SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Implementacija detekcije sudara

Rijeka, rujan 2016.

Vjera Turk
0069064924

Zahvala

Zahvaljujem mentoru doc.dr.sc. Jerku Škifiću na korisnim savjetima, strpljenju i podršci tijekom pisanja ovoga rada te svim profesorima i asistentima na prenesenom znanju i vještinama. Veliko hvala obitelji, prijateljima i kolegama na pomoći i podršci tijekom proteklih godina studija.

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Implementacija detekcije sudara

Mentor: doc.dr.sc. Jerko Škifić

Rijeka, rujan 2016.

Vjera Turk
0069064924

Umjesto ove stranice umetnuti zadatak za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, rujan 2016.

_____

Vjera Turk

Sadržaj

Popis slika

Popis tablica

Poglavlje 1

Uvod

Detekcija sudara odnosi se na računalni problem detektiranja presjeka dvaju ili više objekata u pokretu. Algoritmi za određivanje presjeka objekata u pokretu određuju sjeku li se dva objekta te kada i gdje dolazi do njihovog kontakta. Koriste se kod fizikalno temeljenih animacija, interaktivnog gibanja korisnika (igre, virtualna stvarnost engl. *virtual reality*), u robotici i pri izradi virtualnih prototipova, kako bi se između ostalog izbjeglo prolaženje objekata jednih kroz druge, postiglo gibanje jednog objekta po površini drugog (detekcija kontakta), simulirao odziv na sudar odnosno (elastično/plastično) odbijanje i proračun povratne sile te kako bi se postiglo povećanje realnosti doživljaja.

Aplikacije koje imaju izuzetne zahtjeve za učinkovitim rješavanjem sudara u realnom vremenu. Najčešće se za primjer uzimaju računalne igre. Uz njih slične, a često i veće zahtjeve imaju sustavi za izračun osjeta dodira (engl. *haptic interaction systems*), čestične simulacije, kirurške simulacije (simulacije operativnih zahvata) i druge virtualne simulacije stvarnosti. One podrazumijevaju ispitivanje i određivanje sudara u predstojećem vremenskom trenutku, tj. sav proračun je potrebno obaviti prije renderiranja svake sljedeće slike odnosno iscrtavanja scene. U tom slučaju vrlo je važna učinkovitost svih struktura podataka i algoritma za rješavanje problema detekcije sudara. Idealno rješenje za pojedini slučaj je ono koje osigurava optimalan omjer između dva najvažnija neproporcionalna faktora - brzine i točnosti.

Različite primjene, oblici i ostale osobine objekata čija se dinamika simulira kao i

sveprisutni razvoj snage i mogućnosti računalnog sklopovlja, rezultirali su razvojem različitih metoda za detekciju sudara.

Uz osobine objekata kao što su konkavnost, broj poligona i veličina, sam broj objekata na sceni igra ulogu pri odabiru pristupa implementaciji detekcije sudara. Velik broj objekata na sceni uvodi potrebu za uvođenjem rekurzivnog particioniranja prostora (engl. *spatial partitioning*). Podjelom prostora i opisanim volumenima (engl. *bounding volumes*) postiže se optimizacija ranom eliminacijom parova objekata koji nisu u koliziji, odnosno koje nije potrebno detaljnije provjeravati.

Nakon detekcije sudara najčešće je potrebno simulirati i reakciju odnosno odziv na sudar. U fizikalnim simulacijama krutih tijela, radi se o odbijanju sudarenih objekata.

U ovom radu naglasak je stavljen na detekciju sudara velikog broja dinamičkih objekata u realnom vremenu i elastično odbijanje. Sljedeće poglavlje govori o fazama detekcije kolizije, algoritmima i faktorima koji utječu na njihov odabir u pojedinoj fazi. Treće poglavlje govori o fizikalnoj prirodi elastičnog sudara krutih tijela. U četvrtom i petom poglavlju dan je pregled implementiranog rješenja uz analizu i opis korištenih algoritama.

Poglavlje 2

Detekcija Sudara

2.1 Definicija

Detekcija sudara je složeni problem i predmet je istraživanja u mnogim granama osim interaktivne 3D grafike, uključujući robotiku, fizikalne simulacije i računalnu geometriju [1]. Pojam *intersection detection* koji se prevodi kao detekcija preklapanja odnosi se na utvrđivanje preklapanja prostora koji zauzimaju neka dva objekta, bilo u 2D ili 3D prostoru. Detekcija preklapanja objekata u pokretu naziva se detekcija sudara [2]. U širem smislu, detekcija sudara podrazumijeva utvrđivanje sudara (engl. *collision detection*) i određivanje sudara (engl. *collision determination*). Utvrđivanje sudara (odnosno detekcija sudara u užem smislu) odnosi se na postupke kojima dobivamo jednoznačan odgovor je li došlo do presijecanja ili ne, a određivanje sudara odnosi na određivanje točnog trenutka i točaka kontakta [3],[4]. Nakon detekcije sudara slijedi faza odziva na sudar (engl. *collison response*) čija implementacija ovisi o aplikaciji u kojoj se primjenjuje [5].

2.2 Velik broj objekata

Kada se radi o detekciji sudara velikog broja objekata od kojih se dio ili svi gibaju, potrebno je ustanoviti parove objekata koji se sudaraju. Ako se radi o n objekata koji se gibaju, tada moramo provjeriti sudara li se prvi objekt sa ostalih *(n-1)* objekata (ne moramo provjeravati hoće li se objekt sudariti sam sa sobom), drugi objekt sa ostalih *(n-2)* odnosno ne uračunavaju se provjere s prethodim objektima koje smo već prebrojili. Kada se tako nastavi, ukupan broj parova je $(n-1)+(n-2)+(n-3)\cdots+1$ i računa se prema formuli (2.1) [3]

$$broj\_parova = \frac{n \cdot (n-1)}{2} \tag{2.1}$$

Ako je na sceni N objekata u pokretu, i M statičnih objekata broj parova računa se prema izrazu (2.2)

$$broj\_parova = \binom{N}{2} + N \cdot M \tag{2.2}$$

Iz navedenog proizlazi da je, za konfiguraciju N objekata, najgori slučaj vremenske složenosti bilo kojeg algoritma za detekciju sudara $O(N^2)$, gdje je N broj objekata.[5]

2.2.1 Vremenska i geometrijska koherentnost

Vremenska koherentnost (engl. *temporal coherence*) je svojstvo da se stanje aplikacije ne mijenja znatno između uzastopnih trenutaka ili slika (engl. *simulation frame*). Objekti se gotovo neznatno pomiču iz slike u sliku. Taj gotovo neznatni pomak objekata u vremenu istovremeno rezultira geometrijskom koherencijom (engl.*geometric coherence, spatial coherence*) jer se posljedično ni prostorni odnosi objekata znatno ne mijenjaju.[5]

Zahvaljujući ovom svojstvu, performanse algoritma mogu se poboljšati. Složenost u najgorem slučaju i dalje ostaje $O(N^2)$ ali performanse algoritma mogu se znatno poboljšati ako se iz slike u sliku pamti prethodno stanje.[1]

2.2.2 Sweep and Prune algoritam

Jedan od najranije razvijenih algoritama (za problem svih parova) je Sweep and Prune algoritam. Ideja iza Sweep and Prune algoritma je proći ravninom uzduž volumena prostora, izdvajajući one parove objekata koji istodobno presijecaju ravninu.

Oni parovi objekata koji ne presijecaju ravninu istodobno, ne mogu biti u kontaktu, odnosno ne mogu se presijecati te ih možemo eliminirati. U praksi, kod implementacije, objekti se projiciraju na koordinatne osi, a rezultat projekcije su intervali uzduž osi. Intervali koji se preklapaju označavaju moguće presijecanje objekata čije su oni projekcije.

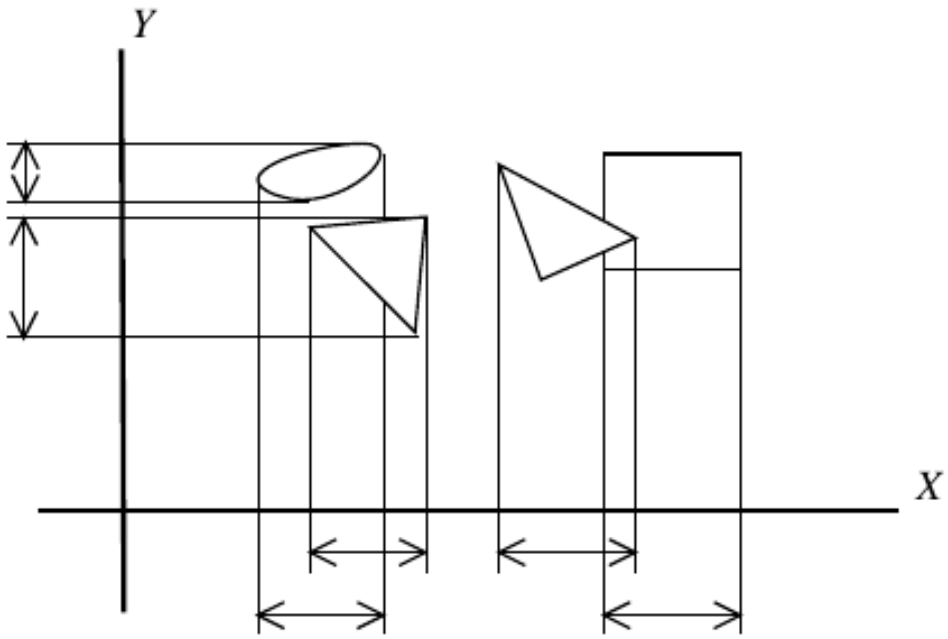
Utvrđivanje preklapanja intervala je proces u 2 koraka:

1. Sortiranje liste intervala u uzlaznu listu koristeći minimume intervala.

2. Proći listom u uzlaznom smjeru testirajući trenutni sa slijedećim intervalom u listi. Ako naiđemo na krajnju točku nekog intervala na većoj poziciji od početne točke slijedećeg intervala, intervali se preklapaju.

Sweep and Prune algoritam također se naziva redukcija dimenzije, jer reducira dimenziju u kojoj se parovi objekata uspoređuju. Vidi sliku 2.1

Dodatne, složenije provjere treba raditi samo na parovima objekata koji se preklapaju u sve tri ravnine. Složenost ovog algoritma uključuje sortiranje 3 liste intervala i testiranje preklapanja intervala u sve tri liste. U općem slučaju, složenost sortiranja liste je $O(n \log n)$, a testiranje preklapanja $O(n^2)$, za najgori slučaj. Zbog vremenske i geometrijske koherentnosti, kod ovog algoritma liste su uvijek gotovo sortirane.



Slika 2.1 Redukcija dimenzije
Objekti su projicirani na x-os. Označeni su intervali presjeka uzduž osi x objekata u mogućem preklapanju. Da bi se objekti preklapali, moraju se preklapati i njihove projekcije na y os što ne vrijedi za lijevi par objekata.

2.3 Podjela u faze

Univerzalna rješenja za detekciju sudara su višeslojna. Problem detekcije sudara između mnogo objekata i problem detekcije sudara između dva kompleksna objekta, dva su odvojena problema s odvojenim rješenjima. U literaturi ta dva problema često se nazivaju problem svih parova (engl. *all-pair problem*) i problem točnih (konkretnih) objekata (engl. *exact object problem*). Još jedan aspekt detekcije sudara tiče se određivanja koji geometrijski dijelovi (engl. *feature*) objekata su u kontaktu. Pod geometrijskim dijelom podrazumijeva se: dio objekta, sastavljen od jednog ili više poligona, brid, ili točka. Softver koji implementira generalno (univerzalno) rješenje detekcije sudara tipično koristi kolekciju algoritama kako bi pokrio svaki od navedena tri aspekta. [1]

Ova tri problema rješavaju se zasebnim algoritmima. Budući da se pojavljuju i rješavaju u vremenskom slijedu, detekcija sudara dijeli se u faze. Riječ je o širokoj i uskoj fazi. Neki autori izdvajaju i srednju fazu (engl. *middle phase*) dok drugi poput Christera Ericsona u svojoj knjizi Real-Time Collision Detection [3] problematiku kojom se bavi srednja faza uvrštavaju u široku.

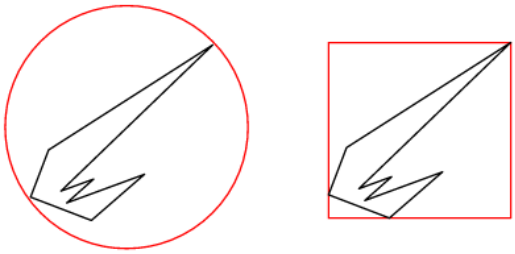
2.3.1 Široka faza

Uloga algoritama široke faze je brza i učinkovita eliminacija parova objekata koji se ne sudaraju primjenom strategije *"podjeli pa vladaj"* (engl. *divide and conqure*). Rezultat ove faze je skup parova u potencijalnom sudaru koji zahtijevaju dodatnu provjeru algoritmima uske faze. Prema [6] možemo ih svrstati u četiri glavne skupine: pristup grube sile, prostorno particioniranje, topološke metode i kinematički pristup.

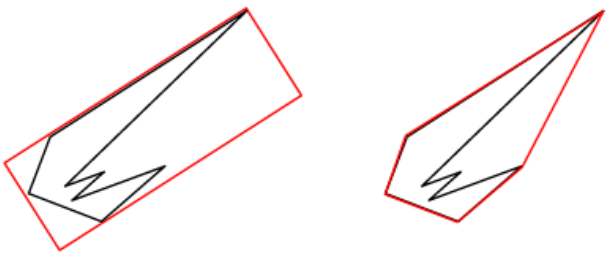
Pristup grube sile (engl. *brute force aproach*) temelji se na direktnoj provjeri svih parova objekata, svih parova opisanih volumena oko objekata ili svih parova poligona na sceni. Ovi testovi vrlo su "iscrpni", a složenost ovakvih testova u svakom slučaju je $O(N^2)$. Postoji mogo vrsta opisanih volumena, a najpoznatiji i najčešće korišteni su sfere, *Axis Aligned Bounding Box (AABB)*, *Oriented Bounding Box (OBB)* i konveksna ljuska (engl. *convex-hull*). Prikazani su na slici 2.2. Opisani volumeni aproksimiraju volumen objekta i omogućuju mnogo jednostavniju provjeru od one koja bi uključivala sve njihove poligone.

Prostorno particioniranje (engl. *spatial partitioning*) temelji se na načelu da dva objekta u udaljenim dijelovima prostora zasigurno nemogu u predstojećem vremenskom koraku. Ova skupina nudi nekoliko metoda za podjelu prostora u manje jedinične ćelije (engl. *unit cells*): obična mreža (engl. *regular grid*), hijerarhijska mreža (engl. *hierarchical grid*), oktalno stablo (engl. *octree*) prikazano na slici 2.3, *loose octree*, kvadratno stablo (engl.*quadtree*), BSP stablo (engl. *Binariy space Partitioning (BSP)*), k-d stablo i voxeli. Ovim metodama upareni će biti oni objekti koji dijele jediničnu ćeliju.

Topološke metode (engl. *topological methods*) bazirane su na poziciji objekta u odnosu na druge objekte. Par objekata se zanemaruje kada se objekti udalje jedan od drugoga. Najpoznatija metoda ovog pristupa je *Sweep and Prune*.



(a) Sfera i AABB



(b) OBB i konveksna ljuska

Slika 2.2 Najčešće korišteni opisani volumeni (2D)

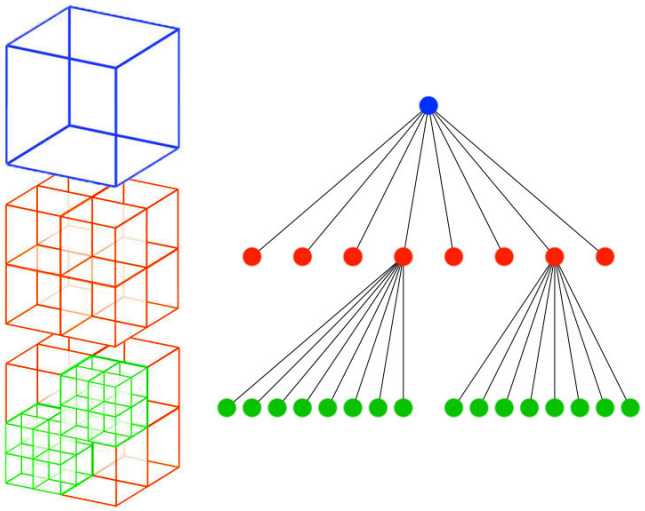
Kinematički pristup uzima u obzir gibanje objekata. Ako se objekti međusobno odmiču, neće se sudariti.

Širokom fazom riješen je problem svih parova (engl. *all-pair problem*). [7]

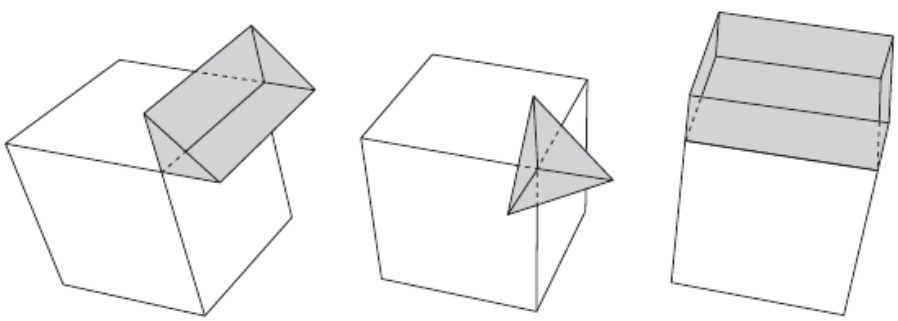
2.3.2 Uska faza

Kao što je prethodno spomenuto, rezultat algoritama široke faze su parovi objekata koji su potencijalno u sudaru. Taj skup parova glavni je ulazni argument algoritama uske faze. Algoritme uske faze također možemo podijeliti u četitri skupine: *Feature based* algoritmi [6],

***Feature based* algoritmi**rade s licima, bridovima i vrhovima (prmitivima) objekata. Prvi puta se takav algoritam spominje 1991. kod Lin-Canny pristupa (ili engl. *Voronoi Marching*) kod kojeg se prostor oko objekta dijeli na Voronoi regije koje omogućavaju da se detektira par najbližih primitiva (lica, brodova, vrhova) između poliedara. Voronoi regije prikazane su na slici 2.4



Slika 2.3 Oktalno stablo



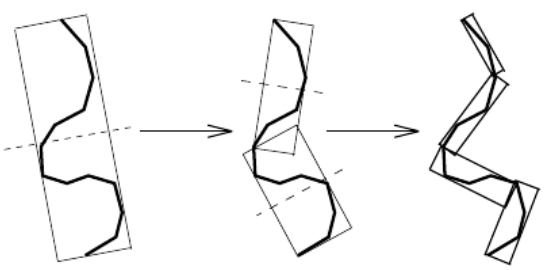
Slika 2.4 Voronoi regije brida, vrha i stranice kocke

Simpleks-bazirani algoritmi(engl. *simplex-based*) među kojima je najpoznatiji Gilbert-Johnson-Keerthi (GJK) algoritam koriste Minkowski razliku poliedara. Dva konveksna objekta su u koliziji samo ako njihova Minkowski razlika sadrži ishodište. SAT metoda (*Separating Axis Theorem*) 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. SAT metodu

koriste algoritmi za posebne slučajeve. Ona radi samo kada su u pitanju poliedri s vrhovima, bridovima i licima ali ne i implicitno zadani objekti. Implicitno zadani objekti su primjerice sfera, cilindar, stožac... Za razliku od SAT algoritma, GJK je algoritam opće namjene i može poslužiti za detekciju sudara eksplicitno zadanih i implicitno zadanih objekata.

Image space-based **algoritmi** rade oslanjajući se na procesor grafičke kartice. Rasteriziraju objekte kako bi proveli 2D ili 2.5D testove preklapanja u prostoru zaslona. Rasterizacija je pretvaranje podataka iz vektorskog u rasterski format, odnosno oostupak transformacije kontinuiranih opisa grafičkih primitiva u diskretne opise pomoću piksela [8].

Algoritmi temeljeni na opisanim volumenima Algoritmi temeljeni na opisanim volumenima (engl *bounding volume-based algorithms*) vrlo su često upotrebljavani jer značajno unapređuju performanse. Oko složenog objekta tvori se hijerarhija opisanih volumena (engl. *BVH*) koja jednostavne opisane volumene (primjerice OBB) raspoređuje u stablo (binarno, kvadratno...) kako bi se samnjio broj testova. Nastajanje BVH prikazano je na slici 2.5. BVH nije dobro rješenje za objekte koji se deformiranju jer uslijed deformacija potrebno bi bilo konstantno mijenjati i hijerarhiju. [7]



Slika 2.5 BVH sastavljena od OBB volumena

Poglavlje 3

Reakcija na sudar

Faza sudara jedna je od najzahtjevnijih faza simulacije općenito. Kao što je spomenuto u uvodu, možemo ju podijeliti u tri faze od kojih je posljednja odziv ili reakcija na sudar (engl. *collison response*)[5]. Što će se sve u ovoj fazi odvijati ovisi o cilju aplikacije. U fizikalnim simulacijama težit će se realnom prikazu stvarnog svijeta. U obzir se moraju uzeti fizikalni zakoni. Elastični sudar je sudar kod kojeg je osim količine gibanja, očuvana i mehanička energija, dok je kod neelastičnog sudara očuvana samo količina gibanja. Obzirom na smjer gibanja čestica prije i poslije sudara, elastični sudar se dijeli na centralni i necentralni sudar.

3.1 Treći Newtonow zakon

Za vrijeme sudara koji traje vrlo kratko, čestice djeluju jedna na drugu silama. Prema 3. Newtonovom zakonu, sila kojom prva čestica djeluje na drugu $\vec{F}_{12}$, jednaka je po iznosu, a suprotna po smjeru sili $\vec{F}_{21}$, kojom druga čestica djeluje na prvu:

$$\vec{F}_{12} = -\vec{F}_{21} \tag{3.1}$$

3.2 Zakon očuvanja količine gibanja

Čestice unutar sustava mogu djelovati jedna na drugu unutrašnjim silama, a tijela izvan sustava mogu djelovati na sustav vanjskim silama. Ako nema vanjskih sila, ili se njihova djelovanja međusobno poništavaju tako da im je rezultanta nula, kažemo da je sustav izoliran ili zatvoren. Za izolirani sustav vrijedi zakon očuvanja količine gibanja.

3.2.1 Impuls sile

Impuls sile je umnožak sile i vremena u kojem ta sila djeluje. Brzine čestica prije sudara označene su s $\vec{v_1}$ i $\vec{v_2}$ a nakon sudara $\vec{v_1}'$ i $\vec{v_2}'$ Količina gibanja prve čestice prije sudara je $m_1\vec{v_1}$, a nakon sudara je $m_1\vec{v_1}'$. Nastala promjena količine gibanja jednaka je primljenom impulsu sile $\vec{I_1}$.

$$\vec{I_1} = m_1\vec{v_1}' - m_1\vec{v_1} \tag{3.2}$$

Isto vrijedi i za drugu česticu: količina gibanja druge čestice je $m_2\vec{v_2}$, a nakon sudara $m_2\vec{v_2}'$, a nastala je primjenom impulsa sile $\vec{I_2}$.

$$\vec{I_2} = m_1\vec{v_2}' - m_1\vec{v_2} \tag{3.3}$$

Zbog ranije spomenutog 3. Newtonovog zakona su i primljeni impulsi sile jednaki po iznosu, a suprotnog smjera $\vec{I_1} = -\vec{I_2}$. Slijedi:

$$m_1\vec{v_1}' - m_1\vec{v_1} = -(m_2\vec{v_2}' - m_2\vec{v_2}) \tag{3.4}$$

$$m_1\vec{v_1} + m_2\vec{v_2} = m_1\vec{v_1}' + m_2\vec{v_2}' \tag{3.5}$$

Kod sudara objekta sa zidom, možemo pretpostaviti da je masa zida beskonačna. Sudar sa zidom je idealna refleksija. Brzina objekta je očuvana, a kut između upadnog i kuta odbijanja je 90°.

3.3 Centralni elastični sudar

Kod centralnog elastičnog sudara tijela se gibaju po istom pravcu odnosno, za centralni elastični sudar vrijedi: $(\vec{v_1} - \vec{v_1}')(\vec{v_1} + \vec{v_1}' - \vec{v_2} - \vec{v_2}') = 0$. Ako je masa prvog tijela veća od mase drugog tijela, tijela će se odbiti jedno od drugog i nastavit će se gibati u suprotnom pravcu drugačijim brzinama. Prvo tijelo će usporiti dok će drugo tijelo ubrzati zbog zakona očuvanja količine gibanja.

3.4 Necentralni elastični sudar - sudar u više dimenzija

Kada se tijela ne gibaju po istom pravcu moramo ih promatrati u dvije ili tri dimenzije. Kada dolazi do sudara u više dimenzija, impuls je očuvan u svakoj od njih zasebno. Zakon očuvanja energije dodaje sustavu još jednu jednadžbu koja povezuje masu i brzinu (3.8). Sustav jednadžbi koji opisuje vezu između količine gibanja prije i poslije sudara u dvije dimenzije:

$$m_1 v_{1x} = m_1 v_{1x}' + m_2 \Delta v_{2x}' \tag{3.6}$$

$$m_1 v_{1y} = m_1 v_{1y}' + m_2 \Delta v_{2y}' tan(\theta) \tag{3.7}$$

$$\frac{m_1}{2}({v_{1x}}^2 + {v_{1y}}^2) = \frac{m_1}{2}({v_{1x}}'^2 + {v_{1y}}'^2) + \frac{m_2}{2}({\Delta v_{2x}}'^2(1 + \tan(\theta)) \tag{3.8}$$

θ je kut između vektora konačne brzine druge loptice i osi x. a je $\tan(\theta)$. Slijedi da su nove brzine loptica:

$$\Delta v_{2x}' = \frac{2(v_{1x} + v_{2x} + a(v_{1y} - v_{2y}))}{(1 + a^2)\frac{m_2}{m_1}} \tag{3.9}$$

$$v_{2x}' = v_{2x} + \Delta v_{2x}' \tag{3.10}$$

$$v_{2y}' = v_{2y} + a\Delta v_{2x}' \tag{3.11}$$

$$v_{1x}' = v_{1x} + \frac{m_2}{m_1}\Delta v_{2x}' \tag{3.12}$$

$$v_{1y}' = v_{1y} - a\frac{m_2}{m_1}\Delta v_{2x}' \tag{3.13}$$

U tri dimenzije prisutan je još jedan kut φ. x i y komponente vektora brzine mogu biti izražene preko z i ova dva kuta. Promatramo brzine loptica u odnosu jedne na drugu. Pretpostavimo da je početna brzina druge loptice 0 . Prva loptica kreće iz ishodišta po osi z prema njoj razlikom njihovih brzina Δv_2. Polazne jednadžbe su:

$$m_1 v_{1z} = m_1 v_{1_z}' + m_2 \Delta v_{2_z}' \tag{3.14}$$

$$m_1 v_{1y} = m_1 v_{1_y}' + m_2 \Delta v_{2_z}' \tan(\theta)\sin(\varphi) \tag{3.15}$$

$$m_1 v_{1x} = m_1 v_{1_x}' + m_2 \Delta v_{2_z}' \tan(\theta)\cos(\varphi) \tag{3.16}$$

$$\frac{m_1}{2}({v_{1x}}^2 + {v_{1y}}^2 + {v_{1z}}^2) = \frac{m_1}{2}({v_{1x}}'^2 + {v_{1y}}'^2 + {v_{1z}}'^2) + \frac{m_2}{2}({\Delta v_{2z}}'^2(1+\tan^2(\theta)) \tag{3.17}$$

Rješavanjem jednadžbi dobiju se sljedeći izrazi za brzine u svakom od smjerova:

$$v_{1_z}' = v_{1z} - \frac{m_2}{m_1}{\Delta v_{2_z}'}^2 \tag{3.18}$$

$$v_{1_y}' = v_{1y} - \frac{m_2}{m_1}{\Delta v_{2_z}'}^2 \tan(\theta)\sin(\varphi) \tag{3.19}$$

$$v_{1_x}' = v_{1x} - \frac{m_2}{m_1}{\Delta v_{2_z}'}^2 \tan(\theta)\cos(\varphi) \tag{3.20}$$

Rješavanjem sustava dobije se izraz (3.21)

$$\Delta v_{2_z}' = 2\frac{v_z + \tan(\theta)(\cos(\varphi)v_{1x} + \sin(\varphi)v_{1y})}{(1+\tan(\theta))(1+\frac{m_2}{m_1})} \tag{3.21}$$

[9][10][11][12]

3.5 Prikaz vektorima

x_1 i x_2 su pozicije središta 2 tijela u trenutku sudara. $\langle\rangle$ zagrade označavaju skalarni produkt 2 vektora.[13]

$$v_1' = v_1 - \frac{2m_2}{m_1+m_2}\frac{\langle v_1 - v_2, x_1 - x_2\rangle}{||x_1 - x_2||^2}(x_1 - x_2) \tag{3.22}$$

$$v_2' = v_2 - \frac{2m_1}{m_1+m_2}\frac{\langle v_2 - v_1, x_2 - x_1\rangle}{||x_2 - x_1||^2}(x_2 - x_1) \tag{3.23}$$

Poglavlje 4

Pregled rješenja

Za izradu računalnog programa korišteno je softversko sučelje i standard Open Graphics Library (OpenGL) te alat OpenGL Utility Toolkit (GLUT). Program je pisan u C++ jeziku u razvojnom okruženju KDevelop.

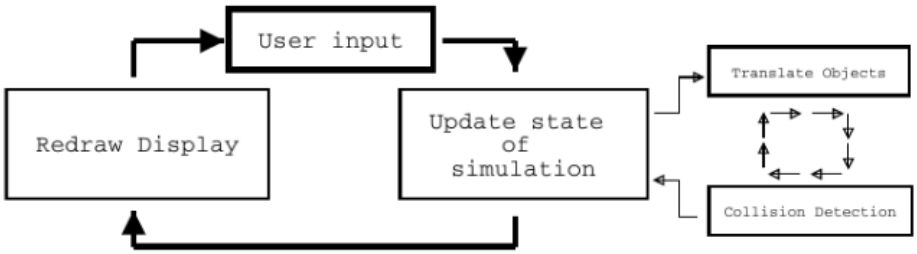
Riješenje se sastoji od tri dijela. Prvi prikazuje elastični centralni i necentralni sudar dvije loptice. Drugi prikazuje sudar većeg broja loptica primjenom Brute Force algoritma odnosno provjere sudara svake loptice sa svakom i svake loptice sa svakim zidom. Treći primjer prikazuje sudare velikog broja loptica smještenih u oktalno stablo.

4.1 Tijek simulacije

Na slici 4.1 prikazan je tok interaktivne grafičke aplikacije. U programskom kodu rješenja iscrtavanje scene (na slici Redraw Display, u kodu funkcija void display()) i ponovno pokretanje ažuriranja (engl. *update*) događa se svakih TIMER_MS milisekundi (funkcija glutTimerFunc(TIMER_MS, update, 0) u mainu). U međuvremenu program "ne miruje" već se ažurira. U manjim vremenskim intervalima vrši se translacija loptica za iznos njihovog vektora brzine pomnoženog s dt ($ball- > pos+ = ball- > v * dt$) i simulirano djelovanje gravitacijske sile, odnosno umanjenje y komponente vektora brzine za iznos jednak umnošku dt i gravitacijske konstante ($ball- > v- = Vec3f(0, GRAVITY * TIME\_BETWEEN\_UPDATES, 0)$. Po-

tom, slijedi detekcija sudara i reakcija na sudar neposredno nakon svake detekcije.

Po isteku vremena TIMER_MS iscrtava se scena u zatečenom stanju (definirana u display() funkciji) i proces ažuriranja započinje ponovno.



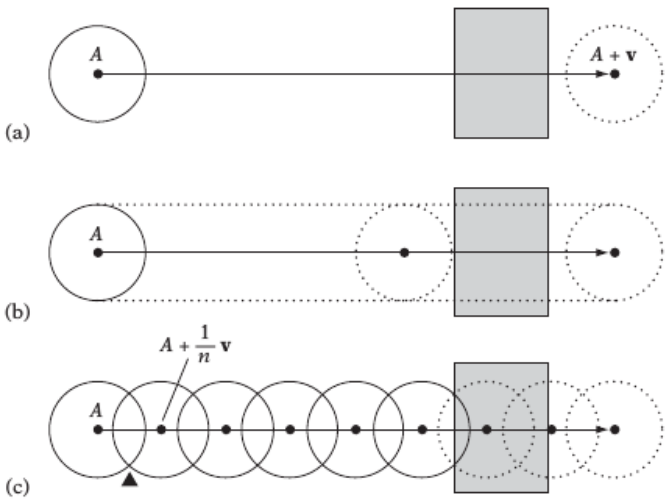
Slika 4.1 Tok (engl. Control Flow) interaktivne 3D grafičke aplikacije

4.1.1 *Tunneling*

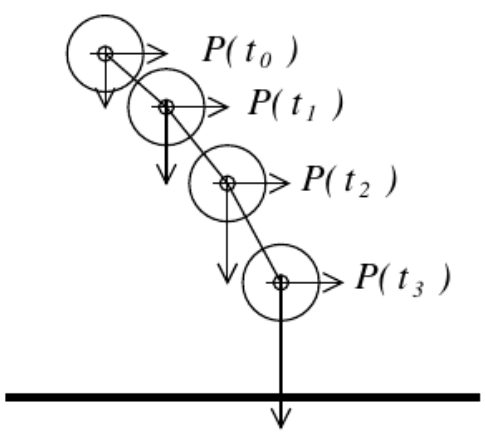
Pojam *Tunneling* označava problem koji je javlja kada se detekcija sudara provjerava u prevelikim vremenskim razmacima, odnosno dovoljno velikim da se zbog njih neki sudari ne zabilježe. Tri primjera možemo vidjeti na slici 4.2

Također, problem mogu stvarati pomaci objekata ako su preveliki. Zato pomak mora biti dovoljno malen da ne bi bila narušena vremenska i geometrijska koherentnost (spomenuta ranije u odlomku 2.2.1). Ako prostorni pomak ovisi o vremenu, onda vremenski interval ažuriranja treba bit dovoljno malen. Vidi sliku 4.3.

$TIMER\_MS$ postavljen je na 25, a $TIME\_BETWEEN\_UPEDATES(dt)$ na 0.01f.



Slika 4.2 Tunneling
Samo primjer (c) ima dovoljno česte provjere presijecanja da bi zabilježio sudar loptice i zida [3]



Slika 4.3 Pomak objekata
P pozicija, t_n vrijeme. Pozicija objekta treba se mijenjati gotovo neznatno kako bi se stvorila glatka animacija i izbjegao *tunneling*. Stalna sila teža uzrokuje akceleraciju prema podu[1]

4.2 Vec3f knjižnica

U programskom rješenju korištena je Vec3f knjižnica za rad s vektorima iz [14]. Ima definiranu klasu Vec3f koja predstavlja 3D vektor. Implementira većinu operacija s dva vektora ili sa skalarom i vektorom (skalarni produkt i sl.). Brzina loptice, pozicija loptice i normale zidova su tipa Vec3f.

4.3 Detekcija sudara

Sudar krutih tijela prikazan je lopticama. Loptice imaju slijedeću strukturu:

```
struct Ball{
  Vec3f pos; // pozicija središta i težišta loptice
  Vec3f v;  // vektor brzine
  float m; // masa loptice
  float r; //radijus loptice
  float t_red; // vrijednost u svezi s intenzitetom crvene
    boje loptice  nakon sudara
  Ball* next; //pokazivač na slijedeću lopticu kada su
   loptice u listi
};
```

4.3.1 Sudar dvije loptice

Detekcija preklapanja dvije loptice, odnosno dvije sfere vrlo se jednostavno implementira ako sfera ima prethodno opisanu strukturu loptice odnosno sadrži podatak o poziciji središta i duljini radijusa. Dvije sfere se presijecaju samo ako je zbroj njihovih radijusa manji od udaljenosti njihovih središta. Kako bi se smanjio broj operacija (za slučajeve kada su pozicije negativnih iznosa) u računu je korištena usporedba kvadrata vrijednosti radijusa i razlike pozicija.

```
bool check_collision(BallPair pair){

   Ball *bA= pair.bA;
   Ball *bB= pair.bB;

   float r = bA->r + bB->r; //zbroj radijusa
   if ((bA->pos - bB->pos).magnitudeSquared() < r*r){
   /* 2 loptice se presjecaju ako je (kvadrat)
   udaljenost(i) njihovih središta manji od
   (kvadrata) zbroja duljina njihovih radiusa */

     Vec3f netVelocity = bA->v - bB->v;
     Vec3f displacement = bA->pos - bB->pos;

    if(netVelocity.dot(displacement)<0){
    //ako se loptice približavaju, sudar je zabilježen (1.
   put)
       bA->t_red = RED;
       bB->t_red = RED;
       return true;
    }
    return false;
    /*U slučaju da se loptice udaljavaju, sudar je već bio
     zabilježen i u prethodnoj provjeri, no odbijanje nije
   poništilo dubinu penetracije, pa je sudar ponovno
   registriran. Može se desiti kod prvog iscrtavanja, ako
   dvije loptice dobiju sličnu random poziciju*/
  }else return false; //nema sudara
}
```

4.3.2 Sudar loptice i ravnine

```
bool check_wall_collision(BallWallPair pair){
  Ball *b = pair.ball;
  Wall *w = pair.wall;
  Vec3f dir = w->direction;//normala ravnine u kojoj leži
   zid

   if( b->pos.dot(dir) + b->r >= BOX_SIZE / 2 && b->v.dot(
   dir) > 0){
   /*
   Loptica presjeca zid ako je zbroj skalarnog produkta(
   operacija .dot()) pozicije njenog središta i normale
   ravnine veći ili jednak polovini dimenzije kutije (čije
    se središte nalazi u ishodištu koordinatnog sustava).
   Skalarnim umnoškom dobijemo to da se gleda samo ona os
   vektora u kojoj se udaljenost treba usporediti. Također
    treba biti zadovoljeno i (&&) da se loptica kreće u
   smjeru prema zidu
   */
   w->t_red=RED;
   return true;
  }
   return false;
}
```

4.4 Određivanje sudara i odziv na sudar

Implementirane su 3 funkcije za odziv na sudar. Funkcije *collision*3*D* i *collision*2*D* imaju trigonometrijski pristup problemu određivanja smjera nakon sudara, dok *angleFreeCollision*3*D* ima isključivo vektorski. Funkcija *collision*2*D* radi za 2D sudare, kada se loptice gibaju samo u x i z koordinatnom sustavu. Funkcija *angleFreeCollision*3*D* implementira izraze 3.22 i 3.23. Pokazala se ispravnija i

više robusna u odnosu na *collision3D* jer ima manje operacija u kojima, zbog zaokruživanja, dolazi do gubitka decimala.

```
void angleFreeCollision3D(BallPair pair){

    Ball *b1=pair.bA;
    Ball *b2=pair.bB;

    float m1=b1->m;
    float m2=b2->m;

    Vec3f v1=b1->v;
    Vec3f v2=b2->v;

    Vec3f v12=v1-v2;
    Vec3f v21=v2-v1;

    Vec3f pos12=b1->pos-b2->pos;
    Vec3f pos21=b2->pos-b1->pos;

    float x12= pos12.magnitudeSquared();
    float x21= pos21.magnitudeSquared();
//izrazi za nove brzine iz poglavlja 3.5. završnog rada:
    b1->v -= pos12*(v12.dot(pos12)/x12)*((2*m2)/(m1+m2));
    b2->v -= pos21*(v21.dot(pos21)/x21)*((2*m1)/(m1+m2));
}
```

Druge dvije funkcije implementirane su radi provjere ispravnosti i usporedbu s funkcijom *angleFreeCollision3D*. Preuzete su s [9] i izmijenjene za potrebe rada. Rezultati usporedbe opisani su u posljednjem poglavlju. Oslanjaju se na račun opisan u odlomku 3.4.

4.4.1 Dvije loptice - centralni elastični sudar

U rješenju su prikazana tri posebna slučaja centralnog sudara na kojima se može vidjeti ispravnost algoritma. Sve tri implementirane funkcije za odbijanje $collision3D$, $collision2D$ i $angleFreeCollision3D$ rade ispravno u sva tri slučaja.

slučaj 1.

U slučaju kad je $m_1 = m_2$ loptice jednostavno zamijene brzine $v_1' = v_2$ i $v_2' = v_1$. Ako druga lopta miruje($v_2 = 0$) , tada je i $v_1' = 0$, a $v_2' = v1$. Nakon sudara prva lopta stane dok druga odleti brzinom koju je prije sudara imala prva lopta.

slučaj 2.

U slučaju kad je $m_1 << m2$, ($v2 = 0$), tj. kad savršeno elastična lopta mase m_1 i brzine v_1 udara u vrlo veliku loptu ili savršeno elastični zid. Slijedi da je $v_1' = -v_1$, tj. lopta se odbija jednakom brzinom kojom je došla, a zid pri tom primi impuls sile $2m_1v_1$. Zid ne dobije nikakvu energiju jer lopta prilikom sudara ne mijenja energiju. Ukupna promjena količine gibanja lopte je $2m_1v_1$.

slučaj 3.

U slučaju kad je $m_1 >> m_2$, ($v_2 = 0$) , tj. kad vrlo velika lopta mase m_1 udari u lopticu koja miruje. Pri tom se brzina velike loptice vrlo malo promijeni dok lagana loptica odleti brzinom 2 puta većom od brzine upadne loptice.

4.5 Brute Force Algoritam

Princip rada je da za svaku lopticu u listi provjeri je li u sudaru sa slijedećom. Tako se izbjegava ponovna provjera parova koji su već ranije provjereni. Broj provjera računa se prema formuli (2.1).

```
void bruteForce(Ball *head){
 Ball *temp1=NULL;Ball *temp2=NULL;
 BallPair bp;
 bp.bA = temp1;bp.bB = temp2;

for(temp1=head; temp1!=NULL; temp1 = temp1->next){
 for(temp2=temp1->next; temp2!=NULL; temp2 = temp2->next){
   bp.bA = temp1;
   bp.bB = temp2;
   if(check_collision(bp))responce(bp);
  }
 }
}
```

4.6 Particioniranje Prostora

Za particioniranje prostora u primjeru 3 služi oktalno stablo (ranije spomenuto u odlomku 2.3.1.). Oktalno stablo implementirano je kao klasa s metodama od kojih su najvažnije opisane u narednim odlomcima. Navedeni kod preuzet je iz [14] i prilagođen potrebama rada.

4.6.1 Klasa Oktalno stablo

Klasa oktalno stablo u biti je klasa jednog čvora stabla. Svaki čvor sa svojom djecom i djecom svoje djece ... (granama) čini oktalno stablo.

```
class Octree {

public:
  Vec3f corner1; //(minX, minY, minZ) donji lijevi
   unitarnji kut
```

```
5   Vec3f corner2; //(maxX, maxY, maxZ) gornji desni vanjski
      kut
6
7   Vec3f center;//((minX + maxX) / 2, (minY + maxY) / 2, (
     minZ + maxZ) / 2) centar prostora čvora
8
9   /* Oktanti (djeca) ovog čvora, ako ih ima.
10  children[0][*][*] znači da je oktant s lijeve strane
     centra (od minX do centerX). Children[1][*][*] imaju
     oktanti s desne strane (od centerX do maxX). Slično
     tome Children[*][0][*] označava donju polovicu (1
     gornju) i Children[*][*][0] prednju (1 stražnju).
    */
11  Octree *children[2][2][2];
12
13  bool hasChildren;//Ima li čvor djece
14  int depth;//Dubina na kojoj se nalazi ovaj čvor u stablu
15  int numBalls; //Broj loptica ukljulujući loptice u
     oktantima ovog čvora
16 private:
17  set<Ball*> balls; //set loptica u ovom čvoru, ako nema
     djece
18
19  //ovdje su definirane metode (...)
20 }
```

Korijen osnovnog oktalnog stabala, koje sadrži sve čvorove je *Octree*∗*_octree*. Stablo počinje s dubinom 1 i veličine je promatranog prostora, u ovom slučaju kutije.

```
1    _octree = new Octree(Vec3f(-BOX_SIZE / 2, -BOX_SIZE /
    2, -BOX_SIZE / 2),Vec3f(BOX_SIZE / 2, BOX_SIZE / 2,
    BOX_SIZE / 2), 1);
2
```

4.6.2 Podijela na oktante

Oktalno stablo raste dijeljenjem. Čvor se dijeli kada broj loptica u čvoru prevrši maksimalan (dozvoljeni) broj loptica u čvoru. Iznimno se ne dijeli ako je čvor na najvećoj dozvoljenoj dubini. Do tog slučaja neće nikada doći ako je maksimalna dubina stabla dobro odabrana. O maksimalnoj dubini ovise dimenzije najmanjih oktanata. Dimenzije najmanjeg oktanta moraju biti dovoljno male da u njega ne stane veći broj loptica od maksimalnog dozvoljenog broja po oktantu. Moraju biti i veće od dimenzija loptice inače bi jedna loptica zauzimala prostor više oktanata istovremeno, što bi rezultiralo nepotrebnim prečestim provjerama i smanjenjem performansi algoritma.

```
1  /*Kada broj loptica u čvoru prerase dozvoljeni broj, čvor
       će se podjeliti*/
2
3  void haveChildren() {
4  /* svakom čvoru potrebno je dodijeliti atribute. Od 8
       čvorova svaki će imati svoj index od 000 do 111 */
5   for(int x = 0; x < 2; x++) {
6    float minX;
7    float maxX;
8
9    if (x == 0) {
10    minX = corner1[0];
11    maxX = center[0];
12   }else {
13    minX = center[0];
14    maxX = corner2[0];
15   }
16   for(int y = 0; y < 2; y++) {
17    float minY;
18    float maxY;
19    if (y == 0) {
20     minY = corner1[1];
```

```
21      maxY = center[1];
22    }else {
23      minY = center[1];
24      maxY = corner2[1];
25    }
26    for(int z = 0; z < 2; z++) {
27      float minZ;
28      float maxZ;
29
30      if (z == 0) {
31        minZ = corner1[2];
32        maxZ = center[2];
33      }else {
34        minZ = center[2];
35        maxZ = corner2[2];
36      }
37      //kod će ovu liniju izvršiti 8 puta, za svaki od
    oktana
38      children[x][y][z] = new Octree(Vec3f(minX, minY, minZ)
    ,Vec3f(maxX, maxY, maxZ),depth + 1);
39    }
40   }
41  }
42  //Makni sve loptice iz čvora i dodjeli ih njegovoj djeci
43  for(set<Ball*>::iterator it = balls.begin(); it !=balls.
      end();it++) {
44   Ball* ball = *it;
45   fileBall(ball, ball->pos, true);
46  }
47  balls.clear();
48  hasChildren = true;
49 }
```

4.6.3 Smještanje u odgovarajući čvor

Loptica će biti smještena u one čvorove kojima pripada po svojoj poziciji. Ako presijeca ravninu koja dijeli 2 čvora, biti će smještena u oba. Loptice imaju samo listovi stabla.

```
void fileBall(Ball* ball, Vec3f pos, bool addBall) {
 //Figure out in which child(ren) the ball belongs

 for(int x = 0; x < 2; x++) {
  if (x == 0) {
   if (pos[0] - ball->r > center[0]) {
    continue;
   }
  }
  else if (pos[0] + ball->r < center[0]) {
   continue;
  }

  for(int y = 0; y < 2; y++) {
   if (y == 0) {
    if (pos[1] - ball->r > center[1]) {
     continue;
    }
   }
   else if (pos[1] + ball->r < center[1]) {
    continue;
   }

   for(int z = 0; z < 2; z++) {
    if (z == 0) {
     if (pos[2] - ball->r > center[2]) {
      continue;
     }
```

```
29      }
30      else if (pos[2] + ball->r < center[2]) {
31       continue;
32      }
33      //Add or remove the ball
34      if (addBall) {
35       children[x][y][z]->add(ball);
36      }
37      else {
38       children[x][y][z]->remove(ball, pos);
39      }
40     }
41    }
42   }
43  }
```

4.7 Analiza i testiranje

4.7.1 Usporedba Brute Force i oktalnog stabla

Analiza je rađena isključivo vizualno na temelju zapažanja brzine kretanja loptica. Povečavan je broj loptica za 20. Testirano je za vrijeme između osvježavanja 0.01 i za 0.001 ($TIME\_BETWEEN\_UPDATES$) . Maksimalna dubina oktalnog stabla 6 a maksimalan i minimalan broj djece 6 i 3.

Tablica 4.1 Broj loptica pri kojem računalo počinje usporavati

TBU	BruteForce	Octree 2
0.001	300	560
0.01	660	860

Broj provjera sudara koji se obrađuje kada počinje usporavati iznosi oko 9000 za listu i oko 600 za oktalno stablo. Iz toga je vidljivo da na performanse stabla ne utječe samo broj provjera sudara koje treba napravit već i struktura samog stabla, njegova dubina i broj djece po čvoru. Broj loptica jedini je faktor koji utječe na *Brute Force* algoritam.

4.7.2 Trigonometrijski i vektorski izračun brzina odziva

Izveden je jednostavan test za usporedbu $AngleFreeCollision3D$ i $Collision3D$. Oba podržavaju sudare u kojima loptice imaju brzine u trenutku sudara zadane 3D vektorima. $Collision3D$ rastavlja vektor na njegove komponente, računa kuteve između njih, sve brzine izražava samo u jednoj dimenziji preko odnosa kuteva. Ima mnogo operacija kojima se zaokruživanjem gubi preciznost. Jednostavnim testom to je i dokazano.

U testu su dvije loptice, postavljene na pravcu točno jedna iznad druge. U stvarnom svijetu ovaj test je nemoguć za izvesti što je objašnjeno u predavanju [12]. Ako viša loptica udari nižu uz minimalni pomak, odrazit će se pod nekim kutem i otskakutati. U idealnom slučaju, kakav je moguće simulirati u ovom rješenju, viša loptica

će se odbijati od niže, niža od poda i tako beskonačno.

Nakon svega tri vizualno ispravna odbijanja, viša loptica u $Collision3D$ udarila je u nižu pod minimalnim kutem i odskakutala. $AngleFreeCollision3D$ računao je stalno s istom preciznošću i ispravna odbijanja su se ponavljala tijekom cijelog vremena testiranja, te nije zabilježeno niti jedno neispravno.

Poglavlje 5

Zaključak

Detekcija sudara je složen računalni problem i sastoji od više problema. Tri su komponente detekcije sudara: utvrđivanje sudara, određivanje sudara i odziv na sudar. Kada se implementira aplikacija koja zahtjeva rješavanje sudara u realno vremenu (prije svakog iscrtavanja scene i češće) važna je efikasnost algoritama i struktura koje se koriste. Kako se ne bi provjeravali svi parovi objekata na sceni, velikom broju objekata treba pristupiti prvo algoritmima široke faze, koji eliminiraju parove objekata koji ne mogu biti u sudaru zbog svoje međusobne udaljenosti ili smjera kretanja. Eliminacijom se smanji broj potrebnih složenih provjera koje se rade na paru konkretnih, često kompleksnih objekata. Algoritmi uske faze izvršavaju te složene provjere, a u cilj im je otkriti preklapaju li se doista dva objekta (koja široka faza nije eliminirala). Određuju točke kontakta koje služe funkcijama za odziv. Odziv ili reakcija na sudar karakteristična je za aplikaciju. U fizikalnim simulacijama, u koju kategoriju bismo mogli svrstati programsko rješenje ovog rada, pri implementaciji je važno u simuliranom prostoru ne narušiti fizikalne zakone koji vrijede u stvarnom svijetu. Ukratko, praktični dio rada omogućuje korisniku da vidi više implementiranih rješenja za iste probleme i usporedi njihove performanse, dok teoretski dio objašnjava zašto su različite i kako su rješenja implementirana. Očekivano, bolje riješenje za veliki broj objekata (300 i više), od onog koje izvršava provjere na svim parovima, pokazalo se ono koje koristi jednu od metoda široke faze za particioniranje prostora. Bolje rješenje za simulaciju elastičnog odbijanja u 3D prostoru pokazalo se ono koje koristi vektorski pristup od onog koje ima trigonometrijski.

Bibliografija

[1] B. J. Lucchesi, D. D. Egbert, and J. Frederick C. Harris, "A parallel linear octree collision detection algorithm," *IJCA, Vol. 21, No. 4*, 2014.

[2] T. Moller and E. Haines, *Real-Time Rendering 2nd edition, Chapter 9.* A K Peters, 2002.

[3] C. Ericson, *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology).* Elsevier, 2005.

[4] K. Fauerby, "Improved collision detection and response," 2003.

[5] M. C. Lin, D. Manocha, J. Cohen, and S. Gottschalk, "Collision detection: Algorithms and applications," University of North Carolina, Chapel Hill, NC, USA, Tech. Rep., 2000.

[6] S. Kockara, K. I. T. Halic, C. Bayrak, and R. Rowe., "Collision detection: A survey," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference*, 2007.

[7] B. A. Quentin Avril, Valérie Gouranton, "A broad phase collision detection algorithm adapted to multi-cores architectures," Université Européenne de Bretagne, France, Tech. Rep., 2010.

[8] L. Banožić and I. Kežman. , s Interneta, http://rg.c-hip.net/2014/seminari/banozic-kezman/ (pristupljeno 3.9.2016.).

[9] P. A. T. S. M.Sc. Physics. , s Interneta, http://www.plasmaphysics.org.uk/collision3d.htm (pristupljeno 8.9.2016.).

[10] N. S. Tonči Andreis, Miro Plavšić, *Fizika 3.* Profil, Zagreb, 2004.

[11] doc. dr. sc. Sanda Pleslić. Pripreme za predavanja iz fizike 1. , s Interneta, https://www.fer.unizg.hr/_download/repository/Predavanja5-2014.pdf

[12] W. Lewin. Lecture 17: Momentum of individual objects, rocket equation - classical mechanics (48:35). , s Interneta, https://www.youtube.com/watch?v=3X63nXD6rsg (pristupljeno 9.9.2016.).

[13] Elastic collision. , s Interneta, https://en.wikipedia.org/wiki/Elastic_collision (pristupljeno 10.9.2016.).

[14] B. Jacobs. , s Interneta, http://www.videotutorialsrock.com/opengl_tutorial/collision_detection/home.php (pristupljeno 24.7.2016.).

Kazalo

Sažetak

Tema završnog rada je implementacija detekcije sudara. Zadatak je bio implementirati program koji simulira elastični sudar krutih tijela, analizirati i opisati algoritme za detekciju sudara. Detekcija sudara je detektiranje preklapanja objekata u pokretu. Dodatno opterećenje za sustav detekcije sudara predstavlja izvođenje u realnom vremenu. Detekcija sudara velikog broja objekata i detekcija sudara dva objekta dva su zasebna problema i zato detekciju sudara možemo podijeliti na dvije faze. Svaka faza ima različite skupine algoritama koje koje koriste različite pristupe pri rješavanju navedenih problema. Iz široke faze implementirane su dvije metode - *Brute Force* i *Oktalno stablo*. Za primjer krutih tijela uzete su sfere pa je pripadajuća funkciju za detekciju sudara jednostavan i lako shvatljiv primjer rješavanja problema konkretnih objekata. Nakon detekcije sudara slijedi odziv na sudar. Kako bi simulacija bila realna, u obzir su uzeti zakon očuvanja količine gibanja i zakon očuvanja energije. U kratkoj analizi rješenja prikazano je kako različitim implementacijama možemo postići različitu točnost kod odbijanja i bolje ili lošije performanse programa.

Ključne riječi **— detekcija sudara, elastični sudar, particioniranje prostora**

Abstract

Subject of this final thesis is Collision Detection Implementation. Assigment was to implement a program that simulates eastic collision of rigid bodies, analize and describe algorithms for collision detection. Collision detection is intersection detection between two or more moving objects. Complexity of implementation grows if detection has to run in real-time. Collisoin detection between many objects and collision detection between two complex objects are two separate problems, which have to be treated separately, and that is why collision detection (in universal solutions) is split in two phases. Each of the phases gathers many algorithms in few families. From the broad phase two methods were implemented- the Brute Force and Octree. The solution to exact object problem is fairly simple example since chosen rigid body(s) are spheres. After collision detection comes collision responce. To ensure quality of simulation the principle of the conservation of momentum (and energy) were considered when implementing this simulation. In a short analysis of implemented program it is shown how different implementations can effect preformance and accuracy.
Keywords **— collision detection, elastic collision, space partitioning**

Dodatak A

Upute za korštenje programa

Selekcija primjera

Korisnik mijenja odabir između 3 glavana primjera redom tipkama '1', '2' i '3'.

A.1 Odabir 1

Ako je odabran primjer 1 (pritiskom na tipku '1') prvi zaslon primjera 1 prikazuje test $angleFreeCollision3D$ i $collision3D$ funkcije. Pritiskom na tipku 'f' koristi se $angleFreeCollision3D$ a 't' $collision3D$. Tipka 'p' pauzira izvođenje i ponovno ga pokreće. Tipka '1' vraća loptice u početne pozicije.

Pritiskom na tipku 'c' prikazuje se 1. posebni slučaj centralnog sudara. Ponovnim pritiskom na tipku 'c' prikazuje se idući slučaj itd.

Tipka '0' postavlja loptice u položaj i dodjeljuje im brzinu i smjer tako da sudare jedna s drugom.

A.1.1 Mijenjanje atributa loptica

Lopticama je moguće promijeniti masu, radijus, brzinu u sva tri smjera i položaj.

Prvo je potrebno pritisnuti tipku 'a' za početak mijenjanja atributa. Nakon a može se odabrati:

'r' za radijus 'm' za masu

'p' a zatim 'x'/'y'/'z' za položaj 'v' a zatim 'x'/'y'/'z' za brzinu

'b' za promjenu lopte čije atribute mijenjamo

Nakon odabira atributa tipkama '+' i '-' mijenjamo vrijednost odabranog atributa. Kada je postignuta željena vrijednost možemo odabrati neki drugi atribut. Po završetku mijenjanja atributa potrebno je pritisnuti 'q'.

A.2 Odabir 2

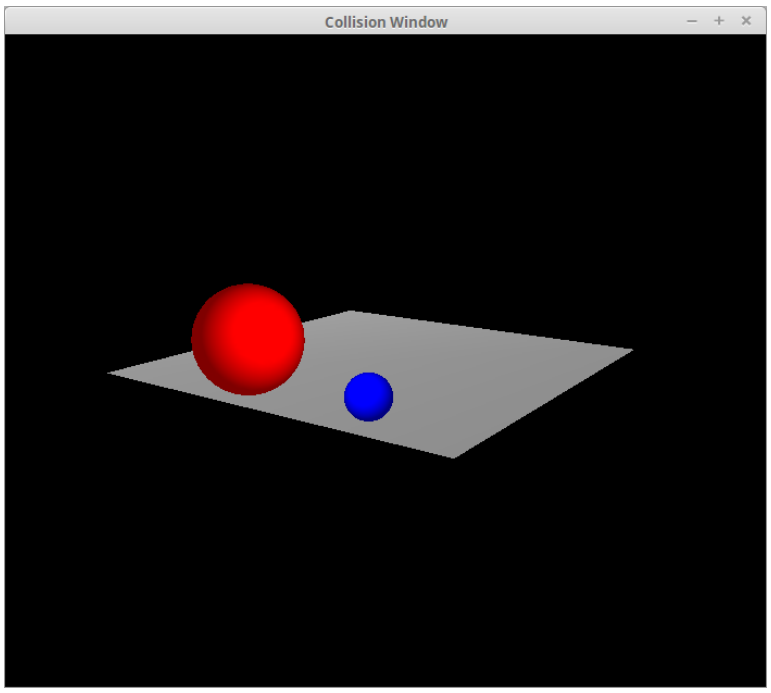
Pritiskom na tipku '2' prikazuje se 2. primjer. Kutija s 20 loptica. Pritiskom na tipku 'w' mijenjamo opciju iscrtavanja zidova. Pritiskom na tipku za razmak na sceni će se iscrtavati 20 loptica više.

A.3 Odabir 3

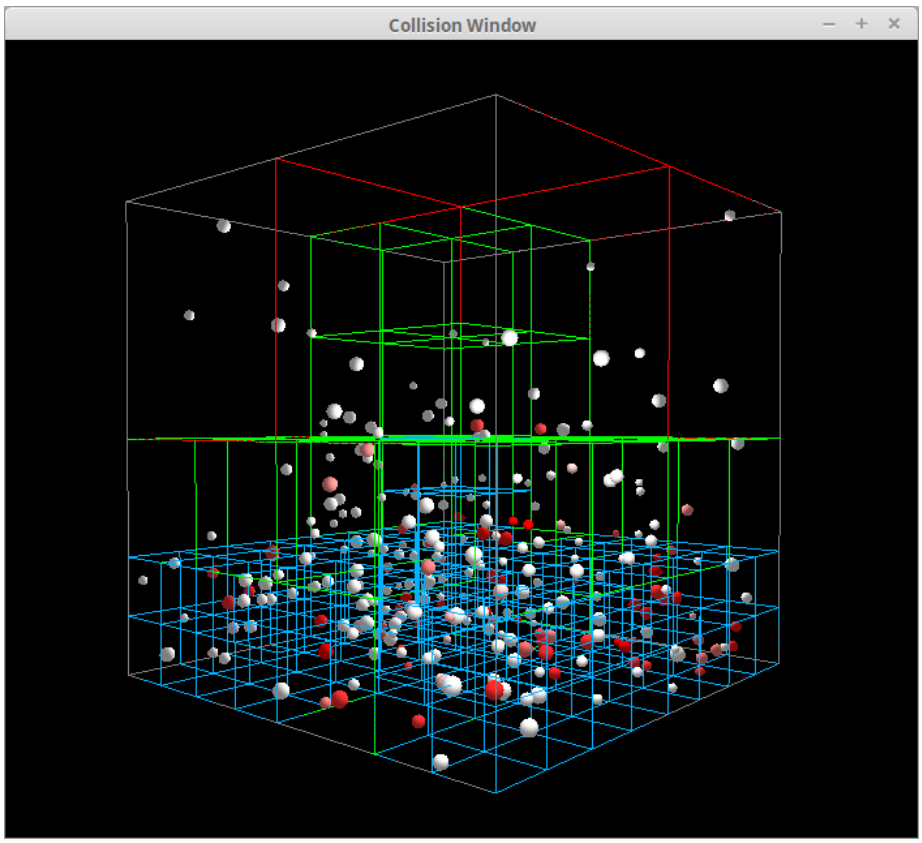
Pritiskom na tipku '3' prikazuje se 3. primjer. Kutija s 20 loptica. Pritiskom na tipku 'w' mijenjamo opciju iscrtavanja zidova. Pritiskom na tipku za razmak dodajemo 20 loptica. Tipkom 'o' mijenjamo opciju iscrtavanja oktalnog stabla.

Dodatak B

Izgled rješenja



Slika B.1 izgled primjera s 2 lopte



Slika B.2 izgled primjera s više objekata - oktalno stablo