UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Simularea interacțiunilor fizice
Subtitlu (ex: versiunea 2018)

Cristian-Andrei SANDU

Coordonator științific:

Prof. dr. ing. Costin-Anton BOIANGIU

BUCUREȘTI

2018

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Diploma Project Title (eg: Diploma project template)
Subtitle (eg: 2018 version)

Cristian-Andrei SANDU

Thesis advisor:

Prof. dr. ing. Costin-Anton BOIANGIU

BUCHAREST

2018

CUPRINS

SINOPSIS

Lucrarea de față are obiectivul de a prezenta o serie de fenomene fizice ce țin de cinematica corpurilor solide, diferiți algoritmi și metode numerice utilizate pentru a simula aceste fenomene și punerea lor în aplicare sub forma unui simulator de interacțiuni mecanice, capabil să aproximeze și să afișeze în timp real mișcarea realistică a unui număr obiecte pe ecran. TODO: rezultate obținute pe scurt (o aplicație cu interfață grafică în stare să ruleze niște demo-uri + modificarea unor scenarii stabilite de dinainte)

ABSTRACT

The aim of this thesis is to provide a closer look at a series of physical phenomena pertaining to the motion of solid objects, as well as to describe various algorithms and numerical methods used in motion simulation and implementing them inside a physics engine able to approximate and render the realistic motion of a number of objects in real time. TODO: rezultate obținute pe scurt (o aplicație cu interfață grafică în stare să ruleze niște demo-uri + modificarea unor scenarii stabilite de dinainte)

MULȚUMIRI

(opțional) Aici puteți introduce o secțiunea specială de mulțumiri / acknowledgments.

1 INTRODUCERE

Simularea interacțiunilor fizice pe un computer se realizează pe baza unui motor de fizică(eng. *physics engine*) care are rolul de a prelua starea scenei la un moment discret de timp t_i și de a determina starea la momentul t_{i+1}. Întrucât acest lucru se realizează într-un spațiu discret, se dorește de fapt obținerea unei aproximări cât mai bună a fenomenelor fizice din realitate. Pentru că fizica este un domeniu extrem de vast, lucrarea de față are ca subiect doar simularea interacțiunilor mecanice dintre corpuri, lăsând aprofundarea altor tipuri de forțe și fenomene la latitudinea cititorilor interesați de domeniu.

1.1 Context

Proiectul a luat naștere ca urmare a interesului autorului pentru mecanică și grafică pe calculator și dorința de a aprofunda pașii necesari implementării unui sistem informatic robust, realistic, ușor de folosit și plăcut vederii. Aplicabilitatea unui astfel de simulator se reflectă într-o multitudine de domenii: jocuri video, proiectare și testare de sisteme mecanice(complexitatea variind de la angrenaje simple până la mașini, avioane), balistică(de natură militară sau civilă), didactică(prin oferirea unei perspective ușor de urmărit și înțeles în studiul mecanicii).

1.2 Problema

Se dorește obținerea unei aplicații care să ofere utilizatorului capabilitatea de a rula simulări pentru niște scenarii definite programatic și editabile în timpul execuției printr-o interfață grafică. Se disting, prin urmare, două subprobleme de rezolvat:

1. motorul de fizică care să simuleze mișcarea și interacțiunile corpurilor din scenă
2. interfața cu utilizatorul

1.3 Obiective

În continuarea celor spuse anterior, sunt delimitate următoarele obiective atinse în elaborarea lucrării de față și a aplicației asociate:

- alegerea unei reprezentări robuste pentru starea(din punct de vedere cinematic) unui obiect al scenei
- integrarea mărimilor secundare(de ex. accelerația, viteza) în vederea obținerii stării noi a obiectului
- detecția potențialelor coliziuni între obiecte
- generarea punctelor de contact între obiectele aflate în coliziune
- rezolvarea contactelor generate cu un răspuns realistic
- desenarea obiectelor pe ecran
- implementarea unui algoritm de ray-casting pentru selectarea unui obiect de pe ecran cu ajutorul mouse-ului
- implementarea unei interfețe grafice pentru controlul simulării și modificarea de elemente ale scenei

1.4 Soluția propusă

În vederea atingerii tuturor obiectivelor de mai sus, este propusă o aplicație OpenGL, capabilă să preia input-ul utilizatorului și să deseneze un număr de scenarii demonstrative. Backend-ul (motorul de fizică în sine) va urmări o arhitectură clasică, folosită cu succes în alte proiecte asemănătoare(ex: [4], [5]). Funcționarea simulatorului este asigurată de o buclă infinită în care la fiecare iterație sunt realizate, pe rând: tratarea input-ului utilizatorului, integrarea(actualizarea stării) corpurilor solide, detecția coliziunilor, rezolvarea lor, desenarea în contextul OpenGL.

1.5 Rezultatele obținute

TODO:Descriere pe scurt a rezultatelor obținute, eventual de ce acestea sunt importante față de alte soluții sau studii.

1.6 Structura lucrării

În continuare voi prezenta pe scurt fiecare secțiune a acestei lucrări, care urmărește, în mare, șablonul oficial.

2. **Motivație și analiza cerințelor**: sunt detaliate atât motivația realizării proiectului propus, cât și funcționalitățile oferite de aplicație, în raport cu cerințele care trebuie acoperite.
3. **Metode existente**: sunt analizate metodele disponibile pentru atingerea fiecăruia dintre obiectivele propuse, modul în care acestea sunt folosite în soluții similare și o evaluare a acestor metode. Fiecare subproces al simulatorului va avea propria subsecțiune.
4. **Soluția propusă**: sunt motivate alegerile și deciziile luate la nivel structural, iar soluția va fi descrisă pe larg, din punct de vedere teoretic.
5. **Detalii de implementare**: este prezentată arhitectura aplicației și orice detalii de implementare considerate a fi relevante(algoritmi folosiți, etapele dezvoltării - cu dificultăți întâmpinate și soluții descoperite)
6. **Evaluare**: analiză a performanțelor aplicației și a gradului de atingere a obiectivelor propuse
7. **Concluzii**: este sumarizat întregul proiect, trecând din nou peste elementele constituente(obiective, implementare, rezultate obținute); în plus, sunt oferite perspective pentru dezvoltarea ulterioară a proiectului.

2 MOTIVAȚIE

Motivația proiectului de față este una personală, aceea de a aprofunda tehnicile matematice și programatice folosite într-o simulare realistică a interacțiunilor mecanice dintre corpuri 3D. În plus, am avut în vedere și posibilitatea îmbunătățirii uneia sau mai multora dintre aceste tehnici.

Proiectul poate fi considerat, în fapt, o "testare a apelor" în domeniul simulării de fizică în timp real, un exercițiu pentru abilitățile mele de programare eficientă, robustă, orientată pe obiecte, dar și o îmbunătățire a cunoștințelor mele de C++.

Consider că cunoștințele dobândite în urma realizării acestui proiect mă vor ajuta să înțeleg mai bine subtilitățile din spatele unui motor de fizică, astfel încât, pe viitor, să fiu capabil de a contribui la proiecte open-source deja existente(bullet[5]) sau, de ce nu, să efectuez muncă în cercetare sau industrie în acest domeniu, la un nivel mai modern sau actual.

3 STUDIU DE PIAȚĂ / METODE EXISTENTE

Pentru început, ar trebui menționat că un motor de fizică este foarte rar întâlnit de sine stătător, el constituind de cele mai multe ori o parte esențială a unei aplicații mult mai complexe(care cuprinde și alte motoare/instrumente necesare funcționării). De aceea, voi ignora sisteme precum motoarele pentru dezvoltarea jocurilor(Unreal Engine, Source, Unity etc.) sau simulatoare științifice și mă voi concentra strict pe expunerea particularităților motoarelor de fizică de sine stătătoare.

3.1 Aplicații similare

Deoarece cele mai multe aplicații nu au nevoie de toate particularitățile unui motor de fizică complex și foarte general, o practică des întâlnită este ca companiile să își implementeze unul propriu, optimizat pentru cerințele specifice ale aplicațiilor dezvoltate. Chiar și așa, putem aminti câteva dintre cele mai populare middleware-uri:

- Box2D[4] este un motor de fizică open source, capabil să simuleze corpuri solide în 2D. Oferă suport pentru detecție continuă a coliziunilor, poligoane convexe și cercuri, corpuri compuse, soluționarea contactelor cu frecare, articulațiilor etc. TODO: menționează arbore dinamic pentru broadphase. Scris în C++, a fost ulterior portat și în alte limbaje și este apreciat pentru simplitatea lui și folosit de dezvoltatori independenți, și a fost inclus chiar și în Unity ca opțiune pentru motorul de fizică 2D.
- Bullet[5] este probabil cel mai cunoscut motor de fizică open source în lumea 3D. Oferă suport pentru detecția discretă sau continuă a coliziunilor pentru toate primitivele de bază, dar și meshe convexe, simularea corpurilor deformabile, articulații și constrângeri complexe, mișcarea vehiculelor etc. Este folosit în jocuri, robotică, efecte speciale în filme și este inclus în software precum Godot, Blender Game Engine sau Unity 3D.

- Din sfera closed source, poate fi amintit NVIDIA PhysX, unul dintre cele mai populare motoare de fizică în industria jocurilor video - inclus în Unreal Engine 3+, Unity 3D și folosit de companii ca EA, THQ, 2K Games. Physx folosește accelerare hardware pe GPU, plăcile video GeForce de la NVIDIA fiind capabile să ofere o creștere exponențială a puterii de procesare a simulărilor fizice.

3.2 Soluții curente

3.2.1 Câteva noțiuni matematice referite și utilizate de algoritmii de mai jos

- O funcție suport a unei mulțimi pe o direcție este definită ca:

$$h_A : \mathbb{R}^n \mapsto \mathbb{R}, \text{ cu } A \subset \mathbb{R}^n,\ h_A(d) = \sup\{d \cdot a \mid a \in A\}$$

  Este utilizată în determinarea punctelor suport(cel mai îndepărtat punct al unui obiect într-o anumită direcție). Formele elementare(cub, sferă, con etc.) au funcții suport foarte ușor de calculat.
- Suma Minkowski a două mulțimi de puncte A și B este mulțimea

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

  și este puțin interesantă în cazul de față.
- Diferența Minkowski a două mulțimi de puncte A și B este mulțimea

$$A \ominus B = \{a - b \mid a \in A, b \in B\}$$

  și are o proprietate remarcabilă. Mulțimile A și B se află în coliziune(au cel puțin un punct în comun) dacă $A \ominus B$ conține originea spațiului geometric. în plus, distanța dintre acestea, în cazul în care nu se intersectează este:

$$dist(A, B) = \min\{\|a - b\| \mid a \in A, b \in B\} = \min\{\|c\| \mid c \in A \ominus B\}$$

- În geometrie, un simplex este o generalizare a noțiunii de triunghi în spații de dimensiune arbitrară. Simplex-urile întâlnite în cadrul algoritmilor de detecție a coliziunii sunt punctul, muchia, triunghiul și tetraedrul.

3.2.2 Detecție coliziuni

Detecția coliziunilor se realizează de obicei în două etape: una preliminară(broad phase) și una precisă(near phase). Motivul este unul evident - algoritmii folosiți în a doua etapă sunt semnificativ mai intensivi computațional decât cei din prima.

Etapa preliminară

Pentru prima fază, fiecare corp din simulare are atașat un volum încadrator sub forma unei primitive(de regulă sferă sau paralelipiped, în cazul 3D) care să îl cuprindă în întregime. Dacă 2 corpuri sunt în coliziune(se intersectează), atunci este sigur că și volumele lor încadratoare se întrepătrund. Testele de intersecție pentru primitive sunt ușor de implementat și computat. Cele mai uzuale volume încadratoare sunt:

- Sfera încadratoare(bounding sphere): este definită de o poziție și o rază. Testul de intersecție este banal: două sfere se intersectează dacă distanța dintre centrele lor este mai mică sau egală cu suma razelor. Este o soluție eficientă ca memorie și ca test de intersecție, dar este inexactă și conduce la multe verificări inutile în faza fină a detecției.
- Axis-aligned bounding box(AABB): este definit de o poziție și de lungimea paralelipipedului pe fiecare dintre cele 3 axe ale sistemului global de coordonate(uzual, se reține jumătatea lungimii fiecărei laturi). Testul de intersecție este din nou destul de banal - se verifică întrepătrunderea celor două AABB-uri pe fiecare dintre cele 3 axe. Este mai precis decât sfera încadratoare, dar dezavantajul este că trebuie recalculat de fiecare dată când corpul este rotit, astfel încât volumul să rămână minim.
- Oriented bounding box(OBB): este definit de o poziție, lungimile paralelipipedului pe fiecare dintre cele 3 axe ale sistemului de coordonate local obiectului și o orientare. Este asemănător unui AABB, dar în loc de axele sistemului global de coordonate, sunt folosite axele sistemului de coordonate locale obiectului. Astfel, el nu trebuie recalculat, atât timp cât obiectul nu este deformabil și este mai precis decât un AABB. Dezavantajul este că testul de intersecție devine mai complicat și

se bazează pe aplicarea Teoremei axei separatoare(SAT, descrisă mai jos TODO: link la SAT de mai jos) pentru 15 axe posibile de separare.

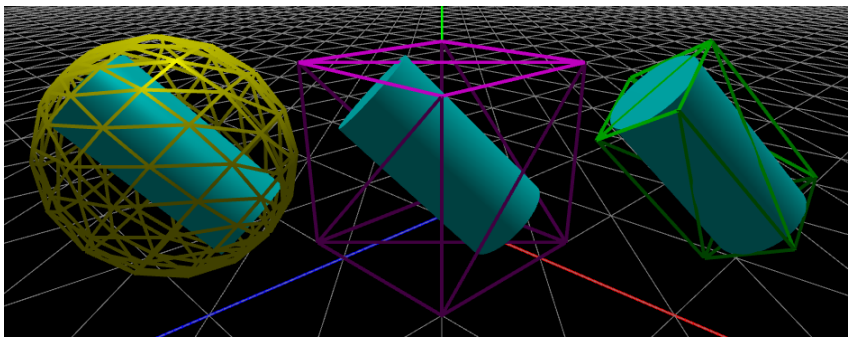


Figura 1: bounding sphere, AABB, OBB

Până acum am acoperit doar coliziunea în cazul unei perechi de obiecte, însă în cadrul unei simulări pot exista un număr foarte mare de obiecte. Abordarea $O(n^2)$, de a verifica fiecare pereche de obiecte din scenă este ineficientă.

O posibilă soluție este aranjarea obiectelor într-un arbore de volume încadratoare(eng. bounding volume hierarchy), care să fie actualizat pe parcursul simulării(odată la una sau mai multe etape ale acesteia). Frunzele arborelui vor fi chiar obiectele individuale ale scenei, iar restul nodurilor vor constitui volumul minim care încadrează toate nodurile copii. Astfel, dacă arborele este menținut echilibrat, timpul de verificare a perechilor de obiecte devine logaritmic, întrucât nodurile copii nu trebuie verificate pentru coliziune dacă părinții lor nu se intersectează.

Un alt tip de arbori folosiți sunt cei care partiționează întreg spațiul scenei, pe baza unor planuri alese astfel încât împărțirea obiectelor să fie cât mai uniformă. Exemple de astfel de arbori sunt arborii BSP(eng. binary space-partitioning tree) sau octrees(și variații ale acestora – quadtrees). O resursă foarte bună pentru mai multe detalii despre aceștia o reprezintă [1, Cap. 6. Bounding Volume Hierarchies] și [1, Cap. 7. Spatial Partitioning] din cartea lui Christer Ericson.

Etapa exactă

A doua parte a detecției de coliziuni o consider ușor mai interesantă, deoarece este responsabilă de stabilirea punctelor de contact dintre corpurile aflate în coliziune. Un exemplu

de caracterizare a unui astfel de punct poate fi:

```
struct ContactPoint {
    vec3 positionA; // pozitia celui mai adanc punct de interpenetrare
    vec3 positionB; // in coordonatele locale ale fiecarui obiect
    vec3 normal; // normala contactului (directia de separare)
    float penetration; // distanta de interpenetrare
    Obj *objA; // pointer catre fiecare dintre obiecte, pentru a accesa
    Obj *objB; // matricele de modelare, functii suport etc.
}
```

Spre acest scop, au fost definiți mai mulți algoritmi care preiau o pereche de obiecte și stabilesc definitiv dacă acestea se intersectează, punctul cel mai adânc de interpenetrare, normala sau direcția de separare și penetrarea în sine.

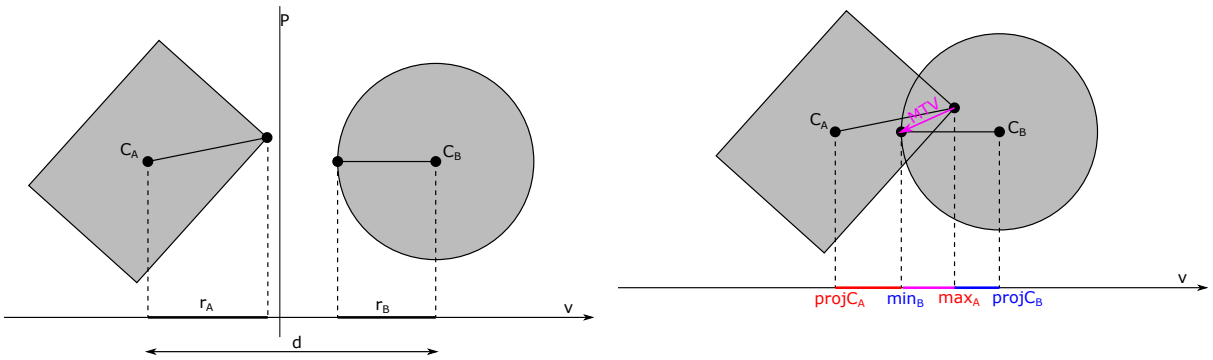
Teorema axei separatoare derivă din teorema hiperplanului separator:

Teorema 1 (Teorema hiperplanului separator)**.** *Dacă A și B sunt două submulțimi disjuncte nevide ale lui $\mathbb{R}^n$, atunci există $v \in \mathbb{R}^n, v \neq 0, c \in \mathbb{R}$, astfel încât $v^T x \leq c, \forall x \in A$ și $v^T x \geq b, \forall x \in B$.*

Adaptată pentru cerințele noastre, ea ne spune că două corpuri se intersectează dacă nu există nicio axă pe care intervalele formate de proiecțiile punctelor celor două corpuri pe acea axă să se intersecteze. Astfel, două obiecte nu se intersectează(pe o axă) dacă suma razelor lor de proiecție este mai mică decât distanța dintre proiecțiile centrelor lor pe acea axă. În cazul poliedrelor convexe, intersecția poate fi de 3 tipuri: față-față, față-muchie, muchie-muchie (vârfurile pot fi considerate muchii degenerate) și este suficient să testăm doar următoarele posibile axe de separare:

- axele paralele cu normalele fețelor obiectului A
- axele paralele cu normalele fețelor obiectului B
- axele paralele cu vectorii rezultați în urma produsului vectorial al tuturor muchiilor lui A cu toate muchiile lui B

Normala de coliziune este chiar axa care a rezultat într-o penetrare minimă, distanța de separare este chiar lungimea vectorului minim de translație ($min_B - max_A$ sau $min_A -$



(a) hiperplanul de separație P, axa separatoare v și razele de proiecție r_A și r_B a două corpuri care nu se intersectează

(b) o axă arbitrară v și vectorul minim de translație pentru această axă(MTV) în cazul intersecției

Figura 2: Teorema axei separatoare

max_B), iar pentru determinarea punctelor de contact, se vor calcula pentru ambele obiecte, punctele suport TODO: link la puncte suport în direcția de separare.

Algoritmul Gilbert-Johnson-Keerthi(GJK) este o altă metodă de a determina cu precizie dacă două obiecte se intersectează. A fost propus inițial în 1988[8], ca metodă de a determina distanța euclidiană între două mulțimi convexe din $\mathbb{R}^n$, iar o implementare eficientă și robustă a fost propusă de Gino van den Bergen în 1998[12].

Algoritmul 1 Testul de intersecție Gilbert-Johnson-Keerthi

```
function GJKTestIntersection(shape_A, shape_B, init_dir)
    new_point ← Support(shape_A, init_dir) − Support(shape_B, −initial_dir)
    simplex ← {new_support_point}
    dir ← −new_point
    loop
        new_point ← Support(shape_A, dir) − Support(shape_B, −dir)
        if dot(new_point, dir) < 0 then
            return False
        end if
        simplex ← simplex ∪ new_point
        simplex, dir, contains_origin ← DoSimplex(simplex)
        if contains_origin then
            return True
        end if
    end loop
end function

function DoSimplex(simplex)
    1. determină simplex-ul cel mai apropiat de origine care se poate forma din cât mai puține din
punctele simplex-ului dat ca parametru
    2. direcția de căutare devine normala către origine a noului simplex
    3. în cazul tetraedru, întoarce True dacă simplex-ul conține originea
end function
```

Practic, la fiecare iterație, simplexul încearcă să se extindă și să cuprindă originea, adăugând mereu un punct suport de pe diferența Minkowski a celor două obiecte, aflat în direcția originii, până când aceasta este cuprinsă în simplex sau nu se mai apropie de acesta.

În cazul în care algoritmul GJK a stabilit că există o coliziune, pentru determinarea normalei, distanței de penetrare și a punctelor de contact, algoritmul EPA[13](eng. expanding polytope algorithm) este continuarea firească. Acesta preia simplex-ul rezultat în urma aplicării GJK și îl extinde iterativ cu puncte suport de pe frontiera diferenței Minkowski, până când distanța minimă dintre politopul rezultat și origine nu se mai modifică. Odată întâlnită această situație, coordonatele baricentrice ale proiecției originii pe triunghiul(în cazul 3D) sau dreapta(în cazul 2D) cele mai apropiate de origine pot fi folosite pentru determinarea punctelor de contact, iar distanța de la origine la triunghi(sau dreaptă) este chiar distanța de penetrare și normala contactului. Expansiunea politopului se face prin divizarea feței celei mai apropiate și crearea de noi triunghiuri sau laturi folosind punctul nou ales și punctele rămase.

Algoritmul 2 Expanding Polytope Algorithm și determinarea punctelor de contact

function EPACreateContact(*shape_A*, *shape_B*, *simplex*)
  polytope ← *simplex.triangles*
  loop
    closest_triangle ← argmin_t rDISTANCE$(tr, origin), tr \in polytope$
    distance ← DISTANCE(*closest_triangle*, *origin*)
    normal ← *closest_triangle.normal*
    new_point ← Support(*shape_A*, *normal*) − Support(*shape_B*, −*normal*)
    new_distance ← DISTANCE(*new_point*, *origin*)
    if $new_distance - distance < threshold$ **then**
      coords ← BARYCENTRIC(*origin*, *closest_triangle*)
      contact_points ← pentru fiecare obiect, se calculează punctul de contact în funcție de corespondențele fiecărui punct al triunghiului din mulțimea de puncte a obiectului respectiv
      return *contact_points*, *normal*, *distance*
    end if
    polytope ← *polytope* \ {*closest_triangle*}
    creează triunghiuri noi folosind *new_point* în spațiul lăsat descoperit
  end loop
end function

3.2.3 Rezolvare coliziuni

În esență, rezolvarea coliziunilor implică aplicarea unui răspuns asupra corpurilor aflate în contact, care să conducă la separarea acestora. În lumea reală, răspunsul vine sub forma forțelor elastice care se opun comprimării(oricât de mică) corpurilor aflate în contact, care determină o accelerație care reduce viteza de ciocnire până la valori negative, când corpurile se separă. În cadrul unui motor de fizică, acest fenomen este simulat cu ajutorul impulsurilor – modificări bruște a vitezelor obiectelor aflate în coliziune, astfel încât acestea să tindă spre separare. În general, sunt suficiente două impulsuri – cel liniar și cel unghiular, fiecare modificând viteza corespondentă.

Soluționarea unei coliziuni este de regulă realizată cu ajutorul conceptului mult mai general de **constrângeri fizice**, definite ca o serie de ecuații și inecuații care trebuie să fie satisfăcute. În cazul rezolvării unei coliziuni, constrângerea care trebuie satisfăcută de vitezele celor două corpuri este:

$$\dot{C}: \left(-\vec{V_A} - \vec{\omega_A} \times \vec{r_A} + \vec{V_B} + \vec{\omega_B} \times \vec{r_B}\right) \cdot \vec{n} + b \geq 0$$

unde:

- $\vec{V_A}, \vec{V_B}$ sunt vitezele liniare ale celor două corpuri
- $\vec{\omega_A}, \vec{\omega_B}$ sunt vitezele unghiulare ale celor două corpuri

- $\vec{r_A}, \vec{r_B}$ sunt definite ca $P_A - C_A$ și $P_B - C_B$, cu P_A, P_B punctele cele mai adânci de interpenetrare și C_A, C_B centrele de masă ale corpurilor
- $\vec{n}$ este normala contactului
- b este un termen de bias, care corespunde vitezei de separare a celor două corpuri și este influențat de coeficientul de restituire al ciocnirii:

$$b = C_R \left(-\vec{V_A} - \vec{\omega_A} \times \vec{r_A} + \vec{V_B} + \vec{\omega_B} \times \vec{r_B} \right) \cdot \vec{n}$$

Frecările sunt rezolvate sub forma unor impulsuri tangențiale, care vor modifica viteza corpurilor în două direcții perpendiculare pe normala de contact, în plus față de impulsul normal.

La calculul impulsurilor se ține cont și de masele celor două corpuri și de tensorii de inerție. O derivare a formulei poate fi urmărită în prezentarea lui Erin Catto[3].

O simulare va conține un număr mare de contacte, care se pot afecta unele pe altele(de ex. în cazul unei stive de obiecte), motiv pentru care rezolvarea acestora se face iterativ, până la convergență. Astfel, dacă rezolvarea unui contact va afecta un altul (este modificată distanța de penetrare, normala sau viteza de întâlnire), acest lucru se va reflecta în iterațiile succesive și sistemul poate găsi soluția corectă. În realitate, într-o simulare, acest lucru se întâmplă destul de rar, dar rezultatele obținute sunt satisfăcătoare, imperfecțiunile fiind neglijabile.

Realistic vorbind, o coliziune este deseori formată din mai multe puncte de contact, care împreună formează un manifold care trebuie rezolvat.

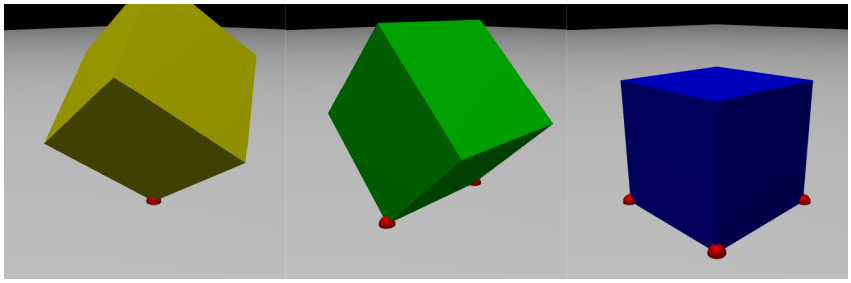


Figura 3: 3 cazuri de contact și manifold-urile lor

În acest scop, există 2 posibile abordări [6, Secțiunea 6.6.2 *Collision response for colliding contact*]:

1. Metoda impulsurilor secvențiale – impulsurile sunt calculate și aplicate iterativ în fiecare dintre punctele de contact.
2. Metoda contactelor simultane – Jacobian-ul folosit pentru calculul impulsurilor este determinat pe baza tuturor punctelor din manifold-ul de contact, iar impulsurile sunt aplicate o singură dată per manifold.

3.2.4 Integrare numerică

Într-un motor de fizică, etapa de integrare constă în actualizarea stării obiectelor, prin integrarea în raport cu timpul a mărimilor fizice derivate. Ne amintim de la orele de fizică din liceu că accelerația unui corp este dată de formula $a = \frac{F}{m}$, unde F este rezultanta forțelor care acționează asupra corpului și m este masa acestuia. Într-o simulare, accelerația este considerată o mărime fizică primară și este cea dintâi calculată în fiecare cadru. Accelerația este totodată definită și ca variația vitezei în timp, iar viteza este variația poziției în timp, ceea ce ne permite să calculăm atât viteza $\dot{p}'$, cât și poziția p', prin integrarea accelerației $\ddot{p}$, respectiv vitezei $\dot{p}$:

$$\dot{p}' = \dot{p} + \ddot{p}t$$

$$p' = p + \dot{p}t + \ddot{p}\frac{t^2}{2} \approx p + \dot{p}t$$

Deoarece într-o simulare avem de-a face cu momente discrete de timp(dictate de diferența de timp dintre două cadre – **deltatime**, aceste valori trebuie aproximate, de unde rezultă nevoia folosirii unor metode de integrare numerică cu pas de timp discret. Se disting o serie de metode, mai mult sau mai puțin precise[7]:

- Metoda Euler explicită presupune determinarea, în ordine, mai întâi a noii poziții și apoi a noii viteze, dar are dezavantajul că pierde din precizie dacă există variații mari ale mărimilor de la un cadru la altul, cu eroare de ordinul $O(deltaTime)$

```
position = position + velocity * deltaTime;
velocity = velocity + acceleration * deltaTime;
```

- Metoda Euler semi-implicită presupune folosirea noii viteze la determinarea noii poziții și este considerabil mai precisă, cu eroare de ordinul $O(deltaTime^2)$

```
velocity = velocity + acceleration * deltaTime;
position = position + velocity * deltaTime;
```

- Metoda Runge-Kutta 4 evaluează derivata stării în 4 puncte diferite din cadrul intervalului de derivare, folosind ca feedback rezultatele anterioare și este mult mai precisă, ceea ce conduce la o eroare de ordinul $O(deltaTime^4)$

3.2.5 Interfață grafică

3.3 Alegeri pentru lucrarea de față

Dintre soluțiile menționate în subsecțiunile de mai sus, am fost nevoit să fac niște alegeri, pe care urmează să le prezint și să le motivez în continuare.

3.3.1 Detecție coliziuni

Deoarece scenele cu care am testat simulatorul sunt de dimensiuni relativ mici(sub 100 de obiecte), am decis că o ierarhie de volume încadratoare nu aduce o îmbunătățire semnificativă în raport cu overhead-ul pe care l-ar aduce etapei de implementare a soluției. La fel, o partiționare a spațiului scenei în arbori, folosind BSP mi s-a considerat nejustificată. Astfel, m-am utilizat doar de un detector $O(n^2)$, care testează intersecția dintre OBB-urile obiectelor folosind Teorema axei separatoare.

Pentru că obiectele din scenă sunt doar corpuri geometrice elementare pentru care nu stochez liste de vârfuri, muchii și fețe, ci doar descrierea geometrică a formei acestora, Teorema axei separatoare nu se pretează pentru determinarea punctelor de contact. În plus, corpurile rotunde(cilindru, con, sferă, capsulă) ar pune probleme în realizarea testului de separare, deoarece ar avea nevoie de un număr foarte mare de axe care să fie testate. În schimb, am ales să folosesc algoritmul GJK, care se putea folosi de descrierea geometrică a formei corpurilor pentru calculul facil al punctelor de suport necesare la determinarea simplex-ului final.

Și, în mod evident, algoritmul GJK se potrivește de minune cu EPA, având în comun o bună parte dintre metode și structuri de date.

3.3.2 Rezolvare coliziuni

Complexitatea pe care ar fi adus-o implementarea metodei rezolvării simultane a contactelor nu este justificată în cazul de față. Ar fi fost nevoie de un redesign al structurilor de date folosite adus de necesitatea operațiilor dintre matrice și vectori de dimensiuni mari, direct proporționale cu numărul de contacte din manifold – ar fi trebuit să îmi scriu propria implementare pentru structurile geometrice de date - vectori și matrice de dimensiuni mai mari decât 4x4. Am ales, astfel, să utilizez metoda impulsurilor secvențiala, care produce rezultate acceptabile pentru o simulare care nu se vrea a fi hiper-exactă.

De asemenea, am decis să nu generalizez contactele la constrângeri fizice și am ales să introduc în schimb alte optimizări, descrise în capitolele ulterioare.

3.3.3 Integrare numerică

În cazul unei simulări, metoda Euler explicită este inferioară din toate punctele de vedere celei Euler implicită, iar precizia altor metode de ordin superior ar fi combătute oricum de micile imperfecțiuni apărute în urma rezolvării coliziunilor. Am ales să păstrez lucrurile simple și să folosesc a doua metodă prezentată mai sus.

4 SOLUȚIA PROPUSĂ

Capitolul conține o privire de ansamblu a soluției ce rezolvă problema, prin prezentarea structurii / arhitecturii acesteia. În funcție de tipul lucrării acest capitol poate conține diagrame (clase, distribuție, workflow, entitate-relație), demonstrații de corectitudine pentru algoritmii propuși de autor, abordări teoretice (modelare matematică), structura hardware, arhitectura aplicației.

Criterii pentru calificativul *NeSatisfăcător*:

- Descriere în limbaj natural.

Criterii pentru calificativul *Satisfăcător*:

- Descriere + diagrame de baze de date, workflow, clase, algoritmi.

Criterii pentru calificativul *Bine*:

- Descriere + diagrame de baze de date, workflow, clase, algoritmi + descrierea unui proces prin care s-a realizat arhitectura/structura soluției.

4.1 Indicații formatare formule

Formulele matematice utilizate în document vor fi centrate în pagină și numerotate.

$$(x+a)^n = \sum_{k=0}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) x^k a^{n-k} \tag{1}$$

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right) \tag{2}$$

5 DETALII DE IMPLEMENTARE

În plus fata de capitolul precedent acesta conține elemente specifice ale rezolvării problemei care au presupus dificultăți deosebite din punct de vedere tehnic. Pot fi incluse configurații, secvențe de cod, pseudo-cod, implementări ale unor algoritmi, analize ale unor date, scripturi de testare. De asemenea, poate fi detaliat modul în care au fost utilizate tehnologiile introduse in capitolul 3.

Criterii pentru calificativul *NeSatisfăcător*:

- Sunt prezentate pe scurt scheme și pseudo-cod.

Criterii pentru calificativul *Satisfăcător*:

- Descriere sumara a implementării, prezentarea unor secvențe nerelevante de cod, scheme, etc.

Criterii pentru calificativul *Bine*:

- Descrierea detaliată a algoritmilor/structurilor utilizați; Prezentarea etapizată a dezvoltării, inclusiv cu dificultăți de implementare întâmpinate, soluții descoperite; (dacă este cazul) demonstrarea corectitudinii algoritmilor utilizați.

5.1 Indicații formatare tabele

Se recomandă utilizarea tabelelor de forma celui de mai jos. Font size : 9. Orice tabel prezent în teză va fi referit în text; exemplu: a se vedea Tabel 1.

Tabela 1: Sumarizare criterii

Calificativ	**Criteriu**	**Observații**
Nesatisfacator	Sunt prezentate pe scurt scheme și pseudo-cod	
Satisfacator	Descriere sumara a implementării, prezentarea unor secvențe nerelevante de cod, scheme, etc.	
Bine	Descrierea detaliată a algoritmilor/structurilor utilizați; Prezentarea etapizată a dezvoltării, inclusiv cu dificultăți de implementare întâmpinate, soluții descoperite; (dacă este cazul) demonstrarea corectitudinii algoritmilor utilizați.	Pot fi incluse configurații, secvente de cod, pseudo-cod, implementări ale unor algoritmi, analize ale unor date, scripturi de testare.

6 EVALUARE

Acest capitol trebuie să răspundă, în principiu, la 2 întrebări și să se încheie cu o discuție a rezultatelor obținute. Cele doua întrebări la care trebuie sa se răspundă sunt:

1. **Merge corect?** (Conform specificațiilor extrase în capitolul 2); Evaluarea dacă merge corect se face pe baza cerințelor identificate în capitolele anterioare.
2. Cât de *Bine* merge / cum se compară cu soluțiile existente? (pe baza unor metrici clare). Evaluarea cât de *Bine* merge trebuie să fie bazată pe procente, timpi, cantitate, numere, **comparativ cu soluțiile prezentate în capitolul 3**. Poate fi vorba de performanță, overhead, resurse consumate, scalabilitate etc.

În realizarea discuției, se vor utiliza tabele cu procente, rezultate numerice și grafice. În mod obișnuit, aici se fac comparații și teste comparative cu alte proiecte similare (dacă există) și se extrag puncte tari și puncte slabe. Se ține cont de avantajele menționate și se demonstrează viabilitatea abordării / aplicației, de dorit prin comparație cu alte abordări (dacă acest lucru este posibil). Cuvântul cheie la evaluare este "metrică": trebuie să aveți noțiuni măsurabile și cuantificabile. În cadrul procesului de evaluare, explicați datele, tabelele și graficele pe care le prezentați și insistați pe relevanța lor, în următorul stil: "este de preferat ... deoarece ..."; explicați cititorului nu doar datele ci și semnificația lor și cum sunt acestea interpretate. Din această interpretare trebuie să rezulte poziționarea proiectului vostru printre alternativele existente, precum și cum poate fi acesta îmbunătățit în continuare.

Criterii pentru calificativul *NeSatisfăcător*:

- Aplicația este testată dar rulează pe calculatorul studentului, nu există posibilități de testare, nu a fost validată cu clienți / utilizatori;
- Nu au fost realizate comparații cu alte sisteme similare.

Criterii pentru calificativul *Satisfăcător*:

- [*Dezvoltare de produs*] Există teste unitare și de integrare, există o strategie de punere în funcțiune (deployment), există validare minimală cu clienții / utilizatorii.
- [*Cercetare*] Principalele componente și soluția în ansamblu au fost evaluate din punct de vedere al performanței, însă nu sunt folosite seturi de date standard, există unele erori de interpretare a datelor.
- [*Ambele*] Discuție minimală asupra relevanței rezultatelor prezentate, comparație minimală cu alte sisteme similare.

Criterii pentru calificativul *Bine*:

- [*Dezvoltare de produs*] Teste unitare și de integrare, instrumente de punere in funcțiune (deployment) utilizate și care arată lucru constant de-a lungul semestrului, lucrare validată cu clienții / utilizatorii, produs în producție.
- [*Cercetare*] Componentele și soluția în ansamblu au fost evaluate din punct de vedere al performanței, folosind seturi de date standard și cu o interpretare corectă a rezultatelor.
- [*Ambele*] Discuție cu prezentarea calitativă și cantitativă a rezultatelor, precum și a relevanței acestor rezultate printr-o comparație complexă cu alte sisteme similare.

7 CONCLUZII

În acest capitol este sumarizat întreg proiectul, de la obiective, la implementare, si la relevanta rezultatelor obținute. În finalul capitolului poate exista o subsecțiune de "Dezvoltări ulterioare".

Criterii pentru calificativul *NeSatisfăcător*:

- Concluziile nu sunt corelate cu conținutul lucrării;

Criterii pentru calificativul *Satisfăcător*:

- Concluziile sunt corelate cu conținutul lucrării, însă nu se oferă o imagine asupra calității și relevantei rezultatelor obținute;

Criterii pentru calificativul *Bine*:

- Concluziile sunt corelate cu conținutul lucrării, și se oferă o imagine precisa asupra relevantei și calității rezultatelor obținute în cadrul proiectului.

BIBLIOGRAFIE

- NU utilizați referințe la Wikipedia sau alte surse fără autor asumat.
- Pentru referințe la articole relevante accesibile în web (descrise prin URL) se va nota la bibliografie și data accesării.
- Mai multe detalii despre citarea referințelor din internet se pot regăsi la:
  - `http://www.writinghelp-central.com/apa-citation-internet.html`
  - `http://www.webliminal.com/search/search-web13.html`
- Note de subsol se utilizează dacă referiți un link mai puțin semnificativ o singură dată; Dacă nota este citată de mai multe ori, atunci utilizați o referință bibliografică.
- Dacă o imagine este introdusă în text și nu este realizată de către autorul lucrării, trebuie citată sursa ei (ca notă de subsol sau referință - este de preferat utilizarea unei note de subsol).
- Referințele se pun direct legate de text (de exemplu "KVM [1] uses", "as stated by Popescu and Ionescu [12]", etc.). Nu este recomandat să folosiți formulări de tipul "[1] uses", "as stated in [12]", "as described in [11]" etc..
- Afirmațiile de forma "are numerous", "have grown exponentially", "are among the most used", "are an important topic" trebuie să fie acoperite cu citări, date concrete si analize comparative.
  - Mai ales în capitolele de introducere, "state of the art", "related work" sau "background" trebuie să vă argumentați afirmațiile prin citări. Fiți autocritici și gândiți-vă dacă afirmațiile au nevoie de citări, chiar și cele pe care le considerați evidente.
  - Cea mai mare parte dintre citări vor fi în capitolele de introducere "state of the art", "related work" sau "background".
- Toate intrările bibliografice trebuie citate în text. Nu le adăugați pur și simplu la final.
- Nu copiați sau traduceți niciodată din surse de informație de orice tip (online,

offline, cărți, etc.). Dacă totuși doriți să oferiți, prin excepție, un citat celebru - de maxim 1 frază- utilizați ghilimele și evident menționați sursa. .

- Dacă reformulați idei sau creați un paragraf rezumat al unor idei folosind cuvintele voastre, precizați cu citare (referință bibliografică) sau cu notă de subsol sursa sau sursele de unde ați preluat ideile.

Trebuie respectat un singur standard de trimiteri bibliografice (citare), dintre următoarele alternative:

- APA (`http://pitt.libguides.com/c.php?g=12108&p=64730`)
- IEEE (`https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf`)
- Harvard (`https://libweb.anglia.ac.uk/referencing/harvard.htm`)
- Cu numerotarea referințelor în ordine alfabetică sau în ordinea apariției în text (de exemplu, stilul cu numere folosit de unele publicații ACM - `https://www.acm.org/publications/authors/reference-formatting`)

În Latex este foarte ușor să folosiți referințe într-un mod corect și unitar, fie prin adăugarea unei secțiuni `\begin{thebibliography}` (vezi la sfârșitul acestei secțiuni), fie printr-un fișier separat de tip bib, folosind comanda `\bibliography{}`, așa cum procedăm mai jos prin folosirea fișierului "bibliography.bib". În orice caz, în Latex va trebui să folosiți comanda `\cite{}` pentru a adăuga referințe, iar această comandă trebuie folosită direct în text, acolo unde vreți sa apară citația, ca în exemplele următoare:

- Articol jurnal: [10];
- Articol conferință: [2];
- Carte: [9];
- Weblink: [11];

Important: în această secțiune de obicei apar doar intrările bibliografice (adică doar listarea referințelor). Citarea lor prin comanda cite și explicații legate de ele trebuie facute în secțiunile anterioare. Citarea de mai sus a fost facută aici doar pentru exemplificare.

BIBLIOGRAFIE

[1]

[2] *Proc. 23rd International Symposium on Distributed Computing (DISC, Elche, Spain, September 2009)*, volume 5805 of *Lecture Notes in Computer Science*, Berlin, Germany, 2009. Springer.

[3] Erin Catto. Modeling and solving constraints. In *Game Developers Conference*, page 81, 2009.

[4] Erin Catto. Box2D. `https://github.com/erincatto/Box2D`, 2014.

[5] Erwin Coumans. Bullet Physics. `https://github.com/bulletphysics/bullet3`, 2017.

[6] Dave H. Eberly. *Game Physics*. Elsevier Science Inc., New York, NY, USA, 2003.

[7] Glenn Fiedler. Integration basics - how to integrate the equations of motion, 2004.

[8] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, April 1988.

[9] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[10] A. Amira H. Baali, H. Djelouat and F. Bensaali. Empowering technology enabled care using iot and smart devices: A review. *IEEE Sensors Journal*, 322(10):891–921, 1905.

[11] J. Silva-Martinez. Elen-325. introduction to electronic circuits: A design approach,. `http://www.ece.tamu.edu/~spalermo/ecen325/Section%20III.pdf`. Last accessed: 28 February 2018.

[12] Gino Van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *J. Graph. Tools*, 4(2):7–25, March 1999.

[13] Gino Van den Bergen. Proximity queries and penetration depth computation on 3d game objects. 2001.

ANEXE

Anexele sunt opționale. Ce poate intra în anexe:

- Exemplu de fișier de configurare sau compilare;
- Un tabel mai mare de o jumătate pagină;
- O figura mai mare mai mare de jumătate pagină;
- O secvență de cod sursa mai mare de jumătate pagină;
- Un set de capturi de ecran ("screenshot"-uri);
- Un exemplu de rulare a unor comenzi plus rezultatul ("output"-ul) acestora;
- În anexe intră lucruri care ocupă mai mult de o pagină ce ar întrerupe firul natural de parcurgere al textului.

A EXTRASE DE COD

...