Симулација физике у 2D простору

Лазар Магазин SV 25/2020

Факултет техничких наука
Универзитет у Новом Саду

3. februar 2022.

Садржај

Увод

Задатак:

- моделовати неке од физичких закона кретања
- за крута тела
- у дводимензионом простору

Проблеми:

- како представити физичке величине
- како представити физички систем
- како решити диференцијалне једначине
- како детектовати сударе тела
- како ограничити кретање тела
- како реаговати нумеричке грешке

Физичке величине

Скалари: маса, момент инерције, коефицијенти - `double`

Вектори: положај, брзина, импулс, сила

```
struct Vec2 {
    double x, y;
}; // utility/mathutil.h
```

Матрица: ротациона матрица

```
struct Mat2x2 {
    double m00, m10, m01, m11;
}; // utility/mathutil.h
```

Физички систем - облик и тело

Облик - особине крутог тела ван простора

```
struct Shape {
    double radius;                    // За кругове
    std::vector<Vec2> vert, norm; // За многоуглове
}; // naphy/shape.x
```

Тело - облик у простору

```
struct PhysBody {
    Shape shape;
    Vec2 pos, vel, force;
    double ang, angvel, torque;
    double m, I, m_inv, I_inv;
}; // naphy/physbody.h
```

Физички систем - сцена

Сцена - физички систем

```
struct PhysScene {
    std::vector<PhysBody> bodies;     // Тела
    std::vector<Arbtier> arbiters;    // Арбитери
    std::vector<Spring> springs;      // Опруге
}; // naphy/physscene.h
```

Једначине кретања (2. Њутнов закон)
Симплектички Ојлеров метод

```
vel += acc * dt; // Експлицитно
pos += vel * dt; // Имплицитно
```

Друге методе (вишекорачне Адамс-Башфорта) прецизније али исте стабилности
Симпл. Ојлер је углавном довољан*

Детекција колзије

круг-круг - поређење растојање са збиром полупречника
круг-многоугао - *SAT*: нормала минималног пресека пројекција
многоугао-многоугао - *SAT*

Оптимизација - екстремна тачка (*support point*)
Исецање многоуглова - *Sutherland-Hodgman* алгоритам

Детекција колзије - арбитер

Арбитер - све информације о колизији између два тела

```
struct Arbiter {
    PhysBody *A, *B;
    double depth;
    Vec2 normal;
    std::vector<Vec2> contact;
}; // naphy/arbiter.h
```

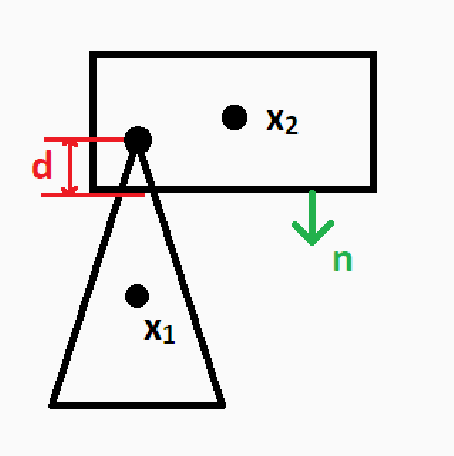
Ограничења (constraints)

$C(x) = 0$
Ограничавамо положај и оријентацију: $C(x) = C(p, r)$
Ограничење упада - потребна су два тела: $C(x) = C(p_1, r_1, p_2, r_2)$

Решавање по изводу: $C(x) = 0 \Rightarrow \dot{C}(x) = 0$

Ограничења - non-penetration constraint

Ако $\dot{C} \neq 0$, потребно је деловати на тела силом



$$C(p_1, r_1, p_2, r_2) = d = n \cdot (x_2 - x_1)$$

Ограничења - non-penetration constraint

$$
\begin{aligned}
&C = (x_2 - x_1) \cdot n \\
&\dot{C} = \tfrac{d}{dt}(x_2 - x_1) \cdot n + (x_2 - x_1) \cdot \tfrac{d}{dt} n \\
&= (\tfrac{d}{dt} x_2 - \tfrac{d}{dt} x_1) \cdot n + (x_2 - x_1) \cdot \tfrac{d}{dt} n \\
&= (v_2 + \omega_2 \times r_2 - v_1 - \omega_1 \times r_1) n + (x_2 - x_1) \cdot \tfrac{d}{dt} n
\end{aligned}
$$

Интересује нас само брзина, па се $(x_2 - x_1) \cdot \tfrac{d}{dt} n$ игнорише:

$$
\begin{aligned}
&= (v_2 + \omega_2 \times r_2 - v_1 - \omega_1 \times r_1) n \\
&= v_2 n + (\omega_2 \times r_2) n - v_1 n - (\omega_1 \times r_1) n
\end{aligned}
$$

Важи особина мешовитог производа: $(a \times b) \cdot c = a \cdot (b \times c)$:

$$
\begin{aligned}
&= v_2 n + \omega_2 (r_2 \times n) - v_1 n - \omega_1 (r_1 \times n) \\
&= \begin{bmatrix} -n & -(r_1 \times n) & n & (r_2 \times n) \end{bmatrix} \begin{bmatrix} v_1 & \omega_1 & v_2 & \omega_2 \end{bmatrix}
\end{aligned}
$$

Ограничења - non-penetration constraint

$$\dot{C} = \begin{bmatrix} -n & -(r_1 \times n) & n & (r_2 \times n) \end{bmatrix} \begin{bmatrix} v_1 & \omega_1 & v_2 & \omega_2 \end{bmatrix} = J \cdot V$$

Ако $\dot{C} \neq 0$, онда постоји ΔV такво да је $\dot{C} = J(V + \Delta V) = 0$
Занима нас импулс* ($p = m\Delta v, L = r \times p$)
$\Delta V = \begin{bmatrix} -\frac{p}{m_1} & -\frac{r_1 \times p}{I_1} & \frac{p}{m_2} & \frac{r_2 \times p}{I_2} \end{bmatrix}$
Знамо правац импулса (n), интензитет је непознат: $p = \lambda n$
$\Delta V = \begin{bmatrix} -\frac{p}{m_1} & -\frac{r_1 \times p}{I_1} & \frac{p}{m_2} & \frac{r_2 \times p}{I_2} \end{bmatrix} =$
$\begin{bmatrix} -\frac{\lambda n}{m_1} & -\frac{r_1 \times \lambda n}{I_1} & \frac{\lambda n}{m_2} & \frac{r_2 \times \lambda n}{I_2} \end{bmatrix} =$
$\lambda \begin{bmatrix} -\frac{n}{m_1} & -\frac{r_1 \times n}{I_1} & \frac{n}{m_2} & \frac{r_2 \times n}{I_2} \end{bmatrix}$
Решити ограничење значи пронаћи λ за које $J(V + \Delta V) = 0$
* зашто импулс?

Секвенцијални импулси

Зашто импулс?

Једна колизија $\Rightarrow$ један арбитер $\Rightarrow$ једно ограничење $\Rightarrow$ једно λ
Више колизија у једном тренутку $\Rightarrow$ више λ које треба решити
Систем ограничења

(А) Глобално решавање система ограничења
(Б) Итеративно решавање система ограничења

Секвенцијални импулси - итеративни метод

Секвенцијални импулси

$\Delta V = \lambda \left[-\frac{n}{m_1} \quad -\frac{r_1 \times n}{I_1} \quad \frac{n}{m_2} \quad \frac{r_2 \times n}{I_2}\right]$
Извучемо масе:

$$\Delta V = \lambda \begin{bmatrix} \frac{1}{m_1} & 0 & 0 & 0 \\ 0 & \frac{1}{I_1} & 0 & 0 \\ 0 & 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & 0 & \frac{1}{I_2} \end{bmatrix} \begin{bmatrix} -n \\ -(r_1 \times n) \\ n \\ (r_2 \times n) \end{bmatrix} = \lambda M^{-1} J^T$$

$J(V + \Delta V) = 0$
$\Rightarrow \lambda = -\frac{JV}{JM^{-1}J^T}$

Секвенцијални импулси

$\lambda = -\frac{JV}{JM^{-1}J^T}, \quad p = m\Delta v = \lambda n$

```
// JV = C' = dv * n
Vec2 dv = (B->vel + cross(B->angvel, r2))
         -(A->vel + cross(A->angvel, r1));
Vec2 dvn = dot(dv, n);
// J M^-1 J^T
double m = (A->m_inv + r1n * r1n * A->I_inv)
          +(B->m_inv + r2n * r2n * B->I_inv);
// p = lambda * n
Vec2 impulse = (-dvn / m) * n;
A->vel -= impulse * A->m_inv;
B->vel += impulse * B->m_inv;
// naphy/arbiter.cpp :: solve(), apply_impulse()
```

Секвенцијални импулси

За мало dt је $p \approx Fdt$
Свако итерација даје мало бољу коначну брзину

1) Применити све спољашње силе (нпр. гравитација) једном:
$v_{(0)} = v_{prev} + m^{-1} F_e dt$
2) У k итерација применити импулсе ограничења: $v_{(i)} = v_{(i-1)} + m^{-1} p$

```
const unsigned iterations = 10;
for (unsigned j = 0; j < iterations; j++) {
    for (unsigned i = 0; i < scene->arbiter.size(); i++) {
        scene->arbiter[i].solve();
    }
} // naphy/physscene.cpp :: scene_update_constraints()
```

Трење

Сличан поступак, али у правцу тангенте колизије

Статичко трење и кинетичко трење

Кулонов закон: $|F_s| <= \mu |F_n|$

```
double lambda_t = - dot(dv, t) / m;
if (abs(lambda_t) > u * abs(lambda))
    lambda_t = -kfric * abs(lambda); // кинетичко
else
    lambda_t = 0;
Vec2 impulse_t = lambda * t; // статичко
A->vel -= impulse_t * A->m_inv;
B->vel += impulse_t * B->m_inv;
A->angvel -= cross(r1, impulse_t) * A->I_inv;
B->angvel += cross(r2, impulse_t) * B->I_inv;
// naphy/arbiter.cpp :: solve()
```

Опруге

Хуков закон: $F = -k\Delta x$

$\Rightarrow$ казнена функција, спољашња сила

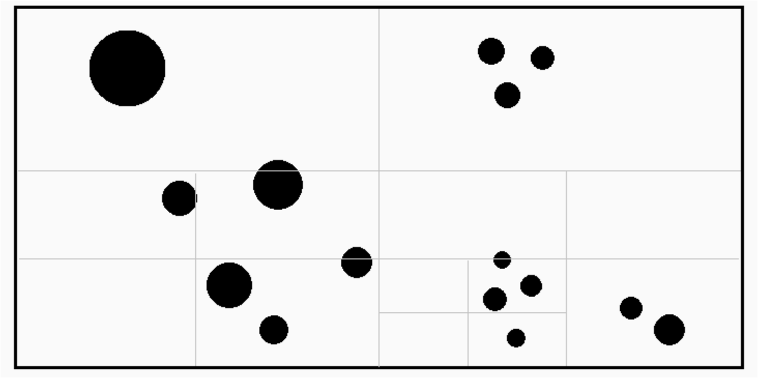
$F = (-k\Delta x) \cdot (p_A - p_B) + c \cdot (v_A - v_B)$

c - фактор пригушења

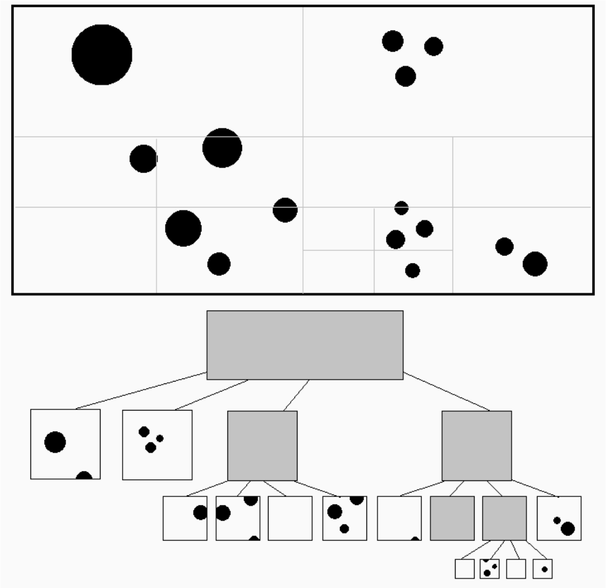
```
struct Spring {
    PhysBody *A, *B;
    double rest_length;
    double k;
    double c;
}; // naphy/spring.h
```

Broad-phase, *middle-phase*, *narrow-phase*

Квадратно стабло



Просторно индексирање - квадратно стабло



Просторно индексирање - квадратно стабло

```
struct QuadNode {
    Vec2 pos, size;
    std::vector<PhysBody*> obj;
    QuadNode* child[4];
    unsigned capacity;
}; // naphy/quadtree.h
for (QuadNode& leaf : leaves) {
    std::vector<PhysBody*>* body = leaf->object;
    for (unsigned i = 0; i < body->size; i++) {
        for (unsigned j = i + 1; j < body->size; j++) {
            PhysBody *A = body[i], *B = body[j];
        }
    }
} // naphy/physscene.cpp :: collision_quadtree()
```

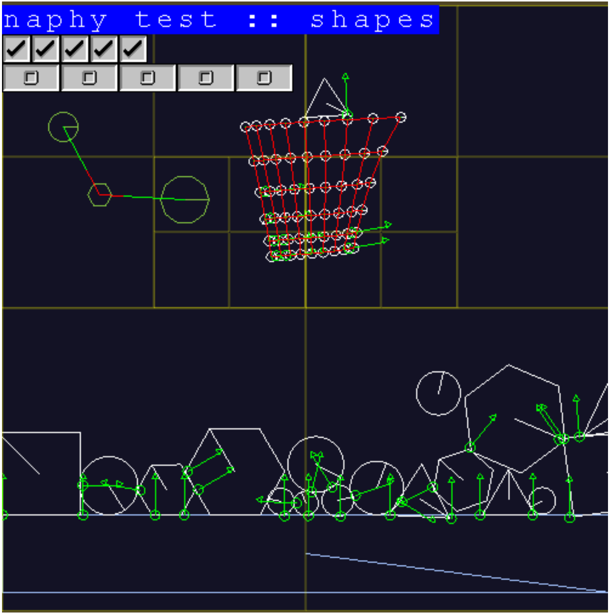
Регулисање нумеричких грешака

Коефицијент реституције: $e \in [0, 1]$
$\lambda = -(1 + e) \cdot \frac{JV}{JM^{-1}J^T}$
Баумгарте стабилизација: $\beta \in [0, 1], b = \beta \cdot \frac{C}{dt}$
$\lambda = -(1 + e + b)\frac{JV}{JM^{-1}J^T}$
Ограничење импулса: λ_{acc}: $\lambda`_{(i)} = max(\lambda_{acc} + \lambda_{(i)}, 0) - \lambda_{acc}$
Допуштен упад: деловати импулсом $\lambda_{slop} = \beta_{bias} max(C - \beta_{slop})$

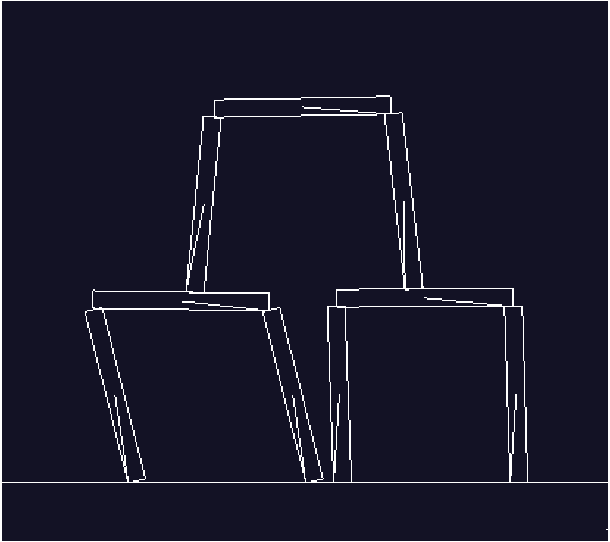
```
double slop = 0.07f;
double bias = 0.6f;
double correction = bias * max(depth - slop, 0.0);
Vec2 pcorr = n * correction / (A->m_inv + B->m_inv);

A->pos -= pcorr * A->m_inv;
B->pos += pcorr * B->m_inv;
// naphy/arbiter.cpp :: post_solve()
```

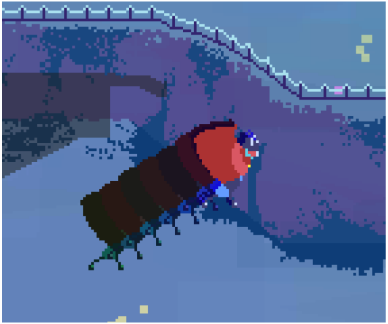
Пример - тела, арбитери, опруге, квадратно стабло

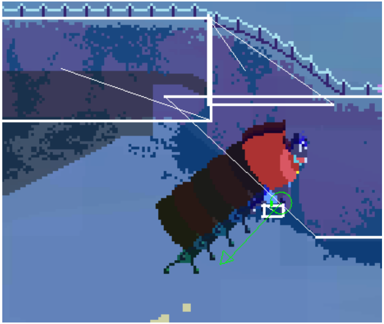


Пример - нестабилност



Пример - нешто конкретније





Литература

Erin Catto: Iterative Dynamics with Temporal Coherence

Erin Catto: Fast and Simple Physics using Sequential Impulses

Erin Catto: Modeling and Solving Constraints

ImpulseEngine

Improving the stability of your physics

Collision Response

Ming-Lun Chou: Constraints Sequential Impulse