboWorld Congress 2006, Proceedings, 2006.**
3. **Елкин Е.В.,** «**Разработка методов технического зрения и обработки данных для комплекса программных средств системы роботов класса MiroSot», дипломная работа, http://www.robosoccer.ru**
4. **Мартыненко Ю.Г., «Динамика мобильных роботов», Соровский Образовательный Журнал, том 6, №5, 2000г, стр. 110-116.**
5. **Симонов С.В.** , **«Моделирование взаимодействий систем твердых тел», дипломная работа, http://vmumf.rtf.ustu.ru/files/simonov.pdf**
6. **D. Baraff. "An introduction to physically based modeling: Rigid body simulation II — Nonpenetration constraints." In Computer Graphics (Proc. SIGGRAPH), 1997.**
7. **D. Barraf “Fast contact force computation for Nonpenetrating Rigid Body“, ACM SIGGRAPH’94**

# Приложение

Рассмотрим код, реализующий описанный алгоритм Данцига. Основной процедурой, реализующей данной алгоритм, является ComputeForces,которая получает матрицы A и b, вычисленные с помощью формул (20),(21). Результат возвращается в массиве f, count – размерность системы. В начале инициализируются множества C и NC, все контактные силы f, приравниваются к нулю, а соответственно все ускорения равны b.

void ComputeForces( double\*\* A,double \*b,double \*f,int count )

{

double \*a = new double[count];

bool \*C = new bool[count]; bool \*NC = new bool[count];

for (int i = 0; i<count; i++)

{ f[i] = 0; a[i] = b[i]; C[i] = NC[i] = false; }

int d;

while (GetD(a,count,d))

{

DriveToZero(d,a,A,f,count,C,NC);

}

delete [] a; delete [] C;delete [] NC;

}

Функция GetD находит первое отрицательное (с вычислительным порогом) ускорение в массиве ускорений a. Если такое ускорение не найдено, происходит выход из главной процедуры ComputeForces.

bool GetD(double \*a, int count, int &d) {

for (int i = 0; i<count; i++)

{ if (a[i] < -DEF\_NULL) {d = i; return true;} }

return false;

}

Функция DriveToZero инициализирует массивы приращений da и df, затем в цикле изменяет силы f до тех пор, пока ускорение под номером d найденным GetD , не станет равным нулю.

void DriveToZero(int d,double \*a,double\*\* A,double \*f,int count,bool\* C,

bool \*NC)

{

double stepParam;int contraint\_index;

double \*df = new double[count]; double \*da = new double[count];

while (true) {

GetFDirection(d,C,A,df,count);

MultMatrixOnVector(A,df,da,count);

GetMaxStep(d,stepParam,contraint\_index,C,NC,f,a,df,da,count);

for (int i = 0; i<count; i++)

{ f[i] += stepParam\*df[i]; a[i] += stepParam\*da[i]; }

if (C[contraint\_index])

{ C[contraint\_index] = false; NC[contraint\_index] = true;} else if (NC[contraint\_index])

{ C[contraint\_index] = true; NC[contraint\_index] = false;}

else { C[contraint\_index] = true; break; }

}

delete [] df; delete [] da;

}

Функция GetFDirection вычисляет возможное df приращение сил контакта покоя. Сначала все приращения обнуляются, приращение силы с номером d ускорения, которое надо привести к нулю приравнивается к единице. Если номер контакта принадлежит множеству *С* (точка контакта ограничена, ускорение в ней равно нулю), то изменение df[i] не должно нарушить того факта, что a[i] = 0. Из формулы (23) мы должны найти такие приращения сил, что приращения ускорений будут равны нулю для элементов множества C, причем приращение силы для элемента d равно единице. Для этого создается матрица Acc, в которой находятся только те строки и столбцы, номера которых принадлежат множеству *С*, а также вектор v1, который представляет собой столбец матрицы *А* с элементами, номера строк которых принадлежат множеству *С*. Приращения искомых ускорений x умножаются на матрицу Acc, а единичное приращение df[d] умножается на столбец v1, причем их сумма должна равняться нулю, что соответствует системе линейных уравнений размерности, равной мощности множества *С*. В функции GetFDirection эта система решается с помощью простого LU-разложения.

void GetFDirection(int d, bool \*C,double\*\* A,double \*df,int count) {

for (int i = 0; i<count; i++) { df[i] = 0; }

df[d] = 1;

int sizeOfC = 0;

for (int i = 0; i<count; i++) { if (C[i]) sizeOfC++; }

if (sizeOfC == 0) { return; }

double \*\*Acc = new double\*[sizeOfC]; double \*v1 = new double[sizeOfC];

for (int i = 0; i<sizeOfC; i++) { Acc[i] = new double[sizeOfC]; }

int rowAcc =0, rowV = 0, colAcc;

for (int i = 0; i<count; i++)

{

if (C[i])

{ colAcc = 0;

for (int j = 0; j < count ; j++)

{ if (C[j]) { Acc[rowAcc][colAcc] = A[i][j]; colAcc++; } }

rowAcc++;

v1[rowV] = -A[i][d]; rowV++;

}

}

double \*x = new double[sizeOfC];

SolveWithLU(Acc,v1,x,sizeOfC));

rowV = 0;

for (int i = 0; i<count; i++)

{ if (C[i]) { df[i] = x[rowV]; rowV++; } }

delete [] x; delete [] v1;

for (int i = 0; i<sizeOfC; i++) { delete [] Acc[i]; }

delete [] Acc;

}

Обратимся снова к функции DriveToZero.Получив возможное изменение сил с помощью функции GetFDirection, мы получаем соответствующее ему приращение ускорений da с помощью функции MultMatrixOnVector умножения вектора на матрицу в соответствии с уравнением (23):

void MultMatrixOnVector(double\*\* Matrix,double\* Vector,double\* ansV,

int count)

{

for (int i = 0; i<count; i++)

{ ansV[i] = 0;

for (int j = 0; j < count ; j++) ansV[i] += Matrix[i][j]\*Vector[j];

}

}

Далее в функции DriveToZero используется функция GetMaxStep, которая находит скаляр stepParam, на который можно умножить приращения сил и ускорений, чтобы корректно обработать переход точек из множеств *С* и *NC*, тем самым не допуская нарушения условий (16). Также данная функция находит индекс ограничивающего фактора, не позволяющий больше увеличивать stepParam.

Вначале stepParam скаляр приравнивается к бесконечности. Потом определяется скаляр stepParam, требуемый для приведения ускорения под номером к нулю. Далее проверяются множества *С* и *NC* и находятся ограничения скаляра stepParam,приводящие к перемещению точек из множеств *С* и *NC*.

void GetMaxStep(int d, double &stepParam, int &contraint\_index, bool\* C,

bool \*NC,double \*f,double \*a,double \*df,double \*da,int count)

{

stepParam = DEF\_INFINITE; contraint\_index = -1;

if (da[d] >DEF\_NULL)

{ contraint\_index = d; stepParam = -a[d]/da[d]; }

double curS;

for (int i = 0; i<count; i++)

{

if (C[i] && (df[i]<-DEF\_NULL))

{ curS = -f[i]/df[i];

if (curS<stepParam) { stepParam = curS; contraint\_index = i; }

} else if (NC[i] && (da[i]<-DEF\_NULL))

{ curS = -a[i]/da[i];

if (curS<stepParam) { stepParam = curS; contraint\_index = i; }

}

}

}

Далее в функция DriveToZero изменяет ускорения и силы с учетом найденным приращений и скаляра stepParam. Если огранивающий индекс contraint\_index контакта принадлежит одному из множеств *С* и *NC*, то он перемещается между этими множествами. Если же ограничивающий индекс не принадлежит ни одному из множеств, значит он равен d, а это в свою очередь значит, что задача по приведению a[d] к нулю выполнена.

В итоге ComputeForces изменит все силы так, что в массиве не останется ни одного отрицательного ускорения, причем силы для положительных ускорений больших нуля, контактные силы будут равны нулю. Таким образом, будут выполняться условия (16).