Продвинутая физика персонажей

Автор: Thomas Jakobsen*(January 21, 2003)*
Перевод: Ткачёв Антон (*December 2,2003****)(misterio_anton@mail.ru)***

Оригинальная статья может быть найдена на
http://www.gamasutra.com/resource_guide/20030121/jacobson_01.shtml

Эта статья объясняет основные элементы метода физически-основанного моделирования, который хорошо подходит для интерактивного использования. Он прост, быстр и стабилен, в своей упрошённой версии метод не требует знаний по высшей математике (хотя он основан на математическом подходе). Он позволяет симулировать трепещущую ткань, мягкие и твёрдые тела, и даже соединённые или ограниченные тела с использованием прямой и обратной кинематики.

Алгоритмы были разработаны для игры «Hitman: Codename47». Там, помимо других вещей, физическая система была ответственна за движение одежды, растений, твёрдых тел, и для создания падения мёртвых человеческих тел, основанное на том, куда попали в человека, и на полном взаимодействие со средой (в результате получим «анимацию смерти как в жизни»). Статья также рассматривает такие тонкости как тест проникновения (penetration test), оптимизацию и обработку трения.

Использование физически-основанного моделирования для создания хорошо выглядящей анимации рассматривалось некоторое время и большинство существующих технологий довольно сложно. Различные подходы были предложены в литературе [Baraff, Mirtich, Witkin, и других] и большинство усилии было потрачено на построение алгоритмов, которые точны и реалистичны. Фактически, точные методы симуляции физики и динамики были известны долгое время в технике. Однако, для игр и интерактивного использования, точность не главная задача (хотя, это конечно всегда приятно) – здесь главные задачи – правдоподобность (программист может жульничать столько, сколько хочет, если игрок всё ещё впечатлён) и скорость выполнения (только небольшая часть времени за кадр будет выделена под физику движка). В случае симуляции физики правдоподобность мира скрывается за стабильностью, алгоритм плох, если кажется, что объекты проходят через препятствия или вибрируют, когда должны лежать спокойно, или если части одежды взмываю вверх.

Методы представленные в этой статье были созданы, в попытке достичь этих целей. Алгоритмы были разработаны и осуществлены автором, для использования в компьютерной игре *«Hitman: Codename 47»* от IO Interactive, и все были объединены во внутреннем игровом движке «Glacier» от IO. Доказано, что методы достаточно просты для осуществления (по крайней мере, в сравнении с остальными), и имеют высокую работоспособность.

Алгоритм является зацикленным с некоторого момента, но может быть остановлен в любое время. Это даёт нам приемлемый компромисс между временем и точностью. Если приемлема небольшая погрешность, то код позволяет работать быстрее, эта погрешность может быть просчитана даже в реальном времени. В некоторых случаях, метод работает быстрее, чем другие существующие. Он также одновременно обрабатывает столкновение и покой в одной структуре, и хорошо обрабатывает застрявшие ящики и другие ситуации, которые напрягают физический движок.

Если кратко, то успех метода идёт от правильной комбинации нескольких техник, которые берут лучшее друг от друга:

- Так называемое интегрирование Верлета.
- Обработка столкновений и проникновений проецированием.
- Простое решение реакций связи при использовании расслабления.
- Хорошее приближение квадратного корня, что даёт повышение скорости.
- Моделирование твёрдых тел, как частиц со связями.
- Оптимизированный движок столкновений со способность вычисления глубины проникновения.

Каждая из описанных тем, будет объяснена кратко. При написании этого документа, автор пытался сделать его доступным для широкой аудитории без потери жизненно важной информации для осуществления. Это значит, что технические математические объяснения и понятия сведены к минимуму там, где они не влияют на понимание предмета. Цель – демонстрация возможности легкого осуществления продвинутой и стабильной физической симуляции без влезания в математическую путаницу.

Содержание организовано следующим образом. Первое, в секции 2, "velocity-less" даётся представление об описании системы частиц. Это имеет несколько преимуществ: главное – стабильность, и тот факт, что связи легки в объяснении. Секция 3, описывает как обрабатываются столкновения. Затем, в секции 4, система частиц расширится связями, позволяющими моделировать одежду. Секция 5 объясняет, как применить связанную систему частиц для эмуляции твёрдых тел. Следующее, в секции 6, демонстрируется, как дальше расширить систему, чтобы позволить создавать подвижные тела (это, системы взаимосвязанных твёрдых тел с угловыми и другими ограничениями). Секция 7 содержит различные заметки и описывает некоторый опыт в осуществлении трения и т.д. Наконец, в секции 8, краткое заключение.

Далее, жирным шрифтом обозначены векторы. Векторные компоненты пронумерованы при использовании нижнего индекса, например $\mathbf{x}=(x_1, x_2, x_3)$.

Интегрирование Верлета

Сердце симуляции – система частиц. Обычно, при создании системы частиц, каждая частица имеет две переменные - это позиция **x** и скорость **v**. Тогда через некоторое время, новая позиция **x'** и скорость **v'** часто вычисляются при помощи правил:

$$x' = x + v \cdot \Delta t$$
$$v' = v + a \cdot \Delta t$$

где Δt временной интервал, а **a** – ускорение, вычисленное при использовании закона Ньютона **f**=m**a** (где **f** – общая сила, действующая на частицу). Это простое объединение Эйлера.

Здесь, однако, мы выбираем скорость меньшую представленной и схему объединения автора: вместо хранения положения и скорости каждой частицы, мы храним её текущую позицию **x** и её предыдущую позицию $\mathbf{x}^*$. Сохраняя временной интервал постоянным, мы улучшаем правила (или интегрируем шаг) и тогда:

$$x' = 2x - x^* + a \cdot \Delta t^2$$
$$x^* = x$$

Это называется интегрированием Верлета и интенсивно используется при симуляции молекулярной динамики. Оно весьма устойчиво, когда скорость задана неявно, а следовательно скорости и позиции сложнее выйти из синхронизации. (Для заметки, хорошо известная демонстрация эффекта колеблющихся волн на воде использует похожий подход). Это работает должным образом из факта, что $2\mathbf{x}-\mathbf{x}^*=\mathbf{x}+(\mathbf{x}-\mathbf{x}^*)$ и $\mathbf{x}-\mathbf{x}^*$ - приближено к текущей скорости (фактически – это дистанция пройденная за последний интервал времени). Это не всегда точно (энергия может покидать систему, т.е. рассеиваться), но это быстро и стабильно. Понижение значения 2, например, до 1,99 также приемлемо для системы.

В конце каждого шага, для каждой частицы текущая позиция **x** получается сохранённой в соответствующей переменной $\mathbf{x}^*$. Замечание, когда обрабатывается много частиц, полезная оптимизация возможна простым обменом массива точек.

Результирующий код должен выглядеть примерно так (Vector3 – класс должен содержать соответствующие член функции и перезагрузчики операторов для манипулирования с вектором):

```
// Образец кода для физической симуляции
class ParticleSystem {
   Vector3    m_x[NUM_PARTICLES];    // Текущая позиция
   Vector3    m_oldx[NUM_PARTICLES]; // Предыдущая позиция
   Vector3    m_a[NUM_PARTICLES];    // Силовые накопители
   Vector3    m_vGravity;            // Гравитация
   float      m_fTimeStep;
public:
   void       TimeStep();
private:
   void       Verlet();
   void       SatisfyConstraints();
```

```
   void       AccumulateForces();
// (конструкторы, инициализация, и т.д. упущены)
};
// Шаг интегрирования Верлета
void ParticleSystem::Verlet() {
   for(int i=0; i<NUM_PARTICLES; i++) {
      Vector3& x = m_x[i];
      Vector3 temp = x;
      Vector3& oldx = m_oldx[i];
      Vector3& a = m_a[i];
      x += x-oldx+a*fTimeStep*fTimeStep;
      oldx = temp;
   }
}
// Эта функция должна накапливать силы для каждой частицы
void ParticleSystem::AccumulateForces()
{
// Все частицы подвержены гравитации
   for(int i=0; i<NUM_PARTICLES; i++) m_a[i] = m_vGravity;
}
// Здесь должны быть удовлетворены связи
void ParticleSystem::SatisfyConstraints() {
// Пока игнорируем эту функцию
}
void ParticleSystem::TimeStep() {
   AccumulateForces();
   Verlet();
   SatisfyConstraints();
}
```

Описанный код был написан для ясности, а не для скорости. Одна оптимизация была бы полезна – массивы float вместо Vector3 для представления. Это также может сделать это проще для осуществления системы на векторном процессоре.

Это, возможно, не звучит очень инновационно, но это пока. К тому же, преимущества должны прийти очень скоро, когда мы начнём использовать связи и переключимся на твёрдые тела.

Попробуйте установить **a**=(0,0,1), например, и использовать начальное значение **x**=(1,0,0), $\mathbf{x}^*$=(0,0,0), затем прогоните цикл вручную и увидите, что получится.

Обработка столкновений и контактов проецированием

Так называемые схемы, базирующиеся на штрафах (penalty-based schemes) обрабатывают контакт, устанавливая пружины (springs, чтобы это не значило) в точке проникновения. Хота это очень легко реализовать, но это имеет несколько серьёзных недостатков. Например, очень сложно выбрать подходящую константу скачка, с одной стороны, объекты не проникают глубоко, а с другой, результирующая система не становиться нестабильной. В других схемах по симуляции физики, столкновение обрабатывается перематыванием времени (например двоичным поиском) для определения точки столкновения, затем обрабатывают столкновения аналитически, потом перезапускают симуляцию – это не практично с точки зрения реального времени, т.к. потенциально код может работать очень медленно там, где много столкновений.

Здесь мы используем всё же другую стратегию. Конечная точка проецируется из препятствия. Под проецированием, проще говоря, мы понимаем, перемещение точки, на малое возможное расстояние, до тех пор, пока она вне препятствия. Обычно это значит перемещение точки перпендикулярно к поверхности пересечения.

Давайте посмотрим на примере. Предположим, что наш мир внутри куба (0,0,0)-(1000,1000,100) и также предположим, что коэффициент восстановления частицы ноль (эти частицы не отскакивают от поверхности при соударении). Чтобы сохранить все положения внутри допустимого интервала, соответствующий код проецирования должен быть таким:

```
// Существующие частицы в кубе
void ParticleSystem::SatisfyConstraints() {
   for(int i=0; i<NUM_PARTICLES; i++) { // Для всех частиц
```

```
      Vector3& x = m_x[i];
      x = vmin(vmax(x, Vector3(0,0,0)),
         Vector3(1000,1000,1000));
   }
}
```

(vmax, работающая с вектором, берёт максимальный компонент, принимая во внимание то, что vmin берёт минимальный компонент). Это позволяет сохранить все положения частиц внутри куба и обработать оба контакта: столкновение и покой. Красота интегрирования Верлета в том, что соответствующие изменения в скорости будут применены автоматически. В следующих вызовах TimeStep(), скорость автоматически регулируется как не содержащая нормальной составляющей к поверхности (что соответствует коэффициенту восстановления ноль). Смотри Рис.1.

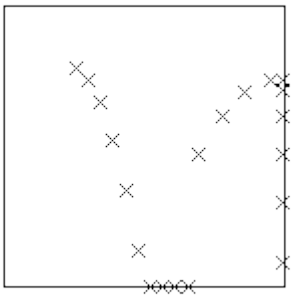


Рис. 1: 10 временных интервалов и 2-е частицы.

Проверьте – не нужно по ходу отменять нормальную составляющую скорости. В то время, как описанное может показаться чем-нибудь тривиальным, когда смотришь на частицу, сила схемы интегрирования Верлета начнёт светиться и станет очевидой, когда появятся связи и связанные твёрдые тела.

Разрешение нескольких действующих связей при помощи расслабления

Обычная модель одежды состоит из простой системы взаимосвязанных пружин (springs) и частиц. Однако, не всегда тривиально решать соответствующую систему дифференциальных уравнений. Это страдает от проблем, сходных с системой основанной на штрафах (penalty-based system): Сильные пружины (springs) ведут к жестким системам уравнений, которые ведут к неустойчивости, если используются простые методы интегрирования, или, по крайней мере, к плохому выполнению, которое болезненно. И наоборот, слабые пружины (springs) ведут к эластично выглядящей одежде.

Хотя, происходит интересная вещь, если мы позволяем жёсткости пружин (springs) уйти в бесконечность: система неожиданно становится решаемой путём очень простого и быстрого приближения. Но перед этим мы продолжим разговор об одежде, давайте пересмотрим предыдущий пример. Куб, рассматриваемый выше, может быть рассмотрен как набор односторонних (неравных) ограничений (связей) (по одной на каждую сторону куба) на положение частицы, которые всегда должны быть удовлетворены:

$x_i \geq 0$ и $x_i \leq 1000$ для $i = 1,2,3$ (C1)

В примере, связи удовлетворены (это, частицы, сохраняемые внутри куба) простым исправлением непригодной позиции, путём проецирования на поверхность куба. Чтобы удовлетворить (C1), мы используем следующий псевдокод:

```
// Псевдокод удовлетворяющий (C1)
for i=1,2,3
   set xi=min{max{xi, 0}, 1000}
```

Можете думать об этом процессе, как о вставке бесконечно жёсткой пружины между частицей и плоскостью проникновения – пружины, которые настолько сильны и подходяще затухающи, немедленно приведут их длины отдыха к нулю.

Теперь мы расширяем эксперимент, для моделирования стержня длиной 100. Мы делаем это путём введения двух индивидуальных частиц (с позициями **x1** и **x2**) и, затем, приказываем им быть на расстоянии 100 частей. Выражаясь математически, мы получили следующее двустороннее (равное) ограничение:

$$|x2 - x1| = 100 \quad \text{(C2)}$$

Хотя частицы могли быть помещены правильно изначально, после одного шага интеграции расстояние разделения между ними могло бы стать недействительным. Чтобы получать правильное расстояние еще раз, мы перемещаем частицы путем проецирования их на набор решений, описанных (C2). Это делается путём выталкивания частиц направлено друг от друга или путем втягивания их ближе друг к другу (в зависимости оттого, что ошибочное расстояние меньше или больше). Смотри Рис.2.

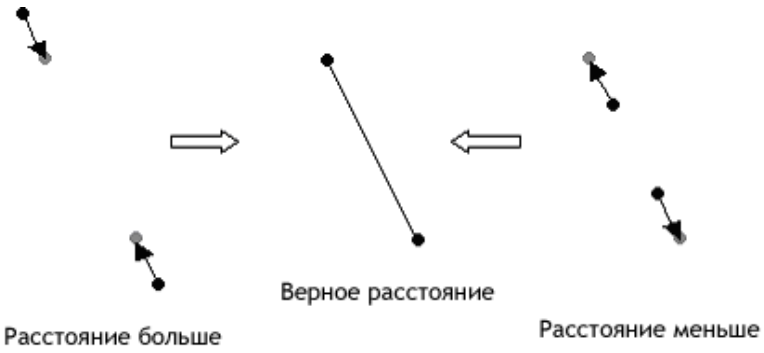


Рис.2: Правка неверного расстояние движением частиц.

Псевдокод, удовлетворяющий реакции (C2):

```
// Псевдокод для удовлетворения (C2)
delta = x2-x1;
deltalength = sqrt(delta*delta);
diff = (deltalength-restlength)/deltalength;
x1 -= delta*0.5*diff;
x2 += delta*0.5*diff;
```

Заметьте, что delta – это вектор, а delta*delta фактически скалярное произведение. С restlength=100 описанный псевдокод будет толкать каждую или тянуть вместе частицы так, чтобы они снова достигли верного расстояние между ними, равное 100. Опять мы можем думать об этой ситуации, как будто очень жёсткая пружина с длиной отдыха 100 была вставлена между частицами, так что они немедленно устанавливаются правильно.

Теперь предполагаем, что мы всё ещё хотим, чтобы частицы удовлетворяли ограничениям (связям) куба. Удовлетворяя ограничениям стержня, однако, мы могли лишиться одного или более ограничений куба, когда выталкивали частицу за куб. Эта ситуация может быть исправлена немедленным проецированием непригодного положения частицы назад на поверхность куба, снова – но тогда мы снова лишаем силы ограничения стержня.

На самом деле, мы должны решать задачу для всех ограничений сразу, оба (C1) и (C2). Это бы вызвало решение системы уравнений. Однако, мы выбрали, что будем продолжать косвенно путём местных повторений. Мы просто повторяем два куска псевдокода неоднократно друг после друга, в надежде, что результат полезен. Это приводит к следующему коду:

```
// Симуляция стержня в кубе
void ParticleSystem::SatisfyConstraints() {
   for(int j=0; j<NUM_ITERATIONS; j++) {
         // Первое удовлетворение (C1)
         for(int i=0; i<NUM_PARTICLES; i++) { // Для всех частиц
            Vector3&             x = m_x[i];
               x = vmin(vmax(x, Vector3(0,0,0)),
                  Vector3(1000,1000,1000));
         }
```

```
         // Затем удовлетворение (C2)
         Vector3& x1 = m_x[0];
         Vector3& x2 = m_x[1];
         Vector3 delta = x2-x1;
         float deltalength = sqrt(delta*delta);
         float diff = (deltalength-restlength)/deltalength;
         x1 -= delta*0.5*diff;
         x2 += delta*0.5*diff;
   }
}
```

(Инициализация этих двух частиц была опущена). В то время как этот подход чистого повторения мог показаться наивным, это убирается тем, что фактически он сходится с решением, которое мы искали! Метод называется расслаблением (или Якоби или Гаусс-Сейдел повторение в зависимости от применения). Он работает, последовательно удовлетворяя различным местным связям, и затем повторяется; если условия верны, то он сходится с общей конфигурацией, которая удовлетворяет всем связям одновременно. Это полезно во многих других ситуациях, где несколько независимых связей должны быть удовлетворены одновременно.

Число необходимых повторений варьируется в зависимости от имитируемой физической системы и числа движений. Он может быть сделан адаптируемым путём измерения изменений с последнего повторения. Если мы остановим повторения рано, то конечный результат может не быть достаточно правильным из-за использования схемы Верлета, в следующем кадре он лучше, в следующем ещё лучше и т.д. Это значит, что остановка рано не разрушит всего, хотя результирующая анимация может оказаться кое-где «кривой».

Моделирование одежды

Тот факт, что связь стержнем может действительно считаться, как абсолютно жёсткая пружина, должен сделать очевидным его полезность для моделирования одежды, как было отмечено в начале этой секции. Предположим, например, что был создан шестиугольный меш из треугольников, описывающий одежду. Для каждой вершины инициализирована частица, а для каждой грани – стержень связи между двумя соответствующими частицами (с ограничениями «длина отдыха», которые просто являются начальными расстояниями между двумя вершинами).

Функция `HandleConstraints()`, тогда использует расслабления для всех связей (ограничений). Цикл расслабления должен быть прогнан несколько раз. Однако, для получения лучше выглядящей анимации, фактически для большинства кусков одежды необходимо всего одно повторение! Это значит, что время, использованное на моделирование одежды, в основном зависит от N квадратных корней и N выполненных делений (где N показывает число граней в меше одежды). Как мы видим, умный трюк, позволяет улучшить это уменьшением до N делений за кадр – что действительно быстро и можно доказать, что нельзя сделать быстрее.

```
// Осуществление моделирования одежды
struct Constraint {
   int particleA, particleB;
   float restlength;
};
// Полагаем, что массив связей, m_constraints, существует
   void ParticleSystem::SatisfyConstraints() {
      for(int j=0; j<NUM_ITERATIONS; j++) {
         for(int i=0; i<NUM_CONSTRAINTS; i++) {
         Constraint& c = m_constraints[i];
         Vector3& x1 = m_x[c.particleA];
         Vector3& x2 = m_x[c.particleB];
         Vector3 delta = x2-x1;
         float deltalength = sqrt(delta*delta);
         float diff=(deltalength-c.restlength)/deltalength;
         x1 -= delta*0.5*diff;
         x2 += delta*0.5*diff;
      }

      // Привяжем одну частицу ткани к «origo»
      m_x[0] = Vector3(0,0,0);
```

```
    }
}
```

Теперь обсудим, как избавиться от операции квадратного корня. Если все ограничения удовлетворены (некоторые частично), мы уже знаем, что результат квадратного корня, используется при длине отдыха r соответствующего стержня. Мы можем использовать этот факт, для приближения функции квадратного корня. Математически, мы приближаем квадратный корень в соответствии с его 1-ым порядку по формуле Тейлора. После некоторых исправлений, мы получаем следующий псевдокод:

```
// Псевдокод для удовлетворения (C2) используя приближения квадр. корня
delta = x2-x1;
delta*=restlength*restlength/(delta*delta+restlength*restlength)-0.5;
x1 -= delta;
x2 += delta;
```

Заметьте, что если расстояние уже правильное (если |delta|=restlength), то каждый получает delta=(0,0,0) и не происходит никаких изменений.

Теперь мы используем ноль квадратных корней за ограничение, и одно деление, а значение квадрата restlength*restlength может быть посчитано заранее. Использование операций потребляющих время теперь снижено до N делений за кадр (и соответственно доступ к памяти) – это нельзя сделать быстрее, чем так и результат выглядит очень хорошо. Фактически, в HitMan'е, полная скорость моделирования одежды была ограничена количеством треугольников, которое было возможно обработать при рендеренге.

Не гарантировано, что ограничения будут удовлетворены после одного повторения, но из-за схемы интегрирования Верлета, система будет быстро сходиться к корректным рамкам после нескольких кадров. Фактически, использование только одного повторения и приближение квадратного корня, убирает неподвижность, которая возникает, когда стержни абсолютно жёсткие.

Помещение поддерживающих стержней между продуманно выбранными парами вершин, разделяя соседей, алгоритм моделирования одежды может быть расширен для моделирования растений. Снова, в HitMan'е только один проход на расслабление был достаточен (фактически, меньшее количество дало растениям правильное состояние поведения изгибания).

Код и уравнения, охваченные в этой секции, предполагают, что все частицы имеют одинаковые массы. Конечно, возможно моделировать частицы с различными массами, просто уравнение усложняется.

Чтобы удовлетворить (C2) , при учёте масс частиц, используйте следующий код:

```
// Псевдокод для удовлетворения (C2)
delta = x2-x1;
deltalength = sqrt(delta*delta);
diff = (deltalength-restlength)
      /(deltalength*(invmass1+invmass2));
x1 -= invmass1*delta*diff;
x2 += invmass2*delta*diff;
```

Где invmass1 и invmass2 – численно инверсированные две массы. Если мы хотим сделать частицу неподвижной, просто ставим для неё invmass=0 (соответствует бесконечной массе). Конечно в предыдущем случае, квадратный корень тоже может быть приближен для увеличения скорости.

Твёрдые тела

Уравнения управляющие движением твёрдых тел были открыты до появления современных компьютеров. Чтобы быть способными сказать что-нибудь полезное в то время, математики нуждались в способности управлять выражениями в символах. В теории твёрдых тел, главными полезными терминами и инструментами стали тензор инерции, угловой импульс, момент, кватернион для представления ориентации и т.д. Однако, с нынешними способностями обработки огромного количества данных в цифровой форме, стало выполнимым и в некоторых случаях даже выгодней свести вычисления к более простым элементам, когда запускаешь моделирование. В случае 3D

твёрдых тел, это может значить моделирование твёрдого тела с 4-мя частицами и 6-ью ограничениями (связями) (даёт верное количество степеней свободы, 4x3-6=6). Это упрощает большинство положений и это то, что мы будем делать далее.

Рассмотрим четырёхгранник и поместим частицу в каждую из 4-ёх вершин. В дополнении, каждой из 6-и граней четырёхгранника, создадим ограничения расстояния при помощи стержня, как это было описано в предыдущей секции. Фактически этого достаточно, для моделирования твёрдого тела. Четырёхгранник может быть выпущен в мировой куб, описанный ранее, и интегрирование Верлета позволит ему двигаться правильно. Функция `SatisfyConstraints` позаботиться о двух вещах: 1) Чтобы частицы оставались внутри куба (как ранее), и 2) Чтобы шесть ограничений расстояния удовлетворялись. Снова, это может быть сделано при использовании метода расслаблений; 3 или 4 повторения должно быть достаточно с приближением квадратного корня.

Для ясности, в общем, твёрдые тела не ведут себя как четырёхугольники при столкновении (но они могли бы вести себя так кинетически). Также есть ещё одна проблема: сейчас, определение пересечения между твёрдым телом и миром основывается лишь на одной вершине, это, если вершина обнаружена за миром, тогда она проецируется обратно внутрь. Это работает хорошо до тех пор, пока мир выпуклый. Если мир не выпуклый, тогда четырёхугольник и мир должны фактически проникать друг в друга каждый раз, когда вершины четырёхугольника в допустимой области (смотри Рис.3, где треугольник представляет 2D аналог четырёхугольника). Эта проблема обрабатывается далее:

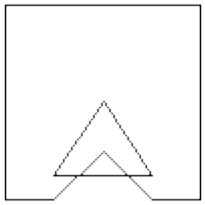


Рис.3: четырёхугольник проникает в мир.

Для начала рассмотрим простую версию проблемы. Рассмотрим пример со стержнем, описанный ранее, и предположим, что мир имеет небольшую выпуклость. Стержень может теперь проникать в мир, когда частицы стержня этот мир не покидают (Смотри Рис.4). Мы не будем вникать в запутанности создания движка по определению столкновения, так как это целая наука. Вместо этого мы полагаем, что есть доступная подсистема, которая позволяет нам определять столкновения. Кроме того, мы полагаем, что подсистема может показать нам глубину проникновения и определить точки проникновения на обоих сталкивающихся объектах. Одно из определений точек проникновения и глубины проникновения похоже на это: Расстояние проникновения d_p – наименьшее расстояние, которое предотвратило бы проникновение двух объектов, если бы один был перемещён на расстояние d_p в соответствующем направлении. Точки проникновения – точки на обоих объектах, которыми только касаются оба объекта после вышеупомянутого перемещения.

Давайте снова посмотрим на Рис.4. Здесь стержень переместился через неровность после шага Верлета. Движок столкновения определил две точки проникновения, **p** и **q**. На Рис.4a, **p** – фактически идентична позиции частицы 1, т.е. **p**=**x1**. На Рис.4b, **p** лежит между **x1** и **x2** на позиции ¼ от длины стержня от **x1**. В обоих случаях, точка **p** лежит между на стержне и далее она пожжет быть выражена, как линейная комбинация **x1** и **x2**, **p**=c1*__x1__+c2*__x2,__ где c1+c2=1. В первом случае, c1=1 и c2=0, а во втором c1=0.75 and c2=0.25. Эти значения говорят нам, как сильно мы должны переместить соответствующие частицы.

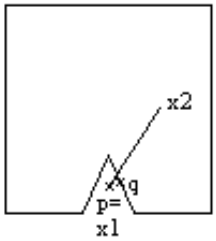


Рис.4a. Столкновение стержня I

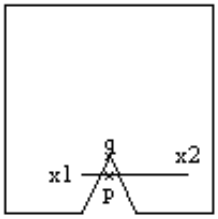


Рис.4b. Столкновение стержня II

Для правки неверной конфигурации стержня, он должен быть как-то перемещён вверх. Наша цель – избежать проникновения перемещением **p** в туже позицию, что и **q**. Мы сделаем это корректировкой положения двух частиц **x1** и **x2** в направлении вектора между **p** и **q**, **Δ=q-p**.

В первом случае, мы просто проецируем **x1** наружу из неверной области, как ранее (в направлении к **q**) и это всё (**x2** не трогаем). Во втором случае, **p** всё ещё ближе к **x1** и это весомая причина к тому, что в последствии **x1** должен быть перемещён больше, чем **x2**. Вообще, из того, что **p**=0.75***x1** + 0.25***x2**, мы перемещение **x1** на составляющую перемещения 0.75, а **x2** перемещаем в количестве 0.25. Другими словами, новые позиции частиц **x1'** и **x2'** получаются из выражения:

$$\begin{aligned} x1' &= x1 + 0.75 \cdot \lambda \cdot \Delta \\ x2' &= x2 + 0.25 \cdot \lambda \cdot \Delta \end{aligned} \quad (*)$$

где λ - некоторое неизвестное значение. Новое положение **p** после перемещения обеих частиц - **p'**=c1***x1'**+c2***x2'**.

Вспомним, что **p'**=**q**, т.е. мы должны выбрать λ точно такой, чтобы **p'** точно совпадала с **q**. Так как мы перемещаем частицы только в направлении **Δ**, то также и **p** перемещается в направлении Δ и следовательно решение уравнения **p'**=**q**, может быть найдено решением:

$$p' \cdot \Delta = q \cdot \Delta \quad (**)$$

для λ. Распишем левую сторону:

$$\begin{aligned} p' \cdot \Delta &= (0.75 \cdot x1' + 0.25 \cdot x2') \cdot \Delta = (0.75 \cdot (x1 + 0.75\lambda \cdot \Delta) + 0.25 \cdot (x2 + 0.25\lambda \cdot \Delta)) \cdot \Delta = \\ &= (0.75 \cdot x1 + 0.25 \cdot x2)\Delta + \lambda(0.75^2 + 0.25^2)\Delta^2 = p \cdot \Delta + \lambda(0.75^2 + 0.25^2) \cdot \Delta^2 \end{aligned}$$

что вместе с правой частью (**) даёт:

$$\lambda = \frac{(q - p) \cdot \Delta}{(0.75^2 + 0.25^2) \cdot \Delta^2}$$

Подставляя λ в (*) получим новые позиции частиц, для которых **p'** совпадает с **q**.

Рис.5 показывает ситуации после перемещения частиц. У нас нет проникновения объекта, но теперь длина стержня связи нарушена. Чтобы исправить это, мы делаем ещё одно повторение цикла расслабления (или несколько) и мы закончили.

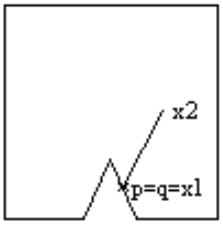


Рис. 5a. Пример I разрешения

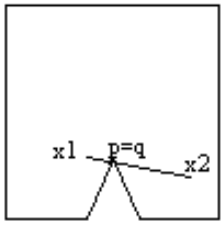


Рис. 5b. Пример II разрешения

Описанная выше стратегия также работает и для четырёхгранника в полностью аналогичной манере. Сначала находятся точки проникновения **p** и **q** (они могут быть также точками внутри треугольника), и **p** выражается как линейная комбинация четырёх частиц **p**=c1***x1**+c2***x2**+c3***x3**+c4***x4**, где c1+c2+c3+c4=1 (это приводит к решению маленькой системы уравнений). После нахождения Δ=**q**-**p**, вычисляем значение:

$$\lambda = \frac{(\overline{q} - \overline{p}) \cdot \overline{\Delta}}{(c1^2 + c2^2 + c3^2 + c4^2) \cdot \overline{\Delta}^2}$$

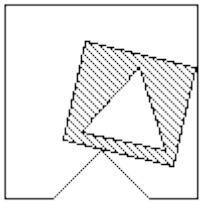
тогда получаем новые положения:

$$x1' = x1 + c1 \cdot \lambda \cdot \Delta$$
$$x2' = x2 + c2 \cdot \lambda \cdot \Delta$$
$$x3' = x3 + c3 \cdot \lambda \cdot \Delta$$
$$x4' = x4 + c4 \cdot \lambda \cdot \Delta$$

Здесь мы столкнули одно твёрдое тело с неподвижным миром. Описанный выше метод обобщает обработку столкновения нескольких твёрдых тел. Столкновения, обрабатываемые для одной пары тел одновременно. Вместо перемещения только **p**, в этом случае оба **p** и **q** перемещаются друг к другу.

Снова, после корректировки положения частиц сделанного для того, чтобы они удовлетворяли ограничениям не проникновения, шесть ограничений расстояний, составляющих твёрдое тело, должны быть учтены и т.д. При помощи этого метода, четырёхгранник может быть даже вложен внутрь другого объекта, который может быть использован вместо четырёхгранника для обработки столкновения. На Рис.6 четырёхгранник вложен в куб.

Сначала, куб должен быть «закреплён» к четырёхграннику каким-нибудь образом. Один подход – выбрать центр масс 0.25*(**x1**+**x2**+**x3**+**x4**), как положение куба и затем получить матрицу ориентации, исследуя текущее положение частиц. Когда обнаружено столкновение/проникновение, точка столкновения **p** (которая в этом случае размещена на кубе) обрабатывается в точности как ранее и позиция частиц обновляется соответственно. Как оптимизация, можно рассчитать заранее все значения c1-c4, для всех вершин куба. Если точка проникновения **p** – вершина, тогда значения c1-c4 могут быть найдены, и использоваться направленно. Иначе, **p** лежит внутри поверхности треугольника или на одной из граней и значения c1-c4 могут быть интерполированы между вычисленными значениями соответствующих вершин треугольника.



Вложение четырёхгранника в другой объект.

Обычно, 3-4 повторения расслабления достаточно. Тела не будут вести себя так, как будто они абсолютно твёрдые, когда повторения расслабления остановлены преждевременно. Это очень хорошая особенность, фактически, нет таких вещей, которые можно считать абсолютно твёрдыми телами – особенно человеческие тела. Это также делает систему более устойчивой.

Перегруппировка положений частиц соответственно меняет физические свойства четырёхгранник (математически, изменение тензора инерции, когда меняются положения и массы частиц).

Другие способы частиц и связей чем четырёхгранник возможны, например помещение частиц в оси базиса системы координат, т.е. в (0,0,0), (1,0,0), (0,1,0), (0,0,1). Пусть **a**, **b** и **c** – векторы из частицы 1 в частицу 2, 3 и 4, соответственно. Ограничьте положения частиц, требуя, чтобы вектора **a**, **b** и **c** имели длину 1 и углы между ними равнялись 90 градусов (скалярное произведение равно 0). Замечание, это снова дает четыре частицы и шесть связей.

Соединённые тела

Можно соединить несколько твёрдых тел при помощи шарнирного, жёсткого соединения, и т.д. Просто даём двум твёрдым телам общую частицу, и они соединены гибко. Разделим между ними две частицы, и они будут соединены жёстко. Смотри Рис.7.

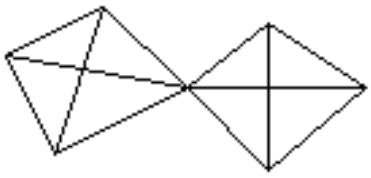


Рис 7a. Гибкое соединение

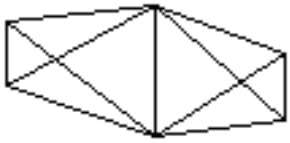


Рис 7b. Жёсткое соединение

Так же можно соединить два твёрдых тела стержневым соединением или любым другим видом соединения – чтобы сделать это, просто добавляем соответствующее исправление в предыдущий код цикла повторения.

Этот подход позволяет построить законченную модель соединённого человеческого тела. Для дополнительного реализма, должны быть реализованы различные угловые ограничения. Есть несколько способов сделать это. Простой путь – использовать стержневые связи, которые приводятся, если расстояние между двумя частицами падает ниже некоторого порога (математически, мы имеем односторонне ограничение расстояния, |**x2**-**x1**|>100). В результате, обе частицы никогда не подойдут слишком близко друг к другу. Смотри Рис.8.

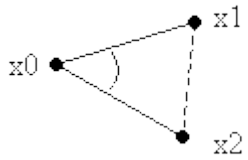


Рис. 8: Два стержневых ограничения и ограничение неравенством (пунктиром).

Другой способ ограничения углов в том, чтобы удовлетворить ограничению скалярного произведения:

$$(x2 - x0) \cdot (x1 - x0) < \alpha$$

Частицы также могут быть ограничены в движении, например, в некоторых плоскостях. И снова, частицы с позициями, не удовлетворяющими вышеупомянутым ограничениям, должны быть перемещены – решение этого несколько сложнее, чем с ограничениями стержня.

Вообще, в HitMan'е трупы не составлены из твёрдых тел смоделированных при помощи четырёхгранников. Они проще, они состоят из частиц соединённых стержневыми связями. Смотри Рис.9.

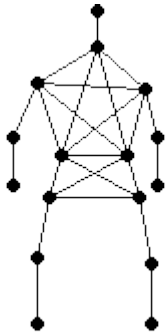


Рис.9:Конфигурация частиц и стержней используемая в HitMan'е для представления человеческого тела.

Другими словами, как видно, каждая изолированная конечность не твёрдое тело с 6-ью степенями свободы. Это значит, что физически поворот вокруг оси длины конечности не моделируется. Вместо этого, система скелетной анимации используется для установки полигонального меша персонажа принудительно, например, ориентировать ногу так, чтобы она смотрелась естественно. Так как вращение рук и ног вокруг своих осей не обязательно участвует в падение человеческих тел, это работает очень хорошо и с хорошей скоростью.

Угловые ограничения вводятся, чтобы применить ограничения человеческой анатомии. О простом самопересечении позаботятся тактически выбранные ограничения неравенствами, как это было описано ранее, например между двумя коленями – сделает так, что ноги никогда не пересекутся.

Для столкновения со средой, которая состоит из треугольников, каждый стержень представлен покрытым цилиндром. Где-то в системе пересечения, подпрограмма обрабатывает пересечение между натянутым цилиндром и треугольником. Когда обнаружено пересечение, извлекаются глубина и точки проникновения, и пересечение обрабатывается для непригодного стержня так же, как и было описано в начале секции 5.

В действительности, множество дополнительных достижений потребовалось, чтобы получить правильные результаты.

Комментарии

Эта секция содержит различные замечания, которые нигде больше не приводятся.

Контроль движения

Для влияния на движение моделируемого объекта, просто перемещаем частицы, на которые оказано влияние. Если персонаж ударен в плечо, переместите частицу плеча назад на расстояние пропорциональное силе удара. Интегрирование Верлета автоматически приведёт плечо в движение.

Это также делает проще моделирование «наследуемых» скоростей от основной, традиционной системы анимации. Просто запишите положения частиц из двух кадров и затем отдайте их интегрированию Верлета, которое автоматически продолжет движение. Бомбы могут быть созданы, путём выталкивания каждой частицы в системе прочь от взрыва, на расстояние обратно пропорциональное квадрату расстояния между частицей и центром бомбы.

Также возможно ограничить конкретную конечность, скажем руку, зафиксировав её положение в пространстве. Таки образом можно создать инверсную кинематику: внутри цикла расслабления, установите положение выбранной частицы (или частиц) в необходимое место. Дайте частице бесконечную массу (invmass=0) – это позволит сделать частицу неподвижной для физической системы. В HitMan'е эта стратегия использована, когда тащите трупы, голова (или шея, или нога) трупа вынуждена следовать за рукой игрока.

Обработка трения

До сих пор не было принято во внимание трение. Это значит, что если мы не сделаем кое-чего ещё, частицы будут скользить по полу так, как будто он сделан изо льда. Согласно модели трения Коломбо, сила трения зависит от размера нормальной силы между объектом и препятствием. Для

создания этого, мы измеряем глубину проникновения d_p, когда оно возникло (до проецирования точки проникновения за препятствие). После проецирования частицы на поверхность, тангенсальная скорость $\mathbf{v_t}$ уменьшается на количество пропорциональное d_p (коэффициент пропорциональности – коэффициент трения). Это делается путём соответствующего изменения $\mathbf{x}^*$. Смотри Рис.10. Надо позаботиться о том, чтобы тангенсальная скорость не изменила своего направления, когда это произойдёт, она должна быть установлена в ноль. Есть и другие, более лучшие, модели трения.

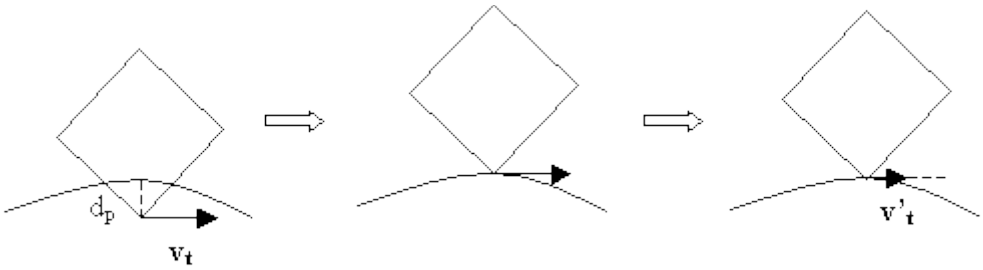


Рис.10: Обработка столкновения с трением (проецирование и изменение тангенсальной скорости).

Определение столкновения

Одно из сложнейших мест в моделировании физики, как здесь сказано – определение столкновения, которое несколько раз повторяется в цикле расслабления.

В HitMan'е система столкновения работает отбрасыванием всех треугольников кроме тех, что внутри ограничивающих боксов моделируемого объекта (это сделано при помощи Octtree). Для каждого (статического, фонового) треугольника, структура для быстрого столкновения снова обрабатывает облегающий цилиндр, сконструированный и кэшированный. Этот метод даёт большой скачёк в скорости.

Для предотвращения перемещения объектов, которые двигаются очень быстро, через другие объекты (из-за большого временного шага), надо провести простой тест. Представьте линию (или цилиндр подходящего радиуса) с началом в середине объекта в прошлом кадре и с концом тоже в середине, но в текущем кадре. Если эта линия что-нибудь ударяет, тогда положение объекта устанавливается в точку столкновения. Теоретически это неправильно, но на практике работает хорошо.

Разное

Количество повторений расслаблений в HitMan'е изменяется от 1 до 10, в зависимости от моделируемого объекта. Хотя этого недостаточно для точного решения глобальной системы связи, но этого достаточно, чтобы сделать движение натуральным. Хорошая вещь этой системы в том, что погрешности не накапливаются или сохраняются визуально невызывая дрейфа объекта или ещё чего такого – в некотором смысле комбинация проецирования и схемы Верлета разбивает сложные вычисления на несколько кадров (другие схемы должны использовать дальнейшую стабилизацию, например стабилизация Баумгарта). К счастью погрешности малы или почти не существуют, когда присуще малое движение, но велики когда большое движение – это хорошо так как быстрое или сложное движение несколько маскирует погрешности для человеческого глаза.

Один вид мягких тел может быть осуществлён при использовании «мягких» ограничений – это ограничения, которым разрешено иметь только несколько процентов от отклонения «восстановления» каждый кадр (т.е. если длина отдыха стержня между двумя частицами 100, но реальное расстояние 60, то код расслабления должен сначала установить расстояние в 80, вместо 100, на следующем кадре 90, 95, 97.5 и т.д.).

(*я решил не переводить эту часть статьи, т.к. она не содержит ничего стоящего, для понимания описанной темы, здесь велось описание о том, где можно посмотреть математическую подноготную, в основном это длинный список авторов и статей.*)

Заключение

Эта статья описывает, как реализована физическая система в HitMan'e. Основанная на философии комбинирования повторяющихся методов со стабильным интегратором, оказалась успешной и полезной для применения в компьютерных играх. Наиболее значима, универсальная объединённая структура на основе частиц, которая обрабатывает пересечение и контакт, и способна обмениваться между скоростью и точностью без накопления визуально-очевидных ошибок. В действительности, есть ещё много специфических особенностей, которые можно улучшить. В частности требует доработки моделирование твёрдых тел четырёхгранником. Это находится в доработке.

В IO Interactive, мы недавно провели несколько экспериментов с интерактивной водой и газом, смоделированными при использовании полного уравнения Навье-Стокса. Сейчас мы ищем применение схожих методов, описанных в этой статье, в надежде сделать быструю и более стабильную модель воды.

Ссылки

[Baraff] Baraff, David, *Dynamic Simulation of Non-Penetrating Rigid Bodies*, Ph.D. thesis, Dept. of Computer Science, Cornell University, 1992. http://www.cs.cmu.edu/~baraff/papers/index.html

[Mirtich] Mirtich, Brian V., *Impulse-base Dynamic Simulation of Rigid Body Systems*, Ph.D. thesis, University of California at Berkeley, 1996. http://www.merl.com/people/mirtich/papers/thesis/thesis.html

[Press] Press, William H. et al, *Numerical Recipes*, Cambridge University Press, 1993. http://www.nr.com/nronline_switcher.html

[Stewart] Stewart, D. E., and J. C. Trinkle, "An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic Collisions and Coulomb Friction", *International Journal of Numerical Methods in Engineering*, to appear. http://www.cs.tamu.edu/faculty/trink/Papers/ijnmeStewTrink.ps

[Verlet] Verlet, L. "Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules", Phys. Rev., 159, 98-103 (1967).

[Witkin] Witkin, Andrew and David Baraff, "Physically Based Modeling: Principles and Practice", Siggraph '97 course notes, 1997. http://www.cs.cmu.edu/~baraff/sigcourse/index.html