Simulacija fizičkih interakcija u 2D prostoru

David Davidović

Matematička gimnazija

NEDELJA^{4}INFORMATIKE

29. mart 2018.

Ukratko o simulaciji fizike

- Simulacija fizičkih zakona računarski je izuzetno velika oblast
- U grubim crtama, ove simulacije se dele na
  - **Naučne**, u kojima je numerička tačnost najbitnija, i koje se koriste u praktičnoj fizici, i
  - **Interaktivne**, u kojima je najvažnije održati interaktivnost simulacije, i koje se najčešće koriste u video igrama
- Tokom ovog predavanja proći ćemo kroz osnovne korake koji su neophodni za kreiranje interaktivne simulacije fizike u 2D prostoru

Cilj: Sistem kuglica

- Želimo da napravimo simulaciju sudarajućih, identičnih loptica na koje deluje gravitacija, ignorišući ugaono kretanje
- Loptice su predstavljene kao krugovi u ravni identičnih poluprečnika
- Želimo da omogućimo lopticama da se sudaraju i odbijaju međusobno
- Želimo da na loptice deluje gravitacija
- Ne želimo da implementiramo ugaono kretanje (spin) loptica, pa ćemo taj fenomen ignorisati

Moguća upotreba

- Ovako simuliran sistem loptica se sam po sebi može koristiti kao aproksimacija fluida
- Uz ispravno iscrtavanje rezultati mogu biti vrlo realni
- Generalnije: uz dodatke nekih bitnih elemenata (kao npr. drugih oblika sem krugova, podrška za momente impulsa i ugaono kretanje), ovaj sistem se može koristiti kao jednostavan fizički *engine* za 2D video igru

Šta je stanje naše simulacije?

- Pre nego što počnemo diskusiju o algoritmima koje ćemo koristiti, moramo se složiti oko toga šta definiše stanje naše simulacije
- S obzirom na to da ovu simulaciju možemo izvesti bez pojma impulsa, ignorisaćemo ga sada
- Neka imamo n loptica. Tada, svaka loptica poseduje sledeće parametre:
  - $\vec{p_i}$ – **pozicija**. Poziciju predstavljamo kao vektor $\vec{p_i} = (x, y)$ gde su x i y koordinate loptice.
  - $\vec{v_i}$ – **brzina**.
  - $\vec{a_i}$ – **ubrzanje**.
  - m_i – **masa**.

Šta je stanje naše simulacije?

- U našem slučaju, $\vec{a}_i$ je uvek konstantno i usmereno je ka dole (gravitacija); nemamo drugog ubrzanog kretanja sem toga
- Celokupno stanje naše simulacije je, onda, skup stanja svih loptica

Šta je korak naše simulacije?

- Pošto je naša simulacija interaktivna, ona će se izvršavati u diskretnim koracima (frejmovima)
- U svakom frejmu, cilj je da na osnovu trenutnog stanja u trenutku t i nekog datog vremena Δt izračunamo stanje u trenutku $t + \Delta t$ — ovo se zove “korak” simulacije
- Podkoraci koji su nam neophodni za jedan “korak” su:
  - **Integracija**. Na osnovu ubrzanja, računamo novu brzinu, a na osnovu brzine, računamo novu poziciju
  - **Detekcija sudara**. Pronalazimo sve parove loptica koje se preklapaju (odnosno, sudarile su se).
  - **Simulacija sudara**. Računamo nove vrednosti za stanje svih loptica koje su se sudarile.

Iscrtavanje i interaktivnost

- Simulacija ne vredi mnogo ukoliko ne možemo da je vidimo!
- Uz interaktivnu simulaciju fizike uvek postoji i interaktivno iscrtavanje trenutnog stanja, uz mogućnost uticanja na scenu pomoću nekog ulaza
- Moramo obezbediti harmoničan odnos između iscrtavanja scene i simulacije, pa ćemo se ukratko pozabaviti i time

Integracija: šta želimo da postignemo?

- Potrebna nam je procedura kojom ćemo, na osnovu parametara $\vec{v}_t$, $\vec{a}_t$ i $\vec{p}_t$ i datog Δt izračunati parametre $\vec{v}_{t+\Delta t}$, $\vec{a}_{t+\Delta t}$, i $\vec{p}_{t+\Delta t}$
- Iako ovo deluje jednostavno, postoje određeni problemi kojih moramo da budemo svesni

Naivni pristup: Ojlerova integracija

- **Ojlerova integracija** je intuitivna metoda koja odmah pada na pamet kada je u pitanju ovaj problem:
  - `loptica.x += loptica.v * dt`
  - `loptica.v += loptica.a * dt`
  - Ovo se zove *eksplicitna Ojlerova integracija*.
- Bolja tačnost bi se dobila kada bi se novoizračunata brzina koristila za izračunavanje nove pozicije:
  - `loptica.v += loptica.a * dt`
  - `loptica.x += loptica.v * dt`
  - Ovo se zove *polu-implicitna* ili *simplektička Ojlerova integracija*.

Problem sa Ojlerovom integracijom

- Tokom jednog Δt, brzina nije konstantna, jer ubrzanje utiče na nju!
- Uprkos tome, prilikom Ojlerove integracije, mi je tretiramo kao da jeste, jer za ceo korak dužine Δt koristimo jednu vrednost za brzinu
- Zbog ovakvog pristupa, naša simulacija vremenom odstupa od idealnih vrednosti
- Ovo može, ali ne mora biti problem — sve zavisi od dužine tipičnog Δt i reda veličine vrednosti sa kojima radimo

Bolji pristup: RK4

- **RK4**, skraćeno od *Runge–Kutta metod četvrtog reda*, je metod integracije koji uzima u obzir vrednost izvoda funkcije u različitim trenucima tokom jednog koraka
- U našem slučaju, izvod funkcije pozicije je funkcija brzine
- Predstavimo stanje loptice kao S i definišemo funkciju $a(t, S)$ kao ubrzanje u trenutku t za stanje S.

RK4 integracija

- Izračunajmo, sada, sledeće koeficijente:

$$k_1 = a(t, S) \tag{1}$$

$$k_2 = a(t + \frac{\Delta t}{2}, S + \Delta t \frac{k_1}{2}) \tag{2}$$

$$k_3 = a(t + \frac{\Delta t}{2}, S + \Delta t \frac{k_2}{2}) \tag{3}$$

$$k_4 = a(t + \Delta t, S + \Delta t k_3) \tag{4}$$

- Tada preciznu vrednost stanja nakon koraka, S', možemo dobiti kao:

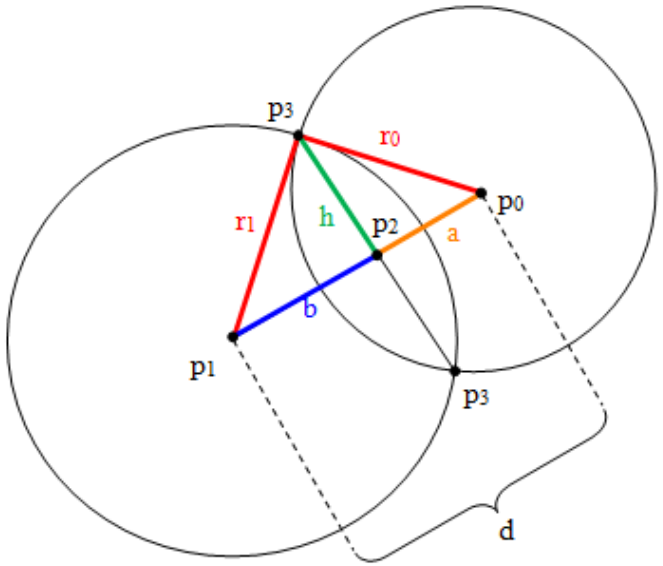
$$S' = S + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Zašto je važna efikasna detekcija sudara?

- U ovako simuliranom fizičkom sistemu se može dogoditi veliki broj sudara u jednom frejmu
- Ukupan broj sudara je, u najgorem slučaju, $O(n^2)$
- Ipak, u većini slučajeva, broj sudara u svakom frejmu će biti dosta niži od teoretskog maksimuma
- Kako sistem mora biti interaktivan, ne sme se previše vremena utrošiti na utvrđivanje koji se objekti međusobno sudaraju

Preklapanje dva kruga

- Kako bismo proverili bilo koji sudar, moramo da znamo način da proverimo da li se dva kruga seku
- Na slici možemo videti da je potreban i dovoljan uslov za ovo da *razdaljina između njihovih centara bude manja od zbira njihovih poluprečnika*
- Formalno, za loptice sa pozicijama $\vec{p_1}$ i $\vec{p_2}$ i poluprečnicima r_1 i r_2 respektivno, presek postoji ako i samo ako važi $|\vec{p_1} - \vec{p_2}| < r_1 + r_2$



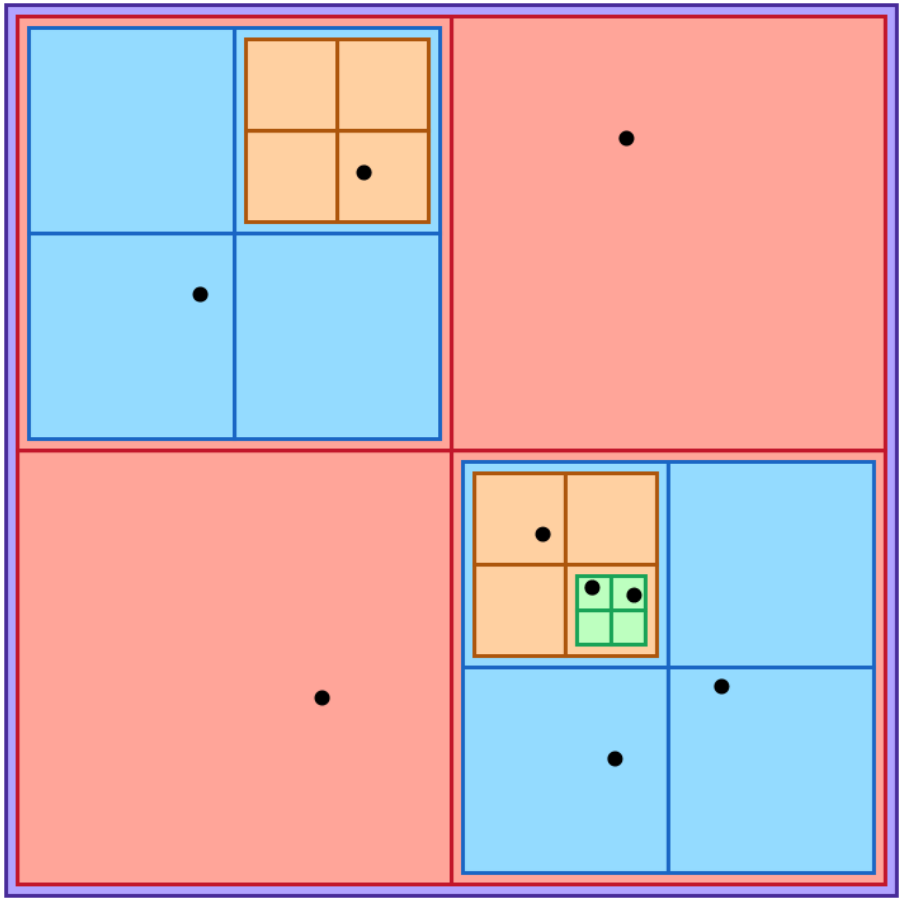
Prvi pokušaj: svaki-sa-svakim

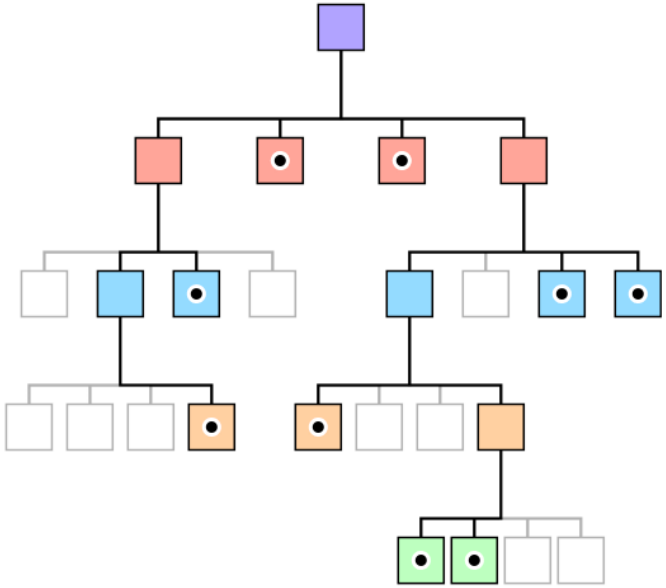
- Logična ideja je da prođemo kroz sve moguće parove loptica i proverimo seku li se
- Složenost ovog pristupa je $O(n^2)$
- U većini slučajeva, ovo je više nego zadovoljavajuće vreme
- Ipak, ukoliko postoji mnogo loptica (n je veliko), potrebno je iskoristiti malo prefinjeniji pristup kako bismo dobili zaodovoljavajuće performanse

Bolji pristup: Quadtree

- **Quadtree** je struktura podataka, korisna za podatke u prostoru (baš kao što su naši), koja omogućava efikasno ubacivanje i brisanje, kao i odgovore na upit “sa kojim objektima se dati objekat može seći?”
- Quadtree radi tako što rekurzivno particioniše ravan u 4 kvadranta, sve dok kvadranti ne budu dovoljno mali (manji od neke predefinisane, konfigurabilne veličine)
- Ovako dobijena struktura je stablo — loptice se mogu ubaciti u čvorove ovog stabla u zavisnosti od toga gde se prostorno nalaze
- Tada, loptica A će moći da se sudara samo sa lopticama koje se nalaze u čvorovima za kvadrante unutar kojih se A nalazi
- Visina ovakvog stabla raste logaritamski sa prostorom u ravni koji obuhvata

Bolji pristup: Quadtree





Quadtree: Insert

- Operaciju ubacivanja loptice A u stablo možemo definisati rekurzivno
- Neka svaki čvor *quadtree*-a sadrži B, pravougaonik oblasti, i L, listu loptica koje su pridružene njemu
- Kada se ubacuje loptica u čvor, proveravaju se B vrednosti za sva 4 deteta čvora. (Ukoliko čvor nema dece, u pitanju je list, pa se u njegovo L samo ubacuje A i algoritam je gotov.) Razlikujemo dva disjunktna slučaja:
  - **Loptica je u potpunosti sadržana u nekom od dece.** Pozivamo funkciju rekurzivno za to dete.
  - **Loptica seče neku od pravih koje dele čvor na 4 dela.** Loptica se dodaje u L trenutnog čvora i algoritam je gotov.

Quadtree: Query

- Operaciju dobijanja skupa kandidata za sudar za lopticu A iz stabla takođe možemo definisati rekurzivno
- Kada se proveravaju kandidati za čvor i lopticu A, razlikujemo dva disjunktna slučaja:
  - **Ako je čvor list**, rezultat je prazan skup u slučaju da A ne dodiruje B tog čvora, ili L tog čvora u suprotnom.
  - **Ako čvor ima decu**, rezultat je unija L tog čvora i vrednosti funkcije za sve čvorove decu čije B pravougaonike dodiruje A.

Detekcija sudara pomoću *quadtree*-a

- Kako bismo detektovali sudare, prosto prolazimo kroz sve loptice i za svaku od njih vršimo *query* operaciju da dobijemo listu kandidata
- Kada imamo listu kandidata trenutnu lopticu A_i, izvršimo test sečenja između A_i i svih kandidata
- Kada neka loptica promeni poziciju, obrišemo je iz *quadtree*-a i dodamo ponovo sa novom pozicijom
- Pod uslovom da je scena dinamična, u proseku sve ove operacije će biti linearne u odnosu na visinu stabla

Šta se dešava tokom sudara?

- Sa fizičke strane, ovo je veoma kompleksna interakcija
- ...ali, u svrhu jednostavnosti, pretpostavićemo da se dešavaju samo elastični sudari između loptica
- Ovo nam omogućava da ne brinemo o raznim detaljima neelastičnih sudara, i čini lakšim da ignorišemo ugaono kretanje

Elastični sudar u jednoj dimenziji

- Ukoliko se prisetimo zakona održanja impulsa i zakona održanja energije, možemo to iskoristiti za izvođenje formuli koje diktiraju kako se brzine menjaju nakon elastičnog sudara
- Posmatrajmo dve loptice u jednoj dimenziji, masa m_1 i m_2 respektivno, koje su pre sudara imali brzine $\vec{v_1}$ i $\vec{v_2}$. Označimo sa $\vec{v_1'}$ i $\vec{v_2'}$ brzine ovih loptica nakon sudara
- Na osnovu istovetnosti ukupnog impulsa, imamo:

$$m_1\vec{v_1} + m_2\vec{v_2} = m_1\vec{v_1'} + m_2\vec{v_2'}$$

- Na osnovu istovetnosti ukupne kinetičke enrgije, imamo:

$$\frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2} = \frac{m_1 v_1'^2}{2} + \frac{m_2 v_2'^2}{2}$$

Elastični sudar u jednoj dimenziji

- Rešavanjem ovog sistema dobijamo jedino netrivijalno rešenje:

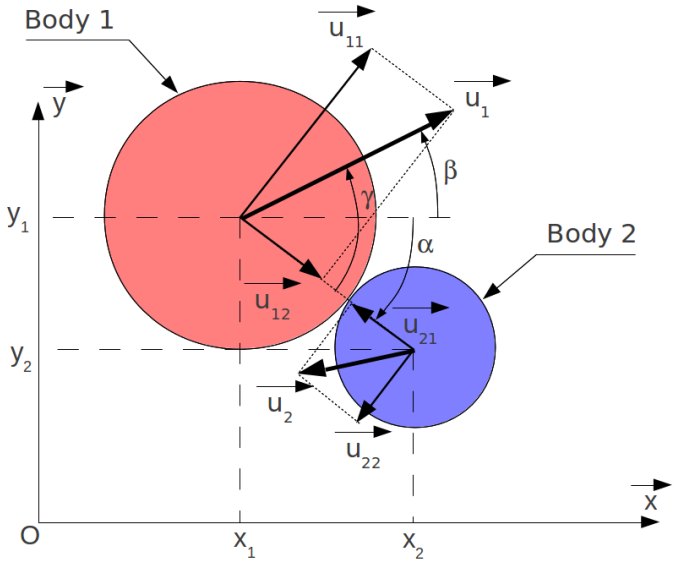
$$v_1' = \frac{v_1(m_1 - m_2) + 2m_2v_2}{m_1 + m_2} \tag{5}$$

$$v_2' = \frac{v_2(m_1 - m_2) + 2m_1v_1}{m_1 + m_2} \tag{6}$$

- Primetimo da se samo smerovi brzina menjaju, dok pravci ostaju isti (zbog toga što posmatramo jednu dimenziju)

Elastični sudar u dve dimenzije

- Mozemo izračunati normalu i tangentu na tačku preseka dveju loptica (zahvaljujući kružnom obliku koji smo odabrali, ovo je lako izvesti)
- Kada imamo normalu i tangentu, projektujemo brzine $\vec{v}_1$ i $\vec{v}_2$ na njih, da dobijemo tangencijalne i normalne komponente za obe
- Nakon toga, samo simuliramo jednodimenzionalni elastični sudar po te dve ose!



Zašto je stabilnost važna?

- **Stabilnost** se odnosi na otpor naše fizičke simulacije na greške koje potiču od nesavršenih metoda simulacije, ili predvidivost na globalnom nivou
- Postoje razne metode kojima možemo integrisati fizički korak i iscrtavanje na ekranu, i svaka ima različite karakteristike stabilnosti
- Stabilnost je bitna, na primer, kod *multiplayer* igara gde se fizika mora izvršavati lokalno, na računarima igrača, ali mora i ostati ažurna sa fizikom koju računa server

Prvi pokušaj: promenljiv korak

- Ovo je najjednostavniji pristup za sinhronizaciju iscrtanog stanja i fizičkog stanja
- Sastoji se u tome da merimo koliko je vremena prošlo od prethodnog frejma i to koristimo kao Δt za simulaciju:

```
t1 = current_time_seconds()
while true:
    t2 = current_time_seconds()
    dt = t2 - t1
    t1 = t2

    step(state, dt)
    render(state)
```

Problemi sa promenljivim korakom

- Simulacija je nepredvidiva — rezultati mogu da se izuzetno razlikuju u zavisnosti od dužine koraka
- Na bržim mašinama, manje Δt vrednosti će biti korišćenee
- Zbog ovoga, simulacija može da ima različit “osećaj” u zavisnosti od brzine računara na kome se pokreće
- Ozbiljniji problem: ako su Δt vrednosti dovoljno velike, možemo imati, na primer, tela koja prolaze kroz druga tela ili “eksplodiranje u beskonačnost”

Bolji pristup: odvajanje fizike i iscrtavanja

- Glavni problem našeg prethodnog pokušaja je što naše Δt zavisi od brzine iscrtavanja
- Novi pristup se sastoji u tome da fiksiramo Δt kako bismo imali predvidivu simulaciju, i konzumiramo vreme u komadima te veličine:

```
dt = 1.0 / 60.0
a = 0.0
t1 = current_time_seconds()
while true:
    t2 = current_time_seconds()
    a += t2 - t1
    t1 = t2

    while a >= dt:
        step(state, dt)
        a -= dt

    render(state)
```

Bolji pristup: odvajanje fizike i iscrtavanja

- Ovaj pristup omogućava da imamo simulaciju koja je uvek ista (deterministička)
- Takođe, zbog fiksiranog Δt, simulacija je predvidivija i lakše se testira
- Jedini problem: šta da radimo sa vremenom koje nam ostaje u akumulatoru `a`?

Interpolacija

- Ukoliko vreme koje je potrebno da se iscrta frejm nije tačno deljivo Δt (a praktično nikada nije), ostaje nam neko "nesimulirano" vreme na kraju svakog frejma
- Zbog ovoga, na ekranu možemo ponekad videti milisekundske "zastoje" zbog razlike u broju Δt koraka koji se svakog frejma izvršavaju
- Rešenje: interpolacija!

Interpolacija

- **Interpolacija**, u ovom kontekstu, se odnosi na kombinovanje dva stanja, S i S', na osnovu neke interpolacione konstante $\alpha \in [0, 1]$
- Definišemo to kao funkciju $f(S, S', \alpha)$
- Ideja je da je vrednost funkcije za $\alpha = 0$ jednaka S, za $\alpha = 1$ jednaka S', a za ostale vrednosti je "kombinacija" ta dva stanja
- Generalno, za pozicije loptica na ekranu, možemo koristiti *linearnu interpolaciju*: $f(\vec{p}, \vec{p'}, \alpha) = \alpha\vec{p'} + (1 - \alpha)\vec{p}$

Interpolacija

- Na osnovu preostalog vremena u akumulatoru, možemo interpolirati između *prethodnog* i *trenutnog* fizičkog stanja
- Ako definišemo $\alpha = \frac{a}{\Delta t}$ i interpoliramo između prethodnog i sledećeg stanja sa tim koeficijentom pre iscrtavanja, sredićemo diskontinuitet i “zastoje”:

```
dt = 1.0 / 60.0
a = 0.0
t1 = current_time_seconds()

while true:
    t2 = current_time_seconds()
    a += t2 - t1
    t1 = t2

    while a >= dt:
        prevState = state
        step(state, dt)
        a -= dt

    alpha = a / dt
    render(interpolate(prevState, state, alpha))
```