SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD 1853.

DETEKCIJA SUDARA ČVRSTIH TIJELA

Tomislav Ostroški

Zagreb, srpanj 2010.

Sadržaj

Popis slika

Popis algoritama

Uvod

Interaktivna računalna grafika je područje koje se ubrzano razvija. Prvobitno je naglasak bio na razvoju što bržih i kvalitetnijih metoda iscrtavanja. Težilo se je kvalitetnijem prikazu umjetne scene. Danas, uz stalni razvoj snage i mogućnosti računalnog sklopovlja, težište istraživanja se pomiče ka što je moguće kvalitetnijem i realističnijem prikazu fizikalnih zakonitosti u interaktivnim aplikacijama. Jedna od najvažnijih implementacija fizikalnih zakona u interaktivnim aplikacijama je dinamika čvrstih tijela.

Pristup izradi simulacije dinamike čvrstih tijela se bitno razlikuje ovisno o ciljevima koje želimo postići samom simulacijom. Pristup opisan o uvom radu za cilj ima postići fizikalno realističnu simulaciju dinamike čvrstih tijela koja će se izvršavati u realnom vremenu.

U uvodnom razmatranju će biti riječi o fizikalnim veličinama i zakonitostima vezanima uz dinamiku krutih tijela. Zatim će biti opisani razni parametri koji moraju biti uzeti u obzir prilikom izrade simulacije dinamike čvrstih tijela te način na koji je čvrsto tijelo prikazano u računalnom programu.

U trećem poglavlju će biti prezentirana metoda koja se koristi za numeričku integraciju varijabli čvrstog tijela. Četvrto poglavlje razmatra metode za detekciju sudara između dva čvrsta tijela dok se peto poglavlje bavi razrješavanjem tih sudara.

U šestom poglavlju će biti prezentirana metoda za izradu konveksnog omotača. Također će biti prezentiran GJK algoritam za detekciju sudara dvaju konveksnih objekata. Biti će opisan vektorski pristup implementaciji GJK algoritma koji je doveo do veće popularnosti ove metode.

Na kraju rada će biti data usporedba GJK algoritma sa standardnim metodama detekcije sudara.

Poglavlje 1

Dinamika čvrstog tijela

Dinamika čvrstih tijela je područje fizike koje se bavi proučavanjem gibanja čvrstih tijela. Čvrsta tijela su tijela koja ne mijenjaju svoj oblik pod utjecajem vanjskih sila. To jest, udaljenost između bilo kojih dviju točaka čvrstog tijela je konstanta bez obzira na utjecaj vanjskih sila. Za čvrsta tijela se smatra da imaju konstantnu gustoću. Gibanje čvrstog tijela možemo podijeliti u linearno gibanje i u kutno ili rotacijsko gibanje.

1.1 Linearno gibanje čvrstog tijela

Linearno gibanje čvrstog tijela se može promatrati kao gibanje čestice čija je pozicija jednaka poziciji centra mase, a masa jednaka masi čvrstog tijela. Centar mase čvrstog tijela u skladu sa [Kulišić02] definiramo kao:

$$\vec{r}_{CM} = \frac{\sum_{i=0}^{n} m_i \vec{r}_i}{m} \tag{1.1}$$

gdje je m masa čvrstog tijela, a $\vec{r}_i$ i m_i radijus vektor svake čestice tijela te masa te čestice. U nastavku ovog rada, poziciju centra mase ćemo označavati sa $\vec{p}$ te ćemo tu točku uzimati kao referentnu prilikom promatranja tijela. To znači da će ishodište lokalnog koordinatnog sustava tijela biti u centru mase.

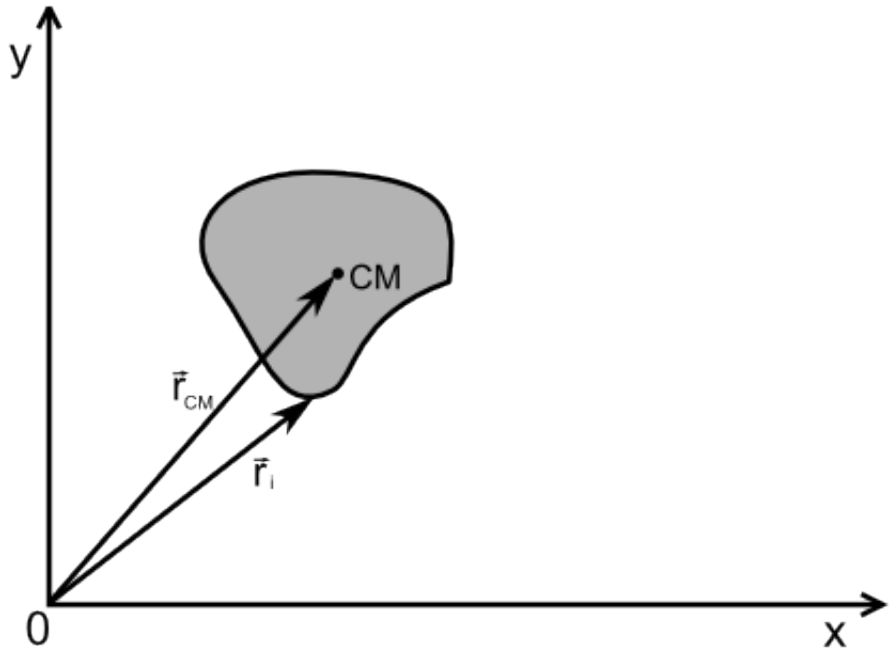
Brzina centra mase je prva derivacija pozicije po vremenu t.

$$\vec{v}\,(t) = \frac{\partial}{\partial t}\vec{p}\,(t) \tag{1.2}$$

Ukupan zbroj svih sila koje djeluju na tijelo u centru mase označavamo sa F. Ukupna sila F uzrokuje ukupnu akceleraciju tijela a.

$$\vec{F} = \vec{F}_1 + \vec{F}_2 + \ldots = \sum_i \vec{F}_i$$

$$\vec{F} = m \cdot \vec{a} \tag{1.3}$$



Slika 1.1: Centar mase čvrstog tijela

Akceleracija je prva derivacija brzine po vremenu što nas u skladu sa (1.2)dovodi do:

$$\begin{aligned} \vec{a}\,(t) &= \frac{\partial}{\partial t}\vec{v}\,(t) \\ \vec{a}\,(t) &= \frac{\partial^2}{\partial t^2}\vec{p}\,(t) \end{aligned} \tag{1.4}$$

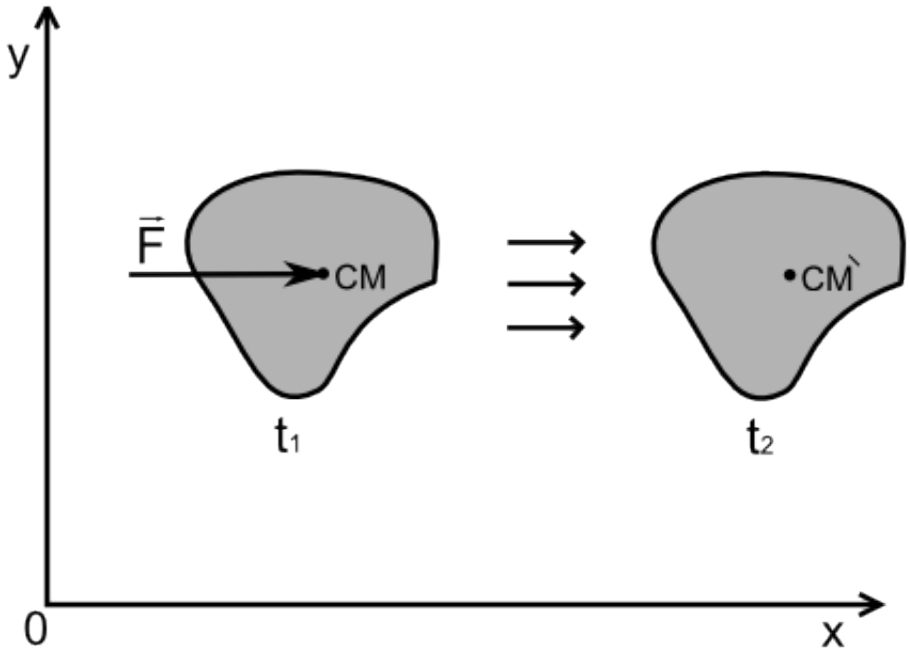
Količina gibanja čvrstog tijela ili linearni moment P je umnožak mase tijela i brzine centra mase.

$$\vec{P} = m \cdot \vec{v}\,(t) \tag{1.5}$$

iz (1.3) i (1.4) dobivamo odnos između sile na tijelo i linearnog momenta tijela:

$$\begin{aligned} \frac{\partial}{\partial t}\vec{P}\,(t) &= m \cdot \frac{\partial}{\partial t}\vec{v}\,(t) \\ \frac{\partial}{\partial t}\vec{P}\,(t) &= m \cdot \vec{a}\,(t) \\ \vec{F}\,(t) &= \frac{\partial}{\partial t}\vec{P}\,(t) \end{aligned} \tag{1.6}$$

Primjena linearne sile na čvrsto tijelo uzrokuje translaciju tijela kao što je prikazano na slici 1.2. Sila koja djeluje na tijelo se primjenjuje u centru mase.



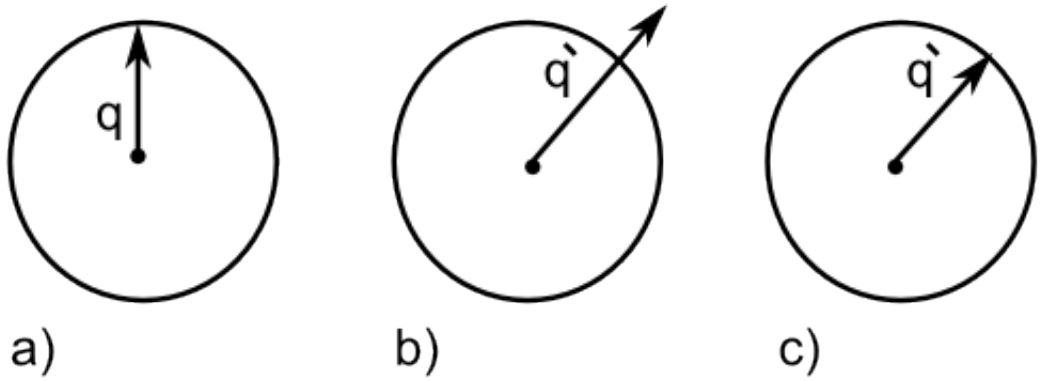
Slika 1.2: Translacija čvrstog tijela

1.2 Kutno gibanje čvrstog tijela

Kutno, odnosno rotacijsko gibanje, je uvelike analogno linearnom gibanju čvrstog tijela. Za sve linearne veličine imamo njihove odgovarajuće kutne vrijednosti. Kutni ekvivalent pozicije tijela je orijentacija. Orijentacija tijela je veličina koja opisuje kako je tijelo poravnato sa prostorom u kojem se nalazi. Orijentaciju tijela definiramo u odnosu na globalni koordinatni sustav. Rotacija tijela je iznos promjene između početne orijentacije i trenutne orijentacije tijela.

Reprezentacija orijentacije

Postoji više načina na koji možemo prikazati orijentaciju tijela. Većina starijih radova iz ovog područja za prikaz orijentacije koristi Eulerove kuteve ili rotacijsku matricu [Baraff01, Hecker96]. Rotacijska matrica je bila preferirani izbor sve do 1985. godine kada Ken Shoemake u svom radu [Shoemake85] prikazuje upotrebu kvaterniona u računalnoj grafici. Osnovni razlog zbog kojeg je Shoemake koristio kvaternione je bila lakša interpolacija između dvaju kvaterniona nego između dviju rotacijskih matrica. No, kvaternioni imaju puno više prednosti [Bergen03]. Naime, rotacijska matrica R sadrži devet realnih brojeva što nam daje devet stupnjeva slobode. Za prikaz orijentacije čvrstog tijela nam trebaju samo tri stupnja slobode. Problem viška stupnjeva slobode rješavamo tako što uvodimo ograničenje kojim zahtijevamo da rotacijska matrica bude ortogonalizirana. To jest, da njena determinanta bude jedan [Bishop04]. Ovo ograničenje broj stupnjeva slobode smanjuje sa devet na potrebna tri.



Slika 1.3: Prikaz orijentacije pomoču kvaterniona
a)Početni jedinični kvaternion b)Isti kvaternion nakon nekoliko računskih operacija više nema jediničnu duljinu pa ne predstavlja orijentaciju c) Nakon postupka normalizacije kvaternion ponovno predstavlja orijentaciju.

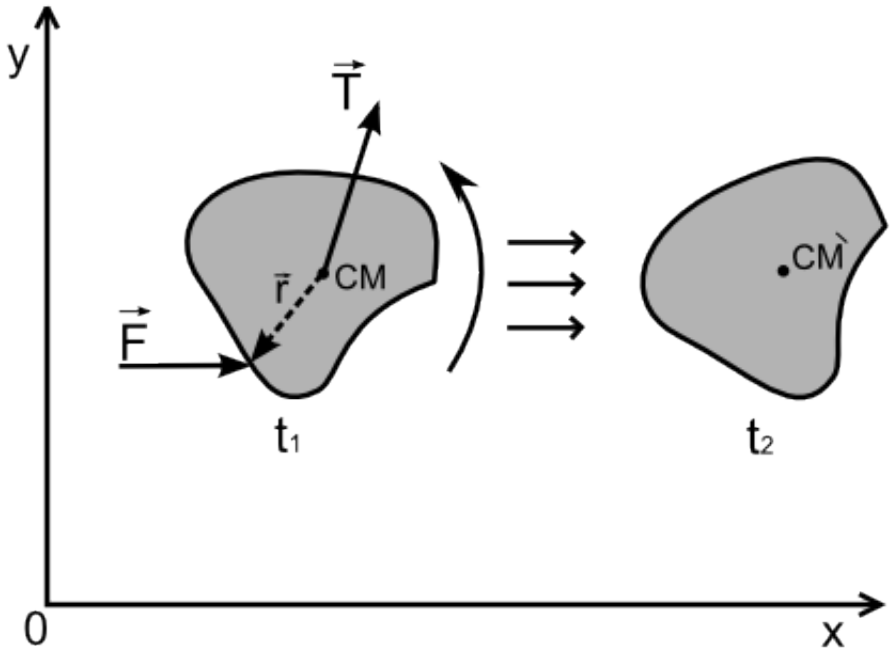
Kvaternion $q\,(s,v)$ se sastoji od četiri skalarne vrijednosti. Jednog skalara s i jednog trodimenzionalnog vektora v.

$$q\,(s,v) = \left(s, x\vec{i} + y\vec{j} + z\vec{k}\right) \tag{1.7}$$

Većina literature opisuje kvaternion kao točku na 4-dimenzionalnoj sferi. Ovaj prikaz je često uzrok težem razumijevanju upotrebe kvaterniona. Pošto kvaternion ima četiri stupnja slobode, potrebna tri stupnja slobode ćemo dobiti tako da uvedemo ograničenje po kojem kvaternion mora biti jedinične duljine. Sada kada jedinični kvaternion posjeduje tri stupnja slobode možemo ga zamisliti kao vektor od središta 3-dimenzionalne kugle do njene površine. Kako pri računanju sa brojevima sa pomičnom točkom dolazi do akumulirane greške, moramo redovito normalizirati kvaternion kako bi on i dalje predstavljao orijentaciju čvrstog tijela kao što je prikazano na slici 1.3. Postupak normalizacije kvaterniona zahtijeva manje računskih operacija od postupka ortogonalizacije rotacijske matrice. To je glavni razlog zbog kojeg ćemo orijentaciju čvrstog tijela prikazivati pomoću jediničnog kvaterniona.

Moment sile

Na slici 1.2 vidimo kako sila koja se primjenjuje na centar mase uzrokuje linearno pomicanje čvrstog tijela, to jest translaciju tijela. No, što ako primijenjena sila nema hvatište u centru mase? U tom slučaju će primijenjena sila $\vec{F}$ uzrokovati ne samo translaciju već i rotaciju tijela. Svojstvo sile da uzrokuje rotaciju tijela oko neke osi nazivamo moment sile i označavamo sa $\vec{M}$. Odnos sile $\vec{F}$ i momenta sile $\vec{M}$ dan je preko izraza:



Slika 1.4: Rotacija krutog tijela izazvana silom F

$$\vec{M} = \vec{r} \otimes \vec{F} \tag{1.8}$$

gdje je $\vec{r}$ vektor od centra mase do točke u kojoj se primjenjuje sila $\vec{F}$. Os rotacije prolazi centrom mase i okomita je na ravninu definiranu vektorom $\vec{r}$ i silom $\vec{F}$ kao što se vidi na slici 1.4.

Kutna brzina i akceleracija

Ukoliko fiksiramo centar mase tako da se ne može micati, jedino gibanje čvrstog tijela će biti ono nastalo rotacijom. Pošto se centar mase ne može micati, os te rotacije će prolaziti kroz njega. Ovo vrtnju opisujemo vektorom $\vec{\omega}(t)$. Smjer ovog vektora je smjer osi rotacije, a iznos $|\omega|$ je brzina rotacije u radijanima u sekundi. Veličina koju označavamo vektorom $\vec{\omega}(t)$ se zove kutna brzina. Iz (1.2) vidimo odnos linearne brzine i pozicije centra mase čvrstog tijela. Moramo pronaći odnos kutne brzine i orijentacije tijela. Taj odnos je dan izrazom: [Millington07]

$$q_1 = q_0 + \frac{\triangle t}{2} \varpi q_0 \tag{1.9}$$

gdje je q_1 rezultirajuča orijentacija nakon vremena t, a $q_0$početna orijentacija. Izrazom ϖoznačavamo kvaternion dobiven iz vektora kutne brzine $\vec{\omega}$ koji smo zatim normalizirali kako bi predstavljao orijentaciju. Kvaternion ϖ se iz vektora $\vec{\omega} = x\vec{i} + y\vec{j} + z\vec{k}$ gradi pomoću izraza:

$$\varpi = \left(0, x\vec{i} + y\vec{j} + z\vec{k}\right) \tag{1.10}$$

Kutna akceleracija čvrstog tijela $\vec{\alpha}$ je prva derivacija kutne brzine po vremenu t i dana je izrazom:

$$\vec{\alpha}\,(t) = \frac{\partial}{\partial t}\vec{\omega}\,(t)$$

Moment inercije

Moment inercije ili moment tromosti je kutni ekvivalent mase. Kao što je masa veličina kojom se opisuje opiranje tijela promjeni linearnog gibanja, tako je moment inercije veličina koja opisuje opiranje tijela promjeni kutnog gibanja. Moment inercije opisuje kako je raspoređena masa čvrstog tijela u odnosu na njegov centar mase. Analogno (1.3) za kutno gibanje imamo izraz:

$$\vec{M}\,(t) = I \cdot \frac{\partial}{\partial t}\vec{\omega}\,(t) \tag{1.11}$$

gdje I označava moment inercije. Izraz (1.11) možemo pisati u matričnom obliku.

$$\begin{bmatrix} \vec{M}_x \\ \vec{M}_y \\ \vec{M}_z \end{bmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial t}\vec{\omega}_x \\ \frac{\partial}{\partial t}\vec{\omega}_y \\ \frac{\partial}{\partial t}\vec{\omega}_z \end{bmatrix} \tag{1.12}$$

Pomnožimo li prvi redak izraza (1.12) dobivamo:

$$\vec{M}_x = I_{xx} \cdot \frac{\partial}{\partial t}\vec{\omega}_x = I_{xx} \cdot \vec{\alpha}_x \tag{1.13}$$

što je vrlo slično izrazu (1.3). Izraz (1.13) nam govori u kakvom su odnosu moment sile oko osi x, moment inercije oko osi x i kutna akceleracija oko osi x. Matrica momenta inercije nam govori u kakvom su odnosu moment sile i kutna akceleracija za cijeli prostor u kojem se tijelo nalazi. Elementi matrice momenta inercije se računaju pomoću sljedećih izraza:

$$I_{xx} = m \int_V \left(y^2 + z^2\right) dV \tag{1.14}$$

$$I_{xy} = -m \int_V xy dV \tag{1.15}$$

gdje je m masa čvrstog tijela, a V njegov volumen. Za tijela koja su simetrična s obzirom na koordinatne osi, nedijagonalni elementi matrice momenta inercije su jednaki nuli.

Moment količine gibanja

Moment količine gibanja $\vec{L}$ je kutni ekvivalent količine gibanja te se analogno (1.5) i (1.6) izražava sa:

$$\vec{L} = I \cdot \vec{\omega} \tag{1.16}$$

$$\vec{M}(t) = \frac{\partial}{\partial t}\vec{L}(t) \tag{1.17}$$

Rečeno je da moment inercije opisuje kako je raspoređena masa čvrstog tijela u odnosu na njegov centar mase. Ukoliko se taj raspored promjeni, doći će i do promjene momenta inercije. U takvom slučaju će se promijeniti kutna brzina tijela, ali ne i njegov moment količine gibanja. Ovo je važno svojstvo koje kaže da bez utjecaja vanjskih sila, moment količine gibanja čvrstog tijela ima konstantnu vrijednost.

1.3 Ukupno gibanje čvrstog tijela

Povezanost kutnog i linearnog gibanja je vidljiva iz izraza kojim se opisuje brzina točke A na čvrstom tijelu:

$$\vec{v}_A(t) = \vec{v}(t) + \vec{\omega} \otimes (\vec{p}_A(t) - \vec{p}(t)) \tag{1.18}$$

gdje je $\vec{p}_A(t)$ pozicija točke A u trenutku t .

Tablica 1.1 daje pregled fizikalnih veličina kojima opisujemo gibanje čvrstog tijela.

fizikalna veličina	linearno gibanje	kutno gibanje	fizikalna veličina
pozicija	$\vec{p}$	q	orijentacija
brzina	$\vec{v}$	$\vec{\omega}$	kutna brzina
akceleracija	$\vec{a}$	$\vec{\alpha}$	kutna akceleracija
masa	m	I	moment inercije
sila	$\vec{F}$	$\vec{M}$	moment sile
količina gibanja	$\vec{P}$	$\vec{L}$	moment količine gibanja

Tablica 1.1: Veličine kojima opisujemo gibanje čvrstog tijela

Poglavlje 2

Simuliranje dinamike čvrstih tijela

2.1 Računanje svojstava mase

Da bismo mogli prikazati dinamiku nekog čvrstog tijela moramo znati konstantne vrijednosti koje pripadaju tom tijelu. To su masa tijela, moment inercije i centar mase. Računalni trodimenzionalni modeli objekata čije gibanje želimo simulirati sadrže samo podatke o geometriji objekta. Ti su podatci spremljeni u obliku mreže poligona(*polygon mesh*). Nekoliko radova se bavi računanjem masenih svojstava tijela iz mreže poligona. [Welzl91, O'Rourke98, Eberly01] Najprimjereniji postupak ovoj simulaciji je onaj opisan u [Eberly01]. Prednost ovog postupka u odnosu na [O'Rourke98] je u tome što ovaj postupak radi sa poligonima u obliku trokuta, a ne sa poligonima u obliku četverokuta.

Da bismo izračunali masu tijela, njegov centar mase i moment inercije, moramo riješiti integrale koji imaju oblik:

$$\int_V p\,(x,y,z)\,dV \tag{2.1}$$

gdje je V volumen integracije, a dV beskonačno mali dio tog volumena. Funkcija $p\,(x,y,z)$ je polinom oblika 1, x, y, z, x^2, y^2, z^2, xy, yz ili zx. U naše slučaju V je dio prostora ograničen površinom čvrstog tijela. Navedeni rad se temelji na primjeni teorema o divergenciji [Apsen69] kako bi se volumni integral pretvorio u plošni integral po površini čvrstog tijela koja je reprezentirana mrežom poligona. Na ovaj način možemo izračunati konstantne vrijednosti čvrstog tijela iz trodimenzionalnog računalnog modela kojim je ono reprezentirano.

2.2 Odabir vrste simulacije

Prije nego pristupimo izradi same simulacije moramo dobro razmisliti što želimo simulirati i koji su ciljevi simulacije. O tome će uvelike ovisiti koje metode ćemo koristiti pri izradi same simulacije.

Realnost ili realističnost

Kada govorimo o fizikalnim simulacijama moramo razlikovati fizikalnu realnost od fizikalne realističnosti. O fizikalnoj realnosti govorimo kada simulacija daje jednake rezultate kakve bi dao i pokus u stvarnom svijetu. Uvjet koji zahtijevamo od fizikalno realnih simulacija je što je moguće veća sličnost između stvarnih rezultata i simuliranih rezultata. Potreba za ovakvim simulacijama se javlja pri izradi različitih simulatora kao što su simulatori leta.

U interaktivnim aplikacijama, kao što su video igre, naglasak je stavljen na kvalitetan grafički prikaz. Zbog toga je fizikalnoj simulaciji ostavljen ograničen vremenski prostor za izvršavanje. Uvjet koji postavljamo pred ovakve fizikalne simulacije je uvjet realističnosti. To znači da simulacija mora dati rezultate koji se doimaju fizikalno realnima, ali to ne moraju doista i biti. Ovo je pristup koji ćemo koristiti u izradi ove simulacije.

Broj objekata

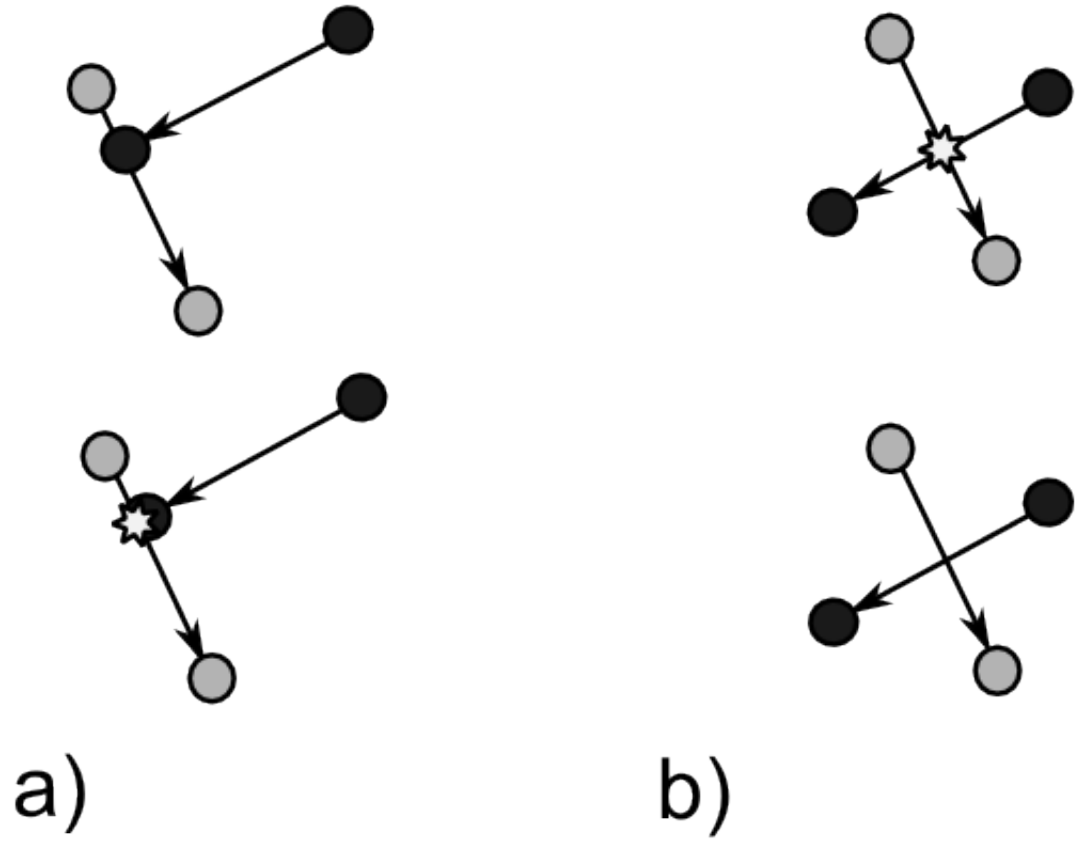
O broju i obliku objekata u sceni ovisi kakve ćemo metode sudara koristiti. Ako imamo velik broj objekata u sceni, provjeravanje sudara svakog objekta sa svim ostalima u sceni oduzima puno vremena. Pristup rješavanju ovog problema je izgradnji posebnim struktura podataka kojima se dijeli prostor simulacije. Tada se radi međusobnog sudara provjeravaju samo objekti koji se nalaze u istom dijelu prostora simulacije. Ovaj pristup detekciji sudara se naziva široka faza (*broad phase*). Različite strukture podataka koje se koriste za široku fazu detekcije sudara su detaljno opisane u [Ericson05] i [Bergen03].

U ovom radu će biti pobliže opisane metode detekcije sudara koje se vrše na paru objekata. U takozvanoj uskoj fazi (*narrow phase*) detekcije sudara. Uska faza je faza detekcije sudara koja prosljeđuje potrebne podatke o sudaru dijelu simulacije koja razrješava sudare. Zbog toga će naglasak u ovom radu biti stavljen upravo na detekciju sudara između dvaju objekata.

Diskretno ili kontinuirano kretanje

Parametar koji može uvelike utjecati na rezultate detekcije sudara, a samim time i na rezultate same simulacije jest kreću li se objekti koje promatramo u trenutku sudara ili su stacionarni. Detekcija sudara dvaju objekata koji se kreću mora u obzir uzeti mnogo faktora. Ukoliko dva objekta nisu trenutno u sudaru ne znači da neće biti u sudaru do kraja vremenskog odsječka za koji vršimo simuliranje. Moramo uzeti u obzir brzinu kojom se tijela kreću te smjer u kojem se kreću. Mogućnost da tijela rotiraju dodatno komplicira provjeru detekcije sudara. Način na koji se vrši detekcija sudara u ovom slučaju je da se tijela izduže u pravcu kretanja za duljinu puta koju bi prevalili u promatranom vremenskom odsječku. Problem nastaje u situaciji kada to kretanje nije pravocrtno. Na primjer, prilikom kosog hica.

U ovoj simulaciji ćemo primijeniti drugačiji pristup. Na početku svakog vremenskog odsječka ćemo tijela promatrati kao stacionarna. To jest, nećemo uzimati u obzir smjer i



Slika 2.1: Istovremeno i pojedinačno gibanje dvaju objekata
a) gornja slika prikazuje istovremeno kretanje bez sudara dok pojedinačno kretanje na donjoj slici uzrokuje sudar
b) gornja slika prikazuje istovremeno kretanje sa sudarom dok pojedinačno kretanje na donjoj slici ne uzrokuje sudar

brzinu njihovog kretanja prilikom detekcije sudara. Ova pretpostavka neće bitno utjecati na rezultate simulacije ako korak simulacije bude mali. U tom slučaju se radi o velikoj sličnosti trenutaka iscrtavanja (*frame coherence*).

Istovremeno ili pojedinačno kretanje

Kada se u sceni nalazi više objekata, moramo odlučiti hoćemo li simulaciju vršiti za sve objekte istovremeno ili ćemo simulirati jedan po jedan objekt. Istovremeno integriranje stanja svih objekata ima prednost što daje fizički realnije rezultate simulacije. Naime, dva objekta koja se inače ne bi sudarila, mogu se sudariti ukoliko ih simuliramo pojedinačnim kretanjem. Primjer za to se vidi na slici 2.1.

Prednosti pojedinačnog kretanja objekata se vide u slučaju kada dođe do sudara. Tada se može poništiti kretanje objekta koji je uzrokovao sudar. Poništiti kretanje svih objekata kada se detektira sudar bi bilo puno teže za istovremeno kretanje. Problemi uzrokovani pojedinačnim kretanjem se mogu ignorirati u interaktivnim aplikacijama jer one u pravilu imaju jako mali korak simulacije [Ericson05].

Dijeljena ili posebna struktura podataka za detekciju sudara

Objekti čije gibanje simuliramo su već reprezentirani u memoriji računala radi potreba iscrtavanja. Čini se ispravnim koristiti te strukture podataka i za detekciju sudara. Ipak, to bi bio pogrešan pristup. Strukture podataka potrebne za iscrtavanje objekata na zaslonu računala sadrže puno podataka sasvim nepotrebnih za detekciju sudara kao što su koordinate tekstura i podatci o svojstvima materijala.

Stvaranje vlastitih struktura podataka za detekciju sudara često dovodi do dupliciranja jednih te istih podataka. No, danas kada se gotovo svi podatci potrebni za iscrtavanje nalaze u memoriji grafičke kartice, dohvat potrebnih podataka za detekciju sudara bi uzrokovalo usporavanje izvršavanja aplikacije.

2.3 Stanje čvrstog tijela

Stanje u kojem se nalazi čvrsto tijelo je opisano vrijednostima njegovih fizikalnih veličina. Fizikalne veličine čvrstog tijela dijelimo na konstantne te primarne i sekundarne veličine.

konstantne veličine masa, moment inercije i centar mase

primarne veličine pozicija, orijentacija, količina gibanja i moment količine gibanja

sekundarne veličine brzina, kutna brzina i spin

Spin je prva derivacija orijentacije po vremenu i računa se po formuli [Baraff01]:

$$s = \frac{\triangle t}{2} \varpi q \tag{2.2}$$

gdje s označava spin, $\triangle t$ korak simulacije, q orijentaciju čvrstog tijela, a ϖ kvaternion kutne brzine dobiven u skladu sa (1.10). Tako da stanje čvrstog tijela u trenutku t možemo definirati izrazom:

$$X\left(t\right) = \begin{pmatrix} \vec{p}\left(t\right) \\ q\left(t\right) \\ \vec{P}\left(t\right) \\ \vec{L}\left(t\right) \end{pmatrix} \tag{2.3}$$

Čvrsto tijelo prelazi iz jednog stanja u drugo promjenom vrijednosti njegovih primarnih fizikalnih veličina. Promjena se događa kroz tijek simulacije.

2.4 Tijek simulacije

Simulaciju možemo podijeliti na više dijelova. To su:

inicijalizacija U ovoj fazi inicijaliziramo sve strukture podataka potrebne za izvršavanje same simulacije. Ova se faza izvršava prije nego započne samo simuliranje.

proračun sila U ovom koraku simulacije izračunavamo vanjske sile koje djeluju na tijela.

integracija U ovoj fazi integriramo fizikalne veličine kao što su brzina i akceleracija te izračunavamo nove vrijednosti pozicije i orijentacije tijela.

detekcija sudara Provjeravamo postoje li sudari između čvrstih tijela.

razrješavanje sudara Ukoliko smo u prethodnoj fazi detektirali sudare, sada ih razrješavamo.

Algorithm 2.1 Tijek izvođenja simulacije

```
Inicijalizacija:
    odredi konstantne veličine
    odredi početne vrijednosti primarnih i sekundarnih veličina
Simulacija:
DOK (vrijeme_simulacije > 0)
    izračunaj i primjeni vanjske sile
    integriraj fizikalne veličine
    osvježi stanje čvrstog tijela
    detektiraj sudare
    razriješi sudare
    vrijeme_simulacije = vrijeme_simulacije - korak_simulacije
```

Poglavlje 3

Integracija

Za svaki vremenski korak simulacije, moramo izračunati nove vrijednosti fizikalnih veličina čvrstog tijela. Postupak izračunavanja novog stanja čvrstog tijela iz prethodnog za proteklo vrijeme se naziva integracija.

U skladu sa izrazima (1.2), (2.2), (1.6) i (1.17) možemo definirati derivaciju stanja čvrstog tijela kao:

$$\frac{\partial}{\partial t}X\left(t\right)=\frac{\partial}{\partial t}\left(\begin{array}{c}\vec{p}\left(t\right)\\ q\left(t\right)\\ \vec{P}\left(t\right)\\ \vec{L}\left(t\right)\end{array}\right)=\left(\begin{array}{c}\vec{v}\left(t\right)\\ s\left(t\right)\\ \vec{F}\left(t\right)\\ \vec{M}\left(t\right)\end{array}\right) \tag{3.1}$$

Jedna od najjednostavnijih metoda integracije je Eulerov postupak. Nažalost ova metoda nije dovoljno precizna za naše potrebe [Hecker96]. Metoda koju ćemo koristiti je metoda Runge-Kutta četvrtog reda ili skraćeno RK4 [Weisstein]. Ova metoda koristi međukorake kako bi poništila greške nižeg reda. Ako početni problem definiramo sa:

$$y'=f\left(t,y\right) \tag{3.2}$$

$$y\left(t_0\right)=y_0 \tag{3.3}$$

tada je rješenje tog problema upotrebom RK4 metode dano sa:

$$y_{n+1}=y_n+\frac{1}{6}h\left(k_1+2k_2+2k_3+k_4\right) \tag{3.4}$$

$$t_{n+1}=t_n+h \tag{3.5}$$

gdje y_{n+1} označava RK4 aproksimaciju za $y(t_{n+1})$, h vremenski interval, a

$$k_1=f\left(t_n,y_n\right) \tag{3.6}$$

$$k_2 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \tag{3.7}$$

$$k_3 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right) \tag{3.8}$$

$$k_4 = f\left(t_n + h, y_n + hk_3\right) \tag{3.9}$$

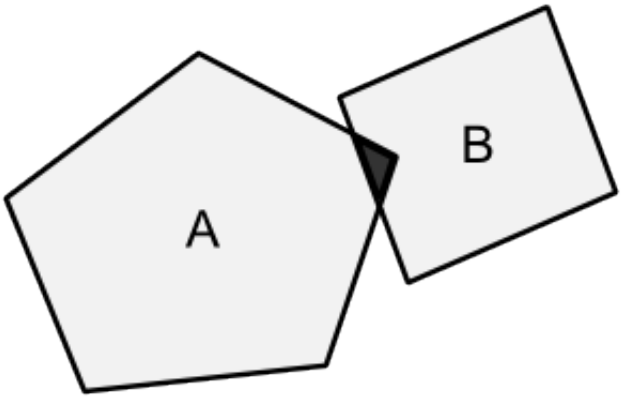
RK4 izračunava sljedeću vrijednost $y\left(t_{n+1}\right)$ iz trenutne vrijednosti $y\left(t\right)$ te umnoška vremenskog intervala h i procijenjenog nagiba krivulje. Nagib krivulje na početku intervala je k_1, k_2 i k_3 su nagibi na sredini intervala, a k_4 je nagib krivulje na kraju intervala. Veća važnost je dana nagibima na sredini intervala. Ukupna akumulirana greška ove metode je reda h^4.

Poglavlje 4

Detekcija sudara

Zadaća sustava za detekciju sudara je u prvom redu odrediti je li došlo do sudara i ako jest proslijediti potrebne podatke sustavu za razrješenje sudara. Sustav za detekciju sudara se sastoji od dva dijela, provjere sudara i generiranja kontakta.

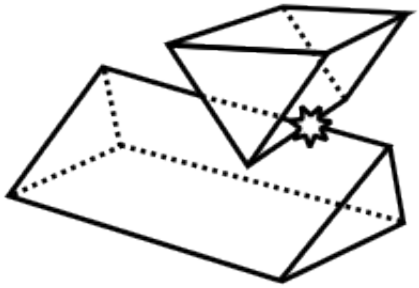
4.1 Provjera sudara



Slika 4.1: Dva objekta u sudaru

Dva su objekta u sudara ukoliko je njihova udaljenost manja od nule, to jest ukoliko postoji dio prostora koji je zajednički i jednom i drugom objektu. Standardni način da se to uradi jest da provjerimo je li koji vrh objekta A sadržan unutar objekta B i obratno. Ovaj pristup se u biti svodi na provjeru jeli zadana točka unutar objekta. Provjeru sudara možemo vršiti na različite načine, ovisno o tome kakve objekte provjeravamo.

Pristup koji se može primijeniti i na konveksne i na konkavne objekte jest da iz točke koju provjeravamo pustimo zraku, najčešće paralelnu sa osi X te provjeravamo koliko trokuta objekta je ta zraka presjekla. Ukoliko je broj neparan, točka leži unutar objekta. Ova metoda nije uvijek pouzdana. Razlog za to leži u mogućnosti da testna zraka prođe kroz brid trokuta, ili kroz vrh objekta. U tom slučaju, zbog nepreciznosti brojeva sa pomičnom točkom, možda nećemo detektirati sudar iako on postoji.



Slika 4.2: Sudar dvaju bridova objekata

Najjednostavniji slučaj jest kada je objekt zadan kao konveksna mreža poligona čije normale pokazuju van tijela. Tada provjeru je li točka unutar objekta vršimo tako da provjeravamo je li točka iza poluravnine svakog poligona. Ukoliko to jest slučaj, točka se nalazi unutar objekta.

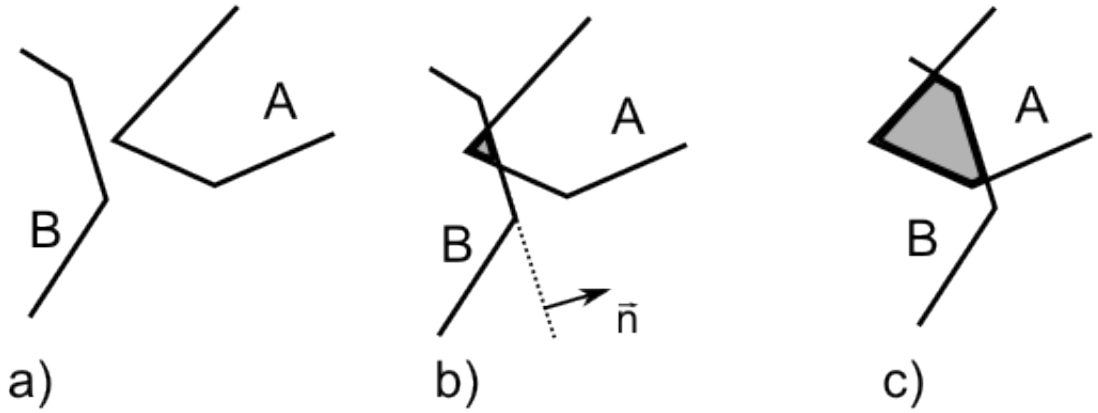
Gornje dvije provjere neće detektirati sudar dvaju bridova objekata, slika 4.2. Ukoliko je broj poligona objekta velik, taj slučaj možemo zanemariti. U protivnom, moramo provjeriti i sudar tipa brid-brid.

Algorithm 4.1 Točka unutar objekta danog kao presjek poluravnina

Zadane su točka *P* i polje ravnina *R*
ZA SVAKU ravninu *r* iz *R*
    d = udaljenost između točke *P* i ravnine *r*
    AKO(*d* > 0) točka *P* je izvan objekta
točka *P* je unutar objekta

4.2 Generiranje kontakta

Dva su tijela u kontaktu kada je njihova udaljenost točno nula. Kada provjera sudara javi da je došlo do prodora jednog objekta u drugi, moramo generirati kontakt za ta dva tijela. Generiranje kontakta podrazumijeva stvaranje skupa podataka koji sadrži točku u kojoj je došlo do sudara i normalu sudara. Normala sudara je po konvenciji vektor usmjeren od tijela B prema tijelu A. Često u taj skup podataka moramo uključiti i podatak o dubini penetracije. Ukoliko je vremenski okvir za koji vršimo simulaciju mali, dubinu penetracije možemo ignorirati jer je jako mala. Tada ćemo reći da simulacija ima veliku sličnost između dvaju vremenskih okvira (*frame coherence*) . To znači da se objekti jako malo pomiču u prostoru za zadani vremenski okvir. Za svaki sudar moramo generirati samo jedan skup podataka makar za promatrani sudar može biti više točaka kontakta. Ukoliko je sličnost između vremenskih okvira velika, proces generiranja kontakta će biti



Slika 4.3: Utjecaj sličnosti između vremenskih okvira na detekciju sudara
a) Objekti prije sudara. b) Velika sličnost rezultira jednom točkom sudara i jasnom normalom sudara n. c) Mala sličnost rezultira sa više točaka sudara i nejasnom normalom sudara n.

jednostavniji. Pri maloj sličnosti između vremenskih okvira, često je problem odrediti normalu sudara.

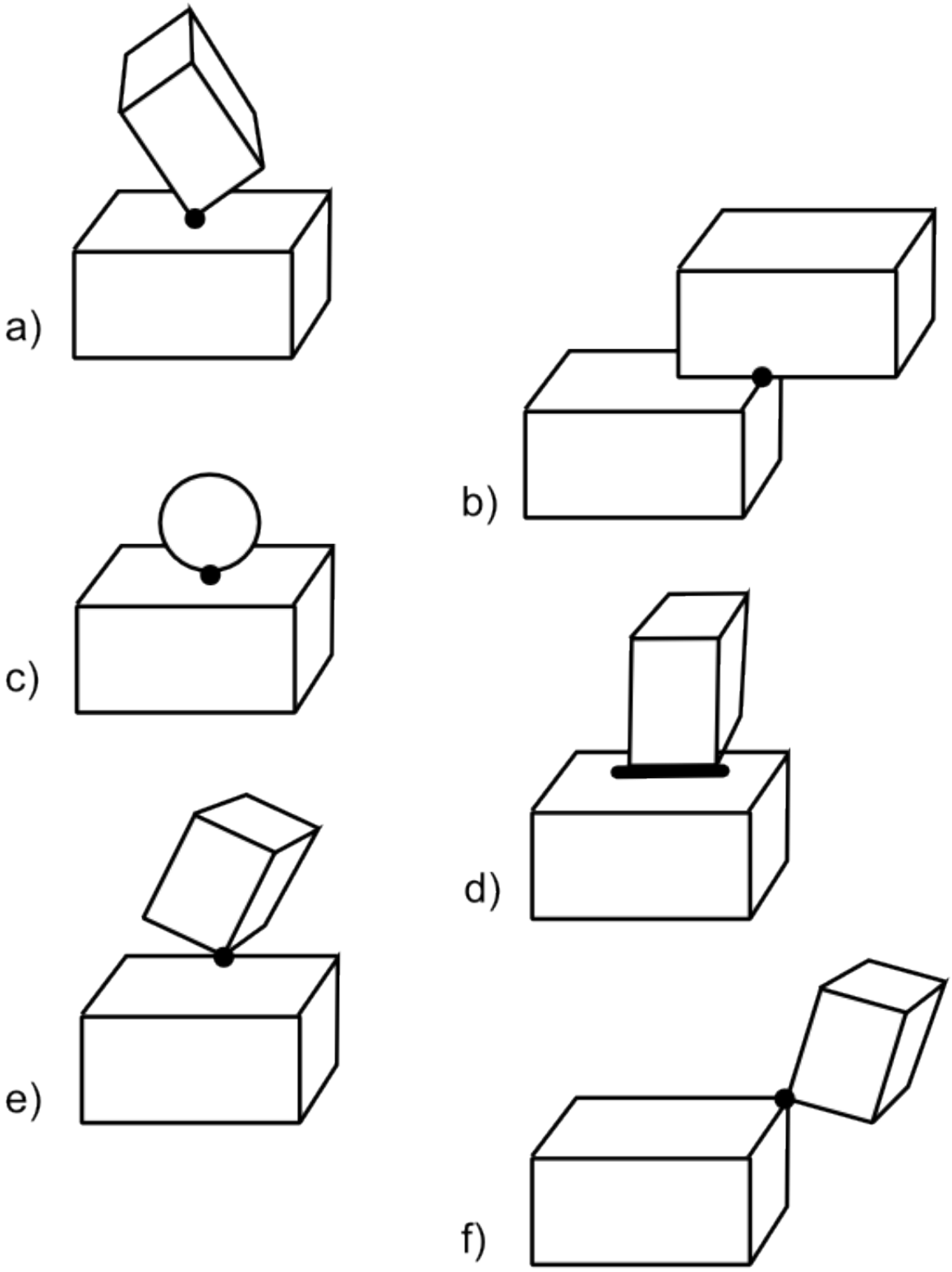
Prilikom kontakta dvaju objekata razlikujemo šest mogućih slučajeva: vrh-lice, brid-brid, lice-lice, brid-lice, vrh-brid i vrh-vrh. Mogući slučajevi se vide na slici 4.4. Slučaj vrh-vrh i slučaj vrh-brid su tako rijetki da ih možemo ignorirati. Dodatan razlog za ignoriranje ova dva slučaja je taj što je kod njih teško odrediti normalu sudara. Kontakt tipa lice-lice se javlja samo kad je jedan od objekata zakrivljena površina. U svim ostalim slučajevima kontakt lice-lice možemo zamijeniti kontaktom brid-brid. Kontakt brid-lice također možemo prikazati kao kontakt brid-brid uz zadržavanje fizikalne realističnosti [Millington07].

4.3 Detekcija sudara dvaju trokuta

U prethodnom smo poglavlju vidjeli da nas zanimaju kontakti tipa brid-brid i vrh-lice. Zbog toga ćemo pri detekciji sudara tražiti sudare koji rezultiraju takvim kontaktima. Algoritam 4.1 daje jednu mogućnost za provjeru sudara dvaju objekata radi kontakta tipa vrh-lice. Za kontakte tipa brid-brid možemo koristiti neki drugi algoritam. Bez obzira rezultira li sudar kontaktom tipa brid-brid ili vrh-lice, u oba slučaja će se raditi o sudaru dvaju mreža trokuta. To jest, uvijek će se raditi o sudaru dvaju trokuta. Zbog toga ćemo koristiti algoritam koji provjerava sudar između dvaju trokuta.

Postoji nekoliko brzih i kvalitetnih načina za provjeru sudara dvaju trokuta [Devillers02, Tropp05]. Ovdje će biti opisan postupak dan u [Moller97].

Definiramo dva trokuta T_1 sa pripadajućim vrhovima V_0^1, V_1^1, i V_2^1 te trokut T_2sa vrhovima V_0^2, V_1^2, i V_2^2. Neka trokut T_1leži u ravnini π_1, a trokut T_2 u ravnini π_2. Jednadžba ravnine π je dana izrazom:



Slika 4.4: Šest mogućih konfiguracija kontakta dvaju objekata
Od a) do f) su najznačajniji do najmanji značajnih. a) vrh-lice b) brid-brid c) lice-lice d) brid-lice e) vrh-brid f) vrh-vrh

$$N \cdot X + d = 0 \tag{4.1}$$

$$N = (V_1 - V_0) \otimes (V_2 - V_0)$$

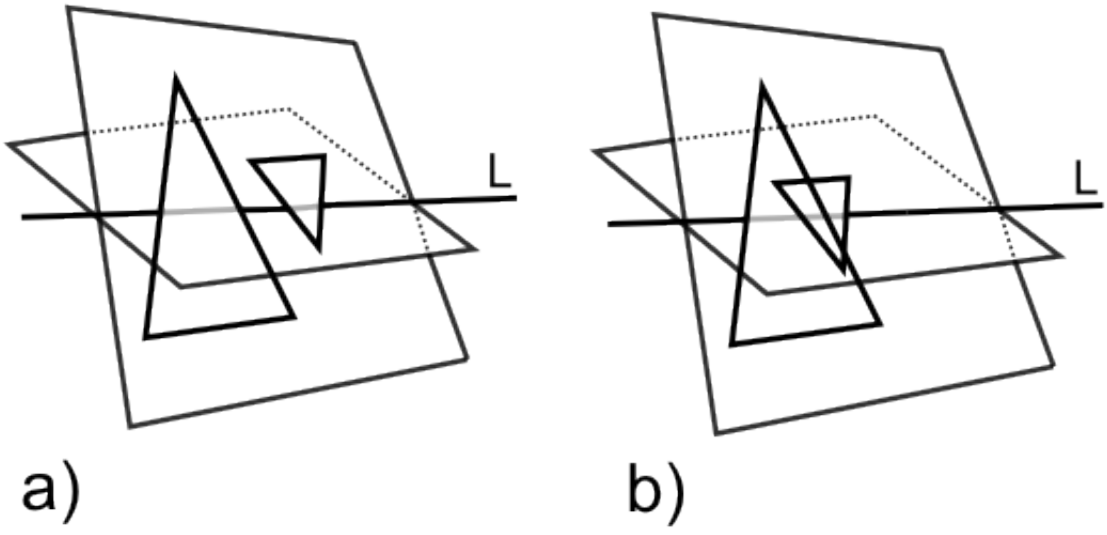
$$d = -N \cdot V_0$$

gdje je X bilo koja točka u ravnini π. Udaljenost točke T od ravnine π dobijemo uvrštavanjem točke T u jednadžbu ravnine π.

$$d_T = N \cdot T + d \tag{4.2}$$

Prvi korak ovog algoritma je provjera leže li svi vrhovi jednog trokuta na istoj strani ravnine drugog trokuta. Ukoliko to jest slučaj, algoritam prekida s radom jer ne postoji sudar.

Ako je udaljenost d za sve točke trokuta jednaka nuli, tada se trokuti nalaze u istoj ravnini. Tada trokute projiciramo na ravninu, poravnatu sa osima koordinatnog sustava, za koju su površine dvaju trokuta najveće. Tada za svaki brid trokuta T_1provjeravamo intersekciju sa svakim bridom trokuta T_2. Ako ni nakon toga sudar nije detektiran, provjeravamo je li jedan od trokuta u potpunosti sadržan unutar drugog. To možemo učiniti tako da za svaki vrh trokuta provjerimo je li unutar drugog trokuta pomoću algoritma 4.2.



Slika 4.5: Trokuti T_1i T_2 i ravnine u kojima leže
a) Nema intersekcije trokuta, intervali se ne preklapaju. b) Intersekcija trokuta, intervali se preklapaju.

Ako su dva trokuta u sudaru, mora postojati linija L smjera $N_1 \otimes N_2$ koja prolazi kroz oba trokuta kao što je vidljivo na slici 4.5.Linija L je dana izrazom:

$$L = O + t\,(N_1 \otimes N_2) \tag{4.3}$$

gdje je O neka točka na liniji L. Sada oba trokuta presijecaju liniju L. Ti presjeci na liniji L tvore intervale. Ako se ti intervali presijecaju, tada se i trokuti presijecaju. Pretpostavimo da je vrh V_1^1 trokuta T_1 na jednoj strani ravnine π_2, a vrhovi V_0^1 i V_2^1 na drugoj strani. Da bismo pronašli skalarnu vrijednost koja prezentira presjek bridova $\overline{V_0^1 V_1^1}$i $\overline{V_1^1 V_2^1}$ sa linijom L, prvo moramo projicirati vrhove trokuta T_1 na liniju L:

$$p_{V_i^1} = (N_1 \otimes N_2) \cdot \left(V_i^1 - O\right) \tag{4.4}$$

Sjecište brida $B = \overline{V_0^1 V_1^1}$ i linije L je dano izrazom:

$$B = \overline{V_0^1 V_1^1} \cap L = O + t_1\,(N_1 \otimes N_2) \tag{4.5}$$

Sada možemo naći vrijednost parametra t_1:

$$t_1 = p_{V_0^1} + \left(p_{V_1^1} - p_{V_0^1}\right) \frac{d_{V_0^1}}{d_{V_0^1} - d_{V_1^1}}$$

Nakon što izračunamo i vrijednost parametra t_2 za sjecište brida $\overline{V_1^1 V_2^1}$ sa linijom L, iste ćemo proračune provesti za trokut T_2. Ako se dobiveni intervali presijecaju, trokuti su u sudaru.

Algorithm 4.2 Točka u trokutu za dvije dimenzije.

Za točku P i trokut ABC
AKO ($(P - A) \otimes (B - A)$ < 0) točka nije u trokutu
AKO ($(P - B) \otimes (C - B)$ < 0) točka nije u trokutu
AKO ($(P - C) \otimes (A - C)$ < 0) točka nije u trokutu
INAČE točka je u trokutu

Algorithm 4.3 Intersekcija dvaju trokuta.

Za zadane trokute T_1 i T_2:
izračunaj jednadžbu ravnine trokuta T_2
izađi ako su sve točke trokuta T_1 na istoj strani ravnine trokuta T_2
izračunaj jednadžbu ravnine trokuta T_1
izađi ako su sve točke trokuta T_2 na istoj strani ravnine trokuta T_1
izračunaj liniju presijecanja L i projiciraj ju na najveću koordinatnu os
izračunaj intervale za svaki trokut
odredi presjecišta intervala

Poglavlje 5

Razrješavanje sudara

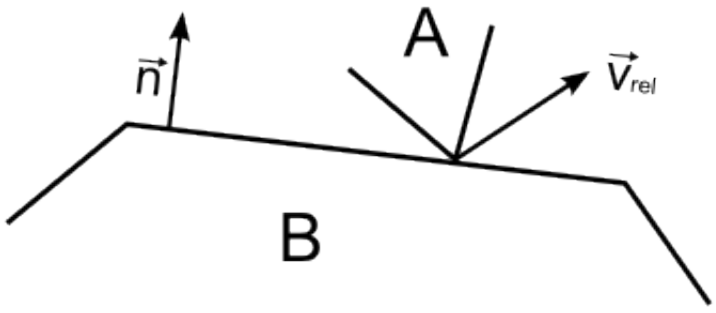
Nakon što je završila faza detekcija sudara, imamo sve potrebne podatke o sudaru. To su točka sudara P i normala sudara $\vec{n}$ te naravno dva čvrsta tijela, A i B, koja se nalaze u sudaru. Iz Section 1.3 znamo da je brzina točke P na na čvrstim tijelima A i B dana izrazima:

$$\vec{v}_{P_A}(t) = \vec{v}_A(t) + \vec{\omega}_A \otimes (\vec{p}_{P_A}(t) - \vec{p}_A(t)) \tag{5.1}$$

$$\vec{v}_{P_B}(t) = \vec{v}_B(t) + \vec{\omega}_B \otimes (\vec{p}_{P_B}(t) - \vec{p}_B(t)) \tag{5.2}$$

Definiramo relativnu brzinu točke $\vec{v}_{rel}$:

$$\vec{v}_{rel}(t) = \vec{n}(t)(\vec{v}_{P_A}(t) - \vec{v}_{P_B}(t)) \tag{5.3}$$



Slika 5.1: Relativna brzina točke P

Ako $\vec{v}_{rel}$ pokazuje u istom smjeru kao i $\vec{n}$, tijela se udaljavaju jedno od drugog.

Ako je $\vec{v}_{rel}$ manja od nule, tada se tijela kreću jedno prema drugom i moramo razriješiti sudar. Promjena u brzini dvaju tijela mora biti trenutna. Nemamo vremena za primjenjivanje sile koja bi dovela do promjene brzine. Trenutnu promjenu brzine čvrstog tijela ćemo postići primjenom impulsa sile. Impuls sile je umnožak sile i vremenskog intervala u kojem djeluje ta sila [Kulišić02].

$$\vec{I} = \vec{F} \triangle t \tag{5.4}$$

Prilikom razrješavanja sudara dvaju tijela, na njih ćemo primijeniti impuls sile da bismo im promijenili brzine. Impuls sile će se primjenjivati trenutno, u trenutku sudara t_0, pa iz daljnjih izraza možemo izostaviti vrijeme t. Smjer impulsa će biti u smjeru normale sudara $\vec{n}$. Zbog toga možemo pisati:

$$\vec{I} = i\vec{n} \tag{5.5}$$

gdje je i skalarna veličina kojom označavamo iznos impulsa sile. Označimo sa $\vec{v}^-_{P_A}$ i $\vec{v}^-_{P_B}$ brzine točke P na tijelima A i B prije primjene impulsa, a sa $\vec{v}^+_{P_A}$ i $\vec{v}^+_{P_B}$ brzine nakon primjene impulsa. Uz ovu notaciju možemo relativnu brzinu u smjeru normale sudara prije primjene impulsa pisati kao:

$$\vec{v}^-_{rel} = \vec{n}\left(\vec{v}^-_{P_A} - \vec{v}^-_{P_B}\right) \tag{5.6}$$

a relativnu brzinu u smjeru normale sudara nakon primjene impulsa kao:

$$\vec{v}^+_{rel} = \vec{n}\left(\vec{v}^+_{P_A} - \vec{v}^+_{P_B}\right) \tag{5.7}$$

Prilikom računanja impulsa koji moramo primijeniti, koristiti ćemo se zakonom o očuvanju količine gibanja pri elastičnom sudaru [Kulišić02]. On kaže da je ukupna količina gibanja prije sudara dvaju tijela jednaka ukupnoj količini gibanja nakon sudara:

$$P_1 + P_2 = P_1' + P_2' \tag{5.8}$$

Pošto su mase dvaju tijela u sudaru jednake prije i nakon sudara, primjena zakona o očuvanju količine gibanja nam daje [Baraff01]:

$$\vec{v}^+_{rel} = \epsilon\vec{v}^-_{rel} \tag{5.9}$$

Veličina ϵ je koeficijent restitucije, faktor koji nam govori koliko se kinetičke energije izgubi prilikom sudara. Vrijednost koeficijenta restitucije je $0 \leq \epsilon \leq 1$. Ako je ϵ jedan, nema gubitka kinetičke energije prilikom sudara i radi se o savršeno neelastičnom sudaru. Ako je ϵ jednak nuli, radi se o savršenom elastičnom sudaru, to jest tijela su slijepljena nakon sudara.

Definiramo još veličine $\vec{r}_A$ i $\vec{r}_B$, vektore od središta tijela A i B prema točki kontakta P:

$$\vec{r}_A = \vec{p}_P - \vec{p}_A \tag{5.10}$$

$$\vec{r}_B = \vec{p}_P - \vec{p}_B \tag{5.11}$$

Sada brzinu točke a nakon sudara možemo pisati kao:

$$\vec{v}_{P_A}^{+} = \vec{v}_A^{+} + \vec{\omega}_A^{+} \otimes \vec{r}_A^{+} \tag{5.12}$$

a brzinu centra mase i kutnu brzinu tijela A nakon sudara:

$$\vec{v}_A^{+} = \vec{v}_A^{-} + \frac{i\vec{n}}{m_A} \tag{5.13}$$

$$\vec{\omega}_A^{+} = \vec{\omega}_A^{-} + I_A^{-1} \cdot (\vec{r}_A \otimes i\vec{n}) \tag{5.14}$$

Kombinirajući (5.12), (5.13) i (5.14) te primjenjujući na tijelo B negativan iznos impulsa sile i dobivamo:

$$\vec{v}_{P_A}^{+} = \vec{v}_{P_A}^{-} + i\left(\frac{\vec{n}}{m_A} + I_A^{-1} \cdot (\vec{r}_A \otimes \vec{n})\right) \otimes \vec{r}_A \tag{5.15}$$

$$\vec{v}_{P_B}^{+} = \vec{v}_{P_B}^{-} - i\left(\frac{\vec{n}}{m_B} + I_B^{-1} \cdot (\vec{r}_B \otimes \vec{n})\right) \otimes \vec{r}_B \tag{5.16}$$

To dovodi do:

$$\vec{v}_{P_A}^{+} - \vec{v}_{P_B}^{+} = \left(\vec{v}_{P_A}^{-} - \vec{v}_{P_B}^{-}\right) + i\left(\frac{\vec{n}}{m_A} - \frac{\vec{n}}{m_B} + \left(I_A^{-1} \cdot (\vec{r}_A \otimes \vec{n})\right) \otimes \vec{r}_A + \left(I_B^{-1} \cdot (\vec{r}_B \otimes \vec{n})\right) \otimes \vec{r}_B\right) \tag{5.17}$$

Izraz 5.17 pomnožimo sa $\vec{n}$ da bismo dobili $\vec{v}_{rel}^{+}$. Kako je $\vec{n}$ jedinične dužine, vrijedi $\vec{n} \cdot \vec{n} = 1$ pa možemo pisati:

$$\begin{aligned}\vec{v}_{rel}^{+} &= \vec{n}\left(\vec{v}_{P_A}^{-} - \vec{v}_{P_B}^{-}\right) + i\left(\frac{1}{m_A} + \frac{1}{m_B} + \vec{n}\left(I_A^{-1} \cdot (\vec{r}_A \otimes \vec{n})\right) \otimes \vec{r}_A + \vec{n}\left(I_B^{-1} \cdot (\vec{r}_B \otimes \vec{n})\right) \otimes \vec{r}_B\right) \\ &= \vec{v}_{rel}^{-} + i\left(\frac{1}{m_A} + \frac{1}{m_B} + \vec{n}\left(I_A^{-1} \cdot (\vec{r}_A \otimes \vec{n})\right) \otimes \vec{r}_A + \vec{n}\left(I_B^{-1} \cdot (\vec{r}_B \otimes \vec{n})\right) \otimes \vec{r}_B\right)\end{aligned} \tag{5.18}$$

Rješavanjem izraza 5.18 za i dobivamo konačni izraz za iznos impulsa sile:

$$i = \frac{-(1+\epsilon)\,\vec{v}_{rel}^{-}}{\frac{1}{m_A} + \frac{1}{m_B} + \vec{n}\left(I_A^{-1} \cdot (\vec{r}_A \otimes \vec{n})\right) \otimes \vec{r}_A + \vec{n}\left(I_B^{-1} \cdot (\vec{r}_B \otimes \vec{n})\right) \otimes \vec{r}_B} \tag{5.19}$$

Konačni izrazi za nove brzine tijela A i B su iz 5.13 i 5.14:

$$\begin{aligned}\vec{v}_A^{+} &= \vec{v}_A^{-} + \frac{i\vec{n}}{m_A} \\ \vec{\omega}_A^{+} &= \vec{\omega}_A^{-} + I_A^{-1} \cdot (\vec{r}_A \otimes i\vec{n}) \\ \vec{v}_B^{+} &= \vec{v}_B^{-} - \frac{i\vec{n}}{m_B} \\ \vec{\omega}_B^{+} &= \vec{\omega}_B^{-} + I_B^{-1} \cdot (\vec{r}_B \otimes i\vec{n})\end{aligned} \tag{5.20}$$

Poglavlje 6

Napredne detekcije sudara

Dosad prikazane metode omogućuju nam da napravimo simulaciju dinamike čvrstih tijela visokog stupnja realističnosti. Problemi će nastati kada u scenu stavimo više objekata ili kada mreže poligona kojima su definirani objekti postanu detaljnije. U oba slučaja doći će do povećanja ukupnog broja poligona u sceni. Pošto se naša rutina za detekciju sudara oslanja na detekciju sudara dvaju trokuta, povećanje broja poligona dovesti će do pada performansi simulacije. Da bismo to spriječili, moramo smanjiti broj trokuta koji se provjeravaju radi sudara. Dva su načina na koji to možemo učiniti, pojednostavljenjem reprezentacije objekta ili smanjenjem dijelova mreže trokuta koje ćemo testirati radi sudara.

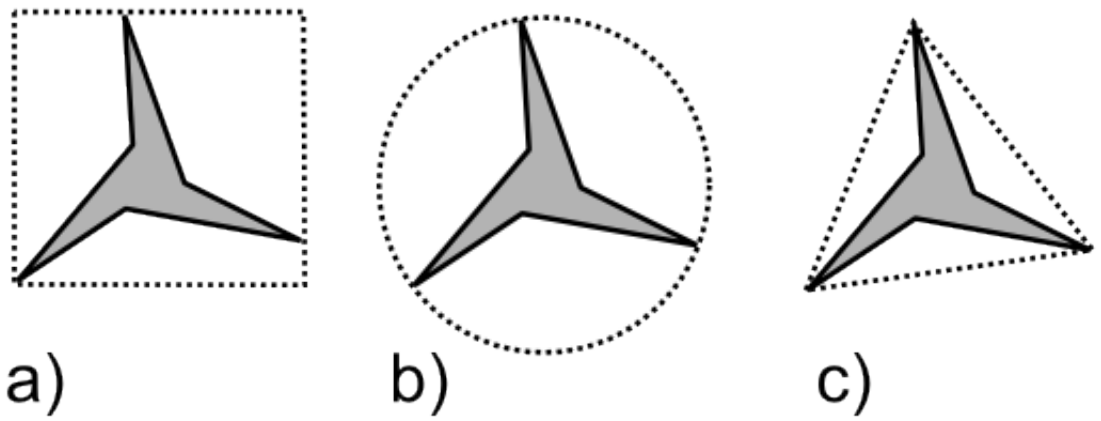
6.1 Obujmice

Već smo rekli da je objekt koji predstavlja čvrsto tijelo dan kao skup poligona oblika trokuta, takozvane mreže poligona. Način na koji se provjerava jesu li dva objekta u sudaru jest da se za svaki trokut objekta A provjerava je li u sudaru sa bilo kojim trokutom objekta B. Ova metoda ima složenost $\mathcal{O}(n^2)$. Ako su objekti A i B sastavljeni od mreže poligona sa 100 trokuta, moramo napraviti 10 000 provjera trokut-trokut. Kada su objekti A i B doista u sudaru, ovakva provjera je nužda da bismo dobili točku sudara i normalu sudara. No, što ako objekti A i B nisu u sudaru? Tada uzalud vršimo 10 000 testova trokut-trokut.

Način da drastično smanjimo broj testova primitiva[1] je da mrežu poligona koja predstavlja neki objekt zamijenimo sa jednom primitivom ili pak manjim brojem primitiva. Ovakve reprezentacije objekta nazivamo obujmicama jer u potpunosti obujmljuju originalni objekt. Slika 6.1 prikazuje par obujmica i način na koji one obujmljuju prvobitni objekt.

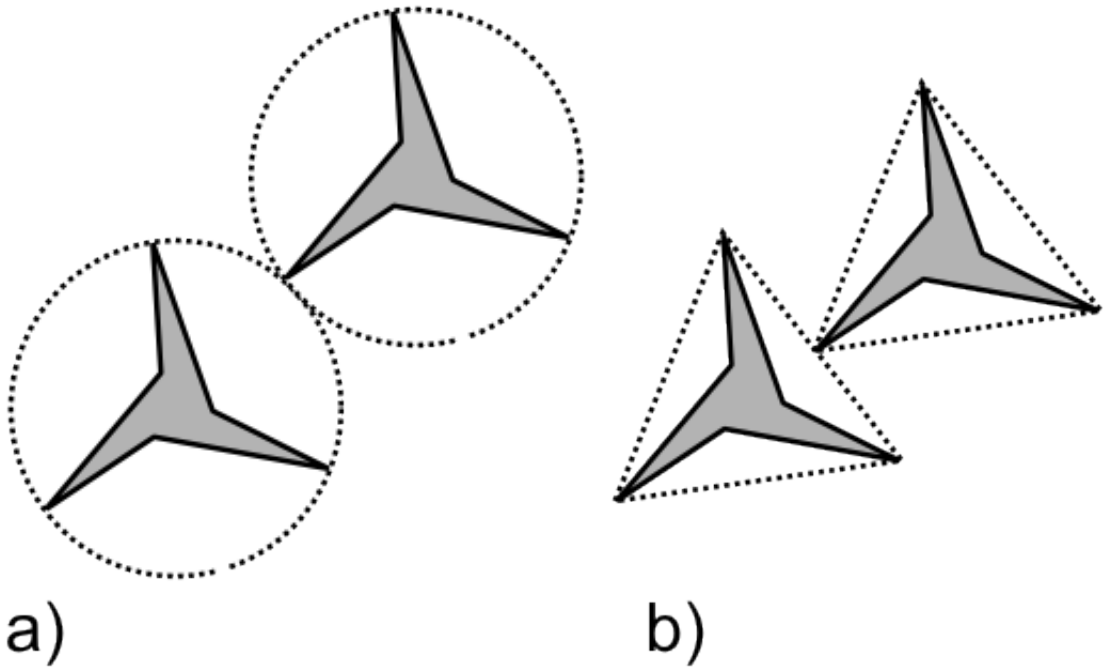
Osnovni tipovi obujmica su obujmice u obliku kvadra i sfere te konveksni omotači (*convex hull*). Prilikom odabira obujmice moramo dobro odvagnuti njihove prednosti i mane. U

[1] Primitiva je geometrijski objekt koji je nedjeljiv u funkciji u kojoj ga promatramo. Na primjer: točka, linija, trokut, tetraedar, sfera...



Slika 6.1: Obujmice
a) Obujmica oblika kvadra. b) Obujmica oblika sfere. c) Konveksni omotač.

ovom radu ćemo se osvrnuti na usporedbu obujmice u obliku sfere i konveksnog omotača kao dviju obujmica čije se karakteristike bitno razlikuju[2]. Zanima nas jednostavnost testiranja dviju obujmica radi sudara te koliko usko *prianjaju* uz originalni objekt. Prianjanje je mjera koja nam govori koliko često će test dviju obujmica javiti sudar kada sami objekti koji su sadržani unutar obujmica nisu u sudaru. Primjer ovog slučaja se vidi na slici 6.2.



Slika 6.2: Slučaj kada dva objekta nisu u sudaru, ali njihove obujmice jesu

Kao što se vidi na slici 6.1, konveksni omotač najbolje prianja uz originalni objekt. Prilikom testiranja sudara dvaju konveksnih omotača, testiranje ćemo vršiti istim metodama

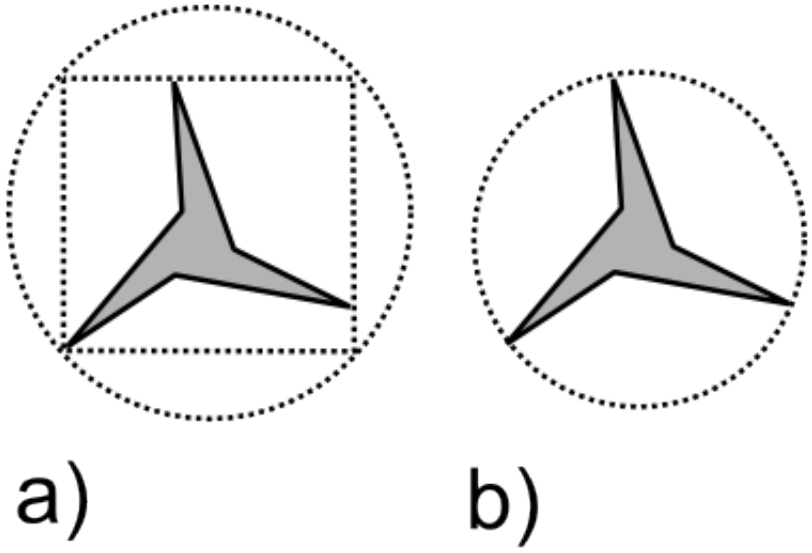
[2]Za vrlo detaljni opis obujmica u obliku kvadra pogledati [Bergen03]

koje koristimo za testiranje sudara dviju mreža poligona. Testiranje sudara dviju obujmica u obliku sfere je puno jednostavnije. Nedostatak upotrebe obujmica u obliku sfere je njihovo slabo prianjanje.

6.1.1 Upotreba obujmica u obliku sfere

Upotreba obujmica u obliku sfere je vrlo česta. Razlog za to leži u činjenici da je takvu obujmicu lako napraviti i testirati radi sudara. Naivni pristup, čistom silom(*brute force*), računanju najbolje prianjajuće obujmice u obliku sfere koristi činjenicu da je sfera jednoznačno definirana sa četiri koplanarne točke. Ovaj postupak jednostavno stvori po jednu sferu za svaku moguću kombinaciju četiri vrha zadanog objekta. Isto napravi i za svaku moguću kombinaciju tri vrha i dva vrha. Od svih stvorenih sfera, zadržava se ona koja sadrži sve ostale, a ima najmanji radijus. Ovaj postupak ima složenost $\mathcal{O}(n^5)$. Puno brži postupak, linearne složenosti, opisan je u [Welzl91].

Jednostavniji je postupak izgradnje obujmice u obliku sfere iz već postojeće obujmice u obliku kvadra. Prvo se iz skupa vrhova zadanog objekta izračuna obujmica u obliku kvadra. Tada se kao promjer sfere uzme najduža dijagonala te kutije. Promjer ovakve sfere može biti i do dva puta duži nego što bi bio promjer najmanje moguće sfere. Iako ne stvara sferu najmanjeg radijusa, dobivena obujmica u većini slučajeva ima zadovoljavajući faktor prianjanja. Usporedba obujmice dobivene ovim postupkom i najmanje moguće obujmice u obliku sfere je prikazana na slici 6.3.



Slika 6.3: Obujmice u obliku sfere
a) Sfera iz obujmice u obliku kvadra. b) Najmanja moguća sfera.

Provjera sudara između dviju sfera je krajnje jednostavna. Ako su sfere, S_1 i S_2, dane svojim centrima, C_1 i C_2, te svojim polumjerima, R_1 i R_2, one su u sudaru kada je udaljenost između njihovih središta manja ili jednaka zbroju njihovih radijusa.

$$\sqrt{(C_1 - C_2) \cdot (C_1 - C_2)} \leq (R_1 + R_2) \tag{6.1}$$

Pošto nas ne zanima stvarna udaljenost između dviju sfera , nego samo omjer te udaljenosti i zbroja njihovih radijusa, izraz 6.1 možemo kvadrirati te se tako riješiti skupe operacije korjenovanja:

$$(C_1 - C_2) \cdot (C_1 - C_2) \leq (R_1 + R_2)^2 \tag{6.2}$$

6.1.2 Upotreba konveksnog omotača

Pošto je u primjeni obujmica jako važno dobro prianjanje obujmice, posebnu ćemo pozornost posvetiti korištenju konveksnog omotača pri detekciji sudara dvaju objekata. Konveksni omotač nekog objekta je najmanje, konveksno geometrijsko tijelo koje može obuhvatiti taj objekt. Ukoliko je sam objekt konveksan, njegova mreža poligona je njegov konveksni omotač.

Postoji nekoliko algoritama za izgradnju konveksnog omotača za zadani skup točaka (Incremental, Divide and Conquer, Gift Wrap, QuickHull). Vjerojatno najpoznatiji od njih je QuickHull. [3]. Detekciju sudara između dva konveksna omotača provodimo istim metodama kojima bismo detektirali sudar dvaju konveksnih objekata.

Inkrementalni algoritam

Jedan od algoritama za izgradnju konveksnog omotača je inkrementalni algoritam. Praktične implementacije ovog algoritma imaju složenost $\mathcal{O}(n^2)$ dok optimalne implementacije imaju složenost $\mathcal{O}(n \log n)$ [O'Rourke98]. Prije nego što prezentiramo inkrementalnu metodu, moramo definirati neke pojmove.

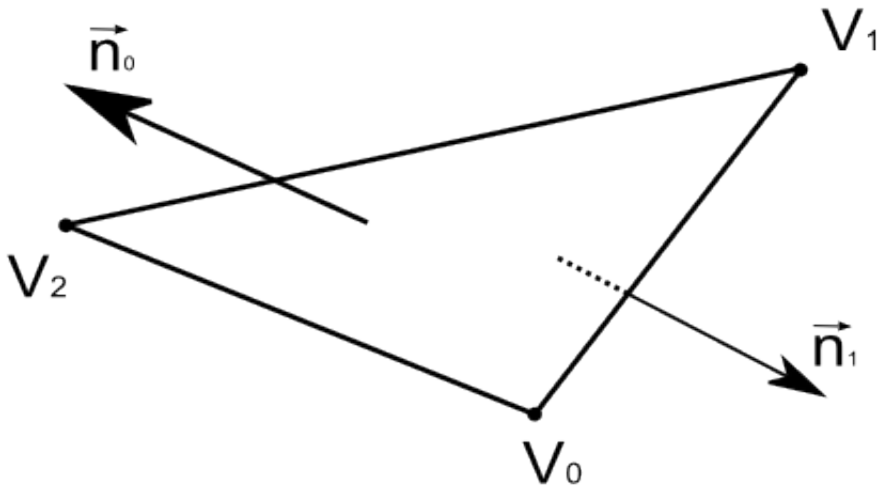
Definiramo volumen tetraedra $P_0P_1P_2P_3$ gdje su $P_0 = (x_{0,}y_0, y_0)$, $P_1 = (x_{1,}y_1, y_1)$, $P_2 = (x_{2,}y_2, y_2)$ i $P_3 = (x_{3,}y_3, y_3)$ kao:

$$V = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.3}$$

ukoliko je trokut $P_0P_1P_2$ pozitivno orijentiran i točka P_3 se nalazi s pozitivne strane ravnine trokuta $P_0P_1P_2$, volumen tetraedra će biti pozitivan. U to slučaju kažemo da točka P_3 „vidi" trokut $P_0P_1P_2$. Ukoliko se točka P_3 nalazi s negativne strane trokuta $P_0P_1P_2$, to jest ako se točka P_3i trokut $P_0P_1P_2$ „ne vide", volumen će biti negativan. Volumen tetraedra $P_0P_1P_2P_3$ će biti nula ako je točka P_3 u istoj ravnini kao i trokut $P_0P_1P_2$.

[3] Svoju popularnost QuickHull uvelike duguje kvalitetnoj implementaciji koja se može slobodno skinuti sa http://www.qhull.org/html/index.htm.

Definiramo d-trokut (*d-triangle*) kao geometrijsko tijelo koje ima dvije stranice, F_0 i F_1, tri vrha, V_0, V_1 i V_2, i tri brida E_0, E_1 i E_2. d-trokut je u biti trokut koji ima dvije strane. Prva strana je $V_0V_1V_2$, a druga $V_2V_1V_0$. Slika 6.4 prikazuje d-trokut.



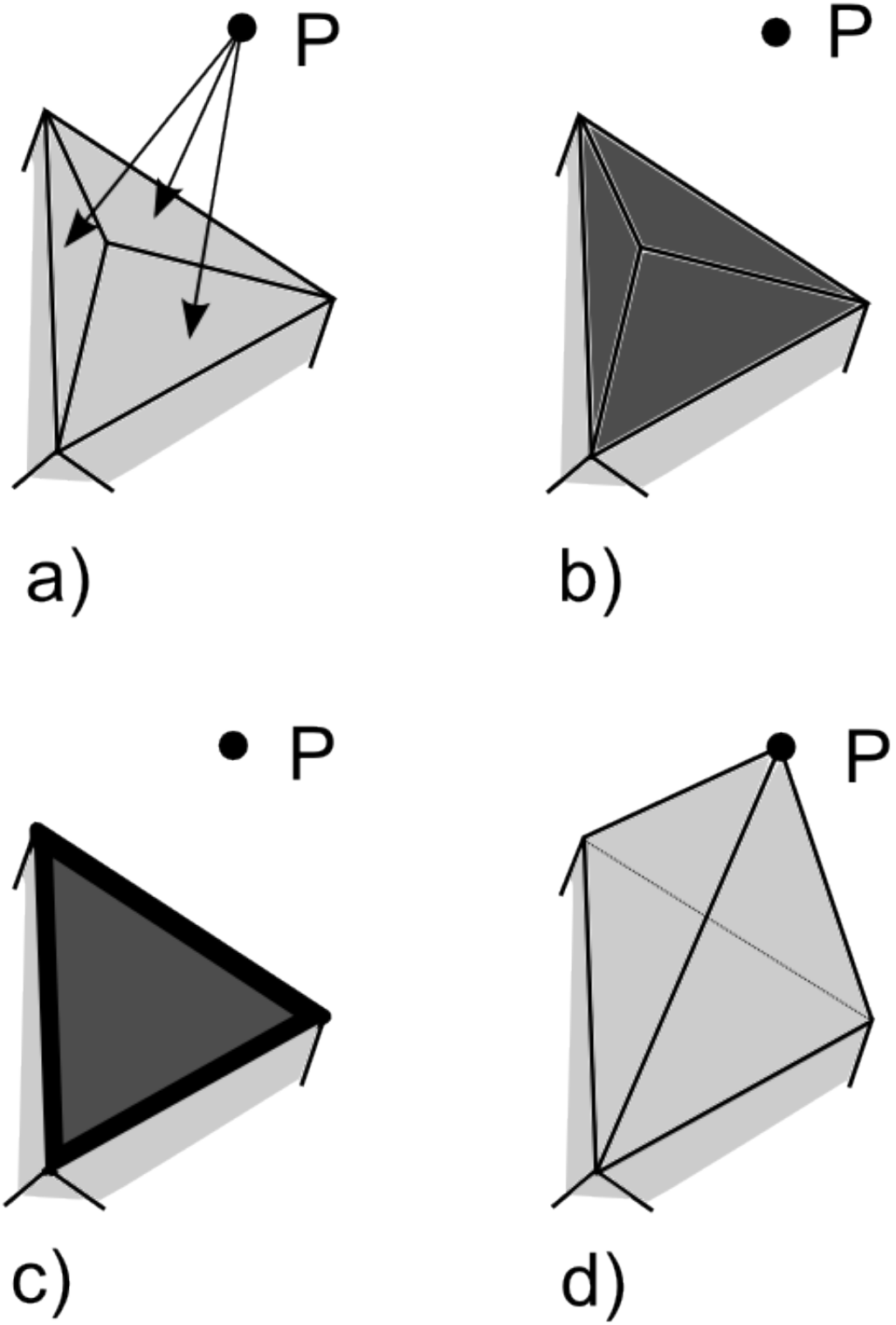
Slika 6.4: d-trokut

Definiramo skup točaka za koji radimo konveksni omotač V, skup trokuta konveksnog omotača T te skup bridova konveksnog omotača E. Također definiramo skup vidljivih bridova E_V.

Izgradnju konveksnog omotača započinjemo izradom d-trokuta. Za d-trokut nam trebaju tri nekolinearne točke. Uzimamo prve tri nekolinearne točke iz skupa točaka V. Nakon što smo napravili d-trokut, dodajmo njegova tri brida u skup E i njegova dva trokuta u skup T. Ovaj d-trokut je naš privremeni konveksni omotač koji ćemo unaprjeđivati sa svakom novom točkom iz skupa V u petlji koja je dana sa 6.1. Proces dodavanja nove točke se vidi na slici 6.5.

Algorithm 6.1 Glavna petlja inkrementalnog algoritma.

```
//Za svaku novu točku V_i
ZA svaki trokut T_i iz skupa trokuta konveksnog omotača T
    AKO ( V_i vidi T_i )
        ZA svaki brid E_i trokuta T_i
            AKO ( E_i nije u skupu vidljivih bridova ) dodaj E_i u skup vidljivih bridova E_V
            INAČE ( obriši brid E_i iz skupa vidljivih bridova E_V )
        obriši trokut T_i iz skupa trokuta konveksnog omotača T
    AKO (skup vidljivih bridova E_V nije prazan)
        ZA svaki brid E_Vi iz skupa vidljivih bridova E_V
            napravi novi trokut T_v
            dodaj trokut T_v u skup trokuta konveksnog omotača T
            obriši brid E_Vi iz skupa vidljivih bridova E_V
```



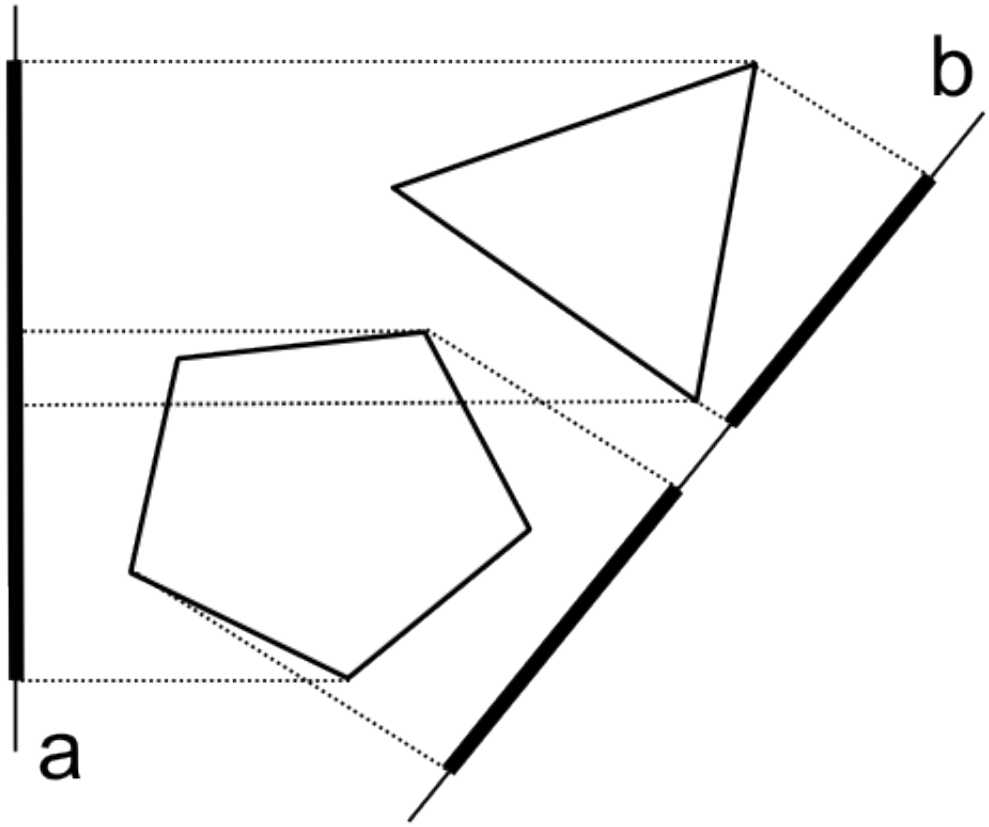
Slika 6.5: Dodavanje točke P konveksnom omotaču
a) Određuju se vidljivi trokuti. b) Trokuti i bridovi koje vidi točka P. c) Nakon uklanjanja unutrašnjih bridova i vidljivih trokuta, ostaje rupa u konveksnom omotaču i bridovi „horizonta”. d) Dodavanjem novih trokuta popunjava se rupa.

6.2 Algoritmi za detekciju sudara konveksnih objekata

Nakon što smo izgradili konveksne omotače za dva objekta čije sudare želimo testirati, moramo odabrati metodu kojom ćemo samo testiranje izvršiti. Dvije metode su posebno zanimljive, SAT algoritam i GJK algoritam.

6.2.1 SAT algoritam

SAT (*Separating Axis Theorem*) je vjerojatno najpopularnija metoda za testiranje sudara dvaju konveksnih objekata. Metoda se zasniva na teoremu o razdvajajućoj osi. Teorem o razdvajajućoj osi kaže da za svaka dva konveksna objekta koja nisu u sudaru, postoji os na kojoj projekcije tih dvaju objekata nisu u sudaru [Eberly01]. Primjer razdvajajuće osi je prikazan na slici 6.6. SAT je trenutno najbrža metoda za testiranje sudara dviju obujmica u obliku kutije.



Slika 6.6: SAT za dva objekta
Os a nije razdvajajuća dok os b jest.

Da bismo utvrdili jesu li dvije obujmice u obliku kutije u sudaru, moramo testirati maksimalno 15 osi da bismo utvrdili je li koja od njih razdvajajuća os [4]. Za svaku os koju testiramo, projiciramo prvi objekt na tu os tako da svaki vrh objekta pomnožimo sa normaliziranim vektorom smjera osi. Razmak između najveće i najmanje vrijednosti projici-

[4] Način odabira ovih osi kao i detaljna rasprava o SAT algoritmu je prezentirana u [Bergen03].

ranih vrhova će dati interval na testiranoj osi. Isti postupak ponovimo i za drugi objekt. Sada samo moramo testirati dva dobivena intervala radi preklapanja. Čim za neku os utvrdimo da je razdvajajuća, prekidamo sa izvršavanjem algoritma. Zbog ovoga je SAT jako dobar u situacijama kada imamo puno objekata u sceni, ali nemamo puno sudara.

Pri testiranju sudara dvaju općenitih konveksnih omotača, za testirane osi uzimamo normale svih lica te okomice na sve parove bridova dvaju konveksnih omotača. Broj osi koje moramo testirati SAT algoritmom da bismo utvrdili sudar dvaju konveksnih omotača je dan formulom izrazom:

$$d = 2k + (3k - 6)^2 \tag{6.4}$$

gdje je k broj orijentacija poluravnina koje tvore konveksni omotač. Proučimo ovu relaciju na primjeru sudara dvaju tetraedra. Tetraedar možemo promatrati kao konveksni omotač koji se sastoji od četiri poluravnine. Dakle, $k = 4$. Po izrazu 6.4, broj osi koje moramo testirati je $d = 44$. SAT algoritam za detekciju sudara konveksnih omotača koji se sastoje od k relativno orijentiranih poluravnina ima vremensku složenost $\mathcal{O}(k^3)$ [Bergen03]. Zbog toga SAT nije dobra metoda za testiranje sudara dvaju konveksnih omotača.

6.3 GJK algoritam

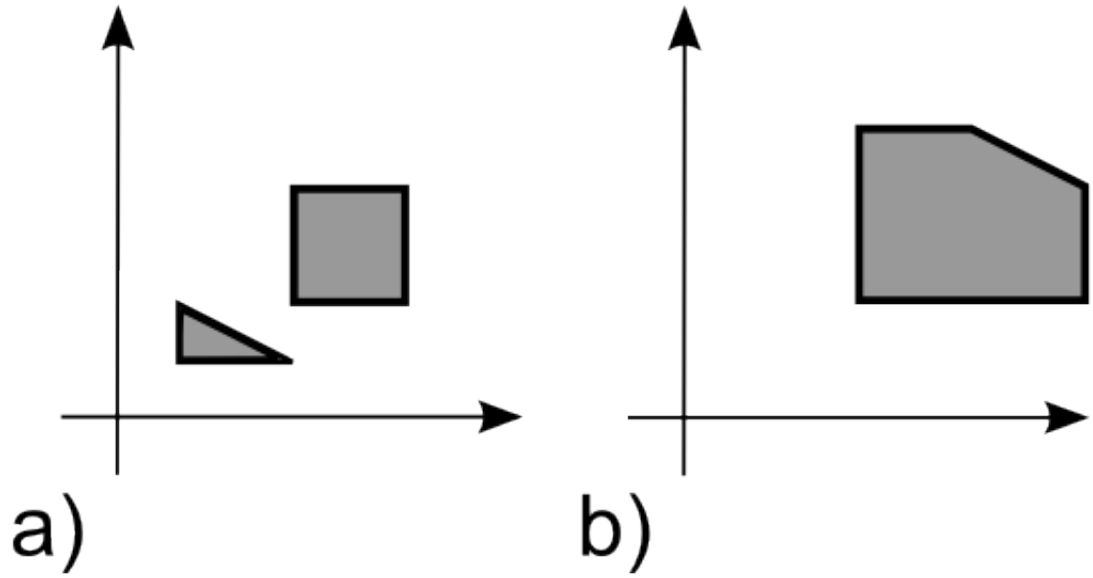
Provjeru sudara dvaju konveksnih omotača vršiti ćemo GJK(*Gilbert–Johnson–Keerthi*) algoritmom. GJK algoritam nije među zastupljenijim algoritmima u interaktivnim simulacijama. Razlog za to leži u ne baš jasnom postupku implementacije algoritma. Većina radova na ovu temu pristupala je GJK algoritmu sa matematičkog gledišta. Christer Ericson u svojoj knjizi *„Real-Time Collision Detection"* zagovara vektorski pristup ovom algoritmu. U ovom radu će biti opisan vektorski pristup koji se nastavlja na rad Christera Ericsona, ali donosi bitna poboljšanja, kako u brzini samog algoritma, tako i u jednostavnosti njegove implementacije.

Prije nego pristupimo objašnjenju samog algoritma, moramo definirati neke pojmove iz područja detekcije sudara.

Suma i razlika Minkovskog

Neka su A i B dva skupa točaka i neka su a i b vektori dviju točaka u tim skupovima. Suma Minkovskog je tada data izrazom:

$$A \oplus B = \{a + b : a \epsilon A, b \epsilon B\} \tag{6.5}$$

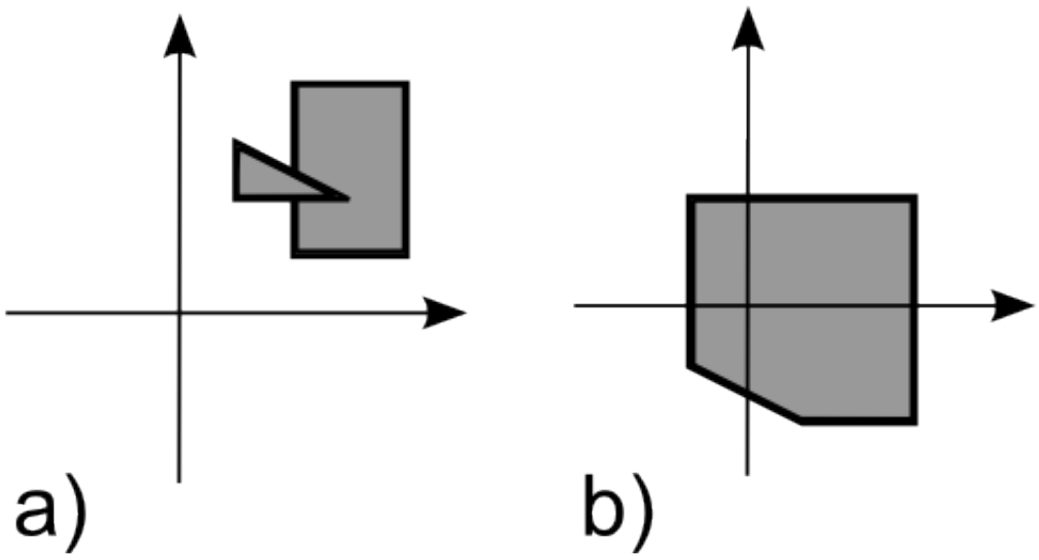


Slika 6.7: Suma Minkovskog
a) Objekti A i B. b) $A \oplus B$

Razlika Minkovskog je:

$$A \ominus B = \{a - b : a \epsilon A, b \epsilon B\} \tag{6.6}$$

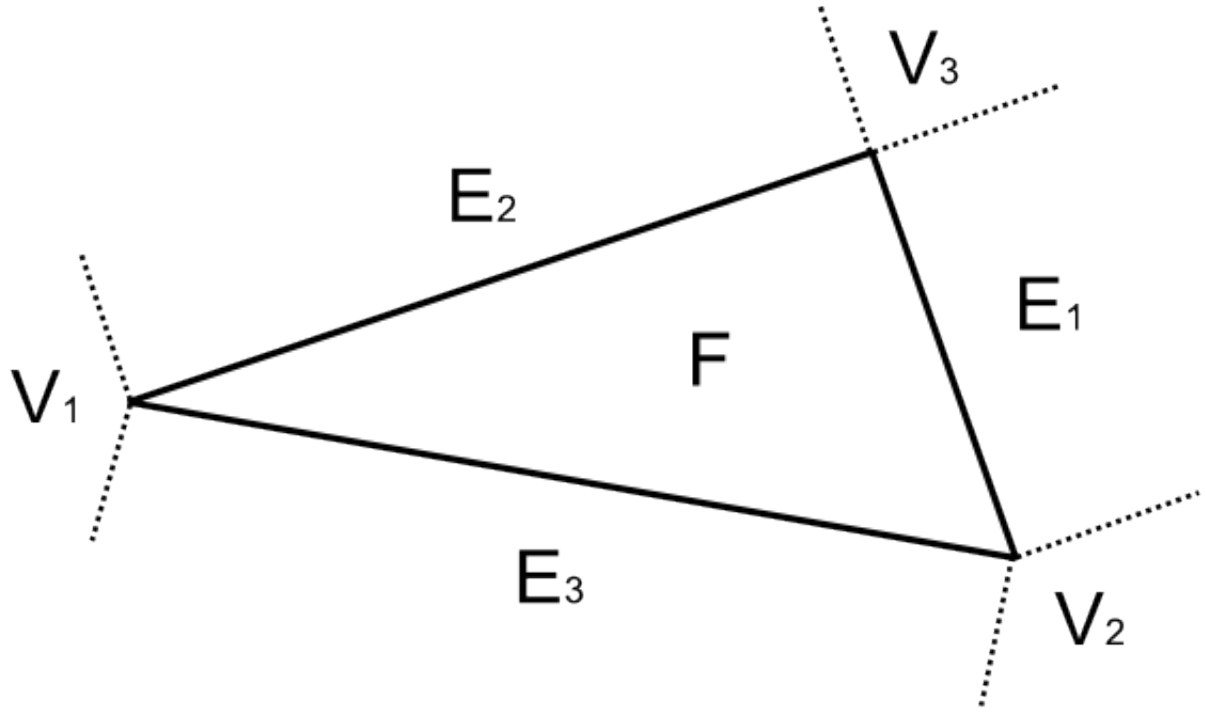
Ako su objekti A i B u sudaru, tada imaju barem jednu zajedničku točku. Ukoliko se ta zajednička točka objekta B oduzme od iste točke objekta A, rezultat će biti nula. Iz ovoga proizlazi važno svojstvo razlike Minkovskog. To je da razlika Minkovskog sadrži ishodište ako su objekti A i B u sudaru. Takav primjer je prikazan na slici 6.8.



Slika 6.8: Razlika Minkovskog
a) Objekti A i B. b) $A \oplus (-B)$

Voronoieve regije

Za konveksni objekt C, definiramo primitivu P kao vrh, brid ili lice objekta C. Voronoieva regija primitive P konveksnog objekta C je tada skup točaka u prostoru koje su bliže primitivi P nego bilo kojoj drugoj primitivi konveksnog objekta C. Slika 6.9 prikazuje sedam Voronoievih regija trokuta.



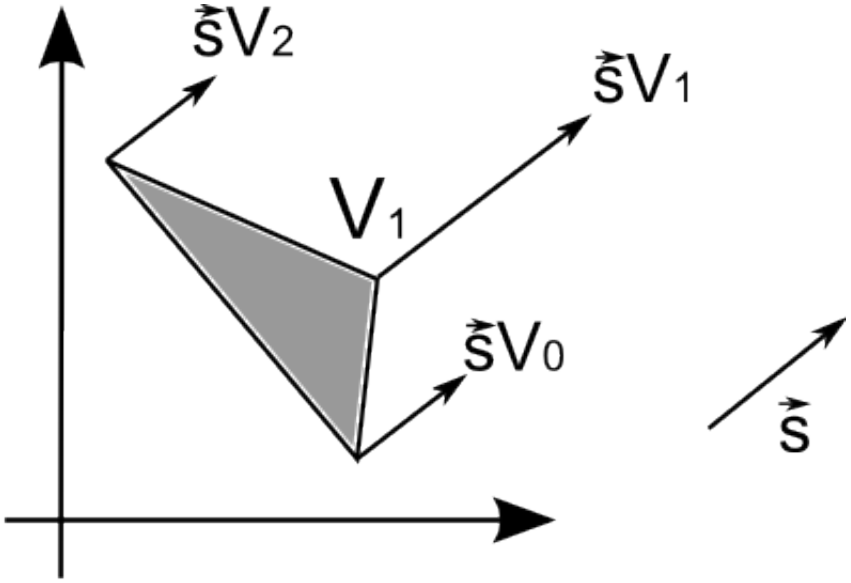
Slika 6.9: Voronoieve regije trokuta
Trokut ima sedam Voronoievih regija. Voronoieve regije vrhova V_1, V_2 i V_3. Voronoieve regije bridova E_1, E_2 i E_3. Voronoieva regija trokuta F.

Funkcija potpore

Često nas za neki konveksni objekt zanima koja je točka tog objekta najudaljenija u nekom smjeru. Smjer u kojem tražimo najudaljeniju točku objekta zovemo vektor potpore i označavamo sa $\vec{s}$.

Funkcija potpore, dana za neki konveksni objekt C, je funkcija $f(\vec{s})$ čija je vrijednost najudaljenija točka objekta C u danom smjeru $\vec{s}$. Za konveksni omotač, funkcija potpore je dana izrazom:

$$f_{CH}(\vec{s}) = \{\max(\vec{s} \cdot P_i) : P_i \epsilon CH, \} \tag{6.7}$$



Slika 6.10: Funkcije potpore
Funkcije potpore trokuta za zadani vektor potpore $\vec{s}$ jednaka je točki V_1.

6.3.1 Glavni algoritam

Glavna ideja GJK algoritma je da se pokuša izgraditi simpleks unutar razlike Minkovskog dvaju objekata koji će obuhvatiti ishodište. Ukoliko se u tome uspije, objekti su u sudaru. Ukoliko GJK nije u stanju izgraditi simpleks, ishodište se nalazi izvan razlike Minkovskog i objekti nisu u sudaru. Simpleks koji GJK pokušava izgraditi je u slučaju trodimenzionalnog GJK algoritma tetraedar, a u slučaju dvodimenzionalnog GJK algoritma trokut. Vrhovi simpleksa su uvijek vrhovi konveksnog omotača razlike Minkovskog.

Algorithm 6.2 GJK algoritam

odaberi početnu točku na konveksnom omotaču razlike Minkovskog A
stavi točku A u simpleks strukturu S
odaberi početni vektor potpore $\vec{s} = -A$
PETLJA
    $A = f_A(\vec{s}) + f_B(-\vec{s})$
    AKO ($A \cdot \vec{s} < 0 \cdot \vec{s}$) VRATI(nema sudara)
    dodaj A u simpleks strukturu S
    AKO (Obradi_simpleks(S, $\vec{s}$)) VRATI(sudar)

Tijek GJK algoritma za dvije dimenzije je prikazan na slici 6.11. Želimo izvršiti provjeru sudara dvaju konveksnih objekata A i B 6.11 a). Tijekom provjere sudara, kretati ćemo se po konveksnom omotaču razlike Minkovskog objekata A i B. Samo razliku Minkovskog za ova dva objekta nećemo računati. To nije potrebno ako znamo da će funkcija potpore za $A \ominus B$,

$$f_{A \ominus B}(\vec{s}) = f_A(\vec{s}) + f_B(-\vec{s}) \tag{6.8}$$

uvijek vratiti točku na konveksnom omotaču razlike Minkovskog $A \ominus B$.

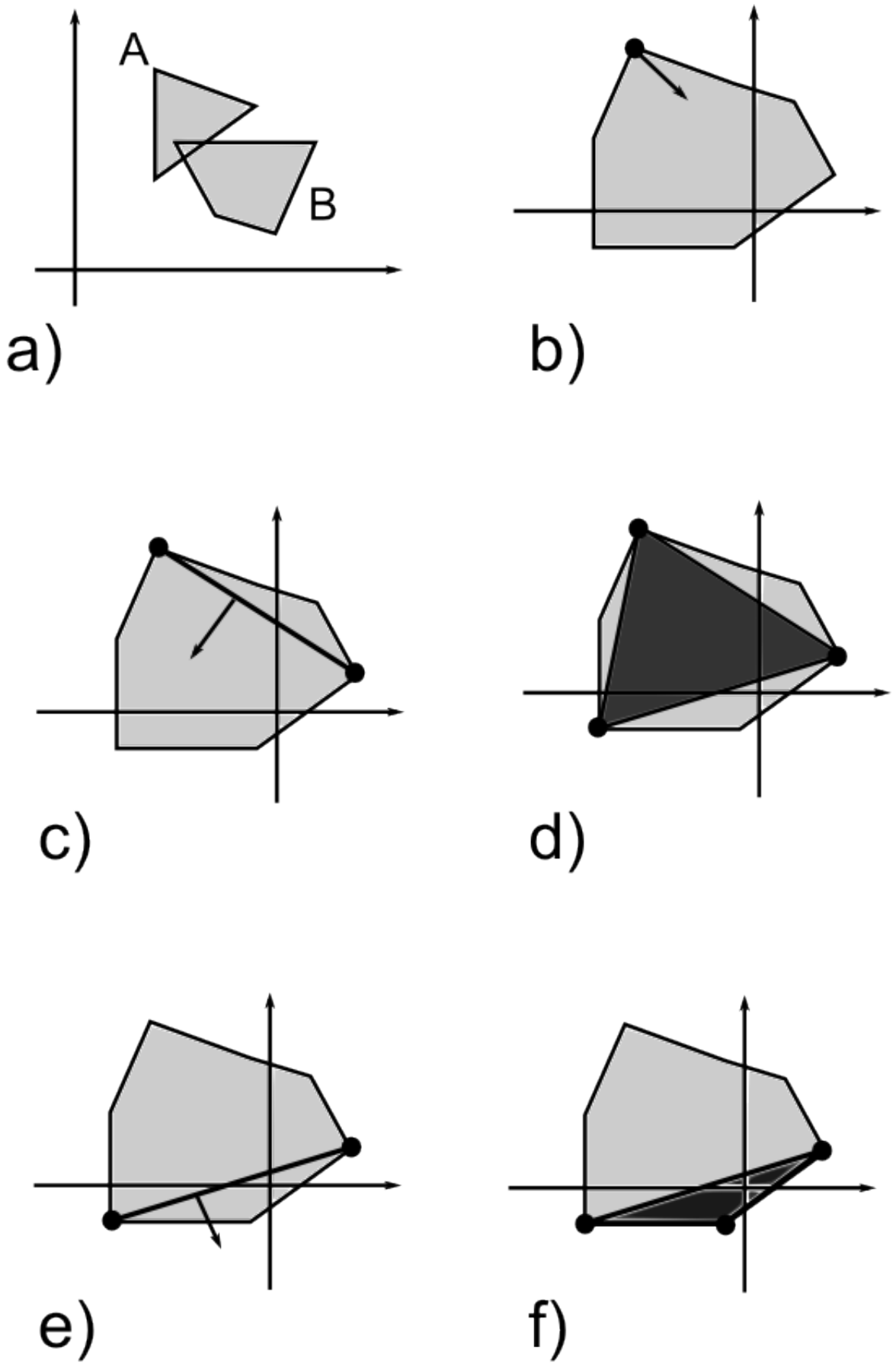
Prije same petlje GJK algoritma, moramo izvršiti inicijalizaciju simpleksa S i vektora potpore $\vec{s}$. Za početnu točku možemo uzeti bilo koju točku na konveksnom omotaču razlike Minkovskog. Vektor potpore $\vec{s}$ se inicijalizira na negativnu vrijednost početne točke. Početnu točku dodajemo u simpleks i započinjemo glavnu petlju GJK algoritma, slika 6.11 b).

Slika 6.11 c). Funkcija potpore je vratila novu točku. Ukoliko vrijedi $A \cdot \vec{s} < 0 \cdot \vec{s}$, prekidamo s izvođenjem algoritma jer sudar ne postoji. Naime, točka A je najudaljenija točka razlike Minkovskog u smjeru $\vec{s}$. Ako je projekcija ishodišta na vektor potpore, $0 \cdot \vec{s}$, veća od projekcije točke A, to znači da je ishodište udaljenije od najudaljenije točke razlike Minkovskog za vektor $\vec{s}$. U protivnom, dodajemo novu točku u simpleks i pozivamo funkciju *Obradi_simpleks*. Zadaća ove funkcije je vratiti skup točaka u čijoj se Voronoievoj regiji nalazi ishodište te izračunati novi vektor potpore. Način na koji se rješava slučaj simpleksa sa dvije točke je opisan u 6.3.1.1.

Slika 6.11 d). Funkcija *Obradi_simpleks* je u prošloj iteraciji vratila nepromijenjeni simpleks i novi vektor potpore. Simpleks nije mijenjan jer se ishodište nalazi u Voronoievoj regiji linije koja tvori postojeći simpleks. Nakon što je funkcija potpore vratila novu točku koja je udaljenija od ishodišta za ovaj vektor potpore, novu točku dodajemo u simpleks i ponovno pozivamo funkciju *Obradi_simpleks.* Način na koji se rješava slučaj simpleksa sa tri točke je opisan u 6.3.1.2.

Slika 6.11 e). Funkcija *Obradi_simpleks* je u prošloj iteraciji vratila samo dvije točke. Razlog za to je što se ishodište nalazi u Voronoievoj regiji linije koja tvori trenutni simpleks. Funkcija potpore vrača novu točku koja je udaljenija od ishodišta te ju dodajemo u strukturu simpleks koja je sada ponovno trokut.

Slika 6.11 f). Funkcija *Obradi_simpleks* javlja da su objekti A i B u sudaru jer se ishodište nalazi unutar trenutnog simpleksa. Glavni GJK algoritam ovime završava.



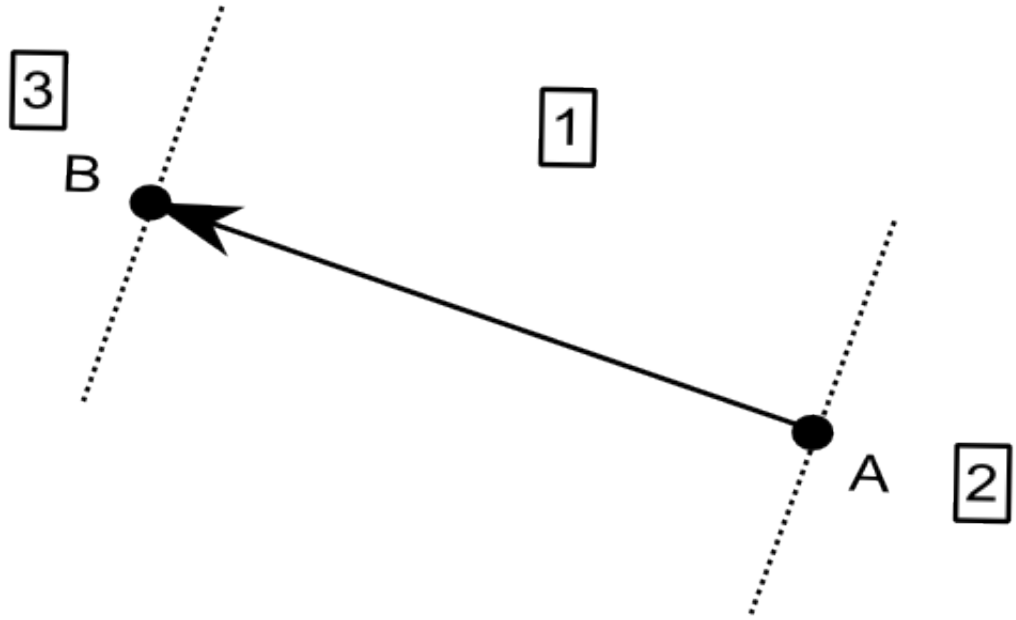
Slika 6.11: GJK - traženje simpleksa.

6.3.1.1 Simpleks sa dvije točke

Kada se simpleks sastoji od dviju točaka one tvore liniju. Točkom A označavamo točku koja je zadnja dodana, dok je točka B već prije bila u simpleksu. Točka A će nam služiti kao referentna točka.

Matematički ispravan pristup bio bi pretražiti sve tri Voronoieve regije za ishodištem. Ako je točka B već bila u simpleksu i ako smo tražeći ishodište iz točke B krenuli u pravcu točke A, tada ishodište ne može biti u Voronoievoj regiji točke B. Ishodište također ne može biti u Voronoievoj regiji točke A jer ju tada ne bismo dodali u simpleks. Jedina Voronoieva regija u kojoj se može nalaziti ishodište je Voronoieva regija broj 1 na slici 6.12, to jest u regiji linije $\overline{AB}$. Simpleks ostaje nepromijenjen a novi vektor potpore je dobivamo izrazom:

$$\vec{s} = \overline{AB} \otimes \overline{AO} \otimes \overline{AB} \tag{6.9}$$



Slika 6.12: Simpleks sa dvije točke
Simpleks se sastoji od dvije točke, A i B te Voronoievih regija 1, 2 i 3.

6.3.1.2 Simpleks sa tri točke

Simpleks sa tri točke nastao je iz simpleksa sa dvije točke. Točak koja je prva dodana u simpleks je točka C. Točka B je sljedeća dodana točka. Ponovno je točka A zadnja dodana točka te također referentna točka. Trokut ima devet Voronoievih regija. Po jednu za svaki vrh i svaki brid, jednu regiju za unutrašnjost trokuta, regiju iznad trokuta i jednu regiju ispod trokuta.

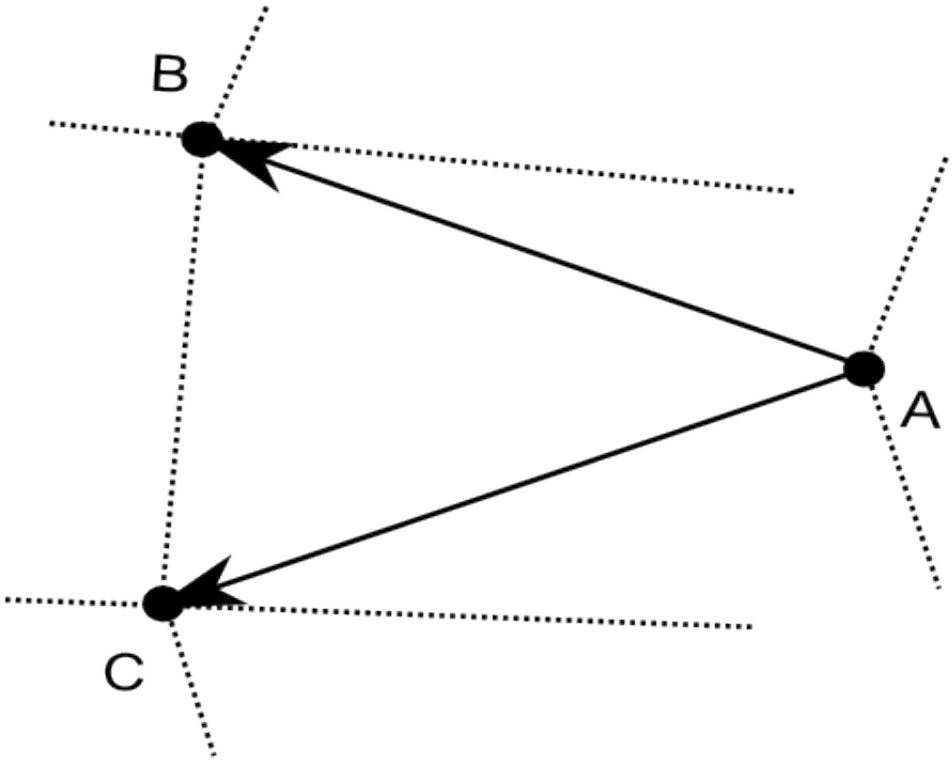
Srećom, ne moramo pretražiti sve regije. Iz simpleksa sa dvije točke znamo da se ishodište nalazi negdje u prostoru omeđenom linijom $\overline{BC}$ i dvjema ravninama okomitim

na liniju $\overline{BC}$ od kojih jedna prolazi kroz točku B, a druga kroz točku C. Također znamo da ishodište nije iza dalje od točke A jer u protivnom ona ne bi bila dodana u simpleks. Na slici 6.13, ishodište mora biti iznad točke B, ispod točke C, desno od linije $\overline{BC}$ i lijevo od točke A.

Ovim promatranjima, sveli smo potragu na samo četiri Voronoieve regije:

- regiju linije $\overline{AC}$
- regiju trokuta ABC za normalu $\overrightarrow{\overline{ABC}} = \overrightarrow{\overline{AB}} \otimes \overrightarrow{\overline{AC}}$
- regiju trokuta ACB za normalu $\overrightarrow{\overline{ACB}} = \overrightarrow{\overline{AC}} \otimes \overrightarrow{\overline{AB}}$
- regiju linije $\overline{AB}$

Algoritam ove pretrage dan je u 6.3.



Slika 6.13: Simpleks sa tri točke

Algorithm 6.3 *Obradi_simpleks* za tri točke

AKO ($(\overrightarrow{AB} \otimes \overrightarrow{ABC}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[B, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AB} \otimes \overrightarrow{AO} \otimes \overrightarrow{AB}$
INAČE AKO ($(\overrightarrow{ABC} \otimes \overrightarrow{AC}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[C, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AC} \otimes \overrightarrow{AO} \otimes \overrightarrow{AC}$
INAČE AKO ($\overrightarrow{ABC} \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[C, B, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{ABC}$
INAČE
    novi simpleks je $[B, C, A]$
    novi vektor potpore je $\vec{s} = -\overrightarrow{ABC}$

6.3.1.3 Simpleks sa četiri točke

Kada se simpleks sastoji od četiri točke, radi se o tetraedar $ABCD$ gdje je trokut BCD baza, a novododana točka A vrh tetraedra. U prijašnjim smo slučajevima odbacili mnogo Voronoievih regija koje bismo inače morali sada pretraživati. Sada nam ostaje sedam regija koje ćemo pretraživati algoritmom 6.4. To su:

- regija trokuta ACD za normalu $\overrightarrow{ACD} = \overrightarrow{AC} \otimes \overrightarrow{AD}$
- regija trokuta ABC za normalu $\overrightarrow{ABC} = \overrightarrow{AB} \otimes \overrightarrow{AC}$
- regija trokuta ADB za normalu $\overrightarrow{ADB} = \overrightarrow{AD} \otimes \overrightarrow{AB}$
- regija linije $\overline{BA}$
- regija linije $\overline{CA}$
- regija linije $\overline{DA}$
- unutrašnjost tetraedra

Algorithm 6.4 *Obradi_simpleks* za četiri točke

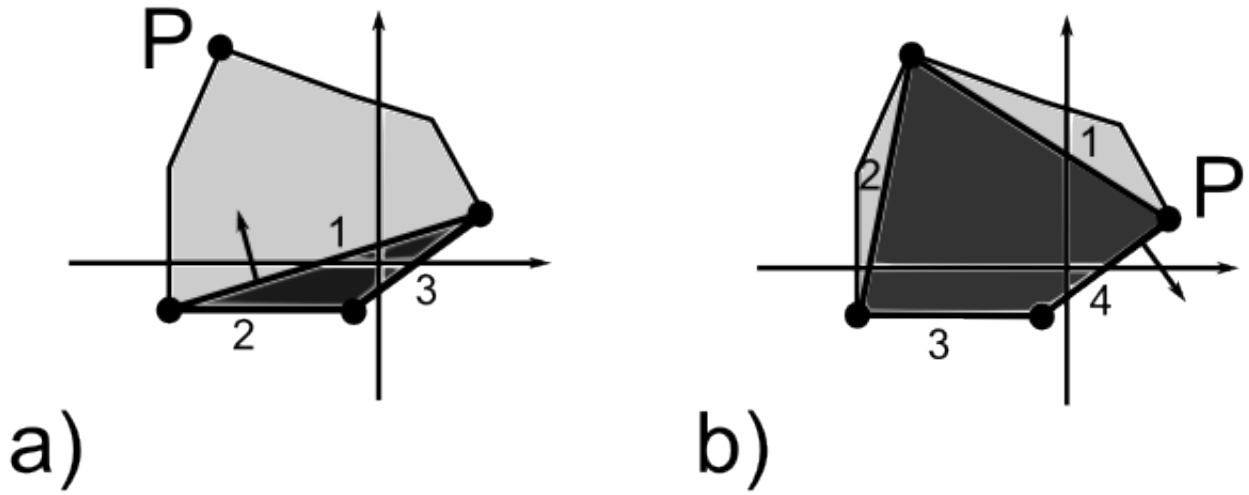
AKO ($\overrightarrow{ABC} \cdot \overrightarrow{AO} > 0$)
  AKO ($(\overrightarrow{ABC} \otimes \overrightarrow{AC}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[C, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AC} \otimes \overrightarrow{AO} \otimes \overrightarrow{AC}$
  INAČE AKO ($(\overrightarrow{AB} \otimes \overrightarrow{ABC}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[B, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AB} \otimes \overrightarrow{AO} \otimes \overrightarrow{AB}$
  INAČE
    novi simpleks je $[C, B, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{ABC}$
INAČE AKO ($\overrightarrow{ACD} \cdot \overrightarrow{AO} > 0$)
  AKO ($(\overrightarrow{ACD} \otimes \overrightarrow{AD}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[D, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AD} \otimes \overrightarrow{AO} \otimes \overrightarrow{AD}$
  INAČE AKO ($(\overrightarrow{AC} \otimes \overrightarrow{ACD}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[C, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AC} \otimes \overrightarrow{AO} \otimes \overrightarrow{AC}$
  INAČE
    novi simpleks je $[D, C, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{ACD}$
INAČE AKO ($\overrightarrow{ADB} \cdot \overrightarrow{AO} > 0$)
  AKO ($(\overrightarrow{ADB} \otimes \overrightarrow{AB}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[B, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AB} \otimes \overrightarrow{AO} \otimes \overrightarrow{AB}$
  INAČE AKO ($(\overrightarrow{AD} \otimes \overrightarrow{ADB}) \cdot \overrightarrow{AO} > 0$)
    novi simpleks je $[D, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{AD} \otimes \overrightarrow{AO} \otimes \overrightarrow{AD}$
  INAČE
    novi simpleks je $[B, D, A]$
    novi vektor potpore je $\vec{s} = \overrightarrow{ADB}$
INAČE detektiran je sudar

Dodatno ubrzanje GJK algoritma možemo postići tako što ćemo pamtiti zadnju točku koju smo dobili pozivom funkcije potpore. Prilikom sljedećeg poziva GJK algoritma, upravo ćemo ovu točku koristiti kao početnu točku. Ukoliko nije došlo do velikih pomaka dvaju objekata između sadašnjeg vremenskog odsječka i prošlog, GJK algoritam će već u prvoj iteraciji vratiti negativan rezultat za sudar tih dvaju objekata. Podatak dobiven u prošlom pozivu algoritma koji se koristi u trenutnom pozivu se naziva svjedokom (*witness*).

6.3.2 EPA algoritam

Nakon što je GJK detektirao sudar možemo pozvati jednostavni algoritam za detekciju dvaju objekata opisan u 4.3. Ako smo GJK koristili za detekciju sudara dvaju konveksnih objekata, za dobivanje normale sudara i dubine penetracije možemo koristiti EPA (*Expanding Polytope Algorithm*) algoritam.

Baš kao i GJK, EPA algoritam koristi razliku Minkovskog. EPA je također iterativni algoritam. Ideja EPA algoritma je u svakoj iteraciji proširivati konveksni podobjekt unutar razlike Minkovskog sve dok granice tog podobjekta ne dotaknu konveksni omotač razlike Minkovskog. Ključ je u tome da se uvijek proširuje ono lice podobjekta koje je najbliže ishodištu. Ovo proširivanje EPA obavlja koristeći istu funkciju potpore kao i GJK. EPA za razliku od GJK algoritma može imati simpleks sa neograničenim brojem točaka. EPA za razliku od GJK algoritma mora imati početni simpleks. To je trokut, ako smo koristili dvodimenzionalnu inačicu GJK algoritma, ili tetraedar, ako smo koristili trodimenzionalni GJK.



Slika 6.14: Primjer izvođenja EPA algoritma.

Slika 6.14 prikazuje primjer izvođenja EPA algoritma nakon što je GJK algoritam na slici 6.11 pronašao sudar.

Na slici 6.14 a) je završni simpleks GJK algoritma. To postaje početni simpleks EPA algoritma. U prvoj iteraciji je brid 1 najbliži pa njegovu normalu koristimo kao novi vektor potpore. Točka P je najudaljenija točka u smjeru novog vektora potpore. Pošto je točka P udaljenija od ishodišta nego što je to brid 1, brid jedan brišemo iz simpleksa i u simpleks dodajemo točku P i nove bridove koje ona tvori.

Slika 6.14 b) prikazuje sljedeću iteraciju algoritma. Najbliži brid ishodištu je sada brid 4. Funkcija potpore za novi vektor potpore vraća točku P. Udaljenost točke P od ishodišta nije veća od udaljenost brida 4 od ishodišta što znači da smo dotakli konveksni omotač razlike Minkovskog. Vektor potpore je normala sudara, a udaljenost ishodišta od brida 4 je dubina penetracije. Projekcija ishodišta na brid 4 je točka sudara.

Algorithm 6.5 EPA algoritam

simpleks S je završni simpleks iz GJK algoritma
PETLJA
    pronađi trokut T koji je najbliži ishodištu
    vektor potpore $\vec{s}$ je normala trokuta T
    točka $P = f_A(\vec{s}) + f_B(-\vec{s})$
    d je udaljenost trokuta T od ishodišta
    AKO ($\vec{s} \cdot P - d < 0$)
        normala sudara je normala trokuta T
        dubina penetracije je d
        točka sudara je projekcija ishodišta na trokut T
    INAČE
        obriši trokut T iz simpleksa
        dodaj u simpleks točku P i nove trokute koje ona tvori

Poglavlje 7

Rezultati

7.1 Implementacija

Simulacija kojom je testiran GJK algoritam rađena je u programskom jeziku C++. Za prikaz animacije je korištena knjižnica GLUT te grafička knjižnica OpenGL. Korišteni trodimenzionalni modeli su u Wavefront formatu i obrađeni su upotrebom MeshLab program za obradu trodimenzionalnih modela. Trodimenzionalni modeli su učitani korištenjem GLM knjižnice [1].

7.2 Usporedba algoritama

U ovom radu je prezentiran GJK algoritam za detekciju sudara dvaju objekata prezentiranih njihovim konveksnim omotačem. Zanima nas kako se GJK algoritam nosi sa primitivnom metodom detekcije sudara dvaju objekata prezentiranih njihovim mrežama poligona. Također nas zanima usporedba GJK algoritma sa korištenjem obujmica.

Primitivnu metodu detekcije sudara u ovom će testu zastupati test za detekciju sudara dvaju trokuta. Od mnogobrojnih metoda koje koriste obujmice, koristiti ćemo jednostavnu metodu obujmica u obliku sfere. Zanima nas kako će se svaki od ova tri načina detekcije sudara nositi sa povećanjem broja poligona kojima su predstavljeni objekti čije sudare provjeravamo.

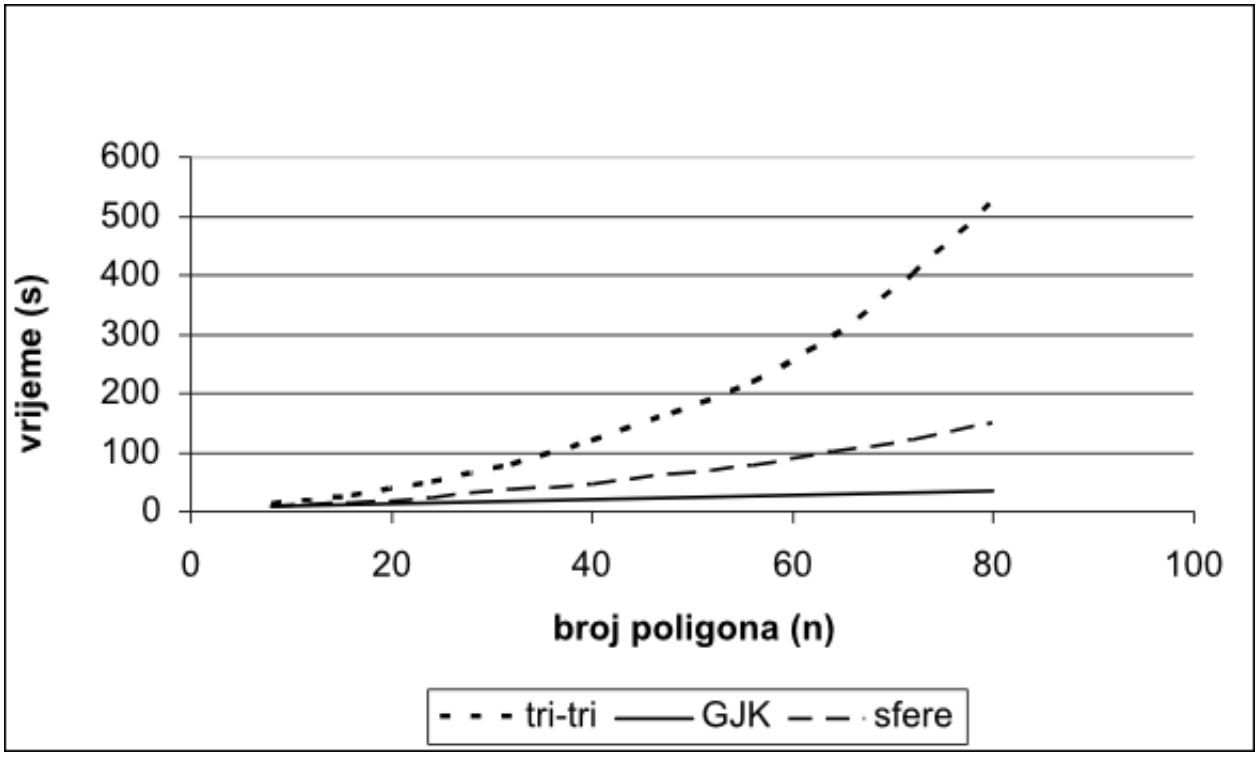
Simulacija sudara čvrstih tijela se odvija u prostoru oblika kocke. Za detekciju sudara tijela sa zidovima kocke koristi se jednostavna provjera s koje strane ravnine zida se nalaze vrhovi objekta koji predstavlja čvrsto tijelo. Da bismo što je moguće više smanjili utjecaj nebitnih varijabli na ovo mjerenje, treba nam simetrično geometrijsko tijelo sa što manjim brojem vrhova. Zbog toga ćemo testiranje vršiti na paru oktaedara. Sam oktaedar predstavlja svoj konveksni omotač, dok ćemo kao obujmicu u obliku sfere koristiti najmanju moguću sferu koja obujmljuje ovaj objekt.

Dva oktaedra su smještena nasumično u prostor simulacije. Ukupan volumen dvaju

[1] http://altuma.ro/opengl/glm.zip

broj poligona	trokut	sfera	GJK	broj poligona	trokut	sfera	GJK
8	11.35	8.55	8.62	48	164.06	64.38	22.38
16	26.31	15.81	11.61	56	216.99	77.55	25.67
24	48.89	23.41	13.67	64	296.23	102.74	29.63
32	79.67	36.65	16.58	72	403.87	121.49	32.07
40	117.54	44.98	19.59	80	524.69	150.34	34.12

Tablica 7.1: Rezultati mjerenja utjecaja broja poligona na vrijeme izvršavanja simulacije. U tablici je dan broj poligona kojim je predstavljeno tijelo i vrijeme potrebno za izvršavanje pojedinog testa.



Slika 7.1: Utjecaj broja poligona na vrijeme izvršavanja simulacije

tijela je jedan posto volumena prostora simulacije. Mjereno je srednje vrijeme potrebno za izvršavanje deset tisuća iteracija simulacije. Vremenski okvir svake iteracije je dvije milisekunde. Rezultati mjerenja su prikazani u tablici 7.1. Slika 7.1 prikazuje graf ovisnosti vremena izvršavanja pojedinog algoritma o broju poligona.

Mjerenja su provedena na mobilnom računalu HP nx6325 sa Mobile AMD Sempron 3500+ procesorom i jednim GB radne memorije sa operacijski sustavom Microsoft Windows XP.

Graf na slici 7.1 prikazuje eksponencijalnu složenost testa sudara dvaju trokuta. To ga čini potpuno neprihvatljivim za korištenje kao jedinog algoritma u interaktivnim simulacijama. Algoritam koji detektira sudare dviju obujmica u obliku sfere te tek tada poziva primitivni trokut-trokut test je dao bolje rezultate. No, i kod njega se vidi gotovo eksponencijalna složenost. Razlog tomu je što sfera ne prianja u potpunosti na zadani objekt te imamo zamjetan broj lažnih sudara. To jest sudara dviju obujmica kada ne postoji sudar dvaju objekata. GJK algoritam je polučio najbolje rezultate. Ovaj algoritam posjeduje

linearnu složenost. Razlog tomu je u korištenju konveksnog omotača koji je najbolje prianjajuća obujmica. Zbog toga je broj lažnih sudara kod GJK algoritma sveden na minimum. GJK algoritam također u potpunosti iskorištava veliku sličnost između dvaju vremenskih odsječaka korištenjem svjedoka što mu u slučaju kada nema sudara između dva čvrsta tijela omogućuje konstantnu složenost.

Zaključak

Simuliranje dinamike čvrstih tijela je važan dio većine današnjih interaktivnih aplikacija. Najveći problem pri simuliranju dinamike čvrstih tijela predstavlja kvalitetna i brza detekcija sudara. Primitivne metode koje se oslanjaju na detekciju sudara trokuta kojima su predstavljeni objekti u sceni su jako osjetljive na porast ukupnog broja poligona. U ovom radu je prezentirano korištenje konveksnog omotača objekta kao način smanjenja ukupnog broja poligona. GJK algoritam, koji je također prezentiran u ovom radu, iskorištava veliku sličnost između dvaju vremenskih okvira kako bi smanjio broj potrebnih testova detekcije sudara. Korištenje GJK algoritma nije dovelo do pada realističnosti same simulacije, ali je dovelo do velikog ubrzanja njenog izvođenja.

Rezultati mjerenja su pokazali da kombinacija konveksnog omotača, kao odlično prianjajuće obujmice, i GJK algoritma, kao metode koja koristi svjedoke, po brzini izvođenja daleko nadmašuje korištenje obujmica u obliku sfere i primitivne metode detekcije sudara dvaju trokuta.

U ovom radu sam prezentirao jednostavan način za implementaciju brze GJK metode. Izbor metoda kojima ćemo se služiti prilikom detekcije sudara dvaju čvrstih tijela ovise o mnogim parametrima te je potrebno veliko iskustvo i znanje za izbor najbolje metode. GJK metoda, uz korištenje konveksnog omotača, pokazala se je kao odličan izbor za detekciju sudara dvaju objekata u simulaciji s velikom sličnošću između dva vremenska okvira.

Bibliografija

[Kulišić02] Petar Kulišić. Mehanika i toplina, 2002.

[Baraff01] David Baraff. An Introduction to Physically Based Modeling: Rigid Body Simulation, 2001.

[Hecker96] Chris Hecker. Rigid Body Dynamics, Game Developer Magazine, 1996./1997.

[Shoemake85] Ken Shoemake. Animating rotation with quaternion curves. Computer Graphics (Proc. SIGGRAPH), svezak 19, stranice 245–254. ACM, Srpanj 1985.

[Hanson05] Andrew Hanson. Visualizing Quaternions, 2005.

[Bishop04] James M. Van Verth, Lars M. Bishop. Essential Mathematics for Games and Interactive Applications, 2004.

[Millington07] Ian Millington. Game Physics Engine Development, 2007.

[Kallay06] Michael Kallay. Computing the Moment of Inertia of a Solid Defined by a Triangle Mesh. Journal of Graphics, GPU and Game Tools, svezak 11, broj 2, 2006.

[Mirtich96] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. Journal of Graphics Tools, svezak 1, broj 2, 1996.

[Eberly09] David Eberly. Polyhedral Mass Properties (Revisited). 2009.

[Ericson05] Christer Ericson. Real-Time Collision Detection, 2005.

[Apsen69] Boris Apsen. Repetitorij više matematike - III dio, 1969.

[Bergen03] Gino van den Bergen. Collision Detection in Interactive 3D Enviroments, 2003.

[Weisstein] Eric Weisstein. „Runge-Kutta Method.” From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Runge-KuttaMethod.html

[Devillers02] Olivier Devillers, Philippe Guigue. Faster Triangle-Triangle Intersection Tests, 2002.

[Tropp05] Oren Tropp, Ayellet Tal, Ilan Shimshoni. A fast triangle to triangle intersection test for collision detection, 2005.

[Moller97] Tomas Moller, A Fast Triangle-Triangle Intersection Test, 1997.

[Welzl91] Emo Welzl. Smallest Enclosing Disks (Balls and Elipsoids), 1991.

[O'Rourke98] Joseph O'Rourke. Computational Geometry in C, 1998.

[Eberly01] David Eberly. Intersection of Convex Objects: The Method of Separating Axes, 2001.

Sažetak

(Detekcija sudara čvrstih tijela)

Kvalitetna simulacija dinamike čvrstih tijela važan je dio današnjih interaktivnih simulacija, od računalnih igara do industrijskih i vojnih simulatora. Vremenski najzahtjevnija komponenta simulacije je detekcija sudara između dva čvrsta tijela. Kako je većina računalnih trodimenzionalnih modela objekata dana u vidu mreže poligona, osnovna detekcija sudara sastoji se od ispitivanja sudara svih trokuta mreže poligona jednog objekta sa svim trokutima mreže trokuta drugog objekta. Naprednim metodama detekcije sudara nastoji se smanjiti broj ovih skupih provjera. U ovom je radu prezentirana metoda detekcije sudara dvaju čvrstih tijela upotrebom GJK algoritma nad konveksnim omotačima tih dvaju objekata. Opisan je vektorski pristup GJK metodi koji bitno pojednostavljuje implementaciju bez gubitka kvalitete same metode.

Ključne riječi: čvrsto tijelo, dinamika čvrstih tijela, detekcija sudara, razrješavanje sudara, obujmice, konveksni omotač, GJK algoritam

Abstract

(Rigid Body Collision Detection)

The quality simulation of rigid body dynamics is an important part of modern day interactive simulations, ranging from computer video games to industrial and military simulators. The most time-consuming part of a simulation is collision detection between two rigid bodies. Since most computer 3D models of objects are given as a polygon mesh, basic collision detection consists of checking for collision between all triangles of one object's polygon mesh and all triangles of the other object's polygon mesh. Advanced collision detection methods are designed to reduce the number of these expensive checks. This paper presents the method of collision detection of two rigid bodies by using the GJK algorithm on the bodies' convex hulls. The paper describes a vector approach to the GJK method that significantly simplifies implementation without quality loss of the method itself.

Key words: rigid body, rigid body dynamics, collision detection, collision response, bounding volumes, convex hull, GJK algorithm