Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Fajdiga

Plavajoča bitja v simulaciji Nvidia Flex

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Iztok Lebar Bajec

Ljubljana, 2019

Zahvala

Zahvaljujem se mentorju izr. prof. dr. Iztoku Lebarju Bajcu za vso pomoč, nasvete in potrpežljivost ter vsem, ki so me ob delu spodbujali in me bodrili.

Peter Fajdiga, 2019

Kazalo

Seznam uporabljenih oznak

oznaka	*pomen*
r_f	velikost delcev tekočin
r_s	velikost ostalih delcev
r_r	največja razdalja, na kateri deluje interakcija med delci
r_p	najmanja razdalja med delci in ravninami
m_f	masa delca tekočine
m_s	masa delcev plavajočega bitja
t	čas
n_c	število členov
l	število rezin
L	dolžina plavajočega bitja
v	hitrost plavanja
f	frekvenca plavanja
a	amplituda gibala
ϕ	faza gibala
u	skrčenost oz. raztegnjenost mišice
l	dolžina mišice
$a(x)$	amplitudna ovojnica
k	valovno število

Povzetek

Naslov: Plavajoča bitja v simulaciji Nvidia Flex

Razvili smo realno časovno simulacijo različnih plavajočih bitij v tekočini z uporabo ogrodja za fizikalno osnovano simulacijo Nvidia Flex, ki temelji na metodah pozicijske dinamike (angl. *position based dynamics*) in pozicijskih tekočin (angl. *position based fluids*). Predstavimo model telesa bitja, ki je sestavljeno iz delcev. Ti so združeni v trdne člene, ki so s pomočjo mišic gibljivi v zglobih med njimi. Z genetskim algoritmom optimiziramo delovanje mišic, da bitje čim bolj učinkovito plava naravnost v simulirani tekočini. Podamo štiri različna telesa bitij, katerim z razvitim sistemom avtomatično generiramo gibanje. Pokažemo, da je plavanje v tako simulirani tekočini mogoče, a počasno in ne ustvarja realistične turbulence. Razvito gibanje primerjamo z načini plavanja resničnih rib, in sicer *anguilliform*, *carangiform* in *thunniform*. Ujemanje z njimi je različno za podana telesa bitij.

Ključne besede

animacija, simulacija tekočin, umetno življenje

Abstract

Title: Swimming creatures in Nvidia Flex

We develop a real time simulation of varied swimming creatures in a fluid using Nvidia Flex for physically based simulation. Nvidia Flex uses position based dynamics and position based fluids to achieve real time performance. We model creatures' bodies as solid clusters of particles that are connected with joints and animated using muscles. A genetic algorithm is used to optimise muscle function in order to allow the creature as efficient forward swimming gait as possible. We present four different bodies and use our system to automatically generate their movement. We show that swimming in such a fluid is possible, albeit slow and it does not result in realistic turbulence. The evolved gaits are compared with those of real fish, namely *anguilliform*, *carangiform* and *thunniform*. The fit is different for different bodies.

Keywords

animation, fluid simulation, artificial life

Poglavje 1

Uvod

Računalniška animacija je prisotna v mnogih različnih medijih, a ne glede na to, kaj animiramo, velikokrat stremimo k temu, da je videti čim bolj realistično, zato se pogosto zatekamo k fizikalno osnovani računalniški simulaciji. Z njeno pomočjo lahko animiramo različne naravne poteke, npr. tok tekočin [1, 2, 3, 4], deformacijo mehkih teles [5, 6], razbitje krhkih teles [7] ter gibanje bitij, npr. hojo [8, 9, 10], letenje [11, 12] in plavanje [13, 14, 15, 16]. Pri načrtovanju realistične simulacije gibanja bitij si lahko pomagamo z zajemom gibanja resničnih živali [17, 18]. S tem lahko dosežemo kar dobre rezultate, a smo omejeni zgolj na bitja in gibanja, ki jih je moč posneti. Da odpravimo to omejitev, lahko gibanje definiramo ročno, vendar je na ta način zamudno in zahtevno doseči rezultat, ki izgleda realistično. Tretja rešitev je programsko generiranje gibanja, kjer določimo cilje, ki naj jih bitje doseže in optimiziramo njegovo učinkovitost doseganja teh ciljev.

Gibanje lahko optimiziramo z različnimi pristopi. Eden izmed njih je spodbujevano učenje (angl. *reinforcement learning*), kjer simulirano bitje z različnimi gibi poskuša doseči cilj ter je ob uspehu nagrajeno in ob neuspehu kaznovano. Nagrajene strategije so v naslednjih iteracijah izbrane pogosteje. Spodbujevano učenje se dobro združuje z globokimi nevronskimi mrežami

(angl. *deep reinforcement learning*) [8, 9, 19], kar omogoča tudi zahtevnejše obnašanje in uporabo kompleksnejših čutil. Drugačen pristop je parametrizacija gibanja in optimizacija parametrov npr. z metodo adaptacije kovariančne matrike [10, 13], simuliranim ohlajanjem [11] ali z evolucijskim algoritmom. Slednjega smo uporabili v tem delu.

Velika domena simuliranega gibanja je plavanje. Tu je pomembno tudi simuliranje tekočine, v kateri simulirano bitje plava, da realistično prikažemo vpliv njegovega gibanja na okolico. Take rešitve praviloma ne tečejo v realnem času. Simulacija tekočine je prav tako koristna pri simuliranju jate rib, ker lahko tako med njimi deluje hidrodinamična interakcija [20]. Eden od načinov simuliranja tekočin je z uporabo poenotene fizikalne simulacije (angl. *unified dynamics solver*), v kateri ima vsak objekt – tekočina, blago, trdno ali mehko telo – enako predstavitev. Glavna prednost tega je to, da ni treba simulirati vsakega tipa objektov ločeno, ampak imamo le en tip simulacije za različne tipe objektov, ki lahko vplivajo drug na drugega brez dodatnih korakov. V zadnjih letih so tovrstne simulacije postale popularna rešitev za posebne učinke. Med njimi so npr. Maya Nucleus [3], Softimage Lagoa in Nvidia Flex [4], ki za razliko od prvih dveh žrtvuje nekaj verodostojnosti, da lahko deluje v realnem času, zaradi česar smo ga izbrali tudi za našo simulacijo. To doseže z uporabo metode pozicijskih tekočin (angl. *position based fluids*) [2].

V tem magistrskem delu smo razvili realno časovno simulacijo plavajočega bitja v tekočini. Plavanje je parametrizirano, parametri so optimizirani z genetskim algoritmom. Cilj bitja je z interakcijo z delci tekočine doseči potisk. Razvite načine plavanja bomo primerjali z gibanjem resničnih rib. Pokazali bomo, da je plavanje v tekočini, simulirani z metodo pozicijskih tekočin, mogoče. Ker Nvidia Flex ni popolnoma realističen, bomo pokazali, kako to vpliva na razvito plavanje in kakšno plavanje je v njem najučinkovitejše.

1.1 Pregled sorodnih del

Podobne cilje našim so si zadali Tan in sod. [13], ki so optimizirali plavanje raznolikih bitij v tekočini, ki so jo simulirali z uporabo Navier-Stokes enačb. Bitja so sestavili iz več trdnih teles, povezanih z zglobi različnih tipov, ki omogočajo rotacijo okrog ene, dveh ali vseh treh osi. Gibanje telesa so opisali z naborom funkcij, ena za vsako os vsakega zgloba. Te funkcije so optimizirali z metodo adaptacije kovariančne matrike tako, da je bitje plavalo naravnost in ni porabilo več energije, kot so dovolili. Dosegli so tudi plavanje po podani poti, ki ji bitje sledi z izbiranjem med naučenimi manevri. Članek ne omenja možnosti uporabe izračunanih gibov v realno-časovni simulaciji.

Drugačen model opisuje Tu [21], kjer simulirana riba deluje kot mehko telo, sestavljeno iz vozlišč z masami. Vozlišča so razporejena v oglišča vzporednih pravokotnikov tako, da v mirujočem položaju skozi njihova središča teče ista premica. Vozlišča so povezana z vzmetmi, ki ohranjajo strukturno stabilnost telesa, a omogočajo, da se upogiba. Nekatere izmed vzmeti delujejo kot mišice – lahko se skrčijo, s čimer se spremeni oblika telesa.

Poleg simuliranih plavajočih bitij, je veliko raziskovalnega dela opravljenega tudi z robotskimi ribami [22, 23, 24, 25, 26]. Barrett in sod. [27] opisujejo robotski mehanizem v obliki tune (*Thunnus thynnus*), sestavljen iz osmih gibljivih členov in prekrit z blagom. Sistem vrvi in škripcev, ki ga poganja šest motorjev, omogoča premikanje skoraj vsakega člena tako, da levo ali desno stran njegove sprednje ploskve potegne proti sprednjem delu ribe. Na ta način delujejo vrvi podobno kot mišice v simuliranih modelih.

Vrnimo se k simuliranem gibanju. Allard [14] je simuliral avtonomno ribo v različnih situacijah. Telo ribe je modeliral kot tetraedrsko mrežo z okostjem, s kostmi razporejenimi od glave do repa in povezanimi z zglobi. Riba poskuša doseči cilj, medtem ko tekočina vpliva na njeno gibanje. Uporabil je več algoritmov za avtomatičen nadzor gibanja, ki vzamejo v poštev spreminjajoče

se tekočinske sile in temeljijo na mehki logiki. Simulacijo je mogoče poganjati v realnem času. Da je to mogoče, ne uporablja polne simulacije tekočin, zato se riba le odziva na tekočinske sile, a nima vpliva na tekočino.

Poleg rib obstaja še veliko drugih plavajočih organizmov. Cortez in sod. [16] so simulirali plavajoče celice in organizme, kot so nematode, v viskozni nestisljivi tekočini. Plavalci so elastične cevaste strukture, katerih obliko določi hidrodinamika. V tem se razlikujejo od ostalih obravnavanih del. Nematode se skozi tekočino premikajo z valovitim plavanjem, ki je podobno plavanju jegulj.

Filella in sod. [20] so modelirali skupinsko vedenje jate rib z združitvijo vedenjskih pravil in hidrodinamične interakcije. Pokazali so, da se s takim modelom pojavi faza skupinskega obračanja, da posamezna riba v jati zaradi vpliva ostalih rib na tekočino plava z višjo povprečno hitrostjo, in da tekočinski tok ojača vedenjski šum.

Metode, s katerimi se modelira plavanje, se lahko uporabi tudi za soroden problem ptičjega leta. S tem sta se ukvarjala Wu in Popović [11], ki sta ptico modelirala kot okostje, ki je gibljivo v členkih, z elastičnimi prožnimi peresi. Za učenje gibanja sta uporabila simulirano ohlajanje. Ločeno sta modelirala več načinov mahanja s krili, kjer sta pri vsakem izbrala parametre, s katerimi je optimizacija pripeljala do najbolj realističnega gibanja. Model razvije gibanje za vnaprej podane želene spremembe skeleta.

Podoben problem je modeliranje hoje dvo- ali večnožnih bitij. V zadnjih nekaj letih se za to pogosto uporablja globoko spodbujevano učenje. Problem hoje dvonožca je mogoče razdeliti na dva nivoja [9], kjer se nižji nivo ukvarja s premikanjem zglobov tako, da bo dvonožec stopil na želeno mesto, višji nivo pa za nižjega izbira ciljne pozicije stopal. Namesto enega se lahko uporabi več akterjev-kritikov, ki se specializirajo za posamezne dele problema [28], s čimer se pohitri učenje.

Za učenje hoje dvo- in štirinožnih bitij so Heess in sod. [19] uporabili globoko spodbujevano učenje. Namesto uporabe kompleksne nagrajevalne funkcije, so tekače nagradili le glede na prehojeno pot, učili so jih na različnih terenih z različnimi ovirami, ki so zahtevale različne gibe. Tako naučeni tekači so bili sposobni teka, skakanja in izogibanja oviram.

Poglavje 2

Metode

2.1 Nvidia Flex

Fizikalna simulacija Nvidia Flex temelji na metodi pozicijske dinamike (angl. *position based dynamics*) [29], ki omogoča izvajanje v realnem času in se zato pogosto uporablja v računalniških igrah in interaktivnih aplikacijah. Ta metoda za razliko od tradicionalnih pristopov, ki temeljijo na silah (te lahko preprosto pretvorimo v pospeške, če poznamo mase elementov), nima korakov integracije pospeškov in hitrosti, ampak deluje direktno na pozicijah elementov. Objekti so definirani z vozlišči in omejitvami (angl. *constraints*), ki veljajo med njimi. Na ta način pozicijska dinamika doseže dobro zmogljivost, a nekoliko manjšo realističnost. Sama učinkovitost je odvisna od zahtevnosti simulacije, ki se veča s številom delcev in s postavljenimi omejitvami. Nvidia Flex nadgradi reševanje omejitev, da se lahko rešujejo vzporedno. To je ključnega pomena, saj Flex za računanje uporablja platformo Nvidia CUDA [30] ali Microsoft DirectCompute, ki omogočata koriščenje grafičnega procesorja.

V magistrskem delu smo uporabili Nvidia Flex različice 1.1.0 in platformo

Nvidia CUDA različice 8.0.27. Simulacijo smo poganjali na računalniku z grafičnim procesorjem Nvidia GeForce GTX 660. Poskrbeli smo, da simulacija ni prezahtevna in lahko deluje s 60 sličicami na sekundo. Funkcijo `NvFlexUpdateSolver`, ki izvede korak simulacije, kličemo sinhrono z osveževanjem zaslona in ji podamo konstanten časovni korak $\Delta t = 60^{-1}s$ in število podkorakov detekcije trkov $n_p = 4$. V vsakem podkoraku se izvede n_i iteracij reševanja omejitev. Več iteracij pomeni bolj toge omejitve, najboljše delovanje smo dosegli z nastavitvijo $n_i = 4$. Parametri simulacije so napisani v dodatku A.

V simulaciji Flex obstajata dve vrsti delcev: delci tekočin in vsi ostali. Glavna razlika je, da med delci tekočin delujejo dodatne sile za simuliranje viskoznosti, površinske napetosti ipd. Vsi delci iste vrste imajo enako velikost. Delcem tekočin smo nastavili velikost $r_f = 0.1$ in ostalim delcem $r_s = 0.15$. Velikost delcev pomeni najmanjšo dovoljeno razdaljo, ki jo simulacija ohranja med središči dveh delcev. Lahko si jo predstavljamo kot premer delcev. Delca različnih vrst bosta med seboj držala razdaljo r_s. Poleg teh dveh nastavitev lahko nastavljamo tudi največjo razdaljo, na kateri deluje interakcija med delci r_r, za katero mora veljati $r_r \geq \max(r_f, r_s)$. Tako vsaj pravi dokumentacija, sami pa smo ugotovili, da se je varneje držati pravila $r_r \geq \max(\alpha r_f, r_s), \alpha > 1$, kjer naj ima α vrednost vsaj 1.1, če ne želimo nepredvidljivih posledic, npr. izginjanja delcev tekočine. Temu pravilu smo zadostili z nastavitvijo $r_r = 0.15$.

Flex podpira različne tipe objektov: trdna telesa, mehka telesa, blago, plastična telesa in tekočine. Vsi so predstavljeni z množicami delcev, razlikujejo se le po omejitvah, s katerimi so implementirani. Od teh smo potrebovali tekočine in mehka telesa. Tekočine so simulirane z metodo pozicijskih tekočin (angl. *position based fluids*) [2], kar pomeni, da so implementirane z omejitvijo gostote tekočine. Ta je v Flex poenostavljena tako, da omeji gostoto na nenegativne vrednosti, torej deluje samo tako, da ločuje delce.

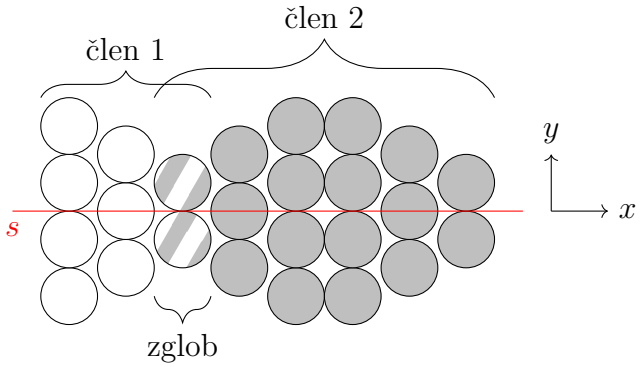
Trdna telesa so implementirana z omejitvijo ujemanja oblik [5], ki poskrbi, da so delci, ki telesu pripadajo, na koncu vsake iteracije fizikalne simulacije spet v prvotni obliki (med iteracijo simulacije so delci obravnavani kot nepovezani) – telo se ne deformira, lahko pa ima po iteraciji simulacije spremenjeno pozicijo ali rotacijo. Vendar pa trdna telesa niso nujno popolnoma trdna, lahko jim namreč nastavimo poljubno togost, ki določa, kako močno nanje deluje omejitev ujemanja oblike. To je pomembno ob prisotnosti drugih omejitev, ki delujejo na iste delce. Pri najnižji togosti 0, trdno telo razpade na posamezne delce.

Omejitev ujemanja oblik se uporablja tudi za mehka telesa, ki so sestavljena iz več trdnih členov (angl. *clusters*), od katerih vsak deluje enako kot trdno telo. Vsak delec mehkega telesa je lahko del več členov. Če tega ne izkoristimo, nič ne drži celotnega telesa skupaj, zato razpade na člene, ki se obnašajo kot samostojna trdna telesa.

2.2 Plavajoče bitje

Mehka telesa smo po vzoru ribe, kot jo je modeliral Tu [21], uporabili za implementacijo plavajočih bitij, ki so po dolžini razdeljena na n_c trdnih členov. Telo bitja je podolgovato in vse njegove elemente, ki so razporejeni po dolžini, bomo oštevilčevali začenši z najbolj sprednjim. Bitje ima $n_c - 1$ zglobov, ki so locirani na mejiščih členov. Točke, ki sestavljajo zglob so del obeh prileţnih členov, ki se zato ob gibanju deformirata. Izjemi sta prvi in zadnji člen, ki lahko pri $n_c > 2$ brez deformacije zadostita omejitvi ujemanja oblike. Primer bitja z enim zglobom prikazuje slika 2.1. Vsem členom smo določili togost 0.2, ki telesu omogoča zadostno deformacijo v zglobih, da bo lahko plavalo.

Plavalec je sestavljen iz delcev velikosti $r_s = 0.15$, ki se lahko prekrivajo drug z drugim, kar izboljša gibljivost plavalca in omogoča, da nastavimo razmik med sosednjimi delci v mirujoči drži na le $\frac{1}{2}r_s$ (zaradi boljše preglednosti je

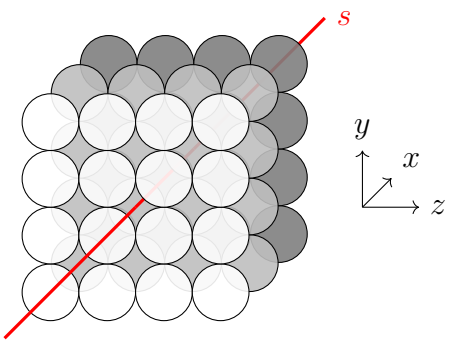


Slika 2.1: Plavajoče bitje z dvema členoma in zglobom med njima. Središča rezin ležijo na premici s.

na ilustracijah v tem dokumentu razmik med delci enak r_s). Z zmanjšanim razmikom preprečimo, da bi se delci tekočine zataknili v notranjosti plavalca. Vsi delci telesa imajo maso $m_s = 1$. Koristno bi bilo, če bi lahko nastavili različnim delcem v telesu različno maso, a to lahko privede do nepričakovanega obnašanja – telo pospešuje v na videz naključno smer. Delci so postavljeni v ploščate pravokotne rezine, ki so razporejene vzporedno ena drugi in pravokotno na dolžino bitja (koordinato x). Lomljenka, ki jo sestavljajo središča rezin bomo v nadaljevanju imenovali hrbtenica. Na začetku je hrbtenica ravna, torej vsa središča rezin ležijo na skupni premici (glej sliko 2.2). Višina in širina vsake rezine je nastavljiva. Prav tako je nastavljiva dolžina celotnega telesa (število rezin).

2.3 Postavitev na začetku simulacije

Bitje je postavljeno v akvarij, ki je napolnjen z delci tekočine. Ti so razporejeni v obliki kvadra s 50 delci po dolžini, 17 delci po širini in 7 delci po višini, razen v prostoru, ki ga zavzema plavalec – kvader tekočine ima votlino v obliki plavalca, tako da se delci plavalca in tekočine ne prekrivajo. Razdalja med sosednjima delcema tekočine je $0.9r_f$, s čimer zmanjšamo pojav, kjer se



Slika 2.2: Bitje s tremi enakimi rezinami širine in višine 4 delcev. Središča rezin ležijo na premici s.

gladina po začetku simulacije nekoliko zniža. Akvarij je z vseh strani, razen od zgoraj, omejen z ravninami (2.1) do (2.5), od katerih delci držijo razdaljo $r_p = \frac{1}{2} r_f$.

$$x = 0 \tag{2.1}$$

$$y = 0 \tag{2.2}$$

$$z = 0 \tag{2.3}$$

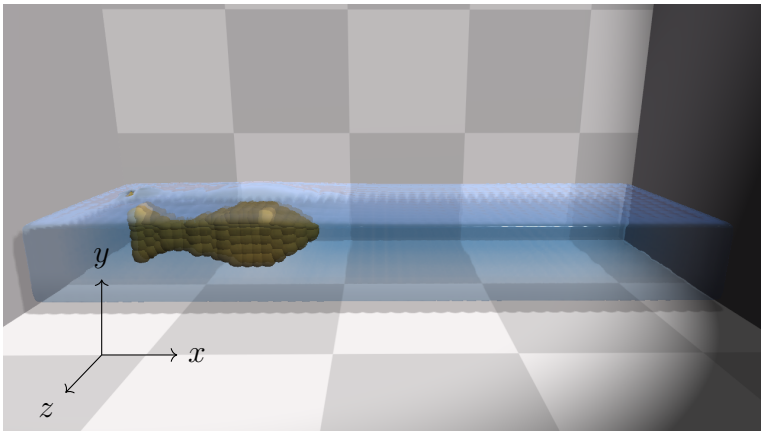
$$x = 4.51 \tag{2.4}$$

$$z = 1.54 \tag{2.5}$$

Da se tekočina ustali, izvedemo pred začetkom simulacije plavanja 100 klicev funkcije `NvFlexUpdateSolver` z argumentoma $n_p = 1$ in $\Delta t = 10^{-4} s$ ter po vsakem klicu nastavimo hitrosti vseh delcev na 0.

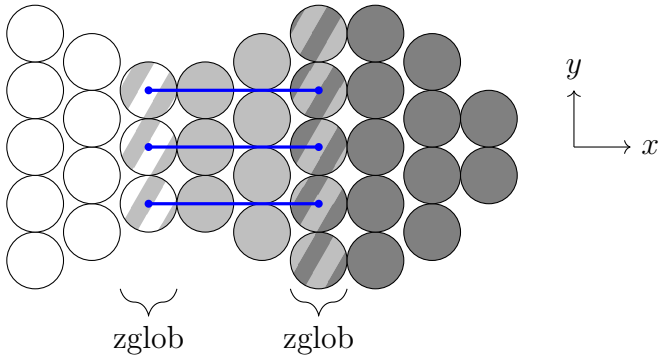
Plavalec je postavljen v sredino začetnega dela akvarija – koordinate središča njegove zadnje rezine so $x = 0.45$, $y = 0.6$ in $z = 0.6$. Sprednji del bitja gleda v smer plavanja, to je v smeri naraščajoče koordinate x (glej zajem zaslona na sliki 2.3). Zgornji del je obrnjen navzgor proti pozitivnemu y. Desno od plavalca narašča koordinata z.

Želimo, da naše plavajoče bitje ves čas simulacije ostane znotraj telesa tekočine. Ne sme priplavati na površino ali se potopiti na dno. Plovnost v Nvidia



Slika 2.3: Zajem zaslona na začetku simulacije

Flex ni odvisna le od razmerja mas in velikosti delcev, ampak tudi od števila delcev, povezanih z omejitvijo ujemanja oblik [4], nastavitev simulacije (št. podkorakov n_p in iteracij n_i), števila vzmeti (omejitev dolžine med dvema delcema). Ko je bitje animirano, lahko tudi gibanje povzroči dvig na gladino tekočine. Zaradi vseh teh spremenljivk je težko doseči, da bitja različnih oblik in z različnimi načini gibanja ostanejo pod gladino tekočine, a nad dnom akvarija. Izziv smo rešili tako, da smo delcem tekočine na dnu akvarija nastavili višjo maso, kot jo imajo delci plavalca, in delcem na vrhu akvarija nižjo. Delci med obema skrajnostma imajo linearno interpolirano maso glede na komponento y začetne pozicije. S poskušanjem smo ugotovili, da se pri masi delcev plavalca $m_s = 1$ in izbranih parametrih simulacije dobro obnese gradient mas delcev tekočin $m_f \in [2, \frac{1}{14}]$. Tak gradient prepreči, da bi bitje izstopilo iz telesa tekočine, a še vedno bitja z različnimi oblikami teles plavajo na različnih globinah – po začetku simulacije se bitje počasi dvigne ali spusti na globino, ki ustreza njegovi plovnosti. Da smo ta pojav zmanjšali, smo telesu tekočine nastavili čim nižjo višino.



Slika 2.4: Mišice med delci dveh zglobov

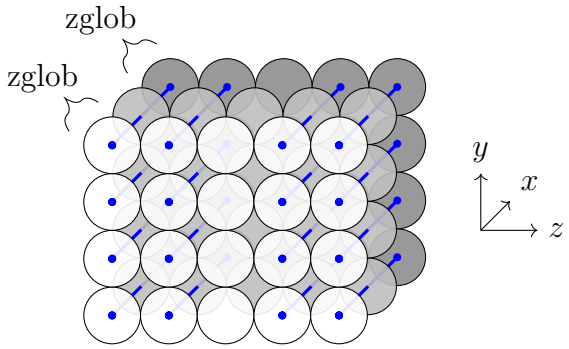
2.4 Gibanje

Naše mehko telo je po dolžini razdeljeno na trdne člene (glej sliko 2.1). Meje med njimi delujejo kot zglobi, a se še niso zmožni premikati. Tu [21] je modeliral zglob ribe s štirimi točkami – oglišči pravokotnika. Vsako izmed teh štirih točk je z mišico povezal z njenim ekvivalentom v sosednjem zglobu. Tak sistem mišic smo prilagodili na polna telesa v Nvidia Flex tako, da z longitudinalnimi mišicami povezujemo tudi notranje delce zglobov.

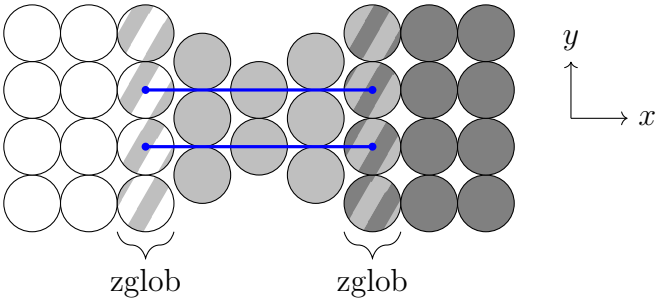
V Nvidia Flex lahko poljubna delca povežemo z vzmetjo. To je preprosta omejitev dolžine – simulacija bo ohranjala nastavljeno razdaljo med izbranima delcema. Vzmetem lahko nastavimo tudi togost, v tem delu uporabljamo le popolnoma toge vzmeti. Da omogočimo gibanje, z vzmetmi povežemo delce sosednjih zglobov (glej sliki 2.4 in 2.5). Ker vzmetem lahko nastavljamo dolžino, lahko s tem premikamo povezane delce v zglobih bolj skupaj ali narazen.

Da dva delca povežemo z vzmetjo, morata zadostovati naslednjim pogojem:

- Delca sta del različnih zglobov.
- Med zgloboma, ki jima delca pripadata, ni drugih zglobov.



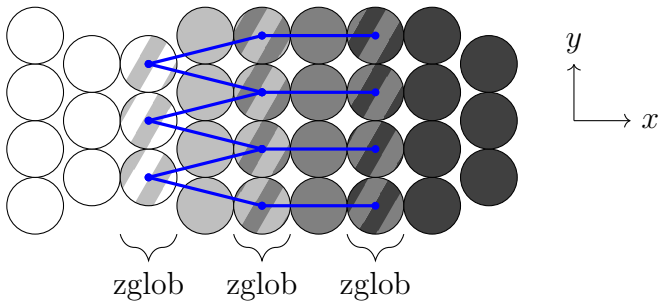
Slika 2.5: Mišice med delci dveh zglobov. Oba zgloba, kot tudi rezina med njima, sta rezini širine 5 in višine 4 delcev.



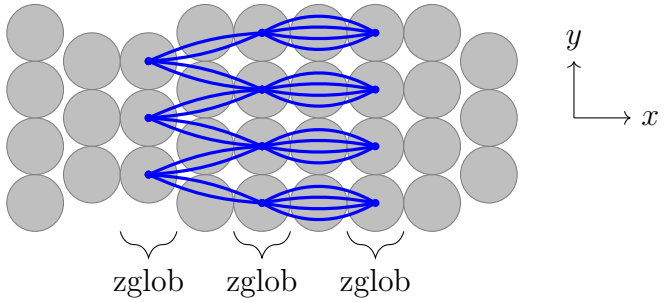
Slika 2.6: Mišice med delci dveh zglobov

- Oba delca sta na isti (levi ali desni) strani telesa. Če je kateri od delcev na sredini lateralne osi z telesa, ju ne povežemo (glej sliko 2.5).
- Daljica od enega do drugega delca seka delec vsake rezine med njima. Tako zagotovimo, da je celotna mišica znotraj telesa plavalca. Na sliki 2.6 se vidi, da so zaradi tega pravila nepovezani skrajno zgornji in skrajno spodnji delci zglobov.
- Premica, na kateri ležita je vzporedna z dolžino telesa (os x), razen kadar to ni mogoče, ker se širina ali višina rezin, katerim delca pripadata, razlikujeta za liho število delcev (glej sliko 2.7).

Zadnje pravilo dovoljuje vzmeti, ki niso vzporedne z osjo x, če sta povezani rezini neskladni. V takih primerih je lahko vsak delec iz prve rezine povezan



Slika 2.7: Mišice med delci treh zglobov

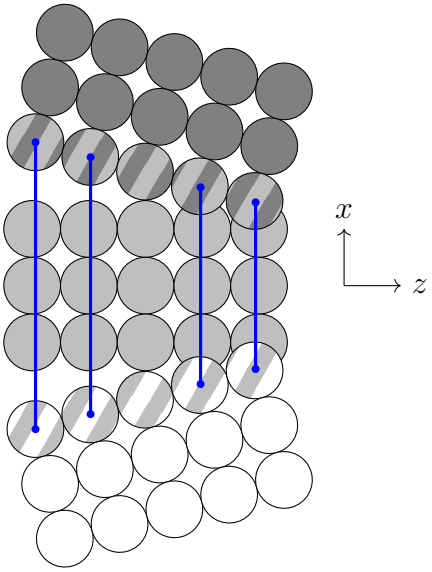


Slika 2.8: Vzmeti med delci treh zglobov

z dvema (če sta rezini neskladni po širini *ali* višini) ali s štirimi (če sta rezini neskladni po širini *in* višini) delci iz druge rezine. Da poravnanost rezin ne vpliva preveč na število vzmeti na delec, poskrbimo, da izhajajo iz vsakega delca (razen nekaterih robnih), ki zadostuje pogojem povezave, točno štiri vzmeti. Ni nujno, da so na drugi strani teh štirih vzmeti različni delci, kar poudarja slika 2.8. Na ostalih ilustracijah ni razločno, da so nekateri pari delcev povezani z več vzmetmi. Skupino vzmeti, ki povezujejo isti par delcev, bomo v nadaljevanju imenovali mišica.

2.5 Parametrizacija gibanja

V prejšnjem razdelku smo omogočili premikanje bitja s tem, da mu spreminjamo dolžine mišic, a da bomo dosegli potisk, bodo morale mišice delovati



Slika 2.9: Mišice v gibalu pri $u < 0$

usklajeno. To uskladitev bomo dosegli z optimizacijo, a najprej moramo definirati parametre, ki jih bomo optimizirali.

Uporabimo besedo gibalo za skupino mišic, ki povezujejo isti par sosednjih zglobov. Za vsako gibalo definirajmo dva parametra: amplitudo $a \in [0, 1]$ in fazo $\phi \in [0, 1]$. Vsa gibala imajo skupno frekvenco $f \in [0.5s^{-1}, 2s^{-1}]$. Osnovna funkcija delovanja mišic je sinusoida, zapisana v enačbi (2.6). Vrednost u vpliva na dolžino mišic l glede na enačbo (2.7).

$$u(t) = \sin(2\pi(ft + \phi)) \tag{2.6}$$

$$l(t) = l_{\text{mirujoča}} + \Delta l u(t) \tag{2.7}$$

Mišice na desni strani telesa so raztegnjene pri $u > 0$ in skrčene pri $u < 0$. Obratno velja za mišice na levi strani. Pri $u = 0$ imajo mišice prvotno dolžino – mišice so v mirujočem stanju $l = l_{\text{mirujoča}}$. Slika 2.9 prikazuje razliko dolžin mišic v gibalu pri $u < 0$. Prav tako se vidi, da se prvi in zadnji člen ne deformirata, v čemer se razlikujeta od vmesnih členov.

Enačba (2.8) definira premo sorazmernost največjega raztezka mišice Δl z:

- dolžino mišice v mirovanju $l_{\text{mirujoča}}$,
- maksimalno longitudinalno deformacijo mišice ϵ (angl. *muscle strain*),
- lateralno pozicijo mišice $z_m \in [-1, 1]$, skalirano glede na širino povezanih zglobov in
- parametrom amplitude a.

$$\Delta l = l_{\text{mirujoča}} \epsilon z_m a \tag{2.8}$$

Lateralna pozicija mišice $z_m = \frac{z_1+z_2}{2}$ je povprečje lateralnih pozicij obeh delcev z_1 in z_2, ki ju povezuje. Lateralna pozicija delca $z_i \in [-1, 1]$ pove, kje na rezini se delec nahaja. Vrednost $z_i = -1$ pomeni, da je delec del skrajno levega roba rezine (nizka vrednost koordinate z), in $z_i = 1$, da je del njenega skrajno desnega roba (visoka vrednost koordinate z).

Za konstanto ϵ smo izbrali vrednost $\frac{1}{2}$, kar pomeni, da je dolžina mišic pri $a = 1$ in $|z_m| = 1$ omejena na območje $l \in \left[\frac{1}{2} l_{\text{mirujoča}}, \frac{3}{2} l_{\text{mirujoča}}\right]$. Izbrana vrednost je blizu največje izmerjene deformacije rdečih mišic v ribah in približno štirikrat višja od tiste v belih mišicah [31]. Nekoliko višja je od izmerjenih vrednosti, da bitju omogoča večjo gibljivost, če bo optimizacija pokazala, da jo potrebuje.

2.6 Optimizacija

Zdaj, ko smo definirali parametrizacijo, moramo najti take parametre, ki pripeljejo do najučinkovitejšega plavanja. V ta namen smo uporabili genetski algoritem. To je način optimizacije, ki se zgleduje po procesu naravne selekcije. Na začetku se generirajo naključne rešitve ali *kromosomi*, ki so sestavljeni iz več *genov* – vrednosti, ki jih optimiziramo. Vsak kromosom se oceni in tisti z najboljšimi ocenami nadaljujejo v naslednjo iteracijo oz.

generacijo. Pred tem se nad njimi izvedejo različni genetski operatorji, npr. križanje ali mutacija. Tako dobimo nove kromosome, ki temeljijo na tistih iz prejšnje generacije. Nekateri izmed njih so lahko boljše rešitve problema.

Da določen kromosom ocenimo, ustvarimo v simulaciji plavanja bitje, ki ga kromosom predstavlja, in štiri sekunde simuliramo plavanje, kot ga določajo parametri, zapisani v kromosomu. Po pretečenem času na standardni izhod izpišemo oceno, ki je enaka v tem času preplavani razdalji, tj. razlika v povprečni koordinati x vseh delcev plavajočega telesa. Če se med ocenjevanjem kateri delec telesa dotakne katere od omejujočih ravnin, ocenjevanje predčasno prekinemo. Enako storimo, če bitje v dveh sekundah ne pokaže nobenega napredka.

Za izvedbo genetskega algoritma smo razvili javanski program, ki uporablja knjižnico JGAP [32]. Napisali smo funkcijo prilagojenosti (angl. *fitness function*), ki oceni posamezen kromosom tako, da požene našo simulacijo plavanja, ji preko standardnega vhoda poda parametre plavalca, ki so zapisani v kromosomu, in počaka na oceno, ki jo prebere s standardnega izhoda simulacije. Kromosomi se ocenjujejo zaporedno, ker istočasno izvajanje več instanc simulacije upočasni vsako izmed njih.

Na začetku vsake iteracije genetskega algoritma ocenimo celotno populacijo kromosomov, razen tistih, ki so ostali nespremenjeni iz prejšnje generacije in so zato že ocenjeni. Temu sledi korak "naravne" selekcije, kjer izberemo skupno 150 kromosomov: 30 najbolje ocenjenih in 120 naključno izbranih z ruletno selekcijo (angl. *roulette wheel selection*), pri kateri imajo bolje ocenjeni kromosomi večjo verjetnost, da so izbrani. Iz izbranih kromosomov se generirajo novi z uporabo treh genetskih operatorjev:

- Enotočkovno križanje (angl. *Single-point crossover*), ki modificira dva vhodna kromosoma tako, da jima zamenja naključni gen in vse naslednje. Izvedemo ga nad 35% populacije.

- Preprosta mutacija, ki naključno izbere 4% vseh genov v populaciji in jim naključno spremeni vrednosti.
- Gaussova mutacija, ki vsem genom v populaciji prišteje naključne vrednosti, vzorčene iz Gaussove porazdelitve.

V naslednjo generacijo nadaljujejo vsi kromosomi, ki so bili izbrani v koraku naravne selekcije, in vsi novo ustvarjeni kromosomi, ki na njih temeljijo. Celoten postopek začnemo s 150 naključno generiranimi kromosomi.

Geni so vrednosti, ki določajo obnašanje gibal. Uporabili smo dve različni sestavi kromosomov, tj. dva različna načina preslikave genov v parametre gibal. Naj bo n število gibal v telesu bitja. Prva sestava kromosomov ima večje število genov, zato jo imenujmo gosta sestava, in zanjo veljajo naslednje trditve:

- Vsa gibala imajo isto frekvenco f, zato zanjo potrebujemo en gen.
- Amplituda a vsakega gibala je določena s svojim genom, zato potrebujemo n genov.
- Vsako gibalo ima svojo fazo ϕ. A ker na gibanje vplivajo le razlike med fazami, ima prvo gibalo konstantno fazo $\phi = 0$ in potrebujemo le $n - 1$ genov za ostala gibala.

Primer kromosoma z gosto sestavo prikazuje tabela 2.1

Pri nekaterih modelih plavajočih bitij z velikim številom gibal smo po nekaj poskusih z gosto sestavo opazili v optimiziranih parametrih vzorce, na podlagi katerih smo osnovali drugo, redko sestavo. Ta omogoča hitrejšo optimizacijo plavanja, ker ima le naslednje štiri gene:

- Enako kot pri gosti sestavi vsebuje en gen za frekvenco f.

parameter	*geni*	*gibalo 1*	*gibalo 2*	*gibalo 3*	*gibalo 4*
f	$2s^{-1}$	$2s^{-1}$	$2s^{-1}$	$2s^{-1}$	$2s^{-1}$
a	$0.6, 0.7, 0.8, 0.9$	0.6	0.7	0.8	0.9
ϕ	$0.4, 0.8, 0.2$	0	0.4	0.8	0.2

Tabela 2.1: Primer preslikave kromosoma z *gosto* sestavo v parametre gibal za bitje s štirimi gibali

parameter	*geni*	*gibalo 1*	*gibalo 2*	*gibalo 3*	*gibalo 4*
f	$2s^{-1}$	$2s^{-1}$	$2s^{-1}$	$2s^{-1}$	$2s^{-1}$
a	$[0.6, 0.9]$	0.6	0.7	0.8	0.9
ϕ	$\Delta\phi = 0.4$	0	0.4	0.8	0.2

Tabela 2.2: Primer preslikave kromosoma z *redko* sestavo v parametre gibal za bitje s štirimi gibali

- Za kodiranje amplitude a kromosom vsebuje dva gena – enega za prvo in enega za zadnje gibalo. Amplitude vmesnih gibal $a_2, a_3, \ldots, a_{n-1}$ so določene z linearno interpolacijo (glej enačbo (2.9)).

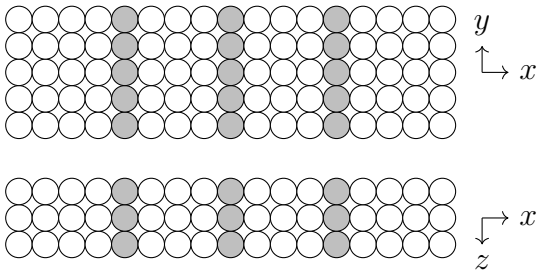
- Zadnji gen kodira fazno diferenco $\Delta\phi$, ki po enačbi (2.10) določa faze vseh gibal $\phi_1, \phi_2, \ldots, \phi_n$. Tako aritmetično zaporedje faz se ujema z opazovanji plavanja resničnih rib [33].

$$a_i = a_1 + \frac{i-1}{n-1}(a_n - a_1) \tag{2.9}$$

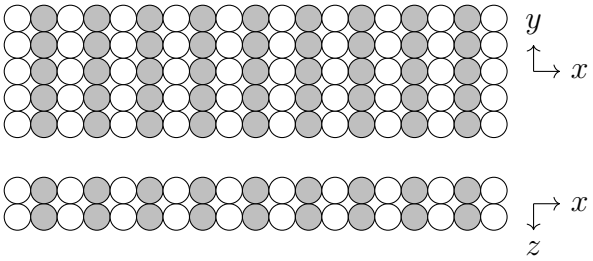
$$\phi_i = \mathrm{frac}(i\Delta\phi) \tag{2.10}$$

$$\mathrm{frac}(x) = x - \lfloor x \rfloor \tag{2.11}$$

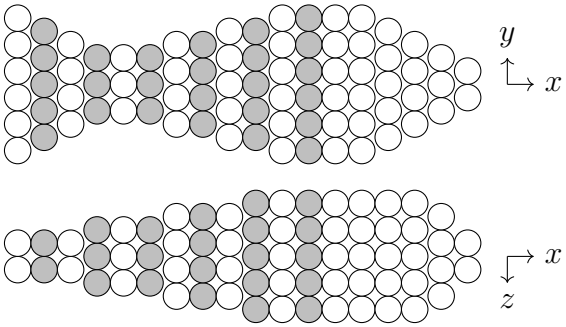
Primer kromosoma z redko sestavo prikazuje tabela 2.2



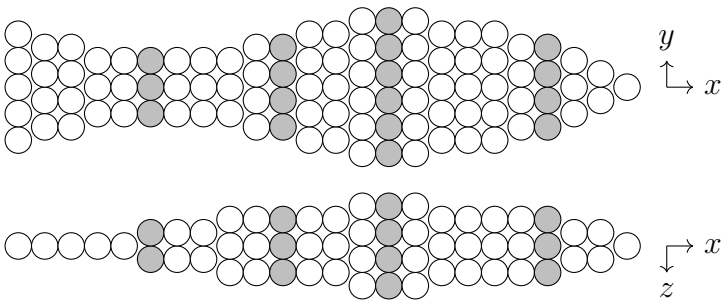
Slika 2.10: Plavajoče bitje 1 s strani (zgoraj) in od zgoraj (spodaj)



Slika 2.11: Plavajoče bitje 2 s strani (zgoraj) in od zgoraj (spodaj)



Slika 2.12: Plavajoče bitje 3 s strani (zgoraj) in od zgoraj (spodaj)



Slika 2.13: Plavajoče bitje 4 s strani (zgoraj) in od zgoraj (spodaj)

2.7 Modeli plavajočih bitij

Razvili smo plavanje štirih modelov bitij, ki se razlikujejo po obliki, številu členov in sestavi kromosomov. Prikazani so na slikah 2.13 do 2.11, kjer so s sivo barvo označeni zglobi. Ti štirje modeli so:

1. Preprost kvader dolžine 17, širine 3 in višine 5 delcev, razdeljen je na štiri člene. Tudi to bitje uporablja gosto sestavo kromosomov.

2. Kvader dolžine 19, širine 2 in višine 6 delcev, razdeljen na 10 členov. Zanj smo uporabili redko sestavo kromosomov.

3. Bitje v obliki ribe, ki ima večino členov v repu in uporablja gosto sestavo kromosomov.

4. Bitje, ki obliko in postavitev zglobov povzema po Tujevi mehkotelesni ribi [21]. Oblika je rasterizacija ilustracij Tujeve ribe s strani in od zgoraj z dvema spremembama: najtanjši del repa smo ojačali po višini, da bolje ohranja obliko, in zadnja dva zgloba smo združili v enega, da lahko bitje nekoliko bolje uporablja repno plavut. Uporabili smo gosto sestavo kromosomov.

2.8 Ocenjevanje realističnosti plavanja

Za vse štiri plavalce smo z genetskim algoritmom razvili čim bolj učinkovit način plavanja skozi simulirano tekočino. Da ocenimo, kako realističen je razvit način plavanja, ga moramo na nek način primerjati s plavanjem resničnih živali. Gibanje velikega števila vrst rib, ki plavajo s telesom in repno plavutjo, opisuje funkcija (2.12) [27, 34], tj. funkcija lateralnega odmika točke x vzdolž hrbtenice ribe v času t. V kontekstu funkcije velja drug koordinatni sistem kot v simulaciji: koordinata x je obrnjena, tako da je v prvi točki

način plavanja	*amplitudna ovojnica*	*valovno število*
anguilliform [35]	$a(x) = 0.1e^{x-1}$	$k = 9.8L^{-1}$
carangiform [36]	$a(x) = 0.004fL - 0.02fx + 0.04fL^{-1}x^2$	$k = 7.0L^{-1}$
thunniform [36]	$a(x) = 0.02L - 0.12x + 0.2L^{-1}x^2$	$k = 5.7L^{-1}$

Tabela 2.3: Amplitudne ovojnice in valovna števila

telesa $x = 0$ in narašča v smeri proti repu. Koordinata z še vedno narašča v smer desno od telesa.

$$z(x,t) = a(x)\sin(kx - 2\pi ft) \tag{2.12}$$

Različne ribe imajo različno amplitudno ovojnico $a(x)$ in valovno število k. Plavanje s telesom in repno plavutjo lahko razdelimo na pet skupin: *anguilliform*, *sub-carangiform*, *carangiform*, *thunniform* in *ostraciiform*. Zapisali smo jih v vrstnem redu glede na to, kolikšen del telesa uporabljajo za plavanje. V skupino *anguilliform* spadajo dolge, tanke ribe, kot so jegulje, ki uporabljajo celotno telo za ustvarjanje potiska. V zadnjo skupino spadajo ribe družine ostraciidae, ki ustvarjajo potisk zgolj z osciliranjem repne plavuti. Simulirano gibanje naših plavajočih bitij bomo primerjali s tremi izmed prej omenjenih naravnih načinov plavanja: *anguilliform*, *carangiform* in *thunniform*. Amplitudne ovojnice in valovna števila, ki opisujejo te tri vrste plavanja, vsebuje tabela 2.3, kjer so nekatere vrednosti odvisne od frekvence plavanja f in dolžine ribe L.

Da izmerimo lateralni odmik $z(x,t)$ točk našega plavajočega bitja skozi čas, poženemo simulacijo plavanja in ob vsakem koraku izpišemo trenuten čas simulacije t ter komponenti $-x$ in z središča vsake rezine telesa. V n korakih simulacije tako dobimo n lomljenk z l točkami, ki predstavljajo hrbtenico plavalca v danem času. Vsako lomljenko zarotiramo tako, da je premica, ki se ji najbolje prilega, vzporedna z osjo x. Nato jo transliramo tako, da od vseh vrednosti x odštejemo sprednjo x_1 in od vseh vrednosti z odštejemo njihovo povprečje $\frac{1}{l}\sum_{i=1}^{l} z_i$. Ker so členi telesa ravni, uporabimo linearno

interpolacijo, da povežemo točke lomljenke v zvezno funkcijo $z(x,t)$. Naj bo $T = \{t_1, t_2, \ldots, t_n\}$ množica zapisanih časov meritev. Z enačbo (2.13) dobimo amplitudno ovojnico $a(x)$, ki jo lahko primerjamo s tistimi, ki opisujejo gibanje resničnih rib.

$$a(x) = \max_{t \in T} |z(x,t)| \tag{2.13}$$

Ko primerjamo razvito plavanje $z_s(x,t)$ z resničnim $z_r(x,t)$, je dobro imeti način za izračunanje skalarne vrednosti razlike e med njima. V ta namen smo sestavili enačbo (2.14), ki temelji na povprečni kvadratni napaki.

$$e = \frac{1}{nx_l} \sum_{t \in T} \int_0^{x_l} e(x,t)dx \tag{2.14}$$

$$e(x,t) = \sqrt{\frac{x}{L}} \left(\frac{\alpha z_r(x,t) - z_s(x,t+\beta)}{a_r(x)} \right)^2 \tag{2.15}$$

Ker imajo vsa razvita plavanja širšo amplitudno ovojnico od resničnih, jih skaliramo s parametrom α, in da oba načina plavanja postavimo v isto fazo, uporabimo parameter β. Oba parametra poiščemo s preprostim linearnim iskanjem s korakom 0.01 tako, da dobimo čim manjšo napako e. Amplitudna ovojnica, ki opisuje način plavanja *carangiform*, je zelo ozka, zato dobimo ob primerjanju kateregakoli plavanja z njo nizke vrednosti e (ob nizkem parametru α). Da izenačimo vse resnične načine plavanja, delimo razliko v odmiku hrbtenice z amplitudno ovojnico primerjanega resničnega plavanja $a_r(x)$. Opazili smo, da se razvita plavanja bolj ujemajo z resničnimi pri glavi in manj pri repu, zato smo napako utežili glede na pozicijo vzdolž hrbtenice s členom $\sqrt{\frac{x}{L}}$. S tem členom dobimo boljša parametra α in β. Za izračun integrala od prve $x_1 = 0$ do zadnje $x_l \leq L$ vrednosti x, ga poenostavimo v vsoto nad množico $X = \{\frac{0}{10^3 x_l}, \frac{1}{10^3 x_l}, \ldots, \frac{10^3}{10^3 x_l}\}$, da dobimo enačbo (2.16)

$$e = \frac{1}{n10^3} \sum_{t \in T} \sum_{x \in X} e(x,t) \tag{2.16}$$

Poglavje 3

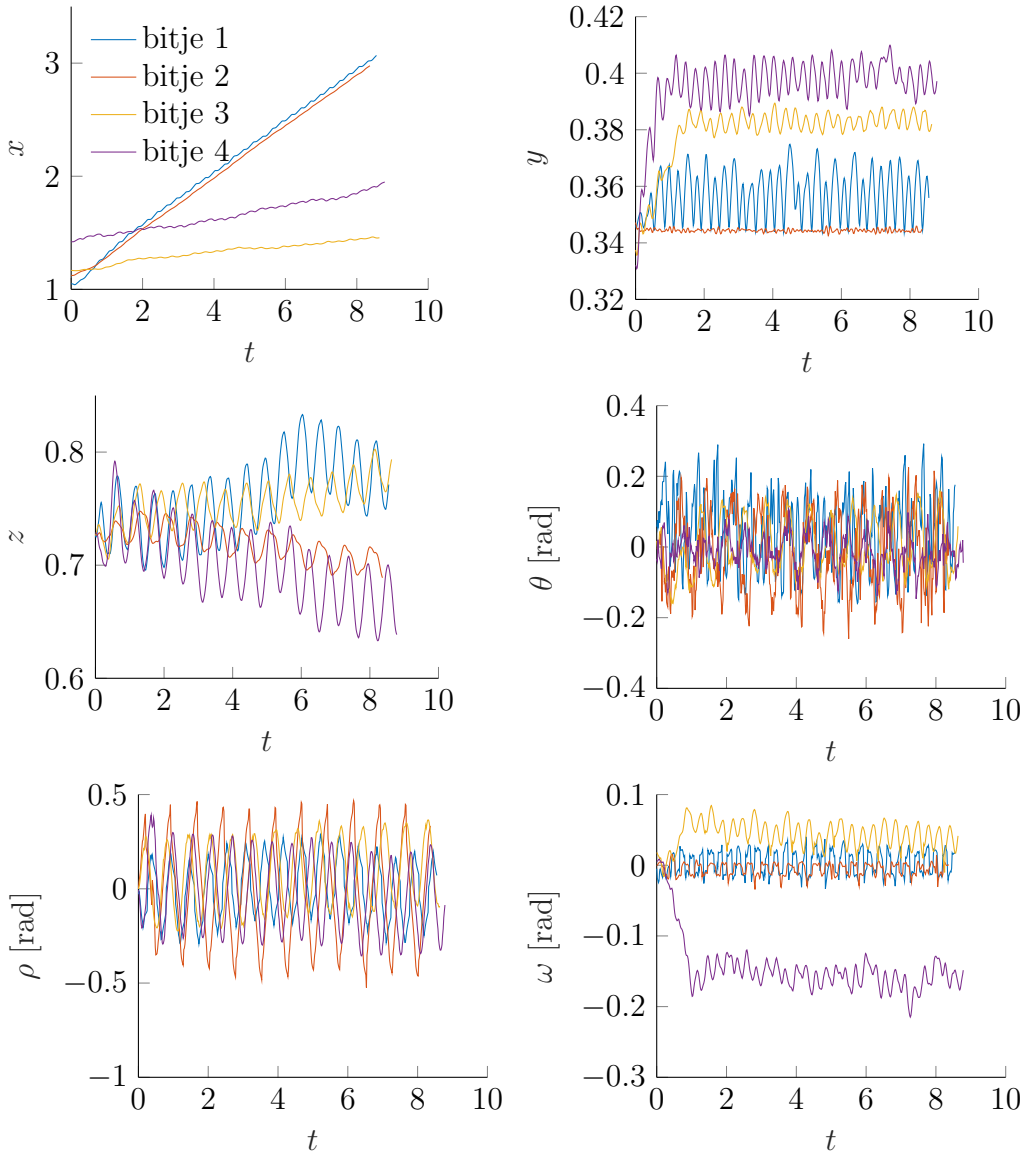
Rezultati

V razdelku 2.7 opisanim modelom plavajočih bitij smo razvili čim bolj učinkovit način gibanja skozi simulirano tekočino. Za vsako bitje smo optimizacijo poganjali 100 generacij. Optimizirani parametri frekvence f, amplitude a in faze ϕ ter hitrosti plavanja v, ki jih omogočajo, so napisani v tabeli 3.1. Najučinkoviteje plava bitje 1.

Bitja plavajo stabilno in naravnost. Na grafih slike 3.1 vidimo, kako se skozi čas spreminjajo koordinate x, y in z središča bitja ter koti rotacije bitja okrog vseh treh osi:

- kot θ okrog osi x,
- kot ρ okrog osi y in
- kot ω okrog osi z.

Videti je nekaj odvečnega nihanja okoli osi x, najverjetneje zaradi heterogene mase delcev tekočine, ki so na vrhu lažji kot spodaj. Bitje 4 plava nekoliko zarotirano v negativno smer okrog osi z. Videti je, da imata njegov sprednji in zadnji del različno plovnost.



Slika 3.1: Pozicija in rotacija plavajočih bitij skozi čas

plavalec	f	a	ϕ	v
model 1	$1.861s^{-1}$	0.973, 0.690	0.000, 0.648	$0.2311s^{-1}$
model 2	$1.338s^{-1}$	$[0.504, 0.184]$	$\Delta\phi = 0.431$	$0.1903s^{-1}$
model 3	$1.606s^{-1}$	0.829, 0.350, 0.763, 0.312, 0.953	0.000, 0.161, 0.123, 0.046, 0.732	$0.0395s^{-1}$
model 4	$1.761s^{-1}$	0.863, 0.503, 0.271	0.000, 0.176, 0.729	$0.0376s^{-1}$

Tabela 3.1: Optimizirani parametri in izmerjene hitrosti plavanja

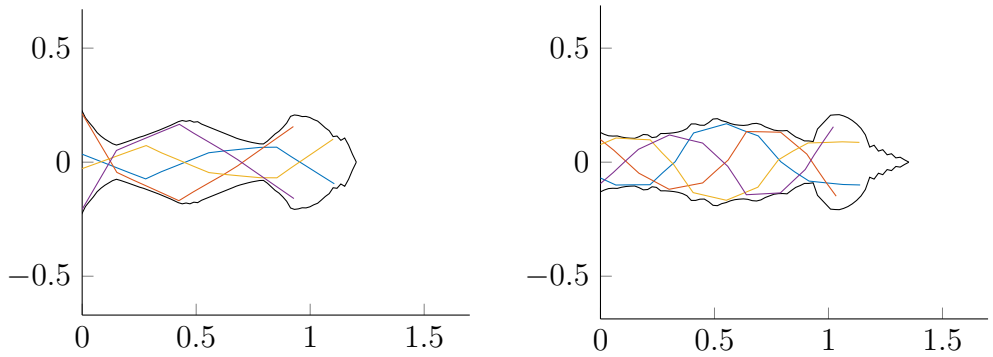
bitje	*anguilliform*	*carangiform*	*thunniform*
model 1	0.1834	0.1511	0.1997
model 2	0.0402	0.1654	0.2099
model 3	0.2539	0.1844	0.2327
model 4	0.2773	0.2199	0.2430

Tabela 3.2: Napake e

Na sliki 3.2 se vidi, kako se telesa bitij deformirajo ob plavanju. Vsaka podslika vsebuje bitje v štirih različnih fazah plavanja in amplitudno ovojnico. Te so prikazane tudi na sliki 3.3 skupaj z amplitudnimi ovojnicami resničnih načinov plavanja *anguilliform*, *carangiform* in *thunniform*. Ker so členi bitij trdni, se pojavijo na ovojnicah vdolbine, in ker so njihova telesa končno dolga, ima ovojnica na zadnjem delu nizke vrednosti.

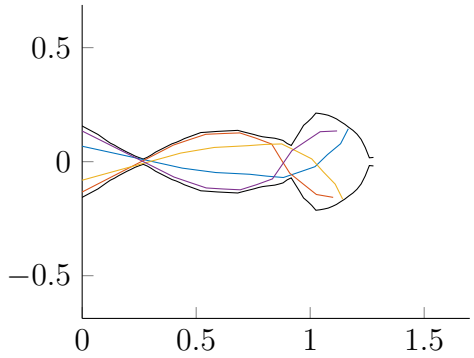
Vse pridobljene amplitudne ovojnice so širše od resničnih. Tudi če skaliramo izmerjene položaje hrbtenice, se večina razvitih gibanj ne ujema z resničnimi – preveč gibanja je pri glavi in premalo pri repu. Izračunane napake e za vsa štiri bitja so napisane v tabeli 3.2. Bitje 2 po skaliranju doseže boljše rezultate od ostalih – njegovo plavanje je podobno plavanju *anguilliform*, s katerim se ujema z večino telesa, razen nekaj odstopanja v repu. To lahko vidimo na zajemih plavanja na sliki 3.4.

Pri plavanju se v tekočini ustvarja turbulenca. Na koncu repa se ob zamahu

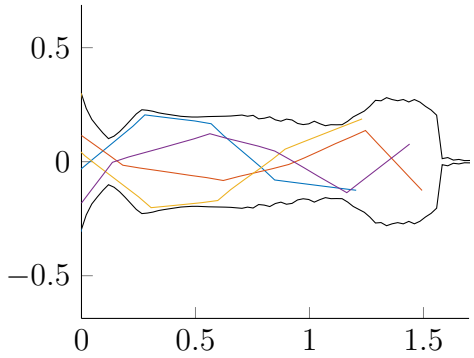


(a) Bitje 1

(b) Bitje 2

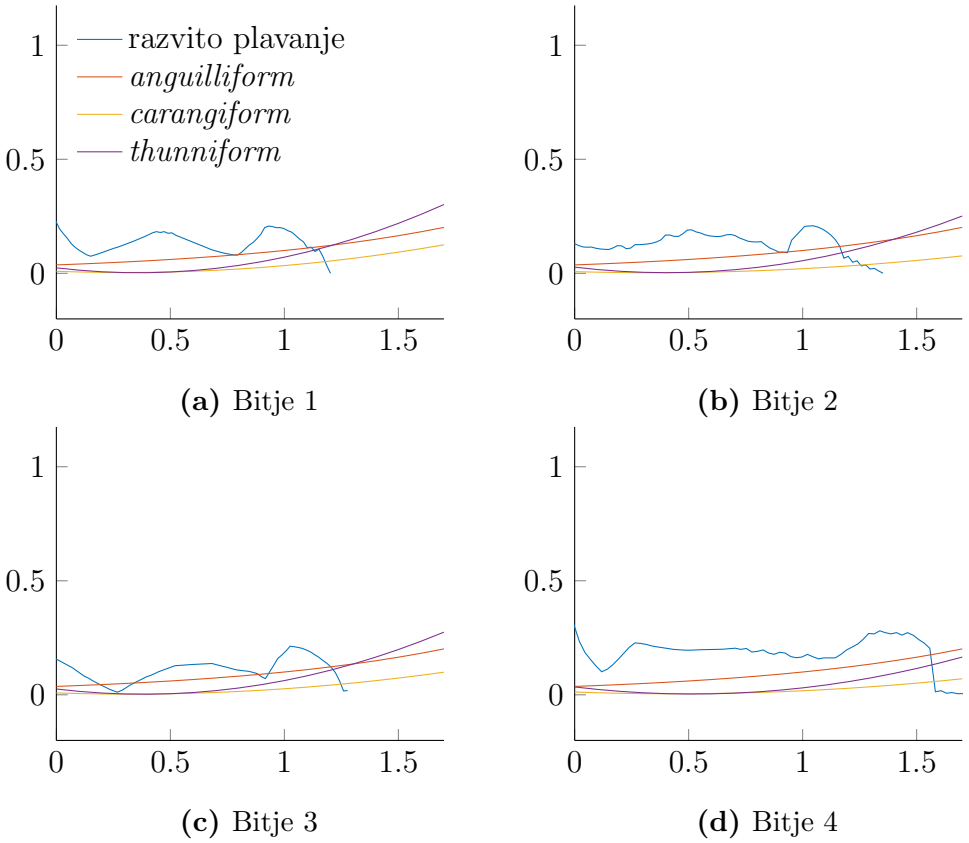


(c) Bitje 3



(d) Bitje 4

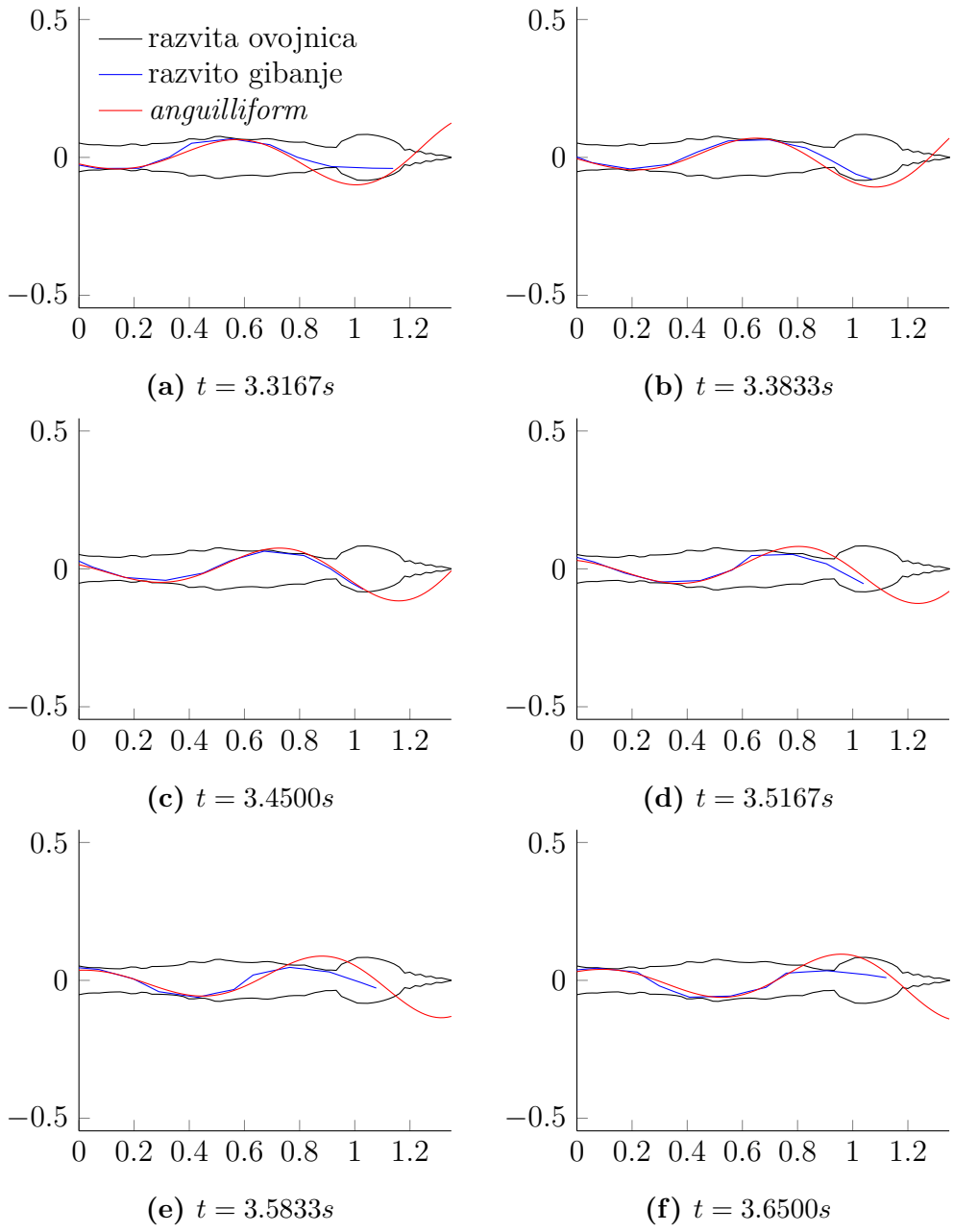
Slika 3.2: Hrbtenica plavajočih bitij v štirih različnih fazah



(a) Bitje 1 **(b)** Bitje 2

(c) Bitje 3 **(d)** Bitje 4

Slika 3.3: Amplitudne ovojnice

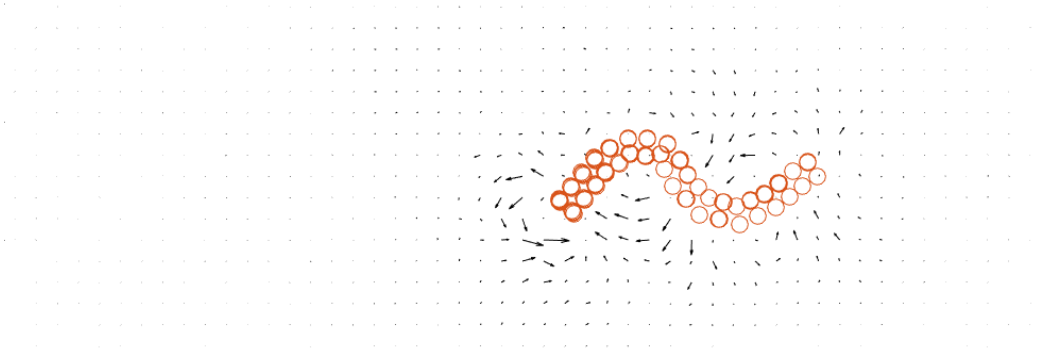


(a) $t = 3.3167s$ **(b)** $t = 3.3833s$

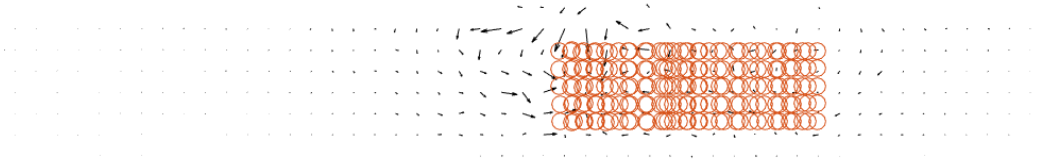
(c) $t = 3.4500s$ **(d)** $t = 3.5167s$

(e) $t = 3.5833s$ **(f)** $t = 3.6500s$

Slika 3.4: Skalirano gibanje bitja 2 v primerjavi s plavanjem *anguilliform*



Slika 3.5: Hitrosti delcev (projekcija od zgoraj)



Slika 3.6: Hitrosti delcev (projekcija s strani)

pojavi vrtinčenje, ko delci tekočine zalijejo praznino, ki ostane za repom bitja. Hitrosti delcev tekočine po tem, ko bitje 2 preplava približno polovico akvarija, so prikazane na slikah 3.5 in 3.6.

Želeli smo izvedeti, ali se naučena bitja premikajo zaradi interakcije z delci tekočine, ali zaradi kakšne anomalije v simulaciji, zato smo izvedli tudi poskus gibanja v vakumu. Pri tem poskusu nismo ustvarili delcev vode, da smo lahko videli, ali se lahko bitja premikajo tudi brez njih. Onemogočili smo gravitacijo, da bitje ostane na isti višini. Enako kot pri plavanju v tekočini smo izmerili hitrost premikanja naprej (v smeri koordinate x). Rezultati so

različni za različna bitja:

- Bitje 1 se nestabilno premika s povprečno hitrostjo $v = -0.1539s^{-1}$ (premika se v vzratno smer).
- Bitje 2 se nestabilno premika s povprečno hitrostjo $v = -0.0342s^{-1}$.
- Bitje 3 se stabilno premika s povprečno hitrostjo $v = -0.0895s^{-1}$.
- Bitje 4 se nestabilno premika s povprečno hitrostjo $v = 0.0605s^{-1}$.

Večina bitij se v vakumu premika v vzratno smer, zato lahko sklepamo, da v tekočini ustvarjajo potisk z interakcijo z njenimi delci. Izjema je bitje 4, ki se v vakumu premika naprej z višjo hitrostjo kot v tekočini, ki ga očitno le ovira.

Poglavje 4

Razprava

Pokazali smo, da je plavanje v simulaciji Nvidia Flex mogoče. Simulacija teče v realnem času s 60 sličicami na sekundo, za kar ne potrebuje pretirano drage strojne opreme (dokler ne uporabimo elipsoidnega prikaza tekočine [37], ki ga Flex podpira). Z genetskim algoritmom smo uspeli razviti učinkovito gibanje za različne modele plavajočih bitij, ki plavajo stabilno in naravnost. Žal se pridobljeni načini plavanja ne ujemajo najbolje z resničnimi. Izjema je bitje 2, katerega plavanje se po skaliranju dobro ujema z načinom plavanja *anguilliform*.

Optimizacija je vsem modelom bitij razvila plavanje z znatno večjo amplitudo, kot jo opazimo pri resničnih ribah. Do tega je optimizacija pripeljala, ker tako plavalci ustvarijo več potiska in plavajo hitreje. Kljub temu so dosežene hitrosti nizke – naše najhitrejše bitje 1 v sekundi preplava le približno desetino svoje dolžine ($v = 0.0963Ls^{-1}$), kar je skoraj osemkrat počasneje od rečne jegulje (*Anguilla anguilla*), ki plava s hitrostjo $v = 0.62ms^{-1} = 0.77Ls^{-1}$ [38]. Če kateremkoli bitju zmanjšamo vse amplitudne parametre a tako, da dobimo plavanje z bolj primerno širino amplitudne ovojnice, se drastično zmanjša tudi ustvarjen potisk – v večini primerov bitje popolnoma izgubi zmožnost plavanja.

Za plavanjem resničnih rib v resnični vodi se ustvarijo vrtinci, ki ostanejo tudi več dolžin za ribo [39, 35]. To ima posledice tudi na ribah v jatah, saj preko vode vplivajo ena na drugo [20]. Tudi v naši simulaciji plavanja se pojavijo vrtinci v neposredni bližini repa plavajočega bitja, a ne sežejo veliko dlje – tekočina se kmalu ustali in ni več dokaza, da je skoznjo plavalo bitje. Hitra ustalitev tekočin je običajna za metodo pozicijske dinamike [2]. Zato simulacije ne bi mogli uporabiti za simuliranje jate rib, kadar je pomembna hidrodinamična interakcija med njimi.

4.1 Vpliv parametrov simulacije na plavanje

Nvidia Flex ima veliko število parametrov in vsak parameter ima lahko več različnih posledic. Nekateri parametri otežijo ali celo onemogočijo plavanje. Parametri simulacije so napisani v dodatku A, tu bomo opisali le posledice, ki jih imajo naslednje nastavitve na plavanje:

- Število podkorakov detekcije trkov n_p in število iteracij reševanja omejitev n_i vplivata na togost omejitev in posledično tudi na plovnost. Poleg tega je pri nizkih vrednostih teh dveh parametrov plavanje nestabilno, pri visokih pa se zveča časovna zahtevnost simulacije.

- Razmerje mas med delci plavalca in delci tekočine vpliva na plovnosti ter na zmožnost odrivanja tekočine. Če so delci tekočine precej težji od delcev bitja, se ta sploh ne more premakniti.

- Kohezija in površinska napestost tekočine otežujeta plavanje, zato smo ju nastavili na 0.

- Viskoznost je koristna za plavanje, ker plavalec z odrivanjem delcev premika tudi njihove sosede. Sila, s katero plavalec odriva delce, se porazdeli mednje, zato previsoka viskoznost otežuje plavanje. Viskoznost

ima tudi nekaj vpliva na turbulenco, ki jo plavanje ustvari v tekočini. A tudi pri najnižje nastavljeni viskoznosti se vrtinčenje hitro ustavi.

- Omejitev vrtinčenja (angl. *vorticity confinement*) [40] naj bi pomagala ohranjati vrtinčenje v tekočini, a nismo opazili razlike pri plavanju ali pri turbulenci, ki pri tem nastaja.

- Največja razdalja, na kateri deluje interakcija med delci r_r vpliva na delovanje tekočinskih parametrov, ki so pri večjih vrednostih r_r bolj izraziti. Opazili smo da v splošnem večja razdalja otežuje plavanje in občutno upočasni delovanje simulacije. Nizke vrednosti lahko povzročijo nestabilnost.

- Dušenje (angl. *damping*) upočasnjuje simulirane delce. Na vsak delec deluje kot pospešek, ki je sorazmeren z njegovo hitrostjo, a obrnjen v nasprotno smer. Dušenje ima vpliv na plavanje, ki ga rahlo upočasnjuje. Ima tudi majhen vpliv na ustvarjeno turbulenco, a se tudi brez dušenja vrtinčenje v tekočini hitro ustavi. Dušenje močno zavira anomalno gibanje bitij v vakumu.

4.2 Aktivacija mišic s potenciranjem

Ob različnih nastavitvah parametrov simulacije smo preizkušali tudi različne parametrizacije gibanja, da bi dosegli plavanje. Enačba (2.6) za aktivacijo mišic v razdelku 2.5 je dolgo časa vsebovala tudi potenciranje, kot je napisano v enačbah (4.1) in (4.2).

$$u_1(t) = \sin(2\pi(ft + \phi)) \tag{4.1}$$

$$u(t) = \left(\frac{u_1(t) + 1}{2}\right)^{p_p} 2 - 1 \tag{4.2}$$

Spremenljivka $p_p \in [\frac{1}{2}, 4]$ je bila dodaten parameter, ki smo ga optimizirali. Bitja so razvila potence p_p in frekvenco f blizu zgornjih mej ter amplitude

v območju $a \in [0.5, 1]$. Razvita vrednost fazne diference $\Delta\phi$ je bila odvisna od razmika med zglobi. S takimi parametri so bitja zelo dobro "plavala": stabilno, naravnost in veliko hitreje kot bitja opisana v razdelku 2.7. Če smo ročno spremenili fazno diferenco na $1 - \Delta\phi$, so bitja plavala enako dobro, a nazaj.

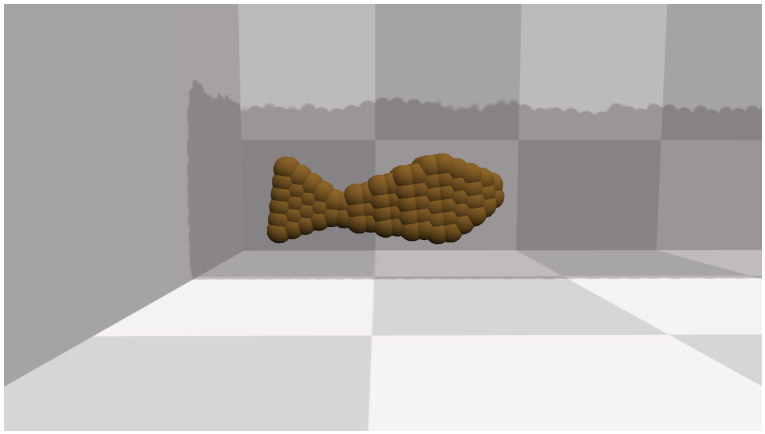
Po poskusu brez tekočine se je izkazalo, da jih je ta zgolj ovirala – bitja so pospeševala tudi v vakumu. Ker to ni plavanje, smo potenciranje odstranili, še vedno pa je to najučinkovitejši način premikanja v simulaciji Nvidia Flex, ki smo ga odkrili.

4.3 Zasnova telesa bitja

Preizkušali smo tudi različne zasnove plavajočih bitij. Začeli smo s preprostim bitjem, sestavljenim iz dveh trdnih teles v obliki kvadrov širine dveh delcev. Telesi sta bili z vzmetmi, ki so tvorile zglob, povezani skupaj v daljši kvader iste širine. Za gibanje so poskrbele dodatne vzmeti, ki smo jim spreminjali dolžino, da so delovale kot mišice. Bitje se je lahko upogibalo v obe smeri, a ni zaplavalo.

Bitje iz trdnih teles smo nadgradili z uporabo enega mehkega telesa. Preden smo implementirali bitja z ročno postavljenimi členi in zglobi, smo poskusili uporabiti trirazsežni model ribe (mnogokotniška mreža v formatu `.obj`) in funkcijo `NvFlexExtCreateSoftFromMesh` iz razširitvene knjižnice Nvidia Flex. Ta funkcija ustvari mehko telo, katerega delce ustvari z vzorčenjem podanega trirazsežnega objekta, in ga avtomatično razdeli na sferične člene s podanim polmerom.

Vzorčenje smo predelali, da deluje le na levi strani objekta in ustvari simetrično telo. Delce na straneh ribe smo simetrično povezali v longitudinalne mišice, podobno kot je predstavljeno v razdelku 2.4. Bitje, ustvarjeno na



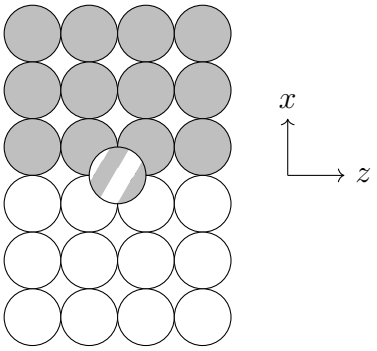
Slika 4.1: Bitje z nekonsistentno plovnostjo v tekočini

ta način, ni bilo zelo gibljivo in tudi z njim nam ni uspelo doseči plavanja. Vzrok je bil najverjetneje v nesimetričnih členih z suboptimalnimi stičišči.

Tudi zasnova bitja, opisana v razdelku 2.2, ima svoje hibe. Zglobi mehkega telesa niso omejeni na rotacijo okrog ene osi, ampak omogočajo gibanje okrog vseh treh. Da se upogibajo le v smeri, ki jo narekujejo mišice, morajo imeti telesa plavalcev v zglobih višino vsaj 2 delca. Vendar se kljub temu lahko neželeno deformirajo, kadar imajo različni deli plavalca različno plovnost, tj. kadar so njegovi členi sestavljeni iz različnega števila delcev. Tako telo je prikazano na sliki 4.1. Zaradi tega menimo, da najbolje delujejo telesa preprostih, kvadrastih oblik z večjim številom členov enakih velikosti. Taka telesa lahko kljub preprosti obliki delujejo vizualno prepričljivo, saj Nvidia Flex za namen prikaza omogoča pripetje poljubnega trirazsežnega objekta na delce telesa.

4.4 Nadaljnje delo

Da bi omejili gibanje v zglobu na vrtenje okrog osi y, bi ga lahko zasnovali drugače. Namesto, da je ena celotna rezina telesa uporabljena kot zglob, bi



Slika 4.2: Predlog drugačne zasnove zgloba

med dva sosednja člena vstavili navpično vrsto dodatnih delcev, ki bi delovali kot zglob in bi bili del obeh členov (glej sliko 4.2). Flex omogoča nastavitev skupin delcev, med katerimi ne računa trkov, kar bi uporabili za delce v zglobu, da nimajo stika s tekočino. Tako bi lahko bila vrsta delcev v zglobu poljubno dolga. Tak zglob bi morda tudi blagodejno vplival na gibljivost bitja, saj se členom ne bi bilo treba deformirati ob upogibu telesa. Bitje s takimi zglobi bi bilo bolj podobno bitjim, kot so jih modelirali Tan in sod. [13] in manj modelu ribe, ki jo je opisal Tu [21]. Slabost tega pristopa je, da bi se delci v zglobu še vedno lahko zaletavali v ravnine, ki omejujejo akvarij.

V tem delu smo se omejili na podolgovata bitja z zglobi, razporejenimi po dolžini, vendar bi bilo zanimivo videti plavanje bolj raznolikih bitij, da bi bil nabor bližje pestrosti, ki so jo dosegli Tan in sod. [13]. Lahko bi poleg gibanja optimizirali tudi obliko bitij in poleg plavanja naravnost še druge cilje, npr. obračanje, dvig ali spust.

Pri mnogih odločitvah smo bili omejeni s strani strojne opreme. V prihodnosti pričakujemo, da bo z močnejšimi grafičnimi procesorji mogoče uporabiti veliko manjše delce tekočine r_f, kar bo morda pozitivno vplivalo na realističnost simulacije. Z večjo razdaljo r_r, na kateri deluje interakcija med delci, bi morda bolje simulirali turbulenco v tekočini za plavajočim bitjem. In z večjim številom delcev tekočine bi lahko sestavili večji akvarij, v katerem bi lahko plavalo več bitij hkrati.

Dodatek A

Parametri simulacije Nvidia Flex

Ta dodatek vsebuje vrednosti uporabljenih parametrov, razen tistih, ki na delovanje simulacije ne vplivajo.

Funkciji `NvFlexInit` podamo strukturo `NvFlexInitDesc`, ki ji nastavimo polja na sledeče vrednosti:

ime polja	*vrednost polja*
`enableExtensions`	`true`
`computeType`	`eNvFlexCUDA`

Argumenti, ki jih podamo funkciji `NvFlexInit`:

ime parametra	*vrednost argumenta*
`version`	110
`errorFunc`	`nullptr`
`desc`	instanca strukture `NvFlexInitDesc`

Argumenti, ki jih podamo funkciji `NvFlexCreateSolver`:

ime parametra	*vrednost argumenta*
`lib`	rezultat funkcije `NvFlexInit`
`maxParticles`	število vseh delcev
`maxDiffuseParticles`	0
`maxNeighborsPerParticle`	96

Argumenti, ki jih podamo funkciji `NvFlexUpdateSolver`:

ime parametra	*vrednost argumenta*
`solver`	rezultat funkcije `NvFlexCreateSolver`
`dt`	$\Delta t = 60^{-1}s$
`substeps`	$n_p = 4$
`enableTimers`	`false`

Funkciji `NvFlexSetParams` podamo strukturo `NvFlexParams`, ki ji nastavimo polja na sledeče vrednosti:

ime polja	*vrednost polja*
`numIterations`	$n_i = 4$
`gravity`	$\{x = 0, y = -10, z = 0\}$
`radius`	$r_r = 0.15$
`solidRestDistance`	$r_s = 0.15$
`fluidRestDistance`	$r_f = 0.10$
`dynamicFriction`	0.2
`staticFriction`	0
`particleFriction`	0.2
`restitution`	0
`adhesion`	0
`sleepThreshold`	0
`maxSpeed`	`FLT_MAX`

maxAcceleration	100
shockPropagation	0
dissipation	0
damping	1
wind	$\{x = 0, y = 0, z = 0\}$
drag	0
lift	0
fluid	true
cohesion	0
surfaceTension	0
viscosity	8
vorticityConfinement	0
solidPressure	0.1
freeSurfaceDrag	0
buoyancy	1
plasticThreshold	0
plasticCreep	0
collisionDistance	$r_p = 0.05$
particleCollisionMargin	0
shapeCollisionMargin	0
numPlanes	5
planes[0]	$\{0, 1, 0, 0\}$
planes[1]	$\{0, 0, 1, 0\}$
planes[2]	$\{1, 0, 0, 0\}$
planes[3]	$\{-1, 0, 0, 4.51\}$
planes[4]	$\{0, 0, -1, 1.54\}$
relaxationMode	eNvFlexRelaxationLocal
relaxationFactor	1

Literatura

[1] M. Müller, D. Charypar, M. Gross, Particle-based Fluid Simulation for Interactive Applications, in: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '03, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 154–159.

[2] M. Macklin, M. Müller, Position Based Fluids, ACM Trans. Graph. 32 (4) (2013) 104:1–104:12. doi:10.1145/2461912.2461984.

[3] J. Stam, Nucleus: Towards a unified dynamics solver for computer graphics, in: 2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics, IEEE, 2009, pp. 1–11. doi:10.1109/CADCG.2009.5246818.

[4] M. Macklin, M. Müller, N. Chentanez, T.-Y. Kim, Unified particle physics for real-time applications, ACM Transactions on Graphics 33 (4) (2014) 153:1–153:12. doi:10.1145/2601097.2601152.

[5] M. Müller, B. Heidelberger, M. Teschner, M. Gross, Meshless Deformations Based on Shape Matching, ACM Trans. Graph. 24 (3) (2005) 471–478. doi:10.1145/1073204.1073216.

[6] G. Irving, J. Teran, R. Fedkiw, Invertible Finite Elements for Robust Simulation of Large Deformation, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA

'04, Eurographics Association, Goslar Germany, Germany, 2004, pp. 131–140. doi:10.1145/1028523.1028541.

[7] T. Pfaff, R. Narain, J. M. de Joya, J. F. O'Brien, Adaptive Tearing and Cracking of Thin Sheets, ACM Transactions on Graphics 33 (4) (2014) 110:1–110:9. doi:10.1145/2601097.2601132.

[8] X. B. Peng, G. Berseth, M. van de Panne, Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning, ACM Transactions on Graphics 35 (4) (2016) 81:1–81:12. doi:10.1145/2897824.2925881.

[9] X. B. Peng, G. Berseth, K. Yin, M. van de Panne, DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning, ACM Transactions on Graphics 36 (4) (2017) 41:1–41:13. doi:10.1145/3072959.3073602.

[10] T. Geijtenbeek, M. van de Panne, A. F. van der Stappen, Flexible Muscle-based Locomotion for Bipedal Creatures, ACM Transactions on Graphics 32 (6) (2013) 206:1–206:11. doi:10.1145/2508363.2508399.

[11] J.-c. Wu, Z. Popović, Realistic Modeling of Bird Flight Animations, ACM Transactions on Graphics 22 (3) (2003) 888–895. doi:10.1145/882262.882360.

[12] D. Pangeršič, Aerodinamika v modelih za računalniško simulacijo letenja ptic v jati, Master's thesis, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani (2015).

[13] J. Tan, Y. Gu, G. Turk, C. K. Liu, Articulated Swimming Creatures, ACM Transactions on Graphics 30 (4) (2011) 58:1–58:12. doi:10.1145/2010324.1964953.

[14] G. Allard, Control of a free-swimming fish using fuzzy logic, IJVR 6 (3) (2007) 23–28.

[15] D. Satoi, M. Hagiwara, A. Uemoto, H. Nakadai, J. Hoshino, Unified Motion Planner for Fishes with Various Swimming Styles, ACM Transactions on Graphics 35 (4) (2016) 80:1–80:15. doi:10.1145/2897824.2925977.

[16] R. Cortez, L. Fauci, N. Cowen, R. Dillon, Simulation of swimming organisms: Coupling internal mechanics with external fluid dynamics, Computing in Science & Engineering 6 (3) (2004) 38–45. doi:10.1109/MCSE.2004.51.

[17] J. Deutscher, A. Blake, I. Reid, Articulated body motion capture by annealed particle filtering, in: Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662), Vol. 2, 2000, pp. 126–133 vol.2. doi:10.1109/CVPR.2000.854758.

[18] J. Deutscher, I. Reid, Articulated body motion capture by stochastic search, International Journal of Computer Vision 61 (2) (2005) 185–205. doi:10.1023/B:VISI.0000043757.18370.9c.

[19] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, D. Silver, Emergence of locomotion behaviours in rich environments, arXiv:1707.02286 (2017).

[20] A. Filella, F. Nadal, C. Sire, E. Kanso, C. Eloy, Model of Collective Fish Behavior with Hydrodynamic Interactions, Phys. Rev. Lett. 120 (2018) 198101. doi:10.1103/PhysRevLett.120.198101.

[21] X. Tu, Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior, Springer-Verlag, Berlin, Heidelberg, 1999.

[22] H. Hu, J. Liu, I. Dukes, G. Francis, Design of 3d swim patterns for autonomous robotic fish, in: Int. Conf. Intell. Robots and Systems, 2006, pp. 2406–2411. doi:10.1109/IROS.2006.281680.

[23] J. Yu, M. Tan, S. Wang, E. Chen, Development of a biomimetic robotic fish and its control algorithm, IEEE Trans. Systems, Man and Cybernetics, B 34 (4) (2004) 1798–1810. doi:10.1109/TSMCB.2004.831151.

[24] J. Liu, H. Hu, Mimicry of sharp turning behaviours in a robotic fish, in: Int. Conf. Robotics and Automation, 2005, pp. 3318–3323. doi:10.1109/ROBOT.2005.1570622.

[25] J. Liu, H. Hu, Biological inspiration: From carangiform fish to multi-joint robotic fish, Journal of Bionic Engineering 7 (1) (2010) 35–48. doi:10.1016/S1672-6529(09)60184-0.

[26] R. J. Clapham, Developing high performance linear carangiform swimming, Ph.D. thesis, University of Essex (2015).

[27] D. Barrett, M. Triantafyllou, D. Yue, M. Grosenbaugh, M. Wolfgang, Drag reduction in fish-like locomotion, Journal of Fluid Mechanics 392 (1999) 183–212.

[28] X. B. Peng, G. Berseth, M. van de Panne, Terrain-adaptive locomotion skills using deep reinforcement learning, ACM Transactions on Graphics 35 (4) (2016) 81:1–81:12. doi:10.1145/2897824.2925881.

[29] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff, Position Based Dynamics, JVCIR 18 (2) (2007) 109 – 118. doi:10.1016/j.jvcir.2007.01.005.

[30] Nvidia CUDA, https://developer.nvidia.com/cuda-zone, dostopano: 2019-10-01.

[31] J. Wakeling, I. Johnston, White muscle strain in the common carp and red to white muscle gearing ratios in fish, Journal of Experimental Biology 202 (5) (1999) 521–528.

[32] JGAP, https://sourceforge.net/projects/jgap/, dostopano: 2019-10-01.

[33] C. Wardle, J. Videler, J. Altringham, Tuning in to fish swimming waves: body form, swimming mode and muscle function, Journal of Experimental Biology 198 (8) (1995) 1629–1636.

[34] J. J. Videler, Fish Swimming, Vol. 10, Springer Science & Business Media, 1993.

[35] I. Borazjani, F. Sotiropoulos, On the role of form and kinematics on the hydrodynamics of self-propelled body/caudal fin swimming, Journal of Experimental Biology 213 (1) (2010) 89–107. doi:10.1242/jeb.030932.

[36] P. Valdivia y Alvarado, Design of biomimetic compliant devices for locomotion in liquid environments, Ph.D. thesis, Massachusetts Institute of Technology (2007).

[37] J. Yu, G. Turk, Reconstructing Surfaces of Particle-based Fluids Using Anisotropic Kernels, ACM Trans. Graph. 32 (1) (2013) 5:1–5:12. doi:10.1145/2421636.2421641.

[38] A. Palstra, V. van Ginneken, G. van den Thillart, Cost of transport and optimal swimming speed in farmed and wild European silver eels (Anguilla anguilla), Comparative Biochemistry and Physiology Part A: Molecular & Integrative Physiology 151 (1) (2008) 37 – 44. doi:10.1016/j.cbpa.2008.05.011.

[39] I. Borazjani, F. Sotiropoulos, Numerical investigation of the hydrodynamics of carangiform swimming in the transitional and inertial flow regimes, Journal of Experimental Biology 211 (10) (2008) 1541–1558. doi:10.1242/jeb.015644.

[40] M. Lentine, M. Aanjaneya, R. Fedkiw, Mass and Momentum Conservation for Fluid Simulation, in: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11,

ACM, New York, NY, USA, 2011, pp. 91–100. doi:10.1145/2019406.2019419.