SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1483

IMPLEMENTACIJA POLUPROCEDURALNE ANIMACIJE KRETNJE LIKOVA U 3D RAČUNALNOJ IGRI

Valentin Berger

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 3. ožujka 2017.

DIPLOMSKI ZADATAK br. 1483

Pristupnik: **Valentin Berger (0036470118)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Implementacija poluproceduralne animacije kretnje likova u 3D računalnoj igri**

Opis zadatka:

Opisati načine animacije likova u 3D računalnim igrama. Posebnu pozornost posvetiti animaciji likova upravljanih igračima te u uvjetima kretanja po neravnoj podlozi. Ostvariti sustav koji za animaciju koristi kombinaciju pristupa analize kretanja kinematskog modela te metodu sklapanja korištenjem ključnih točaka pokreta. U sustav ugraditi detekciju dodira s podlogom te dinamičko prilagođavanje obliku podloge. Ocijeniti ponašanje sustava u uvjetima promjene smjera i brzine kretanja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 29. lipnja 2017.

Mentor:
Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Prof. dr. sc. Siniša Srbljić

Zahvala

Zahvaljujem svojim roditeljima i bratu na podršci koju su mi pružili za vrijeme čitavog dosadašnjeg života i školovanja.

Hvala mentoru prof. dr. sc. Domagoju Jakoboviću na ukazanom povjerenju i pruženoj slobodi prilikom izrade diplomskog rada.

Zahvaljujem cijelom osoblju Croteam-a na ukazanoj podršci prilikom izrade rada te na odličnom sustavu za izradu igara SeriousEngine koji mi je omogućio da izradim ovaj diplomski rad.

Hvala Alenu Ladavcu i Davoru Hunskom na prijedlogu teme i danim savjetima prilikom izrade diplomskog rada.

Hvala Admiru Elezoviću i Nikoli Radoviću za pomoć pri izradi animacija i modela korištenih u diplomskom radu.

Hvala Denisu Ivankoviću i Ivanu Miki za pomoć pri izradi okoline za testiranje sustava koji diplomski rad implementira.

Hvala Karlu Ježu i Robertu Sajku na danim stručnim savjetima i idejama tijekom izrade rada.

Sadržaj

1. Uvod

Računalna animacija danas je neizostavni dio svake računalne igre, a pogotovo je važna za igre koje su fokusirane na realističnost. No, svejedno u proteklih nekoliko desetljeća animacija u igrama nije toliko napredovala kao npr. grafika općenito. Stoga je tako još uvijek moguće vidjeti igre koje su grafički super realistične no izvedba animacija nije ništa realističnija nego npr. u igrama desetljeće prije.

Animacije se danas kod većina igara zasnivaju samo na ključnim pozama (engl. *keyframed animations*) te ne uključuju dodatna proceduralno zasnovana računanja i prilagođavanja za vrijeme njihovog izvođenja. To je uglavnom zbog toga što su animacije zasnovane na ključnim pozama vrlo jednostavne za ostvariti, izgledaju dosta dobro ako se dobro izrade te se mogu vrlo brzo izvoditi, tj. ne zahtijevaju puno računalnih resursa. No, budući da su takve animacije unaprijed napravljene te se ne mogu prilagođavati okolini (svijetu i igri) u kojoj se izvode vrlo je vjerojatno da će se za vrijeme igranja igre dogoditi situacija gdje će primijetiti fizikalni nedostatci takvih animacija. Tako će se npr. u animacijama hodanja koje su zasnovane samo na ključnim pozama dogoditi da takvi likovi, tj. njihove noge, neće realno gaziti po nepravilnim podlogama već će noga ili ulaziti u podlogu ili lebditi nad podlogom.

Kao što je navedeno postoje i drugi načni animiranja likova u igri, a to je uz pomoć proceduralne animacije. Taj način animiranja zasnovan je na kinematičkim i fizikalnim modelima. Likovi i igre koje koriste takvu vrstu animacija neće imati problema s navedenim problemima gaženja po podlozi kod hodanja te će u tom smislu biti realnije, no kod samo takvog načina animiranja može se dogoditi da likovi ne izgledaju prirodno, tj. da više po kretnjama podsjećaju na robote umjesto na živa hodajuća bića. Osim toga, proceduralne animacije zahtijevaju puno više računalnih resursa, nego animacije zasnovane na ključnim pozama.

Zbog svega navedenog postavlja se pitanje koji način animiranja je bolji, onaj zasnovan na ključnima pozama ili proceduralni, te postoji li način da ne moramo raditi kompromis između fizikalne realnosti i stilističke prirodnosti animacija? Odgovor je da postoje bolji načini, bez kompromisa. Jedan takav način bit će opisan

i ostvaren u ovome radu. Drugi mogući načini bit će spomenuti na kraju rada, ali neće biti dodatno razmatrani u sklopu ovog rada.

Cilj ovog rada je ostvariti sustav animiranja kretnje likova tako da budu zadovoljeni kriteriji stilističke i fizikalne realnosti. Takav način animiranja neće biti niti samo proceduralan, niti samo zasnovan na ključnim pozama već će kombinirati obje tehnike, tj. bit će polu-proceduralan. Potrebno je nadograditi postojeći sustav animiranja kretnje likova u sustavu za izradu igara Serious Engine. Sustav je prije ove nadogradnje podržavao je samo osnovni sustav animiranja kretnje likova koristeći samo predefinirane animacije zasnovane na ključnim trenutcima uz neka osnovna dodatna podešavanja kostura, tj. podešavanja korijena kostura kada se naiđe na nepravilnu podlogu.

2. Opis problema

U prošlom uvodnom poglavlju ustanovili smo zadatak ovoga rada, sada nam još preostaje detaljnije opisati domenu problema te sve preduvjete i uvjete koji se moraju ostvariti kako bi sustav bio zadovoljavajuće kvalitete.

Animacijski sustav zahtijeva da likovi koje animira imaju noge te da hodaju s fiksnim brojem nogu po podu (npr. da ne hodaju malo na dvije pa malo na četiri noge). To znači da likovi moraju imati barem jednu nogu za kretanje te ta noga mora biti sastavljena barem od tri zgloba (kuk, gležanj i stopalo). Također, sustav mora jednako dobro animirati likove kojima upravlja igrač kao i one kojima upravlja umjetna inteligencija (engl. *artificial intelligence*). Zatim, likovi koje se animiraju moraju u svakome trenutku biti opisani pozicijom i orijentacijom u svijetu kojem se nalaze, na temelju čijih promjena se mogu računati brzine, kutne brzine i akceleracija kretanja.

Rečeno je da je sustav polu-proceduralan jer će koristiti i animacije s ključnim pozama, ali i proceduralne tehnike animiranja. To znači da je ideja koristiti animacije zasnovane na ključnim pozama kako početne poza za proceduralni dio prilagođavanje nogu podlozi po kojoj se lik kreće. Način koliko će koja animacija utjecati na tu početnu pozu bit će određena na temelju iznosa i smjera trenutne brzine (detaljnije u poglavlju 4.2).

Svaka animacija koju sustav koristi mora biti ciklička tj. u animaciji prva i zadnja poza animacijskog kostura mora biti jednaka kako bi prijelazi između dva ciklusa kretnje bili glatki te da sustav analize animacije bude precizniji. Sustav zahtijeva da animacije prikazuju hodanje ili trčanje lika određenom brzinom u određenom smjeru. Također, sustav dozvoljava različite stilove animacija za hodanje u određenom smjeru, određenom brzinom, ali takve animacije moraju biti dodijeljene različitim animacijskim grupama (detaljnije pojašnjenje u poglavlju 4.3). Sustav po grupi dozvoljava jednu animaciju koja prikazuje mirujuću pozu (engl. *idle pose*), a koja predstavlja kretnje brzinom nula.

Nadalje svaka animacija mora proći proces analize kojom se određuju sve potrebne informacije o kretnji lika u toj animaciji. Proces analize bit će iscrpno opisan u sljedećem 3. poglavlju. Informacije dobivene analizom omogućit će nam odabir

poze u svakome trenutku kretanja lika, a na koju ćemo tada primijeniti proceduralne tehnike prilagođavanja nogu podlozi. Proceduralna tehnika će između ostalog koristiti i inverznu kinematiku kao tehniku koja se koristi kod svih kinematičkih modela kretanja.

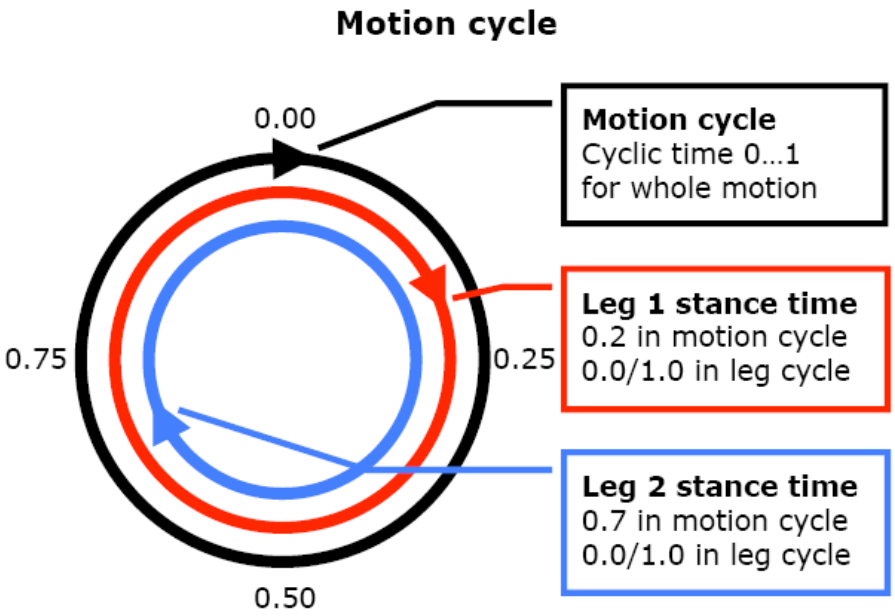
3. Analiza kretnje

U ovome poglavlju će se objasniti postupak automatske analize animacije kretnje lika. Cilj postupka analize je odrediti ključne vremenske trenutke (engl. *keytimes*) animacije kretnje za svaku nogu te brzinu i smjer kretnje nogu i cijelog tijela. Određivanje navedenih informacija se zasniva na analizi krivulja kretnje pete i palca stopala svake noge. Analiza cijele kretnje se obavlja za vrijeme dizajniranja igre te se sve potrebne informacije o animaciji kretnje lika čuvaju i koriste za vrijeme izvođenja igre. Zbog toga ovaj postupak analize kretnje lika ne utječe na brzinu izvođenja igre.

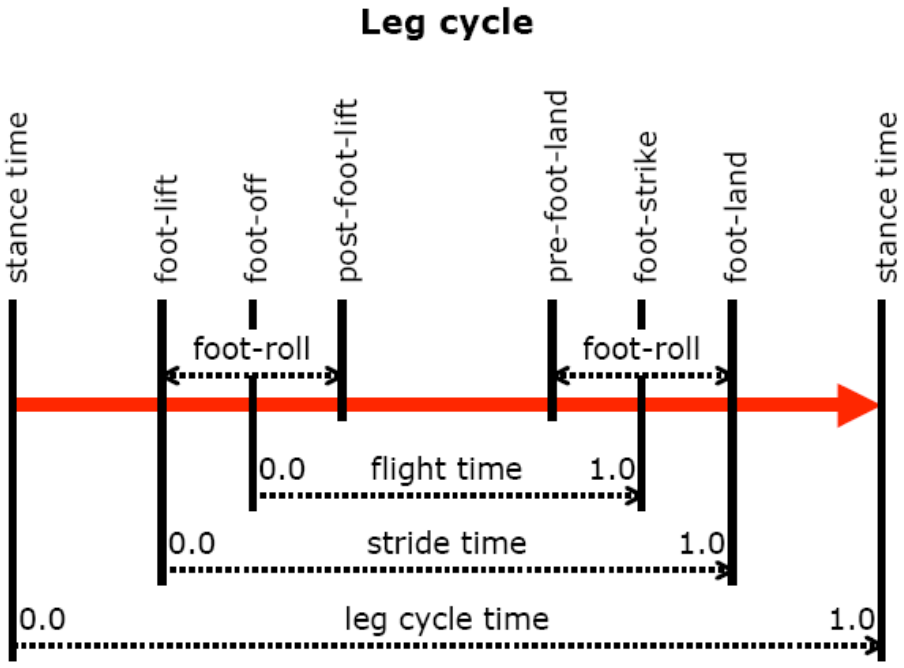
Ključni trenutci animacije određuju se za svaku nogu zasebno. Glavni ključni trenutci su:

- ***Stance time*** – trenutak kada stopalo noge leži na podlozi punom površinom.
- ***Foot-lift*** – trenutak kada se stopalo noge počinje podizat od podloge tj. više nije punim obujmom na podlozi.
- ***Foot-off*** – trenutak kada se stopalo potpuno odvojilo od podloge tj. više ne dodiruje podlogu.
- ***Foot-strike*** – trenutak kada stopalo dodirne podlogu nekim svojim dijelom, a nakon što je bilo potpuno u zraku.
- ***Foot-land*** – trenutak kada stopalo potpunim obujmom dodirne podlogu.

Kako je moguće da peta i palac u istom trenutku dodirnu/napuste podlogu tj. da *foot-strike* i *foot-land* trenutci, odnosno *foot-lift* i *foot-off* trenutci budu jednaki, određuju se dodatni ključni vremenski trenutci ***pre-foot-land*** i ***post-foot-lift***. Oni se ne određuju direktno kao glavni trenutci već se računaju pomoću njih. Važno je napomenuti da su vremenski trenutci unutar domene [0,1] ([početak, kraj] animacije), odnosno trenutak *stance-time* se određuje unutar ciklusa animacije kretnje (Slika 1), dok se ostali vremenski trenutci određuju relativno u odnosu na trenutak *stance-time* (Slika 2).



Slika 1 – Prikaz ciklusa nogu unutar ciklusa animacije kretnje. (Johansen, 2009)



Slika 2 – Prikaz ključnih vremenski trenutaka unutar ciklusa noge. (Johansen, 2009)

Osim ključnih trenutaka, potrebno je još odrediti duljinu stopala (engl. ***foot length***) i smjer stopala (engl. ***foot direction***) u trenutku *stance-time*, točku/pivot (engl. ***footbase***) oko koje će se stopalo rotirati za vrijeme prilagodbe podlozi, te

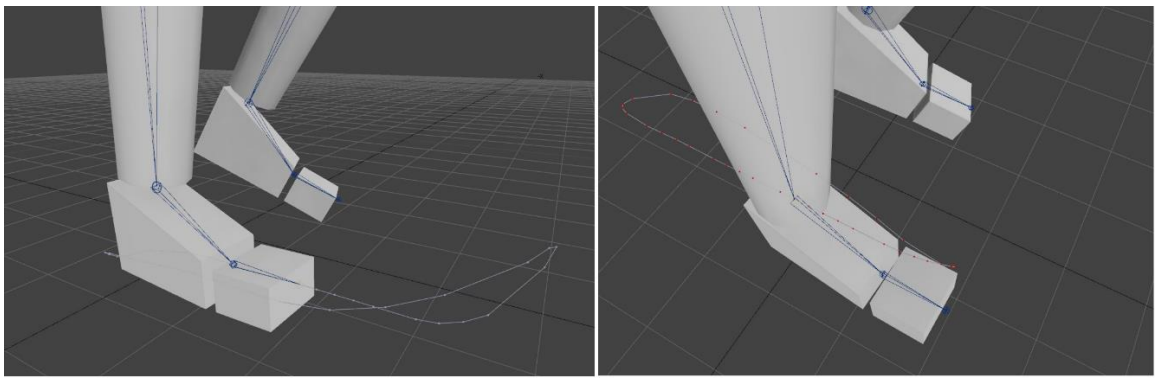
normaliziranu putanju točke *footbase* za vrijeme jednog ciklusa. Detaljnije objašnjenje koncepta pojma *footbase* biti će objašnjeno u narednim poglavljima.

Prije objašnjenja postupka analize još je potrebno odrediti preduvjete i pretpostavke vezane za samu animaciju koju analiziramo i sustav u cjelini:

- Animacija kretnje treba biti ciklička i sadržavati točno jedan korak za svaku nogu lika kojeg animira.
- Stopala lika u animaciji bi po mogućnosti trebala biti u najnižoj poziciji kada stopalo leži na podlozi. Sustav donekle može podnijeti presijecanje podloge i stopala, no veće presijecanje uzrokuje veću vjerojatnost pojave pogrešaka tijekom analize.
- Dok stopalo leži na podlozi (u animaciji) ono se mora kretati pravocrtno konstantnom brzinom u odnosu na podlogu. Animacija koja ne zadovoljava ovu pretpostavku vjerojatno će imati problema s proklizavanjem nogu za vrijeme kretnje lika u igri. Opet, odstupanja su dozvoljena u određenim granicama, nakon kojih se mogu očekivati pogreške u analizi.
- Očekuje se da animacija prikazuje kretnju lika na potpuno ravnoj horizontalnoj podlozi.
- Postupak analize ne pretpostavlja koja noga će prvo zakoračiti i kojim dijelom stopala (peta ili prsti), niti u kojem trenutku.

3.1 Analiza kretnje nogu

Prvi korak u analizi animacije je samo uzorkovanje iste, tj. uzorkovanje krivulje kretnje palca i pete svake noge za vrijeme trajanja animacije (Slika 3).



Slika 3 – Prikaz krivulja kretnje pete i palca stopala noge.

Nakon toga određuje se krivulja srednje vrijednosti između krivulje palca i pete tako da se pronađe aritmetička sredina za svaki uzorak pete i palca:

$$\overrightarrow{\mathbf{middle}_s} = \frac{\overrightarrow{\mathbf{heel}_s} + \overrightarrow{\mathbf{toe}_s}}{2} \tag{1}$$

Zatim se dobivena srednja krivulja projicira (operator ||) na podlogu (**ground**), te se određuje točka **c** kao prosjek svih uzoraka prosječne krivulje:

$$\overrightarrow{\mathbf{middle}_s} = \overrightarrow{\mathbf{middle}_s} \,||\, \mathbf{ground} \tag{2}$$

$$\vec{\mathbf{c}} = \frac{1}{N}\sum_{s}^{N} \overrightarrow{\mathbf{middle}_s} \tag{3}$$

Zatim se između svih točaka srednje krivulje određuje točka **a** koja je najudaljenija (operator *argmax*) od točke **c** te točka **b** koja je najudaljenija od točke **a**:

$$\vec{\mathbf{a}} = \overrightarrow{\mathbf{middle}}_{\underset{s}{\operatorname{argmax}}\,(\|\overrightarrow{\mathbf{middle}_s} - \vec{\mathbf{c}}\|)} \tag{4}$$

$$\vec{\mathbf{b}} = \overrightarrow{\mathbf{middle}}_{\underset{s}{\operatorname{argmax}}\,(\|\overrightarrow{\mathbf{middle}_s} - \vec{\mathbf{a}}\|)} \tag{5}$$

Tada se vektor smjera kretanja noge **axis** određuje kao:

$$\overrightarrow{\mathbf{axis}} = \vec{\mathbf{b}} - \vec{\mathbf{a}} \tag{6}$$

Nakon što se zna smjer kretnja noge može se odrediti uzorak kada se noga nalazi punim obujmom na podlozi *stance-time*. To se određuje tako da se za svaki

uzorak *s* odredi cijena (engl. *cost*) te se uzorak s najmanjom cijenom smatra trenutkom *stance-time*. Računanje cijene uzorka temelji se na iznosu visine pete/palca te udaljenosti uzorka srednje krivulje od točke **c**:

$$cost_s = \frac{max(heelheight_s, toeheight_s)}{\alpha} + \frac{\left|\left(\overrightarrow{\mathbf{middle}_s} - \vec{\mathbf{c}}\right) * \hat{\mathbf{axis}}\right|}{\beta} \qquad (7)$$

Gdje *α* i *β* parametri određuju je li važnije da je *stance-time* uzorak što bliže podlozi ili sredini srednje krivulje kretnje noge. U implementaciji sustava *α* je postavljen na najveći iznos visine pete ili palca između svih uzoraka, a *β* je postavljen na ‖**b** – **a**‖.

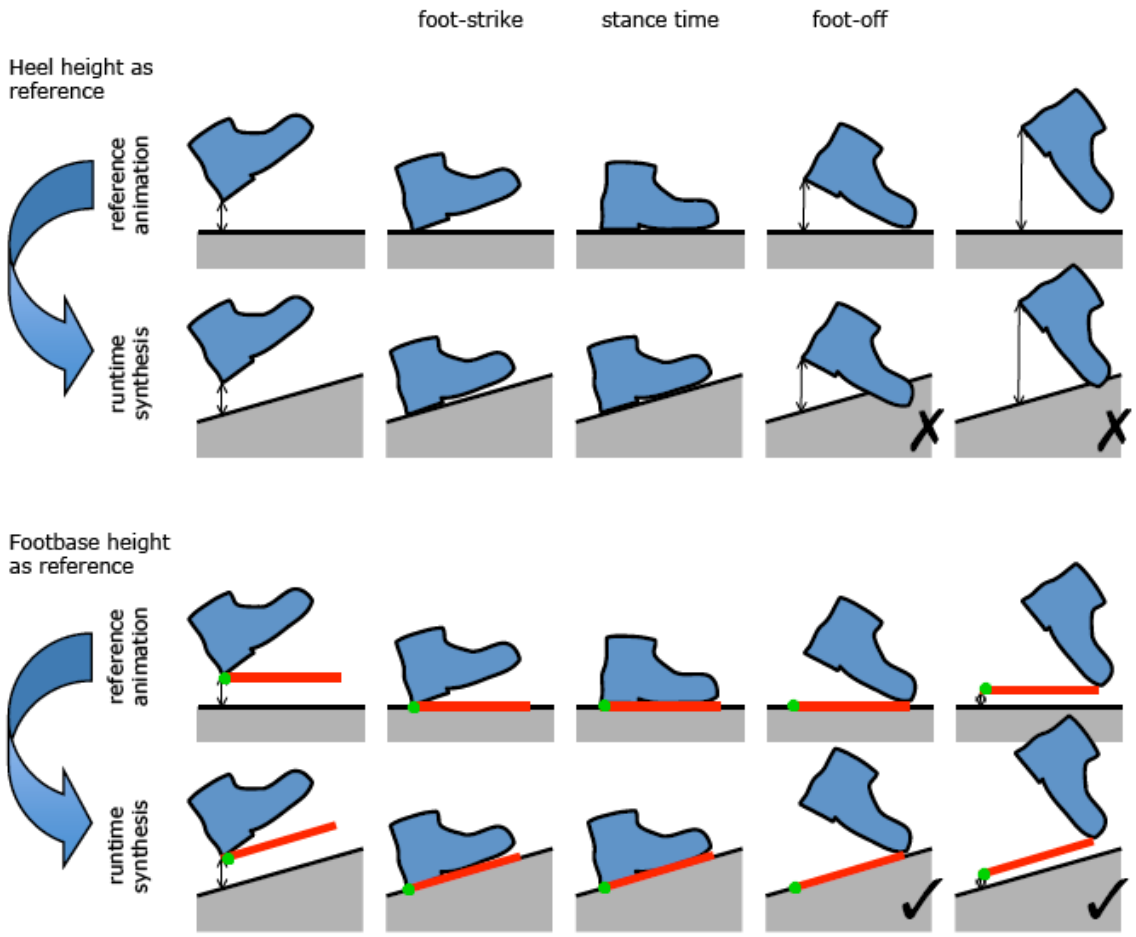
Nakon što je određen *stance-time* trenutak, uzorak tog trenutka se koristi za određivanje smjera i duljine stopala:

$$\overrightarrow{\mathbf{footdirection}} = \hat{\mathbf{d}} \cdot footlength \qquad (8)$$

$$\vec{\mathbf{d}} = \left(\overrightarrow{\mathbf{toe}} - \overrightarrow{\mathbf{heel}}\right) \,||\, \mathbf{ground} \qquad (9)$$

Sljedeći korak analize je odrediti bazu stopala (engl. *footbase*) svake noge za svaki uzorak animacije kretnje. Baza stopala je dio pravca (dužina) koja opisuje nagib podloge relativno u odnosu na stopalo noge. Odnosno, u animaciji kretnje podloga je uvijek ravna ploha, no u stvarnosti podloga je nepravilna, tj. ima neki nagib, stoga je potrebno za vrijeme analize animacije kretnje odrediti položaj podloge relativan u odnosu na stopalo za svaki uzorak kako bi se za vrijeme

izvođenja igre stopalo moglo prilagođavati podlozi tako da relativni odnos noge i stopala ostane sačuvan (Slika 4).



Slika 4 – Prikaz koncepta korištenja baze stopala (engl. footbase) kao referencu za u odnosu na korištenje samo visine pete kao referencu za prilagodbu noge podlozi. (Johansen, 2009)

Baza stopala je određena točkom *footbase* te smjerom stopala koji je uvijek paralelan s podlogom te stopalo bazu stopala uvijek dodiruje petom i/ili palcem. Na prethodnoj slici točka *footbase* je označena zelenom bojom dok je smjer stopala označen crvenom linijom. Kako bi se odredila točka *footbase* potrebno je odrediti koji dio stopala je bliže podlozi, tj. balans stopala. Sljedeća formula prikazuje funkciju izračuna balansa stopala na temelju visine pete i palca te duljine stopala:

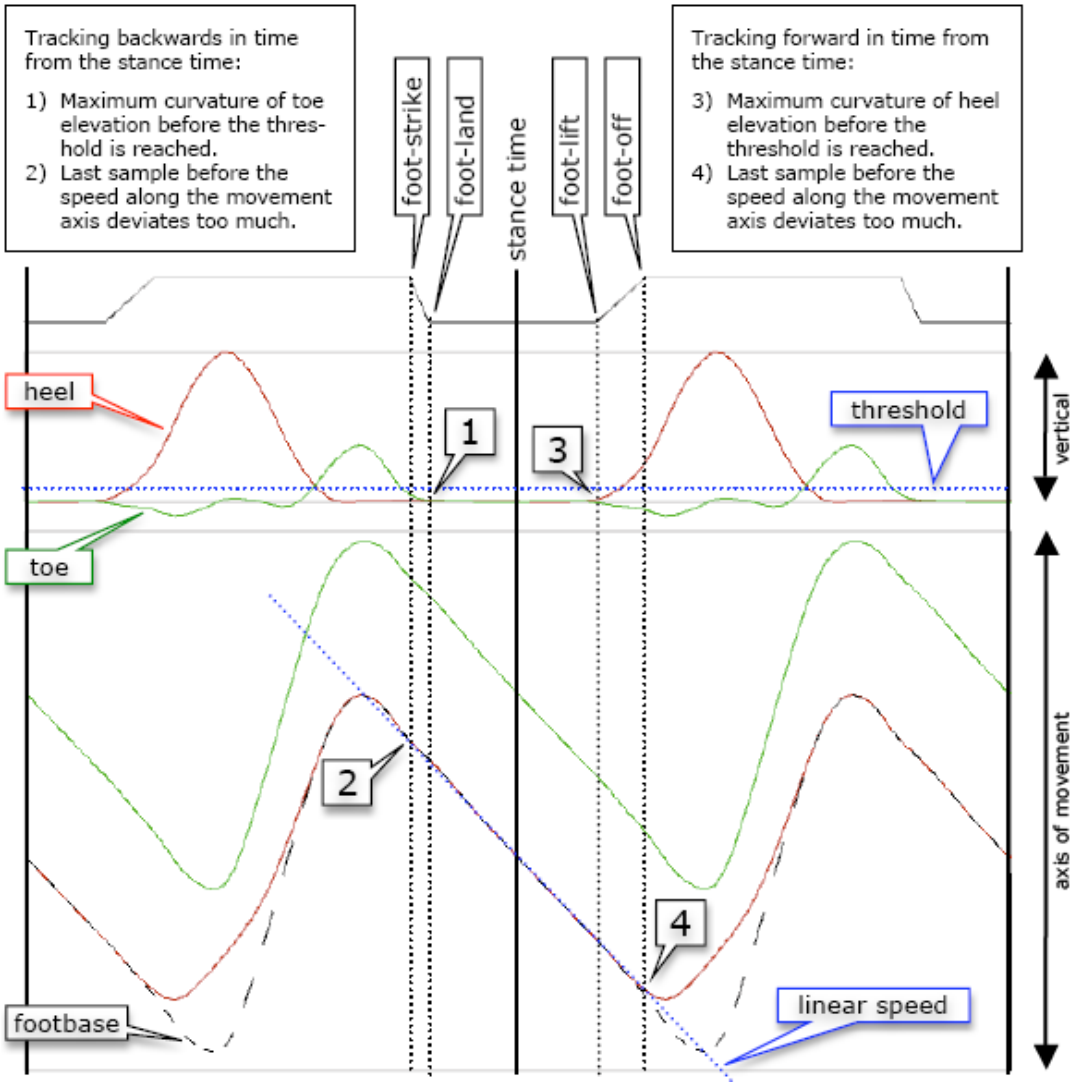
$$balance_s = \frac{\tan^{-1}\left(\frac{heelheight_s - toeheight_s}{footlength} \cdot \alpha\right)}{\pi} + 0.5 \qquad (10)$$

Iz formule se može vidjeti da balans teži k nuli što je visina palca veća od visine pete, tj. teži k jedan što je visina pete veća u odnosu na visinu palca. Parametar α označuje pristranost funkcije k ekstremima (α > 1) odnosno srednjoj vrijednosti od 0.5 (α < 1). Uz pomoću vrijednosti balansa točka **footbase** se određuje kao:

$$\overrightarrow{\mathbf{footbase}_s} = \overrightarrow{\mathbf{heel}_s} * (1 - balance_s) + \left(\overrightarrow{\mathbf{toe}_s} - \overrightarrow{\mathbf{footdirection}}\right) \cdot balance_s \quad (11)$$

Točka *footbase* se računa za svaki uzorak animacije kretnje, za svaku nogu.

Nakon što imamo uzorkovane putanje pete, palca i točke *footbase* možemo odrediti sve ostale ključne trenutke animacije kretnje. Sljedeća slika (Slika 5) prikazuje navedene krivulje, a postupak određivanja ključnih trenutaka iz njih je sljedeći:



Slika 5 – Slika prikazuje krivulju kretnje vertikalne komponente palca i pete, te krivulju kretnje palca, pete i točke footbase projiciranu na smjer kretnje. (Johansen, 2009)

- Pratimo krivulju kretnje vertikalne komponente pete kroz vrijeme od trenutka *stance-time* pa unaprijed, sve do određene granične (engl. *threshold*) visine. Trenutak u kojemu je zakrivljenost krivulje najveća određuje se kao *heel-off* trenutak.
- Ponavljamo prethodni postupak za krivulju palca, a dobiveni trenutak proglašavamo kao *toe-off* trenutak.
- Pratimo krivulju kretnje točke *footbase* projiciranu na smjer kretnje od trenutka *stance-time* pa unaprijed. Za vrijeme trenutka stance-time pa do nekog trenutka u budućnosti stopalo stoji čvrsto na podlozi te se kreće konstantnom brzinom u odnosu na lika. Ta konstantna brzina se računa, a

trenutak kada brzina počne odstupati od te izračunate brzine proglašava se *linear-speed-end* trenutak.

- Trenutci *foot-off*, *foot-lift* i *post-foot-lift* se tada računaju kao:

  foot-lift = min(heel-off, toe-off, linear-speed-end)

  foot-off = max(heel-off, toe-off)

  post-foot-lift = max(foot-lift + α, foot-off)

  Gdje je parametar α predstavlja minimalno trajanje kotrljanja stopala, odnosno vrijeme od trenutka kada peta dodirne podlogu pa do trenutka kada palac napusti podlogu.

- Postupak se ponavlja na analogan način, ali unazad od trenutka stance-time kako bi se odredili trenutci *heel-strike*, *toe-strike* i *linear-speed-begin* trenutak.
- Trenutci *foot-land*, *foot-strike* i *pre-foot-land* se tada računaju kao:

  foot-land = max(heel-strike, toe-strike, linear-speed-begin)

  foot-strike = min(heel-strike, toe-strike)

  post-foot-lift = min(foot-land - α, foot-strike)

3.2 Analiza ciklusa kretnje

Nakon što imamo određenu putanju točke *footbase* te trenutke *foot-strike* i *foot-off* trenutke za svaku nogu možemo odrediti informacije o ciklusu kretnje lika, a to su brzinu noge, smjer i duljinu koraka noge lika:

$$\overrightarrow{\mathbf{displacementvector}} = \overrightarrow{\mathbf{footbase}}_{footoff} - \overrightarrow{\mathbf{footbase}}_{footstrike} \quad [m] \qquad (12)$$

$$stridevelocity = \frac{\|\mathbf{displacementvector}\|}{footoff - footstrike} \quad \left[\frac{m}{cycle}\right] \qquad (13)$$

$$\overrightarrow{\mathbf{stridedirection}} = -\widehat{\mathbf{displacementvector}} \qquad (14)$$

$$\boldsymbol{strdedistance} = stridevelocity \cdot 1[cycle] \qquad [m] \qquad (15)$$

Nakon što imamo određene brzine, duljine i smjer koraka svake noge zasebno možemo odrediti brzinu i smjer kretnje lika:

$$cyclevelocity = \frac{\sum_{l=1}^{legs} stridevelocity_l}{legs} \qquad \left[\frac{m}{cycle}\right] \tag{16}$$

$$\overrightarrow{\mathbf{cycledirection}} = \frac{\sum_{l=1}^{legs} \overrightarrow{\mathbf{stridedirection}_l}}{legs} \tag{17}$$

$$cycledistance = cyclevelocity \cdot 1[cycle] \qquad [m] \tag{18}$$

U prijašnjim poglavljima navedeno je da bi brzine nogu lika kojeg se animira trebale biti jednake, no kako je moguće da to ne bude savršeno zadovoljeno aritmetička sredina brzina svih nogu služi kako bi se greška podjednako raspodijelila na sve noge.

Posljednji korak analize animacije kretnje lika je transformirati putanju krivulje kretnje točke *footbase* svake noge iz lokalnog prostora u globalni prostor. Također takvu putanju treba normalizirati po z osi tako da korak uvijek ide od (x1, y1, 0) do (x2, y2, 1):

$$cycletime_s = time_s - stancetime_s \tag{19}$$

$$\overrightarrow{\mathbf{reference}_s} = \overrightarrow{\mathbf{stanceposition}_l} - (stridevelocity_l \cdot \overrightarrow{\mathbf{stridedirection}_l} \cdot cycletime_s) \tag{20}$$

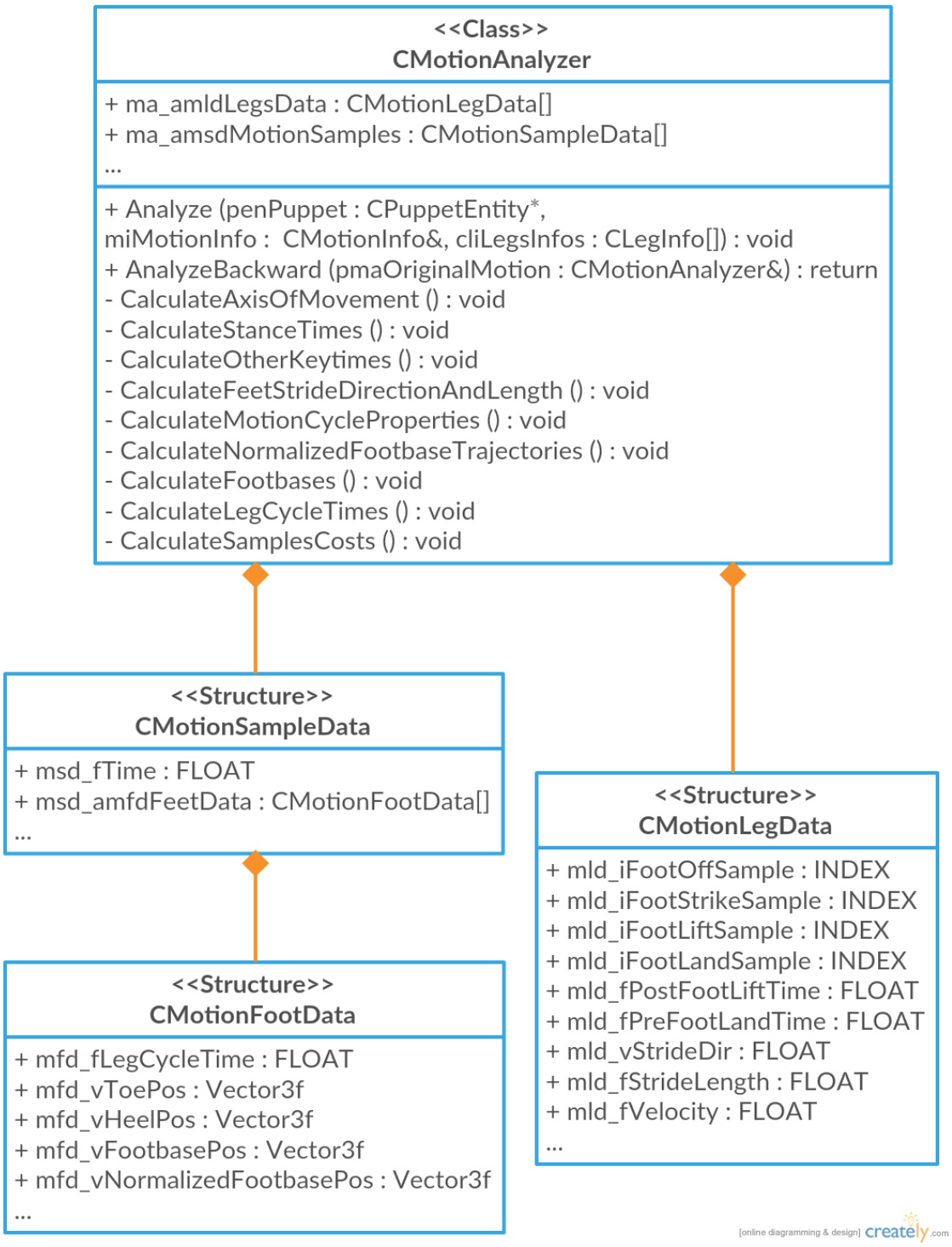
$$\overrightarrow{\mathbf{normalizedfootbase}_s} = \mathbf{Q} \cdot (\overrightarrow{\mathbf{footbase}_s} - \overrightarrow{\mathbf{reference}_s}) \tag{21}$$

$$\overrightarrow{\mathbf{normalizedfootbase}_s}.z = \frac{\overrightarrow{\mathbf{normalizedfootbase}_s}.z}{stridedistance} \tag{22}$$

Rotacija **Q** predstavlja rotaciju potrebnu da se vektor **stridedirection** rotira u vektor (0, 0, 1).

3.3 Programsko ostvarenje

Za potrebe analize animacije izrađen je razred `CMotionAnalyzer` koji implementira sve navedeno u ovome poglavlju. Na sljedećoj slici (Slika 6) nalazi se dijagram razreda koji prikazuje dio koda koji sluzi za tu analizu.



Slika 6 – Prikaz dijagrama razreda za dio koda koji obavlja analizu animacije kretnje.

Nakon što se pozove funkcija `Analyze` razreda `CMotionAnalyzer` animacija s navedenim imenom se uzorkuje na dijelove koji su opisani razredom `CMotionSampleData`. Informacije o kretnji i nogama lika potrebne za analizu proslijeđene su u funkciji `Analyze` preko navedenih parametara. Informacije proizašle iz analize navedene u ovom poglavlju spremaju se u objekt tipa `CMotionLegData` za svaku nogu zasebno. Razred `CMotionFootData` čuva podatke o stopalima nogu za svaki uzorak animacije. Funkcija `AnalzyeBackward` služi da se neka kretnja obradi tako da rezultat analize bude kretnja u suprotnom smjeru. Podatci o svakoj kretnji nakon analize pohranjeni su u analizatoru te se preko razreda `CMotionAnalyzer` koriste u dijelu koda koji obavlja samu polu-proceduralnu animaciju. Osim navedenih metoda razred `CMotionAnalyzer` ima i još neke dodatne metode koje pomažu pri analizi i pristupanju analiziranim podatcima, ali nisu navedeni u dijagramu.

4. Kombiniranje i interpolacija animacija

Kao što je u uvodnim poglavljima spomenuto, za što kvalitetnije funkcioniranje našeg sustava animacije potrebno je imati više različitih primjera animacija kretnje lika. Iako sustav može funkcionirati i sa samo jednom animacijom kretnje, i to animacije hoda, preporučeno je koristiti sljedeći skup animacija:

1. Mirovanje (engl. *idle*)
2. Hodanje/trčanje naprijed (engl. *forward walk/run*)
3. Hodanje/trčanje unazad (engl. *backward walk/run*)
4. Hodanje/trčanje u lijevo (engl. *left strafe walk/run*)
5. Hodanje/trčanje u desno (engl. *right strafe walk/run*)

Opcionalno, sustav podržava da se svaka animacija kretnje može upotrijebiti i za kretnju u suprotnom smjeru. Pa tako se animacija za hodanje unaprijed može koristiti za hodanje unazad, animacije za hodanje udesno za hodanje u lijevo itd. Naravno, kvaliteta takvih kretnji za koje nisu izrađene originalne animacije nisu kvalitetne kao kada su izrađene sa zasebnim animacijama.

4.1 Sinkronizacija kretnji

Na početku prethodnog poglavlja rečeno je da animacije za različite kretnje ne moraju imati identične relativne ključne trenutke, tj. npr. trenutak kada noga zagazi podlogu u jednoj kretnji se može dogoditi na početku animacije, a u drugoj kretnji na kraju animacije. Također, rečeno je da se ključni trenutak *stance-time* određuje apsolutno u odnosu na početak animacije, a da se ostali ključni trenutci određuju relativno u odnosu na njega. Zbog toga je potrebno ostvariti mehanizam sinkronizacije koji će trenutke *stance-time* sinkronizirati tako da budu jednaki za sve animacije (koliko je to moguće).

Postupak sinkronizacije se svodi na određivanje vremenskog pomaka (engl. *time offset*) za svaku animaciju kretnje te kada te pomake dodamo na trenutke *stance-time* svake kretnje dobijemo vremenski iznos koji je jednak za sve kretnje. Vremenski pomak je unutar domene [0, 1] kao i svi drugi vremenski trenutci. Algoritam za određivanje vremenskih pomaka dan je na sljedećem pseudokodu (Kod 1). Algoritam je iterativan i ograničen na fiksni broj iteracija koji je postavljan

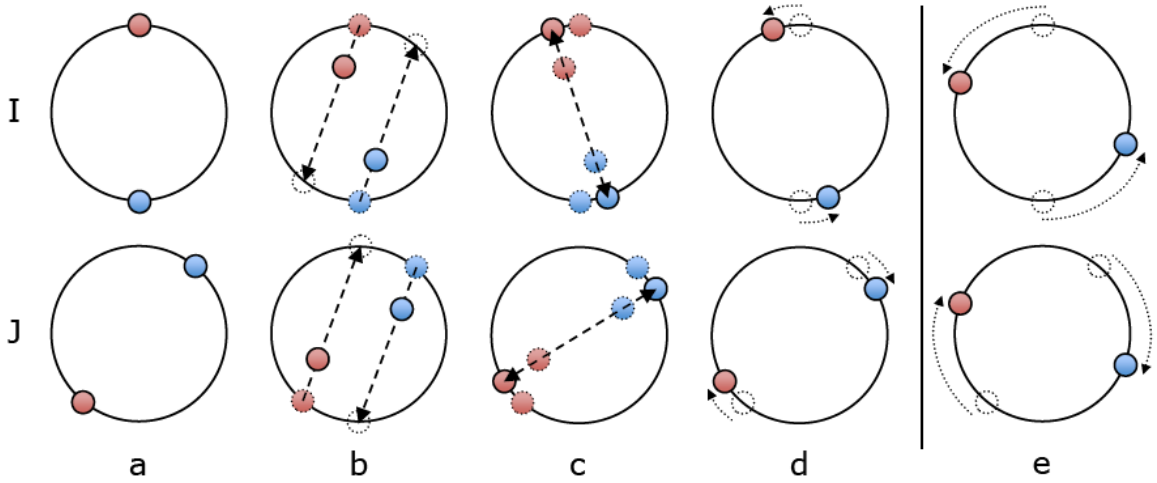
na neki dovoljno veliki broj kako bi maksimalna promjena u jednoj iteraciji mogla pasti na dovoljno nisku razinu.

```
SyncMotions(motions[], legs[]) {
  nMotions = Count(motions);
  nLegs = Count(legs);
  offsets[] = {0};
  changeRatio = 5.0f / (0.5 * (N * N - N));
  iteration = 0;
  while (iteration++ < 100) {
    offsetsChanges[] = {0};
    for (i = 1 to nMotions) {
      motionA = motions[i];
      for (j = 0 to nMotions) {
        motionB = motions[i];
        for (leg = 0 to nLegs) {
          stanceTimeA = motionA.GetStanceTime(leg) + offsets[i];
          stanceTimeB = motionB.GetStanceTime(leg) + offsets[j];
          // First motion point on time circle.
          vA = Vector(sin(stanceTimeA * 2PI), cos(stanceTimeA * 2PI))
          // Second motion point on time circle.
          vB = Vector(sin(stanceTimeB * 2PI), cos(stanceTimeB * 2PI))
          // Tetive that connect that two points.
          vTetive = vB - vA;
          // Move points towards each other.
          vNewA = vA + vTetive * 0.1f;
          vNewB = vB - vTetive * 0.1f;
          // Convert points back to time.
          newStanceTimeA = mod(atan2(vNewA.y, vNewA.x) / 2.0f / PI);
          newStanceTimeB = mod(atan2(vNewB.y, vNewB.x) / 2.0f / PI);
          changeA = mod(newStanceTimeA - stanceTimeA);
          changeB = mod(newStanceTimeB - stanceTimeB);
          if (fChangeA > 0.5f) {
            fChangeA -= 1.0f;
          }
          if (fChangeB > 0.5f) {
            fChangeB -= 1.0f;
          }
          offsetsChanges[iMotion] += changeA * changeRatio;
          offsetsChanges[jMotion] += changeB * changeRatio;
          // Apply new offset changes
          offsets += offsetsChanges;
          // Find max change
          maxChange = Max(offsetsChanges)
          if (maxChange < 0.0001f)
            break;
          }
        }
      }
  }

  return offsets;
}
```

Kod 1 – Pseudo-kod algoritma za određivanje sinkronizacijskih pomaka trenutaka stance-time između različitih animacija kretnji.

Algoritmom se kaže da se za svaki par animacija kretnji koje se sinkronizira treba učiniti sljedeći postupak (Slika 7):

a) Trenutci *stance-time* za animaciju I i J su nesinkronizirani.
b) Trenutci stance-time lijeve (crvena točka) i desne (plava točka) noge se pretvaraju u 2D točke na jediničnoj kružnici koja predstavlja ciklus animacije. Točke animacija I i J (trenutci) se privuku jedna prema drugoj po vektoru koji spaja te dvije točke.
c) Točke se vrate na kružnicu, ali po vektoru koji spaja točke lijeve i desne noge iste animacije.
d) 2D točke se pretvore nazad u vrijeme. Postupak se ponovi za ostale parove animacija.
e) Cijeli postupak se iterativno ponavlja dok sve animacije nisu sinkronizirane.



Slika 7 – Slikovni prikaz algoritma sinkronizacije animacija kretnje. (Johansen, 2009)

4.2 Interpolacija kretnji

Na početku poglavlja naveli smo da naš sustav zahtijeva skup od nekoliko različitih animacija, za različite smjerove i brzine kretanja, koje se koriste za prikaz kretnje lika. Također smo u uvodnim poglavljima naveli da je sustav polu-proceduralan jer još uvijek koristi animacije zasnovane na ključnim točkama pokreta (engl. *keyframed animations*). Zbog tih zahtjeva potrebno je izraditi sustav koji u

svakom trenutku može iz skupa animacija dobiti željenu pozu lika koja će se dalje koristiti za proceduralno prilagođavanje nogu podlozi.

Svaka animacija (a_i) iz našeg skupa primjera kretnji je određena brzinom i smjerom (v_i) kretanja lika kojeg animira. Te brzine su određene za vrijeme postupka analize animacija kretnje. Dalje, potrebno je uz pomoć tih brzina odrediti funkcije utjecaja h_i (engl. *influence function*), odnosno težine w_i (engl. *weights*), koje nam govore koliko je u određenom trenutku, pri određenoj brzini lika, koja animacija bitna. Brzinu kretanja lika možemo smatrati parametrom (**p**) interpolacije, a težine w_i su normalizirane vrijednosti funkcija utjecaja h_i te je njihova suma jednaka 1:

$$w_i(\vec{\mathbf{p}}) = \frac{h_i(\vec{\mathbf{p}})}{\sum_{j=1}^{n} h_j(\vec{\mathbf{p}})} \tag{23}$$

Nadalje, potrebno je odrediti interpolacijsku metodu, tj. njen inverz, uz pomoć koje će se u svakome trenutku za neku brzinu $\boldsymbol{v}$, tj. parametar $\boldsymbol{p}$, odrediti težine w_i za svaku od poznatih brzina $\boldsymbol{v}_i$ koje predstavljaju animacije kretnje lika (a_i). Da bi interpolacijska metoda bila valjana ona mora zadovoljiti sljedeće kriterije:

- **Egzaktnost** – Ako je brzina $\boldsymbol{v}$ jednaka brzini jednoj od brzina primjera animacije $\boldsymbol{v}_i$ tada je težina w_i za tu animaciju jednaka 1, a sve ostale težine za sve ostale animacije jednake su 0.
- **Suma** – Zbroj svih težina w_i za sve animacije kretnje mora biti jednak 1.
- **Kontinuitet** – Interpolacijska funkcija mora biti glatka, tj. male promjene u iznosu parametra $\boldsymbol{p}$, odnosno brzini $\boldsymbol{v}$, uzrokuju male promjene u iznosu težina w_i.

Osim navedena tri kriterija poželjno je da interpolacijska metoda zadovoljava i sljedeće kriterije:

- **Lokalnost** – Samo primjer animacije čija je brzina blizu interpolirane brzine **v** trebaju imati težine veće od nule dok ostale težine, ostalih animacija, trebaju biti nula. Ovaj kriteriji je poželjan jer omogućava da je u jednom trenutku potrebno samo kombinirati *(*engl. *blend)* između nekoliko animacija koje imaju pozitivne vrijednosti težina, a ne svih animacija.

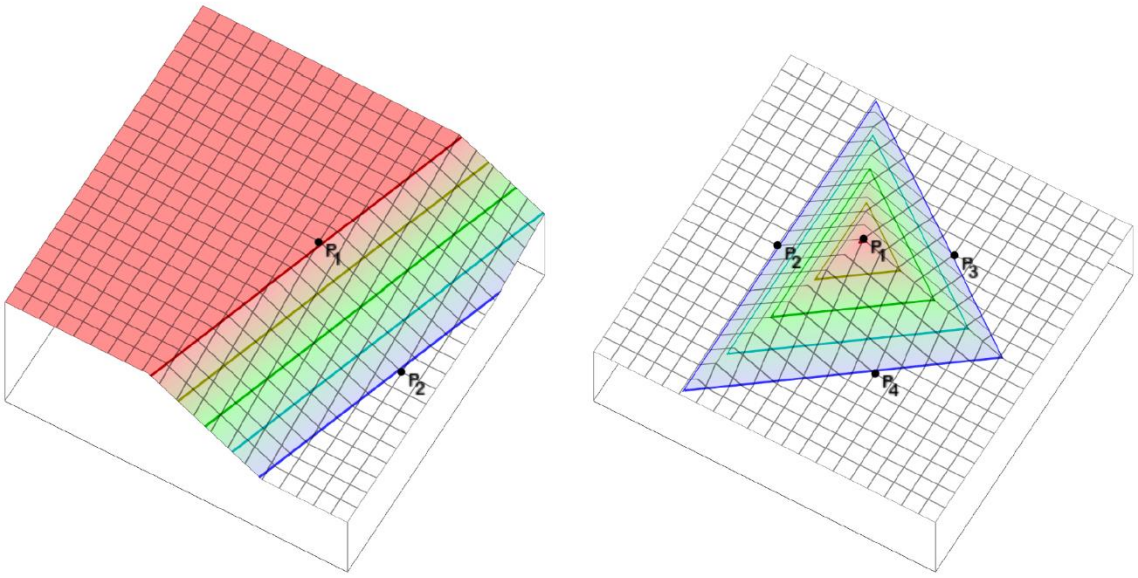
- **Monotonost** – Funkcija utjecaja trebala bi biti monotona tj. monotono se smanjivati od svog globalnog maksimuma do globalnog minimuma, bez lokalnih minimuma. U primjeru to bi značilo da bi se za npr. animaciju trčanja težina trebala smanjivati s jedan do nula, kako se brzina kretanja smanjuje brzine trčanja do brzine hodanja.
- **Neosjetljivost na gustoću** – Interpolacijska metoda treba raditi jednako dobro za jednoliko i nejednoliko raspoređene primjere točke/brzine u prostoru. Npr. to znači da ako imamo više primjera animacija za trčanje različitim brzinama njihov ukupni doprinos nekoj interpolacijskoj brzini/točki ne smije biti veći nego doprinos koji bi bio kada bi imali samo jednu primjer animaciju za trčanje.

Jedna od interpolacijskih metoda koja više-manje zadovoljava navedene kriterije je i metoda gradijentnih prijelaza (engl. *gradient band interpolation*) odnosno njezina inačica u polarnim koordinatama (engl. *polar gradient band interpolation*).

Korištenje ove interpolacijske metode se zasniva se na korištenju brzine kretnje animacije kretnje *i* kao interpolacijske točke $\mathbf{p}_i = (x_i, y_i, z_i)$. Ako novu (aktualnu) brzinu kretnje označimo kao $\mathbf{p} = (x, y, z)$ tada možemo izračunati funkcije utjecaja $h_i(\mathbf{p})$ svake interpolacijske točke na našu novu/traženu brzinu kao:

$$h_i(\vec{\mathbf{p}}) = \min_{j, j \neq i} \left(1 - \frac{\overrightarrow{\mathbf{p}_i\mathbf{p}} \cdot \overrightarrow{\mathbf{p}_i\mathbf{p}_j}}{\left|\overrightarrow{\mathbf{p}_i\mathbf{p}_j}\right|^2} \right) \tag{24}$$

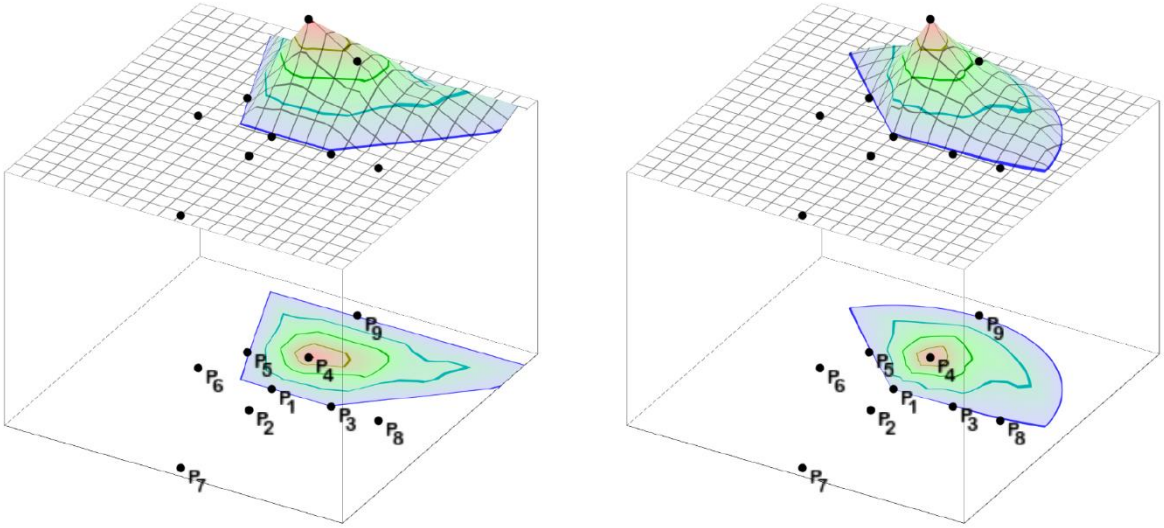
Interpretacija formule kaže da svaka funkcija utjecaja stvara gradijentne prijelaze koji povezuju točku $\mathbf{p}_i$ sa svakom drugom točkom $\mathbf{p}_j$ od kuda i dolazi naziv ove metode. Nakon što se izračunaju funkcije utjecaja lako se dolazi do težinskih funkcija preko formule (23). Sljedeća slika (Slika 8) prikazuje grafički prikaz funkcije utjecaja između dvije točke i tri interpolacijske točke.



Slika 8 – Prikaz funkcije utjecaja h_1 između dvije (prva slika) i tri točke (druga slika). (Johansen, 2009)

Također, na prethodnoj slici (Slika 8) se može vidjeti kako između dvije interpolacijske točke postoji jedan gradijentni nagib, između četiri točke tri nagiba, tj. da točka $\mathbf{p}_1$ stvara nagibe prema svakoj drugoj točki $\mathbf{p}_j$.

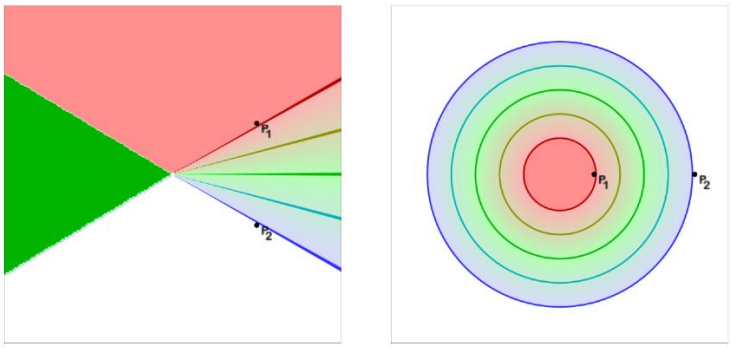
Navedena metoda gradijentnih prijelaza u kartezijevim koordinatama nije dovoljno dobra kada interpolacijske točke predstavljaju vektori brzine kretanja lika. To možemo vidjeti na sljedećoj slici (Slika 9), koja prikazuje grafički iznos funkcije utjecaja $h_4(\mathbf{p})$ za interpolacijsku točku $\mathbf{p}_4$, odnosno točku koja predstavlja brzinu kretanja animacije kretnje hodanja unaprijed.



Slika 9 – Prikaz funkcije utjecaja $h_4(\mathbf{p})$ za točku $\mathbf{p}_4$ u kartezijevim koordinatama (lijevo) i polarnim koordinatama (desno). (Johansen, 2009)

Kako svaka od točaka $\mathbf{p}_i$ predstavlja brzinu i smjer kretanja animacije kretnje možemo vidjeti da u kartezijevim koordinatama funkcija utjecaja $h_4(\mathbf{p})$ točke $\mathbf{p}_4$ ima utjecaja i za brzine koje su puno veće po iznosu nego njezin najbliži susjed točka $\mathbf{p}_9$, tj. animacije kretnje trčanja unaprijed. Također, vidimo da je taj problem otklonjen u polarnoj metodi interpolacije, koja bolje rezultate daje kada interpolacijske točke predstavljaju vektore brzine, a ne samo pozicije u kartezijevom prostoru. Polarna metoda zahtijeva sljedeće uvijete:

- Dvije interpolacijske točke $\mathbf{p}_i$ i $\mathbf{p}_j$ jednako udaljene od referentne interpolacijske točke $\mathbf{p}_r$ moraju imati klinasti oblik gradijentnog prijelaza između njih koje se širi iz referentne točke prema van (Slika 10 lijevo). Tada smjer neke nove točke relativan u odnosu na te dvije toče određuje iznos težina te dvije točke.
- Dvije interpolacijske točke $\mathbf{p}_i$ i $\mathbf{p}_j$ koje se nalaze na istome pravcu zajedno sa referentnom točkom $\mathbf{p}_r$ moraju imati gradijent u obliku kruga sa središtem u referentnoj točki (Slika 10 desno). Tada iznos neke nove točke relativan u odnosu na te dvije točke određuje iznos težina te dvije točke.



Slika 10 – Prikaz oblika gradijentnih prijelaza kod polarne interpolacijske metode. (Johansen, 2009)

Nadalje, potrebno je napraviti transformaciju vektora $\mathbf{p}_i\mathbf{p}_j$ i $\mathbf{p}_i\mathbf{p}$ iz kartezijevih u polarne koordinate:

$$\overrightarrow{\mathbf{p}_i\mathbf{p}_j} = \left(\frac{|\vec{\mathbf{p}}_j| - |\vec{\mathbf{p}}_i|}{\frac{|\vec{\mathbf{p}}_j| + |\vec{\mathbf{p}}_i|}{2}}, \angle(\overrightarrow{\mathbf{p}_j}, \overrightarrow{\mathbf{p}_i})\alpha \right) \tag{25}$$

$$\overrightarrow{\mathbf{p}_i\mathbf{p}} = \left(\frac{|\vec{\mathbf{p}}'| - |\vec{\mathbf{p}}_i|}{\frac{|\overrightarrow{\mathbf{p}_j}| + |\overrightarrow{\mathbf{p}_i}|}{2}}, \angle(\overrightarrow{\mathbf{p}'}, \overrightarrow{\mathbf{p}_i})\alpha \right) \tag{26}$$

gdje je **p'** projekcija vektora **p** na ravninu s normalom $\mathbf{p}_i \times \mathbf{p}_j$, a faktor α služi za definiranje važnosti smjera u odnosu na iznos brzine u interpolaciji. Algoritam metode gradijentnih prijelaza u polarnim koordinatama slijedi u nastavku (Kod 2).

```
Interpolate(samples[], output) {
  nSamples = Count(samples)
  weights[] = {0}
  i = Find(samples, output)
  if (i != -1) {
    weights[i] = 1.0f;
    return weights;
  }

  for (i = 0 to nSamples) {
    isOutsideHull = false;
    weight = 1.0f;
    for (j = 0 to nSamples) {
      if (i == j) continue;

        vSampleI = samples[i];
        vSampleJ = samples[j];
        angMulFactor = 2.0;
        angleJI = angleOI = 0;
        // Projection of output vector.
        vOutputProj = Vector()
        if (vSampleI = nullVect) {
          angleJI = Angle(vOutput, vSampleJ);
          angleOI = 0;
          vOutputProj = vOutput;
          angMulFactor = 1.0f;
        } else if (vSampleJ = nullVect) {
          angleJI = Angle(vOutput, vSampleI);
          angleOI = angleJI;
          vOutputProj = vOutput;
          angMulFactor = 1.0f;
        } else {
          angleJI = Angle(vSampleI, vSampleJ);
          if (angleJI == 0) {
            if (vOutput == nullVect) {
              angleOI = angleJI;
              vOutputProj = vOutput;
            } else {
              normal = CrossProduct(vSampleI, vSampleJ);
              vOutputProj = ProjectOnPlane(vOutput, Plane(normal, nullVect));
              angleOI = Angle(vSampleI, vOutput);
              if (angleJI < PI * 0.99) {
                if (DotProduct(CrossProduct(vSampleI, vOutputProj), normal) < 0) {
                  angleJI *= -1.0f;
                }
              }
            }
        } else {
          vOutputProj = vOutput;
          angleOI = 0;
        }
      }

      …
```

```
      …

      sampleIVectLen = Length(vSampleI);
      sampleJVectLen = Length(vSampleJ);
      outputVectLen = Length(vOutput);
      avgVectLen = (vSampleI + vSampleJ) / 2;
      sampleIVectLen /= avgVectLen;
      sampleJVectLen /= avgVectLen;
      outputVectLen /= avgVectLen;
      vIJ = Vector(angleJI * angMulFactor, sampleJVectLen - sampleIVectLen);
      vIO = Vector(angleOI * angMulFactor, outputVectLen - sampleIVectLen);
      newWeight = 1.0f - DotProduct(vIJ, vIO) / Length(vIJ);
      if (newWeight < 0) {
        isOutsideHull = true;
        break;
      }
      weight = Min(newWeight, weight);
    }
    if (isOutsideHull == false) {
      weight[i] = weight;
    }
  }
  Normalize(weights);
  return weights
}
```

Kod 2 - Prikaz pseudo-koda algoritma interpolacijsku metodu gradijentnih prijelaza

4.3 Grupiranje animacija

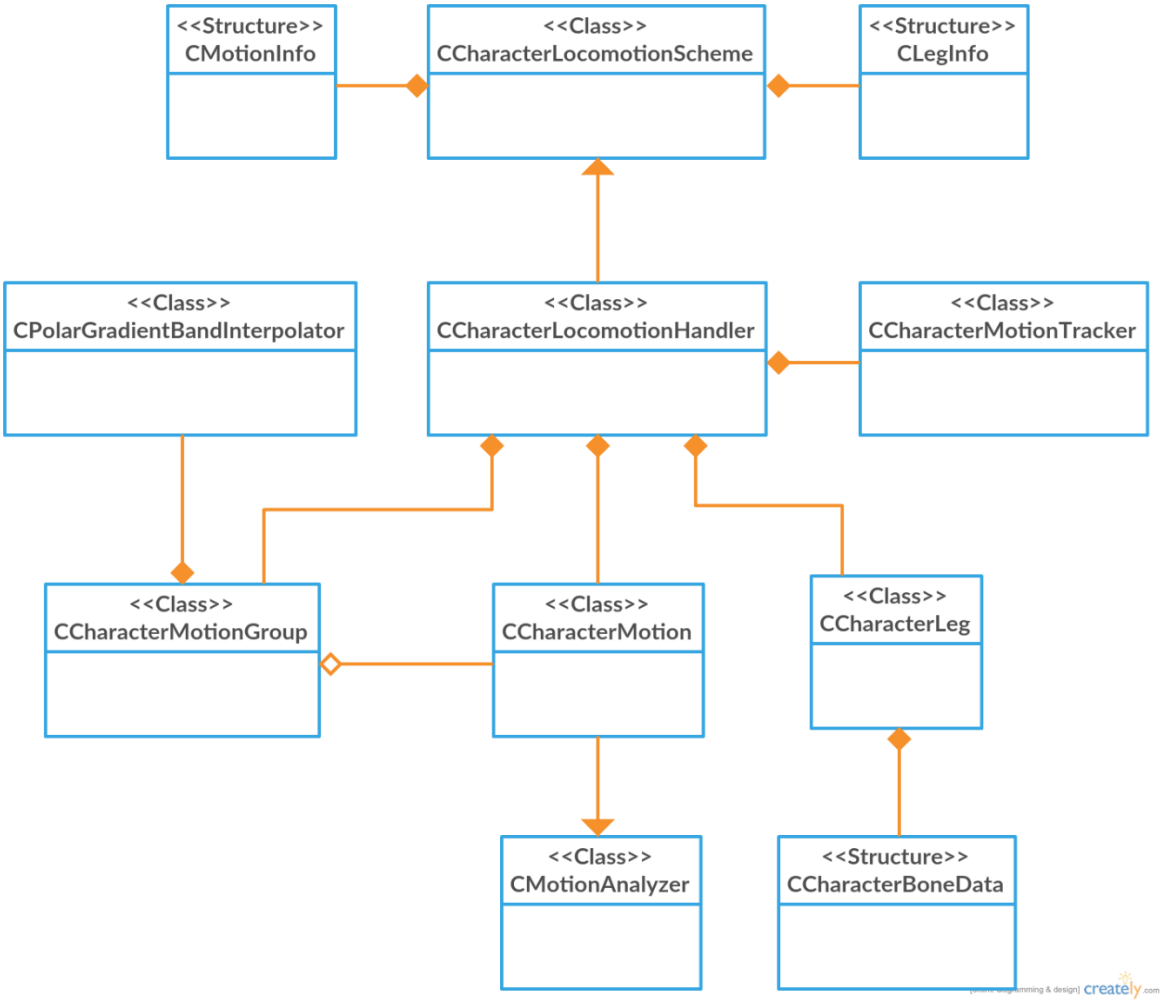
U uvodnim poglavljima je spomenuto da je jedan od razloga zašto je sustav polu-proceduralan taj što se na taj način animatorima ostavlja mogućnost stilističke prilagodbe kretnje lika. Grupiranje animacija to potiče na način da omogućuje dizajnerima grupiranje stilistički različitih animacija za hodanje/trčanje naprijed, nazad, lijevo, desno u animacijske grupe. Svaka grupa dobiva svoju težinu utjecaja na kretnju lika, a sustav za kombiniranje takvih grupa može biti neovisan o našem sustavu. To omogućuje da naš sustav koristi samo jedan parametar (vektor brzine) za interpolaciju između animacija, te da u jednom trenutku interpolira animacije koje se nalaze u grupama koje imaju težinu veću od 0. Na taj način naš sustav animacije ne mora imati kompleksan parametarski prostor što ga čini jednostavnijim i bržim.

5. Polu-proceduralna animacija

Ovo poglavlje dati će potpuni uvid u postupak polu-proceduralne animacije koju naš sustav implementira. Taj postupak koristiti će podatke dobivene iz analize korištenih primjera animacija opisanih u poglavlju 3 te postupaka kombiniranja i interpolacija takvih animacija opisanih u poglavlju 4. Osim toga biti će i opisani i navedeni proceduralni postupci animiranja, poput predikcija gdje se noga tijekom svakog trenutka treba nalaziti, gdje će noge gaziti podlogu te postupci prilagođavanja nogu i kukova lika kako bi zadovoljili nove pretpostavljene pozicije stopala. Proceduralni postupci animiranja i prilagođavanja opisani u nastavku izvedeni su tako da što manje izmjenjuju stilistički izgled originalnih primjera animacija.

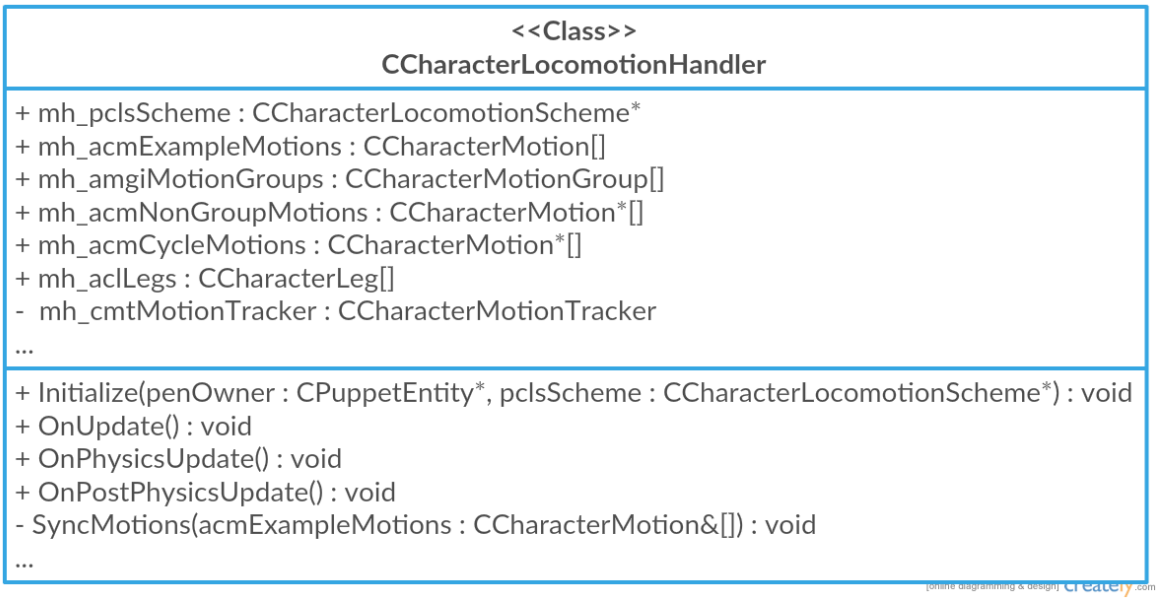
5.1 Lokomocijski sustav

U poglavlju 3.3 opisali smo dio programskog sustava koji obavlja automatsku analizu animacija zasnovanim na ključnim pozama i koje opisuju kretnje lika. Zatim smo u poglavlju 4 opisali postupke sinkronizacije i interpolacije analiziranih animacija. U nastavku ovog poglavlja opisat ćemo ostatak lokomocijskog sustava, nešto u obliku samog primjera implementacije, a nešto u konceptualnom obliku. Sljedeći dijagram razreda (Slika 11) prikazuje povezanost cijelog sustava, a objašnjenje slijedi u nastavku.



Slika 11 – Prikaz cjelokupnog lokomocijskog sustava

Razred `CCharacterLocomotionHandler` centralni je dio lokomocijskog sustava te se preko njega sustav inicijalizira i obavljaju sva proceduralna računanja. Sustav se inicijalizira na temelju podataka definiranih i zapisanih u razredu `CCharacterLocomotionScheme`, a čija je modifikacija pristupačna dizajnerima igre preko grafičkog sučelja tj. uređivača (engl. *editor*). Lokomocijski sustav se inicijalizira pozivom funkcije `Initialize` razreda `CCharacterLocomotionHandler` (Slika 12).



Slika 12 – Dijagram definicije razreda `CCharacterLocomotionHandler`

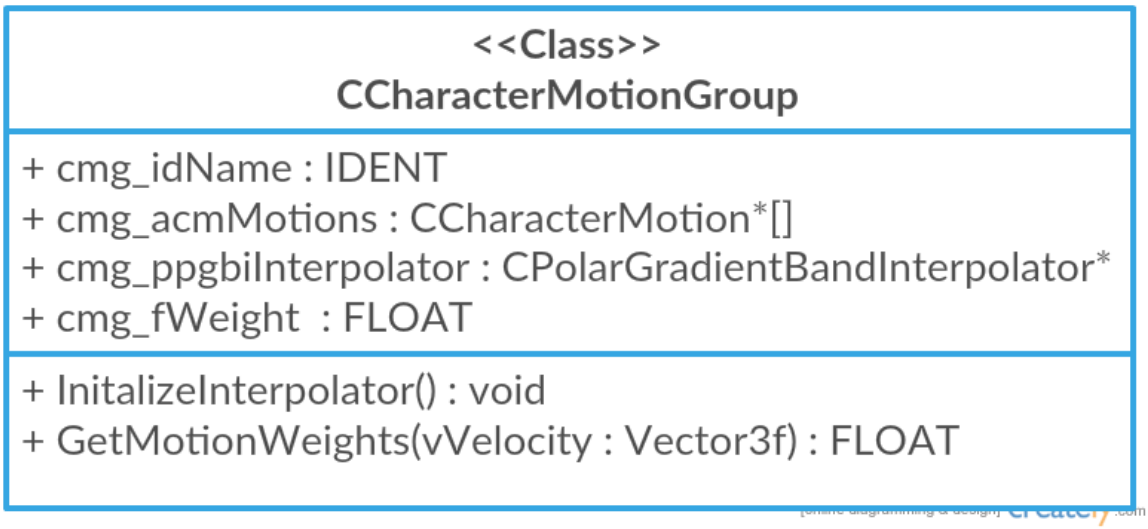
Inicijalizacijom sustava stvaraju se i inicijaliziraju animacije kretnje lika opisane razredom `CCharacterMotion` te se sve grupiraju u animacijske grupe opisane razredom `CCharacterMotionGroups`. Iz definicije razreda (Slika 12) vidimo da se iz svih animacija kretnje izdvajaju one koje predstavljaju kretnje (hodanje, trčanje - bez *idle* animacija) te one kretnje koje nemaju definiranu grupu. Za vrijeme inicijalizacije sustava još se prikupljaju podatci o nogama lika kojeg se animira, tj. podatci o lancima kostiju koje čine noge. Nakon postupka grupiranja i instalacije kretnji provodi se postupak sinkronizacije animacija preko funkcije `SyncMotions`, a čiji je algoritam prikazan i opisan u poglavlju 4.1. Pseudo-kod funkcije `Initialize` prikazan je u nastavku (Kod 3).

```
Initalize(lik, shema) {

  Inicijaliziraj podatke o duljini nogu lika, položaju kukova i kostima koje ćine
noge;

  Za svaku animaciju kretnje navedenu u shemi {

    Analiziraj animaciju kretnje;
    Spremi analiziranu kretnju u skup svih kretnji;
    Ako animacija ima navedenu animacijsku grupu u shemi {
      Dodaj kretnju u animacijsku grupu navedenu u schemi;
    }
    Inače {
      Dodaj kretnju u među skup negrupiranih animacija;
    }

    Ako je kretnja prikazuje aktivno kretanje {
     Dodaj kretnju u skup aktivnih cikličkih animacija;
    }
  }

  Za svaku animacijsku grupu {
    Inicijaliziraj interpolator;
  }

  Sinkroniziraj sve animacije iz skupa aktivnih animacija;
}
```

Kod 3 – Pseudo-kod funkcije `Initialize` razreda `CCharacterLocmotionHandler`

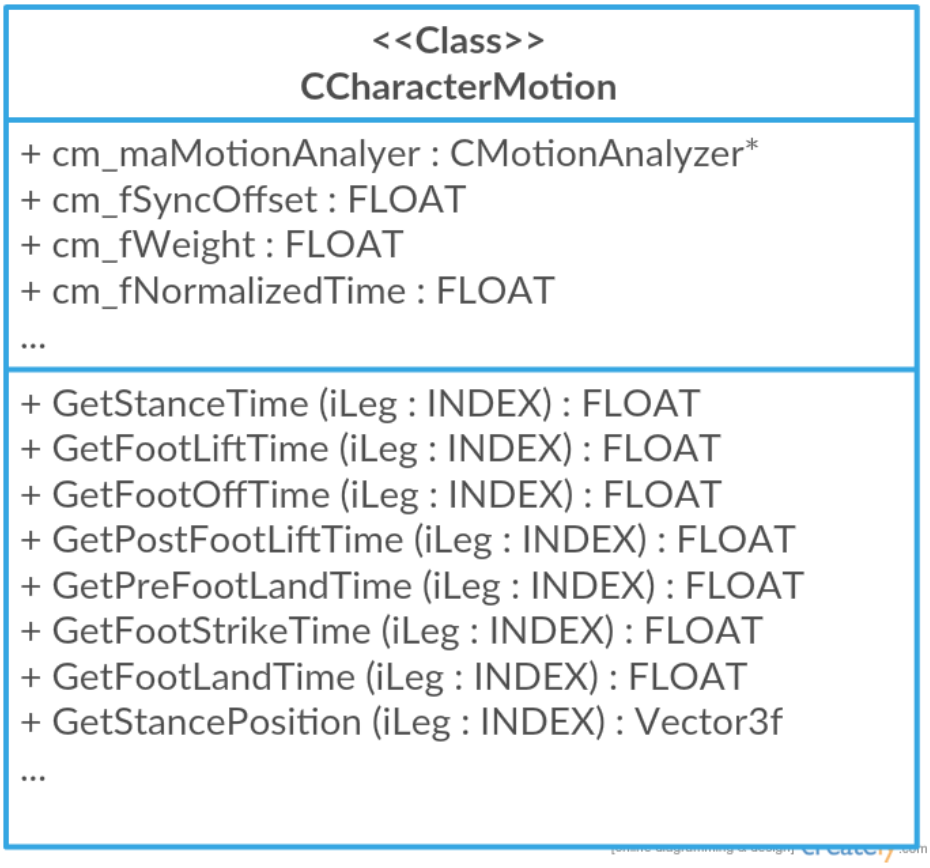
Za vrijeme izvođenja igre osvježavanje i procesiranje sustava (engl. *update*) obavlja se u tri dijela. Ta tri dijela su ostvarena unutar funkcija `OnUpdate`, `OnPhysicsUpdate`, `OnPosPhysicsUpdate` razreda `CCharacterLocomotionSystem`. Svaki od tih dijelova obavlja neke svoje radnje koje su podijeljene po vremenskom trenutku kada se moraju obaviti za vrijeme jednog perioda ažuriranja simulacije/igre (engl. *simulation/game cycle*). Po nazivima funkcija može se vidjeti da su ti trenutci definirani trenutcima ažuriranja fizike, pa tako dio sustava mora biti ažuriran prije, dio tijekom, a dio nakon ažuriranja fizike.

Krenimo redom od dijela koji se može ažurirati prije ažuriranja fizike. U tome dijelu se mogu ažurirati težine animacija kretnji na temelju brzine trenutne brzine lika. Prvo na temelju trenutne brzine lika i interpolatora pridijeljenog svakoj grupi izračunamo nove težine za animacije iz te grupe (Slika 13).

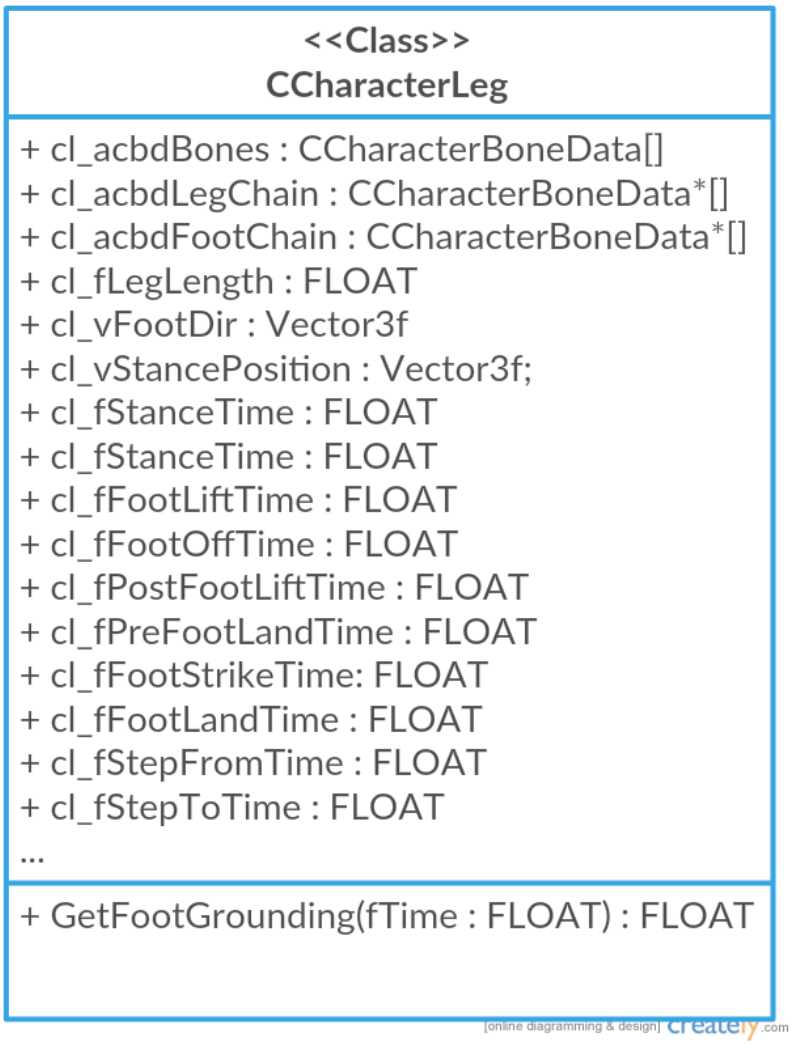


Slika 13 – Dijagram definicije razreda `CCharacterMotionGroup`

Dobivene težine su relativne u donosu na animacijsku grupu pa da bi dobili apsolutne težine u odnosu na sustav, dobivene težine moramo pomnožiti s težinom same grupe. Ne zaboravimo pritom da se težine samih grupa ažuriraju neovisno o našem sustavu i da se naš sustav ne zamara time. Nakon toga se dobivene težine postavljaju kao trenutne težine sa svaku animaciju iz grupe (Slika 14). Razlog zbog kojeg smo u razredu `CCharacterLocomotionHandler` izdvojili animacije koje prikazuju aktivno kretanje je taj što se samo one, tj. njihove težine, koriste za određivanje trenutnih iznos ključnih trenutaka za noge lika, definiranih u razredu `CCharacterLeg` (Slika 15), kao težinske sume ključnih trenutaka svake od animacija kretnji. Slični postupak, ali sa svima animacijama se koristi za utvrđivanje položaj točke *footbase* u trenutku *stance-time* i smjer stopala. Osim navedenog u `OnUpdate` metodi se još i određuje duljina trajanja jednog ciklusa kretanja. Ovome dijelu koda još želimo i reći sustavu da izvodi animacije čije su težine veće od 1, te da ih *blenda* u odnosu na njihove težine. Taj dio sustava već je implementiran u *engine*-u igre te ne zahtijeva pisanje dodatnog koda već samo poziv funkcija za izvođenje animacija. Cjelokupna ideja procedure `OnUpdate` dana je u sljedećem pseudo-kodu (Kod 4).



Slika 14 – Dijagram definicije razreda `CCharacterMotion`

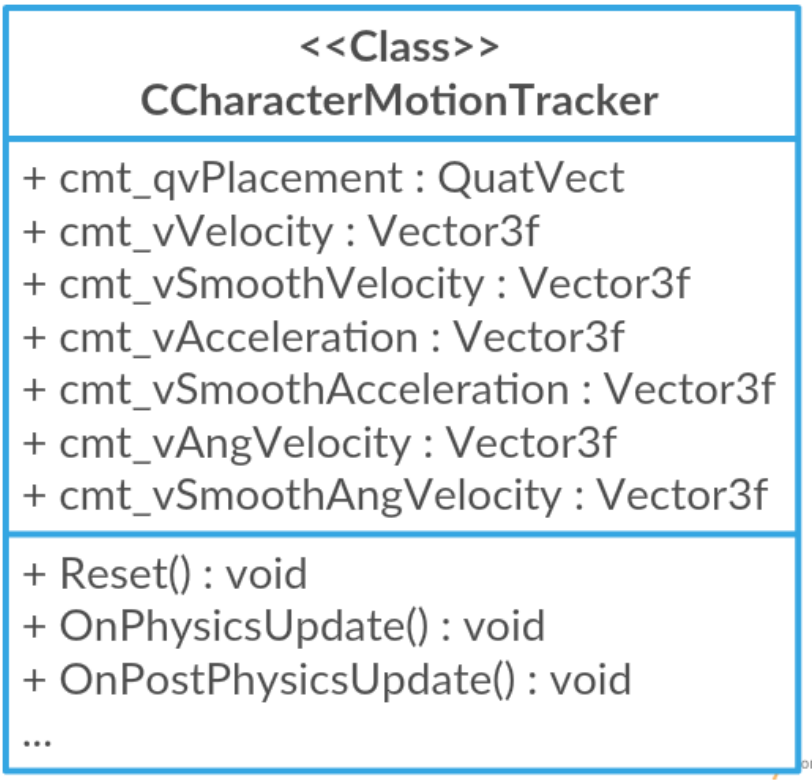


Slika 15 – Dijagram definicije razreda `CCharacterLeg`

```
OnUpdate() {
  Odredi trenutnu brzinu lika u prostoru lika;

  Za svaku animacijsku grupu {
    Ako se je brzina promijenila od zadnjeg ažuriranja {
      Izračunaj relativne težine animacija kretnji iz te grupe;
    }

    Ako se je težina grupe ili brzina lika promijenila od zadnjeg ažuriranja {
      Izračunaj nove apsolutne težine animacija iz te grupe tako da relativne težine
pomnožimo s težinom grupe;
    }
  }

  Ako su se težine animacija kretnji promijenile {
    Izračunaj normalizirane težine animacija kretnji posebno za sve animacije, a
posebno za animacije koje prikazuju aktivnu kretnju;

    Za svaku nogu lika {
      Izračunaj položaj "footbase" noge u trenutku stance-time te smjer i duljinu
noge kao težinsku sumu svih animacija kretnji, tj. njihovih težina.
      Izračunaj ključne trenutke noge kao težinsku sumu samo animacija koje prikazuju
aktivnu kretnju.
    }

    Izračunaj trenutnu procijenjenu duljinu jednog ciklusa kretnje lika.

    Ako sve noge nisu u mirujućem položaju, tj. lik se kreće {
      Sinkroniziraj svaku od animacija koje prikazuju aktivnu kretnju.
    }
  }
}
```

Kod 4 – Pseudo-kod procedure `OnUpdate` razreda `CCharacterLocomotionHandler`

Za potrebe sustava implementiran je posebni razred `CCharacterMotionTracker` koji služi za praćenje kretanja lika, odnosno praćenja njegove pozicije, vektora brzine, vektora kutne brzine i vektora akceleracije (Slika 16). Slično kao i razred `CCharacterLocomotionHandler` i ovaj razred ima posebno definirane dijelove koda koji bi se trebali izvršavati za vrijeme ažuriranja fizike (`OnPhysicUpdate`), tj. nakon ažuriranja fizike (`OnPostPhysicsUpdate`).



Slika 16 – Dijagram definicije razreda `CCharacterMotionTracker`

No vratimo se na dio sustava koji se izvršava za vrijeme ažuriranja fizike. Taj dio je dosta jednostavan jer sve što želimo da se računa za vrijeme ažuriranja fizike jest poziv metode `OnPhysicUpdate` razreda `CCharacterMotionTracker`.

Glavni dio proceduralnog dijela animiranja odvija se u proceduri `OnPostPhysicUpdate` razreda `CCharacterLocomotionSystem`. U njemu se na temelju svih dostupnih, prethodno izračunatih podataka procjenjuje gdje će stopalo noge u budućnosti nagaziti podlogu te se u skladu s time noga lika prilagođava podlozi kako ne bi kolidirala s neravnim dijelovima podloge. U sljedećem poglavlju detaljno ćemo objasniti i prikazati funkcionalnost procedure `OnPostPhysicUpdate`, tj. proceduralnog postupka prilagodbe noge podlozi.

5.2 Proceduralno animiranje

5.2.1 Određivanje poze stopala na podlozi

Proceduralni dio animiranja nogu zasniva se na pretpostavci da svaka noga ima otisak (engl. *footstep*) stopala odakle je krenula i otisak gdje će u budućnosti nagaziti. Trenutak kada se noga nalazi na podlozi određen je s vremenom stance-time, a između dva otiska noge mora proći vrijeme jednog ciklusa kretnje. Stoga se vrijeme sljedećeg otiska može računati kao:

$$stepToTime_l = time + cycleDuration \cdot (1 - currentCycleTime_l) \quad (27)$$

Zatim, ako znamo trenutnu poziciju, brzinu i kutnu brzinu lika možemo odrediti buduću poziciju i orijentaciju otiska noge:

$$\begin{aligned} \mathbf{stepToPosition}_l \\ = \mathbf{characterPosition} + (\mathbf{characterVelocity} \\ * (stepToTime_l - time)) \end{aligned} \quad (28)$$

$$\begin{aligned} \mathbf{stepToRotation}_l \\ = \mathbf{characterRotation} \\ \cdot (\mathbf{characterAngularVelocity} \\ \cdot (stepToTime_l - time)) \end{aligned} \quad (29)$$

U samom kodu implementacije postoje još dodatna računanja gdje se za predviđenu poziciju otiska još u obzir uzima i kutna brzina kretanja lika, odnosno ukoliko se lik kreće nepravocrtno odnosno polukružno, a što znamo iz praćenja njegove kutne brzine, umjesto linearne predikcije koristi se kružna predikcija:

$$\begin{aligned} \mathbf{stepToPosition}_l \\ = \mathbf{characterPosition} + \mathbf{turnCenter} \\ + \mathbf{headingRotation} \cdot (-\mathbf{turnCenter}) \end{aligned} \quad (30)$$

$$\mathbf{turnCenter} = \frac{\widehat{\mathbf{up}} \times \mathbf{characterVelocity}}{turnSpeed} \quad (31)$$

$$turnSpeed = \mathbf{characterVelocity} \cdot \widehat{\mathbf{up}} \tag{32}$$

$$\mathbf{headingRotation} = AxisAngle(\widehat{\mathbf{up}}, \angle(turnSpeed * (stepToTime_i - time)) \tag{33}$$

Dobivene predikcije se odnose na savršeno ravne i pravilne podloge, stoga se otisak mora prilagoditi visinski i nagibom podlozi gdje se nalazi. Taj postupak se obavlja tako da se iz trenutno predviđene horizontalne pozicije i visine koja iznosi maksimalnu dozvoljenu visinu stopala ispuca zraka (engl. *raycast*) prema podlozi. Rezultat toga je vektor normale i pozicije gdje se sijeku zraka i podloga. Na temelju normale se može odrediti rotacija otiska stopala, dok pozicija presjeka zrake i podloge postaje predviđena pozicija otisaka. Implementirani sustav radi preciznosti koristi dvije zrake za provjeru, jednu ispucanu u iz prstiju, a drugu iz pete stopala. Također, postupak u obzir uzima i mogućnost da podloga može biti stepenasta, pa prema tome prilagođava rezultat tako da stopalo uvijek što punijim obujmom stoji na stepenici, a što manje na rubu.

5.2.2 Određivanje putanje kretanja noge

Nakon što imamo potpunu informaciju o sljedećem otisku gdje se stopalo mora nalaziti je odrediti putanju kojom će se noga kretati do tog otiska, a da pritom izbjegne prepreke na putu. U poglavlju 3.1 opisan je koncept pojma *footbase* i normalizirane putanje točke *footbase* za vrijeme jednog ciklusa kretnje. Ta će nam putanja pomoći da odredimo putanju kojom će se noga kretati do sljedećeg otiska i tako ćemo zadržati stilističku informaciju o kretnji. Sam problem određivanja i konstrukcije putanje krivulje kretnje noge može se podijeliti na manje podprobleme:

1. Na temelju normalizirane putanje kretnje točke *footbase* odrediti dinamiku kretnje noge (po z osi).
2. Odrediti horizontalne pomake u odnosu na ravnu normaliziranu putanju točke *footbase*, kako bi se pratila nepravocrtna putanja kretnje lika.
3. Funkcijski odrediti promjenu visine točke *footbase* kako bi se omogućilo izbjegavanje prepreka na neravnom terenu.
4. Podići visinu točke *footbase* noge koja leti do visine stajaće noge, ako je stajaća noga na većoj visini nego leteća.

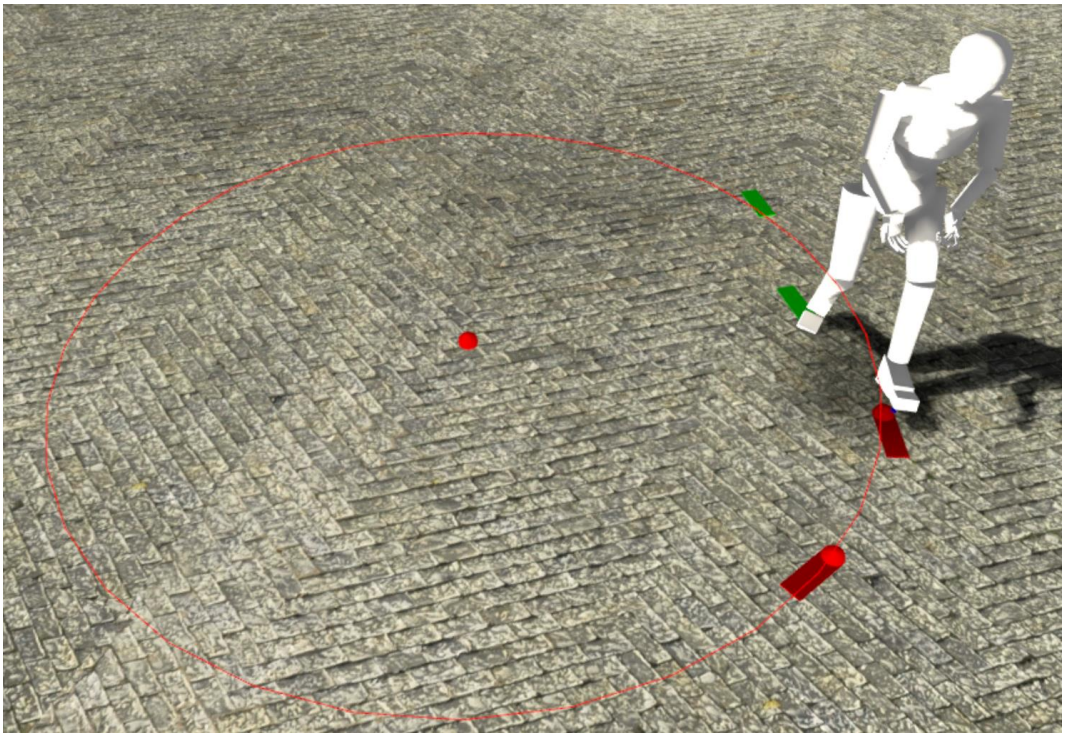
5. Dodavanje vertikalne (y) i bočne (x) komponente normalizirane putanje kretnje točke *footbase* kako bi se ostalo što stilističke vjernije originalnoj animaciji.

Da bi riješili prvi podproblem trebamo se prisjetiti da smo tijekom analize animacija kretnji za svaku animaciju odredili normaliziranu putanju po z osi točke *footbase*. Sada na temelju z komponente te normalizirane putanje te pozicije prijašnjeg i sljedećeg otiska možemo izračunati trenutnu točku *footbase* za vrijeme leta noge:

$$\mathbf{footbasePosition}_l = Lerp(\mathbf{stepFromPosition}_l, \quad \mathbf{stepToPosition}_l, \\ normalizedFootbaseTrajectory_l(flightTime_l).z) \tag{34}$$

$$flightTime_l \\ = Lerp^{-1}(footOffTime_l, footStrikeTime_l, cycleTime_l) \tag{35}$$

Drugi pod problem javlja se kada se lik kreće nepravocrtno odnosno polukružno se okreće. Tada umjesto da noge prate i putuju jedna pokraj druge za vrijeme jednog ciklusa one se međusobno isprepliću. Kako bi izbjegli taj problem i omogućili da noge i za vrijeme polukružnog hodanja ostanu jedna pokraj druge potrebno je odrediti među točku/otisak koja se nalazi između prethodnog i sljedećeg otiska te zahtijevati da noga mora proći preko te točke (Slika 17).



Slika 17 – Prikaz kretnje lika sa primjenom dodatnog horizontalnog pomaka na putanju kretnje noge.

Kako bi horizontalna krivulja kretnje noge između prethodnog otiska i među-otiska te među otiska i sljedećeg otisak bila glatka potrebno je odrediti funkciju ovisnu o parametru z normalizirane trajektorije točke *footbase*, a čiji iznos će biti težina od 0 do 1 koja će određivati jačinu pomaka putanje. Sinusna funkcija zadovoljava taj kriteriji:

$$flightProgressionArc_l = \sin(normalizedFootbaseTrajectory.z \cdot \pi) \quad (36)$$

Vektor pomaka se tada računa na sljedeći način:

$$\begin{aligned} &\mathbf{offset} \\ &= \mathbf{characterPosition} + \mathbf{characterRotation} \cdot \mathbf{stancePosition}_l \\ &- Lerp(\mathbf{stepFromPosition}_l, \mathbf{stepToPosition}_l, cycleTime_l) \end{aligned} \quad (37)$$

Te se taj pomak nadodaje na trenutni iznos točke *footbase*:

$$\mathbf{footbasePosition}_l \mathrel{+}= \mathbf{offset} \cdot flightProgressionArc_l \quad (38)$$

Treći pod problem, vezan za utvrđivanje vertikalnog pomaka noge za vrijeme leta kako bi se izbjegle eventualne neravnine na podu, rješava se na sličan način kao i prethodni pod problem. Također je potrebno naći među točku/otisak između prethodnog otiska i sljedećeg otiska, no potrebno je pronaći i sjecište ravnine u kojoj se nalazi prethodni/sljedeći otisak i pravca koji kreće iz smjera među točke u smjeru globalne koordinate +y. To sjecište se nalazi u određenoj visini u odnosu na među točku, a koristi se za utvrđivanje amplitude sinusoide po koja predstavlja funkcijsku promjenu visine stopala za vrijeme leta noge (Slika 18):

$$\mathbf{stepMidPosition}_l = \frac{\mathbf{stepFromPosition}_l + \mathbf{stepToPosition}_l}{2} \tag{39}$$

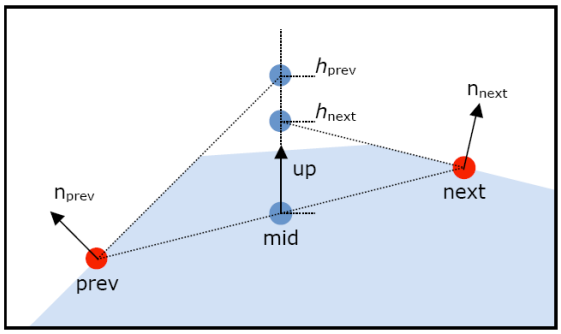
$$height_{from} = \frac{(\mathbf{stepFromPosition}_l - \mathbf{stepMidPosition}_l) \cdot \widehat{\mathbf{normal}}_{\text{from}}}{\widehat{\mathbf{normal}}_{\text{from}} \cdot \widehat{\mathbf{up}}} \tag{40}$$

$$height_{next} = \frac{(\mathbf{stepFromPosition}_l - \mathbf{stepMidPosition}_l) \cdot \widehat{\mathbf{normal}}_{\text{next}}}{\widehat{\mathbf{normal}}_{\text{next}} \cdot \widehat{\mathbf{up}}} \tag{41}$$

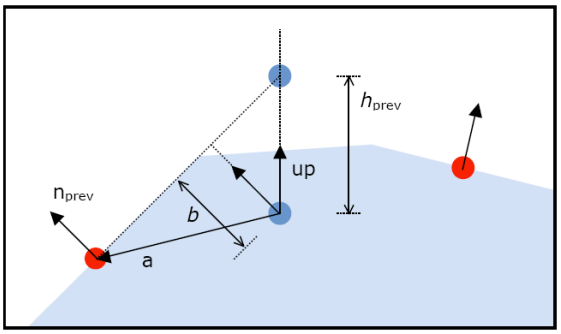
$$height_{max} = \max(height_{from}, height_{next}) \tag{42}$$

$$heightOffset_l = \frac{2 \cdot height_{max}}{\pi} \tag{43}$$

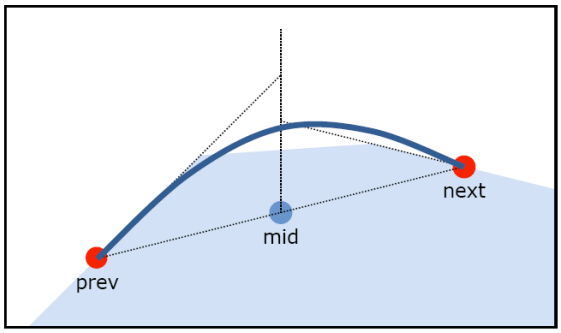
$$\mathbf{footbasePosition}_l \mathrel{+}= \mathbf{up} \cdot heightOffset_l \cdot flightProgressionArc_l \tag{44}$$



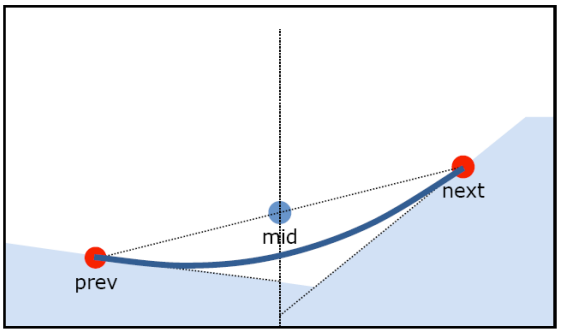
(a) *Intersection of tangents with vertical axis extended from mid-point.*



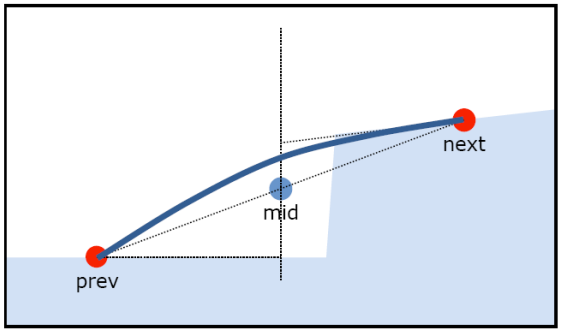
(b) *Calculation of height of tangent intersection above mid-point.*



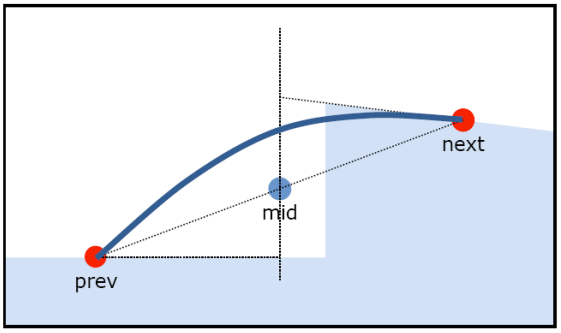
(c) *The sinus curve follows the highest tangent. Its magnitude is based on the height of the higher intersection.*



(d) *Example with valley.*



(e) *Example with a step.*



(f) *Another example with a step; this one requiring higher lifting.*

Slika 18 – Prikaz određivanja vertikalne krivulje kretnje noge kada se noga koreće preko nepravilne podloge. (Johansen, 2009)

Četvrti pod problem vezan je uz situaciju kada je noga koja leti u nižem položaju od noge na koju se lik oslanja. U toj situaciji potrebno je nogu podići na visinu koja neće biti manja od visine nogu na koju je lik oslonjen. Zbog toga je potrebno odrediti visinu podloge iznad koje nalaze stopala svih ostalih nogu, odrediti težine za svaku

od tih visina na temelju faze u kojoj se pripadajuća noga nalazi te težinskom sumom odrediti točnu visinu na koju se leteća noga mora podići. Visina na kojoj se određena noga nalazi određuje se na temelju smjera kretnje lika te prethodnog i sljedećeg otiska te noge:

$$\mathbf{characterDirection} = (\mathbf{characterVelocity} \perp \mathbf{up}) \tag{45}$$

$$\mathbf{stepFromTangent}_l = \boldsymbol{stepFromRotation}_l * \mathbf{characterDirection} * cycleDistance \tag{46}$$

$$\mathbf{stepToTangent}_l = \boldsymbol{stepToRotation}_l * \mathbf{characterDirection} * cycleDistance \tag{47}$$

$$weight_l = -\frac{\cos(\pi * strideTime_l)}{2} + 0.5 \tag{48}$$

$$\mathbf{groundHeightPoint}_l = Lerp(\mathbf{stepFromPosition}_l + \mathbf{stepFromTangent}_l * cycleTime_l, \mathbf{stepToPosition}_l + \mathbf{stepToTangent}_l * (cycleTime_l - 1), weight_l) \tag{49}$$

$$\mathbf{groundHeightPoint}_l \mathrel{+}= \mathbf{characterRotation} * (-\mathbf{stancePosition}_l) \tag{50}$$

Važno je napomenuti da je smjer kretanja lika određen u prostoru lika. Nakon što imamo visine podloge za svaku od nogu potrebno je odrediti utjecaj svake od tih visina podloga na konačnu visinu za koju ćemo stopalo podići:

$$h_l = \frac{\cos(2\pi * cycleTime_l) + 1}{2} + \varepsilon \tag{51}$$

$$\mathbf{supportedGroundHeight}_l = \frac{\sum_{i=1}^{legs} h_i * \mathbf{groundHeightPoint}_i}{\sum_{i=1}^{legs} h_i} \tag{52}$$

$$flightTimeArc = \sin(flightTime_l * \pi) \tag{53}$$

$$liftedHeight = Lerp(\mathbf{footbasePosition}_l \cdot \mathbf{up}, \mathbf{supportedGroundHeight} \cdot \mathbf{up} \quad (54)$$

Ako je dobivena visina veća od trenutne visine izračunate visine stopala odnosno visine točke *footbase* tada se stopalo podiže na tu novu točku.

Na kraju moramo na kretnju noge moramo još nadodati bočna i vertikalne pomake kretanja, a koji su definirani u normaliziranoj putanji točke *footbase*:

$$\mathbf{stepDirection}_l = \boldsymbol{stepToPosition}_l - \mathbf{stepFromPosition}_l \quad (55)$$

$$\mathbf{stepSidewaysDirection}_l = \mathbf{up} \times \mathbf{stepDirection}_l \quad (56)$$

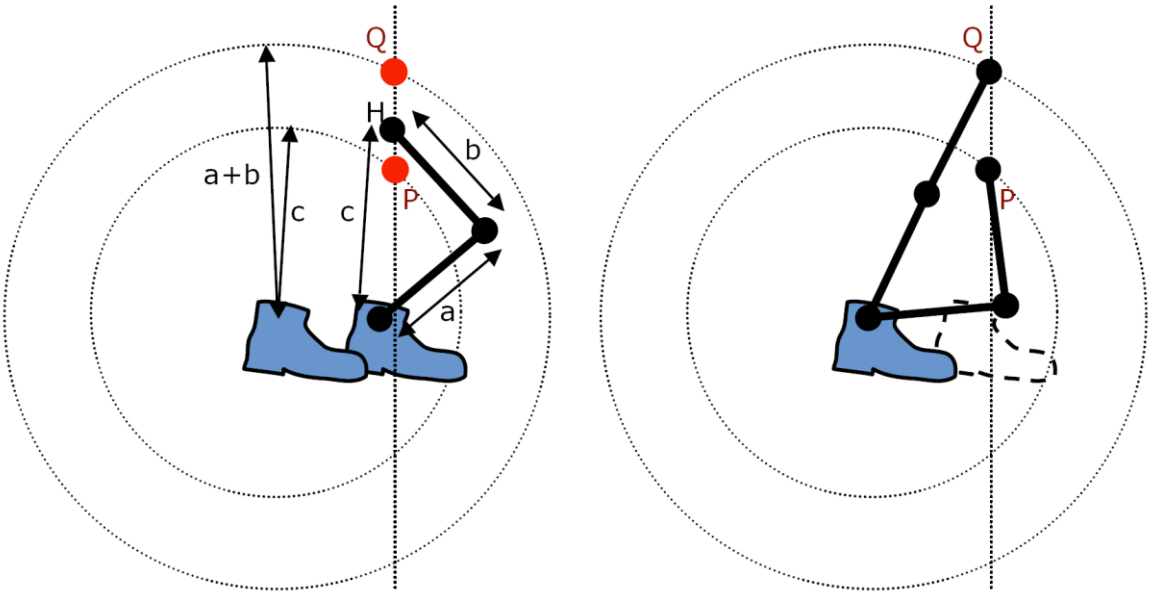
$$\mathbf{footbasePosition}_l + = \mathbf{stepSidewaysDirection}_l * normalizedFootbaseTrajectory_l(flightTime_l).x \quad (57)$$

$$\mathbf{footbasePosition}_l + = \mathbf{up} * normalizedFootbaseTrajectory_l(flightTime_l).y \quad (58)$$

5.2.3 Prilagođavanje kostiju noge

Nakon što imamo poznatu pozu stopala potrebno je ostatak noge, tj. ostale kosti koje čine nogu, također prilagoditi kako bi cijela animacija bila što sličnija primjerima animacija kretnji. U tu svrhu potrebno je podesiti i odrediti poze zglobova kuka i gležnja. Kako bi se održala sličnost s originalnim animacijama potrebno je za vrijeme proceduralnog animiranja što je moguće više zadržati udaljenost između gležnja i kuka koliko god je to moguće. U nekim slučajevima zbog nepravilnog terena taj originalni relativni odnos kuka i gležnja nije moguće održavati pa se stoga određuju dodatne varijable koje određuju željenu (*desiredHipHeight*) i maksimalnu visinu kuka (*maxHipHeight*) za svaku nogu. Željena visina kuka će kontrolirati koliko je noga savinuta, dok je za maksimalni iznos visine noga potpuno izravnata. Te se varijable određuju na način da se nađu presijeci pravca usmjerenog iz točke kuka u originalnoj kretnji prema podlozi sa sferama od kojih je jedna radijusa udaljenosti kuka i gležnja u originalnoj kretnji, a druga radijusa koji je jednak duljini noge općenito, obje s centrom u novom položaju gležnja. Presjek s bližom sferom

predstavlja točku *desiredHipHeight*, a s udaljenijom sferom točku *maxHipHeight* (Slika 19).



Slika 19 – Prikaz određivanja točaka desiredHipHeight (P) i maxHipHeight (Q). (Johansen, 2009)

Postupak je potrebno ponoviti za sve noge, zatim se računaju minimalni, maksimalni i prosječni pomak kuka:

$$maxHipHeight_{min} = \min_{l}(maxHipHeight_{l}) \tag{59}$$

$$desiredHipHeight_{min} = \min_{l}(desiredHipHeight_{l}) \tag{60}$$

$$desiredHipHeighy_{avg} = \frac{\sum_{l=1}^{legs} desiredHipHeight_{l}}{legs} \tag{61}$$

Zatim se može izračunati konačni pomak kuka kao:

$$hipOffset = desiredHipHeight_{min} + \frac{minToAvg * minToMax}{minToAvg + minToMax} \tag{62}$$

$$minToAvg = desiredHipHeighy_{avg} - desiredHipHeight_{min} \tag{63}$$

$$minToMax = maxHipHeight_{min} - desiredHipHeight_{min} \tag{64}$$

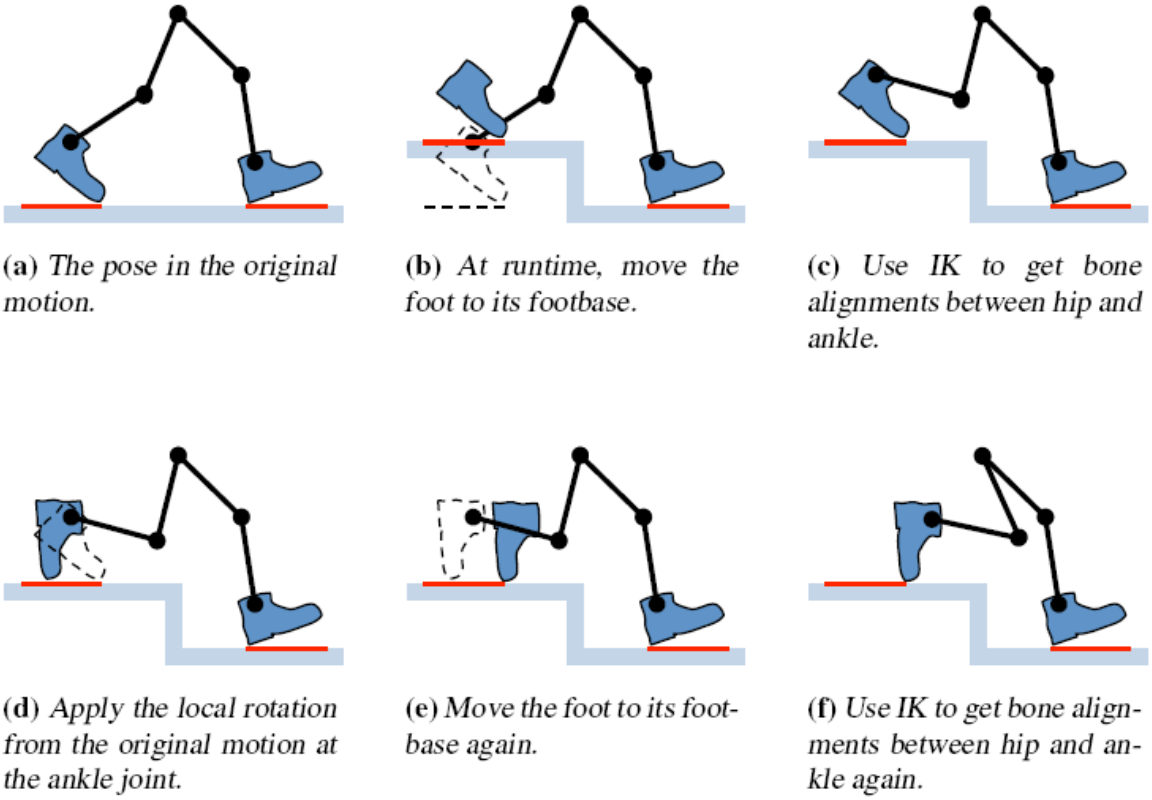
Prije primjene inverzne kinematike za pozicioniranje kostiju nogu lika, još moramo rekonstruirati željenu poziciju i rotaciju gležnja noge na temelju dobivene pozicije točke *footbase*. Ako se prisjetimo kako smo pri analizi animacije dobili željene pozicije točke *footbase* na temelju pozicije palca i pete, sada inverznim postupkom možemo doći do željene pozicije točke gležnja na temelju nove vrijednosti točke footbase. Prvo moramo odrediti pozicije pete i palca u pozi koja je ostvarena samo *blendanjem* animacija (poze dobivene prije proceduralnog računanja). Uz pomoć formule 10 za računanje balansa stopala (poglavlje 3.1) izračuna se novi balans. Zatim možemo izračunati novi pomak pete od gležnja kao:

$$\begin{aligned}\mathbf{newHeelOffset}_l &= newBalance_l \\ &* [(\mathbf{footbaseRotation}_l * \mathbf{footdirection}_l) \\ &+ (\mathbf{heelPositionWorld}_l - \mathbf{toePositionWorld}_l)]\end{aligned} \tag{65}$$

Uz pomoć tog novog pomaka pete, nove pozicije točke *footbase* te pozicije pete možemo izračunati novu željenu poziciju gležnja kao:

$$\begin{aligned}\mathbf{anklePosition}_l &= \mathbf{footbasePosition}_l + \mathbf{newHeelOffset}_l \\ &- \mathbf{heelPositionWorld}_l\end{aligned} \tag{66}$$

Nakon što imamo sve potrebne informacije (poziciju i rotaciju točke *footbase*, poziciju gležnja te poziciju kuka) možemo podesiti cijelu nogu da odgovara žaljenoj pozi. Sljedeća slika (Slika 20) prikazuje korake prilagođavanje noge podlozi.



(a) *The pose in the original motion.*

(b) *At runtime, move the foot to its footbase.*

(c) *Use IK to get bone alignments between hip and ankle.*

(d) *Apply the local rotation from the original motion at the ankle joint.*

(e) *Move the foot to its footbase again.*

(f) *Use IK to get bone alignments between hip and ankle again.*

Slika 20 – Prikaz koraka podešavanja noge na temelju svih izračunatih informacija. (Johansen, 2009)

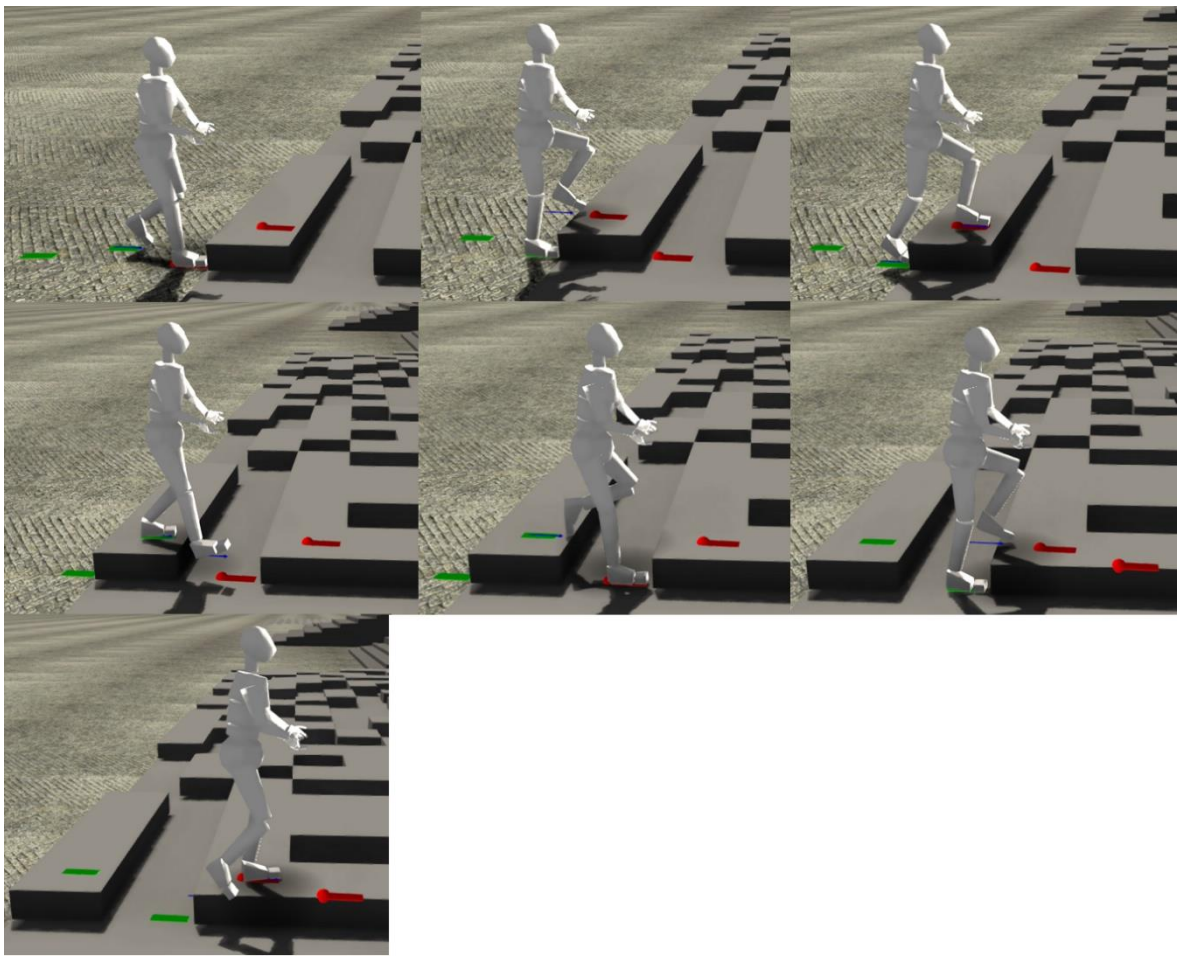
Postupak prilagođavanja započinje s pozom noge dobivene samo *blendanjem* originalnim primjer animacija (a). Na temelju izračunate pozicije i rotacije točke *footbase* te izračunate računa se željena pozicija gležnja (b). Uz pomoć inverzne kinematike na temelju nove pozicije gležnja te pozicije kuka podesimo lanac kostiju/zglobova između kuka i gležnja (c). Zatim ovisno o fazi leta noge nadodamo rotaciju točke *footbase,* također dobivenu u prethodnim kalkulacijama, na gležanj (d). Opet pomaknemo gležanj prema točki *footbase* (e). I na kraju opet uz pomoć inverzne kinematike podesimo lance kostiju/zglobova između gležnja i kuka. Nakon toga noga se nalazi u željenoj poziciji te je postupak završen za taj ciklus osvježavanja igre. Na sljedećem ciklus osvježavanja cijeli postupak se ponavlja ispočetka s nekim drugačijim početnim pozama i uvjetima iz okoliša igre.

Iako je sustav za inverznu kinematiku već prethodno bio implementiran u sustav Serious Engine u dodatnom poglavlju (**Error! Reference source not found.**) biti će kratko opisan sustav animiranja preko kinematičkih lanaca.

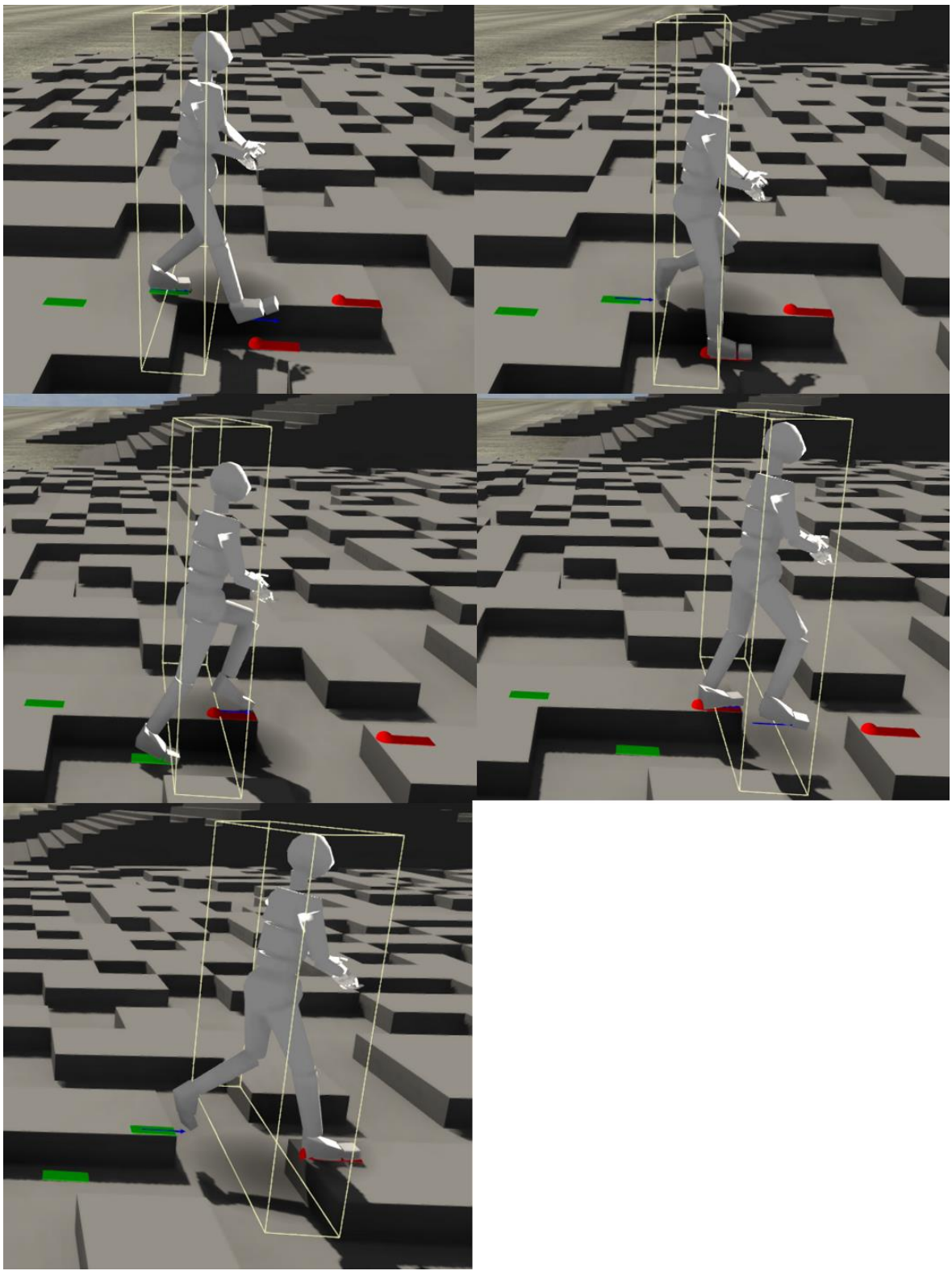
6. Rezultati

Izrađeni sustav testiran je na humanoidnom liku koji ima predefinirane animacije kretnje za kretanje naprijed, nazad, lijevo, desno u trku i hodajući. Kretnja lika je testirana na stepeničastim, kosim i nasumično nepravilnim podlogama. Sustav više manje zadovoljava sve pretpostavljene uvijete, no još ostavlja mogućnosti poboljšanja, kao na primjer bolje predviđanje podloge gdje će stopalo nagaziti.

Na sljedeće dvije slike (Slika 21 i Slika 22) može se vidjeti kretnja lika po stepeničastoj podlozi.



Slika 21 – Prikaz kretnje lika po stepeničastoj podlozi uz pravilnu predikciju gdje noga treba nagaziti.



Slika 22 – Prikaz kretnje lika po stepeničastoj podlozi kada je noga na koju se lik oslanja na različitoj visinskoj razini u odnosu na nogu koja je u letu.

Na drugoj slici (Slika 22) možemo vidjeti kako se korijen kostura i kukovi pravilno pomiču ovisno o grbavosti podloge.

Također, unutar sustava izrađen je i sustav za praćenje i *debugiranje* različitih aspekata sustava kretnji (prikaz predviđenih koraka, prikaz iznosa ključnih trenutka svake animacije, prikaz težina svake od animacija itd.)

Nadalje, za sustav kretanja izmjereni su i utjecaji na duljinu jednog *frame*-a. Tako na konfiguraciji koja uključuje procesor Intel Core i7-4790K 4.2GHz produljenje jednog jednoga *frame*-a u odnosu na izvođenje simulacije bez našeg sustava je:

1 lik (2 noge) => ~0.0 ms

10 likova (20 nogu) => < ~0.1 ms

20 likova (40 nogu) => ~0.2 ms

Možemo vidjeti da su promjene dosta male, tj. da sustav za kretnje nema prevelik utjecaj na performanse. Skalabilnost sustava ovisna je o broju nogu kojima mora upravljati, što znači da ako imamo duplo manje likova koji imaju duplo više nogu sustav će biti jednako brz.

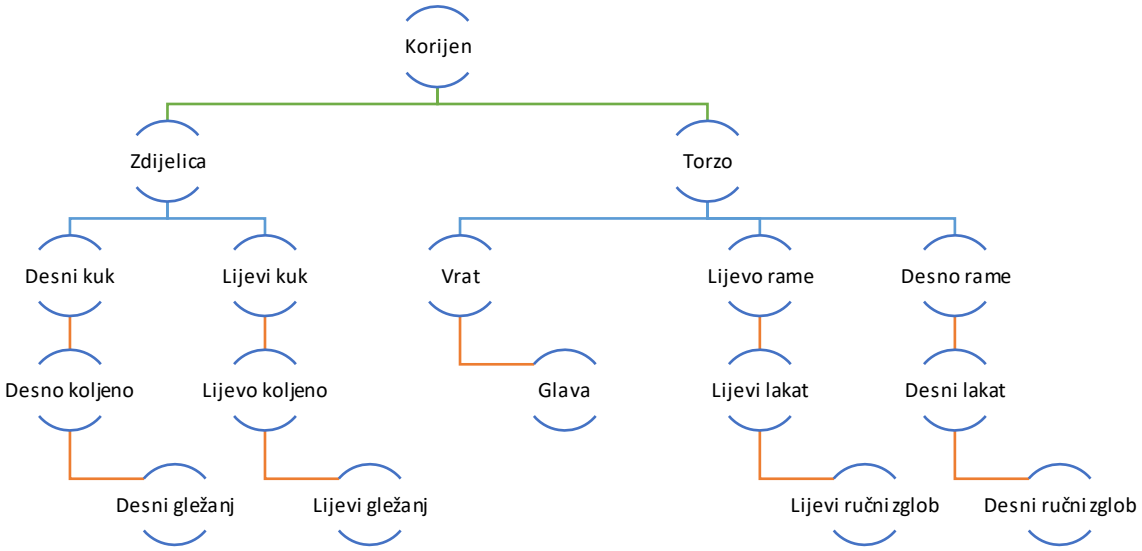
7. Kinematika (Dodatak)

Kinematika je grana mehanike i matematike kojom se opisuje i gibanje nekog objekta ne uzimajući u obzir utjecaj sila i mase na njegovo gibanje. Gibanje objekta se opisuje kinematičkim jednadžbama čije varijable obično opisuju translacijsko i rotacijsko gibanje (kut, kutna brzina, pozicija, translacijska brzina itd.), a sam objekt mora biti na neki način strukturiran prije početka opisivanja gibanja.

7.1 Hijerarhijska struktura

Zbog potrebe za što prirodnijim načinom ostvarivanja animacija, za svaki objekt koji se animira potrebno je definirati kostur (engl. *skeleton*) kao hijerarhijsku strukturu. Kostur se sastoji od segmenata/kostiju (engl. *bones*) i zglobova (engl. *joints*). Segment je definiran svojom duljinom (udaljenost između dva susjedna zgloba), a kut između dva susjedna segmenta definiran je u zglobu. No, u postupku animacije segment/kost nema niti jednu drugu ulogu osim definiranja udaljenosti između dva krajnja zgloba pa se stoga segment i zglob mogu spojiti tako da se taj pomak između dva susjedna zgloba pohrani u hijerarhijski niži zglob. Sada, nakon

navedene transformacije svaki kostur možemo prikazati kao stablo gdje svaki zglob čini jedan čvor stabla (Slika 23).



Slika 23 - Stablasti prikaz kostura

Svaki zglob može imati do 6 stupnjeva slobode (engl. *degree of freedom(DOF)*) u 3-dimenzionalnom prostoru, odnosno do 3 stupnja slobode u 2-dimenzionalnom prostoru, i to na sljedeći način: jedan translacijski stupanj slobode po svakoj od dimenzija (engl. *Prismatic Joint*) te po jedan rotacijski stupanj slobode za svaku ravninu koordinatnog sustava (engl. *Revolute Joint*). Dimenzije zglobova su zanemarive, a složeniji zglobovi se mogu ostvariti povezivanjem više 1DOF zglobova razmaknutih za udaljenost 0. Osim stupnjeva slobode svaki zglob mora imati još definiran i odmak od svojeg zgloba roditelja, ograničenja u zglobovima (minimalne i maksimalne dozvoljene vrijednosti po svakom stupnju slobode). Nakon što smo tako definirali kostur preostaje nam omogućiti neki način specificiranja vrijednosti u zglobovima kako bi odredili pojedine ključne trenutke animacije.

7.2 Izravna kinematika

Nakon što imamo definiran ključni trenutak kostura, tj. određene sve parametre u svim zglobovima, zanima nas koja je pozicija svakog pojedinog zgloba u odnosu na globalni koordinatni sustav u kojemu se kostur nalazi. Odnosno, zanima nas

matrica transformacije svakog pojedinog zgloba koja transformira iz lokalnog koordinatnog sustava zgloba u globalni sustava kostura budući da su parametri svakog pojedinog zgloba kostura definirani lokalno, u koordinatnom sustavu zgloba roditelja. Za računanje toga koristi ćemo direktnu kinematiku na sljedeći način:

1. Krenemo obilaziti stablo (kostur) u dubinu (engl. *depth-first search*) počevši od korijenskog čvora.
2. Za svaki čvor (zglob) izračunamo matricu transformacije W:

$$\mathbf{L} - \text{matrica lokalne transformacije zgloba}$$

$$\mathbf{W} - \text{matrica globalne transformacije zgloba}$$

$$\mathbf{W}_p - \text{matrica globalne transformacije zgloba roditelja}$$

$$\mathbf{W} = \mathbf{L} * \mathbf{W}_p$$

   gdje je oblik matrice **L** ovisi o tipu zgloba (vidi 7.4).
3. Obavimo ostale operacije s objektom na temelju izračunatih globalnih matrica transformacija.

7.3 Inverzna kinematika

U složenim kosturima vrlo je teško postaviti kostur u određenu pozu samo manipulirajući ručno zglobovima. Recimo da odaberemo jedan zglob kostura kojeg ćemo nazvati manipulator i da želimo vrh tog manipulatora dovesti u određenu poziciju i orijentaciju u globalnom koordinatnom sustavu. Kako bi smo to uspjeli napraviti moramo znati parametre (kutove, translacije itd.) u svim zglobovima precima tog manipulatora. Te parametre možemo postavljati ručnim podešavanjem svakog zgloba počevši od korijena, što je mukotrpno i neproduktivno, ili pak možemo iskoristiti inverznu kinematiku da to napravi umjesto nas. Znači, ideja inverzne kinematike je izračunati parametre koji moraju biti u zglobovima precima da bi se vrh manipulatora nalazio u nekoj našoj traženoj poziciji i orijentaciji. Inverzna kinematika ima je dobila zato što izračunava obrnuto nego direktna kinematika, a koja nam na temelju parametara u zglobovima precima izračunava globalnu poziciju i orijentaciju vrha manipulatora. Matematički gledano to se može zapisati na sljedeći način:

$$\phi_i - \text{paramteri u zglobu } i, \text{tj. kutovi ili translacije u zglobu}$$

$e_i -$ paramteri u vrhu manipulaotra, tj. pozicija, orijentacija i ostale DOF vrijednosti

$f_i -$ funkcija koja iz $\boldsymbol{\phi}$ izračunava parametar e_i u vrhu manipulatora

$$\boldsymbol{\phi} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_M \end{bmatrix}, \mathbf{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix}, \mathbf{F} = \begin{bmatrix} f_1(\phi_1, \dots, \phi_M) \\ \vdots \\ f_N(\phi_1, \dots, \phi_M) \end{bmatrix}$$

Direktna kinematika:

$$\mathbf{e} = \mathbf{F}(\boldsymbol{\phi})$$

Inverzna kinamtika:

$$\boldsymbol{\phi} = \mathbf{F}^{-1}(\mathbf{e})$$

Postavlja se pitanje kako izračunati funkciju $\mathbf{F}^{-1}()$, odnosno kako riješiti sustav nelinearnih jednadžbi? Analitičko rješavanje problema bilo bih najkvalitetnije kada bi uvijek postojalo točno jedno rješenje sustava, što za veće kinematičke lance nije slučaj jer u mnogim slučajevima može postojati puno rješenja, a nekad se može dogoditi da rješenja niti nema. Također, u puno slučajeva analitičko rješavanje nekih nelinearnih sustava je nemoguće, stoga je nužno pristupiti numeričkom načinu rješavanju nelinearnih sustava jer je ono općenito primjenjivo. Rješavanje sustava nelinearnih jednadžbi je optimizacijski problem za koje postoje razne metode rješavanja od kojih većina uključuje izračunavanje Jakobijeve matrice (Jakobiana) i njenog inverza, no postoje neke heurističke metode poput CCD-a (engl. *Cyclic-Coordinate Descent*) koji je specijalno namijenjen za rješavanje problema inverzne kinematike.

Numerička metoda rješavanja navedenog sustava zasniva se na pokušaju linearizacije nelinearnog problema uvođenjem Jakobiana kao linearnog operatora:

$$\frac{\Delta \mathbf{e}}{\Delta \boldsymbol{\phi}} \approx \frac{\partial \mathbf{F}}{\partial \boldsymbol{\phi}} = \mathbf{J}(\boldsymbol{\phi}) \Leftrightarrow \Delta \mathbf{e} \approx \mathbf{J}(\boldsymbol{\phi}) \Delta \boldsymbol{\phi}$$

$$\mathbf{J}(\boldsymbol{\phi}) = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{\phi})}{\partial \phi_1} & \cdots & \frac{\partial f_1(\boldsymbol{\phi})}{\partial \phi_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N(\boldsymbol{\phi})}{\partial \phi_1} & \cdots & \frac{\partial f_N(\boldsymbol{\phi})}{\partial \phi_M} \end{bmatrix} = \begin{bmatrix} \frac{\partial e_1}{\partial \phi_1} & \cdots & \frac{\partial e_1}{\partial \phi_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_N}{\partial \phi_1} & \cdots & \frac{\partial e_N}{\partial \phi_M} \end{bmatrix}$$

Njegovim inverzom mogao bi se izračunati pomak ka konačnom rješenju sustava koji je definiran s pozom $\mathbf{g}$ traženog manipulatora:

$$\Delta \mathbf{e} \approx \mathbf{J}(\boldsymbol{\phi}) \Delta \boldsymbol{\phi} \Leftrightarrow \Delta \boldsymbol{\phi} \approx \mathbf{J}^{-1}(\boldsymbol{\phi}) \Delta \mathbf{e} = \mathbf{J}^{-1}(\boldsymbol{\phi}) * \beta(\mathbf{g} - \mathbf{e}), \quad 0 \leq \beta \leq 1$$

te bi iterativnim ponavljanjem postupka uz dovoljno malen β došli do konačnog rješenja **g**:

```
dok (e nije dovoljno blizu g) {
  Izračunaj J(Φ) za trenutnu poziciju;
  Izračuna J^-1;
  Δe = β*(g - e);
  ΔΦ = J^-1 * Δe;
  Φ = Φ + ΔΦ;
  Izračunaj novi e za Φ uz pomoć direktne kinematike;
}
```

Kod 5 – Pseudo-kod inverzne kinematike

Postavlja se nekoliko pitanja: kako izračunati J na temelju konfiguracije zglobova, kako izračunati inverz J^{-1} matrice J budući da ona često nije kvadratna, kako odrediti korak integracije β, te kako odrediti kada prekinuti iteriranje?

Budući da Jakobijeva matrica opisuje kako promjena u zglobovima utječe na kretnju vrha manipulatora, nužno je za svaki zglob u kinematičkom lancu pojedinačno odrediti kako njegova kretnja (promjena DOF parametara) utječu na kretnju vrha manipulatora (promjenu pozicije i orijentacije). Taj opis čini jedan stupac matrice. Za ilustraciju uzmimo da imamo 1-DOF zglob koji se može rotirati oko osi zadane vektorom **a** i da imamo 3-DOF vrh manipulatora (tražimo samo poziciju u 3-dimenzionalnom prostoru, bez orijentacije). Prilikom računanja utjecaja zgloba na manipulator potrebno je da i zglob i manipulator budu u istom koordinatnom sustavu, najbolje u globalnom koordinatnom sustavu kostura (engl. *world space*). Utjecaj takvog zgloba na manipulator može se računati na način:

$\mathbf{a}_i$ – vektor smjera osi rotacije u lokalnom koordinatnom sustavu

$\mathbf{r}_i$ – vektor pomaka zgloba u odnosu na zglob roditelja

$\mathbf{e}$ – pozcija vrha manipulatora u globalnnom koordinatnom sustavu

$\mathbf{a}_i'$ – vektor osi rotacije u globalnom koordinatnom sustavu

$\mathbf{r}_i'$ – vektor pomaka zgloba u odnosu na zglob roditelja u globalnom sustavu

$\mathbf{W}_p$ – matrica globalne transformacije zgloba roditelja

$$\mathbf{r}_i = \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix}, \quad \mathbf{a}_i = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix}, \quad \mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \\ 1 \end{bmatrix}$$

$$\mathbf{a}_i' = \mathbf{a}_i * \mathbf{W}_p$$

$$\mathbf{r}_i' = \mathbf{r}_i * \mathbf{W}_p$$

$$\frac{\partial \mathbf{e}}{\partial \phi_i} = \mathbf{a}_i' \times (\mathbf{e} - \mathbf{r}_i)$$

, a Jakobijeva matrica bi se popunila na slijedeći način:

$$\mathbf{J}(\boldsymbol{\phi}) = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_1} & \cdots & \frac{\partial e_x}{\partial \phi_i} & \cdots & \frac{\partial e_x}{\partial \phi_M} \\ \frac{\partial e_y}{\partial \phi_1} & \cdots & \frac{\partial e_x}{\partial \phi_i} & \cdots & \frac{\partial e_y}{\partial \phi_M} \\ \frac{\partial e_z}{\partial \phi_1} & \cdots & \frac{\partial e_x}{\partial \phi_i} & \cdots & \frac{\partial e_z}{\partial \phi_M} \end{bmatrix} = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_1} & \cdots & \left(\mathbf{a}_i' \times (\mathbf{e} - \mathbf{r}_i)\right)_x & \cdots & \frac{\partial e_x}{\partial \phi_M} \\ \frac{\partial e_y}{\partial \phi_1} & \cdots & \left(\mathbf{a}_i' \times (\mathbf{e} - \mathbf{r}_i)\right)_y & \cdots & \frac{\partial e_y}{\partial \phi_M} \\ \frac{\partial e_z}{\partial \phi_1} & \cdots & \left(\mathbf{a}_i' \times (\mathbf{e} - \mathbf{r}_i)\right)_z & \cdots & \frac{\partial e_z}{\partial \phi_M} \end{bmatrix}$$

Za drugačije tipove zglobova i manipulatora izračunavanje utjecaja se može razlikovati (vidi 7.4).

Drugo postavljeno pitanje je što kada se matrica $\mathbf{J}$ ne može invertirati, odnosno nije kvadratna ili je singularna. Problem ne-kvadratne Jakobijeve matrice proizlazi iz toga da stupanj slobode kinematičkog lanca (broj zglobova u lancu) može biti različit od stupnja slobode u vrhu manipulatora (obično 6-DOF, 3-DOF pozicija + 3-DOF orijentacija). Ako kinematički lanac ima više stupnjeva slobode od vrha manipulatora (engl. *underconstrained system*) tada će vjerojatno biti beskonačno mnogo redundantnih rješenja od kojih je potrebno pronaći ono najbolje. Kada kinematički lanac ima manje stupnjeva slobode nego vrh manipulatora (engl. *overconstrained system*) tada sustav vjerojatno neće imati rješenja te se je u tome slučaju poželjno što bliže približiti ciljnom položaju manipulatora. Rješavanja takvih ne-kvadratnih sustava svodi se na pronalaženju pseudo-inverza $\mathbf{J}^\dagger$ te njegovog korištenja umjesto pravog inverza:

$$\mathbf{J}^\dagger = (\mathbf{J}^\intercal \mathbf{J})^{-1} \mathbf{J}^\intercal$$

$$\Delta\boldsymbol{\phi} \approx \mathbf{J}^\dagger(\boldsymbol{\phi}) \Delta \mathbf{e}$$

lako je računanje preko pravog inverza i pseudo inverza daje najpreciznije rezultate u većini slučajeva, ponekad u nekim konfiguracijama Jakobijeva matrica može biti singularna pa je pronalaženje inverza nemoguće (Slika 24).

Slika 24 - Primjer singularne konfiguracije zglobova

Zato je umjesto inverza i pseudo inverza moguće koristiti i transponiranu Jakobijevu matricu $\mathbf{J}^T$ što se u praksi pokazuje kao vrlo zadovoljavajuće rješenje kada je brzina pronalaženja rješenja važnija nego njegova preciznost. Također, korištenjem transponirane matrice nemamo problema sa singularitetima. Jedna od uobičajenih metoda gdje je potrebno zadržati preciznost je korištenje inverza i pseudo-inverza kada je to moguće, a kada se utvrdi singularitet tada se prebaci na metodu s transponiranom matricom ili CCD metodu. Određivanje singularnosti i računanje inverza/pseudo-inverza Jakobijeve matrice može se ostvariti npr. korištenjem SVD (engl. *singular value decomposition*) ili LU dekompozicije.

Još jedan važni faktor pri pronalaženju rješenja u numeričkoj metodi je određivanja koraka integracije β. Iako se može uzimati fiksni korak, preporučljivo je adaptivno ga mijenjati ili još bolje nekim postupkom optimirati (npr. postupak zlatnog reza). Prethodno naveden i korišten postupak numeričke integracije, poznat kao Eulerov postupak, može biti spor za mali korak integracije, stoga korištenje drugih metoda integracije poput klasičnog ili adaptivnog Runge-Kutta postupka može poboljšati brzinu konvergencije ka rješenju.

Nakon što smo započeli numeričku potragu za rješenjem potrebno je odrediti kriterije prekida potrage. Tri su uvjeta zaustavljanja iteriranja:

1. Kada smo pronašli zadovoljavajuće rješenje. Obično kada je rješenje, tj. vrh manipulatora u određenom tolerantnom području.
2. Zapeli smo u lokalnom minimumu. Tada je moguće ili promijeniti metodu pronalaženja rješenja (npr. prebaciti se na CCD) ili promijeniti veličinu koraka integracije ili završiti pretraživanje i vratiti najbolje pronađeno rješenje.

3. Pretraga traje vremenski predugo. Na primjer, u interaktivnom načinu rada ne bihsmo željeli čekati. Tada također treba vratiti najbolje pronađeno rješenje.

Konačno, na kraju nam još jedino preostaje spomenuti rješavanje problema eksplicitnih ograničenja u zglobovima (npr. zglob se ne može rotirati za 180° nego od 0°-90°). Problem se može riješiti tako da se na kraju svake iteracije integracije vrijednosti u zglobovima koje su van ograničenja pomaknu na bližu granicu eksplicitnog ograničenja.

7.4 Tipovi zglobova

U ovom odjeljku biti će prikazani najčešći tipovi zglobova koji se koristiti za definiranje kostura u 3-dimenzionanom prostoru, kao i njihove pripadne matrice lokalne transformacije.

Translacijski 1-DOF zglob

– translacija po vektoru **a** za parametar t:

$$\mathbf{L}_T(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ r_x + a_x * t & r_y + a_y * t & r_z + a_z * t & 1 \end{bmatrix}$$

r – konstantni vektor pomaka

Translacijski 3-DOF zglob

– translacija za vekor **t**:

$$\mathbf{L}_T(\mathbf{t}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ r_x + t_x & r_y + t_y & r_z + t_z & 1 \end{bmatrix}$$

Rotacijski 1-DOF zglob

– rotacija oko osi x:

$$\mathbf{L}_{Rx}(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x & 0 \\ 0 & -\sin\theta_x & \cos\theta_x & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

– rotacija oko osi y:

$$\mathbf{L}_{Ry}(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_y & 0 & \cos\theta_y & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

– rotacija oko osi z:

$$\mathbf{L}_{Rz}(\theta_z) = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 & 0 \\ -\sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

–rotacija oko osi definirane vektorom **a**:

$$\mathbf{L}_{Ra}(\theta) = \begin{bmatrix} a_x^2 + c_\theta(1-a_x^2) & a_x a_y(1-c_\theta) + a_z s_\theta & a_x a_z(1-c_\theta) - a_y s_\theta & 0 \\ a_x a_y(1-c_\theta) - a_z s_\theta & a_y^2 + c_\theta(1-a_y^2) & a_y a_z(1-c_\theta) + a_x s_\theta & 0 \\ a_x a_z(1-c_\theta) + a_y s_\theta & a_y a_z(1-c_\theta) - a_x s_\theta & a_z^2 + c_\theta(1-a_z^2) & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

$c_\theta \equiv \cos\theta\,, \qquad s_\theta \equiv \sin\theta$

Rotacijski 2-DOF zglob

– rotacija oko osi x pa oko osi y:

$$\mathbf{L}_{Rxy}(\theta_x, \theta_y) = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y & 0 \\ \sin\theta_x * \sin\theta_y & \cos\theta_x & \sin\theta_x * \cos\theta_y & 0 \\ \cos\theta_x * \sin\theta_y & -\sin\theta_x & \cos\theta_x * \cos\theta_y & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

– rotacija oko osi x pa oko osi z:

$$\mathbf{L}_{Rxz}(\theta_x, \theta_z) = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 & 0 \\ -\cos\theta_x * \sin\theta_z & \cos\theta_x * \cos\theta_z & \sin\theta_x & 0 \\ \sin\theta_x * \sin\theta_z & -\sin\theta_x * \cos\theta_z & \cos\theta_x & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

– rotacija oko osi y pa oko osi z:

$$\mathbf{L}_{Ryz}(\theta_y, \theta_z) = \begin{bmatrix} \cos\theta_y * \cos\theta_z & \cos\theta_y * \sin\theta_z & -\sin\theta_y & 0 \\ -\sin\theta_z & \cos\theta_z & 0 & 0 \\ \sin\theta_y * \cos\theta_z & \sin\theta_y * \sin\theta_z & \cos\theta_y & 0 \\ r_x & r_y & r_z & 1 \end{bmatrix}$$

8. Zaključak

Tema ovog rada odabrana je s ciljem da se istraži i implementira neki drugačiji način animiranja kretnje likova u računalnim igrama, koji nije zasnovan samo na predefiniranim ključnim pozama već koristi i neke naprednije proceduralne tehnike animiranja. Jedan od takvih (boljih) načina animiranja kretnje je i onaj opisan u radu Rune Skovbo Johansen-a (2009) u kojemu je opisao način animiranja likova u igrama, a koji je implementiran i sažeto opisan u ovome diplomskim radu. Prilikom izrade ovoga rada nit vodilja je bila da se ostvari sustav identičan onome iz navedenog rada, ali implementiran u sklopu Serious Engine-a. Između ostalog jedan od ciljeva rada je i bio unaprijediti postojeći sustav animiranja unutar navedenog sustava. Jedan od *game engine*-a, koji već ima ugrađene slične algoritme animiranja kretanja likova je i *game engine* RAGE (*Rockstar Advanced Game Engine*), unutar kojeg je izrađena planetarno popularna i jedna od najprodavanija igra svih vremena Grand Theft Auto V, a koja je i poznata po vrlo kvalitetnim animacijama kretanja likova.

Tijekom rada na ovome diplomskom radu došao sam i do informacije da postoji sustav gdje se uz pomoć umjetnih neuronskih mreža ostvaruje vrlo realistične animacije[1]. Sama spoznaja da se uz pomoć neuronskih mreža mogu ostvariti tako realne animacije otvaraju veliku mogućnost i u drugim aspektima animiranja, stoga će zasigurno biti tema budućih istraživanja.

[1] Više informacija na http://theorangeduck.com/page/phase-functioned-neural-networks-character-control

9. Literatura

2009. Automated Semi-Procedural Animation for Character Locomotion. *Automated Semi-Procedural Animation for Character Locomotion.* [Mrežno] 25. May 2009. [Citirano: 21. May 2017.] http://runevision.com/thesis/.

Johansen, Rune Skovbo. 2009. *Automated Semi-Procedural Animation.* 2009.

Technologies, Unity. 2017. Unity User Manual . *Unity User Manual .* [Mrežno] Unity Technologies, 19. 6 2017. [Citirano: 19. 6 2017.] https://docs.unity3d.com/Manual/index.html.

10. Sažetak

Implementacija poluproceduralne animacije kretnje likova u 3D računalnoj igri

Računalna animacija danas je neizostavni dio svake računalne igre. Većina današnjih igara koristi samo osnovne tehnike animiranja likova, zasnovanih na ključnim trenutcima pokreta. Ovaj rad ide korak dalje i te uz osnovne tehnike animiranja opisuje i neke proceduralne tehnike poput dinamičkog prilagođavanja nogu podlozi. Tako sustav omogućava zadržavanje stilističke prirodnosti koje pružaju predefinirane animacije, a iskorištava robusnost proceduralnog računanja. Sustav za animiranje kretnje likova je ostvaren unutar veće postojećeg sustava za izradu računalnih igara SeriousEngine te nadograđuje sustav koji je zasnovan na animacijama preko ključnih poza.

Implementation of Semi-Procedural Animation for Character Locomotion

Today computer animation is one of the most important aspects of every video game. Most of today's video games uses only basic techniques of animating characters, like key-framed animations. This work goes one step forward and combines key-framed animations with some procedural animition techniques like dynamically adjustment of character's legs to the ground. Because of that system keeps stylistic naturalness of key-framed animations and uses robustness of procedural computing. Animation system is implemented as part of already existing game engine SeriousEngine and upgrades system that is based on key-framed animations.

11. Ključne riječi

računalna animacija, proceduralna animacija, video igre, animacije zasnovane na ključnim pozama, inverzna kinematika, polu-proceduralna animacija kretnje likova, C++, SeriousEngine

computer animation, procedural animation, video games, key-framed animations, inverse kinematics, semi-procedural character locomotion, C++, SeriousEngine