МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики
(название кафедры)

А. М. Зеленчук
(И., О., фамилия студента – автора работы)

Исследование хаотичности движения системы неоднородных тел на плоскости
(полное название темы магистерской диссертации)

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
по направлению высшего профессионального образования

230100.68 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема диссертации утверждена распоряжением по НГУ №___ от «__» _____ 20__ г.
Тема диссертации скорректирована распоряжением по НГУ №___ от «__» _____ 20__ г.

Руководитель

Кренделев С. Ф.
(фамилия , И., О.)
к. ф.-м. н., доцент НГУ
(уч.степень, уч.звание)

Новосибирск, 2013 г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики
(название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Лаврентьев М. М.
(фамилия, И., О.)

……
(подпись, дата)

ЗАДАНИЕ
на магистерскую диссертацию

студент Зеленчук Андрей Михайлович
(фамилия, имя, отчество)

факультета ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Направление подготовки 230100.68 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Магистерская программа 230100.68 Компьютерное моделирование
(наименование)

Тема Исследование хаотичности движения системы неоднородных тел на плоскости
(полное название темы)

Цели работы
Построить генераторы псевдослучайных чисел, используя механическую модель. В качестве модели использовать модель движения нескольких объектов с взаимодействиями. Взаимодействия определяются столкновениями. (Такие объекты называются бильярдами.) Требуется изучить модель с точки зрения хаотичности движения объектов.

Руководитель

Кренделев С. Ф.
(фамилия , И., О.)
к. ф.-м. н., доцент НГУ
(уч.степень, уч.звание)

……..
(подпись, дата)

Содержание

ВВЕДЕНИЕ

Случайные числа широко используются в различных областях науки и техники: в методах Монте-Карло, моделировании, криптографии. Многие прикладные задачи криптографии требуют случайных последовательностей, например, генерация ключей, одноразовые блокноты.

Лучшее, что может сделать компьютер – это генератор псевдослучайных последовательности, то есть последовательностей, элементы которой почти независимы друг от друга и подчиняются заданному (обычно равномерному) распределению.

Во многих генераторах псевдослучайных чисел используются линейный конгруэнтный генератор (Linear Congruential Generator, LCG) или сдвиговые регистры с линейной обратной связью (Linear Feedback Shift Register, LFSR). Сами по себе эти генераторы не пригодны для нужд криптографии, поскольку являются предсказуемыми. Однако средством для обобщения этих конструкций является построение связи, желательно нелинейной, между двумя или более генераторами.

Большинство реальных потоковых шифров основаны на сдвиговых регистрах с линейной обратной связью [2]. Было взломано удивительно большое число генераторов на основе сдвиговых регистров, казавшихся сложными [2].

В данной работе рассматривается новый подход к построению генераторов псевдослучайных чисел, основанный на связи между несколькими линейными конгруэнтными генераторами. Для построения связей между генераторами проводится аналогия с механической моделью движения нескольких объектов с взаимодействиями. Взаимодействия определяются столкновениями.

В результате реализован вариант генератора псевдослучайных чисел на основе описанной модели. Были проведены эксперименты, подтверждающие, что распределение генерируемой последовательности близко к равномерному, проверено отсутствие некоторых зависимостей между элементами.

1 Линейный конгруэнтный генератор

Существуют следующие характеристики генераторов псевдослучайных чисел:

1. Статистические характеристики (качество распределения).
2. Криптографическая стойкость.
3. Скорость генерации.

В задачах моделирования, обычно, не важна криптографическая стойкость, но важно качество распределения случайных чисел. В криптографии же требование «качества» случайности меняется от задачи к задаче, но криптографическая стойкость является обязательным требованием.

Стандартным подходом к генерированию псевдослучайных чисел является использование линейного конгруэнтного метода. Этот метод заключается в вычислении членов линейной рекуррентной последовательности над конечными кольцами или полями Z_m (т.е. по модулю некоторого натурального числа m). Рекуррентная последовательность задаётся следующей формулой:

$$x_{n+1} = ax_n + c \ \ (mod\ m)$$

Рекуррентное соотношение k-го порядка:

$$x_{n+k} = a_1 x_{n+k-1} + a_2 x_{n+k-2} + \cdots + a_k x_n + c \ \ (mod\ m)$$

В случае, когда в качестве поля используется Z_2, такие соотношения называются регистрами сдвига с линейной обратной связью.

Такой способ является быстрым и, при некоторых значениях параметров, достаточно качественным, но генерируемая последовательность чисел является предсказуемой, т.е. не обладает криптографической стойкостью, что не позволяет использовать этот метод в криптографии.

Одной из схем построения генераторов является комбинация двух или более генераторов случайных чисел. Можно использовать несколько рекуррентных последовательностей и, собственно, результат получать с использованием связи между ними. Например, можно:

- выход одной последовательности отправлять на вход другой;

- по выходу одной последовательности определять, из какой другой последовательности взять очередное число;
- использовать одну случайную последовательность для изменения порядка в другой последовательности (рандомизация перемешиванием).

2 Движение материальной точки

Проведём аналогию между линейным конгруэнтным генератором и движением материальной точки.

Рассмотрим простейший пример. Пусть точка движется по окружности длины m равномерно (с постоянной скоростью v). Окружность разобьём на несколько частей и пронумеруем их. Через равные промежутки времени Δt будем определять, в какой части окружности находится точка, и номер этой части отправлять на выход генератора псевдослучайных чисел.

Равномерное движение описывается простейшим дифференциальным уравнением первого порядка:

$$\frac{dx(t)}{dt} = v$$

Пусть $x_n = x(n \cdot \Delta t)$. Тогда получим простейшее рекуррентное соотношению первого порядка:

$$x_{n+1} = x_n + v \cdot \Delta t$$

Оно описывает линейный конгруэнтный генератор с коэффициентами:

$$a = 1$$

$$c = v \cdot \Delta t$$

Если материальная точка движется под действием силы, то, согласно закону Ньютона, её движение описывается дифференциальным уравнением второго порядка:

$$\frac{d^2x(t)}{dt^2} = \frac{F}{m}$$

Аппроксимируем оператор дифференцирования разностной схемой с шагом h:

$$\frac{x(t+2h) - 2x(t+h) + x(t)}{h^2} = \frac{F}{m}$$

Это соответствует рекуррентному соотношению второго порядка:

$$x_{n+2} = 2x_{n+1} - x_n + \frac{F}{m}h^2$$

А оно связано с линейным конгруэнтным генератором второго порядка с коэффициентами:

$$a_1 = 2;\ \ a_2 = -1;\ \ c = \frac{F}{m}h^2$$

Для того что бы смоделировать связь нескольких генераторов псевдослучайных чисел, можно разместить в пространстве несколько точек и определить взаимодействие между ними. В качестве выхода такого генератора можно использовать номер области, в которой находится одна из точек.

3 Физическая модель

Рассмотрим следующую физическую модель.

Пусть в двумерном пространстве (на плоскости) расположены объекты двух типов. Объекты первого типа, назовём их «фишками», – это подвижные круги, имеющие следующие характеристики (см. рис. 1):

a) постоянные характеристики:
   1) масса m;
   2) радиус r;
   3) расстояние между геометрическим центром и центром масс b;
b) характеристики начального состояния:
   1) позиция (центра масс) $\vec{p}$;
   2) скорость (центра масс) $\vec{v}$;
   3) угол поворота (вокруг оси, перпендикулярной плоскости и проходящей через центр масс) φ;
   4) угловая скорость ω.

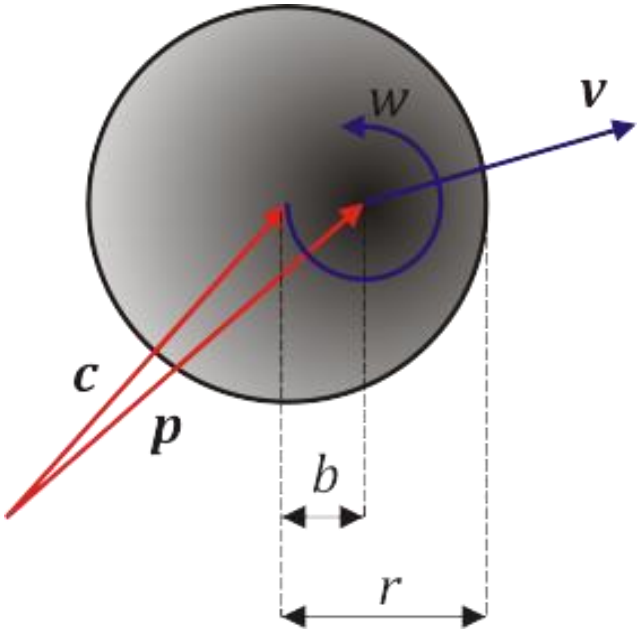

Рисунок 3.1. Схема «фишки»

Описание структуры данных фишки на языке C++ приведено в Приложении А.

Объекты второго типа, назовём их «стенками», – это неподвижные объекты со следующими постоянными характеристиками (см. рис. 2):

1) позиция первого конца отрезка $\overrightarrow{p_1}$;
2) позиция второго конца отрезка $\overrightarrow{p_2}$;
3) толщина отрезка d.

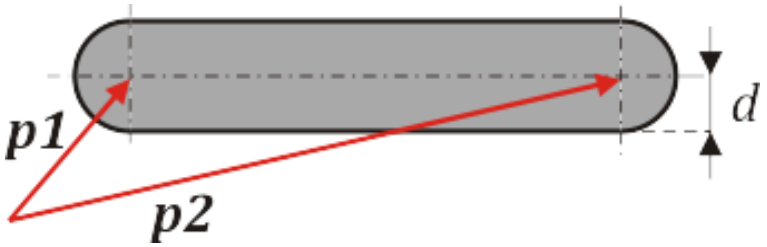

Рисунок 3.2. Схема «стенки»

Описание структуры данных стенки на языке C++ приведено в Приложении Б.

Стенки неподвижны. Они ограничивают замкнутое пространство на плоскости (многоугольник, полигон), в котором находятся фишки. Дополнительные стенки также могут находиться внутри этого замкнутого пространства.

Фишки двигаются и сталкиваются друг с другом и со стенками. Между столкновениями фишки двигаются поступательно с постоянной скоростью и вращаются вокруг своего центра масс с постоянной угловой скоростью.

Алгоритм обработки столкновения между фишками на языке C++ приведён в Приложении В. Алгоритм обработки столкновения между фишкой и стенкой на языке C++ приведён в Приложении Г.

В данной модели остаётся ещё множество варьируемых параметров. Среди них:

1) упругость столкновений;
2) сила трения между фишками и плоскостью;
3) внешние воздействия;
4) количество фишек и их характеристики;
5) наличие или отсутствие смещения центра масс относительно геометрического центра у фишек;
6) форма замкнутой области на плоскости, ограниченной стенками;
7) наличие или отсутствие стенок внутри ограниченной области и их расположение.

Чтобы система была замкнутой можно установить:

1) абсолютно-упругие столкновения;
2) отсутствие трения;
3) отсутствие внешних воздействий.

Движение данной системы, если его смоделировать и визуализировать, выглядит достаточно «случайным». (Оно похоже на броуновское движение.)

На основе представленной физической модели можно построить генератор псевдослучайных чисел. Для этого нужно:

- Выбрать полигональную область, например, квадрат.
- Выбрать количество и характеристики фишек. Например, 32 одинаковые фишки такого радиуса, что занятая ими площадь составляет ¼ от площади квадрата.
- Выбрать смещение центра масс фишек, например, одинаковое для всех фишек, равное ½ их радиуса.
- Определить, что считать выходом генератора. Например, разделить замкнутую область на «квадраты». Выбрать первую фишку «текущей». Каждую единицу времени подавать на выход генератора номер «квадрата», в котором находится геометрический центр «текущей» фишки. Затем, «текущей» выбирать следующую по порядку фишку.
- Задать случайное начальное состояние генератора – начальные позиции и скорости фишек.

Как было показано выше, равномерное движение центра масс фишки соответствует простому линейному конгруэнтному генератору. Каждая фишка представляет свой генератор. А взаимодействие между объектами модели соответствует связи между несколькими генераторами. По отдельности каждый генератор обладает очень плохими характеристиками, но благодаря связи многих генераторов, получается совершенно другой результат.

4 Энтропия

Рассмотрим подробнее описанный выше пример генератора псевдослучайных чисел. Необходимо проверить его качество. Базовой характеристикой распределения генерируемой последовательности является энтропия.

Пусть, квадрат, внутри которого находятся фишки, разбит равномерной сеткой на 16 квадратов. Тогда на выходе генератора получаются целые числа от 0 до 15. Сгенерировав выборку достаточно большого объёма N, можно подсчитать a_i $(i = 0..15)$ – количество вхождений символа i в получившуюся выборку. Тогда вероятности встретить символ i в выборке:

$$p_i = \frac{a_i}{N}$$

И энтропия:

$$H = -\sum_{i=0}^{15} p_i \log_2 p_i$$

Значения энтропии H близкие к максимуму $\log_2 16 = 4$ подтверждают равномерность исследуемого распределения.

Эксперименты проводились над выборками размером $\sim 35 \cdot 10^6$ и $\sim 70 \cdot 10^6$ чисел. В результате экспериментов были получены значения энтропии $3{,}999$ (с точностью до трёх знаков после запятой).

Для проверки генератора на отсутствие некоторых зависимостей в выходной последовательности можно вычислить так называемую *условную энтропию n-го порядка*, характеризующую вероятность появления символа при фиксированных (n-1) предыдущих символах. Обозначим a_i^j $(i = 0..15;\ j = 0..15)$ – количество вхождений символа i сразу после символа j в выборку. Тогда условная энтропия второго порядка:

$$H_j = -\sum_{i=0}^{15} p_i^j \log_2 p_i^j,$$

где $p_i^j = \frac{a_i^j}{\sum_{i=0}^{15} a_i^j}$

Обозначим $a_i^{j_1 \dots j_n}$ $(n \in \mathbb{N};\ i = 0..15;\ j_k = 0..15,\ k = 1..n)$ – количество вхождений последовательности $(j_1; \dots;\ j_n;\ i)$ в выборку. Тогда условная энтропия (n+1)-го порядка:

$$H_{j_1 \dots j_n} = -\sum_{i=0}^{15} p_i^{j_1 \dots j_n} \log_2 p_i^{j_1 \dots j_n},$$

$$\text{где } p_i^{j_1 \dots j_n} = \frac{a_i^{j_1 \dots j_n}}{\sum_{i=0}^{15} a_i^{j_1 \dots j_n}}$$

В результате экспериментов для условных энтропий вплоть до третьего порядка были получены значения, близкие к максимальному. Значения условных энтропий большего порядка неинформативны ввиду недостаточного размера выборок.

Таким образом, распределение генерируемой последовательности достаточно близко к равномерному, а зависимость элемента последовательности от одного или нескольких предыдущих отсутствует.

Эти же тесты были применены к генератору, основанному на модели, в которой центр масс фишек совпадает с геометрическим центром. Были также получены хорошие результаты. Размером выборки составлял $\sim 390 \cdot 10^6$ элементов.

ЗАКЛЮЧЕНИЕ

Целью работы являлось построение генератора псевдослучайных чисел с использованием физической модели движения нескольких объектов с взаимодействиями в виде столкновений.

Была описана физическая модель, проведена аналогия между ней и системой линейных конгруэнтных генераторов. В результате, на основе данной модели разработан и реализован вариант генератора псевдослучайных чисел. Также проведены эксперименты, подтверждающие, что распределение генерируемой последовательности близко к равномерному, и показано отсутствие зависимости значения элемента последовательности от одного или нескольких предыдущих.

Полученный генератор псевдослучайных чисел может применяться в задачах криптографии.

В описанной модели есть множество варьируемых параметров. Направлением для дальнейшей работы является исследование характеристик генератора в зависимости от параметров модели.

Литература

1. Кнут Д. Искусство программирования для ЭВМ. Т.2. Получисленные алгоритмы: Пер. с англ. — М.: Мир, 1976. — 734 с.

2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с.

3. Eberly, David H.; Ken Shoemake (contributor). Game Physics. — Amsterdam; Boston: Elsevier/Morgan Kaufmann. ISBN 1-55860-740-4.

Приложение А

(справочное)

Структура данных фишки. Моделирование свободного движения

```
struct __dib
{
      double r;             // [input field]
      double mass;          // [input field]
      double massinv;       // [calculated field]
      MyMat3 jbody;         // [calculated field]
      MyMat3 jbodyinv;      // [calculated field]

      // position of mass center
      MyVec3 position;      // [input field]

      MyVec3 velocity;      // [input field]
      MyVec3 w;             // [calculated field]

      MyMat3 R;             // [temporary field]
      MyQuat q;             // [calculated field]
      double cdist;         // [input field]

      // position of geometric center (relative to mass center)
      MyVec3 center;        // [calculated field]



      // calculate “calculated” fields
      // (input fields must be initialized before!)
      void CalcFields(double w = 0 , double fi = 0);

      void Update(float dt);                // dt -> 0
      void UpdateAccurately(float dt);      // dt is unconditioned

};
```

```
void __dib::CalcFields(double w, double fi)
{
	assert(cdist < r);

	this->w = MyVec3(0,0,w);
	this->massinv = 1/this->mass;

	this->center = MyVec3(this->cdist, 0, 0);
	this->q.from_axis_angle_unit(MyVec3::UNIT_Z, fi);
	this->q.stabilize_length();
	this->center = this->q.rotate(this->center);

	double r = this->r;
	double l = this->cdist;
	double t = this->mass / (3.0f*r);
	double m1 = t * (r + 2.0f*l),
	       m2 = t * (r - l);
	double x1 = l - r,
	       y1 = 0,
	       z1 = 0,
	       x2 = l + r*0.5f,
	       y2 = r*0.86602540378443864676372317075294f /*sqrt(3)/2*/,
	       z2 = 0;

	double Ixx = m1*(MyMath::sqr(y1)+MyMath::sqr(z1))
	             + m2*(MyMath::sqr(y2)+MyMath::sqr(z2)),
	       Iyy = m1*(MyMath::sqr(x1)+MyMath::sqr(z1))
	             + m2*(MyMath::sqr(x2)+MyMath::sqr(z2)),
	       Izz = m1*(MyMath::sqr(x1)+MyMath::sqr(y1))
	             + m2*(MyMath::sqr(x2)+MyMath::sqr(y2)),
	       mIxy = - m1*x1*y1 - m2*x2*y2,
	       mIxz = - m1*x1*z1 - m2*x2*z2,
	       mIyz = - m1*y1*z1 - m2*y2*z2;

	this->jbody = MyMat3( Ixx, mIxy, mIxz,
	                      mIxy,  Iyy, mIyz,
	                      mIxz, mIyz,  Izz);
	this->jbodyinv = this->jbody.inversed();
}
```

```
void __dib::Update(float dt)
{
	this->position += this->velocity * dt;

	this->q += MyQuat(this->w) * this->q * (0.5*dt);
	this->q.stabilize_length();

	this->center = this->q.rotate(MyVec3(this->cdist,0,0));
}

void __dib::UpdateAccurately(float dt)
{
	this->position += this->velocity * dt;

	MyQuat q1;
	double w_len = this->w.Length();
	if (!EQUAL_ZERO(w_len))
	{
		q1.from_unitaxis_angle_unit(this->w/w_len, w_len*dt);
	this->q *= q1;
		this->center = this->q.rotate(MyVec3(this->cdist,0,0));
	}
}
```

Приложение Б

(справочное)

Структура данных стенки

```
struct __point
{
	MyVec3 pos;
};

struct __wall
{
	__point *point1;
	__point *point2;
	double width;

	MyVec3 normal;  // [calculated field]
	double D;       // [calculated field]



	// calculate calculated fields
	// (other fields must be initialized before!)
	void CalcFields();
};

struct __table
{
	std::vector<__point> point;
	std::vector<__wall> wall;
};
```

Приложение В

(справочное)

Алгоритм обработки столкновения двух фишек

```
void ProcessCollision(__dib *dib1, __dib *dib2)
{
     MyVector c1 = dib1->position + dib1->center,
              c2 = dib2->position + dib2->center;
     dib1->q.to_matrix(dib1->R);
     dib2->q.to_matrix(dib2->R);
     MyMat3 jAinv = dib1->R * dib1->jbodyinv * dib1->R.transposed(),
            jBinv = dib2->R * dib2->jbodyinv * dib2->R.transposed();

          // compute N and P
     MyVector N = (c1-c2);
     //MyVector P = c2 + N * dib2->r / (dib1->r+dib2->r);
     N.Normalize();
     P1 = c1 - N * dib1->r;
     MyVector P2 = c2 + N * dib2->r;

          // compute impulse force
     const double e = 1.0f; // coeffincient of restitution
     MyVector rA = P1 - dib1->position,
              rB = P2 - dib2->position;
     MyVector kA = rA.Cross(N),
              kB = rB.Cross(N);
     MyVector uA = jAinv * kA,
              uB = jBinv * kB;

     double fNumer = -(1+e) * ( N.Dot(dib1->velocity - dib2->velocity)
                   + kA.Dot(dib1->w) - kB.Dot(dib2->w) );
     double fDenom = dib1->massinv + dib2->massinv
                   + kA.Dot(uA) + kB.Dot(uB);
     double f = fNumer / fDenom;
     MyVector impulse = N * f;

          // apply impulse force to bodies
          // to change linear/angular momentum
     dib1->velocity += impulse * dib1->massinv;
     dib2->velocity -= impulse * dib2->massinv;
     dib1->w += uA * f;
     dib2->w -= uB * f;
}
```

Приложение Г

(справочное)

Алгоритм обработки столкновения фишки со стенкой

```
void ProcessCollision(__dib *dib1, __wall *wall)
{
	MyVector c1 = dib1->position + dib1->center;
	dib1->q.to_matrix(dib1->R);
	MyMat3 jAinv = dib1->R * dib1->jbodyinv * dib1->R.transposed();

		// compute N and P
	MyVector N = wall->normal;
	double pn = dib1->position.Dot(N);
	if (pn+wall->D < 0)
		N = -N;
	N.Normalize();
	P1 = c1 - N * dib1->r;

		// compute impulse force
	const double e = 1.0f; // coeffincient of restitution
	MyVector rA = P1 - dib1->position; // rB = 0
	MyVector kA = rA.Cross(N); // kB = 0
	MyVector uA = jAinv * kA; // uB = 0


	double fNumer = -(1+e) * ( N.Dot(dib1->velocity)
				+ kA.Dot(dib1->w) );
	double fDenom = dib1->massinv + kA.Dot(uA);
	double f = fNumer / fDenom;
	MyVector impulse = N * f;

		// apply impulse force to bodies
		// to change linear/angular momentum
	dib1->velocity += impulse * dib1->massinv;
	dib1->w += uA * f;
}
```