VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SIMULACE PEVNÝCH TĚLES

RIGID BODY SIMULATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DENIS LEITNER**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ CHLUBNA**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Leitner Denis**
Program: Informační technologie
Název: **Simulace pevných těles**
Rigid Body Simulation
Kategorie: Počítačová grafika

Zadání:

1. Prostudujte a popište metody simulace pohybu pevných těles v prostoru. Seznamte se s existujícími implementacemi.
2. Prostudujte a popište metody detekce kolizí mezi libovolnými 3D modely, včetně způsobů výpočtu změn pohybu objektů po kolizi.
3. Navrhněte jednoduchou simulační smyčku zajišťující aktualizaci pozic objektů a vykreslování.
4. Navrženou simulaci implementujte a demonstrujte na několika netriviálních scénách.
5. Změřte implementované algoritmy a porovnejte s existujícími implementacemi. Navrhněte možná vylepšení.
6. Vytvořte video, reprezentující výsledky vaší práce.

Literatura:

- Dle zadání vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, experimenty vedoucí k bodu 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Táto práca je zameraná na simuláciu fyziky pevných telies v reálnom čase. Popisuje základné postupy používané pri hernom vývoji zamerané na detekciu kolízie medzi konvexnými mnohostenmi. Ďalej popisuje riešenie zistenej kolízie a spôsob simulácie dynamiky pevných telies. Zaoberá sa takisto aj návrhom a implementáciou simulátoru pevných telies v jazyku C++. Na vykresľovanie scény je použité OpenGL.

Abstract

This thesis deals with rigid body physics simulation in real time. It describes basic methods for collision detection between convex polyhedra, solving collisions and simulation of rigid body dynamics used in game development. Work also describes design and implementation of rigid body simulator written in C++ using OpenGL for rendering.

Kľúčové slová

simulácia, pevné teleso, fyzika, detekcia kolízie, SAT, konvexný mnohosten, impulzná reakcia na kolíziu, C++, OpenGL, 3D, počítačová grafika

Keywords

simulation, rigid body, physics, collision detection, SAT, convex polyhedron, impulse-based collision response, C++, OpenGL, 3D, computer graphics

Citácia

LEITNER, Denis. *Simulace pevných těles*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Chlubna

Simulace pevných těles

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Tomáša Chlubnu. Uviedol som všetky literárne pramene, z ktorých som čerpal.

.....
Denis Leitner
15. mája 2019

Poďakovanie

Rád by som sa poďakoval svojmu vedúcemu Ing. Tomášovi Chlubnovi, za vedenie tejto práce a za cenné rady, ktoré mi v jej priebehu poskytol.

Obsah

Kapitola 1

Úvod

Simulácia pevných telies je disciplína zaoberajúca sa simulovaním fyziky pevných telies skutočného sveta vo svete virtuálnom. Využíva sa hlavne v počítačových hrách, filmovom priemysle ale aj pri prototypovaní robotov. Je to jedna z hlavných častí, ktorá dodáva filmom a hrám zmysel reálnosti. Vďaka nej sa objekty v scéne pohybujú realisticky na základe fyzikálnych zákonov. Cieľom simulácie pevných telies je určiť ako sa objekty v scéne pohybujú a ako medzi sebou interagujú v závislosti na silách, ktoré na ne pôsobia vo virtuálnom svete.

Skladá sa z dvoch hlavných častí, ktoré sú úzko previazané: detekcia kolízie a dynamika pevných telies. Dynamika pevných telies zaisťuje pohyb pevných telies v závislosti na silách, ktoré na ne pôsobia. Detekcia kolízie slúži na zistenie že či, kedy a kde došlo ku kolízii medzi dvoma pevnými telesami. Toto je dôležité pre správne simulovanie následnej reakcie telies po kolízii. Obe časti budú podrobne vysvetlené v nasledujúcich kapitolách.

Cieľom tejto práce bolo navrhnúť a implementovať simulátor pevných telies, ktorý by sa dal použiť ako základ na vytvorenie počítačovej hry zameranej na fyziku. Pre takýto simulátor je dôležité nájsť kompromis medzi presnosťou a rýchlosťou výpočtu. Výstup výsledného produktu by mal preto približne pripomínať správanie objektov v skutočnom svete a mal by bežať plynule pri rozumnom počte objektov v scéne na priemernom počítači súčasnosti (2,3GHz procesor s integrovanou grafickou kartou, 4GB RAM).

Práca je rozdelená na dve hlavné časti. Prvá časť je teoretická, v nej sú vysvetlené teoretické znalosti potrebné na implementáciu simulátoru pevných telies. Druhá časť sa zoberá samotným návrhom a implementáciou simulátoru. V kapitole 2 sú popísané techniky na detekciu kolízie, ktoré sa bežne používajú v hernom priemysle a takisto aj techniky na optimalizáciu tohto procesu. Kapitola 3 sa zaoberá simuláciou fyziky. V kapitole 3.1 je popísaný proces simulácie, jej typy a numerické integračné metódy. Dynamika pevných telies je popísaná v kapitole 3.2. V kapitole 3.2 je takisto popísaný aj spôsob reprezentácie polohy objektu v scéne a sú predstavené veličiny spôsobujúce zmenu posuvného a rotačného pohybu objektov. Sú tu popísané pohybové zákony a rovnice na výpočet potrebných veličín. V kapitole 4 je popísaný návrh a implementácia simulátoru pevných telies. Sú tu spomenuté aj knižnice, ktoré boli použité pri vytváraní tohto programu. V kapitole 5 sú popísané simulačné experimenty vykonávané s výsledným programom a ich výsledky.

Kapitola 2

Detekcia kolízie

Táto kapitola sa zaoberá detekciou kolízie medzi pevnými telesami. Je to veľmi rozsiahla téma a existuje množstvo metód na detekciu kolízie medzi dvoma telesami. V tejto práci je detailne popísaná metóda *Separating-axis Test* a okrajovo je spomenutá metóda *GJK*. V tejto kapitole sú takisto popísané aj optimalizačné metódy na urýchlenie detekcie kolízie.

Cieľom detekcie kolízie je zistiť že či, kedy a kde došlo ku kolízii medzi dvomi pevnými telesami [4]. Výpočtovo je to veľmi náročná úloha, pretože akýkoľvek objekt v scéne môže kolidovať s akýmkoľvek iným objektom v scéne. Objekty sú navyše zložené z veľkého množstva polygónov. Najprimitívnejšie riešenie by bolo zistiť pre každý polygón jedného objektu, či sa neprekrýva s nejakým iným polygónom druhého objektu. Toto riešenie by však bolo veľmi pomalé. Preto sa proces detekcie kolízie väčšinou delí na dve fázy: *broad phase* a *narrow phase*. Úloha *broad phase* je nájsť skupiny objektov, ktoré by spolu mohli kolidovať a vylúčiť tie, ktoré určite nekolidujú. V *narrow phase* sa skontrolujú kandidáti vybraní v *broad phase* a zistí sa či naozaj kolidujú.

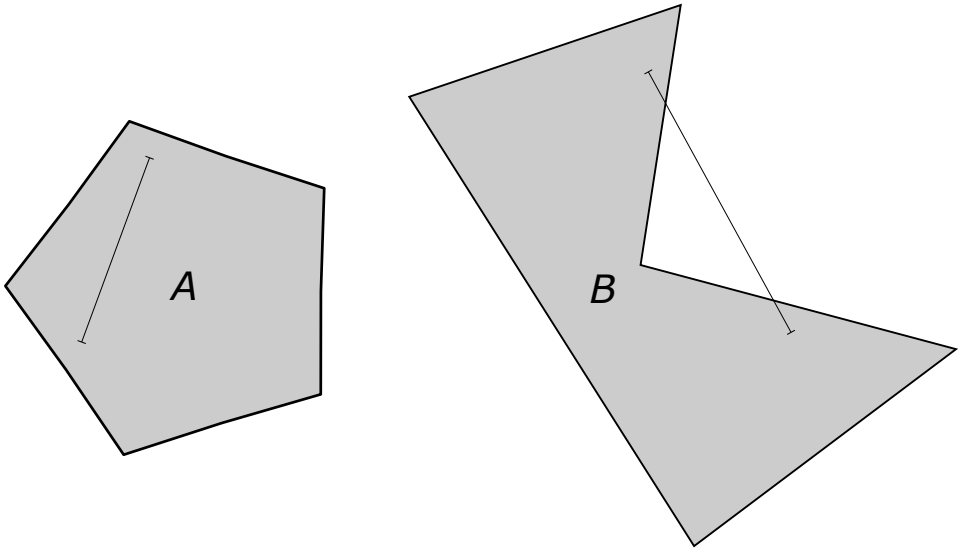
2.1 Separating-axis Test

Separating-axis Test (SAT) je jednoduchý algoritmus na zistenie, že či sa dva *konvexné útvary* prekrývajú. *Konvexný útvar* je útvar, v ktorom úsečka spájajúca akékoľvek dva body patrí tomuto útvaru, viď obrázok 2.1.

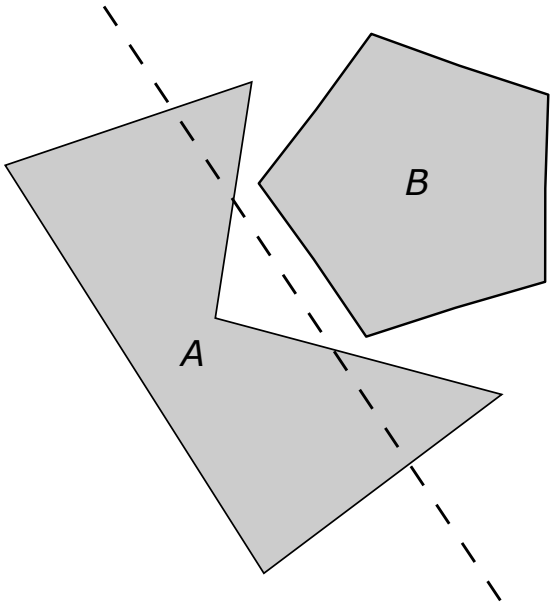
SAT vychádza z *hyperplane separation theorem*. Tento teorém hovorí, že dve konvexné množiny A a B sa buď prekrývajú, alebo existuje *nadrovina*[1] P taká, že každý prvok množiny A je na jednej strane tejto nadroviny a každý prvok množiny B je na druhej strane. Čiže ak sa dva útvary neprekrývajú, je medzi nimi medzera, do ktorej dokážeme vložiť rovinu (priamku), ktorá ich oddeľuje. Teorém však neplatí pre konkávne útvary, pretože rovina nemusí byť dostačujúca na oddelenie útvarov, viď obrázok 2.2. Ak by sme však chceli použiť konkávne útvary, museli by sme ich rozdeliť na niekoľko konvexných a pomocou *SAT* testovať tie.

Pre danú nadrovinu P oddeľujúcu útvary A a B, *deliaca os* (angl. *separating axis*) je priamka o kolmá na P (obrázok 2.3). Pretože deliaca os existuje len ak existuje deliaca nadrovina, pri zisťovaní či sa dva útvary prekrývajú môžu byť testované buď deliace osi, alebo deliace nadroviny. V praxi sa však využíva deliaca os pretože test je rýchlejší.

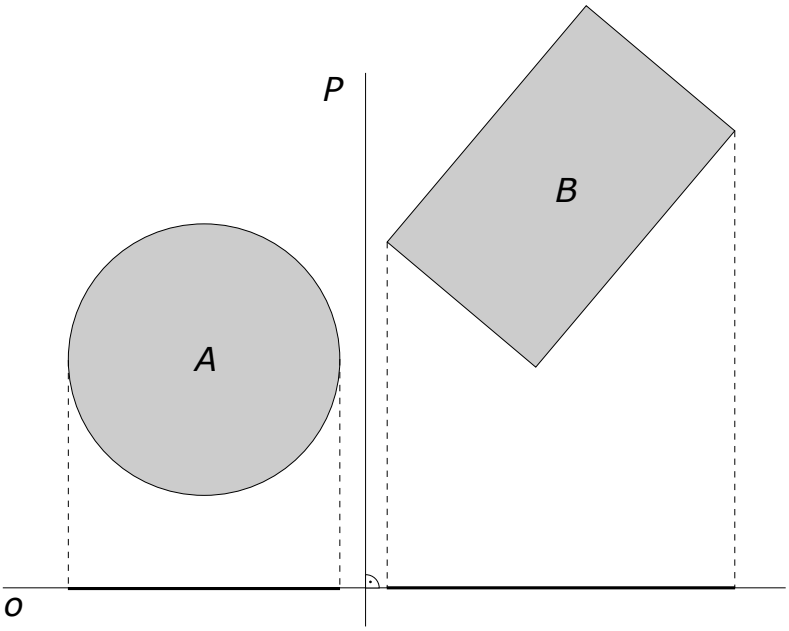
[1] Nadrovinou sa v geometrii rozumie pre daný priestor dimenzie n akýkoľvek jeho podpriestor dimenzie $n-1$. V rovine (2-rozmerný priestor) je nadrovinou každá priamka a v 3-rozmernom priestore je nadrovinou každá rovina.



Obr. 2.1: Rozdiel medzi konvexným (A) a konkávnym (B) útvarom. Pri konkávnom útvare úsečka spájajúca dva body vybieha mimo útvar.



Obr. 2.2: *Separating-axis test* nie je možné použiť pre konkávne útvary, pretože priamka nepostačuje na ich oddelenie.



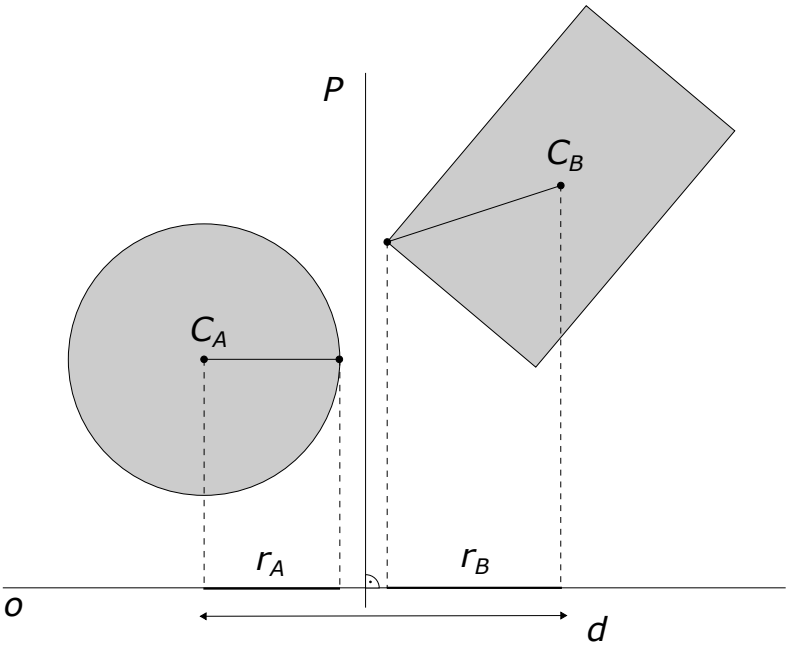
Obr. 2.3: *Separating-axis test:* Dva útvary oddelené priamkou P, ktorých projekčné intervaly na deliacej osi o sa neprekrývajú.

Separating-axis test používa ortogonálny priemet na nájdenie projekčných intervalov útvarov na potenciálnu deliacu os (viď obrázok 2.3). Ak sa projekčné intervaly dvoch útvarov na tejto osi prekrývajú, pokračuje sa na ďalšiu potenciálnu os. Ak sa však intervaly neprekrývajú, test môže skončiť – našla sa deliaca os a úvary sa určite neprekrývajú. Separating-axis test musí skontrolovať všetky potenciálne deliace osi, pretože prekrytie intervalov na jednej osi neznamená že útvary sa prekrývajú. To, že útvary sa prekrývajú je možné usúdiť len ak sa prekrývajú ich projekčné intervaly na všetkých potenciálnych osách. Existuje nekonečne veľa potenciálnych deliacich osí. Pre konvexné mnohosteny je možné znížiť tento počet na nasledujúce prípady:

- Osi rovnobežné s normálami všetkých stien útvaru A.
- Osi rovnobežné s normálami všetkých stien útvaru B.
- Osi rovnobežné s vektormi, ktoré vznikli vektorovým súčinom všetkých hrán útvaru A so všetkými hranami útvaru B.

Projekčný interval sa nájde ortogonálnym priemetom všetkých vrcholov útvaru na potenciálnu deliacu os. Za minimum intervalu sa zvolí minimálny ortogonálny priemet a za maximum sa zvolí maximálny ortogonálny priemet na danú os. Veľkú výhodu pri testovaní pomocou SAT však majú stredovo súmerné útvary, pretože pri nich nie je potrebné vykonať priemet všetkých vrcholov na os. Tieto útvary majú stredový bod S, ktorého priemet na os je v strede ich projekčného intervalu. Preto na test či sa dva stredovo súmerné útvary prekrývajú, stačí vypočítať polomer ich projekčného intervalu a následne porovnať súčet oboch polomerov so vzdialenosťou stredov projekčného intervalu. Ak je vzdialenosť stredov väčšia ako súčet polomerov, útvary sa neprekrývajú. Toto je znázornené na obrázku 2.4.

Ortogonálny priemet sa vykoná skalárnym súčinom pozície vrcholu a smerovým vektorom potenciálnej osi, ktorý musí mať dĺžku jedna (jednotkový vektor). Skalárny súčin



Obr. 2.4: Dva stredovo súmerné útvary sa neprekrývajú ak súčet polomerov ich projekčných intervalov je menší ako vzdialenosť stredov ich projekčných intervalov.

dvoch vektorov je definovaný ako súčet súčinov prislúchajúcich zložiek vektorov:

$$\vec{u} \cdot \vec{v} = (u_1, u_2, \ldots, u_n) \cdot (v_1, v_2, \ldots, v_n) = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n.$$

Výsledok tejto operácie je celé alebo desatinné číslo, nie vektor. Geometricky by sa skalárny súčin dal popísať ako ortogonálny priemet $\vec{v}$ na $\vec{u}$ (obrázok 2.5). Výsledok je dĺžka d vektoru $\vec{v}$ pozdĺž $\vec{u}$:

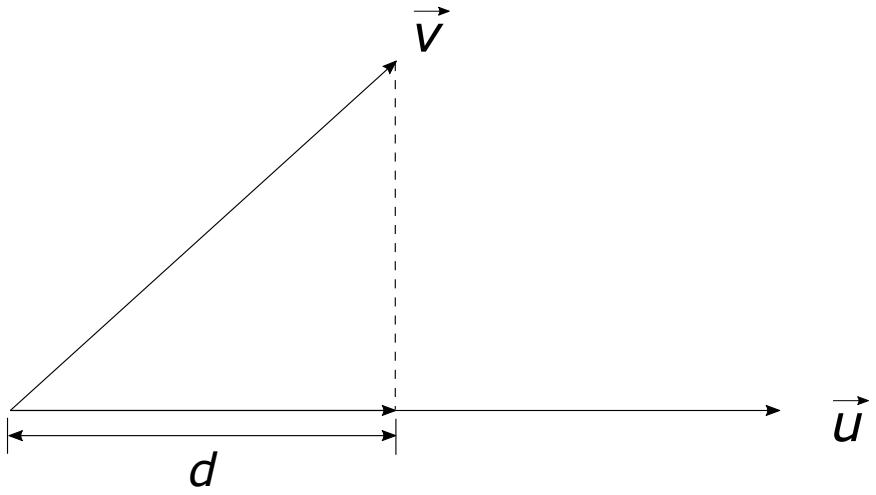
$$d = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}||}.$$

2.2 Gilbert-Johnson-Keerthi algoritmus

Jedna z najefektívnejších metód na zistenie prieniku dvoch mnohostenov je iteratívny algoritmus *Gilbert-Johnson-Keerthi (GJK)* vytvorený v roku 1988 [5]. *GJK* určí prienik dvoch mnohostenov na základe ich vzdialenosti. Algoritmus je založený na skutočnosti, že vzdialenosť medzi dvomi mnohostenmi je rovná vzdialenosti medzi ich *Minkovského rozdielom* a počiatkom súradnicovej sústavy [4]. *Minkovského rozdiel* je jednoduchá geometrická operácia. Od každého bodu útvaru A sa odčíta každý bod útvaru B. Vzniknuté body tvoria Minkovského rozdiel. Užitočná vlastnosť tejto operácie je, že keď je aplikovaná na dva konvexné útvary, výsledný útvar obsahuje počiatok súradnicovej sústavy len ak sa dané dva útvary prekrývajú [8].

2.3 Optimalizačné metódy

V predchádzajúcich sekciách tejto kapitoly boli popísané metódy na detekciu prieniku dvoch objektov. Tieto metódy sú výpočtovo veľmi náročné a navyše musia byť vykonané pre všetky



Obr. 2.5: Skalárny súčin dvoch vektorov sa geometricky dá popísať ako ortogonálny priemet. Výsledok je dĺžka d vektoru $\vec{v}$ pozdĺž $\vec{u}$: $d = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}||}$.

dvojice objektov. Testovanie všetkých objektov v scéne sa stáva časovo veľmi drahé už pri priemernom počte objektov. Preto je vhodné znížiť počet testovaných dvojíc pomocou optimalizačných metód. Toto sa vykonáva v už spomínanej *broad phase*.

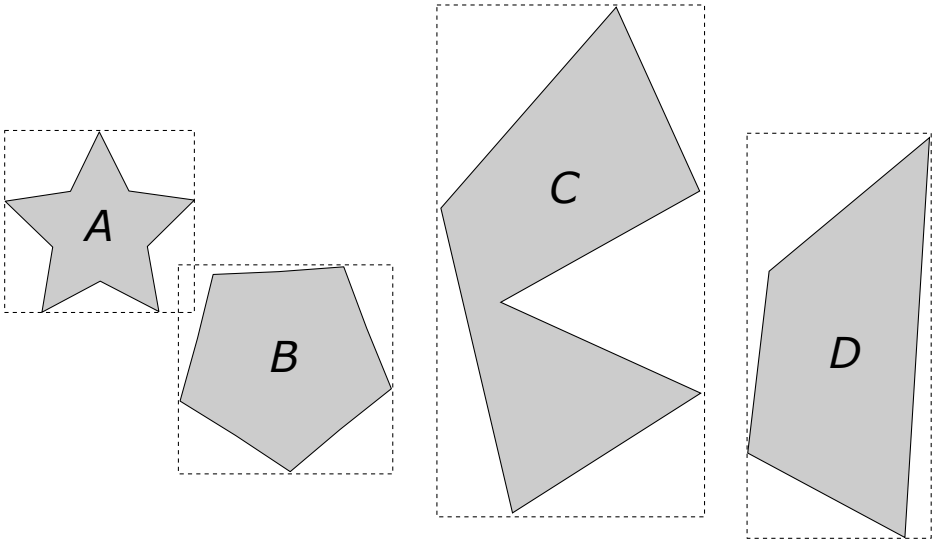
2.3.1 Obalové telesá

Najjednoduchšia optimalizačná metóda je použitie tzv. *obalových telies (bounding volumes)*. Jej cieľom je urýchlenie testu na vylúčenie kolízie. Najskôr sa skontroluje či sa prekrývajú obalové telesá testovaných objektov. Ak sa neprekrývajú, tak sa určite neprekrývajú ani objekty. Ak sa však prekrývajú, objekty sa podrobia detailnejšej detekcii kolízie. Toto je znázornené na obrázku 2.6.

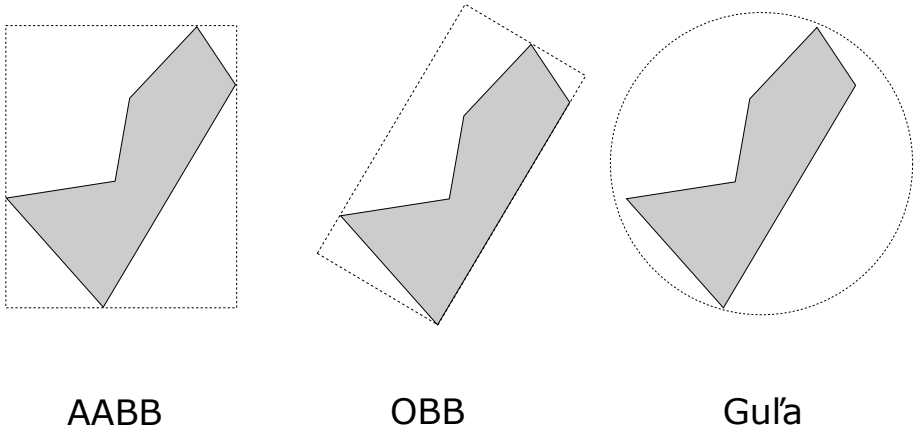
Obalové teleso je jednoduché teleso, ktoré obaľuje jeden alebo viac objektov komplexnejšieho charakteru. Ako obalové telesa sa najčastejšie používajú guľa a kváder, pretože kolízie medzi nimi sú veľmi jednoduché na testovanie. [4]

Keď sa objekty skutočne prekrývajú, test prekrytia obalových telies samozrejme zvyšuje čas výpočtu. Väčšinou je však veľmi málo objektov tak blízko seba, že ich obalové telesá sa prekrývajú. Preto ich použitie výrazne znižuje čas potrebný na detekciu kolízie všetkých objektov v scéne.

Aby bolo využitie obalových telies čo najefektívnejšie, použité telesá musia spĺňať niekoľko charakteristík. Musia to byť jednoduché geometrické útvary aby výpočty boli rýchle a ich reprezentácia zaberala v pamäti čo najmenej miesta. Obalové teleso musí čo najtesnejšie priliehať na obaľovaný objekt aby bol nevyplnený vnútorný objem čo najmenší. Ak je vyplnený vnútorný objem malý a obalové telesá sa prekrývajú, je dosť malá pravdepodobnosť, že aj objekty ktoré sú nimi obalené sa prekrývajú [13]. Porovnanie najpoužívanejších typov obalových telies je na obrázku 2.7.



Obr. 2.6: Použitie obalových telies na rýchlejšie vylúčenie kolízie. Obalové telesá objektov C a D sa neprekrývajú, takže C a D určite nekolidujú. Kolízia medzi A a B nemôže byť vylúčená, pretože ich obalové telesá sa prekrývajú.



Obr. 2.7: Porovnanie najpoužívanejších obalových telies. Na obrázku je vidieť, že najtesnejšie priliehajúce obalové teleso je *oriented bounding box (OBB)*.

Guľa

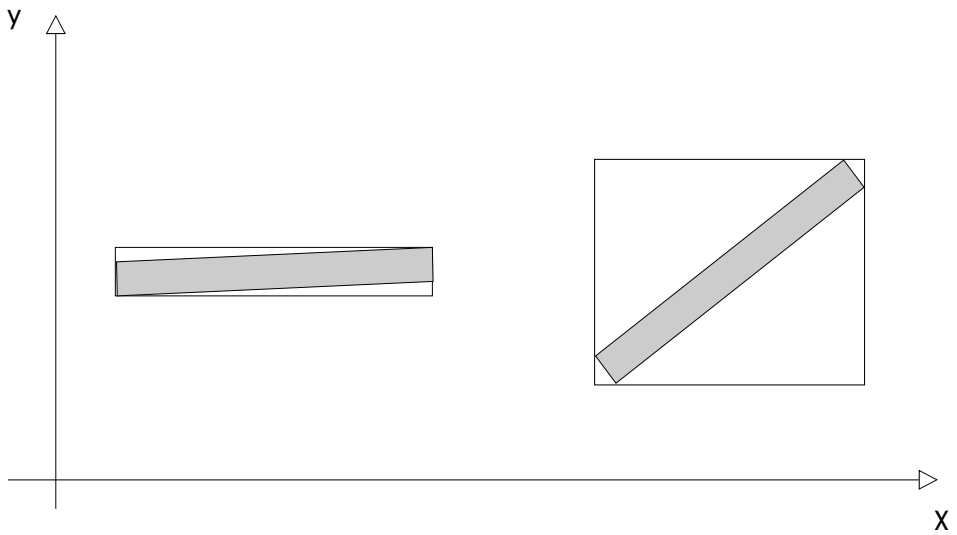
Najjednoduchšie obalové teleso je guľa. Na jej reprezentáciu stačí poznať súradnice stredu a jej polomer. Takisto aj test či sa dva guľové obalové telesá prekrývajú je veľmi jednoduchý. Prekrývajú sa ak vzdialenosť ich stredov je menšia ako súčet ich polomerov. Nevýhodou je zložitosť výpočtu optimálneho guľového obalového telesa. V niektorých prípadoch navyše toto obalové teleso nepresne aproximuje povrch obaľovaného telesa, ako je možné vidieť na obrázku 2.7.

Axis-Aligned Bounding Box

Axis-aligned bounding box (AABB) je jedno z najvyužívanejšich obalových telies. Je to kváder, ktorého steny sú rovnobežné s osami súradnicovej sústavy. Najväčšia výhoda tohto obalového telesa je rýchly test prieniku s iným obalovým telesom. Existujú tri spôsoby reprezentácie *AABB*:

1. Minimálne a maximálne hodnoty súradníc *AABB* pozdĺž každej osi
2. Pozícia minimálneho rohu a dĺžky hrán *AABB*
3. Stred *AABB* a jeho polomery

AABB má však veľkú nevýhodu. Keďže jeho steny musia vždy ostať rovnobežné so súradnicovými osami, musí sa prepočítať vždy keď teleso ktoré je ním obaľované rotuje. Navyše čím menej sú steny obaľovaného telesa rovnobežné so súradnicovými osami, tým horšie toto obalové teleso aproximuje povrch obaľovaného telesa, viď obrázok 2.8.



Obr. 2.8: Na obrázku je možné vidieť nepresnú aproximáciu povrchu obaľovaného telesa pomocou *AABB*, pri odchýlení od súradnicových osí.

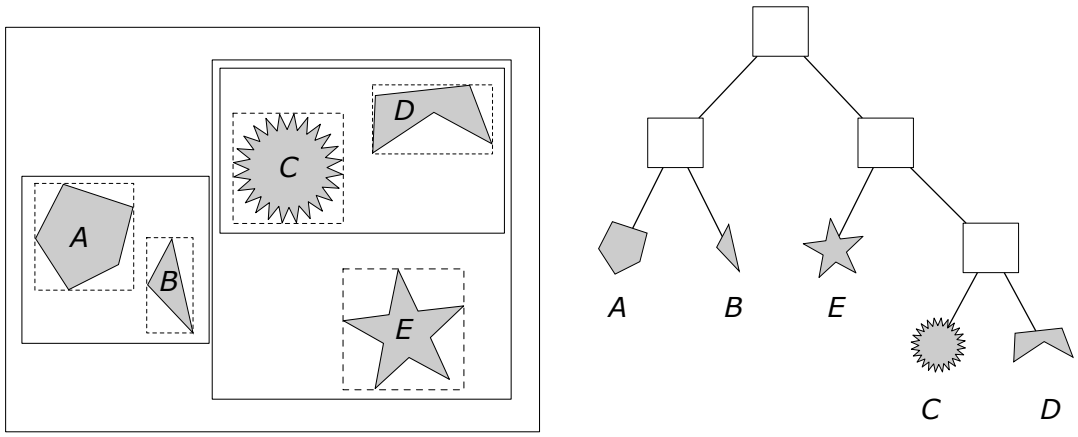
Oriented Bounding Box

Oriented Bounding Box (OBB) rieší vyššie popísaný problém *AABB*. Je to kváder, ktorý nemusí byť rovnobežný so súradnicovými osami ale môže voľne rotovať tak ako rotuje objekt, ktorý je ním obaľovaný. Existuje veľa spôsobov ako ho reprezentovať, napr. súbor ôsmich vrcholov, šiestich rovín, atď. Najčastejšie sa však reprezentuje pomocou stredového bodu, transformačnej matice a polovičných dĺžok hrán. Táto reprezentácia je vhodná pri použití metódy *separating-axis test*, z dôvodu stredovej súmernosti, viď kapitola 2.1.

2.3.2 Hierarchie obalových telies

Aj keď použitie obalových telies výrazne urýchľuje detekciu kolízie, stále sa testujú všetky dvojice objektov. Toto sa dá vyriešiť usporiadaním obalových telies do *hierarchie obalových telies*. Táto hierarchia je vlastne binárny strom.

Pôvodná množina obalových telies tvorí listové uzly tohoto stromu. Tieto uzly sú zoskupené do malej množiny a zapúzdrené vo väčšom obalovom telese. Tieto sú takisto zoskupené a zapúzdrené v ďalšom väčšom obalovom telese. Takto to pokračuje rekurzívne až kým všetky obalové telesá nie sú uzavreté v jednom obalovom telese, ktoré tvorí koreň stromu. Na obrázku 2.9 je jednoduchá hierarchia obalových telies *AABB*.



Obr. 2.9: Hierarchia obalových telies *AABB*. Obalové teleso, ktoré je v koreni stromu zapúzdruje všetky obalové telesá v scéne.

Pri zisťovaní kolízie potomkovia uzlov nemusia byť testovaní ak sa ich rodičovské uzly neprekrývajú. Napríklad na obrázku 2.9 útvary *A* a *E* nemusia byť medzi sebou testované, pretože ich rodičovské uzly sa neprekrývaju. Existuje niekoľko variant hierarchií obalových telies. Medzi najznámejšie patria: *octree*, *BSP strom* a *BVH*.

2.3.3 Rovnomerná mriežka

Ďalším veľmi jednoduchým spôsobom zníženia počtu testovaných dvojíc je rozdeliť scénu do malých oblastí a testovať len dvojice objektov, ktoré spadajú do rovnakej oblasti. Najjednoduchší spôsob je pokryť scénu pravidelnou mriežkou. Táto mriežka rozdelí scénu na rovnako veľké oblasti (bunky). Všetky objekty sú potom priradené do buniek, v ktorých sa

nachádzajú. Testujú sa len tie páry objektov, ktoré zdieľajú rovnakú bunku. Vďaka rovnomernosti mriežky je prístup do bunky prislúchajúcej určitej súradnici veľmi jednoduchý a rýchly.

Pozor si však treba dávať pri volení veľkosti buniek, keďže veľkosť bunky výrazne vplýva na rýchlosť. Treba dbať na to aby bunka nebola tak veľká že sú v nej objekty, ktoré sú ďaleko od seba a určite nekolidujú. Bunka by takisto nemala byť tak malá, že objekt patrí do veľkého počtu buniek. [4]

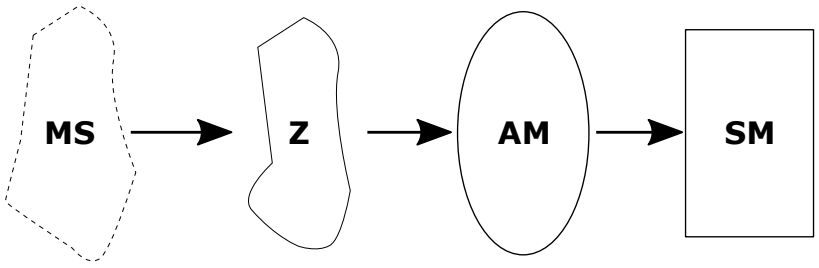
Kapitola 3

Simulácia fyziky

V tejto kapitole sú popísané dve oblasti potrebné na simuláciu fyziky pevných telies: simulácia a dynamika pevných telies. Na konci kapitoly sú predstavené existujúce knižnice slúžiace na simuláciu pevných telies.

3.1 Simulácia

V tejto sekcii je vysvetlené čo je to simulácia, aký je proces vytvárania simulácie, sú tu popísané typy simulácie a numerické integračné metódy. Simulácia je metóda získavania znalostí o systéme experimentovaním s jeho modelom. Systém je súbor elementárnych častí, ktoré majú medzi sebou určité väzby. Systémy sa delia na reálne (tie, ktoré existujú v skutočnom svete) a nereálne (neexistujú v skutočnom svete, napr. počítačové hry). Niektoré reálne systémy sú tak zložité, že nie je možné ich simulovať alebo by získané výsledky boli nepresné. Preto je potrebné pre takéto systémy vytvoriť zjednodušený model, tzv. abstraktný model. Je to model, ktorý nezahŕňa všetky vlastnosti modelovaného systému, ale len tie, ktoré sú pre účely simulácie podstatné. Model je napodobenina systému iným systémom. Na obrázku 3.1 je znázornený proces vytvárania simulácie. [16]



Obr. 3.1: Prvý krok pri vytváraní simulácie je vytvorenie abstraktného modelu (**AM**). Ako bolo spomenuté v úvode tejto kapitoly, abstraktný model je zjednodušený model, ktorý nezahŕňa všetky znalosti (**Z**) o modelovanom systéme (**MS**). Abstraktný model môže byť vyjadrený napríklad formou matematických rovníc. Ná základe abstraktného modelu je vytvorený simulačný model (**SM**). Simulačný model zahŕňa všetky vlastnosti abstraktného modelu. Jediný rozdiel medzi nimi je, že simulačný model je spustiteľný program, ktorý počíta výsledky na základe počiatočného stavu, vstupov a parametrov modelu. Posledná fáza je samotná simulácia. V nej sa experimentuje s vytvoreným simulačným modelom.

3.1.1 Typy simulácie

Podľa typu modelu s ktorým sa experimentuje, je môžné simuláciu rozdeliť na tri typy:

- **Diskrétna simulácia**

  V diskrétnej simulácii sa pracuje s diskrétnym modelom. Stavy tohto modelu sa menia skokovo v diskrétnych časových okamihoch v dôsledku nejakej udalosti. Udalosť je atomická operácia – prebehne celá v jednom okamihu modelového času [16].Diskrétny systém sa zvyčajne popisuje pomocou Petriho sietí, konečných automatov alebo formou programu v programovacom jazyku.

- **Spojitá simulácia**

  Premenné spojitého modelu menia stav spojite. Keďže simulácia pevných telies je spojitá simulácia, v nasledujúcej sekcii je popísaná podrobnejšie.

- **Kombinovaná simulácia**

  Sú v nej použité diskrétne a aj spojité prvky.

3.1.2 Spojitá simulácia

Spojitá simulácia sa využíva v mnohých oblastiach, napr. v priemysle, strojárenstve, fyzike, astronómii, biológii, počítačových hrách atď. Premenné v modeloch spojitých systémov menia svoj stav spojite. Takisto aj modelový čas sa na rozdiel od diskrétnych systémov mení spojite.

Spojité systémy je možné popísať:

- sústavami obyčajných diferenciálnych rovníc,
- sústavami algebraických rovníc,
- blokovými schémami,
- parciálnymi diferenciálnymi rovnicami,
- ...

Pre túto prácu sú dôležité hlavne sústavy obyčajných diferenciálnych rovníc, pretože v kapitole 3.2 bude popísaný pohyb pevných telies práve pomocou nich. Obyčajné diferenciálne rovnice popisujú závislosti zmeny veličín – vyskytujú sa v nich derivácie premenných (obvykle podľa času) [16]. Ako príklad môže poslúžiť rovnica (3.1).

$$\frac{d\mathbf{v}}{dt} = \mathbf{a} \tag{3.1}$$

Rovnica (3.1) vyjadruje, že zmena rýchlosti v čase $\frac{d\mathbf{v}}{dt}$ sa rovná zrýchleniu $\mathbf{a}$. Táto rovnica môže byť vyjadrená ekvivalentne rovnicou (3.2).

$$\dot{\mathbf{v}} = \mathbf{a} \tag{3.2}$$

V rovnici (3.2), $\dot{\mathbf{v}}$ označuje deriváciu rýchlosti podľa času [9]. Na vyriešenie diferenciálnych rovníc sa používajú numerické integračné metódy.

3.1.3 Numerické integračné metódy

Existuje veľké množstvo numerických integračných metód. Každá z nich má svoje výhody a nevýhody. Preto je potrebné na zvolenie najvhodnejšej integračnej metódy poznať vlastnosti jednotlivých integračných metód. Najdôležitejšie vlastnosti sú presnosť, stabilita a zložitosť metódy. Pri použití numerických integračných metód je potrebné zvoliť takú metódu, ktorá vyhovuje požiadavkám na presnosť a rýchlosť výpočtu. Keďže číslicové počítače nedovoľujú riešiť diferenciálne rovnice na spojitom intervale, riešenie je aproximované v určitých bodoch. Interval medzi týmito bodmi sa nazýva integračný krok h. Presnosť s akou sa metóda priblíži skutočnému riešeniu závisí na dĺžke kroku a na zvolenej metóde. Stabilita metódy popisuje správanie metódy, ktoré sa môže pri zväčšovaní integračného kroku skokovo zmeniť. Riešenie je tým nestabilné a výsledky sú chybné. Numerické integračné metódy sa delia na dve skupiny: jednokrokové a viackrokové metódy.

Jednokrokové metódy používajú na výpočet hodnoty v nasledujúcom bode len hodnotu z aktuálneho bodu. Ich princíp spočíva v spočítaní derivácie v určitom bode. Keďže hodnota riešenia na začiatku kroku je známa, je možné využiť hodnotu derivácie v tomto bode na výpočet nasledujúcej hodnoty. Najjednoduchšia metóda, Eulerova metóda, počíta hodnotu priamo. Metódy vyšších rádov počítajú s niekoľkými pomocnými bodmi vo vnútri kroku. Najznámejšie metódy vyšších rádov sú metódy Runge-Kutta. Na obrázku 3.2 je popísaný princíp metódy RK druhého rádu, ktorá na výpočet používa pomocný bod uprostred kroku. Najznámejšia a najpoužívanejšia z týchto metód je RK4. Vzorec na výpočet je v rovnici (3.3).

$$\begin{aligned} k_1 &= hf(t, y(t)) \\ k_2 &= hf(t + \frac{h}{2}, y(t) + \frac{k_1}{2}) \\ k_3 &= hf(t + \frac{h}{2}, y(t) + \frac{k_2}{2}) \\ k_4 &= hf(t + h, y(t) + k_3) \\ y(t + h) &= y(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \end{aligned} \tag{3.3}$$

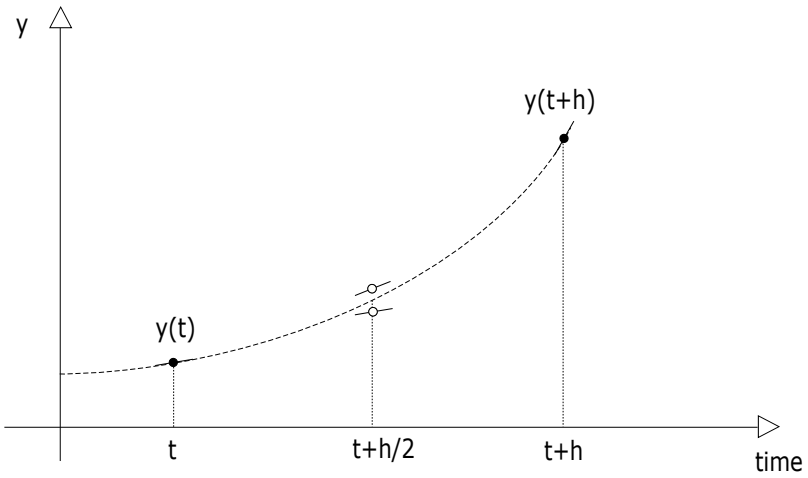
Viackrokové metódy používajú predchádzajúce stavy na výpočet hodnoty v nasledujúcom stave. Tieto metódy majú problém pri štarte, kedy nie je dostupná história stavov. [16]

3.2 Dynamika pevných telies

Jedna z najdôležitejších častí súčasných počítačových hier je fyzika. Vo väčšine prípadov keď sa spomenie fyzika v hrách, myslí sa práve dynamika pevných telies. Dynamika môže byť najjednoduchšie definovaná pomocou *kinematiky*. Tá sa zaoberá len samotným pohybom telies a nezaujíma sa o to, čím je tento pohyb spôsobený [9]. Na druhú stranu, pojem dynamika sa vzťahuje na proces určovania, ako sa menia kinematické veličiny v závislosti na silách, ktoré na teleso pôsobia. Tento proces umožňuje aby pohyb objektov v scéne vyzeral prirodzene – čo je takmer nemožné dosiahnuť animáciou.

Tento text sa zaoberá len tzv. *klasickou dynamikou pevných telies*. To znamená, že sú zavedené dva zjednodušujúce predpoklady:

1. Objekty v simulácii sa riadia Newtonovými pohybovými zákonmi, čiže sa nepredpokladajú žiadne kvantové ani relativistické javy.



Obr. 3.2: Princíp jednokrokových integračných metód. Hodnota na konci kroku $y(t+h)$ sa vypočíta na základe derivácie v bode $t+h/2$, ktorá sa zase vypočíta na základe derivácie v bode t.

2. Všetky objekty v simulácii sú tzv. *pevné telesá*. Pevné teleso je idealizovaný, nekonečne tvrdý, nedeformovateľný tuhý objekt. Jeho tvar zostáva počas simulácie rovnaký.

Tieto dva predpoklady do veľkej miery zjednudušujú výpočty potrebné na simuláciu fyziky.

Pohyb pevného telesa sa dá rozložiť na dve zložky: posuvný a rotačný pohyb. Táto schopnosť rozdeliť pohyb na posuvnú a rotačnú zložku je veľmi vhodná pri simulácii správania pevného telesa. Znamená to, že najskôr je možné vypočítať posuvnú zložku pohybu pevného telesa bez ohľadu na jeho rotáciu a následne k nemu pridať rotačnú zložku. [8]

V tejto sekcii je popísané ako je reprezentovaná poloha objektu v scéne, je popísaný posuvný a rotačný pohyb a veličiny s nimi súvisiace. Ale najskôr je predstavený pojem ťažisko, pretože bez neho by vyššie spomenutá možnosť rozdeliť pohyb pevného telesa na dve zložky nebola možná.

3.2.1 Ťažisko

Ťažisko alebo *hmotný stred* (anglicky *center of mass*), je balančný bod pevného telesa. Inak povedané, hmota pevného telesa je rozložená rovnomerne okolo jeho ťažiska vo všetkých smeroch [8]. Toto je bod, v ktorom keď pôsobí sila, tak nevzniká žiadny rotačný pohyb pevného telesa.Práve tento fakt umožňuje rozdelenie pohybu pevného telesa na dve zložky: posuvný pohyb ťažiska a rotačný pohyb okolo ťažiska.

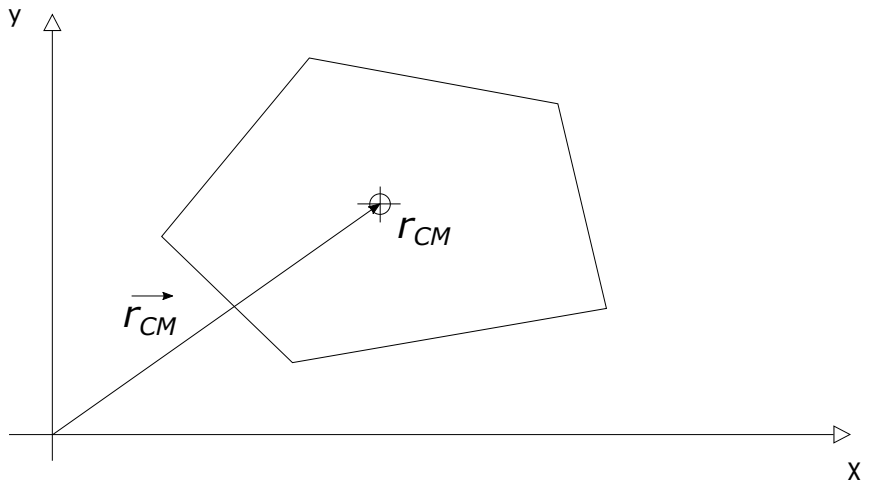
Keby sa pevné teleso rozdelilo na N rovnakých častí, pozícia ťažiska by sa vypočítala ako vážený priemer pozícií týchto častí, viď rovnica (3.4).

$$r_{\vec{CM}} = \frac{\sum_{n=1}^{N} m_i r_i}{\sum_{n=1}^{N} m_i} = \frac{\sum_{n=1}^{N} m_i r_i}{m} \tag{3.4}$$

Celková hmotnosť telesa je označená m a r reprezentuje pozíciu. Pre reálne telesá sa na výpočet ťažiska používajú integrály kde sa veľkosť a hmotnosť častí na ktoré sa teleso rozdelí, blíži k nule. [8]

3.2.2 Posuvný pohyb

Pre zjednodušenie je na začiatok predpokladané, že pevné teleso sa v scéne môže pohybovať len posuvným pohybom. Rotačný pohyb bude pridaný v nasledujúcej sekcii. Z tohto dôvodu je možné v tejto sekcii nahradiť teleso tzv. *hmotným bodom*, ktorý sa nachádza v ťažisku daného telesa a sústreďuje sa v ňom celá hmota pevného telesa. Toto významne zjednodušuje počítanie posuvných pohybových veličín, pretože je zanedbávaný tvar telesa a jeho rotácia, keďže bod je beztvarý a nemá žiadnu orientáciu. Pozícia pevného telesa sa teda dá vyjadriť jediným 3-rozmerným vektorom $r_{\vec{CM}} = \langle x, y, z \rangle$, ktorý smeruje od počiatku súradnicovej sústavy k ťažisku pevného telesa, viď obrázok 3.3.



Obr. 3.3: Pozícia pevného telesa v scéne môže byť vyjadrená pozíciou jeho ťažiska r_{CM}.

Newton popísal tri pohybové zákony pre hmotné body, ktoré vyjadrujú ich pohyb s veľkou presnosťou [13]. Keďže je pevné teleso zjednodušené na hmotný bod, simulácia jeho pohybu môže byť založená práve na týchto zákonoch.

Prvý pohybový zákon alebo zákon zotrvačnosti hovorí, že hmotný bod zostáva v pokoji alebo v rovnomernom priamočiarom pohybe, kým nie je nútený vonkajšími silami tento svoj stav zmeniť. Druhý pohybový zákon alebo zákon sily, vyjadruje že výsledná sila pôsobiaca na hmotný bod sa rovná súčinu hmoty m a zrýchlenia a. Tretí Newtonov pohybový zákon, zákon akcie a reakcie, popisuje interakciu dvoch hmotných bodov. Hovorí, že pre každú akciu existuje rovnaká ale opačná reakcia. Čiže ak jeden hmotný bod pôsobí silou na druhý hmotný bod, druhý hmotný bod pôsobí na ten prvý rovnakou ale opačnou silou. Tento zákon nebude použitý na popis pohybu pevného telesa ako prvé dva, ale pri určení reakcie na kolíziu medzi dvoma pevnými telesami. [14]

Z druhého pohybového zákona vyplýva, že sila pôsobiaca na teleso nemení pozíciu telesa priamo, ale mení jeho zrýchlenie. Zrýchlenie mení rýchlosť telesa a rýchlosť telesa vplýva na zmenu jeho pozície. Rýchlosť je teda zmena pozície v čase (derivácia) a zrýchlenie je zmena rýchlosti v čase. Matematicky je to vyjadrené v rovnici (3.5), kde je pozícia hmotného bodu označená $\mathbf{r}$, vektor rýchlosti $\mathbf{v}$ a vektor zrýchlenia je $\mathbf{a}$.

$$\frac{d^2\mathbf{r}}{dt} = \ddot{\mathbf{r}} = \frac{d\mathbf{v}}{dt} = \dot{\mathbf{v}} = \mathbf{a} \tag{3.5}$$

Tieto kinematické vzťahy hovoria, že ak dokážeme nájsť zrýchlenie objektu, tak vieme určiť aj jeho pohyb v čase. Integrovaním zrýchlenia v čase získame zmenu rýchlosti a dvojitou integráciou zrýchlenia získame zmenu pozície v čase. Na počítači sa integrácia vykonáva numericky. Numerická integrácia je vysvetlená v kapitole 3.1. [9]

Na to aby bolo možné podľa vyššie uvedených vzťahov vypočítať novú pozíciu telesa, je potrebné vypočítať jeho zrýchlenie. S predpokladom že sily, ktoré pôsobia na pevné teleso sú známe, vzorec na výpočet jeho zrýchlenia je možné odvodiť z druhého pohybového zákona, viď rovnica (3.6).

$$\mathbf{F} = m\mathbf{a} \quad \Rightarrow \quad \mathbf{a} = \frac{\mathbf{F}}{m} \tag{3.6}$$

Vektor sily $\mathbf{F}$ určuje smer a veľkosť sily pôsobiacej na teleso, $\mathbf{a}$ je vektor zrýchlenia a m je hmotnosť telesa.

3.2.3 Rotačný pohyb

Doteraz bolo predpokladané, že pevné teleso sa môže pohybovať len posuvne a že je zjednodušené na hmotný bod, ktorý sa nachádza v ťažisku telesa. Na pridanie rotačného pohybu stačí k posuvnému pohybu ťažiska pridať rotáciu telesa okolo tohto body. Na určenie rotácie môže byť uložená orientácia telesa vo forme 3-rozmerného vektoru, ktorého jednotlivé zložky popisujú rotáciu okolo danej súradnicovej osi. Aj keď je táto reprezentácia najjednoduchšia, v praxi sa z matematických dôvodov veľmi nepoužíva [8]. Namiesto toho sa používa reprezentácia vo forme rotačnej 3x3 matice alebo kvaternionu. S kvatenionom sa jednoduchšie počíta a navyše zaberá menej pamäti, keďže rotačná matica má deväť prvkov a kvaternion len štyri. [12]

Takisto ako pri posuvnom pohybe, aj pri rotačnom pohybe musí byť známe akou rýchlosťou teleso rotuje. Toto vyjadruje veličina uhlová rýchlosť ω [10]. Uhlová rýchlosť môže byť reprezentovaná 3-rozmerným vektorom a určuje rýchlosť rotácie pevného telesa okolo jednotlivých súradnicových osí.

3.3 Existujúce riešenia

Cieľom tejto sekcie je zoznámiť čitateľa s existujúcimi implementáciami detekcie kolízie a dynamiky pevných telies. Proces vytvorenia takéhoto fyzikálneho systému je časovo veľmi náročný. Vyžaduje pochopenie fyziky a analytickej geometrie a samotná implementácia zahŕňa veľa úloh, ktoré treba riešiť. Existujúce implementácie sú hojne využívané hernými štúdiami, pretože ich oslobodzujú od nutnosti vytvárať vlastný fyzikálny systém. Zdrojové kódy väčšiny týchto knižníc a enginov sú voľne dostupné, takže vývojári si ich môžu prispôsobiť svojim potrebám.

3.3.1 Open Dynamics Engine

ODE alebo *Open Dynamics Engine*[1] je open-source vysokovýkonná knižnica na simuláciu dynamiky pevných telies. Je platformovo nezávislá a má jednoduché C/C++ API[2]. Ponúka aj pokročilé kĺby a má integrovanú detekciu kolízie s trením. *ODE* je vhodné na simuláciu vozidiel, objektov vo virtuálnej realite a virtuálnych stvorení. Je použitý v počítačových hrách *Call of Juarez* a *Dead Island*, v simulátore *OpenSim*, v robotike a v mnohých ďalších oblastiach.

3.3.2 Bullet

Bullet[3] je moderná knižnica ponúkajúca detekciu kolízie a dynamiku pevných a mäkkých telies. Mäkké telesá zahŕňajú látku, lano a iné. Hojne sa využíva v hernom a aj filmovom priemysle. Medzi najznámejšie hry využívajúce túto knižnicu patria *Grand Theft Auto V*, *Red Dead Redemption*, *DiRT* alebo *Trials HD*. Je integrovaná do 3D modelovacích programov *Maya*, *Houdini*, *Cinema 4D*, *Lightwave*, *Blender* a *Carrara*. Zdrojový kód je dostupný na GitHube[4].

3.3.3 NVIDIA PhysX

NVIDIA PhysX[5] je open-source knižnica, ktorá ponúka hardwérovú akceleráciu na GPU. Bola vytvorená spoločnosťou Novodex AD a a neskôr vylepšená spoločnosťou NVIDIA. Je dostupná na mnoho platforiem, okrem iných napr. Windows, Linux, Android a iOS. Ponúka viacvláknovú simuláciu fyziky. Okrem simulácie pevných telies podporuje simuláciu mäkkých telies, kvapalín, častíc, dynamiku vozidiel a fyziku postáv (ragdoll). Knižnica je obľubená medzi známymi hernými štúdiami. Medzi hry, ktoré používajú technológiu *PhysX* patria napríklad *The Witcher 3: Wild Hunt*, *Batman: Arkham Knight*, *Mafia II* alebo *Metro: Last Light*. *PhysX* je integrované do najznámejších herných enginov, vrátane *Unreal Engine* a *Unity*.

3.3.4 Box2D

Box2D[6] je open-source C++ engine určený na simuláciu pevných telies v 2D. Skorá verzia bola vytvorená na vzdelávacie účely Erinom Cattom pre jeho prezentácie na konferencii GDC. Pevné telesá v *Box2D* môžu byť zložené z konvexných polygónov a kružníc a spojené môžu byť pomocou kĺbov. Je použitý v hrách *Limbo*, *Angry Birds*, *Happy Wheels* a vo veľkom množstve internetových flash hier.

Aj keď sa táto práca zameriava na 3D priestor, tento engine tu bol uvedený preto, že je oveľa jednoduchší ako ostatné spomenuté knižnice. Z tohto dôvodu sa na jeho zdrojovom kóde dajú jednoduchšie pochopiť princípy využívané na detekciu kolízie a simuláciu fyziky. Tieto princípy sa potom len s malými obmenami dajú ľahko preniesť do 3D priestoru.

[1] https://www.ode.org/
[2] Application Programming Interface (API) – Aplikačné programové rozhranie.
[3] https://pybullet.org/wordpress/
[4] https://github.com/bulletphysics/bullet3
[5] https://www.geforce.com/hardware/technology/physx
[6] https://box2d.org/

Kapitola 4

Návrh a implementácia

Táto kapitola sa zameriava na návrh a implementáciu simulátoru pevných telies vo forme spustiteľného programu. Využívajú sa tu teoretické znalosti predstavené v predchádzajúcich kapitolách. V tejto kapitole je popísaný návrh simulátoru, použité technológie a samotná implementácia.

4.1 Návrh simulátoru

V tejto sekcii je popísaná základná štruktúra programu. Pred samotným behom programu sa zo súborov načítajú informácie potrebné na vytvorenie scény a počiatočné stavy simulátoru. Toto zahŕňa pozície, rýchlosti a iné vlastnosti objektov v scéne, 3D modely jednotlivých objektov atď. Keď je všetko inicializované, simulátor beží v nekonečnom cykle až kým nie je ukončený používateľom. Jeden prechod cyklom je jeden vykresľovaný snímok. Počas jedného snímku môže byť vykonaných nula a viac krokov simulácie. Hlavný cyklus programu je popísaný v algoritme 1.

Algoritmus 1: Hlavný cyklus simulátoru.

```
1  msPerUpdate ← 0.01 ;                                   // simulation step
2  accumulator ← 0.0;
3  lastTime ← getCurrentTime();
4  while simulation is running do
5  |   currentTime ← getCurrentTime();
6  |   frameTime ← currentTime - lastTime;
7  |   lastTime ← currentTime;
8  |   processInput();
9  |   accumulator ← accumulator + frameTime;
10 |   while accumulator ≥ msPerUpdate do
11 |   |   update() ;                                     // simulation update
12 |   |   accumulator ← accumulator - msPerUpdate;
13 |   end
14 |   renderFrame();
15 end
```

Funkcia `processInput()` zabezpečuje spracovanie vstupu od používateľa. Funkcia `update()` vykoná jeden krok simulácie počas ktorého sa aktualizujú pozície všetkých objektov v scéne, vykoná sa detekcia kolízie a vyriešia sa zistené kolízie. Každá z týchto častí je bližšie popí-

saná v nasledujúcich sekciách tejto kapitoly. Keď sú už pozície objektov v scéne aktuálne, funkcia `draw()` vykreslí celú scénu.

Na začiatku každého snímku sa počítadlo `accmulator` zvýši o čas trvania minulého snímku. Toto počítadlo indikuje o koľko simulačný čas zaostáva za reálnym časom, čiže aký čas treba počas aktuálneho snímku odsimulovať. Vo vnútornom `while` cykle sa vykonáva simulačný krok s pevnou dĺžkou `msPerUpdate` (dĺžka kroku v milisekundách) až kým simulačný čas nedobehne skutočný čas. Výsledok je, že aplikácia simuluje konštante na všetkých počítačoch. Na pomalších strojoch však obraz vyzerá trhane. [15]

4.2 Použité technológie

Simulátor je naprogramovaný v jazyku C++ za použitia princípov objektovo orientovaného programovania. Na vykresľovanie scény je použité *OpenGL*. *OpenGL* je knižnica na vykresľovanie počítačovej grafiky. Presnejšie to je špecifikácia, ktorá definuje aký má byť výsledok každej funkcie tejto knižnice. Samotná implementácia týchto funkcií závisí na výrobcoch grafických kariet alebo operačných systémov. V tejto práci je použitá verzia `3.3`.

Keďže implementácia OpenGL funkcií je závislá na grafickej karte/operačnom systéme, ich umiestnenie nie je známe pri preklade. Preto je úlohou vývojára získať umiestnenie funkcií ktoré potrebuje, a uložiť si ich do ukazateľov na funkcie [18]. Na tento účel však existuje niekoľko knižníc, ktoré vývojárovi uľahčia prácu. V tejto práci je použitá knižnica *GLAD*.

Na vytvorenie OpenGL kontextu a okna je použité *GLFW*. Je to open-source, multiplatformová knižnica na vývoj OpenGL, OpenGL ES a Vulkan aplikácií. Ponúka jednoduché API na vytváranie okien, kontextov, prijímanie vstupu a udalostí. *GLFW* je napísané v jazyku C a má natívnu podporu pre Windows, macOS, Linux a FreeBSD. Je licencované pod zlib/libpng licenciou. [2]

V grafických aplikáciách sa vo veľkej miere využíva analytická geometria, hlavne vektory, matice a operácie s nimi. V simulátore je na tento účel použitá matematická knižnica *GLM*, ktorá implementuje potrebné dátové štruktúry a funkcie.

Možnosť prekladu na Windowse a aj na Linuxe je zabezpečená pomocou nástroja *CMake*. *CMake* je open-source systém, ktorý slúži na preklad aplikácií pre rôzne operačné systémy. Jednoduchý konfiguračný súbor (`CMakeLists.txt`) v koreňovej zložke projektu slúži na vygenerovanie súborov potrebných na preklad (makefile v Unixe alebo Visual Studio projekt vo Windowse). [1]

4.3 Objekt

Objekt reprezentuje pevné teleso a je implementovaný v triede `Object`. Každý objekt má priradený 3D model, ktorý popisuje jeho geometriu. Objekt ďalej obsahuje informácie o pozícii objektu v scéne, rotácii, kinematických veličinách, má svoje *AABB* obalové teleso atď. Pozícia je uložená ako 3-rozmerný vektor a rotácia ako 3x3 rotačná matica. Všetky objekty sa vytvárajú pri tvorení scény, viď kapitola 4.6.

Pri vytváraní objektu sa vypočítajú všetky jeho vlastnosti potrebné na simuláciu ako sú hmotnosť, pozícia ťažiska a lokálny tenzor zotrvačnosti. Na výpočet je použitý algoritmus popísaný Davidom Eberlym v dokumente *Polyhedral Mass Properties* [3]. Algoritmus na výpočet používa pozíciu vrcholov geometrie objektu uloženú v atribúte `model`.

Objekt môže byť statický alebo dynamický. Statický objekt sa počas simulácie nehýbe a jeho hmotnosť je nekonečno.

4.4 Model

Model popisuje 3D geometriu objektu. Je implementovaný v triede `Model`. V atribútoch tejto triedy sú uložené dáta potrebné na vykreslenie modelu: *vertex* a *index buffer*. Tieto dáta sú po poslaní do pamäte grafickej karty zmazané. Atribút `shape` obsahuje útvar popisujúci povrch modelu. Útvary sú bližšie popísané v nasledujúcej sekcii.

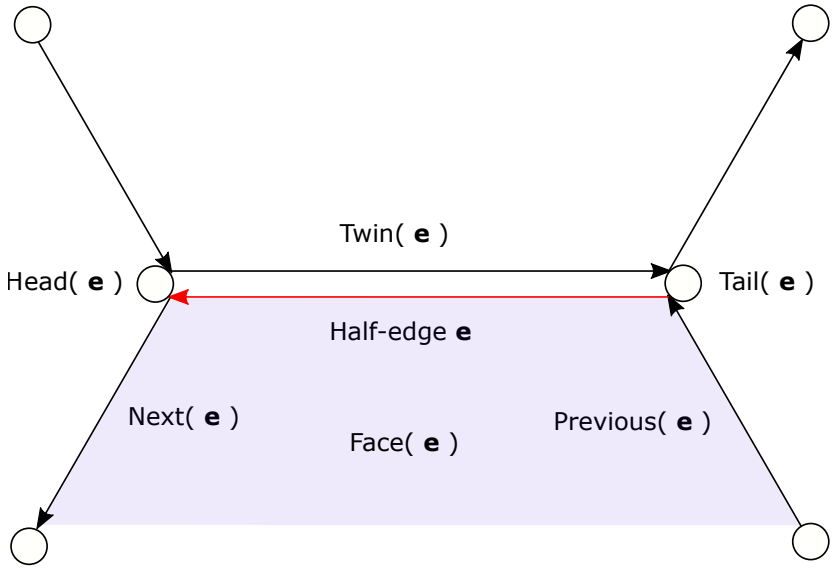
Na načítanie a spravovanie modelov slúži trieda `ModelManager`. Model sa načítava zo súboru vo formáte *Wavefront .obj*. Model je zložený z trojuholníkových stien. Načítanie koordinácií textúr nie je podporované.

4.5 Útvar

Útvar popisuje povrch modelu. Používa sa pri detekcii kolízie. Je implementovaný ako bázová trieda `Shape` z ktorej sú odvodené konkrétne útvary: mnohosten, rovina, guľa.

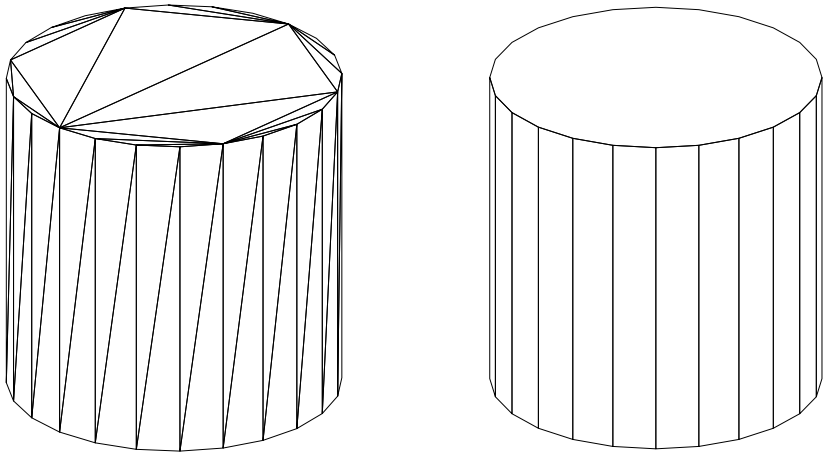
4.5.1 Mnohosten

Mnohosten, implementovaný v triede `Hull`, reprezentuje konvexný mnohosten. Jeho povrch je popísaný pomocou štruktúry *half-edge*. Jej použitie umožňuje napríklad jednoduchú iteráciu hranami steny alebo rýchle nájdenie susednej steny. Štruktúra je znázornená na obrázku 4.1.



Obr. 4.1: Znázornenie štruktúry *half-edge*. Hrana **e** je orientovaná hrana, ktorá si drží ukazatele na vrcholy z ktorých sa skladá, nasledujúcu a predchádzajúcu hranu, stenu v ktorej sa nachádza a svoju dvojičku. Dvojička hrany **e** je hrana s rovnakými ale opačne orientovanými vrcholmi.

Spomenuté výhody sú využité pri vytváraní mnohostenu. Mnohosten sa vytvára popri načítavaní modelu z `.obj` súboru. Najskôr je vytvorená trojuholníková stena. Pre každú hranu tejto steny sa nájde jej dvojička (hrana s opačnou orientáciou vrcholov, viď obrázok 4.1). Ak má stena dvojičky rovnakú normálu ako stena pôvodnej hrany, steny sa zlúčia. Pri použití štruktúry *half-edge* je zlúčenie veľmi jednoduchá operácia – prehodenie niekoľkých ukazateľov. Rozdiel medzi geometriou objektu a jeho reprezentáciou v triede `Hull` je ukázaný na obrázku 4.2.



Obr. 4.2: Rozdiel medzi geometriou objektu (vľavo) a popisom jeho povrchu v triede `Hull` (vpravo).

4.5.2 Rovina

Útvar rovina je odvodený od mnohostenu a je implementovaný v triede `PlaneShape`. Rovina je popísaná rovnako ako mnohosten ale pri detekcii kolízie s guľou metódou *SAT* sú pri nej vykonávané dodatočné testy. Proces detekcie kolízie je popísaný v kapitole 4.7.

4.5.3 Guľa

Najjednoduchší útvar je guľa, ktorá je implementovaná v triede `Sphere`. Na jej reprezentáciu stačí polomer. Polomer je zadaný v súbore na načítanie scény, ktorý je popísaný v kapitole 4.6. Na výpočet atribútov objektov (tenzor zotrvačnosti, hmotnosť, ťažisko), ktoré majú priradený útvar typu guľa nie je použitý algoritmus spomenutý v kapitole 4.3, pretože pre guľu existujú na výpočet jednoduché vzorce. Hmotnosť je násobok objemu a hustoty, ťažisko sa nachádza v strede gule a tenzor zotrvačnosti sa vypočíta podľa vzorca (4.1).

$$I = \frac{2}{5}mr^2 \tag{4.1}$$

Tenzor zotrvačnosti je označený I, hmotnosť je m a polomer r.

4.6 Scéna

Scéna je implementovaná v triede `Scene`. Obsahuje všetky objekty scény, kameru a správcu modelov. Scéna sa na začiatku simulácie načíta zo súboru. Príklad súboru na načítanie scény je vo výpise 4.1.

```
m h cube Models/cube.obj
m p plane Models/big_plane.obj
m s sphere Models/sphere.obj 1.0

o cube0 cube 3500.0
0.39 0.58 0.93 color
-3.0 6.5 0.0 position
0.0 0.0 0.0 rotation
8.0 0.0 0.0 velocity vector
```

Výpis 4.1: Príklad súboru na načítanie scény.

Riadok začínajúci znakom `m` označuje model, ktorý sa má načítať zo zadaného súboru. Druhý znak na riadku znázorňuje typ modelu (h – mnohosten, s – guľa, p – rovina), za ním je názov modelu a relatívna cesta z koreňového adresára aplikácie k požadovanému modelu. Ak je model typu guľa, ako posledná informácia na riadku je uvedený polomer.

Znakom `o` začína riadok označujúci objekt. Za týmto znakom nasleduje názov objektu, názov prislúchajúceho modelu a hustota materiálu z ktorého je objekt vytvorený. Aby bol objekt statický, jeho hustota musí byť `INFINITY`. Predpokladá sa, že objekt nie je dutý ale je celý vyplnený materiálom. Na štyroch nasledujúcich riadkoch je na každom riadku trojica desatinných čísel, oddelených medzerou. Všetky znaky za touto trojicou čísel sa ignorujú. Riadky po poradí značia:

- RGB farbu objektu v rozsahu $\langle 0,0; 1,0 \rangle$,
- vektor počiatočnej pozície objektu v scéne,
- vektor počiatočnej rotácie objektu,
- počiatočná lineárna rýchlosť.

Všetky súbory so scénou musia byť uložené v zložke `Scenes` v koreňovom adresári aplikácie. Názov všetkých súborov musí byť vo formáte `scene_*`, kde za hviezdičku možno doplniť ľubovoľné celé číslo. Toto číslo predstavuje číslo scény. Používateľ si vyberá scénu, ktorú chce simulovať na začiatku simulácie, keď ho program vyzve aby zadal číslo scény.

4.7 Detekcia kolízie – Narrow Phase

Detekcia kolízie medzi dvoma objektmi je implementovaná pomocou metódy *SAT*, ktorá je popísaná v kapitole 2.1. V kapitole 2.1 bolo spomenuté, že ako potenciálne deliace osi sa používajú osi rovnobežné s normálami všetkých stien oboch objektov, a takisto aj osi rovnobežné s vektormi, ktoré vznikli súčinom všetkých dvojíc hrán prvého a druhého objektu. Tento posledný prípad pridáva pri objektoch s väčším počtom hrán veľa potenciálnych deliacich osí, ktoré treba testovať a tým výrazne zvyšuje čas detekcie kolízie. Pre urýchlenie detekcie je však možné tieto osi vylúčiť z testovania bez výrazného zníženia presnosti.

Ďalšie navýšenie výkonu aplikácie je dosiahnuté použitím modifikovanej verzie *SAT* [6]. Táto verzia používa ortogonálny priemet vrcholov iba jedného útvaru na potenciálnu deliacu os na rozdiel od základnej verzie, ktorá premieta vrcholy oboch útvarov. Postup je znázornený v algoritme 2.

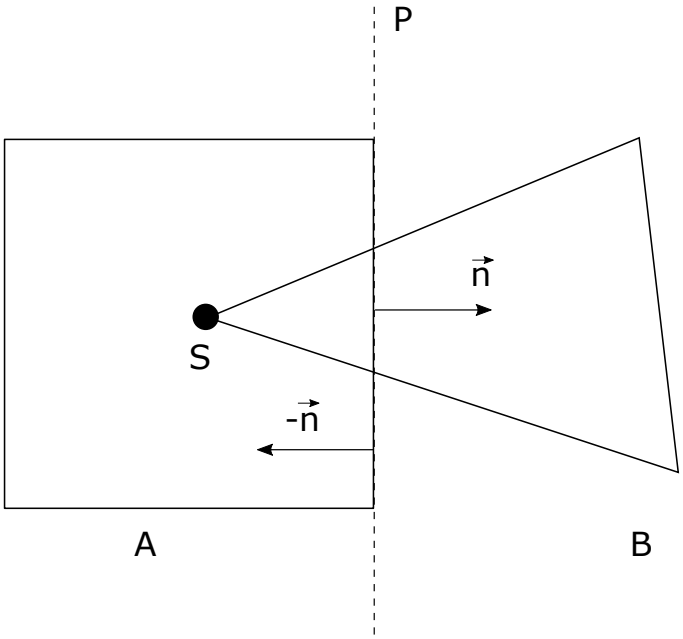
Algoritmus 2: Modifikovaná verzia *Separating-Axis Test* pre konvexné mnohosteny.

```
   input  : Two convex hulls hullA, hullB
   output: Collision query query
 1 bestDistance ← −∞;
 2 bestFaceIndex ← −1;
 3 for faceIndex ← 0 to hullA.faceCount do
 4 |   face ← hullA.faces[faceIndex];
 5 |   plane ← buildPlaneFromFace(face);
 6 |   supportPoint ← getSupportPoint(hullB, -plane.normal);
 7 |   distance ← distancePointFromPlane(supportPoint, plane);
 8 |   if distance > 0.0 then
 9 |   |   query.separationDistance ← ∞;
10 |   |   return query;                          // no possible collision
11 |   end
12 |   if distance > bestDistance then
13 |   |   bestDistance ← distance;
14 |   |   bestFaceIndex ← faceIndex;
15 |   end
16 end
17 query.objectA ← hullA;
18 query.objectB ← hullB;
19 query.faceIndex ← bestFaceIndex;
20 query.separationDistance ← bestDistance;
21 return query;
```

Iteruje sa cez všetky steny mnohostenu A, pričom pre každú stenu sa vytvorí rovina P, v ktorej leží daná stena. Následne sa nájde tzv. *support point*[1] S, mnohostenu B v smere opačnej normály roviny P (viď obrázok 4.3). Ten sa nájde ortogonálnym priemetom všetkých vrcholov mnohostenu B na opačnú normálu $-\vec{n}$ roviny P. Najextrémnejší bod je potom bod s najväčším priemetom. Priemetom tohto bodu na normovanú normálu roviny je následne získaná jeho vzdialenosť od roviny P. Táto vzdialenosť je so znamienkom čo umožňuje určiť, na ktorej strane roviny sa bod nachádza. Ak je vzdialenosť kladná, bod leží mimo útvar A, čo znamená že algoritmus môže predčasne skončiť a útvary sa určite neprekrývajú. Ak algoritmus neskončil predčasne, mnohosteny sa prekrývajú. Algoritmus v tomto prípade vracia index steny, pri ktorej bola najväčšia vzdialenosť (keďže vzdialenosti pri prieniku sú záporné je to je vzdialenosť najbližšia k nule) a aj samotnú vzdialenosť. Normála výslednej steny je nazývaná *kolízna normála* a bude použitá na výpočet reakcie na kolíziu. Absolútna hodnota vzdialenosti udáva hĺbku prieniku útvaru B v útvare A pozdĺž kolíznej normály.

[1] *Support point* je najextrémnejší bod v danom smere.

Tento algoritmus sa vykoná dvakrát. Najskôr sa hľadá *support point* útvaru B pre steny útvaru A a potom naopak. Ak algoritmus neskončil predčasne, z oboch výsledkov sa vyberie ten s menšou hĺbkou prieniku.

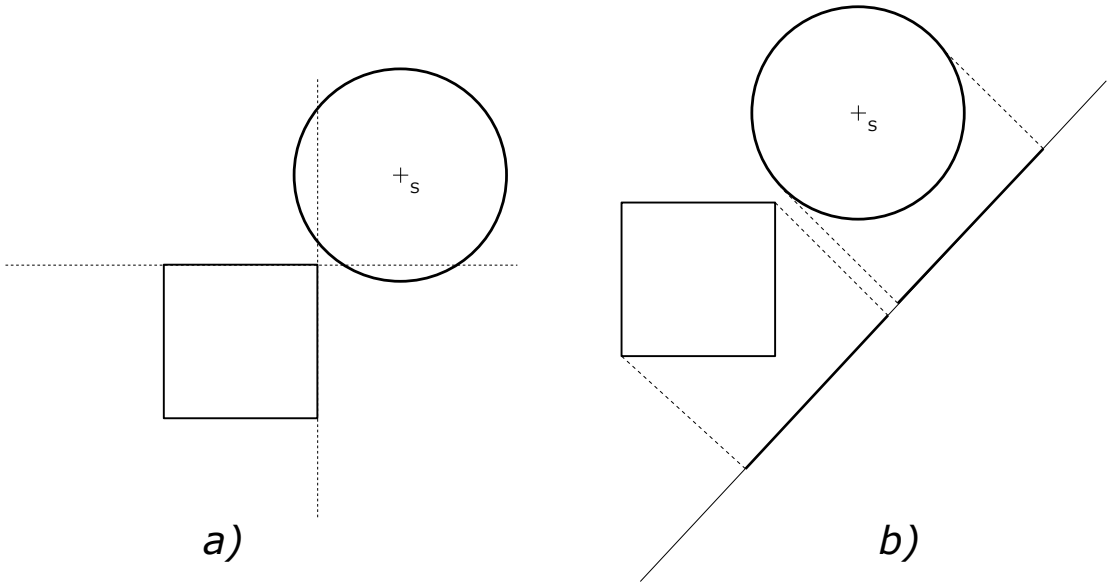


Obr. 4.3: Nájdenie najextrémnejšieho bodu S útvaru B v smere normály $-\vec{n}$. Táto normála je opačná k normále roviny P.

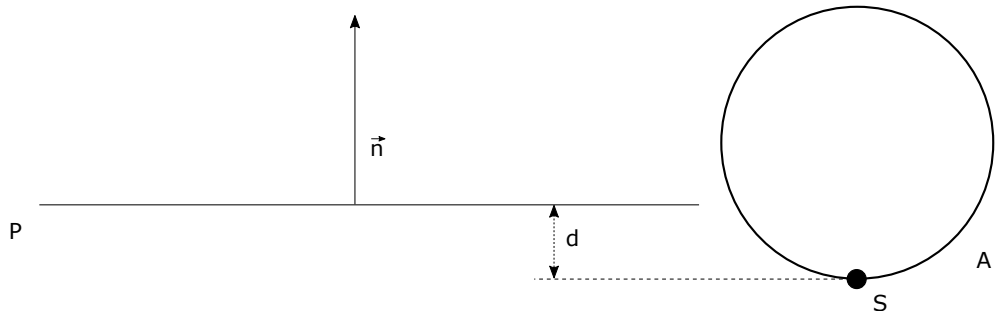
Vyššie spomenutý algoritmus funguje len pre dva konvexné mnohosteny. S menšou úpravou sa však dá použiť aj na detekciu kolízie medzi mnohostenom a guľou. Steny mnohostenu sa použijú na vytvorenie rovín tak ako v hore spomenutom prípade, a *support point* gule sa nájde posunutím stredu gule o dĺžku jej polomeru v smere kolíznej normály. V niektorých prípadoch keď guľa smeruje k hrane mnohostenu (viď obrázok 4.4), normály stien ako potenciálne deliace osi nestačia na vylúčenie kolízie. Detektor v tomto prípade predčasne hlási kolíziu. Z tohto dôvodu sa pri detekcii kolízie medzi guľou a mnohostenom používa ďalší test. Ak detektor nenašiel deliacu os medzi normálami stien, ako ďalšie potenciálne deliace osi sa použijú normované spojnice stredu gule a stredov všetkých hrán mnohostenu. Tieto sa testujú nemodifikovanou verziou algoritmu *SAT* popísanou v kapitole 2.1. Tento test je znázornený na obrázku 4.4.

Podobný postup sa používa aj na zistenie kolízie medzi rovinou a guľou. V tomto prípade má však rovina len jednu stenu a vzdialenosť najextrémnejšieho bodu gule v smere zápornej normály roviny môže byť záporná (čo značí prienik) aj keď sa útvary neprekrývajú. Tento špeciálny prípad je znázornený na obrázku 4.5. Vyriešené to je dodatočnými testami, kedy sa za ďalšie testované roviny zvolia bočné roviny, ktoré sú kolmé na rovinu P a patria do nich hrany tejto roviny.

Najjednoduchší test na detekciu kolízie je medzi dvomi guľami. Ako bolo spomenuté v kapitole 2.3, dve gule kolidujú ak vzdialenosť ich stredov je menšia ako súčet ich polomerov. Kolízna normála je v tomto prípade normovaná spojnica stredov gúľ a hĺbka prieniku je rozdiel vzdialenosti ich stredov a súčtu polomerov.



Obr. 4.4: ***(a)*** V prípade keď guľa smeruje k hrane mnohostenu, testovanie normál stien ako jediných potenciálnych deliacich osí nepostačuje na vylúčenie kolízie. V tomto prípade detektor hlási kolíziu. ***(b)*** Použitie spojnice stredu gule a hrany ako potenciálnej deliacej osi dokázalo vylúčiť kolíziu. Projekčné intervaly útvarov na tejto osi sa neprekrývajú.

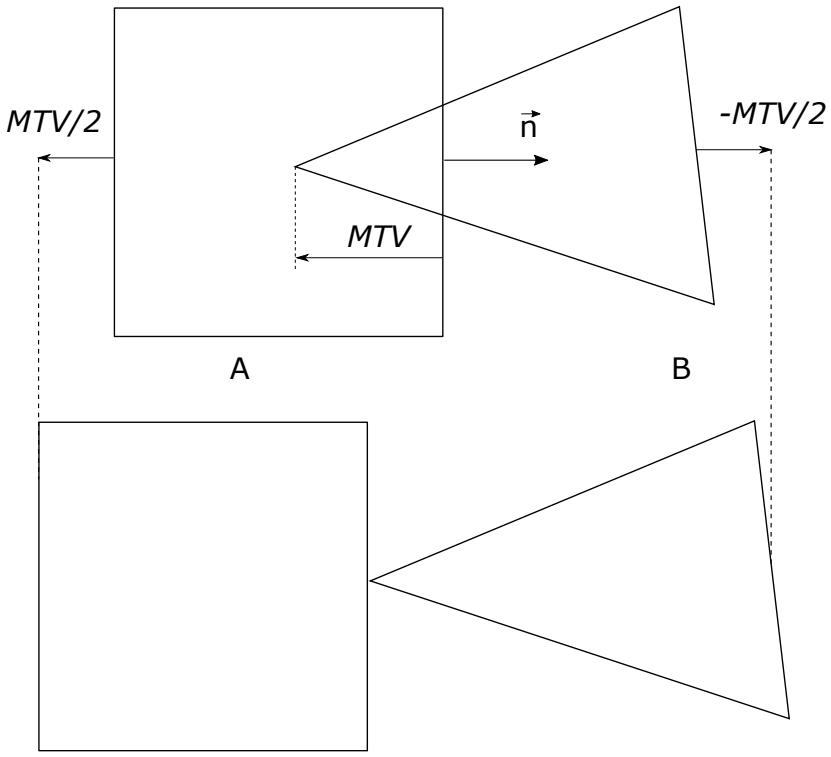


Obr. 4.5: Pri použití iba steny roviny P na test kolízie s guľou A by algoritmus hlásil kolíziu aj keby bola guľa mimo rovinu. Vzdialenosť d najextrémnejšieho bodu S od roviny P je v tomto prípade záporná aj keď sa útvary neprekrývajú.

4.7.1 Vysunutie objektov z kolízie

Kolízia dvoch objektov je detekovaná až keď sa objekty prekrývajú. Aby bola zabezpečená tuhosť objektov a tým realistickosť simulácie, je vhodné pri zistení kolízie okamžite tieto objekty z kolízie vysunúť. Na vysunutie objektov sa môže použiť *minimum translation vector (MTV)*. *MTV* je minimálny vektor, ktorý sa dá použiť na vysunutie objektov do pozície, v ktorej už spolu nekolidujú. *MTV* sa jednoducho získa vynásobením normovanej kolíznej normály a hĺbky prieniku, ktoré boli získané pri detekcii kolízie. Je dobrou praktikou mať tento vektor orientovaný vždy rovnako – napríklad od druhého objektu k prvému. Prvý objekt sa v tomto prípade posunie v smere *MTV* a druhý v opačnom smere, každý o polovicu dĺžky *MTV*. Realistickejšie určenie dĺžky posunu každého objektu, môže byť realizované na základe pomerov ich hmotností. Ťažší objekt sa posunie o menšiu vzdialenosť ako ľahší. Vysúvanie objektov z kolízie za pomoci *MTV* je znázornené na obrázku 4.6.

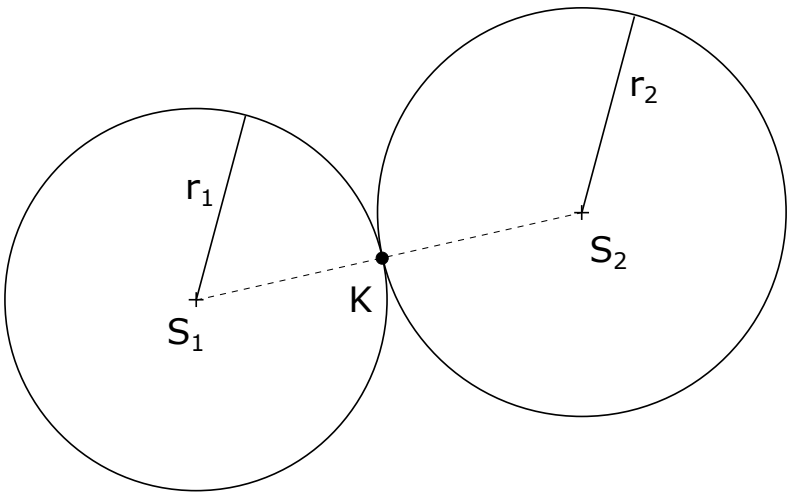
Aj keď objekty po vysunutí už spolu nekolidujú, ich rýchlosti ostávajú rovnaké takže v nasledujúcom snímku by objekty znovu prišli do kontaktu. Je preto potrebné zabezpečiť nejakú interakciu medzi objektmi, ktorá by zmenila ich rýchlosť. Z fyziky je známe, že keď sa objekty zrazia, pôsobia na seba silou a táto sila spôsobuje, že objekty sa od seba napríklad odrazia alebo ostanú stáť na mieste. Reakcia objektov na kolíziu je popísaná v kapitole 4.9.



Obr. 4.6: Použitie *MTV* na vysunutie objektov z kolízie. *MTV* smeruje od objektu *B* k objektu *A*, takže objekt *A* je vysunutý o poloviční *MTV* a objekt *B* je takisto vysunutý o poloviční *MTV* ale do opačnej strany. Týmto je zabezpečené vysunutie objektov do polohy, v ktorej sa neprekrývajú (obrázok dole). *MTV* je rovnobežný s kolíznou normálou $\vec{n}$.

4.7.2 Nájdenie kolíznych bodov

Na výpočet realistickej reakcie na kolíziu je potrebné okrem kolíznej normály poznať aj presné body kontaktu dvoch objektov. Najjednoduchší prípad je nájdenie kolízneho bodu medzi dvomi guľami. Ako je možné vidieť na obrázku 4.7, po vysunutí gúľ z kolízie pomocou *MTV*, leží bod kolízie na spojnici stredov gúľ vo vzdialenosti polomeru od stredu danej gule. Kolízny bod mnohostenu a gule sa nájde podobne. Tento bod leží v smere kolíznej normály od stredu gule, vo vzdialenosti jej polomeru.

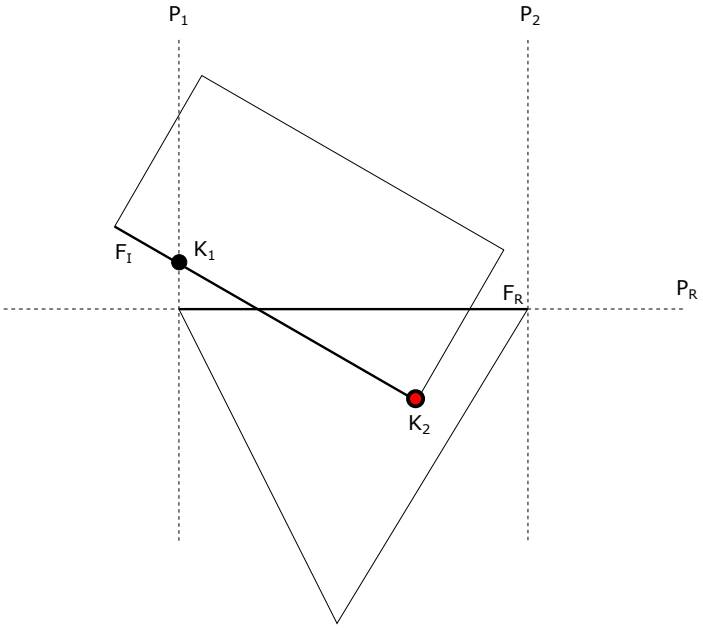


Obr. 4.7: Kolízny bod dvoch gúľ sa nachádza na spojnici ich stredov.

Na nájdenie kolíznych bodov medzi dvomi mnohostenmi treba použiť orezávací algoritmus. Najskôr však treba identifikovať dve steny, ktoré sa prekrývajú. Prvá stena, nazýva sa referenčná (*reference face*), je stena ktorá bola vrátená algoritmom *SAT*. Druhá, incidenčná stena (*incident face*), je stena druhého mnohostenu, ktorá je k prvej stene najviac antiparalelná. Na jej nájdenie stačí preiterovať cez všetky steny druhého mnohostenu a spraviť skalárny súčin ich normál s normálou referenčnej steny. Stena s najmenším skalárnym súčinom je hľadaná stena. Teraz je pomocou orezávacieho algoritmu možné nájsť kolízne body. Orezáva sa incidenčná stena s bočnými rovinami referenčnej steny, viď obrázok 4.8. V tejto práci je na orezanie použitý algoritmus Sutherland-Hodgman [17]. Tento algoritmus berie na vstupe vrcholy tvoriace polygón a rovinu, s ktorou polygón oreže. Výstupom algoritmu sú body tvoriace orezaný polygón. Ak má mnohosten koplanárne steny, algoritmus nenájde všetky správne kolízne body. Z tohto dôvodu sa pri vytváraní mnohostenu, popísanom v kapitole 4.5, zjednocujú koplanárne steny. [7]

Najskôr sa oreže polygón tvoriaci incidenčnú stenu s prvou bočnou rovinou referenčnej steny. Výsledný polygón sa oreže s ďalšou bočnou rovinou a takto sa pokračuje so všetkými bočnými rovinami. Môže sa stať, že algoritmus vráti menej ako tri body a z nich sa nedá zložiť polygón. V tomto prípade sa hľadanie kolíznych bodov ukončí a pokračuje sa ako keby kolízia nenastala. V nasledujúcih snímkoch sa však hľadanie kolíznych bodov podarí, takže takéto ukončenie nie je problém.

Z bodov, ktoré ostanú po orezaní so všetkými bočnými rovinami sa ako kolízne body ponechajú len tie, ktoré sa nachádzajú pod referenčnou rovinou, viď obrázok 4.8. Pre zjednodušenie výpočtu reakcie na kolíziu sú kolízne body ďalej zredukované na jediný bod pomocou aritmetického priemeru.



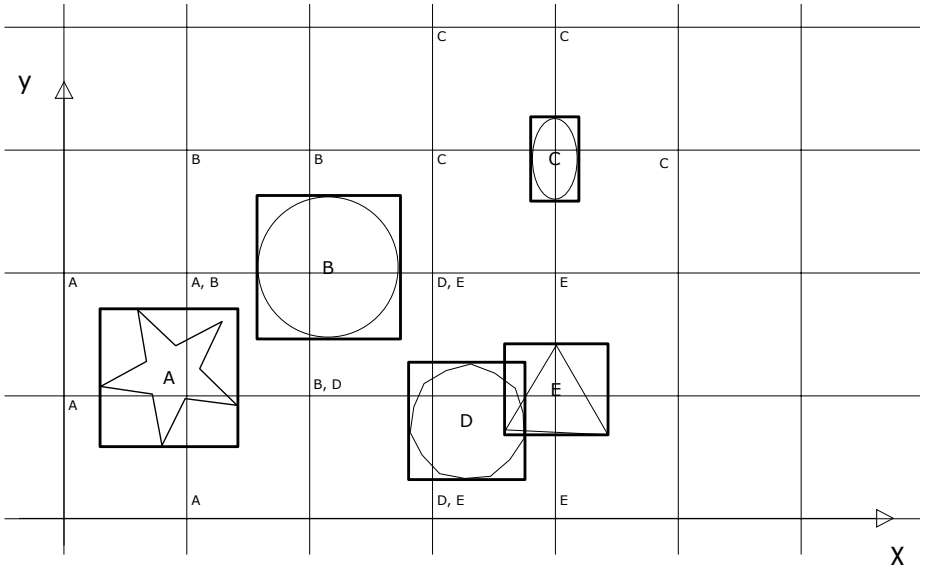
Obr. 4.8: Kolízne body dvoch mnohostenov sa nájdu orezaním incidenčnej steny F_I s bočnými rovinami P_1 a P_2 referenčnej steny F_R. Kolízne body sú všetky výsledné body, ktoré sa nachádzajú pod referenčnou rovinou P_R. V tomto prípade to je len bod K_2.

4.8 Detekcia kolízie – Broad Phase

Broad phase pri veľkom počte objektov v scéne mnohonásobne znižuje čas detekcie kolízie (viď kapitola 5). Bez *broad phase* sa musia testovať všetky dvojice objektov. Úlohou *broad phase* je minimalizovať počet testov tým, že sa testujú len objekty, ktoré sú v tesnej blízkosti.

V tejto práci je na minimalizáciu testovaných dvojíc objektov použitá rovnomerná mriežka popísaná v kapitole 2.3. Mriežka je implementovaná v triede `Grid`. Je to vlastne trojrozmerné pole buniek. Každá bunka obsahuje zoznam statických a zoznam dynamických objektov, ktoré sú v nej uložené. Objekty sa do mriežky vkladajú na základe ich *AABB*. Vložia sa do každej bunky, ktorá obsahuje ich obalové teleso, viď obrázok 4.9. Statické objekty sa vložia na začiatku simulácie a ich uloženie sa ďalej nemení. Uloženie dynamických objektov v mriežke sa prepočítava každý simulačný krok. Najskôr sa v každej bunke vyprázdni zoznam dynamických objektov. Aby sa nemuseli prechádzať všetky bunky mriežky, je v nej uložený zoznam obsadených buniek. Následne sa do mriežky vkladajú objekty. Vždy keď sa objekt vkladá do nejakej bunky, testuje sa jeho kolízia s ostatnými objektmi, ktoré sa v tejto bunke nachádzajú. Na rýchle vylúčenie kolízie sa najskôr skontroluje prekrytie ich *AABB* (viď kapitola 2.3). Ak sa prekrývajú, ich kolízia sa skontroluje presnejším algoritmom, tak ako to bolo popísané v kapitole 4.7.

Používateľ má možnosť zvoliť si či chce spustiť simuláciu s *broad phase* alebo bez. Program sa ho na to spýta v príkazovom riadku pred štartom simulácie.



Obr. 4.9: Spôsob uloženia objektov v rovnomernej mriežke. Objekty sú vkladané do buniek, ktoré obsahujú ich *AABB*. Na kolíziu sa testujú len dvojice objektov, ktoré zdieľajú rovnakú bunku. Potenciálne kolízne dvojice: (A, B), (B, D), (D, E).

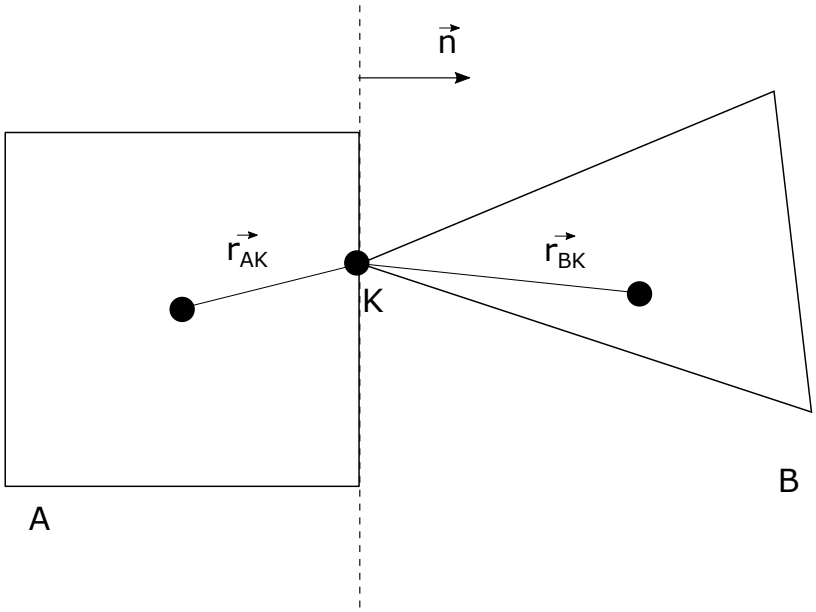
4.9 Reakcia na kolíziu

Po tom ako je detekovaná kolízia dvoch objektov, treba určiť ako na ňu objekty zareagujú. Na zabránenie kolízie objektov v nasledujúcom snímku, treba okamžite zmeniť ich rýchlosť. Jednoduchá a presná metóda, ktorá toto umožňuje je impulzná reakcia na kolíziu. Na okamžitú zmenu rýchlostí používa veličinu *impulz*. Rýchlosti objektov po kolízii sa vypočítajú podľa vzorcov uvedených v rovnici (4.2) [12]. Kolízia dvoch pevných telies je znázornená na obrázku 4.10.

$$\begin{aligned} v_{A_2} &= v_{A_1} + \frac{j}{M_A} n \\ v_{B_2} &= v_{B_1} - \frac{j}{M_B} n \\ L_{A_2} &= L_{A_1} + r_{AK} \times j \\ L_{B_2} &= L_{B_1} - r_{BK} \times j \\ \omega_A &= I_A^{-1} \times L_{A_2} \\ \omega_B &= I_B^{-1} \times L_{B_2} \end{aligned} \tag{4.2}$$

Vysvetlivky k rovnici (4.2):

- v_{A_1}, v_{B_1} – lineárna rýchlosť telies pred kolíziou
- v_{A_2}, v_{B_2} – lineárna rýchlosť telies po kolízii



Obr. 4.10: Kolízia telies A a B. Vektory $r_{\vec{AK}}$ a $r_{\vec{BK}}$ smerujú od ťažiska telesa ku kolíznemu bodu K.

- L_{A_1}, L_{B_1} – moment hybnosti telies pred kolíziou
- L_{A_2}, L_{B_2} – moment hybnosti telies po kolízii
- ω_A, ω_B – uhlové rýchlosti telies
- r_{AK}, r_{BK} – vektory vedúce z ťažiska telies do kolízneho bodu K
- I_A, I_B – tenzory zotrvačnosti telies vo *world-space*
- j – impulz

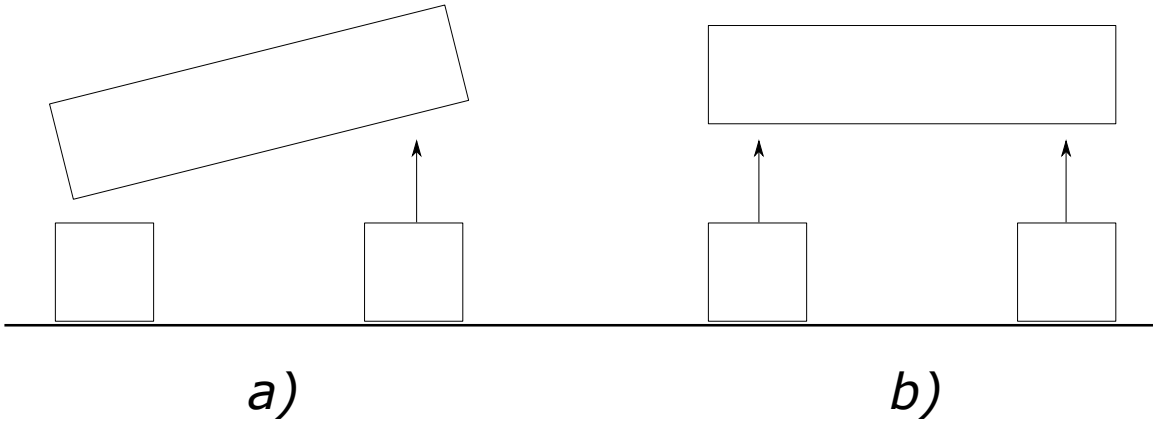
Navýšenie rýchlosti o impulz a výpočet uhlovej rýchlosti z novo-vypočítaného momentu hybnosti sa neaplikuje hneď. Okamžitá aplikácia by ovplyvnila výpočet impulzu pri inej kolízii tohto telesa v aktuálnom simulačnom kroku, viď obrázok 4.11. Preto sa impulzy agregujú a aplikujú sa až na konci simulačného kroku.

Impulz sa vypočíta podľa vzorca (4.3).

$$sj = \frac{-(1+e)v_{AB} \cdot n}{n \cdot n\left(\frac{1}{M_A} + \frac{1}{M_B}\right) + \left[\left(I_A^{-1}(r_{AK} \times n)\right) \times r_{AK} + \left(I_B^{-1}(r_{BK} \times n)\right) \times r_{BK}\right] \cdot n} \tag{4.3}$$

Vysvetlivky k vzorcu (4.3):

- e – koeficient reštitúcie
- v_{AB} – relatívna rýchlosť telies v kolíznom bode K



Obr. 4.11: Rozdiel v spôsobe aplikácie impulzov pri páde hranola na dve kocky. ***(a)*** Pri okamžitej aplikácii impulzu dochádza najskôr k aplikácii impulzu jednej kolízie, čo ovplyvní následný výpočet impulzu druhej kolízie. Jedna strana hranola sa odrazí vyššie ako tá druhá. ***(b)*** Pri aplikácii agregovaných impulzov na konci simulačného kroku sa rotácie spôsobené impulzmi vyrovnajú a obidve strany hranola sa odrazia do rovnakej výšky.

- n – kolízna normála
- M_A, M_B – hmotnosti telies

Koeficient reštitúcie vyjadruje aká časť energie je rozptýlená pri kolízii [11]. Je to číslo v intervale $\langle 0,0; 1,0\rangle$.

Aj keď kolízny bod K má rovnakú pozíciu pri oboch telesách, jeho rýchlosť v_{AK} a v_{BK} môže byť pri oboch telesách rôzna. Rýchlosť kolízneho bodu sa vypočíta podľa rovnice (4.4).

$$\begin{aligned} v_{AK} &= v_{A_1} + \omega_A \times r_{AK} \\ v_{BK} &= v_{B_1} + \omega_B \times r_{BK} \end{aligned} \tag{4.4}$$

Relatívna rýchlosť telies v kolíznom bode K sa vypočíta podľa rovnice (4.5).

$$v_{AB} = v_{AK} - v_{BK} \tag{4.5}$$

V prípade keď je jedno z telies statické, vzorec (4.3) je možné zjednodušiť. Ako je možné vidieť vo vzorci (4.6), týmto vypadne veľká časť výpočtu.

$$j = \frac{-(1+e)v_{AK} \cdot n}{n \cdot n\frac{1}{M_A} + \left[\left(I_A^{-1}(r_{AK} \times n)\right) \times r_{AK}\right] \cdot n} \tag{4.6}$$

4.10 Simulácia

Trieda `Simulation` spravuje celú simuláciu. Pred začatím samotnej simulácie inicializuje všetky moduly (renderer, správca modelov, detektor kolízie, ...). Po úspešnej inicializácii sa spustí metóda `run()`, v ktorej je implementovaný hlavný simulačný cyklus popísaný v algoritme 1.

V metóde `update()` sa vykonáva väčšina práce simulátoru. Metóda je popísaná v algoritme 3. Vykonáva sa v nej numerická integrácia všetkých potrebných veličín. Na integrovanie je použitá Eulerova semi-implicitná metóda. Obyčajný Euler by počítal pozíciu telesa v nasledujúcom kroku z rýchlosti v aktuálnom kroku. Eulerova semi-implicitná metóda však používa na výpočet pozície v nasledujúcom kroku rýchlosť telesa v nasledujúcom kroku.

Ako bolo spomenuté v kapitole 4.3, rotácia je uložená ako 3x3 rotačná matica. Na získanie rotácie v nasledujúcom kroku treba vynásobiť antisymetrickú maticu uhlovej rýchlosti s rotačnou maticou v aktuálnom kroku. Antisymetrická matica vektoru uhlovej rýchlosti $\tilde{\omega}$ sa získa podľa vzťahu vo vzorci (4.7).

$$\tilde{\omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{4.7}$$

Novovzniknutú rotačnú maticu treba následne ortogonalizovať, pretože numerická integrácia nie je úplne presná a po čase by rotačná matica prestala byť ortogonálna.

Po integrácii všetkých veličín všetkých objektov sa vykoná detekcia kolízie, ktorá poskytne zoznam dvojíc objektov ktoré sú v kolízii a takisto aj ich bod kolízie. Metóda `solveCollisions()` následne prejde tento zoznam, vypočíta impulzy potrebné na vyriešenie kolízie a uloží ich do akumulátoru rýchlosti a momentu hybnosti pre konkrétny objekt. Metóda `applyImpulses()` potom pripočíta k rýchlosti objektu akumulovanú rýchlosť z impulzov v aktuálnom simulačnom kroku a vypočíta uhlovú rýchlosť na základe momentu hybnosti.

Algoritmus 3: Metóda `update()` zabezpečujúca vykonanie jedného kroku simulácie.

```
1  msPerUpdate ← 0.01 ;                                   // simulation step
2  foreach object in scene do
3      if objectIsStatic() then
4          continue;
5      end
6      acceleration ← force / objectMass;
7      velocityVector ← velocityVector + msPerUpdate * acceleration;
8      angularVelocity ← inverseWorldInertiaTensor * angularMomentum;
9      applyDamping();
10     position ← position + msPerUpdate * velocityVector;
11     rotation ← rotation + msPerUpdate * skewSymmetricMatrix(angularVelocity)
        * rotation;
12     reorthogonalizeRotationMatrix();
13     computeInverseWorldInertiaTensor();
14     recomputeAABB();
15 end
16 checkCollision();
17 solveCollisions();
18 applyImpulses();
```

Kapitola 5

Experimenty

V tejto kapitole sú popísané experimenty vykonávané s výsledným simulátorom a prezentácia nameraných výsledkov. Cieľom tejto práce nebolo vytvoriť čo najpresnejší simulátor, ale simulátor ktorý je výkonný a pohyby objektov v scéne vyzerajú prirodzene. Výsledky prezentované v tejto kapitole majú za cieľ demonštrovať prínos použitých optimalizácií a dokázať splnenie cieľa tejto práce.

5.1 Pohyb telies

Experimentovaním so simulátorom bolo zistené, že objekty v scéne sa pohybujú prirodzene podľa fyzikálnych zákonov. Jediným nedostatkom ich pohybu sú občasné výbuchy v hromade objektov alebo mierne vibrovanie pri kĺzaní sa po inom objekte (napr. po podlahe), spôsobené impulzmi pri reakcii na kolíziu.

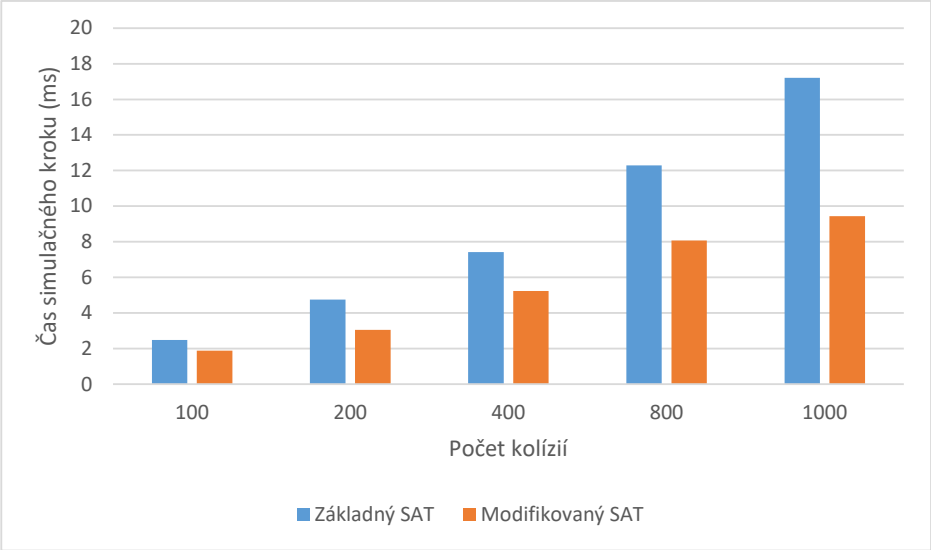
5.2 Popis výkonnostných experimentov

V tejto sekcii sú popísané výkonnostné experimenty, ktoré sú vyhodnotené v nasledujúcich sekciách. Pre tieto experimenty boli vytvorené scény, ktorých základ tvorila rovina slúžiaca ako podlaha. Na túto podlahu bol umiestnený určitý počet objektov *icosphere* s dvadsiatimi stenami, tak aby všetky tieto objekty v čase spustenia simulácie kolidovali s podlahou. Výsledné merania boli získané aritmetickým priemerom času výpočtu prvých sto simulačných krokov. Výpočet simulačného kroku zahŕňa integráciu pozícií a kinematických veličín objektov, detekciu kolízie a výpočet reakcie na zistené kolízie. Pri použití *broad phase* to navyše zahŕňa výpočet $AABB$ všetkých dynamických objektov a aktualizáciu mriežky.

Všetky experimenty popísané v tejto kapitole boli vykonávané na notebooku s procesorom Intel® Core™ i5-6200U s frekvenciou 2,3GHz, grafickou kartou Intel® HD Graphics 520, 8GB RAM a operačným systémom Windows 10.

5.3 Modifikovaná metóda SAT

Ako bolo spomenuté v kapitole 4.7, v simulátore bola namiesto základnej verzie metódy SAT použitá modifikovaná verzia. Na grafe 5.1 je vidieť, že táto zmena viedla k výraznému zvýšeniu výkonu. Detailné porovnanie oboch verzií je v grafe 5.2.

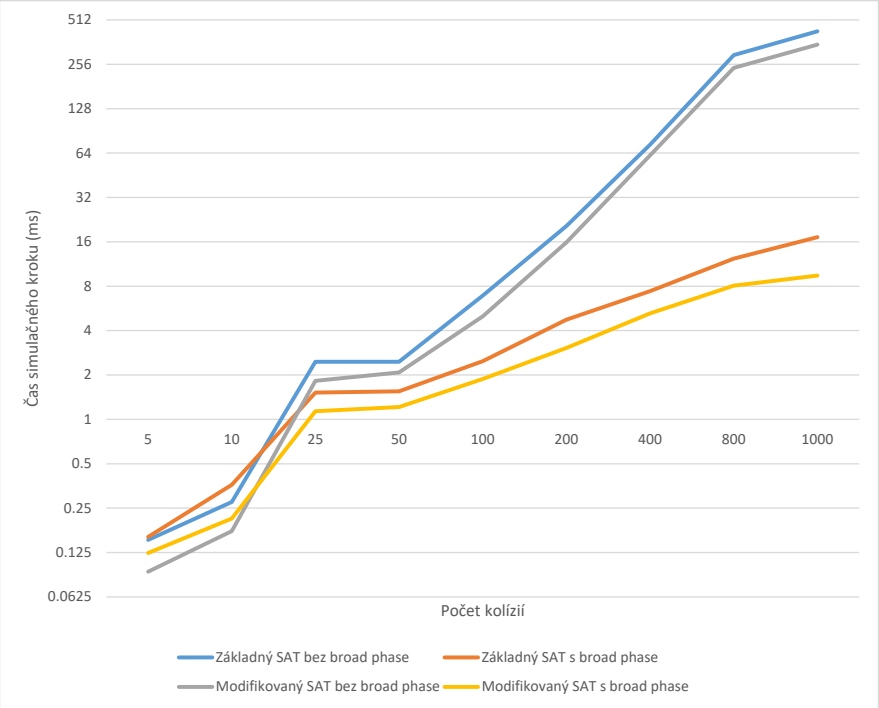


Graf 5.1: Porovnanie základnej a modifikovanej verzie metódy *SAT*. Graf zobrazuje čas výpočtu jedného simulačného kroku. Pri meraní bola použitá aj *broad phase*.

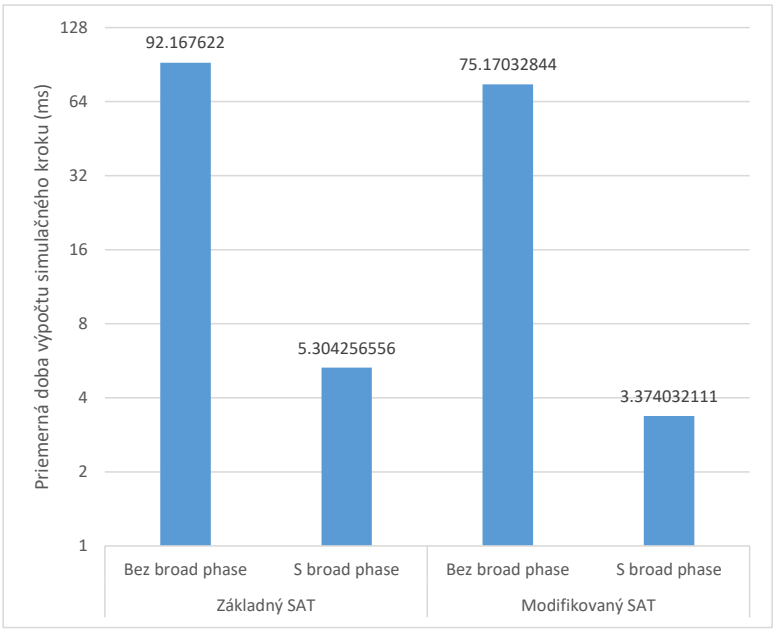
5.4 Použitie broad phase

Použitie *broad phase* pri detekcii kolízie môže výrazne znížiť čas detekcie kolízie (viď kapitola 2.3). Toto je možné vidieť v grafe 5.2. Graf znázorňuje čas výpočtu simulačného kroku pre rôzny počet kolízií. Sú v ňom porovnané všetky namerané časy s použitím *broad phase* a bez, a takisto oboch verzií *SAT*. Z grafu vyplýva, že použitie *broad phase* nie je výhodné pri malom počte objektov v scéne, pretože réžia spojená s vytváraním mriežky je pre taký malý počet objektov príliš vysoká.

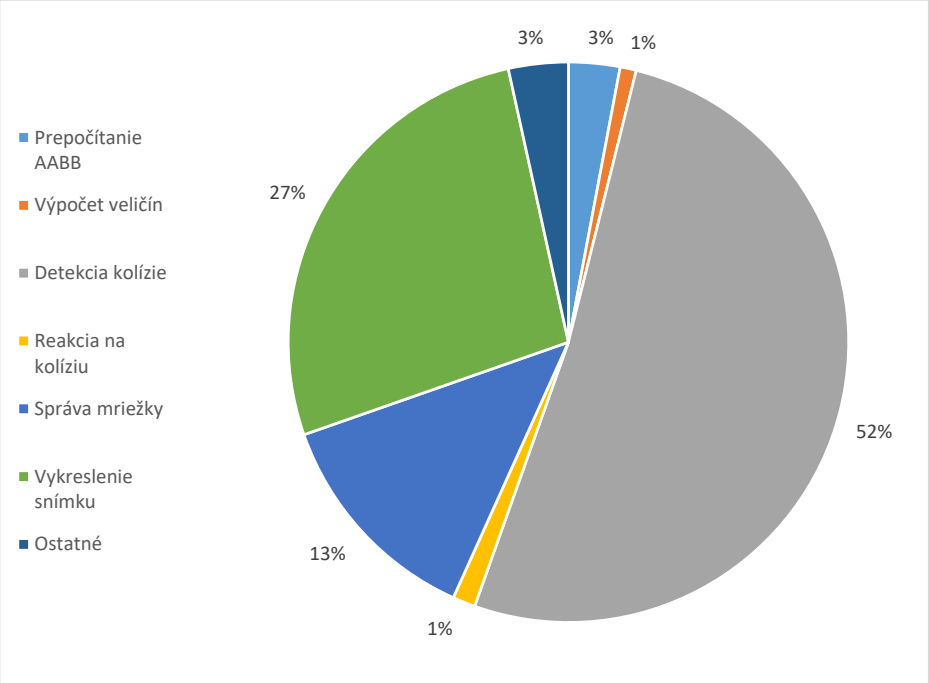
Porovnanie priemerného času výpočtu simulačného kroku pre všetky metódy je v grafe 5.3. V grafe 5.4 je zobrazený podiel jednotlivých častí simulácie na trvaní jedného snímku. Tieto hodnoty boli získané pri kolízii približne dvesto objektov s použitím *broad phase*. Na grafe jasne vidno, že pri veľkom počte kolidujúcich objektov, detekcia kolízie trvá viac ako polovicu času celého snímku.



Graf 5.2: Porovnanie nameraných hodnôt pre rôzne metódy detekcie kolízie. Graf zobrazuje čas simulačného kroku (v milisekundách) v závislosti na počte kolízií.



Graf 5.3: Porovnanie priemerného času výpočtu simulačného kroku (v milisekundách) pre rôzne metódy použité na detekciu kolízie.



Graf 5.4: Zobrazenie podielu jednotlivých častí simulátoru na čase snímku. Výpočet veličín zahŕňa integráciu pozície, rýchlosti a iných kinematických veličín objektov. Správa mriežky zahŕňa vymazanie dynamických objektov z mriežky a ich následné vloženie na základe nových pozícií.

Kapitola 6

Záver

Cieľom tejto práce bolo vytvoriť simulátor pevných telies, ktorý by sa dal použiť ako základ na vytvorenie počítačovej hry zameranej na fyziku. Na základe experimentov s vytvoreným programom je možné skonštatovať, že cieľ sa podarilo splniť. Na vytvorenie konečného výsledku bolo potrebné naštudovať dve hlavné oblasti: detekciu kolízie a simuláciu fyziky pevných telies. Aby bolo možné spojiť tieto dve oblasti do jedného celku, bolo potrebné naštudovať tvorbu herného enginu a takisto aj vykresľovanie grafiky pomocou OpenGL.

Simulátor, ktorý je výstupom tejto práce, je implementovaný v jazyku C++. Simulácia fyziky je implementovaná pomocou základných fyzikálnych zákonov, ktoré popísal Isaac Newton. Na detekciu kolízie je požitá modifikovaná metóda *Seperating-axis Test*. Pri veľkom počte objektov v scéne je potrebné znížiť počet testovaných dvojíc, čo bolo dosiahnuté rozdelením scény do rovnomernej mriežky a testovaním len tých objektov, ktoré zdieľajú rovnakú bunku. Scéna sa načítava z textového súboru vlastného formátu.

Do budúcna by sa dala aplikácia vylepšiť mnohými spôsobmi. Jedným z najvýznamnejších vylepšení by bolo pridanie podpory pre konkávne objekty. Výkon simulátoru by sa dal výrazne zvýšiť hardvérovou akceleráciou kolízie detekcie na GPU. Ako ďalšie vylepšenia by mohli byť: podpora textúr, vylepšená grafika, GUI, editor scén, pridanie kĺbov, lepšie riešenie kolízie pre viac súčasných kolíznych bodov, realistické trenie alebo lepšia integračná metóda.

Literatúra

[1] *About CMake*. [Online; navštívené 29.04.2019].
URL https://cmake.org/overview/

[2] *GLFW - An OpenGL library*. [Online; navštívené 29.04.2019].
URL https://www.glfw.org/

[3] Eberly, D.: *Polyhedral Mass Properties (Revisited)*. [Online; navštívené 29.04.2019].
URL https://www.geometrictools.com/Documentation/PolyhedralMassProperties.pdf

[4] Ericson, C.: *Real-Time Collision Detection*. Boca Raton, FL, USA: CRC Press, Inc., 2004, ISBN 1558607323, 9781558607323.

[5] Gilbert, E. G.; Johnson, D. W.; Keerthi, S. S.: A Fast Procedure for Computing the Distance Between Complex Objects in Threedimensional Space. *IEEE Journal of Robotics and Automation*, ročník 4, č. 2, apríl 1988: s. 193–203, ISSN 0882-4967, tiež dostupné z https://ieeexplore.ieee.org/document/2083.

[6] Gregorius, D.: *The Separating Axis Test between Convex Polyhedra*, 2013, Game Developers Conference.
URL https://archive.org/details/GDC2013Gregorius

[7] Gregorius, D.: *Robust Contact Creation for Physics Simulations*, 2015, Game Developers Conference.
URL http://box2d.org/files/GDC2015/DirkGregorius_Contacts.pdf

[8] Gregory, J.: *Game Engine Architecture, Second Edition*. Natick, MA, USA: A. K. Peters, Ltd., druhé vydanie, 2014, ISBN 1466560010, 9781466560017.

[9] Hecker, C.: Physics, Part 1: The Next Frontier. *Game Developer Magazine*, október/november 1996: s. 12–20, ISSN 1073–922X, tiež dostupné z http://chrishecker.com/images/d/df/Gdmphys1.pdf.

[10] Hecker, C.: Physics, Part 2: Angular Effects. *Game Developer Magazine*, december/január 1996: s. 14–22, ISSN 1073–922X, tiež dostupné z http://chrishecker.com/images/c/c2/Gdmphys2.pdf.

[11] Hecker, C.: Physics, Part 3: Collision Response. *Game Developer Magazine*, február/marec 1997: s. 11–18, ISSN 1073–922X, tiež dostupné z http://chrishecker.com/images/e/e7/Gdmphys3.pdf.

[12] Hecker, C.: Physics, Part 4: The Third Dimension. *Game Developer Magazine*, jún 1997: s. 15–26, ISSN 1073–922X, tiež dostupné z http://chrishecker.com/images/b/bb/Gdmphys4.pdf.

[13] Millington, I.: *Game Physics Engine Development (The Morgan Kaufmann Series in Interactive 3D Technology)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, ISBN 012369471X.

[14] Newton, I.; Motte, A.; Cajori, F.: *Sir Isaac Newton's Mathematical Principles of Natural Philosophy and His System of the World*. zv. 1, University of California Press, 1962, ISBN 9780520009271.

[15] Nystrom, R.: *Game Programming Patterns – Game Loop*. URL http://gameprogrammingpatterns.com/game-loop.html

[16] Peringer, P.: *Modelování a simulace - Studijní opora*. FIT VUT v Brně, 2012.

[17] Sutherland, I.; Hodgman, G.: Reentrant Polygon Clipping. *Communications of the ACM*, ročník 17, č. 1, január 1974: s. 32–42, ISSN 0001-0782, tiež dostupné z http://www.cs.gettysburg.edu/~ilinkin/courses/Fall-2014/cs373/handouts/papers/sh-rpc-74.pdf.

[18] Vries, J.: *Creating a window*. [Online; navštívené 29.04.2019]. URL https://learnopengl.com/Getting-started/Creating-a-window