SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6821

SIMULACIJA DVONOŽNOG MODELA BIPED

Maja Radočaj

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6821

SIMULACIJA DVONOŽNOG MODELA BIPED

Maja Radočaj

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6821

Pristupnica:	**Maja Radočaj (0036505371)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Simulacija dvonožnog modela biped**

Opis zadatka:

Proučiti modele objekata koji se kreću na dvije noge (engl. biped). Proučiti mogućnosti kretanja takvih objekata. Proučiti mogućnosti fizikalnog pogona Open Dynamics Engine (ODE). Razmotriti mogućnosti primjene fizikalnog pogona na dvonožnom modelu biped te ostvariti prikaz simuliranog modela. Razraditi simulaciju kretanja na nekoliko dvonožnih modela. Diskutirati utjecaj različitih parametara. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Koristiti fizikalni pogon Open Dynamics Engine (ODE). Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2020.

Sadržaj

Uvod

Biped (dvonožac) je biće koje koristi dvije noge za kretanje. Iako naizgled jednostavno, kretanje pomoću dvije noge predstavlja kompleksan proces u koji su uključene brojne sile koje omogućuju održavanje ravnoteže i uspravnog hoda. Sve te sile moraju biti savršeno usklađene kako biped ne bi izgubio ravnotežu. Unatoč tome što nam je priroda tek kroz milijune godina evolucije omogućila efikasno i stabilno kretanje po različitim površinama i preprekama, znanstvenici na vodećim sveučilištima i centrima za istraživanje neumorno pokušavaju simulirati procese uključene u autonomno kretanje bipeda.

Simulacija kretanja bipeda ima višestruke moguće primjene. Najatraktivnija od njih zasigurno predstavlja dizajniranje robota na temelju simuliranog modela. Dobro dizajnirani roboti predstavljali bi zamjenu za ljude na opasnim terenima te bi mogli obavljati rizične poslove poput potrage za osobama na nepristupačnim područjima ili obavljanje zadataka u toksičnom ili radioaktivnom okolišu. Već nekoliko takvih robota je osmišljeno i proizvedeno, a samo neki od njih su robot Atlas firme Boston Dynamics [1] i bipedalni roboti dizajnirani na sveučilištu u Massachusettsu [2].

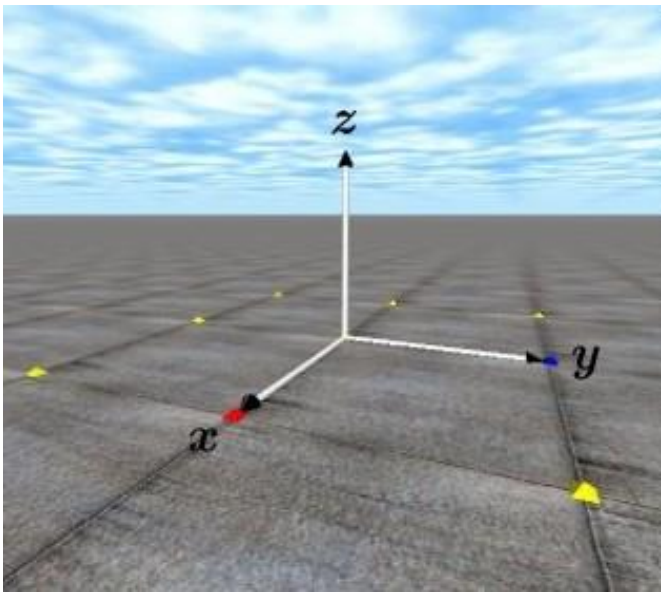
U okviru ovog rada simuliran je model jednostavnog bipeda koristeći fizikalni pogon Open Dynamics Engine (ODE). Detaljno je opisana struktura programa za stvaranje i simulaciju modela povezanog zglobovima, kao i funkcije ODE-a korištene u tu svrhu. Pošto simuliranje kretanja predstavlja vrlo kompleksan problem, bit će prikazani jednostavni pokreti koristeći sile u zglobovima.

1. Open Dynamics Engine

Implementacija simulacije temelji se na fizikalnom pogonu Open Dynamics Engine (ODE). ODE je besplatna biblioteka otvorenog koda za simuliranje fizike krutih tijela [3]. Pogodna je za korištenje jer je stabilna, neovisna o korištenoj platformi i nudi jednostavno sučelje za C i C++. Razvijena je 2001. godine i od tada se koristi u mnogim računalnim igricama i alatima za 3D simulacije dinamičkih interakcija tijela u prostoru. Sav kod u okviru ovog rada napisan je u jeziku C++ koristeći ODE.

1.1. ODE prostor i objekti

ODE nudi reprezentaciju 3D prostora koristeći desni pravokutni koordinatni sustav. Pozicije i orijentacije svakog objekta u sceni zadaju se koristeći vrijednosti x, y i z koordinate (Slika 1.1). Pri tome biblioteka pretpostavlja korištenje SI sustava mjernih jedinica. To znači da će težina svakog objekta biti zadana u kilogramima, duljina u metrima, a vrijeme u sekundama [4].



Slika 1.1 – koordinatni sustav u ODE

Za iscrtavanje 3D prostora i grafiku koristi se biblioteka Drawstuff. To je vrlo jednostavna biblioteka pogodnija za manje simulacije i demonstracije. Primjenjiva je na operacijskim

sustavima Windows i Linux. Korisnik se može pritiskom i pomicanjem tipki miša kretati kroz virtualni prostor [5].

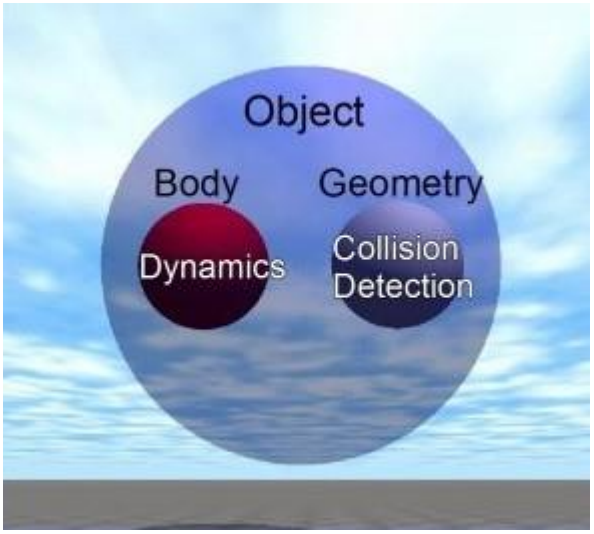
Postoje različiti objekti koji se mogu stvoriti u ODE aplikaciji:

- `dWorld` – dinamički virtualni svijet
- `dSpace` – prostor kolizije
- `dBody` – kruto tijelo
- `dGeom` – geometrija korištena za koliziju
- `dJoint` – zglob
- `dJointGroup` – skupina zglobova

Objekt dWorld stvara se na početku ODE programa i uništava na kraju. Korisnik može po želji postaviti gravitaciju i ostale parametre u svijetu potrebne za izračunavanje dinamike. O ostalim objektima bit će govora u nastavku. Funkcije koje rade s objektima primaju kao argument i vraćaju identifikacijske oznake objekata [6]. Tipovi tih oznaka su `dWorldID`, `dBodyID`, itd.

1.2. Tijelo i geometrija

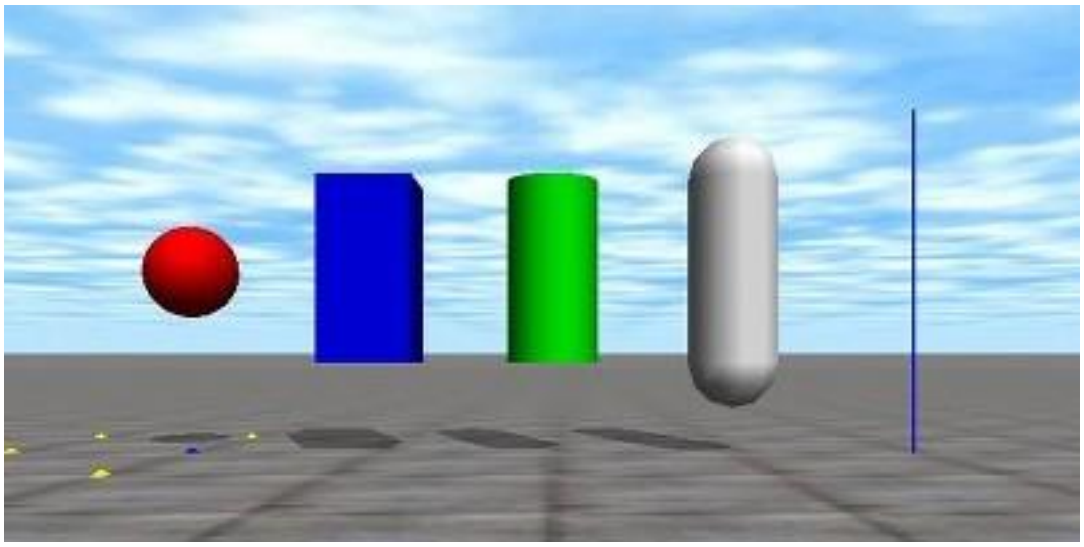
Objekt u ODE-u ima dva dijela: tijelo i geometriju, kao što je prikazano na slici (Slika 1.2). Tijelo služi za simuliranje dinamike, a geometrija za detektiranje kolizije (sudara). Jezgra dinamike i biblioteka za detektiranje sudara implementirani su odvojeno.



Slika 1.2 – grafički prikaz objekta u ODE

Prednost ovakvog pristupa je jednostavno omogućeno korištenje različitih biblioteka za detektiranje sudara. ODE trenutačno koristi OPCODE biblioteku i Gimpact biblioteku za detektiranje sudara. OPCODE je pretpostavljena ODE biblioteka te je brža, dok je biblioteka Gimpact preciznija. Još jedna prednost ovakvog modela je što je za neki objekt moguće primijeniti samo detektiranje sudara bez računanja dinamike ili obrnuto. Zbog toga se u ODE programu na početku stvara svijet `dWorld` u okviru kojeg se provode izračuni potrebni za dinamiku. Ako se računaju i sudari, stvara se prostor za detekciju sudara dSpace.

Kao što je već rečeno, svaki objekt onda ima dva parametra, tijelo i geometriju. Tijelo je tipa `dBody`, a geometrija tipa `dGeom`. Tijela i geometrije mogu biti različitih oblika. Pri tome su za tijelo važne proporcije, masa i centar mase za izračune dinamike. Potrebno je za svako stvoreno tijelo postaviti poziciju i rotaciju u svijetu pomoću funkcija `dBodySetPosition` i `dBodySetRotation`. Obje funkcije primaju identifikacijsku oznaku tijela čiju poziciju ili rotaciju treba postaviti te x, y i z koordinate pozicije u koordinatnom sustavu ili rotacijsku matricu. Stvoreno tijelo može biti oblika kvadra, sfere, cilindra ili kapsule. Geometrija može biti jedno kruto tijelo ili skup drugih geometrija koje služe za ubrzavanje procesa detekcije sudara (to je `dSpace`). Za korištenje sudara potrebno je povezati tijelo s odgovarajućom geometrijom kako bi pogon mogao izračunati poziciju i orijentaciju geometrija koja pripadaju tijelima u određenom trenutku. Dakle, same geometrije nemaju nikakva dinamička svojstva poput mase ili brzine. Svaka geometrija može imati sudar s nekom drugom geometrijom te time stvoriti nula ili više točki dodira.



Slika 1.3 – osnovni tipovi geometrija (sfera, kvadar, cilindar, kapsula, zraka)

Također, svaka geometrija primjerak je jednog razreda koji definira određeni geometrijski oblik. Postoji nekoliko ugrađenih razreda:

- Sfera (engl. sphere) – stvara se funkcijom `dCreateSphere`
- Kvadar (engl. box) – stvara se funkcijom `dCreateBox`
- Ravnina (engl. plane) – stvara se funkcijom `dCreatePlane`
- Kapsula (engl. capsule) – stvara se funkcijom `dCreateCapsule`
- Zraka (engl. ray) – stvara se funkcijom `dCreateRay`

Svaka od funkcija za stvaranje geometrija vraća identifikacijsku oznaku geometrije `dGeomID`. Dodatno, za svaki od razreda definirane su dodatne funkcije za podešavanje parametra geometrija koje kao parametar primaju identifikacijsku oznaku. Navedeni primjeri geometrija prikazani su na slici (Slika 1.3).

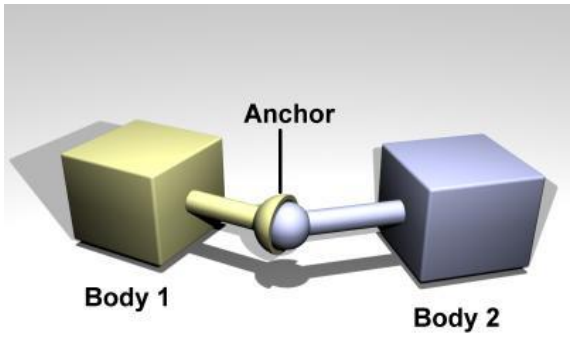
1.3. Zglobovi

Različiti zglobovi nalaze se svugdje oko nas. Osim na vlastitim tijelima, zglobove možemo vidjeti i na predmetima poput zgloba koji omogućuje otvaranje i zatvaranje vrata, prozora, laptopa, zglobovi na stolnim lampama, itd. Svaki zglob povezuje dva tijela. Dakle, zglob predstavlja ograničenje koje definira mogućnosti kretanja dva tijela koje povezuje.

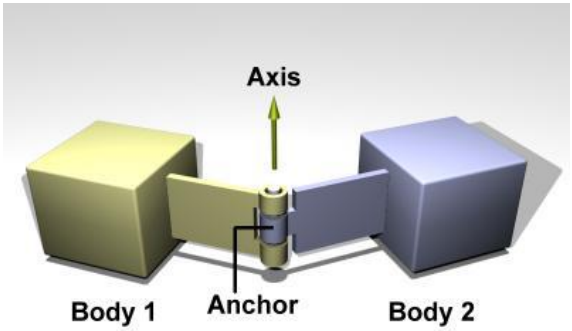
U okviru fizikalnog pogona ODE, zglob povezuje dva postojeća tijela. Funkcije za stvaranje zglobova vraćaju identifikacijsku oznaku zgloba `dJointID`, dok funkcije za povezivanje dva tijela zglobom primaju tu oznaku te dvije oznake `dBodyID` tijela koja trebaju biti povezana. Osim jedinstvenih zglobova, omogućeno je stvaranje i rad sa skupinama zglobova. Neki od tipova zglobova implementiranih u ODE:

- Zglob „kugla i utičnica“ (engl. ball and socket) – zglob bez ograničenja rotacije (Slika 1.4)
- Šarka (engl. hinge) – rotacija omogućena u smjeru jedne osi (Slika 1.5)
- Klizeći zglob (engl. slider) – kretanje omogućeno samo u smjeru definirane osi (Slika 1.6)
- Univerzalni zglob (engl. universal) – poput „kugle i utičnice“, ali ograničene rotacije u jednoj osi (Slika 1.7)

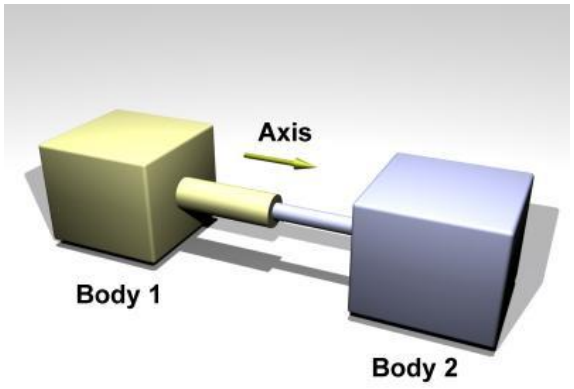
- Fiksirani zglob (engl. fixed) – održavanje jedne pozicije i orijentacije među dva tijela ili tijela i okruženja



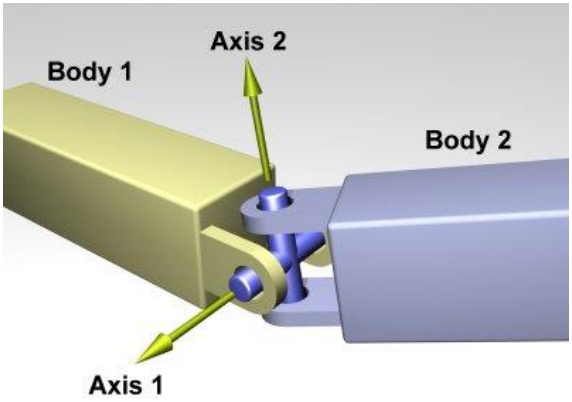
Slika 1.4 – zglob „lopta i utičnica“



Slika 1.5 – šarka



Slika 1.6 – klizeći zglob



Slika 1.7 – univerzalni zglob

Pri stvaranju zgloba potrebno je definirati mjesto hvatišta i osi kretanja. Ako te vrijednosti nisu eksplicitno definirane, hvatište se postavlja na (0, 0, 0), a os na (0, 0, 1) (u smjeru pozitivne z osi). Važno je u početku ispravno pozicionirati tijela koja se povezuju zglobom, inače je moguće da dođe do greške pri inicijaliziranju parametara i povezivanju tijela.

Kad je zglob tek stvoren, ne postoje nikakva ograničenja na njegov raspon kretanja. Raspon kretanja može biti ograničen pozivanjem odgovarajućih funkcija. U tom slučaju, korisnik može definirati najveći ili najmanji kut ili poziciju unutar koje se zglob može kretati; kad zglob dosegne taj kut, zaustavlja se. Također, omogućeno je podešavanje i drugih parametara zglobova, poput parametra uklanjanja greški, parametra odskoka pri zaustavljanju, itd.

U dinamičkom sustavu, u svakom trenutku na svaki zglob djeluje određena sila. Korištenjem motora u zglobovima (u ODE-u `AMotor`) omogućeno je kontroliranje brzina. Moguće je ograničiti i maksimalnu brzinu u danom zglobu.

1.4. Brzina, akceleracija i sila

Svako stvoreno tijelo može imati određenu brzinu. Postoje funkcije koje omogućavaju postavljanje linearne i kutne brzine te dobivanje brzine tijela. Brzina se može postaviti u početnom stanju, ali nije preporučeno mijenjanje brzine u svakom vremenskom koraku simulacije zbog lošeg utjecaja na fizikalni model.

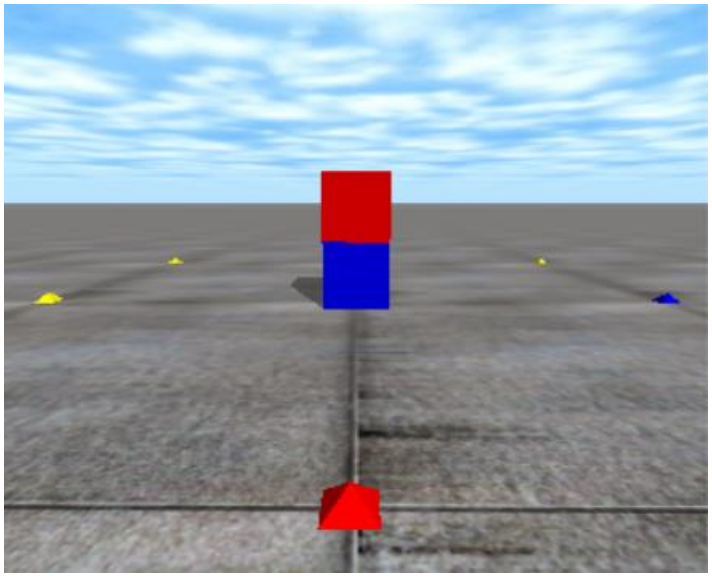
Za razliku od brzine, za dobivanje i mijenjanje akceleracije ne postoje definirane funkcije. Ako želimo dobiti akceleraciju tijela, potrebno je očitati promjenu brzine pomoću postojeće funkcije `dBodyGetLinearVel` i podijeliti ju s promjenom vremena u jednom vremenskom koraku. Vremenski korak simulacije određuje koliko brzo se simulacija provodi te se daje kao drugi argument u funkciji za definiranje koraka `dWorldStep`.

Sile i momente sile također je moguće očitati i postaviti koristeći odgovarajuće funkcije. Postavljanje sile na tijelo rezultira silom usmjerenom iz težišta tijela prema smjeru osi koje je korisnik postavio. Sile se akumuliraju na tijelu te se opet postavljaju na nulu nakon svakog vremenskog koraka. Postoje i funkcije koje osim sile primaju i vektor pozicije koji određuju točku na kojoj se primjenjuje sila (vektor je zadan koordinatama relativnima za tijelo).

Također, osim na tijela moguće je postaviti silu i moment sile na zglob. Za dobivanje informacija o silama i momentima sile na zglobu potrebno je alocirati memoriju i pozvati funkciju `dJointGetFeedback`. Razlog zbog kojeg je potrebno alocirati memoriju je poboljšanje performansi – pošto u većini slučajeva nije potrebno očitavati sile sa svakog zgloba, korisnik može sam odabrati s kojih zglobova su mu potrebne te informacije.

Dok motor u zglobu omogućava postavljanje brzine, implementirane su i funkcije za direktno dodavanja sila na zglob [7]. Te funkcije implementirane su tako da dodaju silu ili moment sile tijelima koje zglob povezuje. Postoje različite funkcije s obzirom na vrstu zgloba koje povezuju. Tako je za zglob tipa šarke omogućeno dodavanje momenta sile pozivom funkcije `dJointAddHingeTorque`. Funkcija primjenjuje moment sile u smjeru osi zgloba na jednom tijelu, a na drugom primjenjuje moment sile jednakog iznosa i

suprotnog predznaka. Funkcija `dJointAddSliderForce` pak primjenjuje silu na kliznom tipu zgloba.



Slika 1.8 – dva tijela povezana fiksnim zglobom

```
78696 Force fx= -0.00 fy= -0.00 fz=  9.80
78697 Force fx=  0.00 fy=  0.00 fz=  9.80
78698 Force fx= -0.00 fy= -0.00 fz=  9.80
78699 Force fx=  0.00 fy=  0.00 fz=  9.80
78700 Force fx= -0.00 fy= -0.00 fz=  9.80
78701 Force fx=  0.00 fy=  0.00 fz=  9.80
78702 Force fx= -0.00 fy= -0.00 fz=  9.80
78703 Force fx=  0.00 fy=  0.00 fz=  9.80
```

Slika 1.9 – ispis sila koje djeluju na zglob

Na prvoj slici (Slika 1.8) prikazana je simulacija dva tijela, oba mase 1 kg, povezanih fiksnim zglobom, dok je na drugoj slici (Slika 1.9) prikazan ispis programa. Pošto je gravitacija u glavnom dijelu programa postavljena na -9.8 m/s^2 (u smjeru negativne z osi, odnosno prema dolje), sila prikazana na zglobu vrijednosti je 9.8 N kao posljedica drugog Newtonovog zakona.

1.5. Tok simulacije

Programi za simulaciju dinamike koristeći ODE prate određeni tok. Aplikacijsko programsko sučelje (API) definira funkcije čiji nazivi počinju s malim slovom „d“, dok funkcije vezane uz biblioteku Drawstuff počinju sa „ds“.

Tipični tok simulacije izvodi se sljedećim redoslijedom:

- Postavljanje parametara biblioteke `Drawstuff`
  - Postavljanje očišta i gledišta kamere funkcijom `dsSetViewPoint`
- Inicijaliziranje ODE-a funkcijom `dInitOde`
- Stvaranje svijeta za izračun dinamike funkcijom `dWorldCreate`
- Postavljanje gravitacije funkcijom `dWorldSetGravity`
- Stvaranje tijela funkcijom `dBodyCreate`
  - Postavljanje mase, pozicije i rotacije tijela
- Stvaranje geometrije tijela
- Izvođenje simulacije u korisnički definiranoj funkciji `simLoop`
  - Poziv funkcije `dWorldStep`
  - Kod za izvođenje simulacije
- Uništavanje svijeta funkcijom `dWorldDestroy`
- Zatvaranje ODE-a funkcijom `dCloseODE`

U funkciji `simLoop`, svaki korak simulacije poziva se funkcija `dWorldStep` čiji zadatak je napraviti vremenski korak u simulaciji te izračunati trenutno stanje tijela u svijetu. U kodu za izvođenje simulacije obavlja se iscrtavanje objekata u sceni pomoću biblioteke Drawstuff. Isto tako, tada se računaju i kolizije tijela. Ako se radi sa sudarima, u ovom dijelu treba pozvati funkciju `dSpaceCollide`, a kao argument predati joj funkciju koju korisnik definira za detektiranje svih mogućih dodirnih točaka geometrija u prostoru. Ta korisnički definirana funkcija obično je naziva `nearCallback`.

2. Implementacija

Primjenom objašnjenih funkcija ODE-a moguće je implementirati jednostavnog bipeda povezanog s nekoliko zglobova. U okviru ovog rada napisan je jednostavni program u C++-u za simuliranje tri različita bipeda. Svaki od njih različitih je dimenzija te se sastoje od četiri zgloba tipa šarke i pet dijelova tijela u obliku kvadrova. Kao integrirano razvojno okruženje (IDE) korišten je program Code::Blocks. Prije samog početka, potrebno je instalirati ODE biblioteku. Taj postupak detaljno je opisan za korisnike Windows operacijskog sustava i Code::Blocks-a [8]. Instalacija dolazi s brojnim jednostavnim demonstracijskim programima u kojima korisnik može pregledati mogućnosti ODE-a.

2.1. Zaglavlja i glavna funkcija

Za korištenje funkcija ODE-a i biblioteke Drawstuff dovoljno je na početku programa navesti sljedeća zaglavlja:

```
#include <ode/ode.h>
#include <drawstuff/drawstuff.h>
```

Ako je struktura projekta namještena prema uputama za instalaciju, ovime će biti omogućeno korištenje svih definiranih funkcija.

Simulacija kreće pokretanjem glavne funkcije programa:

```
int main (int argc, char **argv) {
      setDrawStuff();
      dInitODE();
      world = dWorldCreate();
      space = dHashSpaceCreate(0);
      contactgroup = dJointGroupCreate(0);
      dWorldSetGravity(world,0,0,-0.8);

      ground = dCreatePlane(space,0,0,1,0);
      createBiped();

      dsSimulationLoop (argc,argv,500,500,&fn);
      dWorldDestroy (world);
      dCloseODE();
```

```
        return 0;
}
```

Kod 2.1 – glavna funkcija za početak simulacije

U prvom retku funkcije poziva se `setDrawStuff`, funkcija čija je uloga podešavanje biblioteke Drawstuff. U njoj se postavlja verzija biblioteke, putanja do direktorija s teksturama te se postavljaju pokazivači na funkcije zadužene za izvođenje simulacije `simLoop`, funkcije zadužene za početak `start` i funkcije za registriranje i obradu događanja pritiska tipki `command`. Nešto više o svakoj od ovih funkcija bit će u nastavku.

Kao što je već opisano u poglavlju o toku simulacije, sljedeći korak je inicijaliziranje biblioteke ODE te stvaranje dinamičkog svijeta i prostora za detekciju sudara. Potom se inicijalizira globalna varijabla `contactgroup` potrebna pri detekciji sudara koja predstavlja skup zglobova. Gravitacija je postavljena na -0.8 m/s^2 kako bi se usporilo padanje bipeda i jasnije prikazali utjecaji sile na zglobove.

Sljedeći korak je stvaranje objekata u prostoru. Prvo se stvara pod u obliku površine smještene na visini y = 1 koordinatnog sustava. Površina se odmah povezuje s prostorom radi detekcije sudara. Nakon toga slijedi funkcija za stvaranje bipeda koja će također biti opisana u nastavku. Početak izvođenja simulacije počinje pozivom funkcije `dsSimulationLoop`. Brojčani argumenti u funkciji označavaju veličinu prozora za prikaz.

Konačno, pri završetku simulacije potrebno je uništiti dinamički svijet pozivom funkcije `dWorldDestroy` te zatvoriti biblioteku ODE funkcijom `dCloseODE`.

2.2. Funkcija za stvaranje bipeda

Funkcija za stvaranje bipeda createBiped ovisno o globalnoj varijabli bipedId stvara jednog od moguća tri bipeda. Pri tome inicijalizira svih pet dijelova tijela te stvara i podešava parametre četiri zgloba kojima su dijelova tijela povezani. Za dijelove tijela definirana je pomoćna struktura podataka:

```
typedef struct {
        dBodyID body;
        dGeomID geom;
        dReal proportions[3];
        dReal mass;
```

```
} bodyPart;
```

Dakle, svaki dio tijela definiran je svojim tijelom za dinamiku, geometrijom za sudare, proporcijama te masom. Sve operacije nad dijelovima tijela obavljaju se nad globalnom varijablom polja dijelova tijela veličine pet. Osim dijelova tijela, kao globalne varijable postavljena su i četiri zgloba: `hipR`, `hipL`, `kneeR` i `kneeL`.

Prvi korak u stvaranju bipeda je postavljanje odgovarajućih proporcija dijelova tijela ovisno o tome koji se biped trenutno prikazuje:

```
switch(bipedId) {
      case 1:
            setBodyPartProportions1();
            break;
      case 2:
            setBodyPartProportions2();
            break;
      case 3:
            setBodyPartProportions3();
            break;
}
```

Kod 2.2 – postavljanje proporcija dijelova tijela

Svaka od funkcija postavlja proporcije svakog od dijelova tijela na unaprijed definirane vrijednosti. Vrijednosti proporcija definirane su u globalnim varijablama koje spremaju dužinu tijela u smjeru x, y i z koordinate. Kao što ćemo kasnije vidjeti, prvi biped namješten je da bude visine 2,2 metra, drugi 1,2 metra, a treći 2,5 metara.

Nakon definiranja proporcija tijela, slijedi stvaranje samih dijelova tijela i njima odgovarajućih geometrija za računanje sudara:

```
for (int i = 0; i < NUM; i++) {
      bodyParts[i].mass = 1.0;
      bodyParts[i].body = dBodyCreate(world);
      dMassSetZero(&m[i]);
      dMassSetBoxTotal(&m[i], bodyParts[i].mass,
      bodyParts[i].proportions[0],
      bodyParts[i].proportions[1],
      bodyParts[i].proportions[2]);
      dBodySetMass(bodyParts[i].body, &m[i]);
      bodyParts[i].geom = dCreateBox(space,
      bodyParts[i].proportions[0],
```

```
            bodyParts[i].proportions[1],
            bodyParts[i].proportions[2]);

            dGeomSetBody(bodyParts[i].geom,bodyParts[i].body);
      }
```

Kod 2.3 – petlja za stvaranje tijela i geometrije dijelova tijela

U petlji varijabla `NUM` predstavlja globalnu varijablu koja označava broj dijelova tijela po bipedu (u ovom slučaju, njena vrijednost je 5). Masa svakog dijela postavlja se na 1 kg te se definira geometrija tijela koristeći funkciju `dCreateBox`. Geometrija se postavlja pripadnom dijelu tijela funkcijom `dGeomSetBody`. Nakon stvaranja dijelova tijela, potrebno ih je pozicionirati u prostoru:

```
switch(bipedId) {
      case 1:
            setPositions1();
            break;
      case 2:
            setPositions2();
            break;
      case 3:
            setPositions3();
            break;
}
```

Kod 2.4 – postavljanje pozicija dijelova tijela

Ovisno o trenutno prikazanom bipedu i njegovim proporcijama, svaka funkcija postavlja svaki dio tijela na određenu poziciju nizom poziva funkcije `dBodySetPosition`. Važno je odmah definirati poziciju svakog dijela tijela posebno kako ne bi došlo do problema sa sudarima ako se tijela preklapaju. Također, pozicije tijela ključne su pri definiranju položaja zglobova i osi preklapanja. Pošto nije dovoljno samo „spojiti“ dijelove tijela zglobovima, potrebno je imati točno određene pozicije tijela kako bi se hvatišta zglobova točno odredila.

Jedino što je još ostalo za stvaranje bipeda je upravo dodavanje zglobova i postavljanje njihovih parametara. To je implementirano u funkciji `createJoints`:

```
void createJoints() {
      hipR = dJointCreateHinge(world, 0);
```

```
        dJointAttach(hipR,bodyParts[0].body,bodyParts[1].body);
        dJointSetHingeAxis(hipR, 1.0, 0.0, 0.0);
        // ...
        // Isti postupak ponavlja se za hipL, kneeR i kneeL
        // ...
        switch(bipedId) {
              case 1:
                    setJoints1();
                    break;
              case 2:
                    setJoints2();
                    break;
              case 3:
                    setJoints3();
                    break;
        }
}
```

Kod 2.5 – djelomična funkcija za stvaranje zglobova

Kao što je već spomenuto, sva četiri zgloba su tipa šarke te se stvaraju funkcijom `dJointCreateHinge`. Funkcijom `dJointAttach` spajaju se dva dijela tijela koristeći zglob predan u funkciji. Konačno, postavlja se od na kojoj je ograničena rotacija zgloba funkcijom `dJointSetHingeAxis`. Ovaj isti postupak ponavlja se još tri puta za ostale zglobove, a jedina razlika je u osi rotacije i dijelovima tijela koje zglobovi spajaju. Nakon što su zglobovi stvoreni, potrebno ih je još jedino pozicionirati u prostoru pozivima funkcije `dJointSetHingeAnchor` te postaviti im ograničenja na rotaciju funkcijom `dJointSetHingeParam`. Taj posao obavlja se ovisno o trenutno prikazanom bipedu i ovisi o njegovim proporcijama. Definiranjem tih ograničenja biped je izgrađen i spreman za simulaciju.

2.3. Funkcija za izvođenje simulacije

Nakon što je biped izgrađen i povezan zglobovima, moguće je započeti simulaciju. Petlja simulacije u svakom koraku poziva funkciju `simLoop` koja iscrtava scenu u danome trenutku:

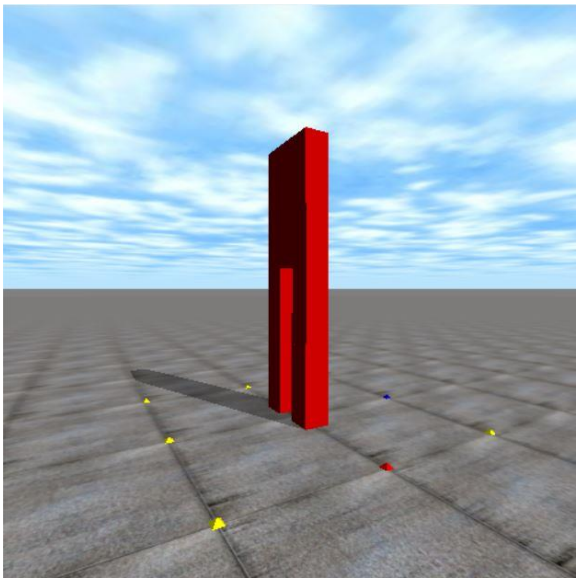
```
static void simLoop (int pause) {
      const dReal *pos[NUM], *R[NUM];
```

```
        dSpaceCollide(space, 0, &nearCallback);
        dWorldStep(world, 0.0004);
        dJointGroupEmpty(contactgroup);

        if (bipedId == 1) dsSetColor(1.0, 0.0, 0.0);
        else if (bipedId == 2) dsSetColor(0.0, 1.0, 0.0);
        else dsSetColor(0.0, 0.0, 1.0);

        //draw biped
        for(int i = 0; i < NUM; i++) {
              pos[i] = dBodyGetPosition(bodyParts[i].body);
              R[i] = dBodyGetRotation(bodyParts[i].body);
              dsDrawBox(pos[i],R[i],bodyParts[i].proportions);
        }
}
```

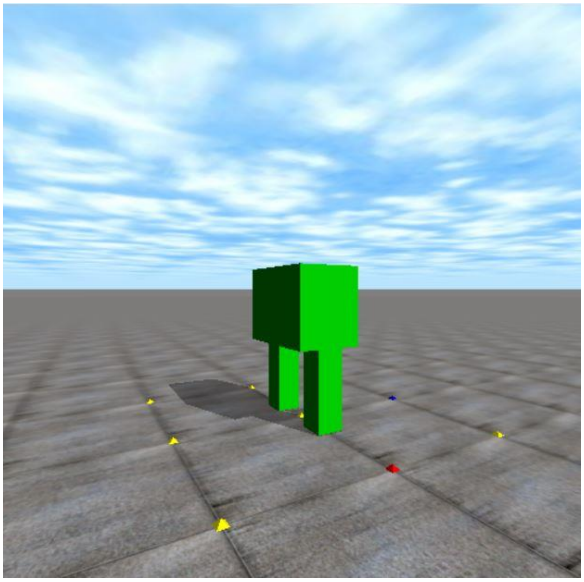
Kod 2.6 – funkcija koja se poziva u svakom koraku simulacije

U svakom koraku simulacije potrebno je na početku izračunati potencijalne sudare tijela s tlom ili tijela međusobno. Korisnik je dužan sam definirati funkciju koja barata sa sudarima, a ovdje je ta funkcija nazvana `nearCallback`. Funkcija prima geometrije koje su se potencijalno sudarile, među njima stvara sudar funkcijom `dCollide` te nad točkama nastalima u sudaru obavlja neke operacije. Ako tijela nisu bila u kontaktu, neće nastati nijedna točka u sudaru. Nakon računanja sudara moguće je napraviti vremenski korak pozivom funkcije `dWorldStep`. Što je vremenski korak manji, to će simulacija sporije ići. Zbog toga je u ovom primjeru postavljen vremenski korak na 0.0004 s (kako bi se kretanje bipeda moglo što bolje i jasnije kontrolirati). Potom se poziva funkcija za ispražnjavanje globalne skupine zglobova korištene pri računanju sudara u funkciji `nearCallback`.

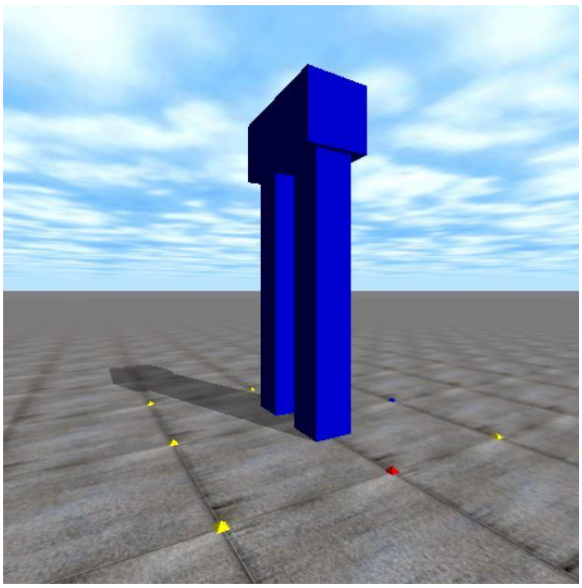
Tek nakon ovih izračuna moguće je prijeći na iscrtavanje objekata u sceni. Ako je prvi biped odabran za prikazivanje, obojit će se u crvenu boju, dok se drugi boji u zelenu, a treći u plavu boju. Nakon toga treba saznati trenutnu poziciju i rotaciju svakog dijela tijela pozivom funkcija `dBodyGetPosition` i `dBodyGetRotation`. Koristeći te podatke i podatke o proporcijama tijela, pojedini dio se iscrtava pozivom funkcije `dsDrawBox`. Rezultat iscrtavanja prvog, drugog i trećeg bipeda prikazan je redom na slikama u nastavku (Slika 2.1, Slika 2.2 i Slika 2.3).



Slika 2.1 – prikaz prvog bipeda



Slika 2.2 – prikaz drugog bipeda



Slika 2.3 – prikaz trećeg bipeda

Kut gledanja prema bipedu definiran je u funkciji `start`:

```
void start() {
      static float xyz[3] = {0.0,-3.0,1.0};
      static float hpr[3] = {90.0,0.0,0.0};
      dsSetViewpoint (xyz,hpr);
}
```

Očište i gledište postavljaju se koristeći funkciju `dsSetViewpoint`. U ovom slučaju, vektor `xyz` predstavlja početno gledište, a vektor `hpr` početno očište. Kao što je već napomenuto, očište i gledište moguće je mijenjati klikom i povlačenjem tipki miša.

2.4. Funkcija za prepoznavanje i obradu događaja

Pri pokretanju simulacije prikazan je samo prvi (crveni) biped. Biped nakon nekog vremena pada na tlo jer ne može održavati ravnotežu bez primjene sila. Međutim, bilo bi korisno kad bi korisnik sam mogao birati koji biped će se trenutno prikazivati. Upravo to implementirano je u funkciji `command`:

```
void command(int cmd) {
	switch (cmd) {
		case 'r':
			restart();
			break;
		case '1':
			bipedId = 1;
			restart();
			break;
		case '2':
			bipedId = 2;
			restart();
			break;
		case '3':
			bipedId = 3;
			restart();
			break;
		//...
	}
}
```

Kod 2.7 – prikaz prvog dijela funkcije `command`

Kad korisnik pritisne neku tipku na tipkovnici, poziva se funkcija `command` za obradu tog događaja. Ako se pritisne tipka r, poziva se funkcija `restart`:

```
void restart() {
	dJointGroupDestroy(contactgroup);
	destroyBiped();
	contactgroup = dJointGroupCreate(0);
	createBiped();
}
```

Kod 2.8 – funkcija sa vraćanje u početno stanje

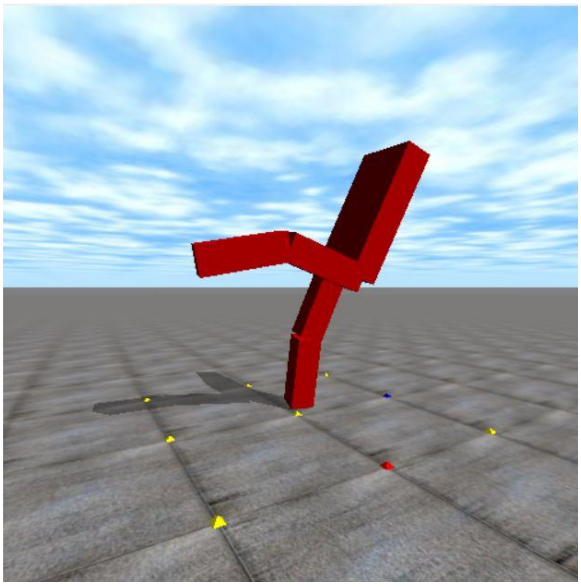
Funkcija uništava trenutnog bipeda pozivom funkcije `destroyBiped` u kojoj se uništava svaki dio tijela i njegova pripadna geometrija, a potom se ponovno stvara novi biped. Pritiskom na tipke 1, 2 ili 3 postavlja se `bipedId` na odgovarajuću vrijednost te se poziva funkcija `restart` koja konačno stvara novog bipeda s novim proporcijama. Time je korisniku omogućeno jednostavno mijenjanje trenutačno simuliranog modela te vraćanje na početno stanje.

Osim mijenjanja trenutno simuliranog modela, korisniku je omogućena i primjena sila na zglobove bipeda:

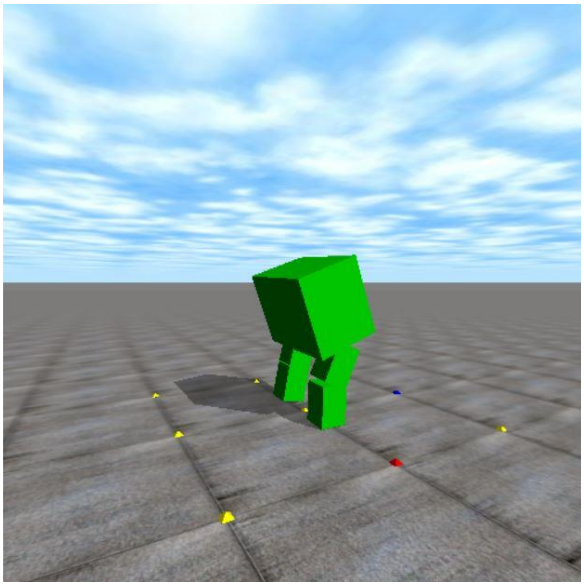
```
void command(int cmd) {
	switch (cmd) {
		//...
		case 'a':
			dJointAddHingeTorque (hipL, 50.0);
			dJointAddHingeTorque (kneeL, -50.0);
			break;
		case 'q':
			dJointAddHingeTorque (hipL, -50.0);
			dJointAddHingeTorque (kneeL, 50.0);
			break;
		case 'd':
			dJointAddHingeTorque (hipR, -50.0);
			dJointAddHingeTorque (kneeR, 50.0);
			break;
		case 'e':
			dJointAddHingeTorque (hipR, -50.0);
			dJointAddHingeTorque (kneeR, 50.0);
			break;
	}
}
```

Kod 2.9 – prikaz drugog dijela funkcije `command`

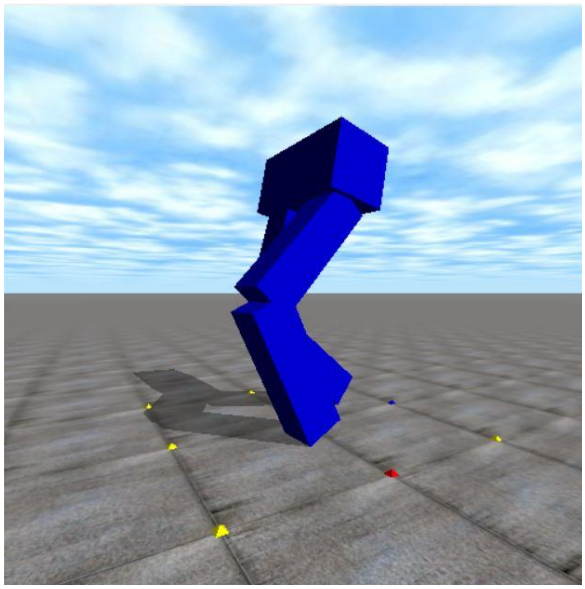
Pritiskom na tipke a, q, d ili e primjenjuju se sile na zglobove bipeda pozivima funkcije `dJointAddHingeTorque`. Te sile namještene su na iznose 50 N i -50 N, ali moguće ih je postaviti i na drugačije vrijednosti te pritom promatrati ponašanje modela. Iako ovakav način primjena sila ne daje najbolje rezultate (biped ubrzo padne), dobra je početna demonstracija utjecaja sila na kretanje bipeda. Bipedi u pokretu nakon primjena sila prikazani su na slikama u nastavku (Slika 2.4, Slika 2.5 i Slika 2.6).



Slika 2.4 – prvi biped u pokretu



Slika 2.5 – drugi biped u pokretu



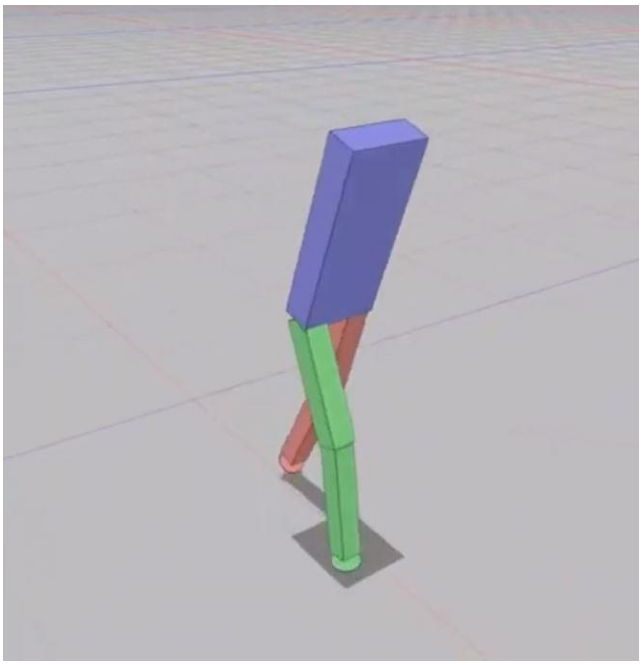
Slika 2.6 – treći biped u pokretu

3. Rezultati

Konačni program korisniku omogućuje izvođenje simulacije modela i kretanja tri bipeda različitih proporcija. Daljnja nadogradnja programa mogla bi uključiti preciznije podešavanje masa dijelova tijela za dobivanje realističnijih rezultata (u ovoj implementaciji svaki dio tijela mase je 1 kg). Također, umjesto zglobova tipa šarke, za kukove je moguće primijeniti i univerzalni zglob. Ljudski kuk može se rotirati po više od jedne osi pa bi univerzalni zglob omogućio dodatni stupanj slobode kretanja.

3.1. Moguće primjene

Iako trenutačni model ne daje impresivne rezultate u pogledu kretanja, moguće ga je primijeniti za razvoj kompleksnijeg rješenja koji bi primjenom sila na zglobove omogućio autonomni uspravni hod bipeda. Jedno od takvih rješenja ponuđeno je u radu *Quadratic Leaky Integrate-and-Fire Neural Network tuned with an Evolution-Strategy for a Simulated 3D Biped Walking Controller* istraživača sa Sveučilišta u Newcastleu [9].



Slika 3.1 – rezultat simulacije rada autora Sveučilišta u Newcastleu

Koristeći neuronsku mrežu te evolucijski algoritam korišten za podešavanje težina sinaptičkih veza, nakon otprilike dvije tisuće generacija biped je dobio mogućnost uspravnog hoda na udaljenosti od nekoliko metara. Biped korišten u tom radu ima sedam dijelova tijela jer su mu dodana stopala u obliku polusfere kako bi u svakom koraku kontakt s tlom bio u samo jednoj točki, čime je olakšan proces učenja hoda u odnosu na stopalo u obliku kvadra. Rezultat rada moguće je vidjeti i na YouTube kanalu jednog od autora [10].

Zaključak

Simulacija bipeda predstavlja problem kojem treba prići sa strpljenjem. Potrebno je pažljivo odrediti proporcije svakog dijela tijela te njihove pozicije. Također, potrebno je dobro odrediti razne parametre zglobova koji će na kraju najviše utjecati na to kako se biped ponaša pri kretanju.

U okviru ovog rada razmotrene su brojne mogućnosti fizikalnog pogona Open Dynamics Engine. Primjena njegovih intuitivnih funkcija uvelike je olakšala proces izrade modela. Posebno pozitivna je činjenica da je ODE biblioteka otvorenog koda te da svatko može na jednostavan način primijeniti i proučavati utjecaje sila na tijelo u dinamičkom okruženju. Osim za izradu bipeda, funkcije ODE-a moguće je koristiti pri modeliranju različitih drugih objekata poput jednostavnih vozila ili robota koji se kreću pomoću više udova.

Implementirani program može se doraditi i primijeniti za naprednije simuliranje kretanja koristeći sile u zglobovima. Moguće je dodati mogućnost primjene sile na svaki zglob posebno, ali i pokušati ostvariti autonomno kretanje modela na način opisan u poglavlju 3.1. Dosad implementirane funkcije predstavljaju dobru osnovu za bilo koju od tih mogućnosti nadogradnje.

Literatura

[1] Boston Dynamics, *Atlas, The Next Generation*, (2016, veljača). Poveznica: https://www.youtube.com/watch?v=rVlhMGQgDkY; pristupljeno 1. lipnja 2020.

[2] Motherboard, *The Two-Legged Robots Walking Into the Future*, (2017, studeni). Poveznica: https://www.youtube.com/watch?v=OEXVGwEfq_E; pristupljeno 1. lipnja 2020.

[3] Smith, R., *Open Dynamics Engine*. Poveznica: https://www.ode.org/; pristupljeno 1. lipnja 2020.

[4] Demura, K., *ODE Tutorials*. Poveznica: https://demura.net/english; pristupljeno 25. travnja 2020.

[5] *Drawstuff*. Poveznica: http://opende.sourceforge.net/docs/group__drawstuff.html; pristupljeno 1. lipnja 2020.

[6] Smith, R., *Open Dynamics Engine v0.039 User Guide*, (2003, lipanj). Poveznica: http://ode.org/ode-0.039-userguide.html#ref20; pristupljeno 25. travnja 2020.

[7] Open Dynamics Engine Community Wiki, *Manual*, (2004). Poveznica: http://ode.org/wiki/index.php?title=Manual#Setting_Joint_Torques.2FForces_Directly; pristupljeno 3. lipnja 2020.

[8] Demura, K., *How to install ODE*. Poveznica: https://demura.net/9ode/6077.html#more-6077; pristupljeno 25. travnja 2020.

[9] Wiklendt, L., Chalup, S. K., Seron, M. M., *Quadratic Leaky Integrate-and-Fire Neural Network tuned with an Evolution-Strategy for a Simulated 3D Biped Walking Controller*, Eighth International Conference on Hybrid Intelligent Systems, Australija, (2008)

[10] Wiklendt, L., *Simulated Biped Walker (2)*. Poveznica: https://www.youtube.com/watch?v=ITGH7ikkCEU; pristupljeno 10. lipnja 2020.

Sažetak

Simulacija dvonožnog modela biped

U okviru ovog rada opisana je primjena fizikalnog pogona Open Dynamics Engine (ODE) za ostvarivanje simuliranog prikaza dvonožnog modela biped. Opisane su značajke ODE-a. Opisane su glavne funkcije ODE-a bitne za ostvarivanje simulacije dvonožnog modela. Implementirana je simulacija tri modela bipeda različitih proporcija. Omogućeno je kretanje bipeda primjenom sila na zglobove. Korisnik može tipkovnicom upravljati trenutno prikazanim modelom te pomicati model primjenom sila na zglobove bipeda. Na kraju su razmotreni rezultati simulacije i moguće nadogradnje programa.

Ključne riječi: fizikalni pogon, Open Dynamics Engine, C++, biped, simulacija kretanja

Summary

Simulation of biped model

This paper describes the application of the physics engine Open Dynamics Engine (ODE) to achieve a simulated representation of a bipedal model. ODE features are described. The main functions of the ODE essential for realizing the simulation of a two-legged model are described. A simulation of three biped models of different proportions was implemented. It is possible to move the biped by applying forces to the joints. The user can control the currently displayed model with the keyboard and move the model by applying forces to the biped joints. Finally, the results of the simulation and possible program upgrades are discussed.

Key words: physics engine, Open Dynamics Engine, C++, biped, motion simulation