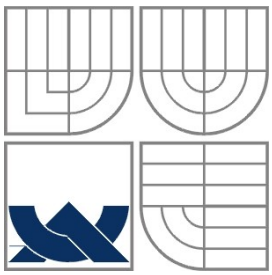
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

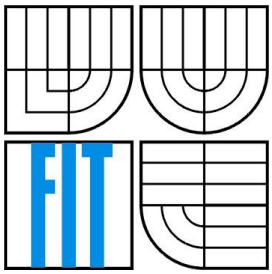
Fakulta informačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2016 Tomáš Chlubna



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KNIHOVNA PRO DETEKCI KOLIZÍ

COLLISION DETECTION LIBRARY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ CHLUBNA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2016

Abstrakt

Tato práce řeší problém detekce kolizí netriviálních polygonálních modelů v trojrozměrném prostoru. Obecně existují postupy, jak tyto kolize matematicky vyjádřit a vypočítat. Pro použití v oblasti informačních technologií jsou však takové metody často nepoužitelné z hlediska výkonu a paměťové náročnosti. Také oproti reálnému světu je třeba pracovat s diskrétním časem, což vede k nutnosti implementace algoritmů, schopných nejen kolize detekovat v daném časovém okamžiku, ale také je předvídat podle dostupných informací o pohybu objektů ve scéně.

Návrh řešení vychází zejména z technik používaných v odvětví herního vývoje a fyzikálních simulací. V práci jsou tedy zahrnuty i mechanismy pro optimalizaci, reprezentaci scény a její vykreslování s využitím grafické karty.

Abstract

This thesis deals with the problem of detecting collisions of nontrivial polygonal models in three-dimensional space. In general, there are methods describing how to mathematically express and calculate such collisions. However, such methods are usually unsuitable for usage in information technology due to the performance and memory requirements. It is also necessary to work with the discrete time that is not present in the real world. That brings the need to implement algorithms that are not only able to detect the collisions in a specific point in time, but also to predict them according to the available data about the movement of the objects in the scene.

The solution uses game development and physics simulations techniques. Therefore, this works describes some optimization techniques as well as suitable scene representation formats and GPU rendering mechanisms.

Klíčová slova

Detekce kolizí, trojrozměrný prostor, polygonální model, SAT, OpenGL, engine, fyzikální simulace, C++, herní vývoj

Keywords

Collision detection, three-dimensional space, polygonal model, SAT, OpenGL, engine, physical simulation, C++, game development

Citace

Tomáš Chlubna: Knihovna pro detekci kolizí, bakalářská práce, Brno, FIT VUT v Brně, 2016

Knihovna pro detekci kolizí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Poloka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Chlubna
18. května 2016

Poděkování

Můj velký dík patří vedoucímu této práce, panu Ing. Lukáši Polokovi za veškerou pomoc, rady a vedení. Dále bych rád poděkoval Fakultě informačních technologií VUT v Brně za znalosti, kterých se mi zde dostalo. V neposlední řadě patří můj dík všem autorům citovaných zdrojů za tuto zprostředkovanou pomoc.

Obsah

1 Úvod

K implementaci funkční detekce kolizí lze přistupovat mnoha způsoby, proto je nutné seznámit se v následujících kapitolách s technikami, které se v různých kombinacích často vyskytují v již existujících implementacích. Techniky dále zmíněné v tomto dokumentu jsou hojně využívány zejména u jader fyzikálních enginů, které slouží nejčastěji jako základ simulací s grafickým výstupem, nacházejícím uplatnění například v herním, vědeckém či filmovém odvětví.

Cílem detekce kolizí je vytvoření části simulačního modelu reálného světa. Z tohoto důvodu je třeba vytvořit abstraktní model potřebných fyzikálních jevů a zákonitostí. K popisu těchto jevů se nabízí množství geometrických a fyzikálních vzorců, které však není vhodné implementovat přímo. Je nezbytné brát v potaz dostupné výpočetní zdroje, jejich strukturu a možný výkon a podle toho navrhnout postupy s co největší možnou optimalizací a minimálním počtem výpočtů. Dokument obsahuje několik kapitol zaměřených na techniky pro akceleraci výpočtů. Je však žádoucí, aby samotné jádro pracovalo efektivně i bez podpůrných optimalizací. Zároveň je nutné zachovat veškeré zákony, které zajišťují reálně se jevící chování objektů ve scéně.

Z výše zmíněných důvodů vyplývá nutnost seznámení se s tématy z analytické geometrie, jako jsou vektorové počty, či projekce do 2D či 1D prostoru. Dále jsou to základní fyzikální vztahy pro popis pohybu tělesa a jeho reakce na síly na něj působící. Na vyšší úrovni je nutné znát možnosti popisu 3D scény a techniky pro práci s ní. Dále práce zahrnuje kapitoly zaměřené na popis reprezentace 3D objektů a jejich modelů, stromových struktur pro reprezentaci scény a specifických technik detekce kolizí. Okrajově je zmíněna také práce s OpenGL a vztah mezi vykreslováním a výpočty fyzikálního rázu v rámci algoritmu řízení simulace.

Druhá kapitola se věnuje teoretickému popisu technik pro realizaci simulace pohybu těles v prostoru. Jsou zde popsány základní fyzikální zákony popisující pohyb těles v reálném světě a od nich odvozeny problémy při simulaci tohoto pohybu za použití výpočetní techniky. Třetí kapitola se zabývá hlavním tématem práce, kterým je detekce kolizí obecných polygonálních objektů v trojrozměrném prostoru. V této kapitole je čtenář nejprve seznámen s reprezentací objektů v simulaci. Objekty mají fyzikální vlastnosti, používané pro výpočty jejich pohybu například po vzájemné kolizi a také obsahují struktury popisující jejich tvar a další vlastnosti určené k vykreslování. Dále jsou popsány základní metody detekce kolizí u jednoduchých objektů a následně v této práci implementována technika s názvem Separating Axis Theorem, použitelná i pro komplexní objekty. Na konci kapitoly jsou zmíněny možné techniky pro akceleraci výpočtů detekce kolizí. Čtvrtá kapitola popisuje samotnou implementaci práce, včetně všech modulů zajišťujících běh simulace jako je algoritmus řízení simulace, modul fyziky, modul detekce kolizí a další implementované struktury pro práci s objekty. Pátá kapitola obsahuje výkonnostní měření nad demonstrační aplikací, která je výstupem této práce.

2 Simulace pohybu těles

Pojem pohyb bude pro potřeby této práce definován jako děj, při kterém dochází ke změně polohy pevného tělesa za určitý čas. Tato kapitola popisuje metody simulace pohybu rigidních těles ve 3D prostoru. Jelikož fyzikální děje v reálném světě jsou příliš komplexní, je třeba definovat míru abstrakce fyzikálního modelu, většinou podle zaměření aplikace a předpokládaných případů užití. První část kapitoly se zabývá zejména popisem pohybu z hlediska fyziky, zatímco druhá část je zaměřena na realizaci samotné simulace pohybu ve výpočetním systému.

2.1 Příčiny změn pohybového stavu těles

Reálný svět se stejně jako počítačový program řídí jistými zákony, jejichž množství a složitost dalece přesahují možnosti modelování. Pro účel této práce bude stačit strohá abstrakce těchto zákonů, které popisují základní pohybové vlastnosti těles. Z těchto zákonů lze odvodit jednoduché vztahy použitelné v simulaci. Jedná se zejména o vztahy popisující vzájemné silové působení více těles na sebe navzájem, a reakce na tyto síly.

2.1.1 Newtonovy pohybové zákony

Důležitým pojmem v oblasti studia pohybu těles je síla. Síla je definována jako vektorová fyzikální veličina. Síla není přímou příčinou pohybu tělesa, jejím působením však dochází ke změnám pohybového stavu těles. Toto působení popisují fyzikální zákony formulované Isaacem Newtonem roku 1687 v jeho díle Matematické principy přírodní filosofie [1].

Prvním zákonem je zákon setrvačnosti. Ten říká, že těleso setrvává v klidu, nebo rovnoměrném přímočarém pohybu, jestliže na něj nepůsobí žádné vnější síly, nebo je výslednice těchto sil nulová. Druhý zákon, také zvaný zákon síly definuje stav, kdy na těleso působí síla. Těleso se pak pohybuje se zrychlením, přímo úměrným působící síle a nepřímo úměrným hmotnosti tělesa. Zákon je možno zapsat následujícím vztahem:

$$F = ma = m\frac{dv}{dt} \tag{2.1}$$

Třetí zákon akce a reakce je nejtěsněji spjat s problémem reakce na detekovanou kolizi mezi dvěma objekty. Tento zákon říká, že jestliže jedno těleso působí na druhé silou, pak druhé těleso působí stejně velkou silou opačné orientace na těleso první. Tyto zákony jsou jádrem fyzikální simulace, pro definování pohybu objektů. Objekty musí reagovat na síly na ně působící, například právě při kolizích mezi sebou.

2.1.2 Reakce těles na změnu pohybového stavu

Z Newtonových zákonů vyplývá, že na těleso musí pro změnu pohybového stavu působit jedna či více vnějších sil. Působení těchto sil může vyvolat dva druhy pohybu tělesa. Prvním druhem je posuvný (translační) pohyb, kdy se všechny body tělesa pohybují stejnou rychlostí po vzájemně rovnoběžných trajektoriích. Druhý druhem je pohyb otáčivý (rotační), kdy se všechny body tělesa pohybují se stejnou úhlovou rychlostí po soustředných kružnicích, jejichž středy leží na ose otáčení. Posuvný pohyb ve směru vektoru rychlosti vypočítáme z působící síly za pomoci základních fyzikálních vztahů.

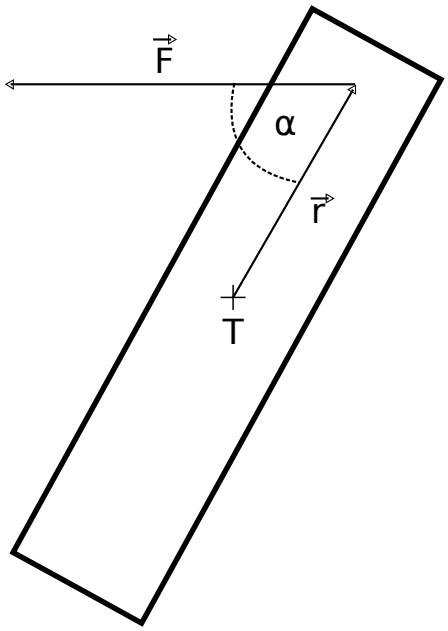
$$\vec{F} = m\vec{a}, \vec{a} = \vec{v}/t, \vec{v} = x/t \tag{2.2}$$

Při srážce dvou těles dochází k předání energie a tělesa získávají nové rychlosti. Fyzika definuje dva základní druhy těchto srážek. Prvním druhem je pružný ráz, při kterém platí zákon zachování mechanické energie a zároveň také zákon zachování hybnosti. Druhým je nepružný ráz, který zachovává pouze zákon zachování hybnosti. V druhém případě dochází ke ztrátám energie vlivem okolních faktorů jako je tření, přeměna na vnitřní energii tělesa apod. Oba rázy shrnují vzorce (2.3), kde konstanta k je koeficient restituce udávající míru ztráty mechanické energie těles. Pokud je tento koeficient nulový, jedná se o dokonale nepružný ráz, pokud je naopak roven jedné tak se jedná o dokonale pružný ráz.

$$\begin{aligned} v_a &= \frac{m_a - km_b}{m_a + m_b} v_a + \frac{(1+k)m_b}{m_a + m_b} v_b \\ v_b &= \frac{(1+k)m_a}{m_a + m_b} v_a + \frac{m_b - km_a}{m_a + m_b} v_b \end{aligned} \tag{2.3}$$

Pro popis otáčivého účinku síly na těleso je definována vektorová fyzikální veličina s názvem moment síly. Moment síly můžeme vyjádřit jako vektorový součin polohového vektoru, který začíná v bodě osy otáčení (v těžišti tělesa) a míří do bodu působiště síly a síly samotné (2.4). Vektory jsou znázorněny na obrázku 2.1.

$$\vec{M} = \vec{r} \times \vec{F} \tag{2.4}$$



Obrázek 2.1: Důležité vektory pro moment síly

Pro popis otáčení ve 3D prostoru je definován vektor úhlové rychlosti, který obsahuje tři složky rotace podle tří souřadných os. Ten lze odvodit právě z momentu síly. Směr vektoru momentu síly určuje směr osy otáčení. Velikost momentu síly lze určit jako součin vzdálenosti působiště síly a osy otáčení (velikost polohového vektoru), velikosti síly a funkce sinus úhlu mezi vektorem síly a polohovým vektorem.

$$\vec{M} = \vec{r}\,\vec{F}\sin\alpha \tag{2.5}$$

Výše zmíněné vztahy postačují k určení základních posuvů či rotací těles. V případě rozsáhlejší fyzikální simulace, kde působí také vnější síly jako je například gravitace však dochází k problémům. Aby bylo možné správně určovat změny rychlosti těles a vyvažování působících sil, definuje fyzika vektorovou fyzikální veličinu zvanou impuls síly. Ta popisuje časový účinek síly na těleso (2.6). Impuls můžeme také vyjádřit jako rozdíl hybností tělesa v daném čase.

$$I = \int_{t1}^{t2} \vec{F}\,dt = \Delta\vec{p} \tag{2.6}$$

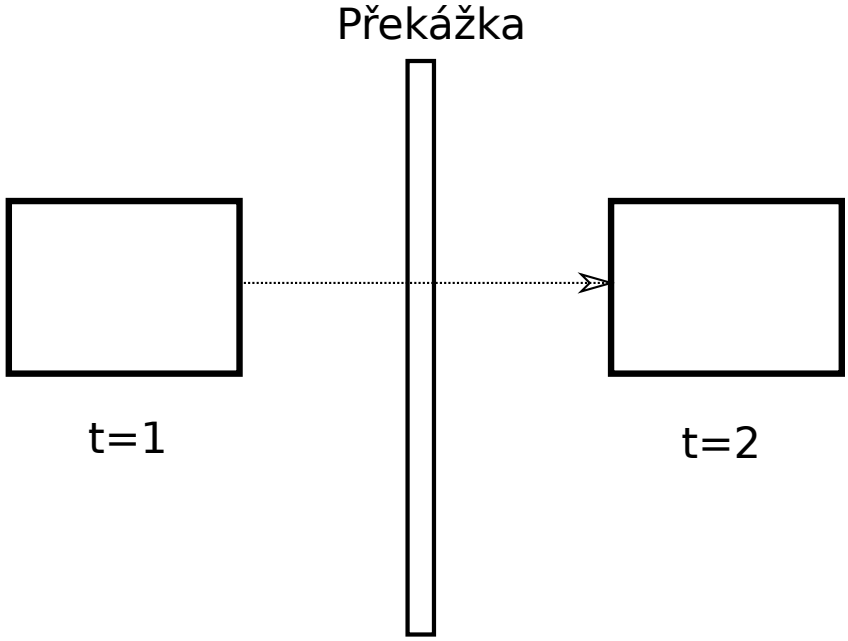
2.2 Spojitý a diskrétní čas

Při simulaci fyzikálního modelu reálného světa je nutné se zabývat problémem času a jeho odlišného pojetí ve světě reálném a v strojově řízené simulaci. V oblasti výpočetní techniky existuje čas diskrétní, zatímco v reálném světě spojitý. Pro diskrétní čas je charakteristická skoková změna hodnot proměnných závislých na simulačním čase, zatímco u spojitého času není možné definovat délku skoku změny. Interval spojitého času tedy lze nekonečně dělit, zatímco interval času diskrétního sestává z konečné množiny přesně definovaných okamžiků.

Jelikož pohyb těles v simulaci je těsně spjat se změnou času, je také ovlivněn problémem diskrétního času v oblasti výpočetní techniky. V oblasti simulací s vykreslováním scény problém ještě narůstá, jelikož simulaci popisuje grafický výstup, který má omezenou možnost překreslování snímků. Typicky se jedná o třicet až šedesát snímků za sekundu.

2.2.1 Tunelování

Objekty v simulacích se skokově posouvají z jednoho místa na druhé, což může vést k chybám, zejména při detekci kolizí. Ani při zvýšeném vzorkování pohybu nelze zaručit bezchybnost. Výrazným problémem detekce kolizí v závislosti na diskrétním čase je tunelování. Tento problém nastane ve chvíli, kdy těleso ve svých dvou diskrétních pozicích mine překážku, která leží na jeho trajektorii. Tato překážka zůstane v mezeře mezi dvěma pozicemi tělesa, jako je znázorněno na následujícím obrázku.



Obrázek 2.2: Problém tunelování objektů v diskrétním čase

Existuje několik způsobů řešení tohoto problému. Již zmíněné zvýšení vzorkování pohybu může zaručit poměrně spolehlivý výsledek. Větší počet poloh tělesa však vyžaduje více výpočtů. Například pokud zvýšíme vzorkování pohybu dvakrát, potom také dvakrát vzroste počet nutných

výpočtů transformací tělesa. Je také třeba brát na zřetel, že tato metoda možnost tunelování neeliminuje, pouze zmenšuje pravděpodobnost, že k tomuto jevu dojde.

Jiným způsobem je využití obalových těles objektů, následná transformace tohoto obalu podle pohybu tělesa a detekce kolize s tímto upraveným obalem. Také je možné vytvoření obalu okolo obou diskrétních pozic tělesa při pohybu. V tomto obalu je pak nutné kontrolovat veškeré potenciální překážky. Je také možné toto obalové těleso rekurzivně půlit a hledat kolize s těmito menšími obaly [3].

Pro účely bezchybné detekce kolizí je však nutné nespoléhat se na přibližné výsledky, ale přesně určit místo a čas kolize. V herním průmyslu například nemusí být vždy nutné mít naprosto přesnou detekci, jelikož hráč nepostřehne chybu v detekci na bázi pixelů. Při fyzikálních simulacích vědeckého charakteru však je třeba maximální možné přesnosti. V této práci je snaha právě o přesnou detekci. Proto nelze využít přibližných metod, jako je transformace obalových těles.

Další možností, kterou využívá knihovna, která je předmětem této práce je výpočet přesného času kolize za pomocí rozkladu složitého geometrického problému na jednodušší. O konkrétních technikách pojednávají další kapitoly. Princip spočívá v nalezení času, kdy jedno těleso potenciálně koliduje s druhým tělesem v závislosti na rychlosti. Takto by měla být zaručena bezchybná detekce, která nevynechá žádný objekt. Tento postup je nezávislý na jakémkoliv vzorkování a řeší tak problém s diskrétním časem.

2.3 Implementace pohybu

Translační pohyb tělesa ve scéně definuje vektor rychlosti, který není konstantní z výše zmíněných příčin. Tento vektor udává o kolik se změní souřadnice těžiště tělesa (případně jiného referenčního bodu tělesa) za jednotku času. Rotační pohyb může být definován vektorem úhlových rychlostí okolo každé souřadné osy. Čas je závislý na výstupu simulace, jelikož primárním cílem je, aby simulace v normálním stavu běžela s konstantní rychlostí vhodnou pro uživatele. Typicky je simulační čas napodobením reálného času.

2.3.1 Geometrické transformace

Při pohybu těžiště pevného tělesa dochází k transformaci všech ostatních bodů tělesa o stejný vektor vzdálenosti pohybu. Tato změna polohy se řeší vynásobením všech vertexů (bodů) modelu objektu transformační maticí. Stejný postup funguje pro translaci i rotaci objektu.

Každý vertex modelu je definován homogenními souřadnicemi. V trojrozměrném prostoru se jedná o čtveřici souřadnic podle souřadných os (2.7). Čtvrtá souřadnice, váha bodu je u afinních transformací vždy definována u bodu jako hodnota 1, u vektoru jako hodnota 0 a nemá výrazný vliv na výpočty.

$$[x, y, z, w] \tag{2.7}$$

V trojrozměrném prostoru se standardně používají transformační matice 4x4. Výhodou matic je možnost skládání transformací do matice jedné. Toto skládání se provede vynásobením jednotlivých transformačních matic zprava. Záleží však na pořadí při násobení. Pro účely této práce je nutné znát zejména matici translační a rotační. Pro sestavení translační matice je nutné znát vektor

posunutí objektu. Vynásobením bodu a translační matice dojde k jednoduchému přičtení souřadnic translačního vektoru k souřadnicím bodu.

$$\vec{T}_v(d_x d_y d_z); T_m = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix} \tag{2.8}$$

Pro sestavení matice rotační je nutné znát úhel, o který se objekt má otočit a také osu, okolo které je otáčen. Pro rotace ve 3D je možné použít tři rozdílné matice, jednu pro každou osu otáčení (2.9). Vynásobením bodu s rotační maticí je provedena rotace bodu okolo počátku souřadného systému. Pokud však je cílem rotace objektu okolo svého středu, je třeba provést dodatečné transformace. Objekt musí být přesunut do počátku souřadnic, zde je otočen a posléze vrácen zpět na svou původní pozici. Rotaci okolo libovolné osy popisuje také tzv. Rodrigues' rotation formula [2].

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

$$R_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

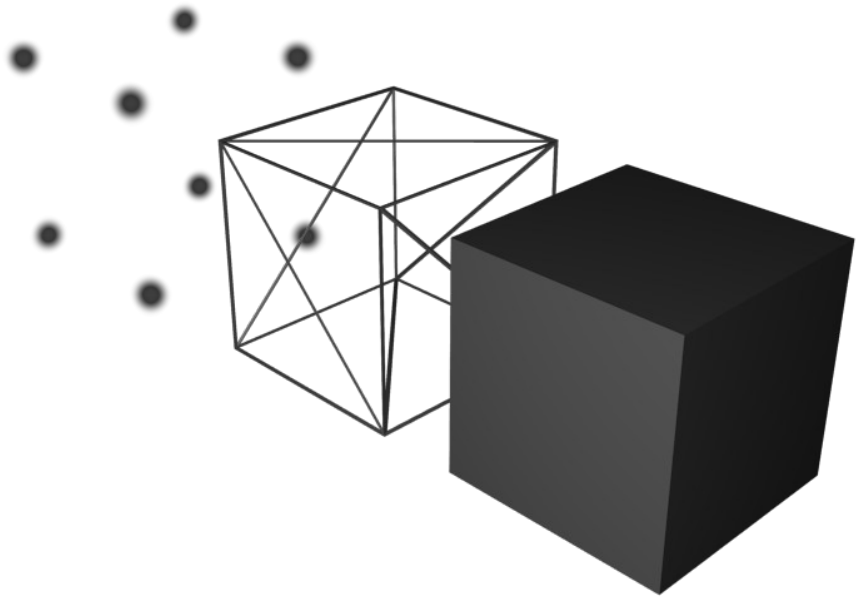
3 Detekce kolizí

Tato kapitola se věnuje popisu běžně používaných technik pro detekci kolizí a způsobům její akcelerace a optimalizací. Pro účely práce je důležitá zejména kolize ve 3D prostoru. Složitost detekce narůstá s komplexností modelů, ale vychází vždy ze stejných principů, jako při detekci kolizí jednoduchých modelů, nejčastěji základních geometrických objektů (kvádr, krychle, koule, jehlan, válec). Většina technik pro 3D prostor je pouhým zobecněním detekcí z 2D či 1D prostorů.

3.1 Reprezentace objektu

Objekt v simulaci je reprezentován množinou fyzikálních vlastností jako je hmotnost, materiálové konstanty, či pohybové vektory. Dále musí objekt obsahovat 3D model, který slouží k vykreslování objektu a také popisuje tvar objektu. Je přípustné a někdy i nutné, aby objekt obsahoval i více modelů.

V předchozí kapitole byla již naznačena struktura pro popis 3D modelu. Model je množina bodů (vertexů), majících mezi sebou předem definované vazby, které sdružují tyto body do polygonů. Mezi nejčastější polygony patří čtverce a trojúhelníky, přičemž trojúhelníky převládají díky podpoře grafického hardware. Tyto polygony tak tvoří povrchovou reprezentaci objektu. Na obrázku 3.1 je znázorněna reprezentace objektu na úrovni bodů, polygonů a materiálu. Na tento povrch je poté možno aplikovat barvy, textury, materiály a jiné vlastnosti pro napodobení reálných objektů.



Obrázek 3.1: Znázornění úrovní reprezentace objektů (vertex, polygon, material)

3.2 Kolize

Obecná detekce kolizí není triviální úlohou. Při popisu teorie detekce kolizí komplexních útvarů ve vícerozměrných prostorech je nutné vycházet z jednodušších problémů a následných zobecňováním přenášet tyto postupy výše. V následujících kapitolách jsou popsány běžně užívané metody detekce kolizí s jejich výhodami a nevýhodami.

3.2.1 Triviální úlohy

S využitím základních geometrických vztahů lze detekovat kolize jednoduchých objektů. U složitějších by tento postup byl možný také. Existují dva důvody proč se tyto techniky neaplikují na komplexní modely. Prvním je vysoká náročnost na výpočetní výkon, což je nevhodné pro simulace v reálném čase. Druhým důvodem je nutnost převedení bodové reprezentace modelů do soustav rovnic, což by vyústilo v další zpomalení simulace a složitou implementaci. Principy těchto technik však jsou důležitým prvkem pro pokročilejší mechanismy.

V předchozí kapitole je popsán problém tunelování. Řešení tohoto problému vyžaduje, aby byl problém řešen v závislosti na pohybu objektů. Nestačí tedy pouhé statické testy podle aktuálních poloh objektů, ale testy dynamické s předvídáním kolizí. Následující příklady detekce kolizí jsou proto vždy uvedeny se statickým i dynamickým řešením. Jedná se pouze o výčet několika možných situací pro demonstraci složitosti.

- **Body**
  Nejjednodušší úlohou je kolize dvou bodů. Jednoduchou úvahou lze dojít k závěru, že dva body kolidují, pokud se rovnají souřadnice jejich polohy v daném souřadném systému. V případě pohybu bodů by byly jejich trajektorie vyjádřeny rovnicemi (přímka u přímočarého pohybu) a nalezl by se průsečík těchto trajektorií. Podle rychlosti bodu je možné vypočítat čas, který by bod urazil na své cestě do průsečíku. V případě, že by oba body dorazily do průsečíku svých trajektorií ve stejném čase, dochází k jejich kolizi.
- **Úsečky**
  Nejprve je nutné vyjádřit obecné rovnice přímek obou úseček. Z těch lze dosazením spočítat průsečík. Pokud tento průsečík leží na obou úsečkách zároveň, existuje kolize. Pokud jsou úsečky v pohybu, postupujeme stejně jako u bodu. Pro každou úsečku je nalezen časový interval po který protíná přímku úsečky druhé a pokud se oba intervaly překrývají, existuje kolize.
- **Kružnice**
  Kružnice je definována bodem středu a poloměrem. Všechny body kružnice jsou umístěny ve vzdálenosti poloměru od středu. Pokud je tedy vzdálenost dvou středů kružnic menší, nebo rovna, součtu jejich poloměrů, dochází ke kolizi. V pohybu je situace podobná, pouze se namísto vzdálenosti středů hledá nejmenší vzdálenost trajektorií pohybu těchto středů.
- **Osově orientované čtverce**
  Tento princip je často využíván pro obalová tělesa, o kterých pojednávají další kapitoly. V tomto postupu jsou uvažovány čtverce, jejichž strany jsou vždy rovnoběžné se souřadnými osami. Pokud je například spodní strana prvního čtverce výše, než horní strana druhého

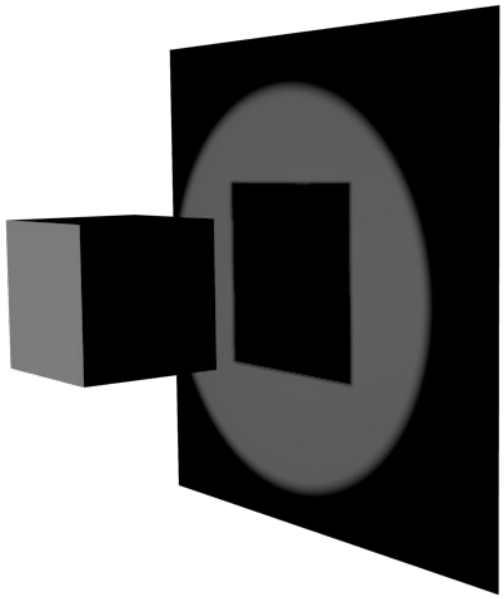
čtverce, nedojde ke kolizi. Podle souřadnic bodů lze určit kolizi složenou podmínkou testů pro všechny strany čtverce. V pohybu by se testovaly podobně intervaly, udávající změnu polohy bodů.

V případě dalších tvarů složitost značně narůstá. Pro základní geometrická primitiva existují funkční a používané algoritmy. Jedním z nich je například Möller algoritmus [4] pro rychlý výpočet kolize dvou trojúhelníků. Právě tyto postupy se dnes pomalu stávají spíše podpůrnými technikami, pro speciální případy hlavně v případě akcelerace výpočtů. Zejména v herním průmyslu byly hojně využívány díky možnosti zjednodušení modelů. Složité modely byly převedeny pro účely detekce kolizí na modely jednodušší, na které bylo možné aplikovat právě tyto postupy.

Vzhledem k rozvoji grafických aplikací, ať už z hlediska komplexnosti modelů či implementaci pokročilé fyziky vyplývá nutnost sjednotit detekci do jedné obecně využitelné techniky, která nebude závislá na jednom konkrétním tvaru či případném zjednodušování modelu.

3.2.2 Projekce

Důležitým pojmem pro další výklad je projekce. Jinými slovy lze definovat projekci jako převod objektu z n-rozměrného prostoru do n-1-rozměrného prostoru. Analogií projekce v reálném světě je vrhání stínu. Projekci ze 3D prostoru do 2D si lze představit jako vržený stín objektu na projekční plátno (viz Obrázek 3.2).



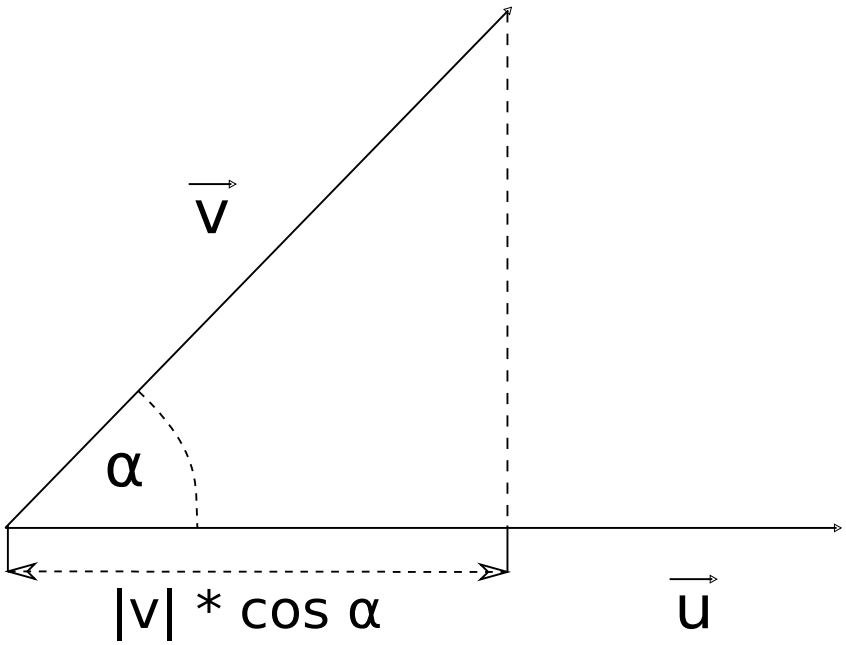
Obrázek 3.2: Vržený stín na projekční plátno

Pro využití v následujících technikách detekce kolizí je potřeba 3D projekce jednoho vektoru na jiný vektor. Stejný postup lze aplikovat i na 2D prostor. Klíčovou operací pro výpočet projekce je skalární součin (3.1). Skalární součin si lze také představit jako součin velikostí dvou vektorů s funkcí cosinus úhlu, který vzájemně svírají.

$$\vec{u}=(u_1,u_2,u_3),\vec{v}=(v_1,v_2,v_3)$$
$$\vec{u}\cdot\vec{v}=u_1v_1+u_2v_2+u_3v_3=|\vec{u}|\cdot|\vec{v}|\cdot\cos\alpha \tag{3.1}$$

Z výše uvedených vztahů lze odvodit vztah pro cosinus úhlu mezi vektory (3.2). S pomocí funkce cosinus můžeme vypočítat velikost úsečky projektovaného vektoru, jako na obrázku 3.3.

$$\cos\alpha=\frac{\vec{u}\cdot\vec{v}}{|\vec{u}|\cdot|\vec{v}|} \tag{3.2}$$



Obrázek 3.3: Skalární součin a projekce vektorů

Nyní lze odvodit výsledný vztah pro projekci. Na konci výpočtu je třeba výsledek vynásobit normovaným vektorem, na který projektujeme. Normalizace vektoru se provede pouhým vydělením vektoru jeho velikostí.

$$proj_u v=\cos\alpha\cdot|\vec{v}|\cdot normal_u=\frac{\vec{u}\cdot\vec{v}}{|\vec{u}|\cdot|\vec{v}|}\cdot|\vec{v}|\frac{\vec{u}}{|\vec{u}|}=\frac{\vec{u}\cdot\vec{v}}{|\vec{u}|^2}\vec{u} \tag{3.3}$$

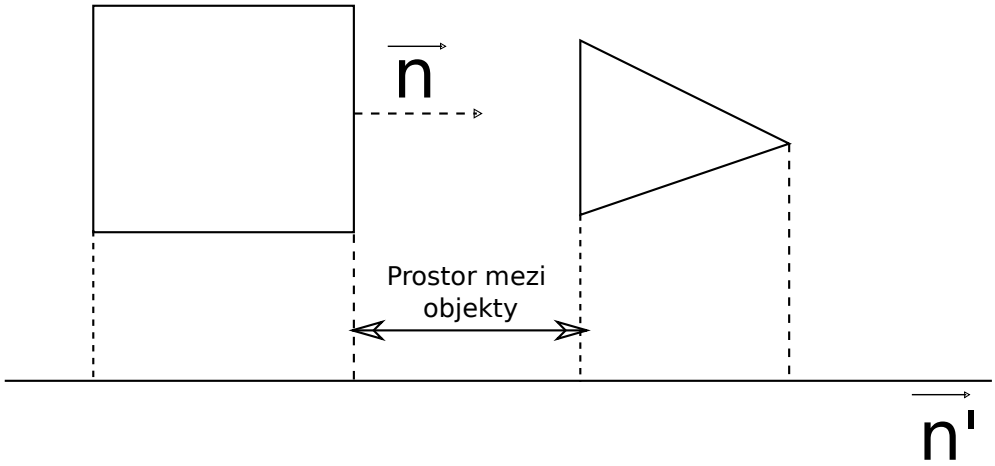
Tento postup není zcela optimální. Celá úloha se dá zjednodušit odvozením (3.4), kde ve výsledku stačí pouze vypočítat skalární součin těchto dvou vektorů, kde je vektor na který se projektuje normalizovaný.

$$\vec{u}\cdot\vec{v}=|\vec{u}|\cdot|\vec{v}|\cdot\cos\alpha$$
$$|\vec{v}|\cdot\cos\alpha=\frac{\vec{u}}{|\vec{u}|}\cdot\vec{v} \tag{3.4}$$

3.2.3 Separating Axis Theorem (SAT)

Separating Axis Theorem říká, že pokud existuje osa, na které nedochází k průniku projekcí dvou objektů, nenastává jejich kolize [5]. Tento teorém lze využít k přesné detekci kolizí v trojrozměrném prostoru, s pomocí výše zmíněné projekce. Situace je převedena pomyslně do dvojrozměrného prostoru a následně rozložena na několik triviálních úloh jednorozměrného prostoru.

Pro nalezení konkrétní osy, oddělující oba testované objekty je nutné provést projekce na všech možných osách. Cílem je nalézt mezeru mezi oběma objekty. Tuto mezeru je možné hledat s pomocí projekce objektu na obecnou osu. Při projekci dvou objektů na jednu osu vznikají dva intervaly, popisující prostor, který objekty zabírají vůči dané ose. Pokud se tyto intervaly nepřekrývají, je mezi objekty prostor a kolize nenastává, jako je znázorněno na obrázku 3.4. Pro otestování všech nutných projekcí objektů je vhodné zvolit jako testované osy všechny normály obou objektů.



Obrázek 3.4: Intervaly a oddělení objektů za použití SAT

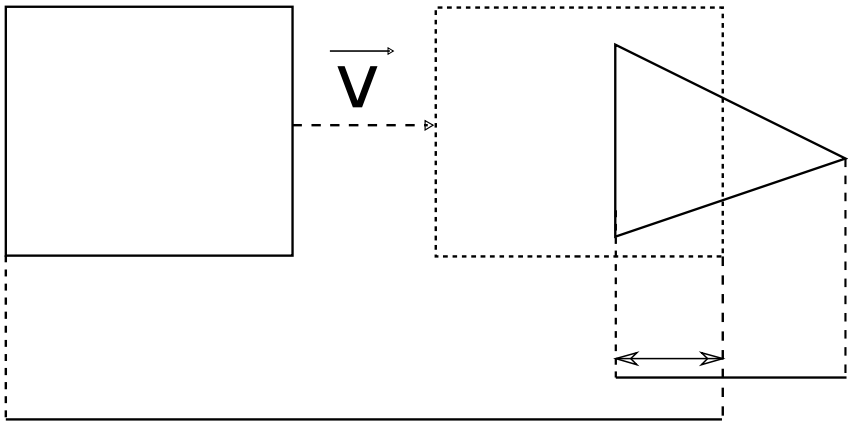
Potenciální kolize tedy nastává v případě, kdy neexistuje normála, na které se projekce obou objektů nepřekrývají. Projekce na normále jsou tvořeny s pomocí skalárního součinu, jak bylo popsáno v předchozí kapitole. Proces projekce probíhá nad všemi body tělesa a krajní body intervalu jsou maximum a minimum z projekcí bodů na danou osu.

Z definice SAT vyplývá, že tato metoda spolehlivě slouží k detekcím kolizí pouze u konvexních objektů. U konkávních objektů je nutné použít rozšiřující techniky. Jednou z nich je rozdělení konkávního objektu na několik konvexních. K zajištění maximální přesnosti této metody u složitých tvarů s hustou polygonální mřížkou by bylo však nutné rozdělit objekt na velké množství podobjektů. Druhou možností je nasadit SAT na jednotlivé trojúhelníky. Proces probíhá stejně. S pomocí SAT jsou testovány všechny trojúhelníky z prvního objektu se všemi trojúhelníky z objektu druhého. Aby došlo ke kolizi objektů, musí kolidovat minimálně jedna dvojice trojúhelníků. Tento proces také umožní přesně definovat místo kolize s přesností trojúhelníků. Pro zpřesnění lze ještě spočítat například analyticky průsečnici či průsečík obou trojúhelníků.

Další výhodou SAT je možnost aplikace této metody na pohybující se objekty a zjištění přesného času a místa kolize. Pohyb objektu je definován vektorem rychlosti. Projekcí vektoru rychlosti na testovanou osu SAT je získána skalární hodnota relativní rychlosti s jakou se pohybuje

interval na dané ose. V případě pohybu obou objektů proti sobě existují dva intervaly na testované ose, blížící se k sobě v čase.

Při vykreslování s určitým počtem snímků za sekundu se objekt posune o vzdálenost ve směru vektoru rychlosti rovnu velikosti součinu rychlosti a času mezi vykreslením dvou snímků. Před vykreslením dalšího snímku je známa pozice objektu ve snímku aktuálně viditelném a pozice ve snímku následujícím. Z těchto pozicí lze určit, jakou část testované osy objekt při pohybu zabere. Pokud se protínají intervaly obou objektů, dochází ke kolizi, jako jen znázorněno na obrázku 3.5.



Obrázek 3.5: Překrytí pohybových intervalů

Pokud mají oba projektované intervaly určitou rychlost, postupuje se při výpočtu času kolize tak, že se jeden z nich zvolí jako stacionární. Rychlost pohybujícího se pak rovná rozdílu obou rychlostí (nutnost zachovat znaménka s ohledem na směry). Vzdálenost obou intervalů je známa, stačí jen spočítat čas, za který ji pohybující se interval urazí (3.5). Každá kolidující dvojice intervalů, tedy každý osa (normála) bude mít svůj čas kolize. Výsledný čas kolize lze získat porovnáním všech těchto časů a nalezením takového časového intervalu, ve kterém dochází ke kolizi na všech osách. Časem kolize je pak levý (uvažována klasická časová osa rostoucí zleva doprava) krajní bod tohoto výsledného časového intervalu. Při výpočtu lze uvažovat jen pohyb krajních bodů intervalů.

$$s = v \cdot t \tag{3.5}$$

Může nastat i případ nechtěného průniku objektů do sebe. SAT tento případ detekuje a vrací navíc míru překrytí obou objektů. Tato hodnota je nejmenší velikostí překrytí projektovaných intervalů ze všech překrytí na všech testovaných osách. Tato míra překrytí společně s testovanou osou se nazývá Minimum Translation Vector (MTV) a může být použita pro vysunutí objektů z kolize.

3.3 Akcelerační metody

Ideálním případem pro detekci kolizí je detekce v rámci dvou těles. Na takovou situaci lze aplikovat výše popsané techniky přímo. Ve většině případů však je potřeba použít detekci kolizí pro scénu s větším množstvím různých objektů. Testování každého objektu s každým by v takových případech bylo vysoce náročné na výpočetní výkon a nedovolovalo by aplikaci provádět další nutné výpočty pro plynulý běh simulace. Proto je nezbytné využívat optimalizační metody pro zrychlení a eliminaci zbytečných výpočtů.

3.3.1 Obalová tělesa

Často využívanou technikou pro eliminaci zbytečných kontrol jsou obalová tělesa. Jedná se o jednoduchá geometrická tělesa (nejčastěji válec, krychle, koule), ohraničující celý objekt jako je znázorněno na obrázku 3.6. Aplikace testuje ve výsledku jen ty objekty, u kterých nastane kolize jejich obalových těles. Obalové těleso musí být výrazně jednodušší, než objekt, který obaluje. Tak jsou provedeny méně náročné a optimalizované detekce, které zabrání detekcím složitějším u objektů, které jsou od sebe příliš daleko.

Obrázek 3.6: Základní typy obalových těles

Tvar obalového tělesa se často volí podle tvaru obalovaného tělesa tak, aby obálka kopírovala svůj obsah co nejtěsněji. Mezi nejčastěji využívané tělesa patří krychle či kvádry, tak zvané bounding boxy. Výhodou je jejich dobrá přizpůsobivost obecným modelům a jednoduchá detekce kolizí s pomocí SAT. Bounding boxy se vyskytují ve dvou provedeních:

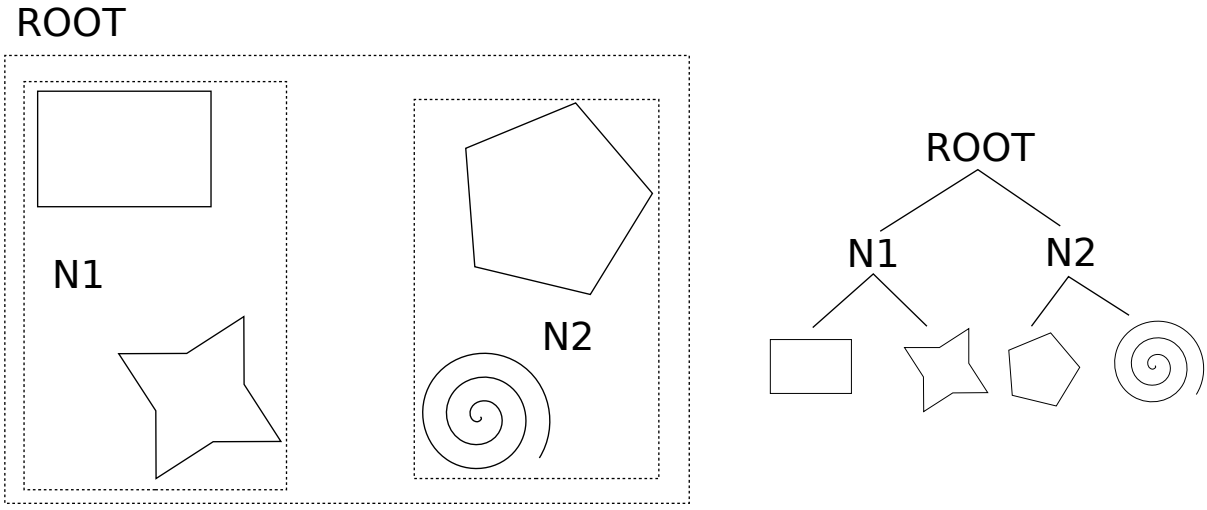
- Axis-aligned bounding box (AABB)
  Tyto boxy jsou orientovány podle souřadných os scény. Výhodou je snadná implementace detekce kolizí s pomocí SAT podle osy x, y a z. Nevýhodou je nutnost měnit jejich rozměry při každé rotaci vnitřního tělesa tak, aby box neustále těsně přiléhal k hranicím objektu.
- Oriented bounding box (OBB)
  Při rotaci objektu rotuje i jeho obalové těleso. Testování kolizí je však složitější.

3.3.2 Hierarchické struktury

Další technikou pro optimalizaci výpočtů je využití hierarchických struktur pro uložení objektů. Tato metoda vychází ze struktur pro reprezentaci scény. Aby bylo možné scénu vykreslit, v závislosti na polohách těles, využívá se pomocných struktur, kde jsou uloženy všechny objekty či odkazy na ně. Tak lze například vykreslit pouze objekty v bezprostřední blízkosti kamery. Této struktury lze využít také pro optimalizaci detekce kolizí.

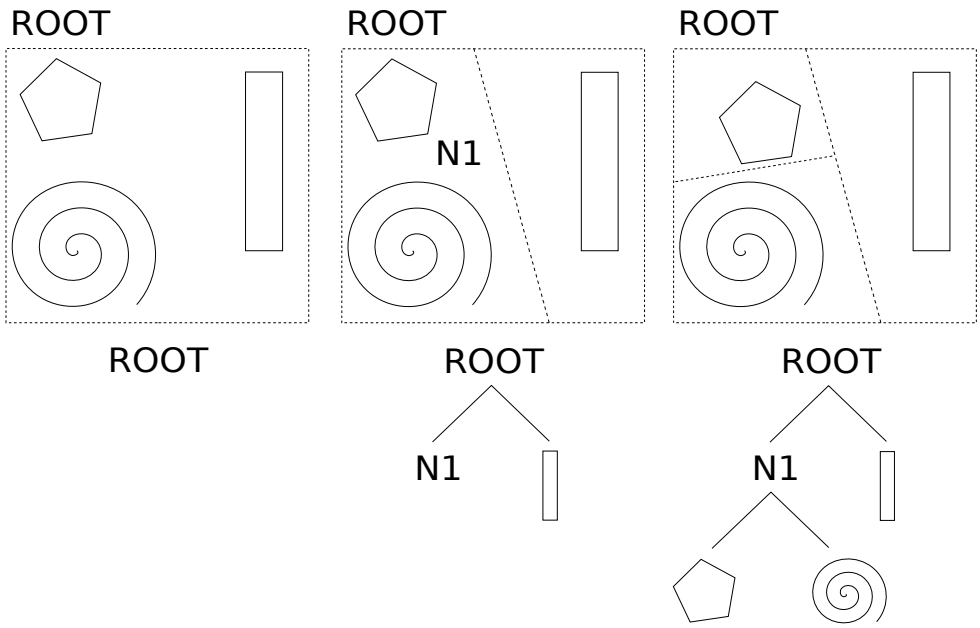
Nejčastěji se jedná o stromové struktury, kde v listech jsou uloženy jednotlivé objekty. Jelikož jsou tyto struktury sestaveny i následně aktualizovány v závislosti na umístění jednotlivých objektů, je možné předpokládat, že objekty budou uloženy i v samotné struktuře blízko sebe, například v jednom uzlu stromu. Tak tedy stačí detekovat kolize jen u objektů, které jsou blízko sebe, nebo které se při aktualizaci hierarchické struktury překrývají v jednom listu.

Bounding volume hierarchy (BVH) je nejjednodušší hierarchickou strukturou pro rozložení objektů ve scéně. Jedná se o stromovou strukturu obsahující obalová tělesa, standardně typu bounding box. Na nejvyšší úrovni, obsahuje kořenový uzel bounding box, který obsahuje celou scénu. Dále je rozdělen na určitý počet další bounding boxů, které rozdělují prostor a uzavírají do sebe skupiny objektů. Na nejnižší úrovni jsou bounding boxy pro jednotlivé objekty (obrázek 3.7). Tato struktura je jednoduchá na implementaci, rozdělení na podstromy však nemusí být vždy optimální. Při sestavování je třeba brát na zřetel rozložení scény, jelikož neexistují přesná pravidla pro rozdělení prostoru vůči počtům objektů. Také může dojít za běhu k nevalidním stavům, kdy objekt koluje mezi více bounding boxy [6].



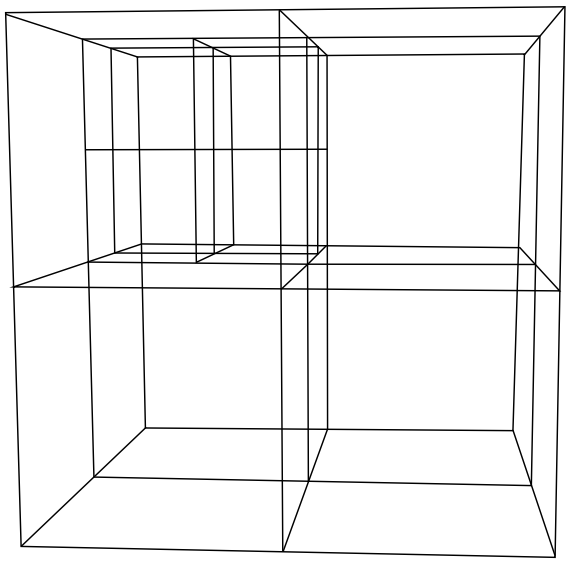
Obrázek 3.7: Bounding volume hierarchy – scéna/strom

Binary space partitioning (BSP) vytvoří BSP strom reprezentace scény, kdy rekurzivně dělí scénu na dvě části. Cílem dělení a ukončující podmínkou může být hustota dělení či dosažení stavu, kdy má každý objekt právě jeden list (obrázek 3.8). Sestavení tohoto stromu je však časově náročné, proto je vhodné pouze pro předdefinované scény či generátory scén [3].



Obrázek 3.8: Binary space partitioning – dělení prostoru

Octree je nejuniverzálnější a nejvíce používaná technika pro dělení prostoru scény. Jednotlivé varianty se využívají pro detekci kolizí, dělení geometrie objektů či vykreslování scény. Hlavním principem octree je pravidlo, že každý uzel má právě osm potomků. V listových uzlech jsou uloženy objekty. Jsou dvě možnosti implementace v případě, kdy objekt přesáhne jeden list. Objekt je uložen do více listů současně, nebo je uložen do vnitřního uzlu vyšší úrovně. V případě detekce kolizí je však výhodnější ukládat objekt vícenásobně. Detekce se pak provede pouze u listů, kde je objekt uložen namísto všech listů uzlu s objektem v nadřazené úrovni. V případě octree je možné si scénu představit jako prostor rozdělený do kostek, které se dále dělí vždy na osm dalších, jako na obrázku níže.



Obrázek 3.9: Octree – dělení prostoru

Kořenový uzel je pomyslný bounding box okolo celé scény a jeho podstromy zase bounding boxy okolo skupin objektů. Jedná se navíc o výše zmíněné osově zarovnané AABB. Každý uzel obsahuje souřadnice, které udávají střed obalové krychle v souřadnicovém systému celé scény. Vkládání pak probíhá například následovně:

1. Strom je prohledáván od kořene k listům
2. Pokud je v aktuálním uzlu místo pro objekt (objekt se vleze do jednoho z listů uzlu, který ohraničuje polohu objektu), je vložen
3. Pokud objekt zasahuje do plného listu, je list rozdělen na 8 a nový objekt i objekt původní jsou vloženy do tohoto nového podstromu
4. Pokud objekt zasahuje do více listů, je uložen do každého z nich

Dělení v bodě číslo 3 může být podmíněno faktem, zda nastává či nenastává kolize. Pokud nastane, tak je třeba ji vyřešit, pokud však nenastane, je možné dělit listy jelikož objekt lze jistě vložit do některého z listů, které vzniknou dělením. Nejčastější podmínkou dělení je však například zaplnění uzlu (počet objektů) či úroveň dělení.

Octree tedy zajistí testování objektu pro kolize pouze se svým nejbližším okolím. Pokud se objekt pohybuje, lze podle počáteční a koncové pozice objektu určit, zda se objekt přesune i do jiných uzlů stromu. Kolize je pak nutné testovat pouze pro listy podstromů, kterými objekt projde. Všechny buňky octree jsou AABB, tudíž je možné aplikovat SAT pro detekci kolize buňky ohraničující pohybující se objekt a ostatních buněk v prostoru. Tak vznikne seznam buněk, kterými pohybující se objekt projde a jejíchž obsah je nutné testovat.

3.4 Přesnost detekce

V různých situacích je požadována různá přesnost detekce kolizí. S využitím výše uvedených metod lze dosáhnout velmi přesné detekce vzhledem k času, kdy kolize nastane a také k místu v prostoru, kde ke kolizi dojde. Přílišná přesnost detekce nemusí být vždy žádaná. Čím více výpočtů je prováděno, tím méně času zbývá na ostatní.

V případě vědeckých simulací, které jsou zaměřeny výhradně na kolize je žádoucí zaměřit veškeré výpočty na kolize samotné, jelikož není třeba jiných výpočtů. Také tomu tak je v aplikacích, kdy nezáleží na běhu v reálném čase, ale záleží na přesnosti výsledku. Naopak například v herních enginech je třeba maximální optimalizace. Kolize proto nemusí být maximálně přesné, stačí pouze, aby hráč měl pocit reálného chování objektů.

Příliš přesná kolize může v herních aplikacích vyvolat i negativní odezvu uživatele. Například v situaci, kdy hráč vidí na obrazovce jasně kolizi, nemusí tato kolize ve skutečnosti proběhnout. Tento problém souvisí s nedokonalostí lidského vnímání. Rozdíl jednoho pixelu není v běžném užívání postřehnutelný lidským okem. Proto se někdy využívá předčasná detekce, kdy herní engine detekuje kolizi ještě před fyzickou kolizí modelů. Povrch modelu je vysunut dopředu a detekce tak probíhá dříve [10] . Situace se pak jeví uživateli jako více reálná.

4 Implementace

V předchozích kapitolách byl popsán teoretický základ pro implementaci knihovny detekce kolizí. Tato kapitola popisuje samotnou implementaci této práce, včetně popisu výběru vhodných postupů a metod a popisu struktury kódu. Následující podkapitoly se zabývají pouze podstatnými částmi kódu, přímo souvisejícími s jádrem řešení zadaného problému. Každý tématický celek (podkapitola) zabývající se implementací určité části projektu je zapouzdřen v jednom modulu, samostatném cpp souboru. Mimo níže zmíněné části obsahuje práce také podporu pro vykreslování v OpenGL, základní práci s shadery včetně jednoduchého osvětlení a texturování a využití SDL knihovny pro vytvoření multiplatformního okna a smyčky zpráv.

4.1 Implementační prostředky

Knihovna samotná, stejně jako demonstrační aplikace je naprogramována v jazyce C++. Vykreslování v demonstraci zajistí grafická knihovna OpenGL. Překlad zajišťuje kompilátor G++ za použití příkazu make pro připravený Makefile. Knihovna byla vyvíjena a primárně testována na 64-bitovém systému Debian GNU/Linux 8 (jessie).

Dále jsou využity následující dodatečné knihovny:

- SDL - pro zajištění multiplatformního vykreslení okna a zachytávání vstupů
- GLEW - pro inicializaci potřebné verze OpenGL
- GLM - která zajišťuje implementaci základních matematických operací pro 3D grafiku

4.2 Objekty

Objekty jsou načítány za běhu programu ze souborů uložených v příslušném adresáři. Pro účely práce je definován jednoduchý formát těchto souborů, popisující vlastnosti jednotlivých objektů ve scéně. Příklad takového souboru s komentáři je ukázán v kódu 4.1.

Jelikož knihovna pracuje pouze nad konvexními objekty, je nutné konkávní objekty definovat pomocí více modelů. V praxi se používá algoritmů pro dělené konkávních objektů na množinu konvexních podle dané tolerance. Pro účely této práce je v souboru objektu možnost explicitně definovat dílčí konvexní modely a jejich umístění v rámci složeného modelu.

Implementaci všech metod nad objektem zajišťuje třída *object*, která zároveň udržuje základní informace o objektu jako je poloha, orientace v prostoru, rychlost apod. Mezi nejdůležitější metody patří například metoda *load* zajišťující nahrání a inicializaci objektu, *setTransVelocity* a podobná metoda *setAngVelocity* zajišťující nastavení translační a rotační rychlosti, další metody typu *get* a *set* pro úpravu či čtení stavu objektu a v neposlední řadě metody pro transformaci objektu jako *transform* nastavující polohu, orientaci a velikost objektu a metoda *move* využívající interně metody *transform* pro posun objektu v závislosti na jeho rychlosti o daný čas.

Mimo jiné vlastnosti jako je již zmíněná cesta k souboru modelu či textuře objektu je důležité znát aktuální stav objektu definovaný vektorovými proměnnými pro translační rychlost, rotační rychlost a pozici. Dodatečné vlastnosti pro výpočty jsou v proměnných pro tvrdost objektu, třecí koeficient, pozici těžiště a hmotnost.

```
1  #komentáře uvozené znakem „#“ na začátku řádku
2  #hmotnost objektu
3  mass:1.2
4  #tvrdost objektu definuje míru odrazu při kolizi <0;1>
5  hardness:0.4
6  #tření udává zpomalení objektu při pohybu po povrchu <0;1>
7  friction:0.1
8  #těžiště objektu, souřadnice x,y,z
9  centerOfMass:0.1,0.5,10.
10 #udává zda je objekt pohyblivý (například zdi nejsou) 1/0
11 movable:1
12 #modely objektu ve formátu: soubor; textura; orientace; měřítko; pozice
13 model:data/models/box.obj;data/textures/box.bmp;0.0,0.0,0.0;1.0,1.0,1.0;0.0,0.0,0.0
14 model:data/models/box.obj;data/textures/box.bmp;0.0,0.0,0.0;1.0,1.0,1.0;2.0,0.0,0.0
```

Kód 4.1: Formát souboru objektu

4.3 Modely

Každý objekt může obsahovat jeden či více modelů, ze kterých se skládá. Tyto modely se pak v simulaci pohybují stejnou rychlostí a mají společné těžiště. Třída *model* implementuje podobné metody jako *object*, pouze na nižší úrovni. Například pro transformaci objektu volá metoda třídy *object* metody třídy *model* pro transformace nad jednotlivými modely.

Model je nahráván ze souboru typu Wavefront .obj. Jedná se o jednoduchý formát popisující 3D modely na bázi vertexů, normál, texturových souřadnic a polygonů. Pro účely zpracování tohoto souboru obsahuje tato práce speciální funkci ve třídě *model*. Po nahrání souboru vznikají pole obsahující všechny vertexy, normály a polygony modelu. Pole s vertexy a normálami je třeba zachovat, neboť poslouží pro výpočty projekcí u detekce kolizí. Následně jsou podle polygonů (v případě této práce trojúhelníků) vytvořeny potřebné datové struktury pro vykreslení v OpenGL.

4.4 Realizace detekce kolizí

Pro samotnou implementaci detekce kolizí je použita výše zmíněná metoda SAT. Výhodou této metody oproti jiným je rychlost a řešení problémů s diskrétním časem a pohybujícími se objekty. Tato metoda je vhodná zároveň i pro optimalizace výpočtů za pomocí obalových těles, jelikož její časová složitost roste se složitostí povrchové reprezentace těles.

Abychom mohli testovat kolize, je nutné provést podobné výpočty jako jsou prováděny ve vertex shaderu na GPU, tedy transformovat body a normály na aktuální polohu. To je prováděno aplikací transformačních matic. Nejprve je inicializována obecná transformační matice která je vynásobena maticí translační a následně třemi maticemi rotačními (podle všech tří souřadných os).

Touto maticí jsou vynásobeny souřadnice každého bodu modelu. Normály jsou transformovány podobně, jen jsou násobeny transponovanou, invertovanou transformační maticí. Jakmile jsou modely objektů nachystány v aktuálních pozicích, je spuštěna metoda implementující SAT nad každou dvojicí modelů z obou objektů. Vertexy a normály pro testování kolizí jsou uloženy v odlišných datových strukturách, než jsou struktury pro vykreslování. Normály jsou navíc normalizovány na jednotkovou délku.

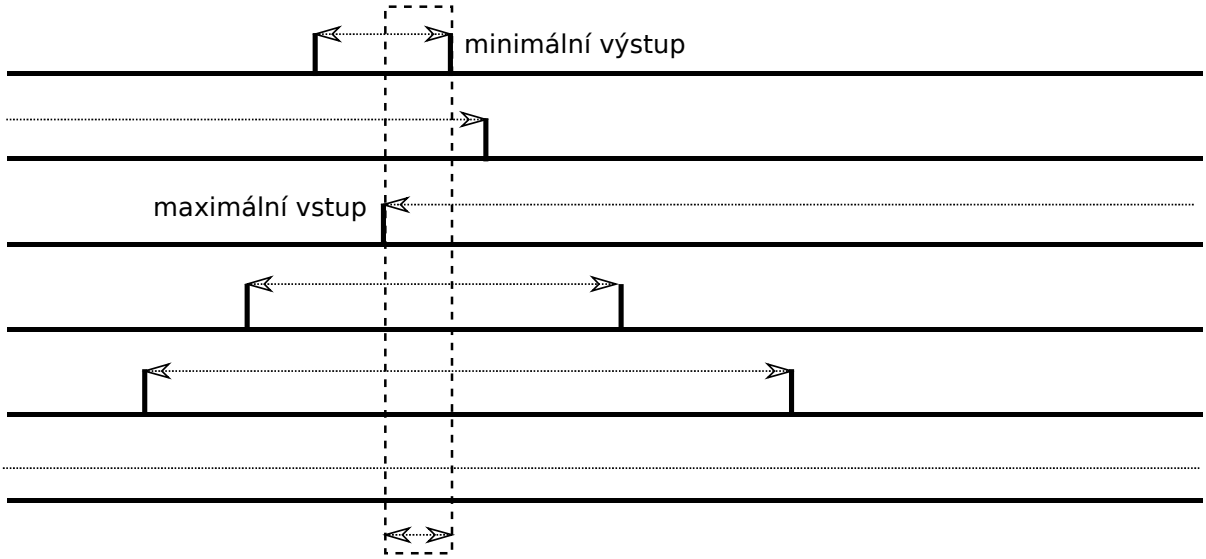
Jako testované osy slouží normály obou testovaných modelů. Tak je zajištěno testovaní všech orientací modelů, které má smysl testovat. Ke kolizi totiž dochází vždy buďto skrze dva vertexy, což je velmi vzácný scénář, nebo dvě hrany, nebo dva polygony. Testy na normálách odhalí všechny tyto případy s dostačující přesností. Pseudokód 4.2 popisuje strukturu metody implementující SAT.

```
1  foreach (všechny normály obou objektů)
2    relativníRychlostA = skalárníSoučin(rychlostA, aktuálníNormála)
3    relativníRychlostB = skalárníSoučin(rychlostB, aktuálníNormála)
4    //model A uvažován jako pohybující se a B jako statický
5    relativníRychlost = relativníRychlostA – relativníRychlostB
6    foreach (všechny vertexy A)
7      temp = skalárníSoučin(aktuálníVertex, aktuálníNormála)
8     //nalezení intervalů na normále
9      if (temp < intervalA[0])
10       intervalA[0] = temp
11     if (temp > intervalA[1])
12       intervalA[1] = temp
13   ….. //model B
14   if (intervalA+relativníRychlost protíná intervalB)
15     if (intervalA je nalevo od B)
16       //čas vstupu a výstupu do/z kolize
17       časVstupu = absHodnota(intervalB[0] – intervalA[1]/relativní rychlost)
18       časVýstupu = absHodnota(intervalB[1] – intervalA[0]/relativní rychlost)
19     …. //podobně když je A napravo od B
20     if (intervalA a B se protínají)
21       průnik = rozdíl krajních bodů podle polohy
22     ulož maxima a minima vstupů a výstupů
23 if (minimálníVýstup >= maximálníVstup)
24   nastává kolize, vrať uložená data (normála, časVstupu, průnik)
```

Kód 4.2: Pseudokód SAT

Takto je zajištěna správná detekce i v případě pohybujících se těles. Metoda předvídá kolizi a vrací čas, kdy ke kolizi dojde. V případě že kolize není detekována předem, vrací zase metoda míru průniku testovaných těles. Potřebná je zejména kolizní normála, která podle konvence směřuje k prvnímu z dvojice testovaných modelů a je kolmá ke kolizní ploše jednoho z objektů. Tato normála hraje důležitou roli v následující reakci těles na kolizi.

Jelikož musí být detekce kolizí poměrně rychlou operací, jsou v kódu navíc přidány některé optimalizační podmínky, například pokud výsledná rychlost indikuje pohyb intervalů od sebe, netestuje metoda dál, ale okamžitě hlásí absenci kolize. Stejně tak pokud dochází ke kolizi až v čase větším než je čas jednoho snímku je detekce zastavena. Časy vstupů a výstupů se ukládají, aby bylo možné detekovat že dochází ke kolizi na všech osách současně. Může totiž docházet k případům, kdy objekty kolidují na jedné ose pouze krátkou chvíli, zatímco na druhé například po celou dobu detekce. Problém ilustruje obrázek 4.1.



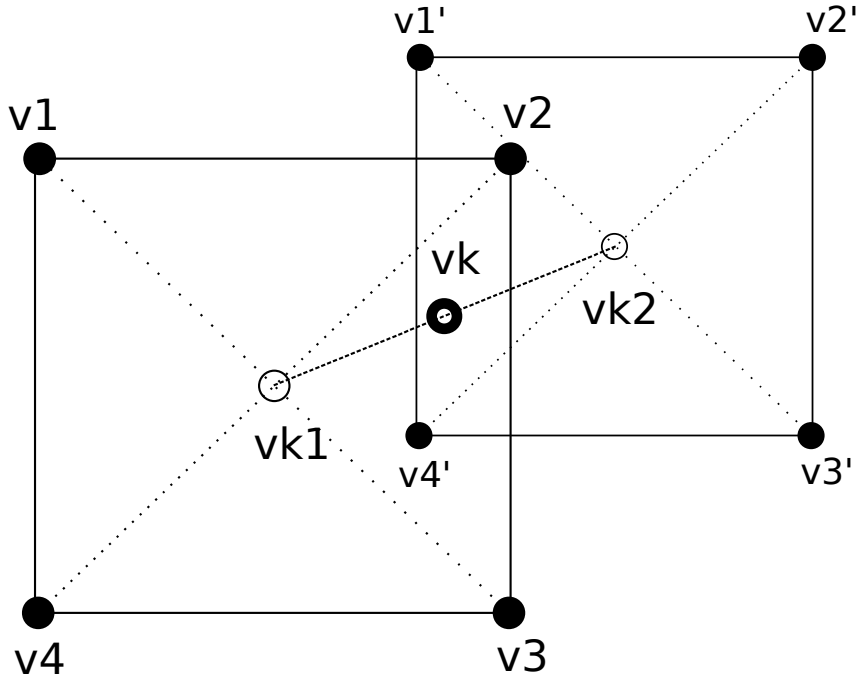
Obrázek 4.1: Časové intervaly po které objekty kolidují na všech testovaných osách (cílem je nalézt jejich nenulový interval průniku pro potvrzení kolize)

4.5 Určení kolizního bodu

Pro implementaci translačního pohybu by nebylo třeba nic dalšího. Simulace však pracuje i s pohybem rotačním, kde je třeba znát bod kolize pro výpočet výsledných sil. Metoda SAT přímo nevrací tento bod. Obecně se používá často přídavný algoritmus pro výpočty těchto bodů. V případě této práce je implementován jednoduchý odhad kolizního bodu pomocí vertexů polygonů podílejících se na kolizi.

Kolizní bod není určen zcela přesně. Výpočty však nejsou příliš složité a ve výsledku nelze nepřesnost kolizního bodu výrazně postřehnout. Vyhledání kolizního bodu zajišťuje samostatná funkce *GetContactPoints*. Při detekované kolizi dojde k nové projekci modelů na již zjištěnou kolizní normálu a naleznou se krajní body intervalů. Zároveň se ukládají vertexy, které byly na tyto krajní body projektovány. Dále se určí které krajní body budou kolidovat a z jejich vertexů se vypočítá průměr. Funkce tak vrací dva samostatné lokální kontaktní body, pro každý model jeden.

Pro fyzikální výpočty je možné použít buďto každý bod zvlášť pro daný objekt, nebo ještě dopočítat průměr těchto dvou bodů a tento výsledek vzít jako kontaktní bod. Situace pro dvě čtvercové kontaktní plochy je popsána na obrázku 4.2.



Obrázek 4.2: Přibližné určení kontaktního bodu

v1, v2, v3, v4 – vertexy kontaktní plochy prvního modelu

v1', v2', v3', v4' – vertexy kontaktní plochy druhého modelu

vk1 – střed kontaktní plochy prvního modelu

vk2 – střed kontaktní plochy druhého modelu

vk – výsledný kontaktní bod

Pro přesné určení kontaktních ploch by bylo nutné nejprve určit polygony definující tyto plochy. Pomocí vertexů podílejících se na kolizi (vertexy promítnuté na krajní body intervalů v SAT) lze vytvořit 2D konvexní obálky popisující obě kontaktní plochy. K tomu lze použít convex hull [7] algoritmů. Následně je nutné nalézt průnik těchto dvou ploch například pomocí ořezávacích algoritmů jako je Sutherland-Hodgman [8]. Tímto způsobem je definována skutečná kontaktní plocha pro danou kolizi. Bod kolize lze následně přesně určit jako střed této plochy.

4.6 Fyzika a reakce na kolize

Fyzikální modul zajišťuje základní fyzikální výpočty zahrnující především reakce na vzniklé kolize a aplikaci okolních sil prostředí. Výsledná rychlost kolidujících objektů je vypočítána pomocí impulsů síly. Impulsy zajišťují správnou orientaci i velikost výsledné síly a zajišťují jednotný výpočet pro translační i rotační pohyb. Opakované kolize a aplikace impulsů zajišťují také ustálení objektů za určitý čas nečinnosti. Odvození následujícího vzorce není zcela triviální [12], proto je zde uvedena pouze konečná forma [13], přímo implementovaná v kódu (4.1). Vektory použité ve vzorci jsou znázorněny na obrázku 4.3.

$$I = \frac{-(1+e)v_{AB}\cdot n}{n\cdot n\left(\frac{1}{M_A}+\frac{1}{M_B}\right)+\left(\left(J_A^{-1}(r_A\times n)\right)\times r_A+\left(J_B^{-1}(r_B\times n)\right)\times r_B\right)\cdot n} \tag{4.1}$$

Vysvětlivky ke vzorci (4.1):

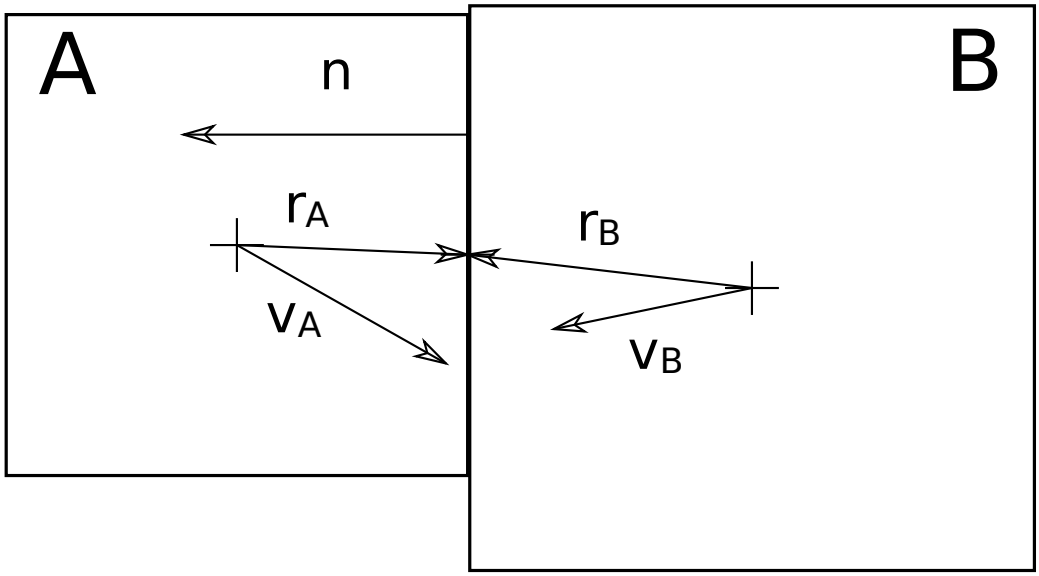
- I – výsledný impuls
- e – koeficient restituce
  (udává zmenšení výsledné rychlosti v závislosti na materiálu těles)
- v_{AB} – relativní rychlost objektů $V_A - V_B$ (výpočet relativní rychlosti popisuje (4.2))
- n – kontaktní normála směřující k objektu A
- M_A, M_B – hmotnosti těles
- r_A, r_B – vektory směřující z těžiště těles do kontaktního bodu
- J_A, J_B – momenty setrvačnosti těles

Relativní rychlost těles je nutno vypočítat nejen v závislosti na rychlosti translační, ale i rychlosti rotační. Celková rychlost tělesa je počítána následovně:

$$v = vt + vr \times r \tag{4.2}$$

Vysvětlivky ke vzorci (4.2):

- v – celková rychlost tělesa
- vt – translační rychlost tělesa
- vr – rotační (úhlová) rychlost tělesa
- r – vektor směřující z těžiště tělesa do kontaktního bodu



Obrázek 4.3: Stav objektů v kolizi, vektory pro výpočet impulsu

Takto vypočítaný impuls je použit pro výpočty nových rychlostí podle následujících vzorců.

$$\begin{aligned} Vt_A &= Vt_A + \frac{I}{M_A} n \\ Vt_B &= Vt_B - \frac{I}{M_B} n \\ Vr_A &= Vr_A + \frac{1}{J_A}(r_A \times (nI)) \\ Vr_B &= Vr_B - \frac{1}{J_B}(r_B \times (nI)) \end{aligned} \tag{4.3}$$

Vysvětlivky ke vzorcům (4.3):

- Vt_A, Vt_B – výsledné translační rychlosti
- Vr_A, Vr_B – výsledné rotační rychlosti
- I – vypočítaný impuls
- zbývající proměnné stejné jako u (4.1)

Tření při vzájemném pohybu těles po povrchu je vypočítáno jen zjednodušeně za použití koeficientu tření násobeného konstantou z intervalu <0;1>. Tento postup není zcela přesný, avšak je schopný zajistit tlumení pohybu tělesa při smykovém tření. Velkou nevýhodou je ignorace valivého odporu a tedy absence vzniku valivého pohybu. Tento pohyb alespoň částečně nastává díky impulsům

a jejich aplikaci při kolizi. Aplikace tření a jeho vliv na výslednou rychlost je popsán následující rovnicí:

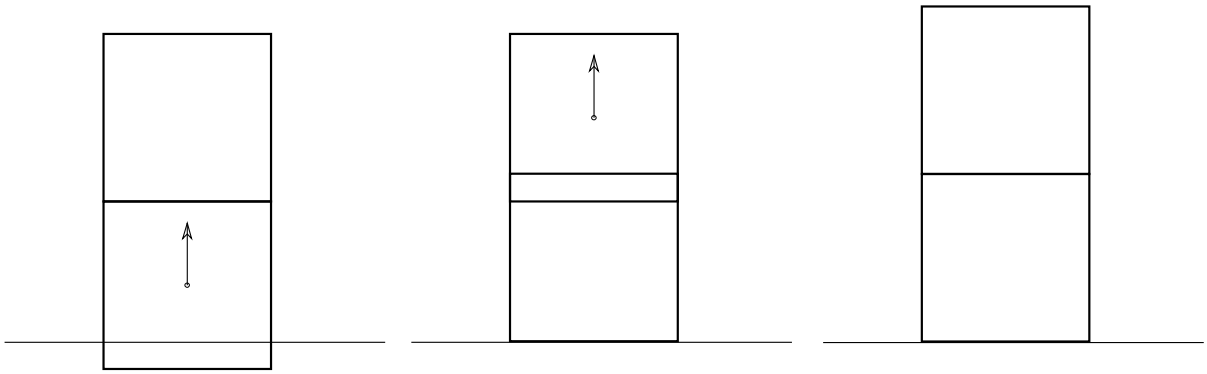
$$\begin{aligned} Vt_x &= Vt_x - (1 - |n_x| Vt_x e\, c) \\ Vt_y &= Vt_y - (1 - |n_y| Vt_y e\, c) \\ Vt_z &= Vt_z - (1 - |n_z| Vt_z e\, c) \end{aligned} \tag{4.4}$$

Vysvětlivky ve vzorci (4.4):

- Vt_x, Vt_y, Vt_z – relativní translační rychlosti na dané ose
- n_x, n_y, n_z – jednotlivé složky vektoru normály
- e – koeficient restituce
- c – třecí konstanta

Někdy nastává speciální případ, kdy není detekována kolize předem, ale objekty již jsou zaklíněné v sobě. Tento stav nastává především u velmi malých rychlostí, kdy dochází k nepřesným výpočtům u čísel typu float. Další příčinou je nesprávné pořadí výpočtů reakcí na kolize ve stejném čase. Tento stav také způsobují rotace u objektů v blízkosti jiných objektů. SAT nemá mechanismy pro předvídání rotačních změn a jejich vlivu na projektované intervaly. Prioritou při řešení tohoto stavu je v první řadě přesun objektů ven z kolize.

Nejjednodušším řešením je návrat objektů na předchozí pozici (kde nenastala kolize), zastavení rotačního pohybu a obrácení směru rychlosti tělesa podle kolizní normály. Rychlost je také vynásobena zpomalující konstantou (hodnota přibližně v rozsahu <0.9;0.99>). Vrácení objektu na předchozí pozici zajistí vysunutí i více objektů na sobě, jako je znázorněno na obrázku 4.4. Nevýhodou toho přístupu je poměrně velká pravděpodobnost výskytu situace, kdy objekt osciluje okolo dvou poloh. Takto zacyklený objekt se vrátí na předchozí pozici z průniku, ale má tendenci stále konat ten stejný pohyb zpět do průniku. Objekt se tak pohybuje stále tam a zpět.

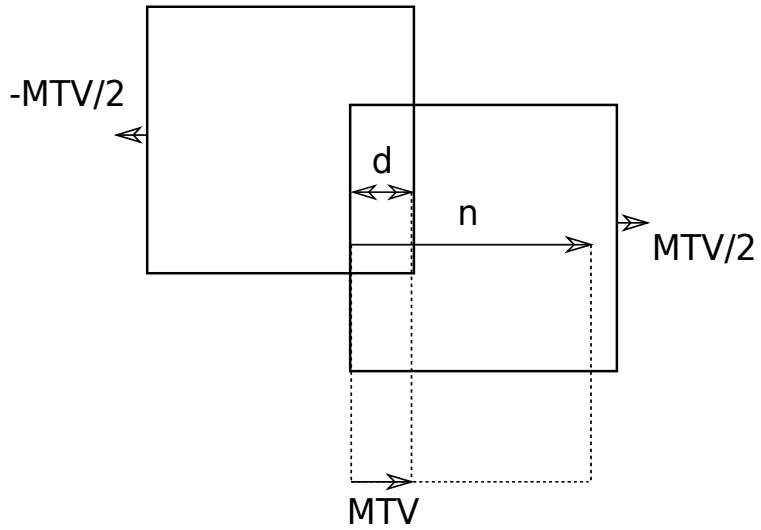


Obrázek 4.4: Vysouvání objektů na bázi návratu na předchozí pozici

Nejprve je navrácen spodní objekt, ten však zasahuje do vrchního, proto je vrácen i ten.

Objekty lze vysouvat i za pomocí MTV z techniky SAT. V případě velké hromady objektů v těsné blízkosti může však docházet k nechtěným explozím. Pro tyto případy existují pokročilé

algoritmy na základě vyhledávání cest [14]. Pro většinu případů však stačí jednoduchý přístup vysouvání vzhledem k pozicím právě kontrolovaných objektů. MTV vektor definuje směr a velikost průniku modelů. Aby se objekty dostaly opět ven z kolize, je k pozici každého z nich přičtena či od této pozice odečtena polovina MTV. Zda je MTV přičten či odečten rozhoduje orientace normály vzhledem k tělesům. V případě kolize pohybujícího se objektu s explicitně statickým (nelze rozpohybovat, například podlaha) je přičten celý MTV k pozici pohybujícího se objektu. Případ dvou pohybujících se objektů znázorňuje obrázek 4.5.



Obrázek 4.5: Vysunutí objektů na bázi aplikace MTV

Rozšířením této metody je mechanismus používající MTV na základě rozdělení tohoto vektoru v poměru rychlostí obou objektů. Každý objekt se vysune podle rychlosti, se kterou do kolize vstoupil. V případě že jsou tyto rychlosti nulové, tak lze využít konstantního posunu podle výše zmíněné metody půlení MTV. V této práci je využit princip půlení MTV bez dělení podle rychlostí. Experimenty ukázaly, že u dělení podle rychlostí dochází k větší chybovosti díky nepřesnosti výpočtů u malých čísel typu float.

Samotné vysunutí objektů však často nestačí, jelikož objekty mohou oscilovat neustále okolo této kolizní pozice. V implementaci je zahrnut jednoduchý mechanismus, který ošetřuje tyto stavy průniků. Ihned po vysunutí je upravena rotační rychlost objektů (nejčastější důvod průniků). Tato úprava spočívá ve vynulování rotační rychlosti či jejím zmenšení na velikost blízkou nule. Nulová rychlost někdy způsobuje příliš dlouhé balancování objektu na hraně. Podle druhu objektů nastávají dva možné scénáře a jejich řešení.

Prvním případem je kolize pohybujícího se objektu s pevným. Aby objekt jistě směřoval ven z kolize, je vektor rychlosti jednoduše reflektován za pomocí kolizní normály opačným směrem (4.5). V případě dvou pohybujících se objektů dojde ke zpomalení rychlejšího objektu, který vnikl do objektu pomalejšího a předání energie původně pomalejšímu objektu. K tomu slouží vzorec pro pružný ráz (2.3). Pravidla pro změny rychlostí těles jsou v kódu definována explicitně a je tak přerušeno zpracování rychlostí pomocí impulsů. Jedná se o kritický moment, kdy by mohla reakce za účasti impulsů selhat například díky velmi malým rychlostem.

$$\vec{v} = \vec{v} - 2(\vec{v} \cdot n)n \tag{4.5}$$

4.7 Optimalizace a reprezentace scény

Pro reprezentaci scény je použita datová stromová struktura octree. Octree slouží zároveň k optimalizaci výpočtů při samotné detekci kolizí. Existuje velké množství variant této populární datové struktury, lišících se v postupu při vkládání, vyhledávání a samotném rozdělení uzlů.

4.7.1 Vkládání objektů

V této práci je použit octree s dvojitým omezením do hloubky. Prvním omezením je počet objektů v jednom uzlu a druhým omezením je maximální úroveň dělení. Při vkládání tedy dojde ke kontrole, zda v uzlu není více objektů než je povoleno. Pokud jejich počet přesahuje limit, dochází k další kontrole, tentokrát na hloubku dělení. Pokud je aktuální uzel na úrovni přesahující limit dělení, vloží se objekt i přes omezení maximálního počtu objektů. V opačném případě dochází k dělení uzlu na osm potomků a novému vkládání. Případ rozdělení plného uzlu a nového vkládání je jedna z nejnáročnějších operací nad stromem, proto je nutné nastavit počáteční podmínky maximální úrovně dělení a maximálního počtu objektů vhodně vzhledem k předpokládanému rozložení scény.

Velké objekty přesahující více uzlů jsou automaticky vkládány do více listových uzlů zároveň. Při vkládání objektů je nutné vyhledat patřičné uzly octree, do kterých má být objekt vložen. Jedná se znova o detekci kolizí, jelikož jednotlivé uzly octree jsou osově zarovnané obalové krychle (AABB). Z tohoto faktu lze odvodit, že detekce nebude tak složitá, jako u obecných modelů. Aby bylo možné provést rychlou detekci, pracuje algoritmus s AABB vkládaného objektu, namísto polygonální reprezentace objektu samotného. Pro test AABB kolizí lze použít optimalizovanou verzi SAT, která v tomto případě testuje pouze na třech souřadných osách. Algoritmus pracuje rekurzivně se stromovou strukturou od kořene k listům.

Problémem mohou být pohybující se objekty. Jejich pohyb způsobuje neustálou potřebu přestavování stromu, jelikož objekty mohou při svém pohybu zaujímat místo v mnoha různých uzlech a z jiných mohou zase mizet. Pohybující se objekty jsou proto ukládány na speciální zásobník, přítomný ve třídě *octree*, avšak nezávislý na stromové struktuře a uzlech. Pseudokód 4.3 popisuje situaci vkládání a je shrnutím všech výše zmíněných pravidel:

```
1  if (objekt->pohybujícíSe)
2    zásobníkPohybujících->push(objekt)
3    return
4  if (uzel->máPotomky)
5    foreach (potomek)
6      if (aabbKolize(objekt->aabb, uzel->aabb))
7        vložit (objekt, potomek) //rekurzivní volání metody tohoto kódu
8  else
9    if ((uzel->početObjektů<maximálníPočet) or (uzel->úroveň>=maximálníÚroveň))
10     uzel->přidej(objekt)
11   else
12     uzel->rozděl()
13     foreach (původníObjekt)
14       vložit (původníObjekt, uzel) //původní objekty z uzlu před rozdělením
15     vložit (objekt, uzel)
```

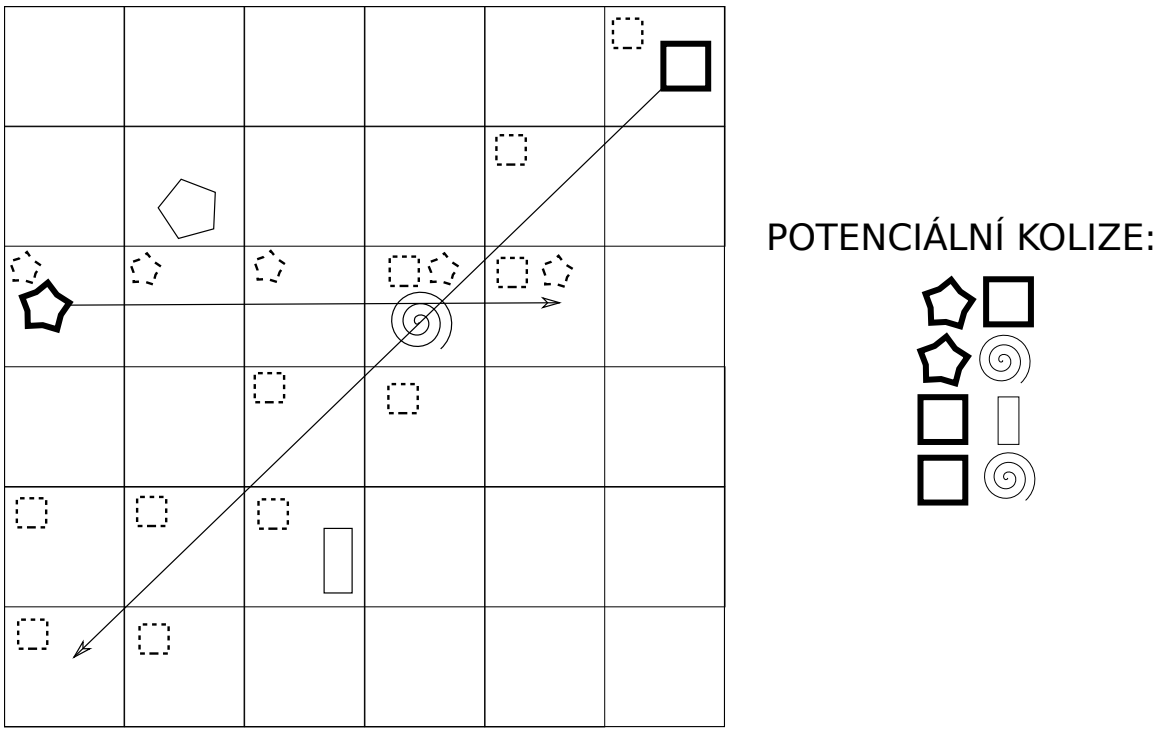
Kód 4.3: Vkládání do octree

4.7.2 Výběr potenciálních kolizních dvojic

Aby bylo možné použít octree pro optimalizaci detekce kolizí, je nutné použít algoritmus prohledávání stromu vzhledem k fyzickému umístění objektů ve scéně. Usnadnění situace přináší fakt, že pohybující se objekty jsou odděleny od stromové struktury uzlů.

Algoritmus prochází postupně všechny pohybující se objekty a provede podobný průchod, jako při vkládání objektů do stromu. Za pomocí AABB objektu lze získat všechny uzly, které objekt v jednom časovém úseku projde. Je tak nalezena pomyslná cesta stromem, kterou objekt prochází. Implementačně je to zajištěno pomocí ukládání odkazu na objekt na speciální zásobník v uzlu. Tento zásobník obsahuje po otestování všech pohybujících se objektů odkazy na objekty, které tímto uzlem prochází. Jinými slovy lze říct, že objekt zanechá v každém navštíveném uzlu stopu. Následně dojde k druhému průchodu stromem, tentokrát přes všechny uzly a v případě, že je zásobník stop pohybujících se objektů neprázdný, vybere algoritmus tento uzel pro detekci kolizí mezi jeho objekty.

Jakmile je vybrán konkrétní uzel, nastávají dvě možnosti. První z nich je kolize pohybujícího se objektu (který zanechal odkaz na sebe na zásobníku stop) s objektem statickým, který je uložen v daném uzlu. V případě více statických objektů je nutné provést test kolizí pro všechny. Druhou možností je kolize dvou pohybujících se objektů. Je tedy nutné ještě testovat objekty ze zásobníku stop mezi sebou. Situace je zjednodušena na dvojrozměrný prostor a znázorněna na obrázku 4.6. Výše popsaný postup skýtá riziko opakovaného testu kolizí. Pro tento účel má modul fyziky implementován další zásobník kolizních dvojic, který při vkládání zajistí jedinečný výskyt dvojice a zabrání tak opakovaným detekcím v jednom kroku fyziky.



Obrázek 4.6: Quadtree – algoritmus vybírání potenciálních kolizních dvojic

Na konci každého kroku fyziky dochází k úpravě stromu. Uzly s prázdnými potomky se opět slučují. Ke slučování dochází pouze v rámci daného uzlu, tedy v případě že jsou jeho potomci bez objektů, sloučí se. Nedochází k rušení uzlů, pokud jsou v dané úrovni dělení v rámci uzlu stále objekty, jelikož se tyto objekty mohou s velkou pravděpodobností ještě vrátit. Statické objekty které jsou nyní v pohybu se odstraňují z uzlů a vkládají na zásobník pohybujících se objektů a naopak zastavené pohybující se objekty se vkládají jako statické do stromu. Struktura octree také zajišťuje vykreslování objektů. K tomuto účelu je použit klasický průchod stromem přes všechny uzly. Tato práce neřeší optimalizaci vykreslování, ale často bývá implementován algoritmus, který vykresluje pouze potenciálně viditelné objekty podle pozice kamery. Dochází tak k výraznému zrychlení výpočtů, zejména v případě výskytu mnoha objektů s hustou polygonální mřížkou.

4.8 Průběh simulace

Samotná detekce kolizí a implementace reakcí na ně ještě nezaručuje správný běh simulace. Simulace samotná musí být řízena podle požadavků na aplikaci. Nejčastější primární podmínkou běhu simulací s grafickým výstupem pro uživatele v reálném čase bývá počet snímků za sekundu, který musí být dostatečně vysoký, aby uživatel viděl plynulý pohyb objektů. Za minimální hranici počtu snímků je považováno 25 snímků za sekundu. Někteří jedinci však dokáží rozpoznat rozdíl mezi touto spodní hranicí a vyššími frekvencemi zobrazování.

V případě příliš nízkého počtu snímků za sekundu lidský mozek nedokáže interpretovat pohyb objektů jako plynulý a obraz se jeví jako trhaný sled statických obrázků. Některé aplikace, jako jsou počítačové hry vyžadují vysoké frekvence zobrazování, aby byla zachována přirozená reakce člověka na dění ve scéně. Po každém vykreslení dochází mezi dvěma snímky také ke zpracování vstupů od

uživatele či dodatečným výpočtům, ať už výše zmiňované fyziky či například umělé inteligence a jiných mechanismů.

Pro zajištění správného běhu je použit upravený algoritmus řízení simulace s podporou kalendáře událostí [11]. Tento algoritmus je implementován v metodě třídy *scene* s názvem *update*. Vykreslování následně zajišťuje další metoda téže třídy s názvem *render*. Tyto funkce jsou volány z hlavní programové smyčky v modulu *main* způsobem zapsaným v pseudokódu 4.4.

```
1  while (running)
2    start = aktuálníČas
3    zpracujVstupy() //zde může dojít k nastavení running na false
4    mainScene->update()
5    mainScene->render()
6    konec = aktuálníČas
7    dobaSnímku = konec – start
8    if (dobaSnímku <= 1/početSnímkůZaSekundu)
9      čekej(1/početSnímkůZaSekundu – dobaSnímku)
10   else
11     //například varování že nastalo zpoždění (lag)
```

Kód 4.4: Hlavní programová smyčka

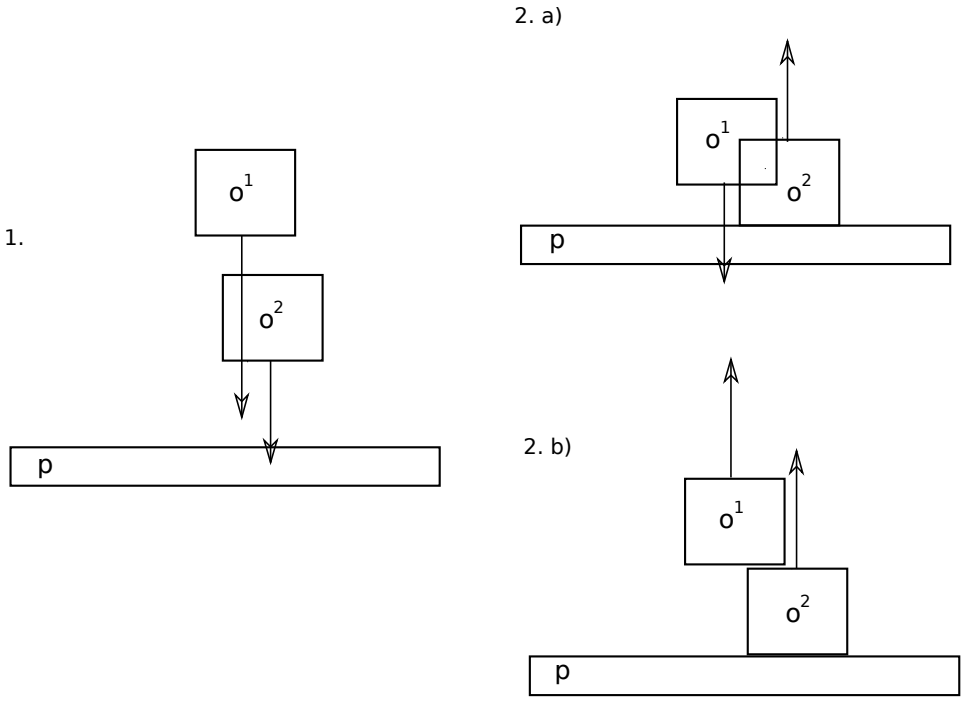
Funkce *update* dále implementuje kroky fyziky následovně:

```
1  zbývá = 1.0
2  uplynulo = 1.0
3  konec = false
4  aplikujSílyProstředí() //například gravitace
5  while (konec == false)
6    if (uplynulo == 0) //aby nedošlo k zaseknutí na nulových kolizích
7      konec = true
8    vyřešKolize()
9    uplynulo = vyřešFyziku(zbývá) //pokud je událost v čase > zbývá, vrací UKONČIT
10   if (uplynulo == UKONČIT) break
11   posunoutObjekty(uplynulo) //posune všechny objekty o daný čas
12   přestavětStrom()
13   zbývá -= uplynulo
14 posunoutObjekty(zbývá) //doposunout objekty do konce snímku
15 přestavětStrom()
```

Kód 4.5: Simulační smyčka

Funkce označená v pseudokódu jako *vyřešFyziku*, v kódu této práce jako *summarizePhysics* je implementována v modulu fyziky a zajišťuje přechod mezi algoritmem řízení simulace a modulem fyziky. Je v podstatě vstupním bodem do modulu fyziky. Tato funkce pracuje se zásobníkem událostí fyziky. Každý záznam v tomto zásobníku, který lze nazvat podle terminologie odvětví modelování a simulací také kalendářem obsahuje popis události a její aktivační čas. V případě této práce obsahuje tento kalendář pouze kolizní dvojice objektů a časy jejich kolizí.

V každém kroku fyziky dojde k výběru záznamů s časem menším či rovným nejmenšímu nenulovému aktivačnímu času z tohoto kalendáře a k jejich zpracování. V případě absence tohoto času vezme všechny nulové. Záznamy jsou seřazeny před výběrem pomocí řadícího algoritmu bubble sort [9]. Událost fyziky je zpracována a jsou nastaveny případné nové rychlosti objektů. Následně funkce vrací nejvyšší aktivační čas právě zpracovaných událostí algoritmu řízení simulace a dojde k posunu objektů na tento čas. Pokud je událost dál, než je zbývající čas do konce snímku či je kalendář prázdný, vrací funkce speciální konstantu ohlašující přerušení cyklu fyziky a možnost postoupení na další snímek. Obrázek 4.7 ilustruje více přístupů k výběru záznamů z kalendáře.



Obrázek 4.7: Výběr událostí z kalendáře

a) Výběr všech událostí v daném čase najednou

b) Výběr pouze první události

Na obrázku 4.7 je popsána situace výběru událostí. Pohybující se objekty o^1 a o^2 mají různé rychlosti stejného směru směrem ke statické překážce p. Oba mají kolizi s p, přičemž čas kolize o^1 je větší než čas kolize o^2, jelikož objekt o^1 je dál od překážky. V tomto stavu nemají objekty o^1 a o^2 vzájemnou kolizi, nebo ji mají až v čase přesahujícím aktuální snímek.

V případě zpracování všech událostí stejného času v jednom kroku dojde ke stavu, kdy objekty skončí zaklíněné v sobě, jelikož o^1 nemá kolizi s o^2 a nezjistí, že o^1 se ve svém pohybu zarazí

o překážku. Výhodou tohoto postupu je eliminace vícenásobného přepočítávání scény, nevýhodou je nutnost řešit nechtěné průniky objektů.

V případě výběru pouze první kolize dojde ke zpracování kolize o^1, následnému přepočítání scény a zpracování kolize obou pohybujících se objektů. Jelikož se scéna ihned po první kolizi přepočítá, ví už druhý o^2, že mu stojí v cestě překážka. Výhodou tohoto přístupu je eliminace kolizních chyb při kolizích objektů ve stejném čase, nevýhodou je zvýšená výpočetní náročnost. V případě situace na obrázku 4.7 dojde v prvním scénáři k jednomu kroku fyziky, zatímco u druhého ke krokům dvěma. U prvního je však nutné připomenout nutnost vyšší režie pro řešení průniku objektů.

Nicméně ani při výběru pouze první kolize nelze zaručit bezchybnost. Algoritmus nepozná, která z kolizí se stane dřív, pokud mají stejný čas. Výběr první kolize tudíž nemusí vždy eliminovat všechny nechtěné průniky. Pro účel této práce byla implementována metoda výběru více událostí ve stejném čase, ba dokonce v čase následující kolize.

4.9 Zásobníková struktura

V rámci implementace této práce vznikla také univerzální datová struktura s pracovním názvem *superStack* silně inspirovaná implementací zásobníku. Cílem bylo navrhnout jednotnou dynamickou datovou strukturu pro použití na více místech. Konkrétně je využívána jako kalendář událostí fyziky, seznam odkazů na objekty v uzlu *octree*, seznam stop pohybujících se objektů v *octree* a seznam pohybujících se objektů ve třídě *octree* mimo stromovou strukturu.

Implementačně je řešena, jako C++ template pracující z odkazy na objekty jedné třídy. Struktura se chová jako klasický zásobník s metodami *push* a *pop*, navíc však obsahuje také metodu *top* pro výběr prvního elementu bez odstranění ze zásobníku, *pushUnique* pro vložení objektu za podmínky, že již není přítomen na zásobníku, *remove* pro odstranění objektu z libovolného místa na zásobníku a *sort* pro seřazení objektů na zásobníku pomocí algoritmu bubble sort. Pro použití metod *pushUnique* a *sort* je nutné, aby třídy daných objektů měly implementováno přetížení logických operátorů pro porovnávání. Struktura dává také k dispozici veřejně odkaz na pole objektů, přes které lze iterovat.

Jelikož je tato struktura dynamická, musí docházet k jejímu rozšíření v případě nedostatku místa. Položky jsou uloženy jako klasické pole. Pole je alokováno pro předem danou velikost, která je definována konstantou *PRE_ALLOCATE*. Tuto konstantu lze měnit podle předpokládaného množství elementů. Je alokováno zpravidla více místa, aby nedocházelo k častému rozšiřování a nové alokaci. Až v případě překročení volného místa dojde k zdvojnásobení alokovaného paměťového prostoru.

4.10 Matematické funkce a struktury

V projektu je využita knihovna GLM, zajišťující základní operace pro vektory a matice při práci s 3D geometrií, zejména u OpenGL. Datové struktury GLM jsou však využity pouze při výpočtech. Staticky uložená data jako je pozice objektu, rychlost a podobné jsou uloženy v datových strukturách definovaných za použití nativního C++. Všechny přídavné struktury a funkce jsou implementovány v modulu *generalMath*.

Důležitou částí je definice datových struktur pro práci s modelem. Jedná se o uživatelem definované datové typy nad strukturou, obsahující zpravidla tři prvky pozice kartézského souřadného systému. Jedná se o typ *vertex*, *texture*, *normal* a *vector3D* pro obecné použití. Tyto typy by samozřejmě mohly být implementovány pomocí jednoho. Z hlediska zásad pro bezpečné programování však je správné definovat raději více datových typů pro lepší typovou kontrolu. Tyto typy mají také přetížené potřebné operátory, které jsou třeba během výpočtů.

Datový typ *face* pro polygon modelu se liší obsahem. Namísto tři souřadnic má tři odkazy na *vertex*, tři odkazy na *normal* a tři odkazy na *texture*. Definuje tak jeden trojúhelník modelu se souřadnicemi bodů, normál a texturových souřadnic.

Mimo jiné se vyskytuje v tomto modulu také několikrát přetížená funkce pro výpočet skalárního součinu vektorů. Existuje několik verzí pro dané datové typy z čehož jedna varianta je optimalizována pro výpočet projekce na jednu ze tři souřadných os. V takovém případě není třeba používat vzorec pro výpočet, ale pouze je vrácena hodnota projektovaného vektoru na pozici příslušné souřadnice požadované projekční souřadné osy.

4.11 Implementační problémy

Při implementaci postupů popsaných v tomto dokumentu dochází k různým problémům, které nesouvisí přímo s implementovanou problematikou, avšak vyžadují ošetření pro správný chod programu. V této podkapitole jsou popsány nejčastější problémy, které se vyskytly při implementaci této práce.

4.11.1 Porovnávání proměnných typu float

Při pohybu objektů prostorem dochází k výše popsaným změnám v rychlostech, které mohou nabývat jakýchkoliv hodnot. Vyjádření čísla v plovoucí řádové čárce však není vždy naprosto přesné. Někdy může pouhá změna pořadí u komutativních aritmetických operací způsobit změnu hodnoty výrazu. Tyto rozdíly hodnot zpravidla nejsou příliš velké. Nicméně i například rozdíl na úrovni miliontin může způsobit chybné vyhodnocení logického výrazu.

Jedním z možných řešení je definování tolerance tohoto rozdílu. Vzorec (4.6) ukazuje převod operátoru rovnosti za pomocí tolerance.

$$a == b \quad => \quad absVal(a-b) < EPSILON \tag{4.6}$$

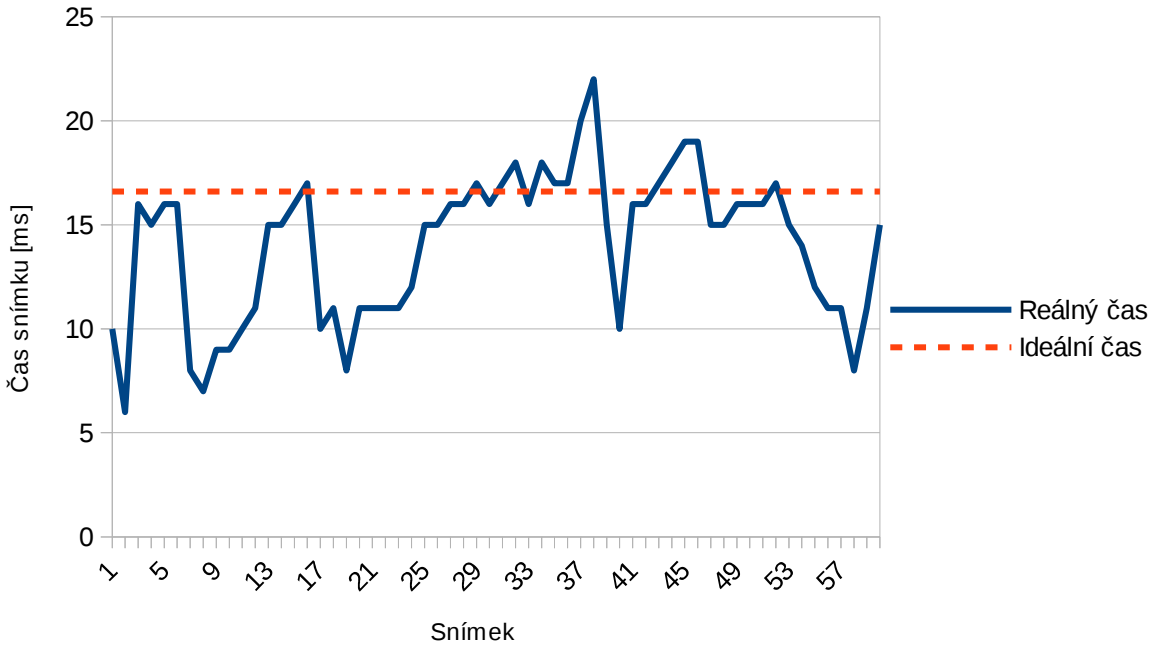
Porovnání je náročnější, jelikož dochází k navýšení operací o rozdíl a absolutní hodnotu. Číslo *EPSILON* udává maximální chybu a mělo by být dostatečně malé, aby nedocházelo zase k chybám kvůli příliš velké toleranci. Jedná se o nejjednodušší postup, který je pro účely této práce dostačující. Pro lepší výsledky lze použít relativní tolerance či převodu na jiné datové typy, jako je integer.

5 Měření

Funkcionalita implementované knihovny je předvedena v demonstrační aplikaci. Zde lze provádět různé testy a měření. Exaktní měření může být aplikováno například na výpočetní náročnost, zpoždění snímků či počet provedených kolizních detekcí. Plynulost a přirozenost pohybu těles je však nutno hodnotit subjektivně srovnáním s reálným světem.

5.1 Časová náročnost

Simulace je navržena, aby běžela rychlostí 60 snímků za sekundu. V této frekvenci dochází k překreslování scény a přepočítávání fyziky. Graf 5.1 ukazuje časový průběh simulace při kolizi pěti objektů ve velmi krátkém čase ve scéně č. 3 z demo aplikace. Veškerá časová měření v této kapitole probíhají na průměrně výkonné sestavě *12 GB RAM, Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz, GeForce GTX 550 Ti 1024 MB*. Jednotlivé naměřené časy se vztahují na jeden snímek, který zahrnuje potřebné kroky fyziky a vykreslení scény. Předpokládaný optimální čas snímku je 1/60=16.6 ms.



Graf 5.1: Sekundový výřez časového průběhu simulace – demo 3
Průměrná doba snímku: 14.05 ms

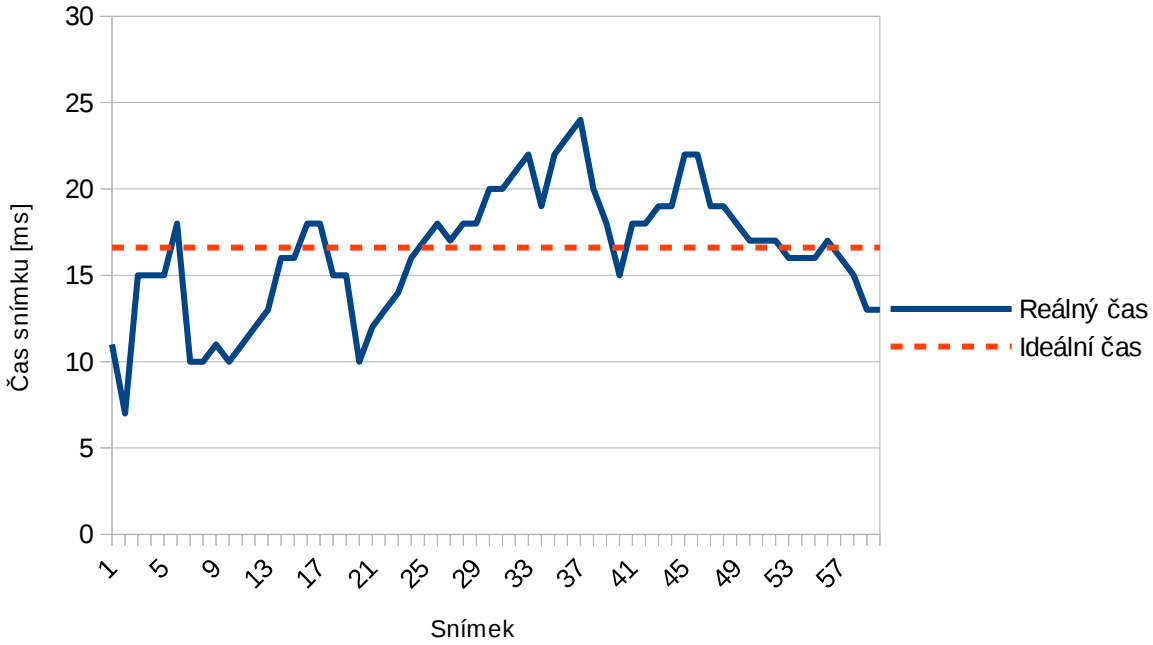
Z tabulky lze vyčíst nárůst zpoždění v několika snímcích za sebou. Jedná se o stav, kdy bylo nutné vícekrát přepočítat scénu díky vícenásobným kolizím a případným průnikům objektů a korekcím. Zpoždění je však v normě a změna rychlosti není běžným uživatelem postřehnutelná. Průměrná délka snímku od spuštění scény do ustálení objektů činí 12.645 ms.

5.2 Pohyb těles

Hodnocení kvality výsledku je především založeno na subjektivním pozorování pohybu těles ve scéně. Tělesa se pohybují podle očekávaných fyzikálních zákonů a blíží se pohybům těles v reálném světě. Mírné problémy při pohybu je možné pozorovat zejména při ustálení těles. Tělesa se místy pohybují trhavými pohyby a oscilují okolo rovnovážné polohy. Tento stav je způsoben korekcí poloh těles při průnicích, ke kterých dochází díky rotačním pohybům tělesa v těsné blízkosti tělesa jiného. Často k tomuto jevu dochází při ustálení těles například na podlaze.

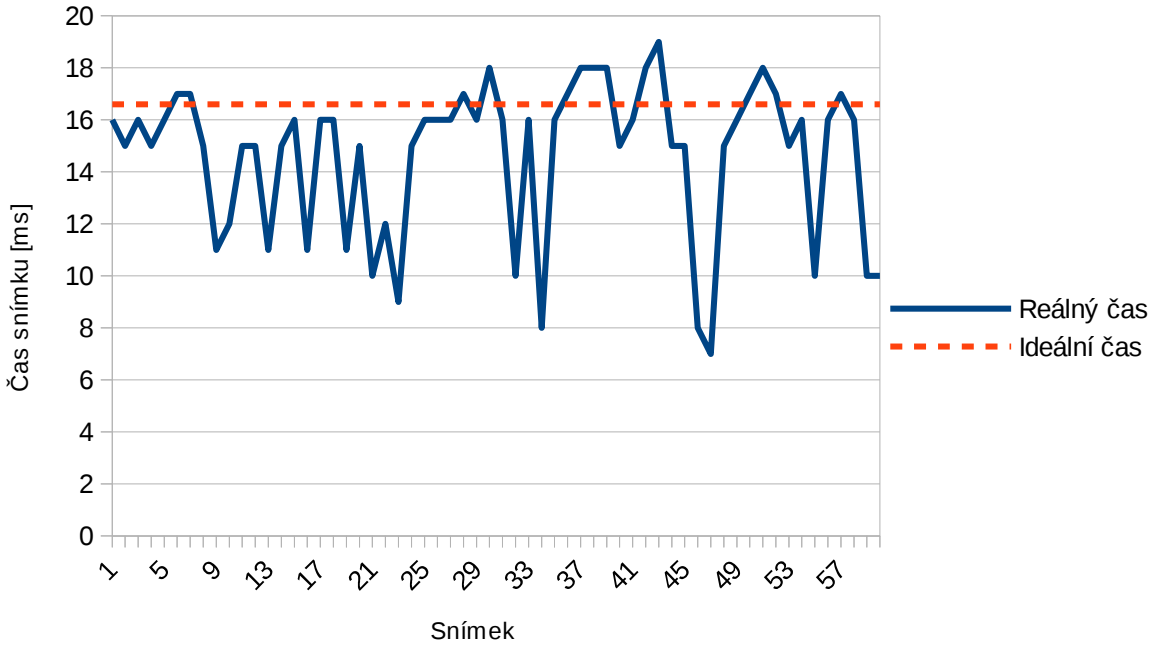
5.3 Octree optimalizace

Optimalizace pomocí octree je účinná pouze v případě výskytu většího množství objektů pohybujících se v různých částech scény, což je v pořádku a podle předpokladů. Například v případě pohybu dvou objektů ve středu scény a zapnutém dělení prostoru do první úrovně dochází k větší režii, kdy octree hlásí modulu fyziky kolizi těchto objektů osmkrát, jednou pro každý podstrom. V případě pohybu například dvou dvojic objektů, kdy je každá dvojice v jiné části scény (v jiném octree uzlu) dochází k hlášení dvou kolizních dvojic na rozdíl od šesti (pravidlo kombinace pro dvojice z množiny o 4 prvcích). Paradoxně dochází tedy ke zpomalení u scény č. 3 z demo aplikace, kde jsou objekty umístěny přibližně uprostřed scény. Graf 5.2 vykazuje nárůst zpoždění oproti grafu 5.1.

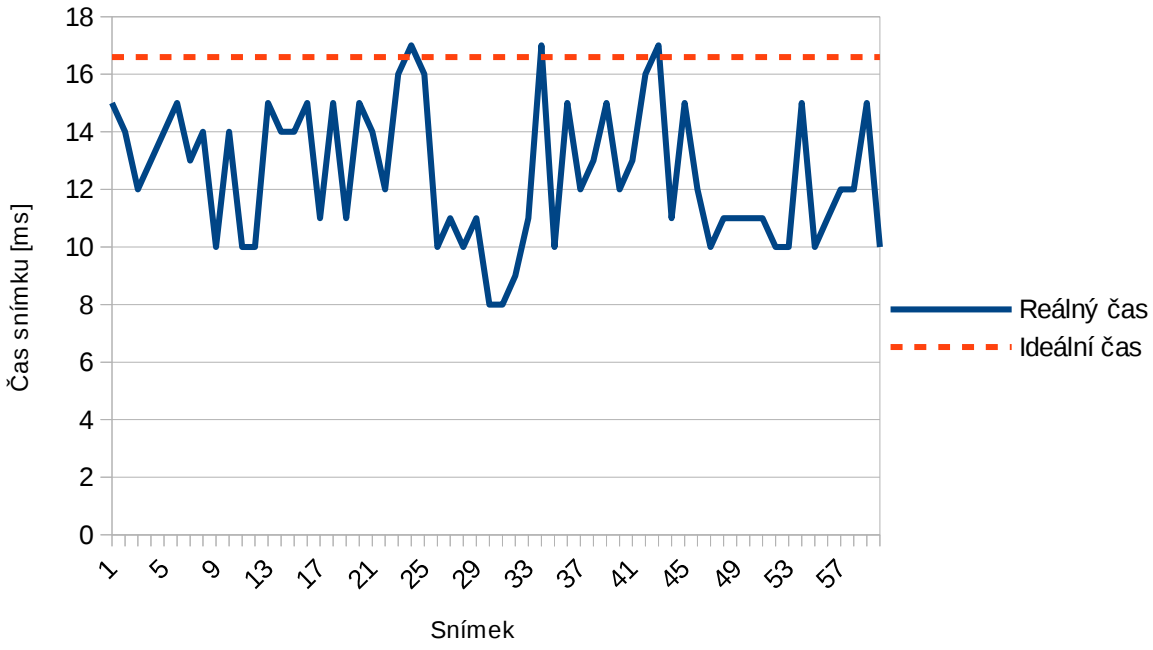


Graf 5.2: Sekundový výřez časového průběhu simulace – demo 3 (octree)
Průměrná doba snímku: 16.33 ms

Grafy 5.3 a 5.4 naopak ukazují zrychlení v případě osmi kolizních dvojic umístěných tak, aby se po zapnutí optimalizace nacházely každá v jednom oktantu stromu.

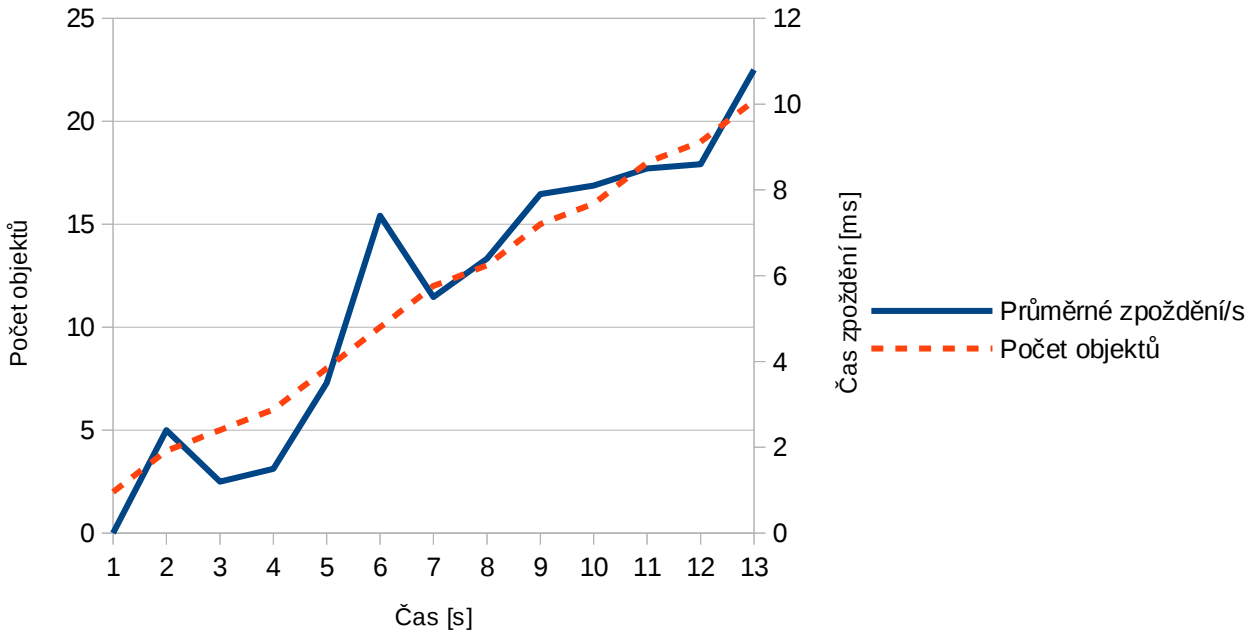


Graf 5.3: Sekundový výřez časového průběhu simulace – ideální scéna bez octree
Průměrná doba snímku: 14.65 ms



Graf 5.4: Sekundový výřez časového průběhu simulace – ideální scéna s octree
Průměrná doba snímku: 12.56 ms

Poslední měření 5.5 ukazuje nárůst zpoždění při nárůstu kolidujících objektů ve scéně bez zapnuté optimalizace. Ve scéně bylo několik objektů, které postupně padaly na sebe a tvořily tak hromádku. Graf zobrazuje průměrné zpoždění za sekundu a počet kolidujících objektů.



Graf 5.5: Nárůst zpoždění snímků při nárůstu počtu kolidujících objektů

6 Závěr

V této práci jsem se seznámil se základními principy činnosti fyzikální simulace s grafickým výstupem ve formě trojrozměrné scény. Hlavním úkolem byla implementace detekce kolizí, která byla implementována pomocí techniky Separating Axis Theorem. Pro realizaci této práce bylo nutné nastudovat teoretické znalosti z oblasti fyzikálních výpočtů, geometrie, modelování a simulací, tvorby herního enginu a akcelerace geometrických výpočtů. Z praktického hlediska bylo zapotřebí naučit se pracovat s grafickou knihovnou OpenGL, multimediální knihovnou SDL a také bylo nutné seznámit se s mírně pokročilejšími technikami jazyka C++ jako je template, přetížení funkcí a operátorů a problematika srovnávání čísel s plovoucí řádovou čárkou. Nejproblematičtější částí implementace byla realizace odezvy na kolize za použití modulu fyziky, zejména po aktivaci gravitace a zahrnutí rotačních pohybů těles.

Knihovna, která je výstupem této práce je použita pro běh demonstrační aplikace, která zobrazuje základní 3D scénu, kde dochází ke kolizím mezi objekty různých typů. Pohyb těles místy vykazuje nedokonalosti, zejména u případů, kdy dochází k nechtěným průnikům těles a jejich korekci, nebo při ustálení objektů. Mimo tyto problémy však pohyb objektů působí relativně přirozeně a objekty interagují mezi sebou na dostatečně kvalitní úrovni.

Výsledná funkcionalita práce by se měla alespoň tématicky blížit moderním fyzikálním či herním enginům jako je CryEngine, Havok, Unreal Engine, nebo fyzikální jádro programu pro 3D modelování Blender. Implementace naprosto stejné funkcionality jako je u těchto nástrojů již dalece přesahuje zadání této práce. Vývoj enginu podobného rozsahu jako výše jmenované je náročným procesem zahrnujícím celé programátorské týmy. Nicméně je zde prostor pro rozšíření této práce o další funkcionalitu. Může se jednat například o implementaci pokročilé detekce kolizí a reakcí na ně zahrnující i jiná tělesa než rigidní, deformace či rovnou exploze objektů. Dále je možné přidat podporu animací objektů, pokročilé fyziky, moderních vykreslovacích technik pro realistický vzhled scény, nebo částicových efektů. Většího výkonu by bylo možné dosáhnout také implementací dalších optimalizačních technik například i na úrovni assembleru či s využitím hardwarové akcelerace na úrovni GPU, programování více jader CPU či paralelního programování.

Literatura

[1] NEWTON, Isaac. *Philosophiæ Naturalis Principia Mathematica*. STREATER, Joseph. London: Royal Society, 1687.

[2] GRAY, Jeremy. *Olinde Rodrigues' Paper of 1840 on Transformation Groups.* Archive for History of Exact Sciences 21.4. New York, USA: Springer, 1980.

[3] ERICSON, Christer. *Real-Time Collision Detection.* The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2004. ISBN: 1-55860-732-3.

[4] MÖLLER, Tomas. *A Fast Triangle-Triangle Intersection Test*. *Journal of graphics tools 2.2.* Stanford, CA, USA: Stanford University, 1997.
Dostupné z: http://web.stanford.edu/class/cs277/resources/papers/Moller1997b.pdf

[5] EBERLY, David. *Intersection of Convex Objects:The Method of Separating Axes* [online]. 2001, 2008 [cit. 2015].
Dostupné z: http://geometrictools.com/Documentation/MethodOfSeparatingAxes.pdf

[6] WELLER, René. *New Geometric Data Structures for Collision Detection and Haptics: Springer Series on Touch and Haptic Systems.* New York, USA: Springer, 2013. ISBN: 978-3-319-01020-5.

[7] GRAHAM, Ronald. *An efficient algorithm for determining the convex hull of a finite planar set.* Information processing letters 1.4. 1972.

[8] VAN DAM, Andries. *Introduction to computer graphics.* FEINER, Steven. HUGHES, John. Vol. 55. Boston, USA: Addison-Wesley, 1994. ISBN: 0-201-60921-5.

[9] CORMEN, Thomas. *Introduction to Algorithms.* LEISERSON, Charles. RIVEST, Ronald. STEIN, Clifford. Cambridge, Massachusetts, USA: MIT Press, 1990. ISBN: 978-0-262-03384-8.

[10] EBERLY, David. *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic*. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2005. ISBN: 0-12-229064-X.

[11] RÁBOVÁ, Zdeňka. *Modelování a simulace.* ZENDULKA, Jaroslav. ČEŠKA, Milan. PERINGER, Petr. JANOUŠEK, Vladimír. Brno, ČR: Vysoké učení technické v Brně, 1992. ISBN: 80-214-0480-9.

[12] HECKER, Chris. *Physics, part 3: Collision Response*. [online]. 1997, [cit. 2015]. Dostupné z: http://chrishecker.com/images/e/e7/Gdmphys3.pdf

[13] VAN VERTH, James. *Essential Mathematics for Games and Interactive Applications*. BISHOP, Lars. Boca Raton, Florida, USA: CRC Press, 2004. ISBN: 978-0-12-374297-1

[14] MOJTAHEDZADECH, Rasoul. *A principle of minimum translation search approach for object pose refinement*. LILIENTHAL, Achim. In: *IEEE (ed.), 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Hamburg, Germany: IEEE Press, 2015 [cit. 2015].