WebGL Boidi

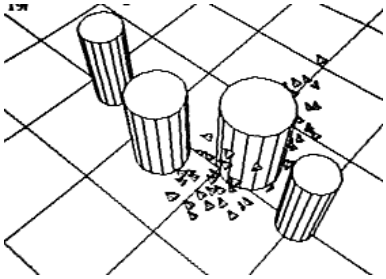
3. laboratorijska vježba

Roko Franetović

21.01.2022.

1 Boidi

Boidi su primjer sustava čestica koji emuliraju ponašanje jata, krda, i sličnih bioloških fenomena. Osnovnu ideju je prezentirao Craig Reynolds 1986. godine. Da bismo mogli replicirati spomenute prirodne fenomene, potrebno je ostvariti **koordinaciju**. Pri tome ćemo koristiti lokalni pristup da ostvarimo željeno ponašanje.



Slika 1: primjer sustava boida (1986.)

Sažeto do minimuma, svaki čestica pretražuje svoju okolinu te prilagođava svoje ponašanje prema česticama unutar svoje okoline. Ovisno o tome kako definiramo pojmove okoline čestice te kako definiramo prilagođavanje okolini možemo emulirati različite prirodne fenomene.

Kroz ovaj rad ćemo proučiti osnovnu funkcionalnost i svojstva boida, te ćemo vidjeti prednosti mane rada s WebGL-om (uspoređeno s OpenGL-om).

2 Osnovni sustav čestica

Sustav čestica ćemo definirati kao skup objekata te ćemo manipulirati individualnim svojstvima objekata kroz neki korak vremena. Ovako opisan sustav možemo prikazati sa sljedećim pseudo-kodom:

```
particle_system = {particle_1, ..., particle_n}

time_step(particle_system, dt):
  updated_particle_system = {}
  for particle in particle_system:
    updated_particle_system = update(particle, dt)
  particle_system = updated_particle_system
```

Umjesto da ažuriramo prijašnje stanje sustava čestica, mi stvaramo novo stanje želeći izbjeći probleme s redom ažuriranja čestica. Detalje oko ažuriranje čestica i crtanje sustava bit će objašnjeni u budućim poglavljima.

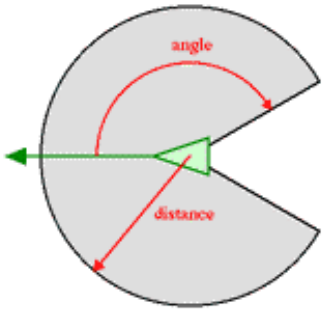
3 Okolina boida

Svaka naša čestica (zvat ćemo je boid) ima svoju okolinu, te ovisno o drugim boidima unutar njene okoline boid modificira svoje ponašanje. Koristit ćemo sljedeću analogiju za definiciju okoline: "Okolina čine svi drugi boidi koje boid može vidjeti." Prema tome definiramo okolinu:

$$nearest_b = \{b' \in boids \mid \|b - b'\| < r \wedge \angle(b, b') < \theta\}$$

$$\angle(b, b') = \arccos \frac{b \cdot b'}{\|b\| \|b'}$$

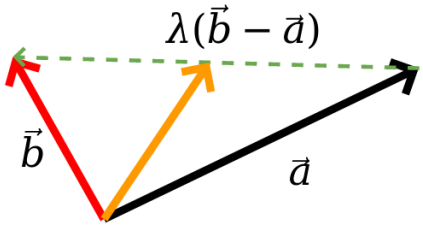
Kao udaljenost ćemo koristiti euklidsku udaljenost, te ćemo radijus i kut modificirati ovisno o željenom ponašanju sustava.



Slika 2: okolina boida

4 Mehanizam upravljanja

Ovisno o boidima unutar okoline, boid će mijenjati svoje ponašanje. Konkretno, mi ćemo mijenjati brzinu našeg boida. Da bismo ostvarili takvo ponašanje, moramo definirati mehanizam promjene brzine.



Slika 3: promjena brzine

Pretpostavimo da iz okoline dobijemo željeni vektor brzine $\vec{b}$ te označimo trenutačni vektor brzine boida sa $\vec{a}$. Vektor koji dava smjer od trenutačne brzine boida prema željenoj jest $\vec{b}-\vec{a}$. S obzirom na to da želimo kontrolirati količinu usmjerenja, dodat ćemo parametar λ te će naš boid uskladiti svoju brzinu prema željenoj na sljedeći način:

$$\vec{a'} = \vec{a} + \lambda(\vec{b} - \vec{a})$$

No ovakva promjena brzine može imati jednu neželjenu posljedicu. Vidimo da mijenjamo smjer brzine, ali također mijenjamo količinu brzine. Ovo najčešće rezultira smanjivanjem brzine, te će naši boidi nakon nekog vremena konvergirati svoje brzine u 0. Zbog toga ćemo zadržati originalni iznos brzine na sljedeći način:

$$\vec{a'} = \frac{\vec{a'}}{\|\vec{a'}\|}\|a\|$$

5 Pravila boida

Kako dobiti željenu brzinu iz okoline? To upravo određuju pravila boida, gdje svako pravilo rezultira sa svojim željenim vektorom promjene ($\lambda(\vec{b} - \vec{a})$ vektor). Osnovni sustav boida ima 3 pravila: **separacija, usklađenost, kohezija**.

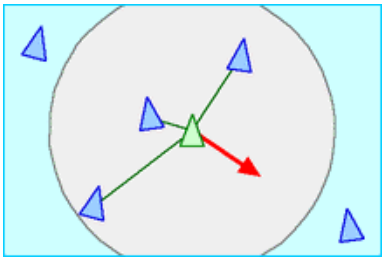
Ovisno o tome kakvo ponašanje želimo dobiti, možemo dodati nova pravila koja detaljnije opisuju utjecaj okoline. U ovom radu ćemo se držati osnovnih pravila, jer su dobar primjer lokalne jednostavnosti koja rezultira s poprilično složenim sustavom.

Nakon što za svako pravilo izračunamo vektor promjene, konačna promjena je zbroj svih izračunatih vektora. Na kraju cijelog postupka zbrajamo brzinu boida s dobivenom sumom, čime dobivamo okolina upravlja brzinom boida.

5.1 Separacija

Separacija nam modelira izbjegavanje sudara među našim boidima. Za svaki boid u našoj okolini računamo vektor prema boidu $-(\vec{b'} - \vec{b})$, gdje su $\vec{b'}, \vec{b}$ pozicije boida. Također skaliramo izračunatu razliku inverzno sa udaljenošću, tako da bliži boidi imaju veći utjecaj.

Konačna promjena je suma svih suprotnih vektora. Postupak možemo opisati sa sljedećim pseudo-kodom:

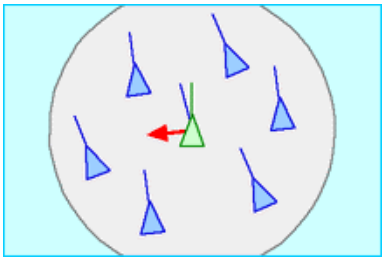


Slika 4: primjer separacije

```
separation(current, others, scale):
  change = 0
  for other in others:
    distance = euclid_distance(current.position, other.position)
    change += - (other.position - current.position) * (1 / distance)
  return change * scale
```

5.2 Usklađenost

Usklađenost nam modelira usklađivanje smjera naših boida, što stvara dojam koordinacije našeg sustava. Usklađenost se jednostavno računa kao prosječna brzina boida u okolini.

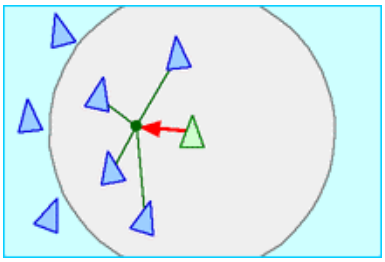


Slika 5: primjer usklađenosti

```
allignment(current, others, scale):
  change = 0
  for other in others:
    change += other.velocity
  change /= others.size
  return (change - current.velocity) * scale
```

5.3 Kohezija

Kohezija nam modelira stvaranje lokalnih grupa boida. Drugim riječima, kohezija "drži boide skupa". Ovo ostvarujemo tako da svaki boid se giba prema prosječnoj poziciji boida u okolini.



Slika 6: primjer kohezije

```
cohesion(current, other, scale):
  change = 0
  for other in others:
    change += other.position
  change /= others.size
  return (change - current.velocity) * scale
```

6 Ažuriranje čestica

Nakon što smo definirali sva pravila našeg sustava, potrebno je detaljnije opisati metodu ažuriranja čestice. U ovom radu ćemo koristiti polu-implicitni Eulerov postupak (radi jednostavnosti implementacije i dobrih rezultata).

Prvo ćemo odrediti promjenu brzine boida, što će biti suma vektora promjene svih pravila. Nakon toga jednostavno ažuriramo poziciju sa promijenjenom brzinom. Postupak možemo opisati sljedećim pseudo-kodom:

```
update(particle, dt):
  others = system_get_visible_to(particle)
  c1 = separation(particle, others, speration-scale)
  c2 = alignment(particle, others, alignment-scale)
  c3 = cohesion(particle, others, cohesion-scale)

  particle.velocity += (c1 + c2 + c3) * dt
  scale_to_original_magnitude(particle.velocity)

  particle.position += particle.velocity * dt
```

7 WebGL

WebGL je web standard za 3D grafiku baziran na OpenGL ES pod trenutačnim razvojem Khronos Group-e. Koristeći WebGL možemo grafički *pipeline* povezati s web-preglednikom koji podržava WebGL (većina modernih preglednika). Programski jezik koji koristimo za razvoj WebGL aplikacija je JavaScript, dok *shadere* pišemo u GLSL ES-u.

Slika 7: WebGL logo

WebGL se nadovezuje na trenutačni HTML5 standard te koristi `<canvas>` objekt koji nam dava sav potrebni kontekst za stvaranje aplikacije. Nakon što dobijemo dobivene kontekst iz `<canvas>` objekta, dobiveni kontekst koristimo kao u OpenGL-u. Zbog ovoga je arhitektura i razvoj WebGL aplikacija vrlo slična OpenGL aplikacijama.

Glavna prednost WebGL-a je portabilnost aplikacije. S obzirom na to da većina modernih web-preglednika podržava WebGL, razumno je očekivati da će naša aplikacija moći raditi na bilo kojem modernom računalu. Također, ne moramo se brinuti o upravljanju prozorima (koristeći npr. FreeGLUT).

Glavna mana WebGL-a su slabije performanse naspram nativne OpenGL aplikacije. Iako iscrtavanje ima jednake performanse (jer se WebGL povezuje na grafičku karticu), ostali dio koda se izvodi kroz JavaScript interpreter koji je sporiji od nativno prevedenih aplikacija.

Također ćemo koristiti glMatrix biblioteku za operacije nad matricama i vektorima.

8 Arhitektura projekta

Aplikaciju ćemo podijeliti na dva dijela: podatkovni i programski dio. Podatkovni dio će nam se sastojati od `.html` stranice gdje ćemo sve potrebne podatke zapakirati. Programski dio nam se sastoji od JavaScript datoteke u kojoj se nalaze sve potrebne instrukcije za izvođenje aplikacije.

Već možemo vidjeti prednost rada s WebGL-om: umjesto da stvaramo datotečnu strukturu preko koje organiziramo našu aplikaciju, mi možemo cijelu strukturu staviti unutar naše stranice. Konkretno, kod GLSL *shadera* i `.obj` datoteka će se nalaziti unutar skrivenih `<div>` objekata kojima ćemo pristupiti kroz naš JavaScript kod.

Ovakva struktura nije poželjna za veće projekte, ali u ovom slučaju nam olakšava razvoj i pregled projekta. Također ne moramo koristiti `http` poslužitelj da bismo izbjegli sigurnosne mjere poslužitelja za *Cross site scripting (XSS)* slabosti.

Sustav čestica, scenu i pojedinačne čestice ćemo implementirati koristeći podatkovnu strukturu rječnika. Također koristimo funkcionalnost JavaScript-a koja nam omogućuje da nam funkcije mogu biti vrijednosti unutar rječnika. Ovakav pristup nam olakšava kod tako da izbjegavamo definiranje klasičnih objekata unutar objektno orijentirane paradigme.

Za iscrtavanje `.obj` objekata potrebno je definirati VBO i IBO polja koja prosljeđujemo *fragment i vertex shaderu*. Za iscrtavanje koristimo perspektivnu projekciju, pa je potrebno definirati `mvp` matrice.

9 Korišteni materijali

- `http://www.red3d.com/cwr/boids/` - stranica Craig Reynoldsa (originalni autor) o boidima

- `https://www.khronos.org/webgl/` - službena stranica WebGL-a
- `https://glmatrix.net/` - službena stranica glMatrix biblioteke