VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KNIHOVNA PRO SIMULACI DYNAMIKY TUHÝCH TĚLES

LIBRARY FOR RIGID BODY DYNAMICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LIBOR MORAVČÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Dr. Ing. PETR PERINGER**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Moravčík Libor**
Program: Informační technologie
Název: **Knihovna pro simulaci dynamiky tuhých těles**
Library for Rigid Body Dynamics
Kategorie: Modelování a simulace
Zadání:

1. Seznamte se s existujícími knihovnami pro simulaci pohybu a kolizí těles v prostoru (2D i 3D). Prostudujte základní principy simulačních počítačových her.
2. Navrhněte knihovnu pro základní simulaci pohybu a kolizí tuhých těles. Pomocí této knihovny navrhněte vhodné ilustrační příklady pro její použití. Zaměřte se na vhodnost použití ve výuce jako úvod do problematiky implementace fyziky v počítačových hrách.
3. Implementujte navržený systém v prostředí C++/WebAssembly a ověřte funkčnost na různých platformách.
4. Zhodnoťte dosažené výsledky a navrhněte možná vylepšení.

Literatura:

- Bourg D.: Physics for Game Developers. O'Reilly Media, 2001.
- Další dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Peringer Petr, Dr. Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Práca predkladá základnú teóriu simulovania dynamiky tuhých telies v počítačových hrách. Praktickým výsledkom práce je knižnica, ktorá názorne implementuje preberané princípy. Tuhé telesá sú zjednodušené na konvexné kolízne útvary v dvojrozmernom prostredí, ktoré je možné spájať. Detekcia kolízie je riešená v dvoch fázach, široká a úzka. Základom širokej fázy je dynamický aabb strom. Pre úzku fázu je použitý Gilbert-Johnson-Keerthi (GJK) s rozšírením o detekciu kolíznych bodov.

Abstract

This thesis sums up a basic knowledge about rigid body simulations in two dimensional space of computer games. Practical result is a hands-on library written in C++. Collision geometry of rigid bodies is simplified to convex polygons and circles. Multiple bodies can be joined together via a joint. Collision detection is split in to two phases, broad and narrow. Broad phase is implemented using a dynamic aabb tree while narrow phase uses Gilbert-Johnost-Keerthi (GJK) algorithm with Expanding Polytope Algorithm as an extension for detecting collision points between two polygons.

Kľúčové slová

dynamika, tuhé teleso, dynamika tuhých telies, kolízie tuhých telies, počítačové hry, klasická mechanika, lineárna algebra, c++, webassembly

Keywords

dynamics, rigid body, rigid body dynamics, rigid body collisions, computer games, classical mechanics, linear algebra, c++, webassembly

Citácia

MORAVČÍK, Libor. *Knihovna pro simulaci dynamiky tuhých těles.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dr. Ing. Petr Peringer

Knihovna pro simulaci dynamiky tuhých těles

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Dr. Ing. Petra Peringera. Uviedol som všetky literárne pramene, publikácie a zdroje z ktorých som čerpal.

.....
Libor Moravčík
4. júna 2020

Poďakovanie

Ďakujem vedúcemu bakalárskej práce pánovi Dr. Ing. Petrovi Peringerovi za účinnú, metodickú, pedagogickú a odbornú pomoc pri spracovaní mojej bakalárskej práce.

Obsah

Kapitola 1

Úvod

Vývojári hier používajú celú sadu rôznych špecializovaných nástrojov a knižníc, pomocou ktorých prostredie hry obohacujú nielen o audiovizuálne prvky. Jednou z nich je aj systém simulujúci fyzikálne zákony [4]. Každá rešpektovaná herná spoločnosť dnes už implementuje prvky reality do svojich hier. Požiadavkami na takýto systém sú stabilita, presnosť, vierohodnosť a nízka výpočtová naročnosť. Stabilita je dôležitá, keďže nechceme aby hra padala a presnosť výpočtov by mala byť v správnom pomere s výpočtovou náročnosťou. Výsledná hra by mala pôsobiť vierohodne. Ak hra pôsobí dojmom, že nesimuluje fyziku korektne, zožne od fanúšikov digitálnej zábavy kritiku a upadne do zabudnutia. Najčastejšie simulovaným prvkom v hrách je mechanika tuhých telies. Tuhé teleso ako abstrakcia reálneho telesa, umožňuje do hry zaniesť prvky reality za pomerne nízke výpočtové náklady a dodáva herným objektom pocit masívnosti a pevnosti. Navyše poskytuje pre laikov vynikajúci názorný úvod do dynamiky a fyziky počítačových hier.

Cieľom tejto práce je poodhaliť tajomstvá skryté v pozadí komplikovaných herných enginov a ukázať čitateľovi, že jednoduchý fyzikálny engine je schopný zostaviť aj sám. Praktickým výstupom práce je knižnica v jazyku C++, ktorá implementuje teoretické poznatky v praxi. V nasledujúcej kapitole si čitateľ môže osviežiť svoje vedomosti z vektorovej algebry [14]. Vektorová algebra ako matematické pozadie je nutnou znalosťou pre popis simulovaných fyzikálnych zákonov. Pre jednoduchosť sa celá praca drží väčšinu času dvojrozmerného prostredia. Tretia kapitola ponúka prehľad Newtonovej mechaniky [11] a definuje pojem tuhého telesa a jeho tvar. Štvrtá kapitola nám zodpovie otázku, či sa nám telesá zrazili a ako riešiť ich zrážku. V piatej kapitole je predstavený návrh knižnice spolu s jednotlivými objektami. Šiesta kapitola ponúka názornú ukážku využitia knižnice a navrhuje možné vylepšenia.

Kapitola 2

Prehľad použitých matematických metód

'Kniha prírody je napísaná jazykom matematiky' - Galileo Galilei

Fyzika je prírodná veda zviazaná v intímnom vzťahu s matematikou. Matematika prináša svoj logický rámec pre popis a testovanie fyzikálnych javov. Fyzika jej na oplátku ponúka nové náhľady a myšlienky. V každom projekte zameranom na simulovanie reality tak nájdeme sadu matematických štruktúr, ktoré slúžia pre popis fyzikálneho modelu. Uvedieme si známe štruktúry lineárnej algebry a ich použitie v súvislosti s počítačovými hrami. Budeme sa pohybovať primárne v dvojrozmernom priestore s malou dopomocou tretieho rozmeru. Súčasne platí predpoklad, že súradnicový systém bude pravotočivý. Jednotlivé osi sú určené pravidlom pravej ruky.

2.1 Vektory

Pod pojom vektor rozumieme prvok vektorového priestoru $\mathbb{X}^n$ nad ktorým je definovaný vektorový súčet a skalárne násobenie. V algebraickom kontexte je vektor $\vec{u} \in \mathbb{X}^n$ pole n prvkov $\vec{u} = (u_1, u_2, ..., u_n)$. Budeme sa pohybovať v priestore nad množinou reálnych čísel s definovanou euklidovskou metrikou $\mathbb{R}^n$. Prakticky nás bude zaujímať $\mathbb{R}^n$, kde $n = 2 \vee n = 3$ s počiatkom súradnicového systému v bode $O = [0, 0]$ pre $n = 2$, resp. v bode $O = [0, 0, 0]$ pre $n = 3$ a ortonormálnou bázou s bázovými vektormi $\vec{i}, \vec{j} \in \mathbb{R}^2$ resp. $\vec{i}, \vec{j}, \vec{k} \in \mathbb{R}^3$. Veľkosť vektora $|\vec{u}|$ v kontexte metrického priestoru $\mathbb{R}^2$ je definovaná ako [1]:

$$|\vec{u}| = \sqrt{\vec{u}_x{}^2 + \vec{u}_y{}^2} \tag{2.1}$$

,kde $\vec{u}_x$ a $\vec{u}_y$ sú súradnice vektora $\vec{u}$. Vektor $\vec{u}$ sa dá zároveň vyjadriť ako lineárna kombinácia bázových vektorov:

$$\vec{u} = a\vec{i} + b\vec{j} \tag{2.2}$$

,kde $a, b \in \mathbb{R}$.

Analogicky pre vektor $\vec{v} \in \mathbb{R}^3; a, b, c \in \mathbb{R}$:

$$|\vec{v}| = \sqrt{\vec{v}_x{}^2 + \vec{v}_y{}^2 + \vec{v}_z{}^2} \tag{2.3}$$

$$\vec{v} = a\vec{i} + b\vec{j} + c\vec{k} \tag{2.4}$$

Na bod v priestore môžeme nahliadať ako na polohový vektor. Ďalej však budeme hovoriť o vektoroch ako prvkoch, ktoré so sebou nesú informáciu o smere a veľkosti a namiesto pojmu polohový vektor budeme používať pojem bod. Pri transformáciách bude toto rozlíšenie dôležité[1].

Skalárny súčin

Nech $\vec{u}, \vec{v} \in \mathbb{R}^n$; $n = 2 \vee n = 3$. Skalárny súčin dvoch vektorov je definovaný ako suma súčinov ich korešpondujúcich prvkov.

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} \vec{u}_i \vec{v}_i \tag{2.5}$$

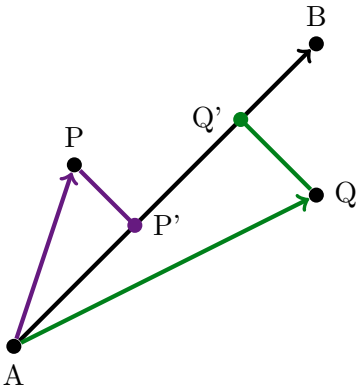
Geometricky môžeme skalárny súčin definovať ako súčin veľkostí vektorov a kosínusu uhla medzi vektormi. Pre skalárny súčin platí komutatívnosť: $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$.

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos \alpha \tag{2.6}$$

Výsledkom skalárneho súčinu je skalárna hodnota, pre ktorú platí:

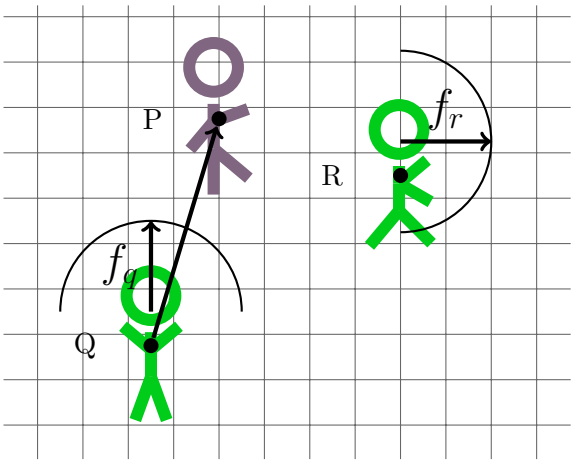
$$\begin{aligned} &a) \quad \vec{u} \cdot \vec{v} < 0 \Rightarrow \alpha > 90^\circ \\ &b) \quad \vec{u} \cdot \vec{v} = 0 \Rightarrow \alpha = 90^\circ \\ &c) \quad \vec{u} \cdot \vec{v} > 0 \Rightarrow \alpha < 90^\circ \end{aligned} \tag{2.7}$$

Pokiaľ sa na skalárny súčin pozrieme ako na priemet vektora na vektor (Obr. 2.1), dostaneme hodnotu rovnajúcu sa vzdialenosti priemetu od počiatku vektora na ktorý premietame. Takýto náhľad nám môže vnuknúť myšlienku využit skalárny súčin pre nájdenie najbližšieho či najvzdialenejšieho bodu v určitom smere. Skalárny súčin nám tak neskôr pomôže pri hľadaní kolíznych bodov[1, 2, 4].



Obr. 2.1: Najbližší bod v smere danom smerovým vektorom $\vec{AB}$, bude bod P, ktorého smerový vektor $\vec{AP}$ bude mať najmenšiu kladnú hodnotu skalárneho súčinu s $\vec{AB}$.

Príkladom využitia skalárneho súčinu v hrách je aj kontrola toho, či sa objekt nachádza v zornom poli iného objektu. Predpokladajme veľkosť zorného poľa objektu hráča a objektu zombie $\angle 180^\circ$ (Obr. 2.2). Smerový vektor $\vec{f_Q}$ pohybu zombie v bode Q a smerový vektor $\vec{f_R}$ pohybu zombie v bode R. Vďaka tomu, že funkcia kosínus je párna, zombie vidí hráča P pokiaľ $\vec{QP} \cdot \vec{f_Q} > 0$. Viď rovnica c) v 2.7 [2].



Obr. 2.2: Hráč P sa nachádza v zornom poli zombie Q, pretože smerový vektor $\vec{QP}$ a smer, ktorým sa pozerá zombie $\vec{f_q}$ majú hodnotu skalárneho súčinu > 0 [2].

Vektorový súčin

Nachvíľu sa presunieme do tretieho rozmeru a definujeme si vektorový súčin.

Nech $\vec{u}, \vec{v}, \vec{w} \in \mathbb{R}^3$. Výsledkom vektorového súčinu $\vec{u}, \vec{v}$ je na ne kolmý vektor $\vec{w}$:

$$\vec{w} = \vec{u} \times \vec{v} = |\vec{u}||\vec{v}| \sin(\alpha)\vec{n} \tag{2.8}$$

,kde $\vec{n} \in \mathbb{R}^3 \wedge |\vec{n}| = 1 \wedge \vec{n} \perp \vec{u} \wedge \vec{n} \perp \vec{v}$. Praktickejší výpočet vektorového súčinu po prvkoch:

$$\begin{aligned} \vec{w}_x &= (\vec{u}_y\vec{v}_z) - (\vec{u}_z\vec{v}_y) \\ \vec{w}_y &= (\vec{u}_z\vec{v}_x) - (\vec{u}_x\vec{v}_z) \\ \vec{w}_z &= (\vec{u}_x\vec{v}_y) - (\vec{u}_y\vec{v}_x) \end{aligned} \tag{2.9}$$

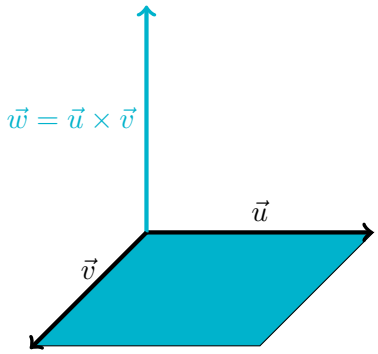
Geometricky je veľkosť vektora $|\vec{w}|$ rovná veľkosti obsahu rovnobežníka skonštruovaného nad vektormi $\vec{u}$ a $\vec{v}$ (Obr. 2.3). Narozdiel od skalárneho súčinu pre vektorový súčin **neplatí** komutatívnosť $\vec{u}\times\vec{v} \neq \vec{v}\times\vec{u}$. Namiesto toho **platí** $\vec{u}\times\vec{v} = -\vec{v}\times\vec{u}$. Pomocou vektorového súčinu vieme nájsť normálový vektor roviny. Pre nájdenie uhla, ktorý zvierajú dva trojrozmerné vektory potrebujeme nájsť tretí vektor, ktorý predstavuje os otáčania. Tú vieme nájsť práve pomocou vektorového súčinu. Výsledná normála bude osou otáčania.

Pozorný čitateľ si môže všimnúť, že v dvojrozmernom priestore je os otáčania vždy rovnobežná s pomyselnou osou z. Pri rotácii telesa v 2D nám tak stačí vedieť veľkosť uhla a bod okolo ktorého chceme teleso otočiť [2, 3, 5].

Vektorový súčin v 2D

Bežná znalosť zo strednej školy je, že vektorový súčin má zmysel len v trojrozmernom priestore. Napriek tomu narazíme často v matematických knižniciach na implementovanú obdobu vektorového súčinu aj pre dvojrozmerný priestor. Nech $\vec{u}, \vec{v} \in \mathbb{R}^2$, potom ich vektorový súčin je definovaný ako:

$$\vec{u} \times \vec{v} = (u_x v_y) - (u_y v_x) \tag{2.10}$$



Obr. 2.3: Veľkosť vektora $\vec{w}$ je rovná veľkosti obsahu štvoruholníka so stranami $\vec{u}$ a $\vec{v}$.

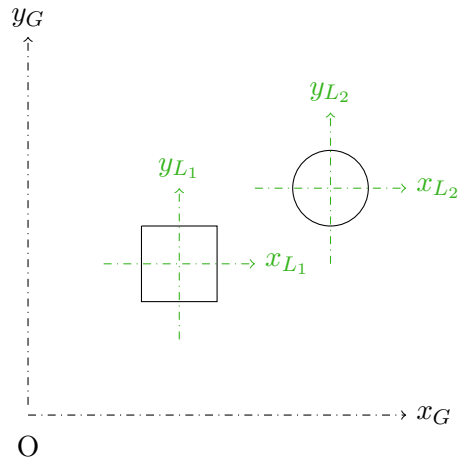
Totožnosť s rovnicou pre výpočet $\vec{w}_z$ v rovnici 2.9 nie je náhodná. Pokiaľ prevedieme dva dvojrozmerné vektory do tretieho rozmeru pridaním z súradnice s hodnotou 0, môžeme využiť vektorový súčin definovaný pre trojrozmerný priestor. Výsledkom takéhoto vektorového súčinu bude vektor $\vec{w} = (0, 0, w_z)$. Na základe znamienka $\vec{w}_z$ vieme určiť relatívnu pozíciu dvoch vektorov. Pokiaľ $\vec{w}_z$ je kladné, vektor $\vec{u}$ bude napravo od vektora $\vec{v}$ a naopak. Vieme povedať, že sa jedná o rotáciu v smere prípadne v protismere hodinových ručičiek. V dynamike v dvojrozmernom priestore môžeme prostredníctvom rovnice 2.10 nájsť napr. uhlovú rýchlosť, ktorá je reprezentovaná skalárnou hodnotou [2][5].

2.2 Transformácie vektorov

Transformácia je zobrazenie, ktoré priradí vektor z jedného súradnicového systému k vektoru z druhého. Pri vytváraní herných objektov sú súradnice vrcholov modelu určené v rámci lokálneho súradnicového systému. Následne takto definovaný model je potrebné uviesť do herného sveta, ktorý má vlastný, globálny súradnicový systém (Obr. 2.4). Keď teleso v hre mení svoju pozíciu, napríklad na základe pôsobenia síl, chceme získať nové súradnice vrcholov. V oboch prípadoch potrebujeme nad zoznamom vrcholov patriacich telesu vykonať transformáciu. Transformáciou súradníc vektora, rozumieme vzťahy medzi súradnicami toho istého vektora, vyjadreného v dvoch rôznych sústavách bázových vektorov. Vďaka tomu ak chceme zistiť ako sa prejaví transformacia na ľubovolnom vektore, môžeme najprv pozorovať ako sa prejaví na bázových vektoroch. Tento fakt si ozrejmíme pri rotáciach. Vo všeobecnosti môžeme transformovať pole n prvkov. Budeme mať však na pamäti, že pri transformáciách rozlišujeme medzi interpretovaním poľa prvkov ako bodu a ako vektora [3].

Posunutie

Posunutie alebo translácia je aditívna **nelineárna** transformácia (pri transformovaní súradnicového systému sa **mení** poloha počiatku systému), ktorej výsledkom je nová poloha bodu. Translácia nám hovorí o koľko jednotiek chceme bod posunúť. Keďže vektor nesie so sebou informáciu o smere a veľkosti, nemá veľmi zmysel vektor posúvať. Majme bod $A \in \mathbb{R}^2$ a vektor $\vec{p} \in \mathbb{R}^2$. Potom nové súradnice posunutého bodu A' pri aplikovaní vektoru posunutia $\vec{p}$ budú [6]:



Obr. 2.4: Dva objekty definované v lokálnych súradniciach sú aplikovaním transformácií položené do globálneho systému.

$$\begin{aligned} A'_x &= A_x + \vec{p}_x \\ A'_y &= A_y + \vec{p}_y \end{aligned} \tag{2.11}$$

Pokiaľ sa na transláciu pozrieme ako na transformáciu súradnicového systému, dostaneme nový súradnicový systém, ktorého nový počiatok $O' = [O_x + \vec{p}_x, O_y + \vec{p}_y]$.

Rotácia [6, 7]

Rotácia je multiplikatívna **lineárna** transformácia (pri transformovaní súradnicového systému sa **nemení** poloha počiatku systému). Rotovať budeme ako vektory tak aj body. Odvodíme si rovnice pre nové súradnice bodu/vektora po aplikovaní rotácie. Majme lokálny súradnicový systém L s počiatkom v bode O = [0,0] a jednotkovými bázovými vektormi $\vec{i}_L, \vec{j}_L \in \mathbb{R}^2$ a globálny súradnicový systém G s rovnakým počiatkom ako L a jednotkovými bázovými vektormi $\vec{i}_G, \vec{j}_G \in \mathbb{R}^2$. Majme bod $P \in L \wedge G$. (Obr. 2.5) Vektor $\vec{OP}$ smerujúci od počiatku súradnicového systému k bodu P bude mať v jednotlivých súradnicových systémoch zápis:

$$\begin{aligned} \vec{OP}^G &= \vec{OP}_x^G \vec{i}_G + \vec{OP}_y^G \vec{j}_G \\ \vec{OP}^L &= \vec{OP}_x^L \vec{i}_L + \vec{OP}_y^L \vec{j}_L \end{aligned} \tag{2.12}$$

Vektor $\vec{i}_L$ a $\vec{j}_L$ môžeme pomocou $\vec{i}_G$ a $\vec{j}_G$ vyjadriť ako:

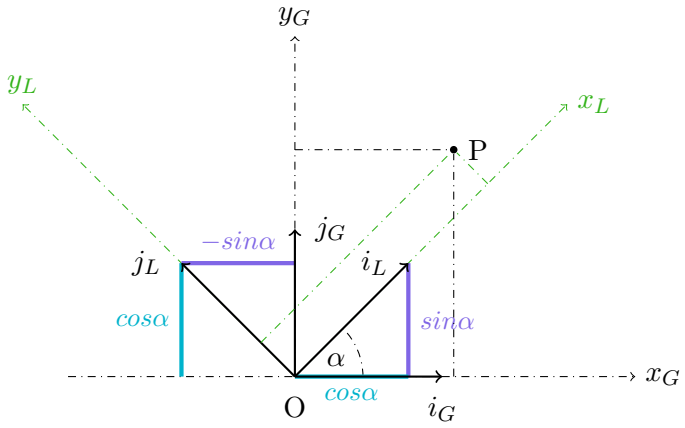
$$\begin{aligned} \vec{i}_L &= \cos(\alpha)\vec{i}_G + \sin(\alpha)\vec{j}_G \\ \vec{j}_L &= -\sin(\alpha)\vec{i}_G + \cos(\alpha)\vec{j}_G \end{aligned} \tag{2.13}$$

Dosadením $\vec{i}_L$ a $\vec{j}_L$ z (2.11) do $\vec{OP}^L$ v (2.10) a upravením dostávame:

$$\vec{OP}_L = (\vec{OP}_x^L \cos(\alpha) - \vec{OP}_y^L \sin(\alpha))\vec{i}_G + (\vec{OP}_x^L \sin(\alpha) + \vec{OP}_y^L \cos(\alpha))\vec{j}_G \tag{2.14}$$

Súradnicové systémy L a G zdieľajú rovnaký počiatočný bod a teda $\vec{OP}^G$ a $\vec{OP}^L$ je ten istý vektor. Z rovnice 2.14 a z rovnice pre $\vec{OP}^G$ v 2.12 potom vyplýva, že koeficienty pre $\vec{i}_G$ a $\vec{j}_G$, musia byť zhodné. Dostávame tak rovnice pre rotáciu vektora:

$$\begin{aligned} \vec{OP}_x^G &= \vec{OP}_x^L \cos(\alpha) - \vec{OP}_y^L \sin(\alpha) \\ \vec{OP}_y^G &= \vec{OP}_x^L \sin(\alpha) + \vec{OP}_y^L \cos(\alpha) \end{aligned} \tag{2.15}$$



Obr. 2.5: Súradnice bodu P v dvoch súradnicových systémoch.

Škálovanie [7]

Škálovanie, podobne ako rotácia, je **lineárna** transformácia. Majme bod $A \in \mathbb{R}^2$ a škálovanie definované nenulovým vektorom $\vec{s} \in \mathbb{R}^2$. Potom nové súradnice bodu A' budú:

$$\begin{aligned} A'_x &= \vec{s}_x A_x \\ A'_y &= \vec{s}_y A_y \end{aligned} \tag{2.16}$$

Afinné transformácie

Zložením posunutia a lineárnych transformácii dostávame afinné transformácie v $\mathbb{R}^2$:

$$\begin{aligned} x' &= c_{11}x + c_{12}y + c_{10} \\ y' &= c_{21}x + c_{22}y + c_{20}, \end{aligned} \quad kde \quad \begin{vmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{vmatrix} \neq 0 \tag{2.17}$$

Čísla $c_{ij} \in \mathbb{R}$ jednoznačne popisujú túto afinnú transformáciu [7].

2.3 Matice

Zovšeobecnením pojmu vektor v algebraickom kontexte dostaneme dvojrozmerné pole čísel. Matica ako prvok maticového priestoru $\mathbb{R}^{r \times s}$; $r, s \in \mathbb{N}$ sa bude skladať z r riadkov a s stĺpcov. Vektor môžeme zapísať aj ako maticu, kde $r = 1 \vee s = 1$. Vznikne nám tak vektor riadkový alebo stĺpcový. Podľa spôsobu zápisu rozlišujeme matice na Row-major a Column-major. Budeme sa držať zápisu **Column-major** a body/vektory budeme zapisovať do stĺpca. Pri aplikovaní transformácie budeme násobiť transformovaný bod/vektor transformačnou maticou zľava [5].

Násobenie matíc

Majme maticu $A \in \mathbb{R}^{r \times p}$, maticu $B \in \mathbb{R}^{p \times s}$. Nech $a_{ik} \in A$ a $b_{kj} \in B$ značia jednotlivé prvky matice. Matice A, B je možné násobiť pokiaľ počet stĺpcov matice A je rovný počtu riadkov matice B. Potom súčinom matíc A, B je matica $C = (c_{ij})$ pre ktorú platí:

$$c_{ij} := \sum_{k=1}^{p} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + ... + a_{ip} b_{pj}. \tag{2.18}$$

Prvok ležiaci v i-tom riadku a v j-tom stĺpci získame tak, že prechádzame i-ty riadok v matici A a jeho prvky postupne násobíme prvkami ležiacimi v j-tom stĺpci matice B a súčiny sčítame. Násobenie matíc **nie je** komutatívne. Keďže sa držíme kontextu transformácií v dvojrozmernom priestore uvedieme si príklad násobenia matíc $A, B \in \mathbb{R}^{2 \times 2}$ [5, 8].

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1a + 2c & 1b + 2d \\ 3a + 4c & 3b + 4d \end{pmatrix} \tag{2.19}$$

Jednotková matica

Špeciálnym prípadom transformačnej matice je jednotková matica, ktorá vyjadruje transformáciu, ktorá mapuje každý vektor a bod na seba samého. Nech $A \in \mathbb{R}^{2 \times 2}$, $B \in \mathbb{R}^{3 \times 3}$ [1]:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.20}$$

Rotačná matica

Transformáciu rotácie (Rov. 2.15), môžeme kompaktne zapísať použitím matice $M \in \mathbb{R}^{2 \times 2}$.

$$M(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{2.21}$$

Keďže táto práca operuje s maticami typu Column-Major stĺpce matice predstavujú bázové vektory. Platí tak, že pokiaľ vynásobíme vektor $\vec{u} \in \mathbb{R}^{2 \times 1}$ maticou M zľava, dostaneme zrotovaný vektor $\vec{u'}$ [1, 5, 9].

$$\begin{pmatrix} \vec{u'_x} \\ \vec{u'_y} \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} \vec{u_x} \\ \vec{u_y} \end{pmatrix} \tag{2.22}$$

Škálovacia matica

Transformáciu škálovania (Rov. 2.16), môžeme kompaktne zapísať použitím matice $S \in \mathbb{R}^{2 \times 2}$ [4, 5].

$$S(\vec{s}) = \begin{pmatrix} \vec{s}_x & 0 \\ 0 & \vec{s}_y \end{pmatrix} \tag{2.23}$$

Platí tak, že pokiaľ vynásobíme vektor $\vec{u} \in \mathbb{R}^{2 \times 1}$ maticou S zľava, dostaneme vektor $\vec{u'}$:

$$\begin{pmatrix} \vec{u'_x} \\ \vec{u'_y} \end{pmatrix} = \begin{pmatrix} \vec{s}_x & 0 \\ 0 & \vec{s}_y \end{pmatrix} \begin{pmatrix} \vec{u_x} \\ \vec{u_y} \end{pmatrix} \tag{2.24}$$

Translačná matica

Z definície afinných transformácií 2.17 a z definície pre násobenie matíc 2.18 vidíme, že zakódovať transláciu v dvojrozmernom priestore do matice $T \in \mathbb{R}^{2\times 2}$ nie je možné. Riešením je využiť opäť raz tretí rozmer a rozšíriť prvky dvojrozmerného vektora o prvok $z = 1$ a maticu T rozšíriť navyše o jeden riadok a stĺpec aby sme dostali maticu $T \in R^{3\times 3}$. V trojzmernom priestore tak môžeme využiť lineárnu transformáciu skosenia (ang. shear), ktorej výsledný transformovaný objekt sa nám spätne premietne do dvojrozmerného priestoru ako posunutie [4, 5].

Naša výsledná translačná matica T so zakódovaným vektorom posunutia $\vec{p}$ z rovnice 2.11 a transformovaný bod A zapísaný ako stĺpcový vektor, budú vyzerať nasledovne:

$$T(\vec{p}) = \begin{pmatrix} 1 & 0 & \vec{p}_x \\ 0 & 1 & \vec{p}_y \\ 0 & 0 & 1 \end{pmatrix}, \quad A = \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix} \tag{2.25}$$

Vo výsledku tak dostávame nové súradnice bodu A':

$$A' = \begin{pmatrix} 1 & 0 & \vec{p}_x \\ 0 & 1 & \vec{p}_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix} = \begin{pmatrix} A_x + \vec{p}_x \\ A_y + \vec{p}_y \\ 1 \end{pmatrix} \tag{2.26}$$

Skladanie transformácii

Máme dve možnosti ako reprezentovať vyššie uvedené transformácie pre dvojrozmerný priestor. Majme bod A a vektor $\vec{u}$; $A, \vec{u} \in \mathbb{R}^{2\times 1}$. Ďalej nech $\vec{p} \in \mathbb{R}^{2\times 1}$ je vektor posunutia a matica $T_{rs} \in \mathbb{R}^{2\times 2}$ vyjadruje zloženú transformáciu rotácie a škálovania. Ak chceme byť explicitný môžeme transformácie zakódovať do matice $T \in \mathbb{R}^{3\times 3}$ a dočasne previesť transformované body a vektory do tretieho rozmeru [5, 9]:

$$A^{3\times 1} = \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix}, \quad \vec{u}^{3\times 1} = \begin{pmatrix} u_x \\ u_y \\ 0 \end{pmatrix}, \quad T_{rs} = \begin{pmatrix} \vec{s_x}\cos\alpha & -\sin\alpha \\ \sin\alpha & \vec{s_y}\cos\alpha \end{pmatrix} \tag{2.27}$$

$$T = \begin{pmatrix} \boldsymbol{T_{rs}} & \begin{matrix} \vec{p}_x \\ \vec{p}_y \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{pmatrix} = \begin{pmatrix} \vec{s_x}\cos\alpha & -\sin\alpha & \vec{p}_x \\ \sin\alpha & \vec{s_y}\cos\alpha & \vec{p}_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.28}$$

Daný bod/vektor vynásobíme transformačnou maticou T zľava. Dostaneme transformované súradnice x' a y' pre ktoré platí 2.17.

Kompaktnejšia reprezentácia dvojrozmerných transformácii je $T_k \in \mathbb{R}^{2\times 3}$. Pri transformovaní vektorov a bodov vychádzame opäť z rovnice o afinných transformáciách 2.17. Poradie transformácií bude vždy škálovanie, rotácia a potom translácia. Keďže rotáciu a škálovanie máme zakódovanú v T_{rs} môžeme túto transformáciu aplikovať ako násobenie matice a vektora. Následne využijeme tretí stĺpec matice T_k a aplikujeme transláciu samostatne.

$$T_k = \begin{pmatrix} \boldsymbol{T_{rs}} & \begin{matrix} \vec{p}_x \\ \vec{p}_y \end{matrix} \end{pmatrix} = \begin{pmatrix} \vec{s_x}\cos\alpha & -\sin\alpha & \vec{p}_x \\ \sin\alpha & \vec{s_y}\cos\alpha & \vec{p}_y \end{pmatrix} \tag{2.29}$$

Kapitola 3

Dynamika tuhého telesa

Kapitola pojednáva o fyzike tuhých telies so zameraním na dvojrozmerný priestor. V istých prípadoch budeme môcť zjednodušiť teleso na hmotný bod. Pre prípady, keď rozmery objektov nemôžu byť zanedbané budeme na teleso nahliadať ako na sústavu veľkého počtu hmotných bodov, ktorých vzájomná poloha sa pri pohybe ako celku nemení. Kinematické veličiny opisujúce pohyb si predstavíme aj v rámci klasickej newtonovskej mechaniky a poznatky využijeme pri popise posuvného a pousvno-otáčavého pohybu telesa.Obmedzenie geometrického tvaru telesa na konvexné polygóny a kruhy nám poslúži k zjednodušeniu simulácie. Pre túto prácu neuvažujeme Hamiltonovskú mechaniku.

3.1 Tuhé teleso

Teleso je reálny objekt, ktorého hmota je vo forme látky v tuhom skupenstve. Teleso tvorí celok, je súdržné, nepriepustné, vymedziteľné údajmi a deformovateľné. Pokiaľ deformácia je z hľadiska riešeného problému nepodstatná, hovoríme o **tuhom telese**. Systém tuhých telies berie do úvahy tvar a hmotnosť telesa. Vnútorné sily pôsobiace v telese sú zanedbané. Teleso tak počas simulácie nemení svoj tvar. V homogénnom gravitačnom poli alebo zotrvačnom poli môžeme všetky gravitačné a zotrvačné sily pôsobiace na tuhé teleso sústrediť do jediného bodu, tzv. hmotného stredu. Pri translačnom pohybe môžeme nazerať na tuhé teleso ako na hmotný bod [10, 11].

3.2 Hmotný stred telesa

Hmotný stred telesa je unikátny bod pre distribúciu hmotnosti telesa v priestore. V hmotnom strede telesa je vážený priemer pozícií hmotných bodov rovný nule. Zjednodušene povedané jedná sa o bod, ktorý sa pohybuje ako keby v ňom bola sústredená celá hmotnosť sústavy a pôsobili v ňom všetky sily pôsobiace na sústavu. Pokiaľ sa teleso nachádza v tiažovom poli zeme, môžeme hovoriť o hmotnom strede ako o ťažisku. Hmotný stred sa môže nachádzať aj mimo telesa [11].

Nájdenie hmotného stredu

Zjedodušený príklad pre hmotný stred si môžeme uviesť na jednorozmernej sústave v ktorej sa nachádzaju dve častice (Obr. 3.1). Keďže máme len x-ovú súradnicu, hmotný stred

(COM) x_{com} častíc bude rovný:

$$x_{com} = \frac{m_1 x_1 + m_2 x_2}{m_1 + m_2} \tag{3.1}$$

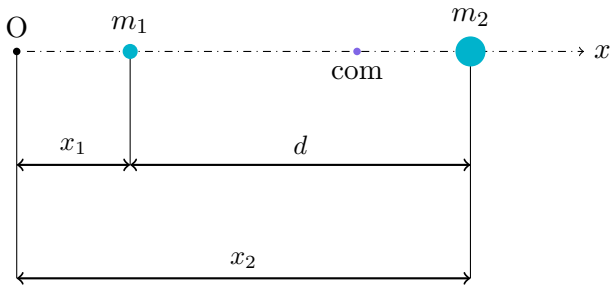
Zovšeobecniť môžeme pre sústavu s n časticami ako suma:

$$x_{com} = \frac{1}{M} \sum_{i=1}^{n} m_i x_i \tag{3.2}$$

Pokiaľ sú častice rozdistribuované vo viacrozmernom priestore, analogicky k rovnici 3.1 a 3.2 nájdeme hodnoty súradníc pre y, z atď.

Pre praktické účely potrebujeme nájsť hmotný stred pre tuhé telesá. Tuhé teleso z pravidla obsahuje veľké množstvo častíc a tak musíme nahliadať na výpočet hmotného stredu ako na spojité rozloženie hmotnosti telesa. Suma z rovnice 3.2 sa zovšeobecní na integrál:

$$x_{com} = \frac{1}{M} \int_S x dm \tag{3.3}$$

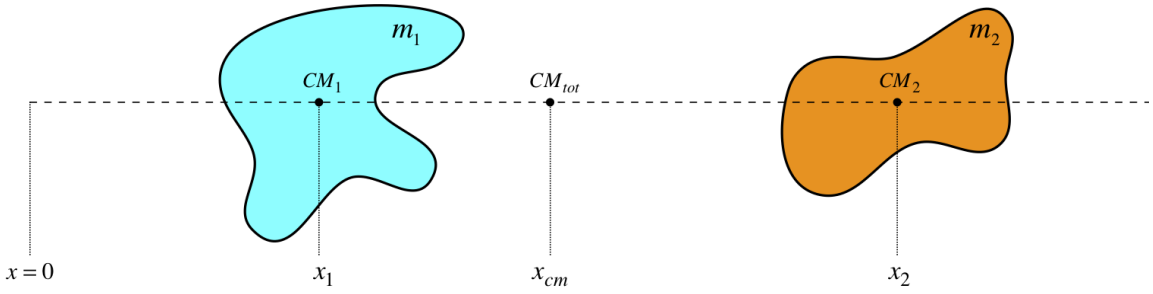


Obr. 3.1: Hmotný stred sústavy dvoch hmotných bodov

Vyhodnotenie integrálu pre väčšinu objektov by v skutočnosti bolo náročné, keďže hustota telies je často nerovnomerná. Budeme predpokladať uniformnú hustotu a teda pre každú časť telesa bude hmotnosť na jednotku objemu rovnaká. Telesá majú svoju geometrickú symetriu, napr. stredovú. Poloha ťažiska symetrického homogénneho telesa úzko súvisí s jeho symetriou. Ak je teleso stredovo symetrické, splýva jeho ťažisko so stredom symetrie. V prípade rovinných geometrických útvarov je tak možné predpokladať, že hmotný stred bude totožný so stredom geometrickým a úloha nájdenia hmotného stredu sa nám zjednoduší na nájdenie stredu geometrického. V prípade, že poznáme hmotný stred pre každé teleso v systéme, môžeme nájsť hmotný stred systému vypočítaním váženého priemeru hmotných stredov jednotlivých telies. V podstate opäť nazeráme na telesá ako na hmotné body/častice [10, 11].

3.3 Geometria telesa

Reálne telesá majú obvykle zložitý tvar. Náročnosť výpočtu simulácie sa zvyšuje s počtom vrcholov, ktoré ohraničujú teleso. Keďže počítačové hry vyžadujú výpočet v reálnom čase, teda čo najrýchlejší, teleso je potrebné dostatočne zjednodušiť. Vďaka tomu má výsledný objekt menej vrcholov. Prílišné zjedodušenie môže mať za následok, že výsledok nebude pôsobiť vierohodne. Príkladom v hrách je zjedodušenie postavy človeka na sústavu kolíznych



Obr. 3.2: Hmotný stred sústavy dvoch telies [12].

kvádrov a sfér (Obr.3.3). V našom prípade (2D) sa budeme sústrediť na kruhy a polygóny z ktorých následne vyskladáme zložitejšie kolízne geometrie.

Obr. 3.3: Zjednodušenie kolíznej geometrie postavy na sústavu sfér [13].

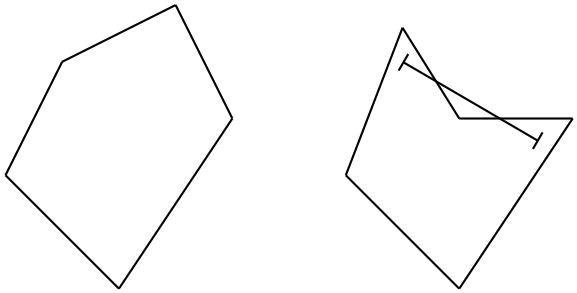
Kruh

Hranicu kruhu vymedzuje kružnica, ktorá predstavuje množinu bodov s rovnakou vzdialenosťou od určeného stredu, ktorej sa hovorí polomer. Keďže sa pohybujeme v rovine a naše telesá majú uniformnú hustotu automaticky je tak hmotný stred kruhu totožný s jeho geometrickým stredom. Kruh ako kolízny tvar nám poskytuje jednoduchú implementáciu detekcie kolízie telies. Viac v kapitole o kolíziách.

Polygón

Polygón, alebo mnohouholník, je uzatvorený geometrický tvar s n stranami, definovaný usporiadanou množinou vrcholov v poradí, že dva po sebe nasledujúce vrcholy a posledný vrchol s prvým tvoria hrany mnohouholníka. Dva vrcholy spolu susedia pokiaľ zdieľajú spoločnú hranu. Polygóny môžeme rozdeliť na konvexné a konkávne. Polygón je konvexný ak úsečka spájajúca ľubovolné dva body vo vnútri polygónu, leží celá vo vnútri polygónu. Konvexný polygón má všetky vnútorné uhly menšie ako 180°. Ak aspoň jeden uhol je väčší ako 180° hovoríme o konkávnom polygóne. Trojuholník je jediný polygón pri ktorom

môžeme garantovať, že je vždy konvexný. Väčšina testov na prienik dvoch telies je rýchlejšia pokiaľ predpokladáme tvar telesa zložený práve z konvexných polygónov [14].

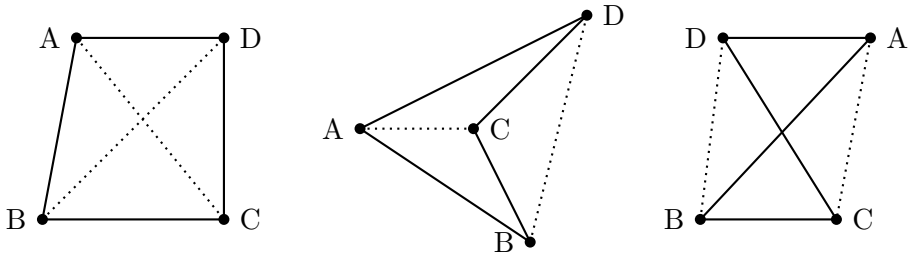


Obr. 3.4: Konvexný a konkávny polygón

Konvexnosť polygónu

Test na konvexnosť polygónu je vhodný aby sme sa presvedčili, že zoznam vrcholov patriacih polygónu skutočne tvoria konvexný polygón. V počítačových hrách okrem trojuholníka je najčastejšie používaným polygónom štvoruholník, ktorého test na konvexnosť je o niečo jednoduhší ako test na konvexnosť pre všeobecný n-uholník. Štvoruholník je konvexný práve vtedy, keď jeho dve diagonály ležia vo vnútri polygónu. Prípadne tiež platí, že ak sa diagonály pretínaju, štvoruholník je konvexný. Dve diagonály sa pretínaju práve vtedy ak vrcholy A a C ležia na opačných stranách diagonály BD rovnako ako body B a D ležia na opačných stranách diagonály AC (Obr. 3.5). Dá sa dokázať, že tento test je ekvivalentný otázke, či vrcholy tvoriace trojuholník BDA nasledujú v opačnom smere ako vrcholy tvoriace trojuholník BDC. Respektíve, že ak postupnosť vrcholov trojuholníka ACD je v protismere hodinových ručičiek a postupnosť vrcholov trojuholníka ACB je v smere hodinových ručičiek, môžeme prehlásiť, že štvoruholník ABCD je konvexný. Platí, že štvoruholník ABCD je konvexný ak [14]:

$$(\vec{BD} \times \vec{CA}) \cdot (\vec{BD} \times \vec{BC}) < 0 \wedge (\vec{AC} \times \vec{AD}) \cdot (\vec{AC} \times \vec{AB}) < 0 \tag{3.4}$$

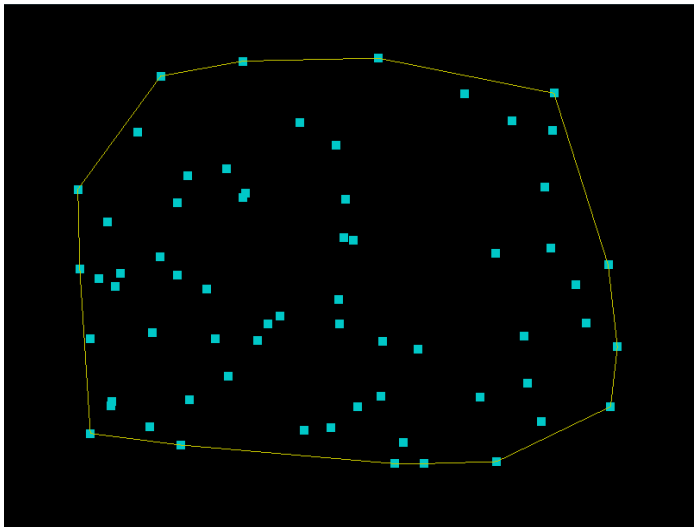


Obr. 3.5: Konvexný a konkávny polygón

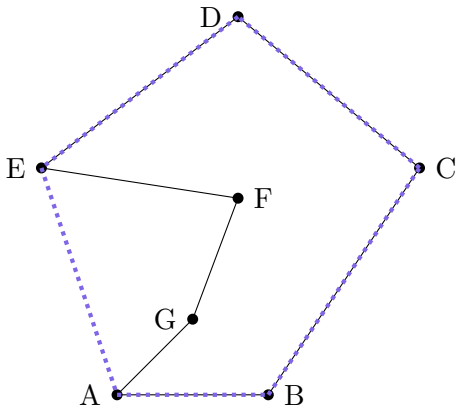
Všeobecný polygón prechádzame hranu po hrane a sledujeme polohu ostatných vrcholov (okrem vrcholov aktuálnej hrany). Ak vrcholy ležia na jednej strane hrany, posunieme sa na ďalšiu hranu. Ak takto prejdeme všetky hrany, polygón je konvexný.

Konvexná obálka

V prípade, že na vstupe dostaneme množinu náhodných bodov, môžeme tieto body obaliť do konvexnej obálky (Obr. 3.6). Okrem toho, že takto vieme vytvoriť ľubovolný konvexný polygón z náhodnej množiny bodov, môžeme vďaka tomu vytvárať tesné kolízne geometrie, ktoré budú detailnejšie ako napr. AABB (viac v kapitole o kolíziách). Pomocou konvexnej obálky môžeme zjednodušiť tvar konkávnych telies (Obr. 3.7). V prípade kolízií tak môžeme stanoviť približnú kolíziu telies a oddialiť výpočet testu kolízie konkávnych tvarov. Existuje množstvo algoritmov s rôznou časovou náročnosťou (Tab. 3.1). Vysvetlíme si princíp algoritmu balenia darčeka (z ang. "Gift Wrapping Algorithm"), ktorý je využitý aj v samotnej knižnici [15, 16, 17].



Obr. 3.6: Konvexná obálka obalujúca množinu bodov.



Obr. 3.7: Konvexný polygón ABCDE obaľujúci konkávny polygón ABCDEFG.

Gift wrapping algorithm

Tiež Jarvis's Algorithm si môžeme predstaviť ako proces obalovania darčeka baliacim papierom. Poradie bodov tvoriacich výslednú obálku sa môže vo výsledku líšiť a definovať tak

Názov	Komplexnosť	
	Priemerná	Najhoršia
Gift Wrapping Algorithm	$O(nh)$	$O(nh)$
Graham's scan	$O(n \log(n))$	$O(n \log(n))$
Quickhull	$O(n \log(n))$	$O(n^2)$
Chan's Algorithm	$O(n \log(n))$	$O(n \log(n))$

Tabuľka 3.1: Porovnanie časovej komplexnosti algoritmov pre nájdenie konvexnej obálky [16].

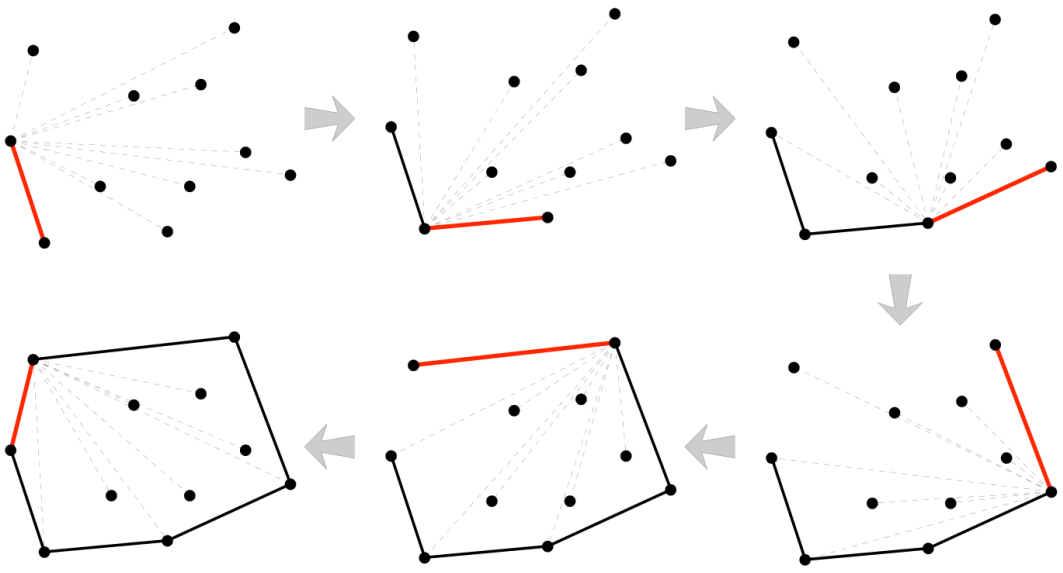
konvexný mnohouholník v smere alebo protismere hodinových ručičiek. Budeme sa držať prípadu, kedy chceme aby výsledný polygón bol definovaný množinou bodov, ktorej poradie je v protismere hodinových ručičiek:

1. Nájdeme bod s najväčšou/najmenšou x-ovou alebo y-ovou súradnicou. Je garantované, že tento bod S sa bude nachádzať vo výslednej obálke. Určíme bod P ako bod naposledny pridaný do konvexnej obálky.

2. Vyberieme ľubovolný bod Q taký, že $Q \neq P$ a prechádzame ostatné body $R_i; R_i \neq Q$. Ak je niektorý bod R_i napravo od vektora $\vec{PQ}$ položíme bod Q $= R_i$.

3. Ak je $Q = S$ skončíme, inak pridáme Q do konvexnej obálky a vraciame sa späť ku kroku 2.

Platí, že pokiaľ bod Q má byť v konvexnej obálke, body PQR budú v protismere hodinových ručičiek. Komplexnosť je $O(nh)$, kde n je celkový počet bodov a h je počet bodov tvoriacich výslednú obálku. Pri zisťovaní vzájomnej polohy vektorov, využijeme vektorový súčin pre 2D (2.10) [18].



Obr. 3.8: Postup vytvárania konvexnej obálky krok po kroku [18].

Geometrický stred

Geometrický stred telesa je bod, do ktorého je možné umiestniť pôsobisko sily rovinného telesa. Geometrický stred nezávisí na homogenite telesa, teda na rozložení hmoty v telese. Ak ide o rovinný geometrický útvar, leží ťažisko a geometrický stred v rovnakom bode. Geometrický stred kruhu záleží jedine od pozície kruhu, keďže pozícia kruhu je definovaná súradnicami jeho stredu [19].
Geometrický stred trojuholníka ABC:

$$\begin{aligned} S_x &= (A_x + B_x + C_x)/3 \\ S_y &= (A_y + B_y + C_y)/3 \end{aligned} \tag{3.5}$$

Geometrický stred štvoruholníka definovaného bodmi ABCD:

$$\begin{aligned} S_x &= (B_x + A_x)/2 \\ S_y &= (C_y + B_y)/2 \end{aligned} \tag{3.6}$$

Geometrický stred konvexného polygónu:

Pokiaľ vieme, že polygón je pravidelný potom jeho geometrický stred bude podobne ako u trojuholníka váženým priemerom jeho súradníc. Avšak pri nepravidelných polygónoch je postup o niečo zložitejší. Nech P je konvexný uzavretý polygón definovaný n vrcholmi $V_1 = (x_1, y_1), V_2 = (x_2, y_2), ..., V_n = (x_n, y_n)$ v poradí určenom v protismere hodinových ručičiek. Polygón rozdelíme triangluláciou na trojuholníky a vypočítame ich stredy C_i a určíme obsahy trojuholníkov $S_i; i = 1, 2, ..., n-2$. Celkový obsah polygónu S_p je potom [20]

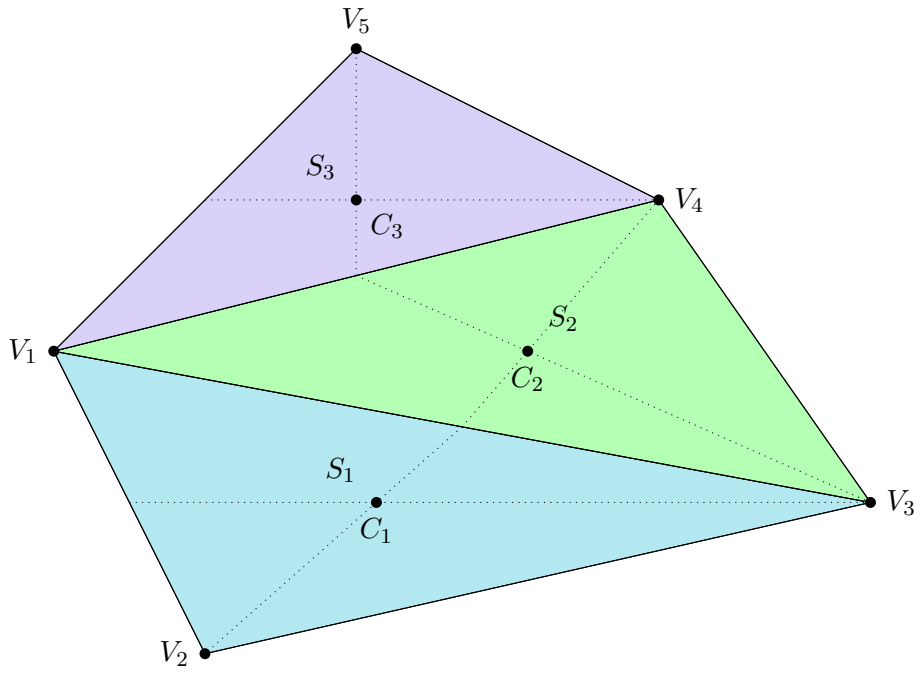
$$S_p = \sum_{i=1}^{n-2} S_i \tag{3.7}$$

Výsledný stred polygónu C_p bude:

$$C_p = \frac{1}{S_p} \sum_{i=1}^{n-2} S_i C_i \tag{3.8}$$

3.4 Dynamika telesa

Ak vieme nájsť hmotný stred telesa môžeme pomocou aplikovania síl uviesť teleso do pohybu. Teleso má v dvojrozmernom priestore tri stupne voľnosti. Pohyb po súradniciach x a y (translačný pohyb) a rotácia (napr. okolo hmotného stredu). Aplikovaním sily, ktorej vektor prechádza hmotným stredom telesa meníme jeho zrýchlenie a tak nepriamo meníme jeho pozíciu. Pokiaľ aplikujeme silu, ktorej vektor prechádza mimo hmotného stredu, môžeme navyše uviesť teleso do otáčavého pohybu a teleso tak rotovať. Kinematické veličiny ako pozícia, rýchlosť a zrýchlenie sú úzko prepojené. Pozícia telesa nám vyjadruje polohu telesa v danom súradnicovom systéme. Pokiaľ sa pozeráme na zmenu pozície telesa v čase dostávame rýchlosť a ak chceme zájsť ďalej a zistiť ako sa v čase mení rýchlosť dostávame veličinu zrýchlenia. Vďaka tomu vieme na základe znalosti o zrýchlení telesa získať integráciou jeho rýchlosť a ďalším integrovaním jeho pozíciu. Ostáva nám tak určiť správne zrýchlenie telesa a práve tu do hry vstupuje dynamika. Detailnejšie si uvedieme dôležité veličiny z dynamiky, ktoré nám napovedia ako sa bude výsledná simulácia správať [10, 11].



Obr. 3.9: Triangulácia polygónu, kde C_1, C_2, C_3 značia geometrické stredy jednotlivých trojuholníkov a S_1, S_2, S_3 značia ich obsahy [20].

Sila

Sila je vektorová veličina vyjadrujúca mieru interakcie telies alebo polí. Sila je základným pojmom pri štúdiu mechaniky tuhých telies. Uvádza teleso do pohybu, mení jeho smer prípadne zastaví pohyb telesa. Sila môže teleso zrýchliť alebo ho deformovať. Keďže pojednávame o tuhých telesách deformácie vypustíme. Poznáme sily vnútorné a vonkajšie.

Vnútornými silami na seba pôsobia prvky objektu alebo systému, ktorého pohybový stav je pozorovaný. Vnútorné sily držia teleso pohromade aj keď sú jednotlivé štruktúry telesa pod napätím a tlakom.

Vonkajšie sily pôsobia na teleso v dôsledku interakcie s jeho okolím. Môžeme ich ďalej rozdeliť na kontaktné a nekontaktné. V prípade kolízií telies budeme pracovať s kontaktnými silami a v prípade aplikovania ťiažového zrýchlenia budeme pracovať s nekontaktnou silou. Medzi kontaktné sily patrí napríklad normálová sila a trecia sila.

Pri aplikovaní sily platí princíp superpozície. Ak na teleso pôsobí v danom čase viacero síl, tieto sily môžeme skladať a výsledné zrýchlenie bude rovnaké. Matematicky sa jedná o súčet vektorov [11, 21].

Trecia sila

Trecia sila je sila pôsobiaca proti smeru pohybu. Trenie vzniká ako výsledok interakcie medzi molekulami povrchov v kontakte. Pokiaľ sa jednotlivé povrchy vzájomne nepohybujú hovoríme o statickom trení. Naopak ak sa povrchy vzájomne pohybujú hovoríme o trení dynamickom. Matematicky môžeme vyjadriť treciu silu vzťahom $F_t = \mu F_n$, kde F_t je statická alebo dynamická trecia sila a μ je súčiniteľ statického alebo dynamického trenia, t.j číslo vyjadrujúce vplyv jednotlivých materiálov na treciu silu a F_n je normálová kontaktná sila. Väčšinou je koeficient statického trenia väčší ako koeficient dynamického trenia [11].

Impulz sily a hybnosť

Z druhého Newtonovho zákona nepriamo vyplýva vzťah medzi dobou trvania pôsobiacej sily a zmenou pohybového stavu telesa. Tento vzťah môžeme popísať pomocou veličiny hybnosť a veličiny impulz sily. **Hybnosť** ($\vec{p}$) nám umožňuje vyjadriť mieru pohybu a zotrvačnosti telesa jednou hodnotou. Keďže tuhé telesá majú konštantnú hmotnosť, platí, že ak sa nemení ich rýchlosť, hybnosť ostáva konštantná. Celková hybnosť izolovanej sústavy telies sa vzájomným silovým pôsobením telies nemení. **Impulz sily** ($\vec{I}$) je výsledkom súčinu pôsobiacej výslednej sily $\vec{F}(t)$ a doby trvania pôsobenia tejto sily Δt. Impulz sily je príčinou zmeny pohybového stavu a teda zmeny hybnosti [11, 21]. Matematicky môžeme túto skutočnosť vyjadriť ako:

$$\vec{I} = \int_s^k \vec{F} dt = m(\vec{v_k} - \vec{v_s}) = \vec{p_k} - \vec{p_s} \tag{3.9}$$

,kde m je hmotnosť pohybujúceho sa telesa, v_k je rýchlosť na konci uvažovaného časového intervalu pôsobenia sily a v_s je rýchlosť na začiatku uvažovaného časového intervalu. Definícia je zovšeobecnená na prípad, kedy sa sila v danom čase mení. My budeme predpokladať konštantnú silu a zjednodušíme si integrál na rovnicu:

$$\vec{I} = \vec{F}t \tag{3.10}$$

Newtonove pohybové zákony

Zákon zotrvačnosti

Teleso zotrváva v pokoji alebo v rovnomernom priamočiarom pohybe pokiaľ nie je nútené tento pohybový stav zmeniť vplyvom vonkajších síl, ktoré na neho pôsobia. Pokiaľ teda sily pôsobiace na teleso sú vo výsledku nulové, teleso je v pokoji, alebo sa pohybuje stále rovnakou rýchlosťou a priamočiaro [21].

Zákon sily

Newton pôvodne definoval veľkosť sily rovnajúcu sa zmene hybnosti telesa v čase. Pričom hybnosť telesa závisí priamo úmerne od jeho hmotnosti a rýchlosti.

$$\vec{F} = \frac{d\vec{p}}{dt} = \frac{d(m\vec{v})}{dt} \tag{3.11}$$

V tomto momente nevieme presne určiť ako sa zmenila hmotnosť telesa v čase a ako sa zmenila rýchlosť. Zjednodušením pre nás bude predpoklad, že simulované telesá nemenia v čase svoju hmotnosť a tak sa nám pôvodná rovnica zjednoduší na viac známu:

$$\vec{F} = m\frac{d\vec{v}}{dt} = m\vec{a} \tag{3.12}$$

Ak by sme chceli simulovať let vesmírnej rakety, ktorej sa v čase míňa palivo a mení tak v čase svoju hmotnosť, držali by sme sa pôvodnej rovnice 3.11. My sa budeme držať rovnice 3.12, ktorá nám hovorí, že zmena zrýchlenia telesa je priamo úmerna pôsobiacej sile a nepriamo úmerná hmotnosti telesa [4].

Zákon akcie a reakcie

Ak na jedno teleso pôsobí iné teleso, tak pôsobia na seba rovnakými silami, opačného smeru, pričom pôsobia v jednej priamke. Dôsledkom tretieho zákona je zdanlivý paradox, že sila, ktorou pôsobí osoba na Zem je rovnako veľká, ako sila, ktorou pôsobí Zem na osobu. Zákon nám pomáha vysvetliť ako sily pôsobia a na čo pôsobia. Nehovorí nám priamo, aký spôsobujú pohyb telies. Hovorí nám len, že sily vznikajú vo dvojiciach a každá z nich pôsobí na iné teleso. Preto sa sily akcie a reakcie vektorovo neskladajú [11].

Moment zotrvačnosti

S momentom zotrvačnosti sa stretávame pri štúdiu dynamiky rotačného pohybu, ktorý je analógiou 2. Newtonovho zákona pre translačný pohyb. Moment zotrvačnosti telesa je mierou vlastnosti telesa, ktorú by sme mohli nazvať zotrvačnosťou pri rotačnom pohybe. Závisí nielen od hmotnosti telesa, ale aj od jej rozloženia vzhľadom k osi rotácie.Kvantitatívne vyjadrenie miery zotrvačnosti rotujúceho telesa. Zatiaľ čo zotrvačnosť lineárne pohybujúcich sa telies je závislá len na jednej veličine (hmotnosti), zotrvačnosť rotujúcich telies je závislá na hmotnosti a vzdialenosti telesa od osi otáčania [11, 21].

Moment zotrvačnosti pre hmotný bod:

$$J = mr^2[kg.m^2] \tag{3.13}$$

V prípade sústavy hmotných bodov:

$$J = \sum_{i=1}^{n} m_i r_i^2 \tag{3.14}$$

Ak uvažujeme o reálnom telese, v ktorom je hmota rozložená spojite.

$$J = \int r^2 dm = \int_V \rho r^2 dV \tag{3.15}$$

Steinerova veta

Pri výpočte momentu zotrvačnosti vzhľadom na inú os ako máme vypočítané, je výhodné použiť Steinerovu vetu [22]:

$$J = J_t + md^2 \tag{3.16}$$

,ktorá hovorí:

> „Moment zotrvačnosti J tuhého telesa vzhľadom na ľubovolnú os sa rovná momentu zotrvačnosti J_t tohoto telesa vzhľadom na os paralelnú s danou osou a prechádzajúcu ťažiskom telesa T plus súčin hmotnosti telesa a štvorca vzdialenosti d medzi týmito osami.“

Kapitola 4

Kolízie

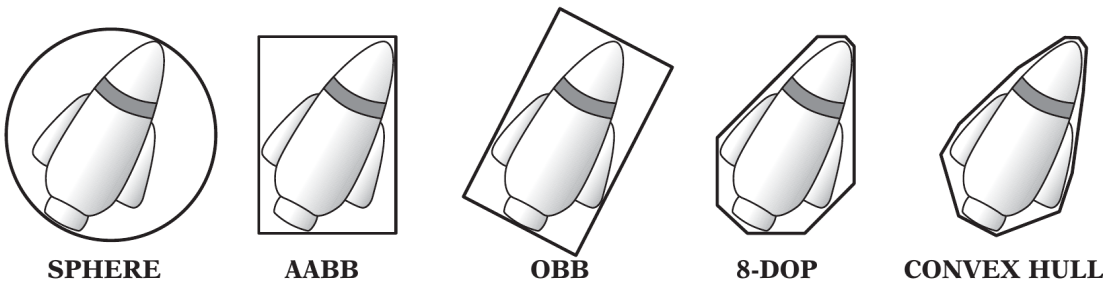
Pokiaľ chceme testovať kolíziu dvoch telies, musíme sa ubezpečiť, že operujeme v rovnakom súradnicovom systéme. Ak tomu tak nie je, použijeme transformácie 2.2. Dve telesá sa zrazili pokiaľ sa pokúsili v danom čase v spoločnom súradnicovom systéme zaujať rovnakú pozíciu. Budeme uvažovať kolízie telies, ktorých kolízne útvary sú konvexné polygóny a kruhy. Detekciu kolízií si rozdelíme na dve fázy. Hrubá (Broad) fáza zabezpečuje približný odhad, ktoré telesá môžu so sebou potencionálne kolidovať. Presná (Narrow) fáza potom rieši či potencionálne kolidujúce telesá naozaj kolidujú a jej výsledkom je tak informácia o kolízií dvoch telies a ich kolíznych bodov [14].

4.1 Hrubá fáza

Výsledkom hrubej fázy sú potencionálne kolízie, ktoré slúžia ako približný odhad dvojíc telies, ktoré sa zrážajú. Pri vytváraní telies predpočítame ich približné vymedzenia v priestore, tzv. bounding volume, ktoré usporiadame do hierarchií.

Bounding Volumes (BV)

Z angličtiny voľne preložené ako ohraničujúce miery. Jedná sa o jednoduché geometrické tvary zostrojené nad komplexnými modelmi telies (Obr. 4.1). Najčastejšie sa využívajú kružnice a štvoruholníky pre dvojrozmerný priestor. Pre trojrozmerný priestor sú to potom sféry a kvádre. Jedným z BV je napríklad aj spomínana konvexná obálka obalujúca jeden alebo viacero polygónov. Priamy test na kolíziu telies by bol výpočtovo náročný a namiesto



Obr. 4.1: Typy ohraničujúcich geometrií, tzv. Bounding Volume (BV) [14].

toho sa ako aproximácia využívajú BV. Pointou je oddialiť presnú fázu testovania kolízie dvoch telies a časovo náročnejší test tak vo väčšine prípadov úplne vynechať.

Axis-aligned Bounding Box (AABB)

Doslova preložené ako osovo zarovnaná ohraničujúca debna je najčastejšie používaným BV. Pre dvojrozmerný priestor sa jedná o štvoruholník a pre trojrozmerný priestor potom o kváder. Typické je zarovnanie stien AABB so súradnicovými osami. Najväčšou výhodou AABB je rýchly test na zistenie prekrytia s ostatnými AABB. Najvýhodnejší spôsob je reprezentácia pomocou stredu a rozpätia pozdĺž jednotlivých súradnicových os. Pokiaľ tuhé teleso mení svoju pozíciu len translačne, stačí nám aktualizovať stred AABB. Navyše pokiaľ je rozpätie AABB pozdĺž súradnicových os zhodné čo do veľkosti, môžeme AABB interpretovať aj ako kružnicu (Bounding Sphere). Test prieniku dvoch AABB je triviálny a nezávislý od reprezentácie. Dve AABB sa prelínajú práve vtedy ak sa prelínajú vo všetkých súradnicových osiach.

BV Hierarchie

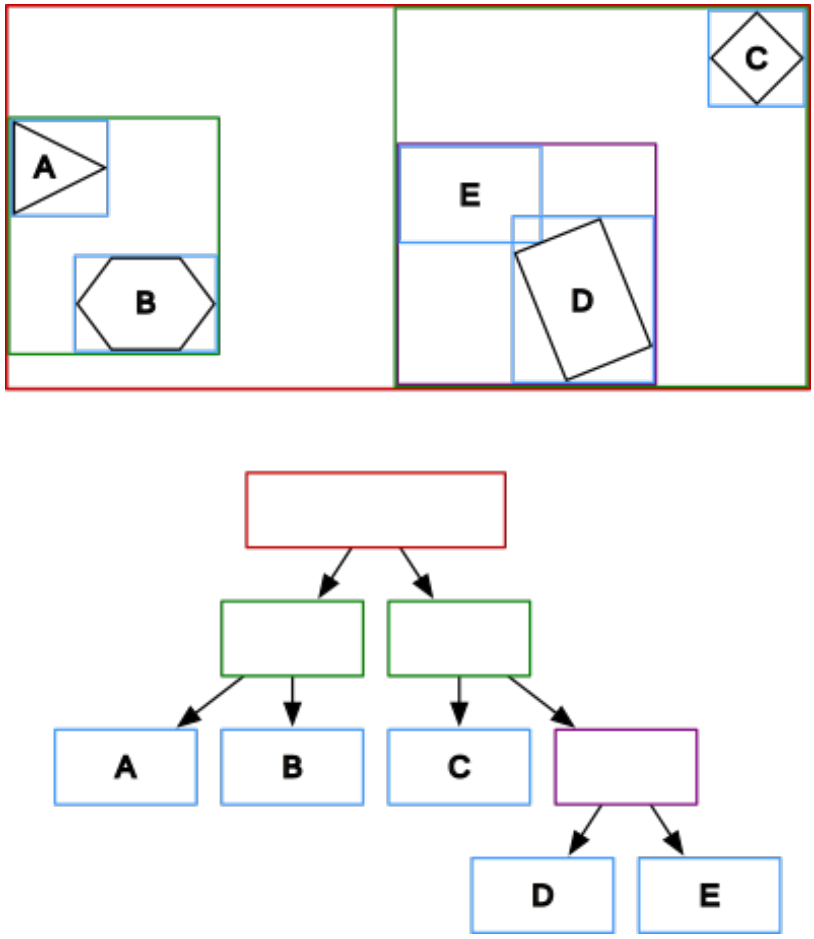
Simplifikácia telies pomocou BV môže viesť k zlepšeniu výkonu simulácie. Napriek tomu však stále prechádzame rovnaký počet párov telies. Časová komplexnosť ostáva rovnaká $O(n^2)$. Potrebujeme znížiť počet testovaných dvojíc BV hierarchickým usporiadaním do stromov (Obr. 4.2). Časová komplexnosť sa pri dobre zostrojenom strome zníži z kvadratickej na logaritmickú. Pôvodná množina BV je potom uložená v listoch stromu, ktorý reprezentuje danú hierarchiu. Uzly stromu sú zoskupené na malé množiny obalené väčšími BV. Tie sú následne zoskupené a uzatvorené v ďalších BV rekurzívne smerom ku koreňovému uzlu s jedným obrovským BV. S takouto hierarchiou počas kolízie nemusíme testovať potomkov uzlov stromu ak BV ich rodičovského uzlu nekoliduje s testovaným objektom.

Stupeň stromu

Dôležité je správne určiť stupeň rozvetvenia stromu. Čím vyšší stupeň stromu n, tým menšia výška stromu. Pri nízkej výške je čas priechodu stromu z koreňa k listu nízky. Avšak súčasne viac práce sa vynaloží pri návšteve každého uzla pri kontrole kolízie jeho potomkov. Naopak čím nižší stupeň stromu, tým dlhšia doba priechodu z koreňa k listomu, ale doba detekcie kolízie jeho potomkov je kratšia. N-árny strom s l listami má $\frac{l-1}{n-1}$ vnútorných uzlov a spolu $\frac{ld-1}{n-1}$ uzlov v celom strome. Čím vyšší stupeň stromu, tým menej vnútorných uzlov je potrebný pre zostrojenie stromu. Použitý stupeň stromu je ovplyvnený rôznymi faktormi, okrem iného aj architektúrou platformy na ktorej má simulácia bežať. Najčastejšie používané sú binárne stromy. Sú jednoduché ako na zostrojenie tak na prechádzanie uzlov.

Prechádzanie stromu

Nech H_1 a H_2 sú dve hierarchické reprezentácie BV objektov, ktorých potencionálne prieniky aktuálne skúmame. Hierarchie H_1 a H_2 sú v kolízii ak existuje aspoň jeden pár uzlov $n_1 \in H_1$ a $n_2 \in H_2$ taký, že ich BV sú v kolízii. Test na kolíziu sa vykonáva smerom zhoradolu a súčasne sa kontrolujú jednotlivé páry interných uzlov n_1 a n_2. Ako pomocnú štruktúru použijeme frontu Q do ktorej budeme dočasne umiestňovať všetky páry uzlov, ktoré ešte potrebuejeme skontrolovať. Frontu inicializujeme na pár obsahujúci koreňové uzly oboch



Obr. 4.2: Stromová hierarchia rozdeľuje telesá v priestore na podpriestory na základe ich BV. Hore je znázornené rozdelenie priestoru a dole príklad odpovedajúcej stromovej štruktúry.

hierarchií. V každej iterácii odstránime z fronty jeden pár uzlov a skontrolujeme prienik ich BV. Ak nie je kolízia zistená, riešime ďalší prvok vo fronte. Pokiaľ je kolízia zistená pridáme nové páry zostrojené z ich potomkov do fronty Q. V špeciálnom prípade, kedy oba uzly tvoriace pár sú listovými uzlami, pridáme objekty naviazané na tieto uzly do výsledného zoznamu potencionálne kolidujúcich objektov/telies. Alogirtmus pre zistenie intersekcie telies v rámci jedného stromu je obdobný. Prioritná fronta je však inicializovaná na koreňový uzol testovaný so samým sebou. Nasledovný algoritmus sumarizuje proces detekcie kolízie objektov v rámci jednej hierarchie [26]:

1. Získaj a odstráň prvý prvok z fronty Q. Nech $n_1 \in H_1$ a $n_2 \in H_1$ sú uzly odpovedajúce jednému páru.

2. Ak $n_1 \neq n_2$, potom skontroluj či BV jednotlivých uzlov sú v prieniku. Ak nie, odstráň pár a posuň sa na ďalší prvok vo fronte.

3. Skontroluj či oba uzly sú listami stromu. Ak sú listami, pridaj ich do výsledného zoznamu. Odstráň pár z prioritnej fronty a posuň sa na ďalší. Inak skontroluj či aspoň jeden uzol je list. Ak aspoň jeden je list povedzme n_1, potom:

   a) Pridaj pár $(n_1, (n_2)_l)$ do fronty, kde $(n_2)_l$ je potomok naľavo od uzla (n_2).

b) Pridaj pár $(n_1, (n_2)_r)$ do fronty, kde $(n_2)_r$ je potomok napravo od uzla (n_2).

4. V tomto momente, n_1 a n_2 sú oba vnútorné uzly. Pridaj nasledovné páry do fronty:

   - $((n_1)_l, (n_2)_l)$,
   - $((n_1)_l, (n_2)_r)$,
   - $((n_1)_r, (n_2)_r)$,
   - ak $n_1 \neq n_2$ pridaj pár $((n_1)_r, (n_2)_l)$.

Na konci dostávame zoznam uzlov, ktorých BV sú v kolízii. Štandardne sa pri vytváraní stromu na každý listový uzol naviaže príslušné teleso. Ako výsledok dostávame zoznam telies, ktoré sú potencionálne v kolízii a ktoré je potrebné ďalej skontrolovať v rámci detailnej fázy kolízie telies[26].

4.2 Detailná fáza

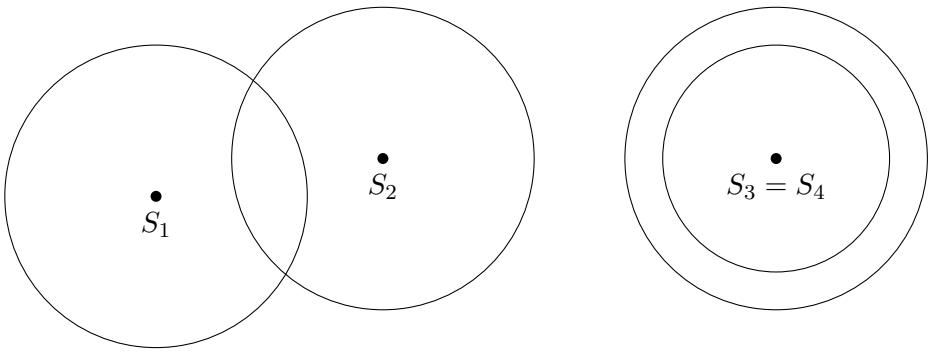
Detailná fáza dostáva na vstupe zoznam párov telies, ktoré sú potencionálne v kolízii. Vezmeme zjednodušený prípad, kedy každému telesu prislúcha jedna kolízna geometria. Na základe párov kolíznych geometrií telies potom musíme rozlíšiť, ktorý test vykonáme. Výsledkom detailnej fázy sú kolízne body, kolízna normála a hĺbka penetrácie telies. Na vyriešenie kolízie hľadáme vždy najmenšiu hĺbku penetrácie, ktorá nám zabráni v prieniku dvoch telies.

Kruh vs kruh

Najjednoduhší prípad pre detekciu kolízie telies. 4.3 Dva kruhy sú v kolízii práve vtedy, keď ich vzdialenosť (od stredu k stredu) je menšia ako súčet ich polomerov. Kolízna normála bude smerový vektor od stredu kruhu S_1 k stredu druhého kruhu S_2 a hĺbka prieniku d bude rovná:

$$d = r_1 + r_2 - |\vec{S_1 S_2}| \tag{4.1}$$

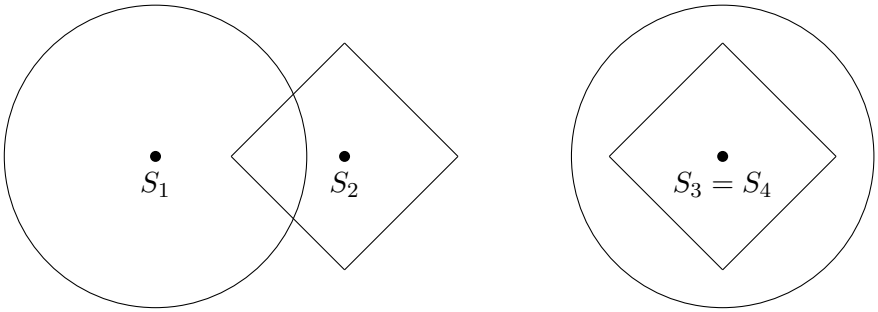
, kde r_1 a r_2 sú polomery kruhov. Špeciálny prípad nastáva vtedy ak sa kruhy nachádzajú na rovnakej pozícii, vtedy zvolíme ľubovolný smer kolíznej normály a hĺbku penetrácie nastavíme na polomer prvého kruhu [4].



Obr. 4.3: Vzájomná poloha dvoch kruhov v priestore. Štandardná situácia vľavo a špeciálny prípad vpravo.

Kruh vs polygón

Test prieniku kruhu a polygónu spočíva v nájdení najbližšej hrany polygónu ku kruhu. 4.4 Chceme nájsť hranu e polygónu, ktorej vrchol bude mať najmenšiu hĺbku prieniku s kruhom. Prechádzame vrcholy polygónu V_i, ktoré spájame so stredom kruhu S. Tým sme získali hĺbku prieniku kolízie. Ďalej nájdeme najmenší skalárny súčin s_i vektora $\vec{v} = \vec{V_iS}$ s normálou hrany $\vec{n_e}$. V prípade, že stredy oboch útvarov sú rôzne potom platí, že stred kruhu bude najbližší vrcholu V_i ak $s_i <= 0$. Inak je stred kruhu najbližší hrane polygónu. V prvom prípade ako pozíciu kolízneho bodu tak prehlásíme vrchol V_i, kolíznou normálou bude vektor $\vec{SV_i}$. V druhom prípade bude kolíznou normálou $-\vec{n_e}$ a kolízny bod určíme ako $-\vec{n_e} * r_S + S$, kde r_S je polomer kruhu so stredom S. Opäť pre špeciálny prípad, kedy pozícia kruhu a polygónu je zhodná určíme kolíznu normálu ako $\vec{n_e}$, pozíciu kolízneho bodu ako $\vec{n_e} * r_S + S$. A hĺkbu prieniku ako r_S [4].



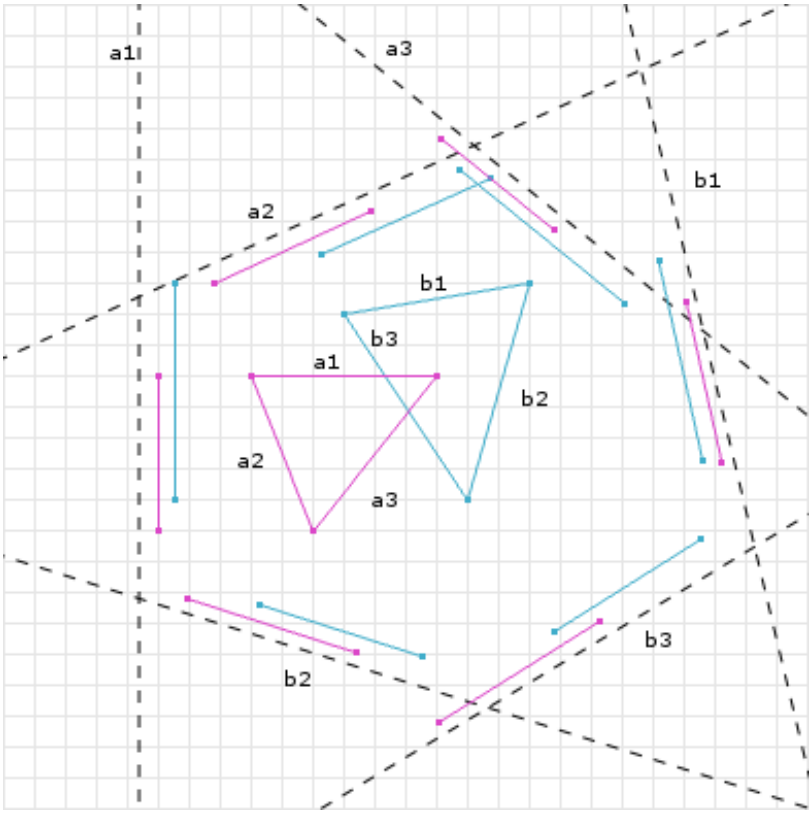
Obr. 4.4: Vzájomná poloha kruhu a polygónu priestore. Štandardná situácia vľavo a špeciálny prípad vpravo.

Polygón vs polygón

Test prieniku dvoch polygónov je o niečo zložitejší. Najčastejšie sú dva prístupy. Teorém o separačnej osi (ang. Separating axis theorem), ďalej SAT a algoritmus Gilbert-Johnson-Keerthi, ďalej GJK. Algoritmy si uvedieme v nasledovných samostatných podkapitolách.

4.3 Separating axis theorem (SAT)

Jednoduchosť SAT spočíva v premise, že ak existuje os, ktorá oddeľuje dva konvexné objekty, potom tieto objekty nie sú v kolízii. Tiež platí, že objekty nie sú v kolízii ak ich projekcie na zvolenú os sa neprelínajú. Pomocou SAT môžeme otestovať všetky osi a ak narazíme na separačnú os, môžeme prehlásiť, že telesá nie sú v kolízii. Ak sa projekcie telies prelínajú pre všetky osi, môžeme prehlásiť, že telesá sú v kolízii. Osi, ktoré takto musíme otestovať sú normálami hrán testovaných polygónov. Na určenie kolízie tak prechádzame cez zoznam vrcholov, pričom si bokom uložíme najmenší prienik a os. Vďaka tomu vieme určiť kolíznu normálu a hĺbku penetrácie [27].



Obr. 4.5: Na obrázku sú znázornené dva polygóny v kolízii a projekcie ich hrán na jednotlivé osi. Vidíme, že vo všetkých prípadoch sú projekcie odpovedajúcich hrán polygónov na všetkých osách v prieniku, to znamená, že aj polygónu sú v prieniku [27].

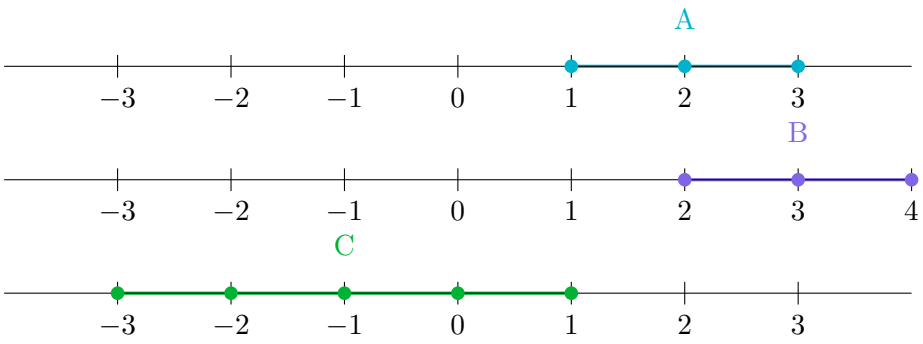
4.4 Algoritmus Gilbert-Johnson-Keerthi (GJK)

GJK tak isto ako SAT operuje len nad konvexnými polygónmi. GJK je iteratívna rýchlo konvergujúca metóda. V prípade trojrozmerného prostredia je dokonca adekvátnejšia než SAT. Algoritmus je intuitívny a vyžaduje len základné matematické znalosti. Princípom algoritmu je aritmetický rozdiel bodov jednotlivých geometrií. Ak dva útvary zdieľajú spoločný bod potom rozdieľ týchto bodov bude rovný 0. V skutočnosti algoritmus nepotrebuje vypočítať rozdiel medzi všetkými možnými pármi bodov, ale bude postačovať malá podmnožina.

GJK v 1D [28]

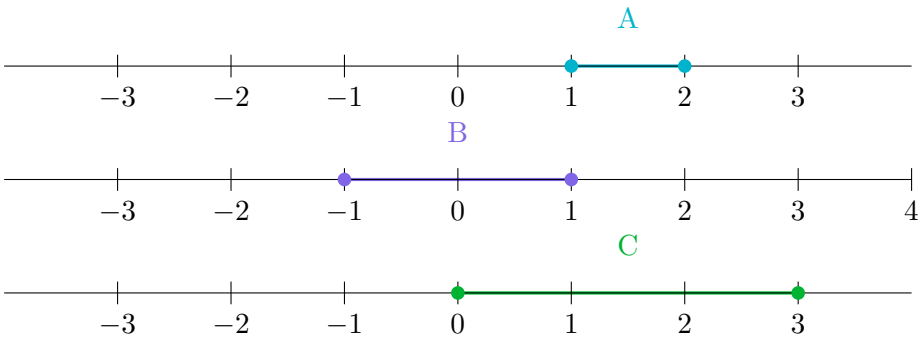
V jednorozmernom priestore znázornenom osou x (Obr. 4.6), budeme operovať s množinou bodov, ktorých súradnice sú dané jedinou hodnotou. Nech počiatok súradnicovej osi je v bode 0. Na obrázku vidíme, že množina bodov $A = \{1, 2, 3\}$ a $B = \{2, 3, 4\}$ tvoria regióny, ktoré zdieľajú spoločnú množinu bodov/región $= \{2, 3\}$. Rozdiel jednotlivých bodov v B a v A nám dáva množinu bodov $C = \{-3, -2, -1, 0, 1\}$. Keďže vo výslednom regióne C sa nachádza počiatok súradnicovej osi, potom prehlásime, že regióny sú v kolízii.

Zrýchlenie algoritmu spočíva v ignorovaní vnútorných bodov v rámci regiónu. Úplne postačuje pokiaľ využijeme hraničné body regiónov a následne skúmame výsledný zjednodušený región, tiež nazývaný simplex (z ang. the simplest possible) (Obr. 4.8). Navyše



Obr. 4.6: Kolíze regiónov

GJK funguje aj v prípade, že simplexy majú len jeden spoločný bod (Obr.4.7) . Ak sme teda schopní zostaviť simplex, ktorý v sebe zahŕňa aj počiatok súradnicovej sústavy, regióny majú aspoň jeden bod prieniku.



Obr. 4.7: Kolíze regiónov

Obr. 4.8: Typy simplexov.

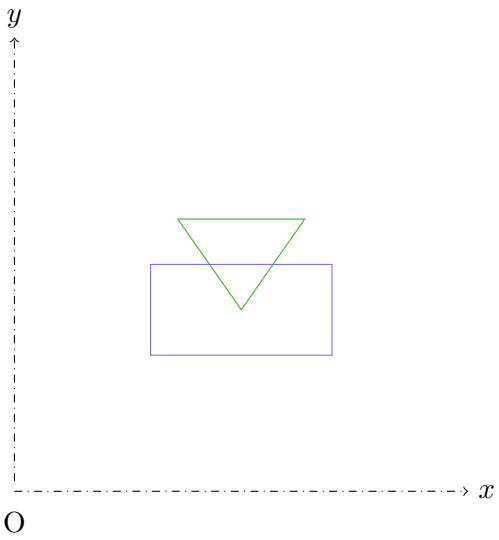
GJK v 2D [23, 28]

Pri dvojrozmernom priestore je princíp analogický k jednorozmernému priestoru. Avšak namiesto regiónu, resp. úsečky budeme hladať dvojrozmerný útvar, ktorý bude ohraničovať počiatok súradnicovej sústavy. Majme dva rovinné útvary trojuholník A a štvoruholník B (4.9). Pokiaľ vykonáme operáciu aritmetického rozdielu nad množinou ich vrcholov, dostávame nový rovinný útvar. Operácia sa v tomto prípade nazýva aj **Minkowského rozdiel/súčet** (MS) (4.10):

$$A \oplus B = \{\vec{a} + \vec{b} : a \in A, b \in B\} \tag{4.2}$$

,kde $\vec{a} + \vec{b}$ je vektorový súčet polohových vektorov $\vec{a}$ a $\vec{b}$. Analogicky je definovaný Minkowského rozdiel:

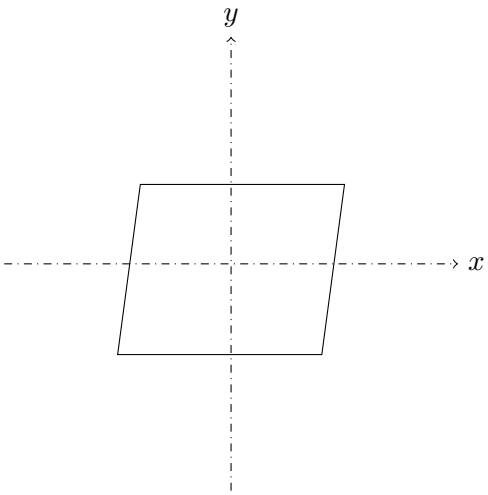
$$A \ominus B = \{\vec{a} - \vec{b} : a \in A, b \in B\} \tag{4.3}$$



Obr. 4.9: Príklad kolízie dvoch polygónov.

Pre dva konvexné polygóny A a B, ktorých MS je $R = A \oplus B$ platí, že R je konvexný polygón a vrcholy R sú súčtom vrcholov A a B. V konečnom dôsledku je minimálna vzdialenosť medzi A a B rovná vzdialenosti medzi R a počiatkom súradnicovej sústavy v rámci Minkowského priestoru. Pri hľadaní kolíznych bodov medzi A a B tak môžeme problém zjednodušiť na nájdenie najbližšieho bodu patriaceho R.

Na prvý pohľad sa môže zdať, že sa nejedná o žiadne vylepšenie, keďže množina bodov patriacich R môže byť rôzne veľká. Algoritmus GJK však nepotrebuje naraz celú množinu bodov tvoriacich MS. Pomocou tzv. podpornej funkcie (ang. support function) vieme postupne nájsť tri nekolineárne body patriace R pre ktoré bude platiť, že ak tieto body ohraničujú stred súradnicového systému Minkowského priestoru, potom aj MS R ohraničuje daný stred. Vo výsledku to znamená, že pôvodné dva útvary A a B sa prelínajú, resp. sú v kolízii. Nájdený trojuholník je podobne ako úsečka v 1D simplexom. Ak žiadna trojica bodov neuzatvára stred súradnicového systému, útvary A a B sa neprelínajú.



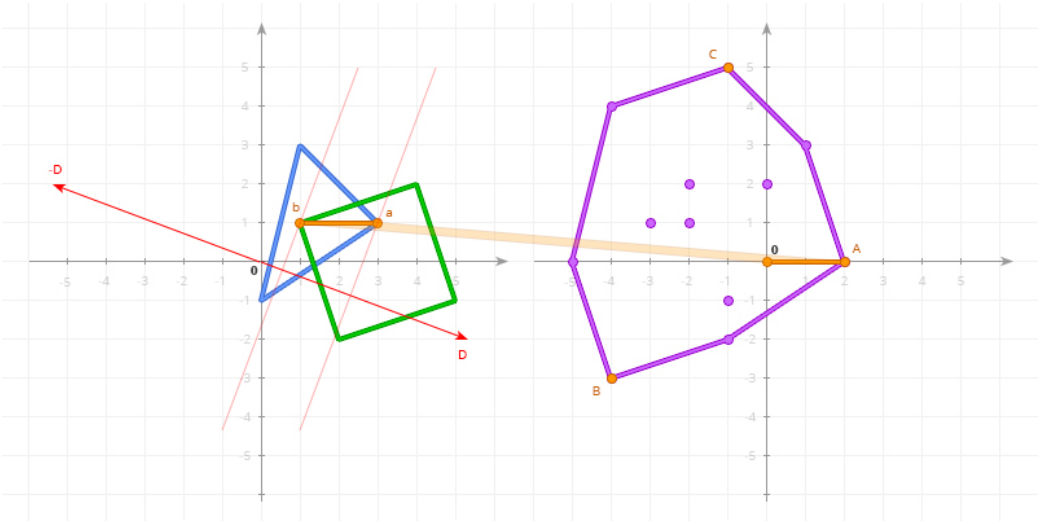
Obr. 4.10: Výsledok Minkowského súčtu útvarov z Obr. 4.9

Podporná funkcia pre GJK

Podobne ako v prípade 1D na získanie výsledného 1-simplexu môžeme ignorovať vnútorné body a počítať len s vonkajšími bodmi. Podporná funkcia získa rozdiel vonkajších bodov v 2D. Mapuje rozdiel dvoch telies na body do Minkowského priestoru. Support funkcia funguje nasledovne (Obr. 4.11):

1. Začni v počiatku súradnicovej sústavy a vyber ľubovolný smer.
2. Vyber jeden z dvoch útvarov. Jedno aký. Od stredu súradnicového systému nájdi najvzdialenejší bod v rámci prvého útvaru v smere danom D.
3. Ďalej podobne nájdeme najvzdialenejší bod v smere $-D$ v druhom tvare.
4. Nájdené body odčítame od seba. Nezáleží na poradí. Výsledný vektor je ich namapovaný rozdiel do 2D minkowského priestoru.

Algoritmus v skratke spočíva v nájdení 2-simplexu z pôvodnej množiny bodov tvoriacich MS. Pokiaľ sme schopný nájsť taký simplex, ktorý ohraničuje stred súradnicového systému v Minkowského priestore môžeme tvrdiť, že samotný MS takisto ohraničuje stred súradnicového systému. Tým pádom pôvodné útvary z ktorých sme získali MS sú v kolízii. V tejto forme algoritmus odpovedá len na otázku či sú dva objekty v kolízii. Pokiaľ chceme získať kolíznu normálu, hĺbku kolízie a kolízne body musíme algoritmus rozšíriť o EPA (Expanding Polytope Algorithm) a kolízne body získať Clipping metódou [23, 24, 25].



Obr. 4.11: Na obrázku vidíme ako support funkcia mapuje vektor z dvoch vrcholov a a b do Minkowského priestoru. [28]

Kapitola 5

Návrh a implementácia

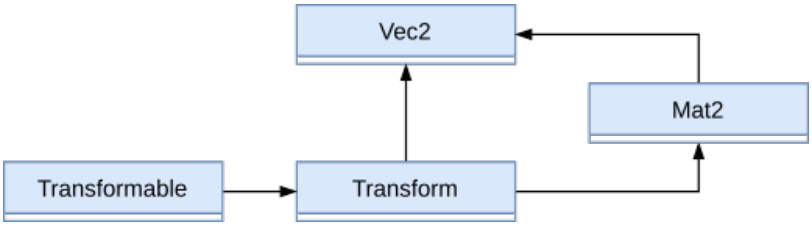
> „Bad programmers worry about the code. Good programmers worry about data structures and their relationships.“ - Linus Torvalds

Voľný preklad citátu: Zlí programátori sa starajú o kód. Dobrí programátori riešia dátové štruktúry a ich vzťahy.

Citát od známeho tvorcu Linuxového jadra nám napovedá, že základom dobrého projektu je návrh dátových štruktúr a ich vzťahov či väzieb medzi nimi. Dátovou štuktúrou budeme rozumieť zoskupenie jednoduchých dátových typov ako je float, int, double. Pri objektovo orientovanom návrhu sa dátove štruktúry rozšíria o množinu operácií, ktoré môžeme s danou dátovou štruktúrou vykonávať a v kontexte jazyka C++ tak pôjde o dátové typy ako napr. trieda (class) a štruktúra (struct). Budeme sa držať múdrej rady, predstavíme si jednotlivé moduly a k nim prislúchajúce štruktúry. Objekty v rámci modulov uvedieme do vzťahov a stručne vysvetlíme ich účel. Použijeme kombináciu objektovo orientovaného a procedurálneho programovania.

5.1 Matematický modul

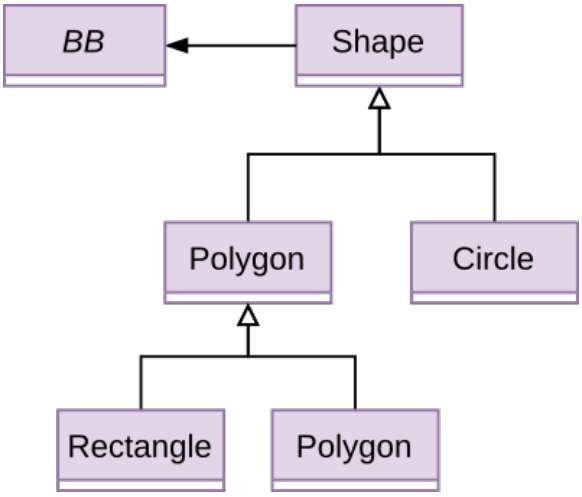
Súčasťou matematického modulu sú štruktúry reprezentujúce vektor, maticu a transformácie (Obr. 5.1). Dáta obsiahnuté v štruktúre vektor a matica sú rovné prvkom dvojrozmerného vektora a dvojrozmernej štvorcovej matici. Implementácia drží predloženú Column-major notáciu a bázové vektory matice sú tak uložené v stĺpcoch. Štruktúra transform reprezentuje spomínanú kompaktnú transformačnú maticu 2.29. Užívateľ knižnice tak môže očakávať implementované štandardné vektorové a maticové operácie spolu s afinnými transformáciami pre transformovanie bodov a vektorov. Súčasťou je aj virtuálna trieda *Transformable*. Inštancie objektu *Transformable* môžeme vytvárať priamo, alebo môžeme využiť dedičnosť a danú triedu využiť ako rodičovskú pre objekty pri ktorých očakávame, že sa budú nachádzať v nejakom priestore. Príkladom je trieda *Body*. Uvedené objekty sú deklarované v hlavičkových súboroch (*.h) v priečinku *include/dnz/maths* a definované v zdrojových súboroch (*.cpp) s odpovedajúcimi názvami v priečinku *src/maths*.



Obr. 5.1: Zjednodušený diagram matematického modulu.

5.2 Modul geometrických tvarov

Základom modulu je abstraktná trieda *Shape*, ktorá definuje všeobecný geometrický útvar pomocu zoznamu vrcholov. Pri vytváraní objektov typu *Shape* sa automaticky uloží aj zoznam hrán pokiaľ sa jedná o typ útvaru *Polygon* spolu s ďalšími dodatočnými informáciami o tvare ako je obsah, polomer, alebo jeho ohraničujúce miery (AABB/BB). Vrcholy sú určené v ľubovolnom lokálnom súradnicovom systéme (podobne ako v prípade na Obr. 2.4). Je na koncovom užívateľovi knižnice aby pri vytváraní inštancií *Shape* si bol vedomý v akom súradnicovom systéme definuje vrcholy tvaru. Modul je obmedzený na konvexné polygóny, ktorých vrcholy musia byť definované v protismere hodinových ručičiek. Súčasťou modulu je aj AABB, ktorý je definovaný pomocou stredu a polomeru (radius) rozpätia pre os x a y. *BB* je samostatný typ. Pri vytvorení inštancie *Shape* sa v prípade typu *Polygon* na základe vstupného zoznamu vrcholov vytvorí v prípade potreby konvexná obálka, aby sme mali istotu, že výsledný polygón bude konvexný. Na základe konkrétneho tvaru sa určí hmotný stred a zoznam vstupných vrcholov sa upraví tak, že poloha hmotného stredu bude súhlasiť s počiatkom súradnicovej sústavy, teda v bode O = [0,0].



Obr. 5.2: Zjednodušený diagram modulu geometrických tvarov.

5.3 Modul dynamiky

V rámci modulu dynamiky 5.3 sú definované objekty *Space*, *Body*, *Joint*, *Arbiter*, *BBTree*, *Contact*, *Collider*. Modul ďalej obsahuje implementované algoritmy pre hrubú a presnú fázu. Hrubá fáza definovaná v zdrojovom súbore *broadphase.cpp* je riešená pomocou dynamického

aabb stromu, trieda *BBTree* a presná fáza v zdrojovom súbore *narrowphase.cpp*. Súčasťou presnej fázy je aj GJK algoritmus definovaný v zdrojovom súbore *gjk.cpp*.

Space

Základným objektom je *Space* prostredníctvom ktorého môžeme vytvárať inštancie objektu *Body* a pokiaľ máme aspoň dve telesá môžeme od inštancie *Space* vyžiadať ich spojenie prostredníctvom objektu *Joint*. Pri vyžiadaní nových telies musí užívateľ uviesť tvar *Shape* telesa a môže uviesť typ telesa a ukazateľ na inštanciu triedy *Transformable*. Objekt *Space* je zároveň zodpovedný za posun simulácie dopredu v čase o hodnotu Δt a od užívateľa sa očakáva, že túto hodnotu dodá.

Body

Objekt reprezentuje tuhé teleso. Hmotnosť telesa a moment zotrvačnosti sú automaticky vypočítané na základe jeho tvaru, rozmerov a nastavenej hustoty. V prípade potreby je možné hmotnosť telesa a moment zotrvačnosti nastaviť manuálne. Teleso môže byť statické, kinematické a dynamické. Statické teleso má nekonečnú hmotnosť a nulovú rýchlosť. Keďže má nekonečnú hmotnosť správa sa ako keby na neho neboli aplikované žiadne sily. Navyše statické teleso nekoliduje s inými statickými alebo kinematickými telesami. Kinematické teleso má taktiež nekonečnú hmotnosť, ale jeho rýchlosť a uhlovú rýchlosť môže užívateľ nastaviť ľubovolne a simulácia určí jeho novú pozíciu. Kinematické teleso koliduje len s dynamickými telesami. Dynamické teleso má určenú nenulovú hmotnosť a pôsobia na neho sily. Objekt *Body* je typu *Transformable* a zároveň drží ukazateľ na inštanciu *Transformable*. Pokiaľ používateľ pri vytváraní inštancie objektu *Body* zadá ukazateľ na svoj vlastný objekt typu *Transformable*, objekt telesa *Body* inicializuje svoju pozíciu, rotáciu a škálovanie podľa klientského objektu a zároveň keď teleso zmení svoju polohu, alebo rotáciu, aktualizuje zároveň klienta.

Dynamické teleso koliduje s ostatnými dynamickými telesami ako aj s kinematickými a statickými telesami.

Joint

Objekt Joint predstavuje spojenie dvoch telies a obmedzuje ich voľnosť pohybu. Jeho inštancia je pre užívateľa potrebná len ak chce daný spoj odstrániť.

Collider

Collider slúži ako proxy objekt, ktorý spája objekt typu *Shape* s objektom typu *Body*. (jedná sa o zjednodušenú verziu triedy ProxyShape v projekte ReactPhysics3D).

Arbiter

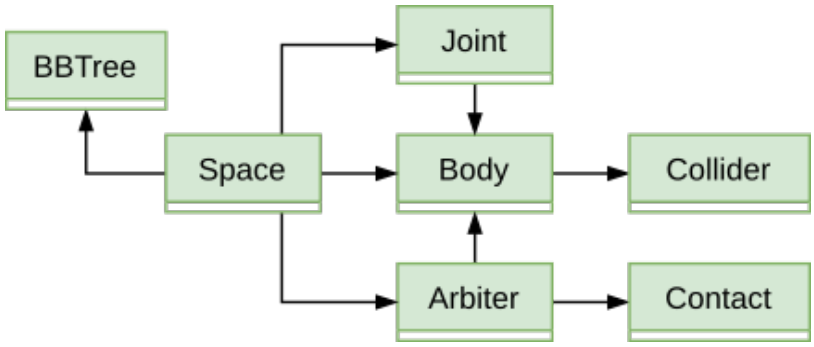
Arbiter slúži ako pomocná trieda pre riešení kolízií v rámci presnej fázy. Jeho úlohou je zistiť či dané telesá sú v kolízii a ak áno neskôr ich kolízie vyriešiť aplikovaním impulzných síl.

Contact

Štruktúra reprezentuje kontakt. Obsahuje kolíznu normálu, pozíciu kolízneho bodu a hĺbku penetrácie telies.

BBTree

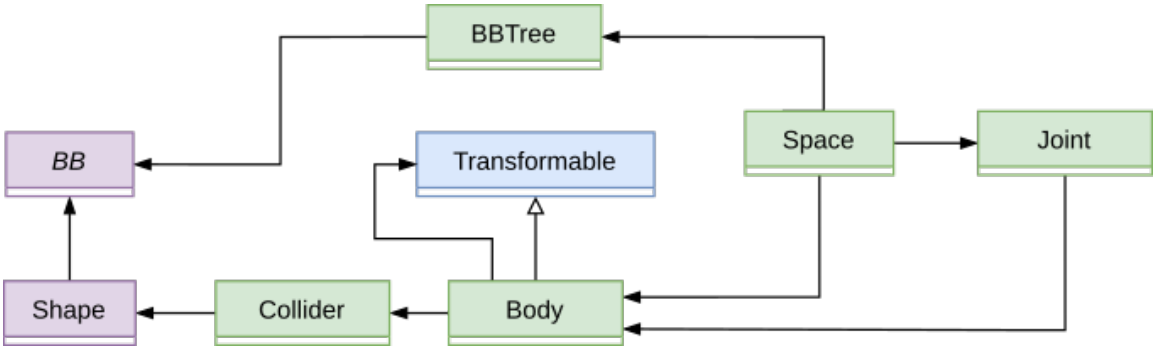
V triede BBTree je definovaný dynamický AABB strom. Jedná sa o binárny strom, ktorého uzly sú uložené v lineárnom zozname a uzly na seba odkazujú pomocou indexov na zoznam. Jedná sa o odlišný prístup oproti štandardnej implementácie, kedy jednotlivé uzly na seba odkazujú pomocou ukazateľov a uzly sú dynamicky vytvárané v náhodných miestach v pamäti.



Obr. 5.3: Zjednodušený diagram modulu dynamiky.

5.4 Spolupráca jednotlivých modulov

Jednotlivé moduly spolupracujú. Diagram 5.4 nám približuje celkový pohľad na objektový návrh.



Obr. 5.4: Zjednodušený diagram prezentuje spoluprácju jednotlivých modulov.

Kapitola 6

Príklad použitia

6.1 Zostavenie knižnice

Samotná knižnica nevyužíva žiadne špeciálne dodatočné knižnice a na jej zostavenie stačí GCC [1]. Pre zautomatizovanie procesu je použítý nástroj GNU Make [2], ktorý je súčasťou väčšiny unixových platforiem. V koreňovom adresári projektu *dnz* sa nachádza súbor *Makefile*, ktorý slúži ako recept na zostavenie projektu. V podadresári *dnz/makefiles* sú dodatočne definované recepty pre zostavenie samotnej knižnice s názvom *dnz* ako aj ukážkovej aplikácie *sandbox*. Pri zavolaní príkazu

```
$ make help
```

z koreňového adresára projektu *dnz* sa vypíšu možnosti konfigurácie prekladu. Príkaz

```
$ make dnz config=[debug|release|wasm] linkage=[static|shared]
```

zostaví knižnicu pre danú konfiguráciu. Pre optimálny výkon sa odporúča použiť *release* konfiguráciu prípadne pri zostavovaní pre WebAssembly [3] potom konfiguráciu *wasm*. Pre zostavenie pre WebAssembly je potrebné mať nainštalovaný a správne nastavený nástroj emscripten [4]. Pri použití konfigurácie pre WebAssembly je však možné použiť len statické linkovanie.

6.2 Aplikácia Sandbox

Súčasťou projektu je aj sandboxová (v zmysle sandboxových počítačových hier) aplikácia, ktorá prezentuje funkčnosť knižnice. Zdrojové súbory k aplikácii sa nachádzajú v priečinku *dnz/sandbox*. Jej zostavenie je analogické k zostavovaniu knižnice s tým rozdielom, že pre zostavenie na desktopovú platformu je nutné mať navyše nainštalovanú knižnicu GLFW[5]. Stručne si uvedieme využitie knižnice v praxi.

[1] https://gcc.gnu.org/
[2] https://www.gnu.org/software/make/
[3] https://webassembly.org/
[4] https://emscripten.org/
[5] https://www.glfw.org/

Hlavný cyklus

Na to aby sme mohli v simulácii napredovať v čase, potrebujeme zavolať funkciu *Step(float dt)* objektu *Space*. Ako vidíme metóda vyžaduje atribút, ktorý jej povie o koľko simuláciu v čase posunúť.

Každá počítačová hra obsahuje tzv. game loop, ktorý je jej ústrednou časťou. Akýkoľvek kód, procedúra, ktorá robí hrú interaktívnou a dynamickou je súčasťou tohto nekonečného cyklu. Štandardne sa rozdeľuje na niekoľko fáz. Budeme uvažovať fázu aktualizácie herných objektov (*step*) a fázu vykreslovania (*draw*) s fixným krokom simulácie $MS\_PER\_UPDATE = 1/60$ pre funkciu *Step(float dt)*. Príkladom môže byť aj nasledovný cyklus 6.1 [29]:

```
while (!context_->IsClosed())
{
        double current = glfwGetTime();
        dt   = current - last;
        accumulator += current - last;
        last = current;
        while (dt >= MS_PER_UPDATE)
        {
                context_->Step(MS_PER_UPDATE);
                dt -= step;
        }
        context_->Draw();
}
```

Výpis 6.1: Príklad hlavného cyklu pre herný engine.

Premenná *context_* v 6.1 je inštanciou objektu *Context*, ktorý môžeme vnímať ako príklad hernej scény, ktorá manažuje jednotlivé herné objekty *GameObj* a ich posun v simulácii dopredu pomocou inštancie objektu *Space* a vykreslovanie herných objektov funkciou *Draw()*. Herný objekt *GameObj* môžeme chápať ako entitu v hernom engine s istými atribútmi a komponentami. V našom prípade môžeme priradiť komponent pre vykreslenie tvaru *Shape* a komponent pre tuhé teleso pridaním ukazateľa na inštanciu triedy *Body*, ktorá v konštruktore dostane ukazateľ na objekt *Transformable*. Keďže *GameObj* je typu *Transformable* použijeme jeho inštanciu. Vďaka tomu sa nám pozícia a rotácia nášeho herného objektu v čase aktualizuje.

Vytvorenie herného objektu spolu s komponentami pre vykreslenie a komponetom telesa potom môže vyzerať nasledovne 6.2:

```
GameObj *go = context_->NewGameObj();

go->AddRenderableComponent(*shapes_[selectedShape_], COLOR_KINEMATIC);
go->SetPosition(cursor);
go->SetScale(Vec2(5, 5));

Body * body = go->AddBodyComponent(*shapes_[selectedShape_]);
body->SetKinematic();
```

Výpis 6.2: Príklad vytvorenia inštancie triedy GameObj spolu s pridaním komponentu tuhého telesa.

Klávesa	Akcia
Ľavé tl. myši	Vytvorenie statického objektu
CTRL + Ľavé tl. myši	Vytvorenie kinematického objektu
Shift + Ľavé tl. myši	Vytvorenie dynamického objektu
Pravé tl. myši	Výber telesa
Prostredné tl. myši	Vytvorenie kyvadla
1,2,3,4	Výber tvaru pre nové teleso
W,A,S,D	Posun označených telies
Q,E	Rotácia označených telies
H	Zapnutie/Vypnutie vykresľovania BV Hierarchie
C	Zapnutie/Vypnutie vykresľovania kolíznych bodov

Tabuľka 6.1: Ovládanie aplikácie Sandbox.

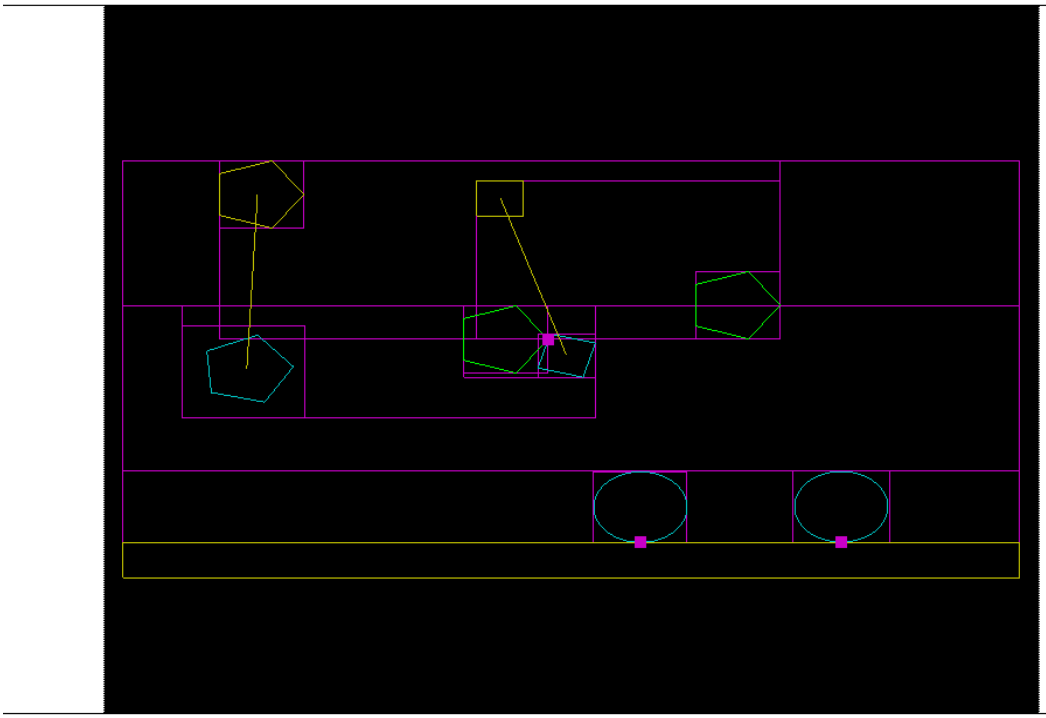
Pričom funkcia *AddBodyComponent()* 6.3 využíva inštanciu triedy *Space* uloženú v objekte *Context*:

```
Body * GameObj::AddBodyComponent(const Shape & shape)
{
        if (body_)
        {
                body_->Release();
        }

        body_ = context_.space_.NewBody(shape, Body::Type::STATIC, this);

        return body_;
}
```

Výpis 6.3: GameObj si uloží vytvorenú inštanciu objektu Body

Funkčnosť aplikácie

Aplikáciu môžeme spustiť po preložení buď ako desktopovú alebo ako webovú aplikáciu (po nasadení na server), ktorú nájdeme v zložke *build/"config"/sandbox/"linkage"/bin*. Kde *"config"* je názov konfigurácie, debug, release, wasm a *"linkage"* je buď static alebo shared podľa spôsobu linkovania knižnice. Po spustení aplikácie môžeme otestovať jej funkčnosť, vytváraním herných objektov, ktoré majú automaticky priradené inštancie objektu telies *Body*. Ovládanie je popísané v tabuľke 6.1. Obrázok funknčnej simulácie s niekoľkými telesami, kyvadlom vo webovom prehliadači a vykreslenou hierarchiou BVH a kolíznymi bodmi (fialová farba) (Obr. 6.1).



Obr. 6.1: Ukážka funknčnej simulácie s vytvorenými dynamickými kruhmi (modré), kinematickými polygónmi (zelené) a statickými polygónmi (žlté). Mriežka znázorňuje rozdelenie priestoru pomocou BVH a ružové štvorčeky znázorňujú kolízne body.

Kapitola 7

Záver

Cieľom práce bolo ozrejmiť čitateľovi elementárne princípy simulovania dynamiky tuhých telies v počítačových hrách a naštudované teoretické znalosti aplikovať pri vývoji samostatnej knižnice v jazyku C++. Samotná téma je veľmi obsiahla a častokrát aj ja som mal problém sa zorientovať v závale informácií a poznatkov. Zoznámil som sa s niektorými existujúcimi riešeniami a z nich som sa snažil vyextrahovať dôležité základné funkcionality a preniesť ich do tejto práce. Kvôli dodržaniu rozsahu bakalárskej práce som sa sústredil na dvojrozmerný priestor. Matematické a fyzikálne znalosti sa ukázali ako veľmi prospešné a nutné pokiaľ má čitateľ záujem podielať sa na procese vývoja herných enginov. V teoretickej časti práce je tak čitateľ uvedený do sveta lineárnej algebry, vektorov a matíc, ktoré slúžia hlavne ako matematický aparát pre popis fyzikálnych veličín v rámci tretej kapitoly. Štvrtá kapitola pojednáva o kolíziách telies. Následne z uvedených teoretických princípov je predložený návrh výslednej knižnice v piatej kapitole. Šiesta kapitota nakoniec názorne prezentuje zostavenie a použitie knižnice. Vďaka WebAssembly je možné zostaviť statickú knižnicu, ktorú je možné použiť pri vytváraní webovej aplikácie. Príkladom použitia je testovacia aplikácia, Sandbox, ktorú je možné po zostavení umiestniť na webový server. To nám zaručí, že výslednú aplikáciu spustíme v prostredi webového prehliadača. Stačí ak prehliadač podporuje WebGL verziu 2.0. Po prečítaní práce bude mať čitateľ elementárnu znalosť od ktorej sa môže ďalej odraziť pri práci na podobných projektoch. V priebehu práce som sa naučil rozpoznať relevantné zdroje. Práca mi pomohla prepojiť teoretické znalosti z viacerých oborov a uviesť do praxe abstraktné matematické koncepty. Možností rozšírenia a pokračovania práce sú rôzne.Možné sú optimalizácie jednotlivých algoritmov, podpora kolízii konkávnych tvarov. Môžeme rozdeliť priestor do buniek a každú buňku riešiť v samostatnom vlákne. Podpora dvojnásobnej precíznosti pri výpočtoch pomocou double. Využitie AVX a SSE inštrukcií pre zrýchlenie násobenia matíc a vektorov. Použitie kvadratického stromu namiesto binárneho. Môžeme použit nielen AABB, ale aj iné približné ohraničujúce geometrie. Samozrejme aj dvojrozmerný priestor má stále čo ponúknuť, zjavným rozšírením je prechod do tretieho rozmeru. V závere jednoznažne odporúčam pre záujemcov knihu Real-Time Collision Detection, autor Christer Ericson.

Literatúra

[1] Arak M. Mathai, H. J. H. *Linear Algebra: A Course for Physicists and Engineers.* Walter de Gruyter GmbH & Co KG, 2017. ISBN 978-3-11-056235-4.

[2] *Vector Math* [online]. GodotEngine, 2020 [cit. 2020-01-12]. Dostupné z: https://docs.godotengine.org/en/stable/tutorials/math/vector_math.html.

[3] Millington, I. *Game Physics Engine Development.* Elsevier, 2007. ISBN 978-0-12-369471-3.

[4] Szauer, G. *Game Physics Cookbook.* Packt Publishing Ltd, 2017. ISBN 978-1-78712-366-3.

[5] Tremblay, C. *Mathematics for Game Developers.* Stacy L. Hiquet, 2004. ISBN 1-59200-038-X.

[6] Šedivý, O. *Geometria II.* Nitra : UKF, 2008. ISBN 978-80-8094-345-5.

[7] Marsh, D. *Applied Geometry for Computer Graphics and CAD.* Springer, 1999. ISBN 978-1-84628-109-9.

[8] *Lineární Algebra* [online]. mathonline, 2015 [cit. 2020-02-04]. Dostupné z: https://mathonline.fme.vutbr.cz/download.aspx?id_file=1220.

[9] Zlatoš, P. *Lineárna Algebra a Geometria* [online]. uniba, 2011 [cit. 2020-02-08]. Dostupné z: http://thales.doa.fmph.uniba.sk/zlatos/la/LAG_A4.pdf.

[10] Zdeněk Florian, M. S. *Technická mechanika 1 - Statika.* 2004.

[11] David Halliday, J. W. *Fundamentals of Physics.* John Wiley & Sons, Inc., 2011. ISBN 978-0-470-46908-8.

[12] Weideman, T. *Systems with Multiple Particles* [online]. 2019 [cit. 2020-03-02]. Dostupné z: https://phys.libretexts.org/Courses/University_of_California_Davis/UCD%3A_Physics_9HA_%E2%80%93_Classical_Mechanics/4%3A_Linear_Momentum/4.2%3A_Center_of_Mass.

[13] *Hitbox* [online]. 2019 [cit. 2020-04-12]. Dostupné z: https://counterstrike.fandom.com/wiki/Hitbox.

[14] Ericson, C. *Real-Time Collision Detection.* Morgan Kaufmann Publishers Elsevier, 2005. ISBN 1-55860-732-3.

[15] Sikchi, H. *Convex Hulls: Explained* [online]. 2017 [cit. 2020-04-12]. Dostupné z: https://medium.com/@harshitsikchi/convex-hulls-explained-baab662c4e94.

[16] Skala, V., Smolik, M. a Majdisova, Z. Reducing the Number of Points on the Convex Hull Calculation Using the Polar Space Subdivision in E2. In:. Október 2016, s. 40–47.

[17] Mei, G., Tipper, J. C. a Xu, N. An algorithm for finding convex hulls of planar point sets. *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*. 2012, s. 888–891.

[18] Erickson, J. *Convex Hulls* [online]. jeffe.cs.illinois.edu, 2018 [cit. 2020-16-3]. Dostupné z: https://jeffe.cs.illinois.edu/teaching/compgeom/notes/01-convexhull.pdf.

[19] Altshiller Court, N. *College Geometry*. Doven Publications, Inc., 2007. ISBN 0-486-45805-9.

[20] Bellot, G. *The Centroid of Convex Polygons* [online]. 2019 [cit. 2020-05-12]. Dostupné z: https://bell0bytes.eu/centroid-convex/.

[21] Jandačka, D. *Základy biomechaniky tělesných cvičení* [online]. 2012 [cit. 2020-05-12]. Dostupné z: http://www.fsps.muni.cz/emuni/data/reader/book-1/Impresum.html.

[22] *7 Mechanika tuhého telesa* [online]. [cit. 2020-03-25]. Dostupné z: https://fyzika.uniza.sk/sk/zaklady/zaklady/07.pdf.

[23] dyn4j. *GJK (Gilbert–Johnson–Keerthi)*. 2010 [cit. 2020-05-12]. Dostupné z: http://www.dyn4j.org/2010/04/gjk-gilbert-johnson-keerthi/.

[24] dyn4j. *EPA (Expanding Polytope Algorithm)*. 2010 [cit. 2020-05-12]. Dostupné z: http://www.dyn4j.org/2010/05/epa-expanding-polytope-algorithm/.

[25] dyn4j. *Contact Points Using Clipping*. 2011 [cit. 2020-05-12]. Dostupné z: http://www.dyn4j.org/2011/11/contact-points-using-clipping/.

[26] Coutinho, M. G. *Guide to Dynamic Simulations of Rigid Bodies and Particle Systems*. Springer Science & Business Media, 2012 [cit. 2020-05-05]. ISBN 9781447144175.

[27] dyn4j. *SAT (Separating Axis Theorem*. 2010 [cit. 2020-05-12]. Dostupné z: http://www.dyn4j.org/2010/01/sat/.

[28] Kroitor, I. *Gjk.c – Gilbert-Johnson-Keerthi In Plain C*. 2019 [cit. 2020-05-22]. Dostupné z: https://github.com/kroitor/gjk.c.

[29] Fiedler, G. *Fix Your Timestep!* 2004 [cit. 2020-05-30]. Dostupné z: https://gafferongames.com/post/fix_your_timestep/.

Príloha A

Obsah pamäťového média

- **dnz** - Priečinok obsahujúci zdrojové súbory knižnice a názornej aplikácie Sandbox.
- **sprava** - Priečinok obsahujúci zdrojové súbory Latex.
- **projekt.pdf** - Text bakalárskej práce v PDF.