VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

FYZIKÁLNÍ SIMULÁTOR PRO HRY TYPU SANDBOX

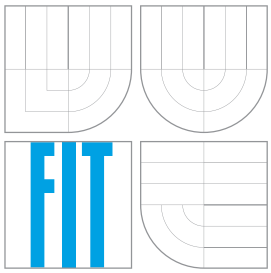
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PATRIK KOTULIČ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

FYZIKÁLNÍ SIMULÁTOR PRO HRY TYPU SANDBOX

PHYSICS SIMULATOR FOR SANDBOX TYPE GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PATRIK KOTULIČ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN HRUBÝ, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá tvorbou real-time fyzikálního simulátoru mechaniky tuhých těles. Čtenář je seznámen s principy, na kterých jsou založeny současné simulátory mechaniky tuhých těles. Práce se dále zabývá numerickou integrací pohybových rovnic a řešením kolizí včetně jejich detekce. V závěru je simulátor představen na sadě ukázek. Výsledná aplikace může sloužit jako hra, popř. nástroj pro experimentování a získávání znalostí.

Abstract

This thesis focuses on the creation of a physics simulator of rigid body mechanics. The reader is familiarized with the principles used in current simulators of rigid body mechanics. The thesis then deals with numerical integration of the equations of motion, collision detection and collision resolution. The simulator is presented on a series of experiments. The resulting application can be used as a video game or as a tool for experimentation.

Klíčová slova

spojitá simulace, mechanika tuhých těles, dynamika, kinematika, detekce kolizí, numerická integrace, počítačová hra

Keywords

continuous simulation, rigid body mechanics, dynamics, kinematics, collision detection, numerical integration, video game

Citace

Patrik Kotulič: Fyzikální simulátor pro hry typu Sandbox, bakalářská práce, Brno, FIT VUT v Brně, 2012

Fyzikální simulátor pro hry typu Sandbox

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Martina Hrubého, Ph.D.

.....
Patrik Kotulič
9. května 2012

Poděkování

Rád bych poděkoval Ing. Martinu Hrubému, Ph.D. za vedení této práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je navrhnout a popsat proces tvorby simulátoru mechaniky tuhých těles, s jehož uplatněním se setkáme nejčastěji v herním průmyslu. V současnosti na využití fyziky ve videohrách narazíme velmi často – ať už jde o let fotbalového míče, závody aut nebo posouvání objektů v bludišti. Zatímco v minulosti se fyzikální jevy pouze napodobovaly, dnes již hry využívají simulace a dosahují realistických výsledků nejen „na oko“.

Fyzikální simulace ve hrách se liší od většiny ostatních simulací tím, že musí pracovat v reálném čase, aby s herním světem mohl uživatel za běhu interagovat. Kvůli tomu je potřeba dělat kompromisy mezi přesností a rychlostí. Jelikož si každý herní vývojář nemůže dovolit vyvíjet vlastní fyzikální moduly a jelikož je často potřeba jednou navržený kód recyklovat, vznikají fyzikální enginy a simulátory obecného charakteru, které se dají přizpůsobit požadovaným účelům. Na simulaci mechaniky tuhých těles je založen např. engine Box2D [2], s nímž se můžeme setkat ve hře *Angry Birds*, která od prosince roku 2009 prodala přes 12 milionů kopií.

Navržený simulátor může sám sloužit jako hra. Podobné hry označujeme jako Sandbox (pískoviště), neboť v nich lze stavět a bourat, nemají žádný konkrétnější cíl a v podstatě nikdy nekončí. Simulátor zároveň může sloužit jako zdroj znalostí, pomocí něhož se lze přesvědčit o platnosti některých fyzikálních zákonů.

V kapitole 2 se budeme zabývat fyzikálními zákony a veličinami, které jsou potřeba pro návrh a implementaci simulátoru. Podíváme se na simulátory zabývající se stejnou problematikou a na mechanismy, které se v nich využívají. Návrh simulátoru a jeho součástí je předmětem kapitoly 3. Kapitola 4 popisuje zajímavé části implementace. V kapitole 5 je prezentováno několik ukázek z vytvořeného simulátoru.

Kapitola 2

Simulace mechaniky tuhých těles

Abychom mohli navrhnout fyzikální simulátor, musíme si nejprve objasnit, které fyzikální prvky budeme modelovat, a kterými zákony a veličinami se budeme zabývat. Dále nahlédneme na již existující fyzikální enginy a simulátory a popíšeme si postupy, na kterých jsou založeny.

2.1 Mechanika tuhých těles

Tato sekce má sloužit jako fyzikální přehled, na kterém bude stavět zbývající část práce, a vychází ze znalostí čerpaných z [9].

Simulátor bude zkoumat *tuhá tělesa*. Tuhé těleso je takové těleso, které nepodléhá žádným deformacím a nezávisle na okolí drží svůj původní tvar. Na první pohled by se mohlo zdát, že tuhé a pevné těleso jsou zaměnitelné pojmy, nicméně pevná tělesa přeci jen mění svůj tvar. U opravdu pevných těles je toto vnímatelné pouze na velmi malém měřítku, nicméně nesmíme zapomenout, že i gumový míček či kus látky jsou pevná tělesa a těm lze změnit tvar podstatně viditelněji (dokonce i trvale). Tuhá tělesa nelze ani lámat či navzájem spojovat. I přes tuto limitaci můžeme použít koncept tuhého tělesa jako velmi dobrou abstrakci pro objekty reálného světa. Pro většinu počítačových her je tato abstrakce více než dostačující a pokud bychom chtěli modelovat např. kapaliny nebo deformovatelné objekty, lze toho docílit určitými oklikami i v simulátorech tuhých těles (ovšem s výraznou ztrátou přesnosti).

Základním požadavkem na tělesa je, aby se pohybovaly. Oblast fyziky, která se zabývá pohybem, nazýváme *kinematika*. Abychom mohli diskutovat polohu a pohyb, musíme zavést souřadný systém. Jelikož budeme simulovat dvourozměrný prostor, použijeme kartézskou soustavu souřadnic s osami x a y. Polohu každého bodu budeme určovat vzhledem k počátku souřadnicového systému a popíšeme ji dvěma skaláry – ty označme x a y podle os, ke kterým se vztahují. Tato dvojice skalárů tvoří tzv. *polohový vektor*, tedy vektor posunutí daného bodu P vůči počátku soustavy O. Značíme jej:

$$\boldsymbol{r} = \overrightarrow{OP} = (x, y)$$

Závislost polohy na čase popisuje veličina zvaná *rychlost*. Okamžitou rychlost, tedy rychlost v daném čase t, definujeme pro jeden rozměr (např. složku x) jako:

$$v_x = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} = \frac{\mathrm{d}x}{\mathrm{d}t}$$

kde derivaci podle času můžeme zkráceně zapsat tečkovou (Newtonovou) notací:

$$\dot{x} = \frac{\mathrm{d}x}{\mathrm{d}t}$$

Pro popis rychlosti a polohy ve více rozměrech můžeme použít vektorový zápis:

$$\boldsymbol{v} = \dot{\boldsymbol{r}} \tag{2.1}$$

Za předpokladu, že se rychlost bodu v čase nemění, můžeme určit jeho polohu v jakémkoli následujícím okamžiku. Takový pohyb označujeme jako *rovnoměrný přímočarý pohyb*. V realitě ovšem často pozorujeme změny rychlosti – brzdící automobil, volný pád, apod. Potřebujeme tedy sledovat i závislost rychlosti na čase. Veličina, která tuto závislost popisuje, se nazývá *zrychlení* a značíme ji:

$$\boldsymbol{a} = \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \dot{\boldsymbol{v}} = \ddot{\boldsymbol{r}} \tag{2.2}$$

Když už jsme schopni pohyb popsat, bude nás zajímat, co jej vůbec může vyvolat. Tím se zabývá jiná část fyziky – *dynamika*. Osamocené těleso, které neinteraguje s žádnými objekty, nikdy nezmění svou rychlost. Bude setrvávat v klidu nebo v rovnoměrném přímočarém pohybu. Tento fakt popisuje první Newtonův pohybový zákon. Až působením okolních těles můžeme dosáhnout změn rychlosti. Tato působení označujeme jako *síly* a jejich souhrnný vliv na dané těleso definujeme pomocí druhého Newtonova pohybového zákonu jako:

$$\sum \boldsymbol{F} = m\boldsymbol{a} \tag{2.3}$$

kde $\sum \boldsymbol{F}$ je výslednice všech sil, které působí na dané těleso, $\boldsymbol{a}$ je zrychlení tělesa a m jeho hmotnost. Hmotnost je základní vlastnost každého tělesa, která určuje poměr mezi účinkující silou a zrychlením, která tato síla vyvolá. Integrací síly podle času získáme *hybnost*:

$$\boldsymbol{p} = \int \boldsymbol{F}\mathrm{d}t = m \int \boldsymbol{a}\mathrm{d}t = m\boldsymbol{v}$$

Pro popis náhlých změn hybnosti (např. při srážkách) definujeme *impulz síly* ve tvaru:

$$\boldsymbol{J} = \int_{t_1}^{t_2} \boldsymbol{F}\mathrm{d}t = \int_{t_1}^{t_2} \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}\mathrm{d}t = \Delta\boldsymbol{p} \tag{2.4}$$

Rovnice pro sílu a hybnost neplatí pouze pro bod, ale pro celé těleso. Je tomu tak díky existenci *těžiště*, což je bod, na který má tíhová síla stejný účinek jako na celé těleso. Těžiště $\boldsymbol{r}^T$ určíme jako vážený průměr všech hmotných bodů daného tělesa:

$$\boldsymbol{r}^T = \frac{\sum\limits_{i=1}^{n} m_i \boldsymbol{r}_i}{\sum\limits_{i=1}^{n} m_i} = \frac{\sum\limits_{i=1}^{n} m_i \boldsymbol{r}_i}{M}$$

kde $\boldsymbol{r}^i$ jsou polohy jednotlivých bodů a M je celková hmotnost tělesa.

Pro popis posuvného (lineárního) pohybu můžeme tělesa nahradit hmotnými body v polohách jejich těžišť. Síly, které nepůsobí v těžišti, mohou vyvolat otáčivý (rotační) pohyb.

Rotace na rozdíl od translace vyžaduje určitou referenční osu (resp. bod, do kterého tuto osu promítneme, pracujeme-li ve dvou dimenzích), kolem které chceme těleso otáčet. Ve většině případů bude středem otáčení těžiště.

Body tělesa při otáčivém pohybu opisují kružnici se středem v ose otáčení. Pro úhel, o který je těleso otočeno z jeho původní polohy, platí vztah:

$$\theta = \frac{s}{R}$$

kde s vyjadřuje délku opsaného oblouku a R je poloměr kružnice. Úhel θ budeme měřit v kladném směru osy x. Kladné hodnoty úhlových veličin vyjadřují pohyb proti směru otáčení hodinových ručiček. Otočením $\Delta\theta$ rozumíme změnu úhlu θ_1 na úhel θ_2 podle rovnice:

$$\Delta\theta = \theta_2 - \theta_1$$

Rotaci tělesa v čase popisuje *úhlová rychlost*. Určíme ji jako:

$$\omega = \dot{\theta} \tag{2.5}$$

a podobně závislost úhlové rychlosti na čase vyjadřuje *úhlové zrychlení*

$$\varepsilon = \dot{\omega} = \ddot{\theta} \tag{2.6}$$

Úhlová rychlost a zrychlení jsou ve výše uvedených rovnicích popsány skaláry. Ale ve skutečnosti jde o vektorové veličiny – mají směr osy otáčení. Nás bude ovšem zajímat pouze velikost, jelikož pracujeme ve dvourozměrném prostoru a otáčet budeme vždy kolem os kolmých na pozorovanou plochu. V trojrozměrném prostoru by šlo o vektory $(0, 0, 1)$ a $(0, 0, -1)$, které jsme schopni rozlišit pouze znaménkem.

Úhlová rychlost a zrychlení jsou ve výše uvedených rovnicích popsány skaláry. Ale ve skutečnosti jde o vektorové veličiny – mají směr osy otáčení. Jelikož pracujeme ve dvourozměrném prostoru, budeme tělesa otáčet vždy kolem os kolmých na pozorovanou plochu. Směr úhlových veličin můžeme popsat vektory $(0, 0, 1)$ a $(0, 0, -1)$, které jsme schopni rozlišit pouze znaménkem, a proto k jejich popisu stačí skaláry.

Za časový interval Δt opíší všechny body stejný úhel θ. Je tedy patrné, že body ležící dále od středu otáčení urazí větší vzdálenost a pohybují se rychleji. Pro nás bude často důležité umět tuto skutečnost vyjádřit a provádět převody mezi lineární a úhlovou rychlostí. Převodní vztah můžeme odvodit z pohybu pro kružnici:

$$v \ = \frac{\mathrm{d}s}{\mathrm{d}t} = \frac{\mathrm{d}(\theta R)}{\mathrm{d}t} = \omega R \tag{2.7}$$

a derivací podle času odvodíme i rovnici pro zrychlení

$$a \ = \varepsilon R$$

Otáčivý účinek síly vyjadřuje veličina zvaná *moment síly*. Velikost momentu síly pro danou sílu $\boldsymbol{F}$ určíme jako:

$$\tau = rF \sin\alpha$$

kde $\boldsymbol{r}$ je poloha působiště síly $\boldsymbol{F}$ vůči středu otáčení O a α značí úhel, který svírá $\boldsymbol{F}$ a $\boldsymbol{r}$. Daný vztah výrazně připomíná vzorec pro vektorový součin a opravdu jej lze zapsat ve vektorovém tvaru jako:

$$\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F} \tag{2.8}$$

Souhrnný vliv všech sil na otáčivý pohyb tělesa vyjadřuje suma momentů těchto sil a platí:

$$\sum \tau = I\ \varepsilon \tag{2.9}$$

přičemž ε je celkové úhlové zrychlení tělesa a I je *moment setrvačnosti*. Podobný význam jako měla hmotnost ve vztahu 2.3 pro sílu má zde právě moment setrvačnosti. Jde o koeficient vyjadřující poměr mezi momentem síly a přiděleným úhlovým zrychlením. Moment setrvačnosti udává, jak náročné bude roztočit těleso v daném bodě, a obecně jej určíme jako:

$$I\ = \sum m_i r_i^2 \tag{2.10}$$

kde r_i je vzdálenost i-tého bodu od středu otáčení, a proto se moment setrvačnosti na rozdíl od hmotnosti se neváže pouze k tělesu, ale i ke zvolené ose. Vztah pro moment setrvačnosti najdeme pro konkrétní tělesa v jednodušší (upravené) podobě v tabulkách. Například pro kruh s poloměrem R a hmotností m otáčející se kolem svého středu platí:

$$I\ = \frac{1}{2} m R^2 \tag{2.11}$$

Uvedeme si také vztahy pro síly, které budeme modelovat. *Tíhová síla* má působiště v těžišti a pro tělesa na Zemi, jejichž hmotnost je vůči Zemi zanedbatelná, ji určíme jako:

$$F_g = -mg \tag{2.12}$$

kde g je *tíhové zrychlení*, jehož střední hodnota pro tělesa na Zemi, se kterou budeme počítat, je $g = 9.80665$. Druhou modelovanou silou bude *pružinová síla*, kterou popisuje Hookův zákon jako:

$$F_p = -kx \tag{2.13}$$

kde x je výchylka konce pružiny z jeho rovnovážné polohy a k je *tuhost pružiny*, která udává odolnost pružiny vůči stlačení.

Pro popis pohybu v kapalině, zjednodušenou verzi odporu vzduchu nebo tlumenou pružinu můžeme použít *brzdnou sílu*:

$$F_b = -bv \tag{2.14}$$

kde b je *součinitel útlumu* a v je rychlost tělesa, proti němuž brzdná síla působí.

Leží-li těleso na podložce, působí na něj *tlaková síla* F_n, která má směr tzv. *normály* (kolmice na podložku). Velikost tlakové síly závisí na ostatních silách – tlaková síla bude tak velká, aby vyrovnala síly, kterými působí těleso na podložku.

Pokud bychom se snažili posunout po podložce těleso, které je vůči podložce v klidu, bude proti směru zamýšleného pohybu působit *statická třecí síla* F_s tak velká, aby přesně vyrovnala síly způsobující pohyb. Těleso se po podložce začne pohybovat, jakmile síla způsobující pohyb překoná maximální hodnotu statické třecí síly:

$$F_{s,max} = f_s F_n \tag{2.15}$$

kde f_s je koeficient statického tření, což je materiálová konstanta společná pro obě tělesa. Jakmile je těleso na podložce uvedeno do pohybu, třecí síla skokově klesne na hodnotu tzv. *dynamické třecí síly* F_d s hodnotou:

$$F_d = f_d F_n \tag{2.16}$$

kde f_d je koeficient dynamického tření, který lze podobně jako f_s vyčíst z tabulek.

2.2 Současné simulátory mechaniky pevných těles

Simulací mechaniky pevných/tuhých těles v reálném čase se v dnešní době zabývá celá škála softwaru. Ve většině případů jde o videohry, resp. fyzikální enginy, které se ve videohrách využívají. V historii tomu tak ale nebylo – hlavní zábranou byl nedostatek výpočetního výkonu. Musíme si uvědomit, že ve hrách je potřeba v každé iteraci herního cyklu vykonat i různé další činnosti jako například vykreslování scény, reakce na uživatelské vstupy, umělá inteligence a další. Tyto činnosti jsou často časově náročné a na případnou fyzikální simulaci nezbyl výkon. Fyzika se proto výrazně zjednodušovala a obcházela různými kličkami. Používaly se prvky jako magické konstanty pro řízení pohybu, „rychlost“ nezávislá na čase nebo se interakcím s objekty (např. převrácení sudu) vyhýbalo předdefinovanou animací.

S vývojem výpočetních technologií pokročila fyzika ve hrách až do stavu, kdy pro ni existují dedikované hardwarové jednotky – PPU (Physics Processing Unit). Existují také možnosti paralelizace a využití výkonu grafické karty – GPGPU (General Purpose processing on Graphics Processing Unit), na čemž je stavěna např. technologie PhysX[1]. Často se stále využívá pouze výkonu procesoru.

Fyzikální enginy můžeme klasifikovat podle různých hledisek. Hlavní problémy, při jejichž řešení se setkáme s odlišnými implementacemi jsou:

- reprezentace objektů,
- rezoluce kolizí a kontaktů.

Objekty bývají reprezentovány dvěma způsoby:

- jako skupina částic,
- jako celá tělesa.

Metoda *reprezentace objektů částicemi* [12] rozkládá tělesa na hmotné body. V nejjednodušší podobě si můžeme představit čtverec složený ze 4 hmotných bodů v jeho vrcholech. Tyto body bychom pak navzájem pospojovali pomocí nekonečně tuhých pružin. Tuhé pružiny by zajistili potřebnou vzdálenost mezi body (délku hrany). Pružiny lze implementovat rovnicemi s omezeními (angl. constraints) ve tvaru:

$$|x_2 - x_1| = l$$

kde x_1 a x_2 jsou body dané hrany a l je požadovaná délka hrany. Pro čtverec ve dvourozměrném prostoru by tyto rovnice byly čtyři. Jakobsen [12] používá iterační přístup, kdy se opakovaně prochází omezeními pro každou hranu, čímž se sice naruší hodnoty ostatních hran, ale celkové řešení postupně konverguje ke správné hodnotě (tzv. Jacobiho, popř. Gauss-Seidelova metoda).

Při *reprezentaci objektů celými tělesy* chápeme těleso jako spojitý celek a musíme s ním tak i pohybovat. Oproti reprezentaci částicemi je zde potřeba zahrnout do pohybu i rotaci. Částice na sobě byly navzájem nezávislé a o případné otočení se postaraly rovnice s omezeními. Rovnice popisující kinematiku a dynamiku tělesa se po zahrnutí rotací podstatným způsobem zkomplikují, nicméně výhodou je, že nemusíme řešit rovnice pružin – tělesa budou držet svůj původní tvar implicitně. Další výhodou je možnost reprezentace více tvarů. Například kruh můžeme reprezentovat jako spojitý obrys, kdežto u částic bychom jej museli aproximovat několika body ležícími po obvodu.

[1] http://www.geforce.com/hardware/technology/physx

Nejčastěji se fyzikální enginy liší tím, jak nakládají s *kolizemi* a *kontakty*. *Kolizí* rozumíme srážku dvou těles s relativní rychlosti $|\boldsymbol{v}_2 - \boldsymbol{v}_1| > 0$. *Kontakt* je souvislý dotyk těles, přičemž pro relativní rychlost platí $|\boldsymbol{v}_2 - \boldsymbol{v}_1| = 0$. Všimněme si závislosti obou třecích sil z rovnic 2.15 a 2.16 na tlakové síle F_n. Síla F_n sama závisí na všech ostatních silách působících na dané těleso, a proto se výsledné síly určují obtížně. Existuje několik základních přístupů k řešení kontaktů a kolizí. V praxi se můžeme setkat i s jejich kombinacemi. Jde o následující metody:

- Metody založené na silách (tzv. *force-based*, popř. *constraint-based* metody).
- Metoda penalizační síly (tzv. *penalty force method*).
- Metody založené na impulzech (tzv. *impulse-based* metody).

2.2.1 Metody založené na silách

Metody založené na silách se snaží najít globální řešení. Pro všechny objekty současně sestaví rovnice s omezeními (angl. *constraints*), v nichž jako neznámé vystupují síly (popř. zrychlení). Výsledná soustava rovnic představuje tzv. *linear complementarity problem*, jehož řešením se zabývá optimalizace (obor matematiky). Pokud ignorujeme tření, počítají se pro kontakty pouze normálové síly. Jakmile ale zahrneme dynamické tření (což je pro věrohodnost simulace velmi důležité), dostáváme dvě constraints na každý kontaktní bod. Statické tření, které závisí na normálových silách, tento počet constraints zvýší na tři a pokud simulujeme trojrozměrný prostor přechází výsledná soustava na tzv. *nonlinear complementarity problem*.

Metody založené na silách potřebují znát přesný čas srážek a nepočítají s částečným vzájemným proniknutím, tzv. *penetrací*, dvou objektů, protože ve skutečnosti by nikdy k penetraci u tuhých těles nemělo dojít. Podstatnou nevýhodou metod založených na silách je, že výsledná soustava rovnic někdy nemusí mít řešení, neboť se například vlivem numerických chyb a penetrací těles můžeme dostat do situace, která by ani v realitě nemohla nastat. S takovouto situací by se měl simulátor umět vypořádat. Celkově jsou metody založené na silách složité, výpočetně (a tedy i časově) náročné, ale zároveň nejpřesnější. Setkáme se s nimi například v simulátoru CapSim [16], který je vytvořen v prostředí MATLAB.

2.2.2 Metoda penalizační síly

Metoda penalizační síly (penalty force method) viz [7] pracuje s kontakty jako s tlumenými pružinami. Kontakty jsou řešeny po jednom (kontakt zahrnuje dvojici těles) a pro každý z nich se snažíme najít sílu, která jednak odstraní penetraci objektů a zároveň vyřeší případný odraz. Velikost této síly určíme například jako:

$$F = max(\frac{k_p d - k_v v}{r}, 0)$$

kde d je hloubka penetrace, v je relativní rychlost těles v bodě kontaktu, r je celkový počet bodů dotyku, ve kterých je uplatněna nenulová síla, a k_p a k_v jsou vhodně zvolené koeficienty. Tyto koeficienty mohou nabývat například podoby $k_p = 100(m_1 + m_2)$ a $k_v = 50(m_1 + m_2)$, přičemž m_1 a m_2 jsou hmotnosti daných těles. Základním problémem metod penalizačních sil je právě určení obou koeficientů. Tyto koeficienty budou fungovat dobře pouze pro určité modelované situace. Řešení každého kontaktu pouze minimálně zohledňuje globální řešení a výsledek bude v určitých situacích nepřesný (např. při skládání

velkého množství objektů na sebe). Nicméně s určitými optimalizacemi viz [7] lze dosáhnout vizuálně věrohodného řešení. Metoda penalizační síly vyniká především svou jednoduchostí a rychlostí. S jejím využitím se můžeme setkat např. v robotice.

2.2.3 Metody založené na impulzech

Metody založené na impulzech na rozdíl od metod založených na silách nerozlišují kolize a kontakty – u obou aplikují impulzy. Metody založené na impulzech pracují lokálně – prochází dvojice těles v kontaktu a kalkulují potřebný impulz, který by daný kontakt korektně rozřešil. Tímto by ale mohly narušit jiný kontakt, který už byl vyřešen. Proto opakovaně procházejí všemi kontakty a postupně konvergují ke globálnímu řešení. Jedna z metod založených na impulzech zvaná *sequential impulse method*, kterou najdeme v enginu Box2D [2], je funkčně ekvivalentní iterativnímu řešení soustav rovnic s omezeními u metod založených na silách. Metody založené na impulzech jsou velmi přesné, rychlé a relativně jednoduché na pochopení. Nevýhodou je, že pracují v iteracích, jejichž počet může být obtížné určit. V současné době jsou tyto metody díky svým výhodám nejpoužívanější. Můžeme je najít například v enginech Bullet [5], Chipmunk [14] a IBDS [1].

Kapitola 3

Návrh fyzikálního simulátoru

Vytvářený simulátor má být *cross-platform*, tedy schopný běhu na více platformách (hlavně Microsoft Windows a Linux). Simulátor musí umět určitým způsobem vytvořit okno, zobrazit simulované objekty (geometrická primitiva) a přijímat uživatelské vstupy.

Jelikož simulujeme dvourozměrný prostor, bude mít velká část modelovaných fyzikálních veličin a vlastností těles dvě složky – x a y. Práci s těmito veličinami si výrazně usnadníme, pokud s nimi budeme pracovat ve vektorovém tvaru. V simulátoru proto vytvoříme matematický modul, který umožní práci s vektory a maticemi.

V návrhu se budeme zabývat pohybovými rovnicemi (kinematikou) ze sekce 2.1, k jejichž řešení na počítači využijeme numerických metod. Pro simulaci interakcí těles (dynamiky) potřebujeme navrhnout systém detekce kolizí a způsob, jak kolize rozřešit.

Dále se budeme v návrhu zabývat řízením chodu aplikace a samotné simulace. Především potřebujeme navrhnout způsob řízení času a určit pořadí, ve kterém budou prováděny jednotlivé činnosti simulátoru.

3.1 Multiplatformní implementace

Pro aplikaci jsem jako implementační jazyk zvolil C++, abych splnil požadavek multiplatformní implementace. Vzhledem k jednoduchosti vykreslovaných objektů (geometrická primitiva) není potřeba sofistikovaných grafických knihoven. Pro vykreslování objektů jsem zvolil knihovnu OpenGL hlavně kvůli tomu, že je multiplatformní a bývá implicitně nainstalována na většině operačních systémů.

K vytvoření okna na více platformách využijeme knihovnu, po níž bychom chtěli, aby umožňovala v daném okně vykreslovat OpenGL kontext. Rozhodoval jsem se mezi knihovnami GLUT a SDL [13]. Obě knihovny poskytují prostředky pro odchytávání systémových událostí. Vybral jsem knihovnu SDL proto, že má logičtější design oproti GLUTu, v němž je potřeba používat callbacky, a knihovnu SDL tak lze lépe zakomponovat do objektově orientované struktury programu. Knihovna SDL zároveň poskytuje jednoduché rozhraní pro komplikovanější grafiku (textury, různé efekty apod.) v případě, že bychom chtěli simulátor rozšířit.

3.2 Matematický modul

Tato část simulátoru bude umožňovat práci s vektory a maticemi. Výhodou práce s vektory oproti práci s jednotlivými složkami x a y je především přehlednost a srozumitelnost zdro-

jového kódu. Druhou výhodou je rozšířitelnost. Podstatná část fyzikálních rovnic má vektorový tvar a tyto rovnice bývají podobné (často totožné) pro dvojrozměrný a třírozměrný prostor. Pokud simulátor vhodně navrhneme, bude stačit provést změny pouze v matematickém modulu, některých geometrických operacích a způsobu vykreslování objektů a mohli bychom provádět simulaci v trojrozměrném prostoru.

Matematický modul bude obsahovat i třídu pro matice se dvěma řádky a dvěma sloupci. Tu využijeme například pro rotace nebo jako úložiště dvou příbuzných vektorů.

Pro vektory budeme potřebovat základní operace jako např. součet, rozdíl, součin vektoru se skalárem, přiřazení, nulování apod. Zároveň se nám bude hodit skalární součin dvou vektorů (angl. *dot product*), vektorový součin (angl. *cross product*) a doplňkové funkce jako normalizace (převod na jednotkový vektor), zjištění délky vektoru a rotace kolem bodu.

Za zmínku stojí použité jednotky. V rámci celého simulátoru zavedeme jednotný systém: metr pro délky a vzdálenosti, kilogram pro hmotnosti, sekundy pro čas a radiány pro úhly. Tímto se vyhneme jakýmkoli nedorozuměním a vidíme-li například rychlost, víme, že je uvedena v metrech za sekundu, nikoliv kilometrech za hodinu apod.

3.3 Životní cyklus simulátoru

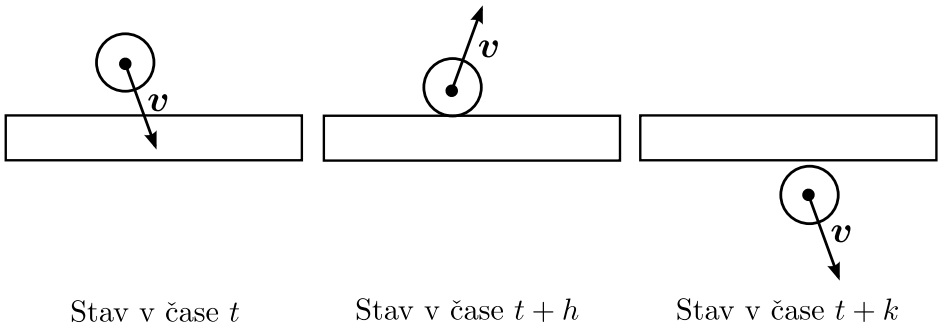
V této části bude navržen životní cyklus simulátoru, čímž rozumíme posloupnost činností prováděných od startu aplikace po její ukončení. Zaměříme se hlavně na řízení a posun času.

3.3.1 Spojitá simulace

Simulace kinematiky tuhých těles má spojitý charakter, jelikož pohyb těles určují spojité funkce. U dynamiky se navíc vyskytují diskrétní události – kolize. Takovéto události, které vzniknou, jakmile spojité veličiny dosáhnou určitých hodnot, označujeme jako *stavové události*. Pokud například simulujeme pohyb dvou těles v čase t s krokem k, mohlo by se nám stát, že ke kolizi by mělo dojít v čase $t + h$ takovém, že $h < k$, ale detekujeme ji až v čase $t + k$. Tento problém lze řešit například metodou půlení intervalů, při níž bychom se vraceli v čase, snižovali krok na polovinu a postupovali dopředu, dokud bychom opět nedetekovali kolizi, což bychom opakovali až do dosažení minimálního kroku.

Pokud bychom využili mechanismu návratu v čase, mohli bychom zjistit přesný čas, ve kterém dojde ke kolizi. Bohužel ani s využitím mechanismu návratu v čase nedosáhneme dokonalé přesnosti. Pro rychlá či malá tělesa může například nastat situace z obrázku 3.1, kdy kolizi v čase $t + k$ nedetekujeme, ale i přesto k ní mělo v čase $t + h$ dojít. Obrovskou nevýhodou mechanismu návratu v čase je paměťová a výpočetní náročnost – potřebujeme uchovávat staré i nové hodnoty všech veličin, abychom se k nim mohli vrátit, a opakovaně provádíme několik mezikroků v rámci jediného kroku. Všechny tyto mezikroky by znovu prováděly detekci kolizí, což je nejnáročnější operace celé simulace. Dále je potřeba brát v potaz, že ke kolizím dochází prakticky neustále. Tělesa setrvávající v klidu na podložce by způsobila, že by neustále docházelo k posunu s minimálním krokem. Takovouto trvalou kolizi označujeme jako *kontakt*. Mohli bychom ale zavést mechanismus, který detekuje kontakty, a snížili bychom čas, po který simulujeme minimálním krokem.

Mechanismus návratu v čase bychom mohli aplikovat *globálně* nebo *lokálně*. Při *globálním* přístupu bychom vrátili stav všech objektů. Výpočetní náročnost by sice byla obrovská, ale odhalili bychom korektně pořadí srážek při současné kolizi více objektů. Například bychom zjistili, že po rozřešení kolize, která nastala první, druhá kolize vůbec nenastane.



Obrázek 3.1: Ukázka propadnutí dvou těles

Při lokálním uplatnění mechanismu návratu v čase bychom vraceli stav pouze dvou zúčastněných objektů, což by bylo výrazně rychlejší. Bohužel za běžných situací může být v kontaktu spousta těles a při drobném pohybu kteréhokoliv z nich by všechny kontakty musely být přehodnoceny a došlo by k časovému posunu minimálním krokem pro každou dvojici těles až do dalšího ustálení. Musíme si uvědomit, že je potřeba brát v potaz právě nejhorší situace, které mohou nastat. Při nich totiž může dojít k výraznému snížení počtu snímků za vteřinu (angl. frames per second – FPS) a tyto náhlé poklesy pociťuje uživatel daleko hůře než konstantní běh na nižších FPS.

V real-time simulátorech jsem se nesetkal s implementací mechanismu návratu v čase. Některé simulátory sice hledají konkrétní čas nárazu, ale pouze jej aproximují. Pro každý objekt mohou například zkontrolovat nejbližší objekty (do určité tolerance), aproximovat jejich trajektorii a vypočítat přibližný čas nárazu s určitou přesností. Tento proces se často označuje jako *spojitá detekce kolizí* (angl. *continuous collision detection*) a s jeho implementací viz [6] se setkáme například v enginu Bullet [5]. Spojitá detekce kolizí navíc umožňuje odhalit i situaci z obrázku 3.1.

Rozhodl jsem se nevyužít spojité simulace s mechanismem návratu v čase. Pokud máme k dispozici dostatečný výkon, můžeme podobného efektu dosáhnout snížením simulačního kroku. Využijeme tedy čistě spojité simulace a nebudeme určovat přesný čas srážek těles. Nevýhodou je hlubší penetrace těles a horší přesnost.

3.3.2 Herní cyklus

Po aplikaci bychom chtěli, aby pracovala v reálném čase. Úplně přesně se reálného času držet nebudeme, neboť ani nemůžeme zajistit přítomnost dostatečných hardwarových a softwarových prostředků. Bude nám stačit, pokud bude modelový čas přibližně roven reálnému času tak, aby uživatel nepoznal rozdíl a mohl se simulátorem interagovat v čase, který se mu zdá jako reálný. Multiplatformní funkce knihovny SDL pro zjišťování času a čekání poskytují podle dokumentace [13] rozlišení v řádu milisekund, s čímž si vystačíme (v praxi bývá přesnost řádově lepší, informace v dokumentaci SDL uvádějí nejhorší možnou přesnost).

Ve hrách se uplatňují dva základní principy implementace herního cyklu [17]:

- konstantní herní rychlost,
- variabilní herní rychlost.

Oba druhy herního cyklu se liší hlavně v tom, jak se chovají při vyšší zátěži.

V případě *konstantní herní rychlosti* má aplikace vyhrazený čas, který musí odsimulovat a vykreslit. Pokud to nestihne, začne se celá hra zpomalovat – simulovaný úsek bude sice stejně dlouhý, ale v reálném čase bude trvat déle.

U *variabilní herní rychlosti* je délka kroku závislá na zátěži. Méně výkonný počítač tak bude stíhat např. 20 snímků za vteřinu a bude tedy simulovat úseky o délce 50 milisekund. Výkonnější počítač může simulovat např. 100 FPS po 10-ti milisekundách. Tento způsob má tu výhodu, že je schopný se přizpůsobit. Simulace tedy může probíhat v reálném čase i při výraznějším zpomalení. Nevýhodou je, že simulace bude při příliš velkých krocích znatelně nepřesná a může dojít k různým chybám jako například propadávání objektů.

Rozhodl jsem se zahrnout do vytvářeného simulátoru mechanismus variabilní i konstantní herní rychlosti, neboť se implementačně moc neliší. Implicitně bude nastavená variabilní herní rychlost s tím, že počet snímků za sekundu omezíme shora, abychom dosáhli stabilní hodnoty a zbytečně nevytěžovali procesor, když to není potřeba. Při dostatečném výkonu se budou obě implementace herního cyklu chovat totožně a simulace bude běžet s konstantním krokem. Omezení FPS shora zajistíme pomocí funkce `SDL_Delay`, kterou se aplikace krátkodobě uspí, pokud proběhl aktuální snímek s menším krokem (rychleji) než je hodnota minimálního kroku.

Do životního cyklu aplikace potřebujeme zařadit následující činnosti:

- Simulace jednoho kroku (pohyb těles, detekce kolizí, rozřešení kolizí).
- Zpracování událostí operačního systému jako např. uživatelské vstupy (myš, klávesnice) a žádosti o změnu velikosti okna apod.
- Volání funkce modelu `PopulateWorld`, která vytvoří objekty herního světa.
- Vykreslení objektů herního světa.

Výsledný algoritmus životního cyklu simulátoru konceptuálně popisuje Algoritmus 1.

3.4 Reprezentace těles

Ze způsobů reprezentace objektů naznačených v sekci 2.2 použijeme reprezentaci celými tělesy. V simulátoru budeme modelovat dva základní tvary těles:

- konvexní n-úhelníky (polygony),
- kruhy.

Konvexní n-úhelník má všechny úhly menší než 180° a pro různé výpočty se s ním pracuje výrazně jednodušeji než s konkávními n-úhelníky. Ve fyzikálních enginech dokonce bývá běžné, že se konkávní n-úhelníky rozkládají na několik konvexních n-úhelníků, které se následně spojují určitými silami tak, aby držely pohromadě.

Tvar n-úhelníku budeme reprezentovat seznamem vrcholů a tvar kruhu budeme vyjadřovat středem a poloměrem. Oba tvary se budou lišit především tím, jak vypočteme těžiště a jak s nimi budeme nakládat při detekci kolizí. Budeme modelovat pouze tělesa s rovnoměrným rozložením hmoty, a proto bude těžiště kruhu v jeho geometrickém středu.

Algoritmus 1: Životní cyklus simulátoru

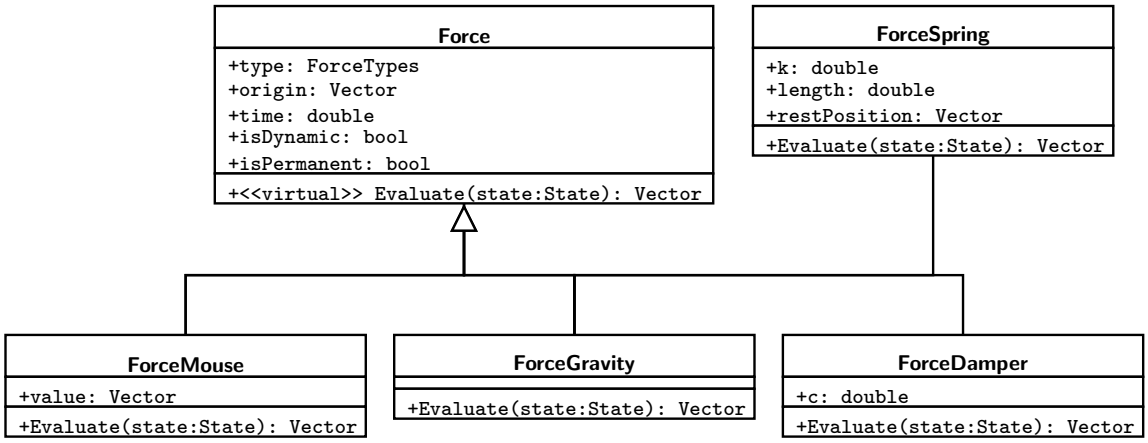
1. Inicializuj knihovnu SDL a vytvoř okno aplikace
2. Instancuj objekty modelu funkcí `PopulateWorld`
3. Čas = zjisti čas (pomocí funkce `SDL_GetTicks`)
4. Dokud uživatel přes uživatelské rozhraní nepožádá o ukončení, pokračuj. Jinak jdi na 15
5. Aktuální čas = zjisti čas
6. Krok = aktuální čas – čas
7. Čas = aktuální čas
8. Zpracuj události operačního systému
9. Simuluj krok (pohyb těles)
10. Vykresli objekty herního světa
11. Aktuální krok = zjisti čas – čas
12. Pokud je aktuální krok menší než minimální krok pokračuj, jinak jdi na 14
13. Čekej po dobu minimální krok – aktuální krok (pomocí funkce `SDL_Delay`)
14. Konec snímku, vrať se na 4
15. Konec simulace

Střed n-úhelníku vypočteme z jeho vrcholů se souřadnicemi x a y jako:

$$\begin{aligned} A &= \frac{1}{2}\sum_{i=0}^{n-1}\left(x_i y_{i+1} - x_{i+1} y_i\right) \\ C_x &= \frac{1}{6A}\sum_{i=0}^{n-1}\left(x_i + x_{i+1}\right)\left(x_i y_{i+1} - x_{i+1} y_i\right) \\ C_y &= \frac{1}{6A}\sum_{i=0}^{n-1}\left(y_i + y_{i+1}\right)\left(x_i y_{i+1} - x_{i+1} y_i\right) \end{aligned}$$

kde A je obsah n-úhelníku, C_x a C_y jsou hledané složky polohy středu n-úhelníku, n je počet vrcholů a i značí indexy vrcholů.

Pro všechna tělesa budeme potřebovat dopředu vypočítat moment setrvačnosti I. Vztah pro kruh jsme si uvedli v rovnici 2.11. Moment setrvačnosti pro libovolný konvexní n-úhelník



Obrázek 3.2: Diagram tříd zachycující dědičnost jednotlivých sil

získáme jako:

$$I = \frac{M}{6} \frac{\sum_{i=0}^{n-1} \|\boldsymbol{P}_{i+1} \times \boldsymbol{P}_i\| \left((\boldsymbol{P}_{i+1} \cdot \boldsymbol{P}_{i+1}) + ((\boldsymbol{P}_{i+1} \cdot \boldsymbol{P}_i) + ((\boldsymbol{P}_i \cdot \boldsymbol{P}_i)\right)}{\sum_{i=0}^{n-1} \|\boldsymbol{P}_{i+1} \times \boldsymbol{P}_i\|}$$

kde P_i jsou polohové vektory jednotlivých vrcholů.

V simulátoru zavedeme dva základní typy těles: *statická* a *dynamická* tělesa. Dynamická tělesa se budou moci volně pohybovat. Statická tělesa slouží jako abstrakce pro dynamická tělesa s hmotností $m \to \infty$, na něž nebudou působit žádné síly. Statická tělesa lze využít převážně pro modelování země a podobných objektů, které se nemají pohybovat, a proto je můžeme vypustit z mnoha výpočtů (např. řešení pohybových rovnic).

3.5 Řešení pohybových rovnic – Kinematika

V čase t chceme simulovat časový úsek (krok) o délce h. Cílem jednoho kroku simulace je provedení potřebných výpočtů k získání nového stavu simulovaných objektů v čase $t + h$. Pohyb způsobují síly a impulzy, z nichž lze následně vyvodit ostatní veličiny pomocí rovnic ze sekce 2.1.

3.5.1 Reprezentace sil

Ve vytvářeném simulátoru bude k dispozici síla pro pohyb těles kurzorem (myší) a sada sil ze sekce 2.1. Je ovšem žádoucí, aby šlo tuto sadu případně rozšířit a aby bylo možné nakládat se silami obecně. Proto jsem pro reprezentaci sil navrhl abstraktní třídu `Force`, která obsahuje společné vlastnosti všech sil. Abstraktní třídu `Force` budou rozšiřovat ostatní síly – ty budou obsahovat vlastní parametry a implementaci abstraktní metody `Evaluate`, která z předaného stavu daného objektu vypočte a vrátí vektor síly. Vztah mezi třídami sil zachycuje obrázek 3.2. Pseudokód metody `Evaluate` pro tíhovou sílu z rovnice 2.12 zachycuje algoritmus 2. Jako příklad komplikovanější síly si uvedeme pružinovou sílu z rovnice 2.13, jejíž metodu `Evaluate` popisuje algoritmus 3.

Algoritmus 2: Pseudokód pro vyhodnocení tíhové síly

```
Evaluate(State state) {
  return -9.80665 * state.m
}
```

Algoritmus 3: Pseudokód síly pro 1D pružinu

```
double length                                    // klidová délka pružiny
double k                                                 // tuhost pružiny
double restPosition                          // klidová pozice konce pružiny
Evaluate(State state) {
  // aktuální délka = aktuální pozice - klidový stav
  double currentLength = state.position - restPosition
  // F = -k * x
  return -k * (currentLength - length)
}
```

3.5.2 Numerická integrace pohybových rovnic

Rovnici 2.3 pro Newtonův druhý pohybový zákon můžeme zapsat jako:

$$m\frac{\mathrm{d}x(t)}{\mathrm{d}t} = F(x(t))$$

což je tzv. *obyčejná diferenciální rovnice* (dále jen ODR). Obecně platí, že síly mohou záviset na poloze tělesa x, resp. jeho rychlosti $\dot{x}$. ODR lze řešit analyticky jen obtížně, někdy vůbec. Budeme se proto věnovat numerickému řešení ODR (také označováno jako numerická integrace).

Jedna z metod pro numerické řešení ODR se nazývá *Eulerova metoda*. Eulerova metoda aproximuje vývoj proměnné y lineárně (tzn. přímkou viz obrázek) jako:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

kde y_{n+1} je nová hodnota hledané neznámé na konci kroku h, y_n je její hodnota na začátku kroku a f je funkce určující míru změny y.

Ze sil můžeme podle rovnice 2.3 určit zrychlení tělesa. Z rovnic 2.2 a 2.1 vyjádříme neznámé proměnné (rychlost a polohu) a pomocí Eulerovy metody můžeme vyjádřit jejich nové hodnoty v čase $t + h$ soustavou rovnic:

$$v_{n+1} = v_n + ha_n$$
$$r_{n+1} = r_n + hv_n$$

a podobně určíme úhlové veličiny z rovnic 2.5 a 2.6 pomocí vzorců:

$$\omega_{n+1} = \omega_n + h\varepsilon_n$$
$$\theta_{n+1} = \theta_n + h\omega_n$$

Všimněme si, že u Eulerovy metody stačí, když určíme síly (resp. zrychlení) pouze jednou během celého kroku – na jeho začátku. Standardní Eulerova metoda je explicitní, což znamená, že počítá následující stav pouze na základě současného stavu.

V případě rovnic klasické mechaniky lze použít *semi-implicitní Eulerovu metodu*, která částečně využívá současný i následující stav k výpočtu následujícího stavu. Po výpočtu rychlosti, můžeme nově vzniklou hodnotu použít pro výpočet polohy, což vyjádříme jako:

$$v_{n+1} = v_n + ha_n$$
$$r_{n+1} = r_n + hv_{n+1}$$

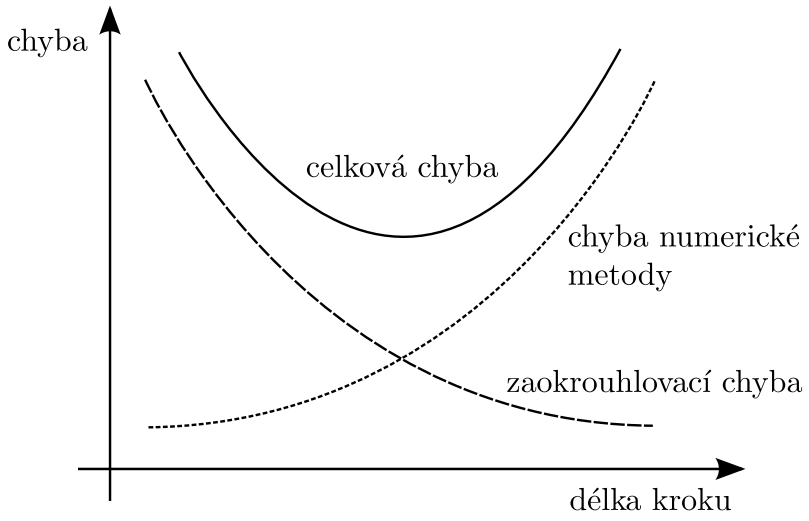
Semi-implicitní Eulerova metoda sice poskytuje podobnou přesnost jako její explicitní varianta, ale lze s ní dosáhnout lepší stability (např. u pružin). Semi-implicitní Eulerova metoda je použita například v enginu Box2D [2].

Obě Eulerovy metody jsou metody prvního řádu. Řád metody vyjadřuje její přesnost, přičemž metody vyšších řádů jsou přesnější. Z metod vyšších řádů si uvedeme jednu často používanou metodu – *Runge-Kutta čtvrtého řádu* (dále jen RK4). RK4 počítá pomocné hodnoty (body), z jejichž poměru určí výslednou hodnotu y_{n+1} na konci daného kroku h. Metodu RK4 počítáme pomocí následujících vzorců:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\ k_3 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + \frac{1}{6}h1(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{3.1}$$

kde k_1 až k_4 jsou pomocné hodnoty.

Pro numerické výpočty na počítačích platí, že snižováním kroku nemusíme vždy dosáhnout větší přesnosti. Snížením kroku sice numerická metoda bude přesnější, ale zároveň se zaokrouhlovací chyby budou šířit rychleji. Celková chyba tak může dokonce začít růst. Vztah mezi zaokrouhlovací chybou, chybou numerické metody a celkovou chybou vyjadřuje obrázek 3.3. Všechny výpočty s reálnými čísly budeme provádět s přesností datového typu `double`, abychom snížili vliv zaokrouhlovacích chyb.



Obrázek 3.3: Vztah mezi jednotlivými druhy chyb při numerické integraci

3.5.3 Výběr metody pro numerickou integraci

Pro naše účely má metoda RK4 velkou nevýhodu – vyhodnocuje funkci f opakovaně během jednoho kroku. Uvědomme si, že při kolizích a kontaktech těles závisí rychlost, popř. zrychlení na poloze *všech* ostatních těles. Bylo by tedy nutné opakovaně provádět detekci kolizí, což je v nezoptimalizované verzi operace se složitostí O^2, a rozřešení kolizí, což je také velmi náročná operace v závislosti na použitém kolizním systému (viz 2.2).

Eulerova metoda oproti RK4 předpokládá, že se síly a zrychlení během kroku h nemění. Tento předpoklad může např. pro gravitační sílu, ale síly např. u harmonických oscilátorů závisí na poloze a Eulerova metoda by pak mohla být v určitých případech velmi nestabilní. Millington [15] a Catto [4] v enginech Cyclone a Box2D používají metody prvního řádu a shodují se, že pro uplatnění ve hrách jejich přesnost stačí.

Millington [15] zmiňuje možnost použít RK4 pro všechny běžné síly mimo kolizních sil, které budou detekovány a řešeny pouze jednou v rámci celého kroku (podobně jako u Eulerovy metody). Tento přístup nabízí dobrý kompromis mezi rychlostí a přesností, a proto jsem se jej rozhodl v práci uplatnit. Implementace metody RK4 je popsána v sekci 4.1.

3.6 Kolize a kontakty těles – Dynamika

V této části bude popsána metoda rozřešení kontaktů pomocí impulzů (*sequential impulse method*) naznačená v sekci 2.2. Tuto metodu jsem vybral hlavně proto, že je velmi přesná a rychlá. Zároveň navrhneme jednoduchý systém pro detekci kolizí a zapojíme kolizní systém do simulátoru.

3.6.1 Propojení kolizního systému se zbytkem simulátoru

Jak již bylo naznačeno v sekci 3.3.1, budeme srážky zpracovávat pouze jednou za celý simulační krok. Nebudeme hledat přesný čas kolizí a zpracujeme je všechny najednou. Na začátku simulačního kroku provedeme numerickou integraci pohybových rovnic každého tělesa. Tělesa se dostanou do nových poloh a zjistíme, která z nich se protínají (detekce kolizí). Fáze detekce kolizí `DetectCollisions` sestaví seznam všech kontaktů. Výsledný seznam kontaktů bude předán systému pro rozřešení kolizí `ResolveCollisions`, který upraví pozice a rychlosti těles. Na konci kroku odstraníme síly, jejichž doba trvání byla časově omezená a právě vypršela. Pseudokód jednoho kroku simulace zachycuje algoritmus 4.

3.6.2 Detekce kolizí

Detekce kolizí je zpravidla časově nejnáročnější operace u většiny simulátorů mechaniky tuhých těles. Při simulaci v trojrozměrném prostoru je časová náročnost ještě znatelnější. V naivní verzi můžeme pro každé těleso provádět kontrolu se všemi ostatními tělesy a porovnávat jejich přesnou geometrii. Časová náročnost detekčních algoritmů se snižuje nejčastěji dvěma způsoby:

- Redukuje se počet kontrol, které se mají provést – např. organizací objektů do skupin a hierarchií, které se navzájem nedotýkají.
- Zjednoduší se průběh kontroly např. provedením detekce kolizí na zjednodušené geometrii (opsané kružnice či obdélníky).

Algoritmus 4: Pseudokód jednoho kroku simulace

```
Simulate(dt) {
  foreach(object) {
    object.RK4_Step(dt)
  }
  contacts = DetectCollisions()
  ResolveCollisions(contacts)
  foreach(object) {
    foreach(force) {
      force.time = force.time - dt
      if (force.time < 0)
        delete force
} } }
```

V praxi se nejčastěji používá vícefázový systém detekce kolizí. V první fázi můžeme například provést naivní kontrolu všech dvojic těles se zjednodušenou geometrií a ve druhé fázi můžeme kontrolovat zbývající dvojice s přesnou geometrii. Podrobněji se detekcí kolizí zabývá [8] a [15]. V rámci této práce použijeme naivní systém detekce kolizí bez jakýchkoli optimalizací.

Pro reprezentaci jednoho kontaktu vytvoříme třídu `Contact`, která bude pro danou dvojici těles uchovávat informace o srážce. Hlavními informacemi, které o srážce potřebujeme zjistit, jsou:

- bod (popř. body) dotyku P,
- normála $\boldsymbol{n}$,
- hloubka penetrace d.

Neznáme-li přesný čas kolize, nemůžeme určit ani přesný bod, ve kterém se tělesa střetla. Jako bod dotyku proto budeme brát nejhlouběji zanořený bod jednoho z těles. Tato aproximace nemusí být vždy správná, ale pro naše účely bude stačit – čím menší bude krok a rychlosti těles, tím bude tato aproximace přesnější. Normálu $\boldsymbol{n}$ určíme jako kolmici k povrchu jednoho z těles – pokud vrchol tělesa B prostoupí hranou tělesa A (viz obrázek 3.4a), budeme pracovat s normálou hrany tělesa A (u tělesa B bychom nevěděli, kterou hranu použít). Pokud se tělesa dotýkají hranami jako na obrázku 3.4b, můžeme určit normálu vůči povrchu kteréhokoli z nich. Hloubku penetrace d určíme jako vzdálenost ve směru normály, kterou musí bod dotyku P urazit, aby tělesa přestala být v kontaktu.

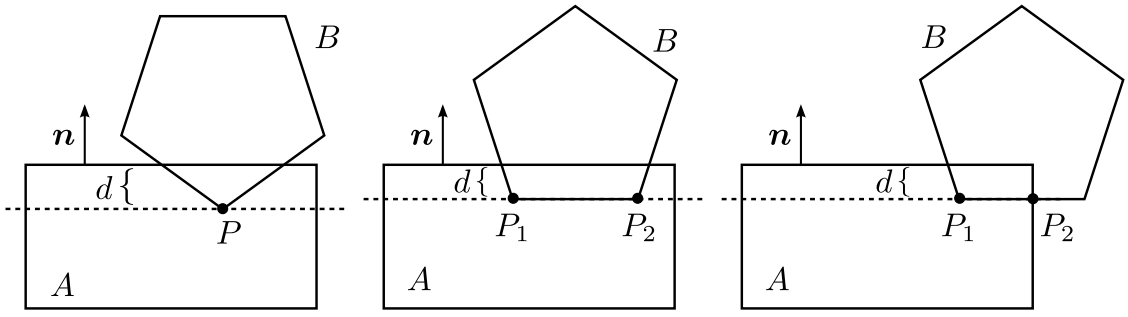
Algoritmy detekce kolizí se budou lišit pro různé kombinace těles ze sekce 3.4:

- kolize dvou kruhů
- kolize n-úhelníku a kruhu
- kolize dvou n-úhelníků

Ke *kolizi dvou kruhů* dojde, pokud je součet jejich poloměrů větší než vzdálenost jejich středů. Bod dotyku leží na spojnici středů ve vzdálenosti jednoho z poloměrů.

Při detekci *kolize n-úhelníku a kruhu* stačí postupně projít všechny hrany n-úhelníku a najít bod, který je nejblíže středu kruhu.

Nejsložitější je detekce *kolize dvou n-úhelníků*, neboť existuje více způsobů, jak do sebe tyto objekty mohou zapadnout viz série obrázků 3.4. Dva n-úhelníky se mohou dotýkat hranami a takový dotek detekujeme až ve stavech zachycených na obrázcích 3.4b a 3.4c (hloubka penetrace d je přehnaná pro lepší ilustraci). Abychom detekovali dotyk hranami, zkontrolujeme, zda jsou příslušné hrany rovnoběžné. V případě dotyku hranami hledáme body P_1 a P_2, což jsou krajní body úsečky dotyku. Pro každý z bodů P_1 a P_2 můžeme generovat vlastní kontakt a lze s nimi v systému rozřešení kolizí nakládat odděleně.



Obrázek 3.4: Příklady kolizí dvou n-úhelníků A a B

3.6.3 Výpočet normálového impulzu

Reálné objekty se při srážkách postupně deformují v místě dotyku. Velikost této deformace závisí na materiálech a hybnostech zúčastněných těles. V obou tělesech postupně rostou vnitřní síly, které se snaží (alespoň částečně) vrátit tělesa do jejich původního stavu. Jakmile velikost vnitřních sil převýší síly, které způsobují deformaci, dojde k postupnému oddělení objektů a může dojít i k odrazu. U tuhých těles je situace odlišná – nelze je zdeformovat, a proto kolize musí proběhnout v čase $t \to 0$. Vnitřní síly, které vznikají u reálných těles, nahradíme u tuhých těles impulzy, jelikož má dojít k okamžitým změnám rychlostí.

Při srážkách dvou těles nás bude zajímat relativní rychlost těles v bodě dotyku P. Tento bod existuje dvakrát (jednou na každém tělese) a v moment dotyku obě jeho verze splývají. Bod P se na tělese A pohybuje jednak lineárně (stejně jako těžiště), ale i otáčivým pohybem, jenž převedeme na lineární pohyb pomocí vztahu 2.7. Rychlost tělesa A v bodě P je pak dána jako:

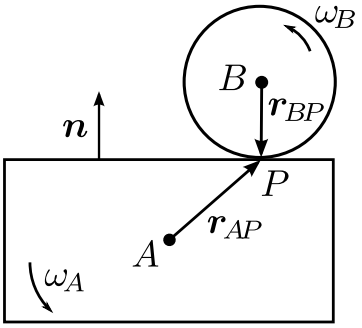
$$\boldsymbol{v}_{AP} = \boldsymbol{v}_A + \boldsymbol{\omega}_A \times \boldsymbol{r}_{AP} \tag{3.2}$$

kde $\boldsymbol{r}_{AP}$ je spojnice těžiště tělesa A a bodu dotyku P. Relativní rychlost těles A a B v bodě dotyku P pak určíme jako:

$$\boldsymbol{v}_r = \boldsymbol{v}_{BP} - \boldsymbol{v}_{AP} \tag{3.3}$$

kde $\boldsymbol{v}_{AP}$ a $\boldsymbol{v}_{BP}$ jsou rychlosti obou těles v bodě P. Nejprve budeme zkoumat srážky pouze ve směru tzv. *normály*. Normálou rozumíme jednotkový vektor $\boldsymbol{n}$ kolmý k povrchu jednoho z těles v bodě dotyku. Relativní rychlost ve směru normály určíme jako $\boldsymbol{v}_r \cdot \boldsymbol{n}$. Příklad popisované srážky najdeme na obrázku 3.5.

Abychom byli schopni určit, zda se tělesa pohybují od sebe nebo k sobě, musíme zavést pro normálu a relativní rychlosti jednotnou konvenci. Normálám budeme přiřazovat směr od A k B a relativní rychlost budeme určovat podle rovnice 3.3, pak bude platit, že:



Obrázek 3.5: Srážka těles dvou těles v bodě P

- Tělesa s $\boldsymbol{v}_r \cdot \boldsymbol{n} > 0$ se vzdalují.
- Pokud platí $\boldsymbol{v}_r \cdot \boldsymbol{n} = 0$, tělesa jsou v kontaktu.
- Při $\boldsymbol{v}_r \cdot \boldsymbol{n} < 0$ se tělesa přibližují.

Jelikož chceme, aby se srážky tuhých těles chovaly podobně jako srážky těles v realitě, musíme určitým způsobem zohlednit i materiálové vlastnosti těles. K tomu využijeme *Newtonův zákon restituce*, který má tvar:

$$\boldsymbol{v}'_r = -k\boldsymbol{v}_r \tag{3.4}$$

kde v'_r je relativní rychlost těles po srážce, v_r je relativní rychlost těles před srážkou a k je *součinitel restituce*. Součinitel restituce udává poměr mezi rychlostmi před srážkou a po srážce. Pro *dokonale pružnou srážku* má součinitel restituce hodnotu 1. Míček dopadající na zem by se při dokonale pružné srážce odrazil do své původní výšky. Pokud má součinitel restituce hodnotu 0, hovoříme o *dokonale nepružné srážce*, po níž zůstávají tělesa v dotyku.

Pro rozřešení kontaktu těles A a B hledáme takový impulz J_n, který změní rychlosti před srážkou na rychlosti po srážce podle rovnice 3.4. Směr hledaného impulzu známe – na těleso B bude impulz působit ve směru normály (ta vede od A k B) a na těleso A bude podle zákona akce a reakce působit stejně velký impulz opačného směru. Tělesa A a B si při srážce předají hybnost $\Delta\boldsymbol{p}$, kterou podle rovnice 2.4 způsobí právě impulz J_n. Vliv impulzu J_n na rychlosti můžeme zapsat jako:

$$\begin{aligned} \boldsymbol{v}'_A &= \boldsymbol{v}_A - \frac{\Delta\boldsymbol{p}}{M_A} = \boldsymbol{v}_A - \frac{\boldsymbol{J}_n}{M_A} \\ \boldsymbol{v}'_B &= \boldsymbol{v}_B + \frac{\Delta\boldsymbol{p}}{M_B} = \boldsymbol{v}_B + \frac{\boldsymbol{J}_n}{M_B} \end{aligned} \tag{3.5}$$

kde $\boldsymbol{v}'_A$ a $\boldsymbol{v}'_B$ jsou rychlosti po srážce. Impulz J_n nepůsobí v těžišti, ale v bodě P, a proto vyvolá i změny úhlových rychlostí. Nahradíme-li sílu v rovnici 2.8 impulzem, můžeme pomocí rovnice 2.9 vyjádřit úhlové rychlosti po srážce jako:

$$\begin{aligned} \boldsymbol{\omega}'_A &= \boldsymbol{\omega}_A - \frac{\boldsymbol{J}_n}{I_A} \\ \boldsymbol{\omega}'_B &= \boldsymbol{\omega}_B + \frac{\boldsymbol{J}_n}{I_B} \end{aligned} \tag{3.6}$$

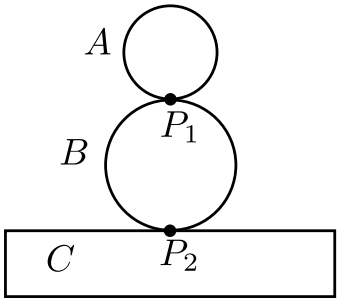
Zbývá nám pouze zjistit velikost impulzu J_n. Impulz J_n můžeme vyjádřit ze vztahů 3.5, 3.6, 3.4 a 3.3 jako:

$$J_n = \frac{-(1+k)\,\boldsymbol{v}_r \cdot \boldsymbol{n}}{\left[\boldsymbol{n}\left(\frac{1}{M_A}+\frac{1}{M_B}\right)+\frac{\boldsymbol{r}_{AP}\times\boldsymbol{n}\times\boldsymbol{r}_{AP}}{I_A}+\frac{\boldsymbol{r}_{BP}\times\boldsymbol{n}\times\boldsymbol{r}_{BP}}{I_B}\right]\cdot\boldsymbol{n}} \tag{3.7}$$

Podrobné odvození tohoto vztahu najdeme v [11] a [10]. Všechny veličiny ze vztahu pro normálový impulz v okamžiku kolize známe. Po výpočtu J_n můžeme jeho hodnotu dosadit do vzorců 3.5 a 3.6. Získáme lineární a úhlové rychlosti obou těles po srážce, čímž je kontakt rozřešen.

3.6.4 Iterativní rozřešení kolizí a kontaktů

Pokud bychom prošli všechny kontakty pouze jednou a rozřešili je normálovým impulzem z rovnice 3.7, pozorovali bychom nepřesné výsledky při vícenásobném kontaktu jednoho tělesa. Příklad takové situace zachycuje obrázek 3.6 – kontakty v bodech P_1 a P_2 o sobě neví a nezávisle na tom, který rozřešíme dříve, budou mít tělesa A i B špatně přidělenou rychlost. Propagace výsledků jednotlivých kontaktů dosáhneme tak, že všechny kontakty projdeme několikrát. Pokud bychom ale při každé iteraci prováděli výpočet z rovnice 3.4, aplikovali bychom opakovaně koeficient restituce a všechny kontakty by postupně konvergovaly ke stavu pro restituci $k = 0$. V metodě `Prepare` proto dopředu vypočítáme relativní rychlost v_k, ke které má daný kontakt konvergovat, což je relativní rychlost, kterou bychom určili při první iteraci. Výpočet a aplikaci normálového impulzu J_n provedeme v metodě `Resolve`, ve které musíme od relativní rychlosti v_r odečíst předpočítanou rychlost v_k.



Obrázek 3.6: Vícenásobný kontakt tělesa B s tělesem A a se zemí C

Jelikož neurčujeme přesný bod srážky, bude docházet k částečnému vzájemnému proniknutí (penetraci) těles. Aplikací impulzů v metodě `Resolve` se změní pouze rychlosti těles a tělesa do sebe budou stále zanořená. Penetraci odstraníme v metodě `RemovePenetration` posunutím těles ve směru normály do prvního bodu, ve kterém se tělesa neprotínají. Tento způsob odstranění penetrace je jednoduchý a v některých situacích nepřesný. Přesnější metody odstranění penetrace, které zohledňují i rotaci nebo rychlosti, popisuje [15]. Tyto přesnější způsoby odstranění penetrace se bohužel setkávají se stejnými problémy jako metoda posunu ve směru normály. Pokud bychom situaci z obrázku 3.6 rozšířili více tělesy stavěnými např. do tvaru pyramidy, i přesnější způsoby odstranění penetrace by způsobily drobný pohyb těles, přestože tělesa mají být v klidu. V takovém případě bychom dosáhli nejlepších výsledků detekcí přesného času srážek. V simulátoru budeme odstraňovat penetraci lineárním posunutím těles ve směru normály.

Algoritmus 5: Pseudokód iterativního rozřešení kolizí a kontaktů

```
ResolveContacts(Contacts) {
  foreach(contact in Contacts) {
    contact.Prepare()
    contact.RemovePenetration()
  }
  for(i = 0; i < iterations; i++) {
    foreach(contact in Contacts)
      contact.Resolve()
  }
}
```

Výsledný algoritmus iterativního rozřešení kolizí popisuje Algoritmus 5.

Millington [15] navrhuje upravenou verzi iteračního algoritmu, v níž prochází kontakty od těch s nejvyšší relativní rychlostí $v_r - v_k$. Tím lze dosáhnout rychlejší konvergence. Bohužel abychom mohli kontakty projít v určitém pořadí, museli bychom opakovaně hledat maximum a přepočítávat rozdíl $v_r - v_k$ pro každý kontakt, který s aktuálně rozřešeným kontaktem sdílí alespoň jedno těleso. Millingtonovu verzi algoritmu 5 jsem zkoušel implementovat, ale nepozoroval jsem žádné výrazné zlepšení. Změny rychlostí se sice propagují rychleji, nicméně upravený algoritmus stihne provést za stejnou dobu méně iterací.

3.6.5 Výpočet třecího impulzu

Pro daný kontakt budeme hledat třecí impulz J_t. Třecí síly působí ve směru tečny k povrchu tělesa. Zavedeme proto tečný vektor $\boldsymbol{t}$ kolmý k normále, jehož směr bude roven směru tečné složky relativní rychlosti v_r. Výsledný impulz J_t pak musí působit proti směru vektoru $\boldsymbol{t}$. Známe-li velikost normálového impulzu J_n, můžeme podle rovnice 2.16 určit velikost dynamického tření jako:

$$J_{t,dyn} = f_d J_n \tag{3.8}$$

Statické tření je komplikovanější – hledáme právě tak velký impulz $J_{t,stat}$, aby vyrovnal jakýkoli tečný pohyb mezi tělesy. Podobný koncept jsme již jednou uplatnili v sekci 3.6.3. Normálový impulz jsme vytvářeli právě tak velký, aby omezil pohyb jednoho tělesa směrem do druhého. Pro statické tření můžeme použít stejný postup a provedeme v podstatě odraz s restitucí $k = 0$ (chceme pouze vyrovnat pohyb, nic víc) ve směru tečny. Vztah pro impulz statického tření můžeme zapsat následovně:

$$J_{t,stat} = \frac{-\boldsymbol{v}_r \cdot \boldsymbol{t}}{\left[\boldsymbol{t}\left(\frac{1}{M_A} + \frac{1}{M_B}\right) + \frac{\boldsymbol{r}_{AP} \times \boldsymbol{t} \times \boldsymbol{r}_{AP}}{I_A} + \frac{\boldsymbol{r}_{BP} \times \boldsymbol{t} \times \boldsymbol{r}_{BP}}{I_B}\right] \cdot \boldsymbol{t}} \tag{3.9}$$

Statické tření je nutno omezit podle rovnice 2.15 na maximální přípustnou hodnotu:

$$J_{t,max} = f_s J_n \tag{3.10}$$

Zbývá pouze určit, který z impulzů 3.8, 3.9 a 3.10 použít. Pokud si na začátku kroku uložíme hodnoty lineárních a úhlových rychlostí těles, můžeme při rozřešení kontaktu spočítat relativní rychlost, kterou tělesa měla na začátku kroku. Bude-li relativní rychlost menší

než určitá hodnota ϵ (pod níž prohlásíme rychlost jako nulovou), byla tělesa vůči sobě v klidu a použijeme rovnice pro statické tření.

Velikost impulzu J_t v případě rovnic 3.8 a 3.10 závisí na velikosti normálového impulzu J_n. Jelikož v každé iteraci Algoritmu 5 velikosti korektivních impulzů J_n klesají (konvergují k nule), nesmíme při výpočtu tření v aktuální iteraci vycházet pouze z aktuální hodnoty J_n. Je potřeba ukládat historii impulzů a provádět výpočty tření s celkovým akumulovaným impulzem [3].

Kapitola 4

Detaily implementace simulátoru

Tato kapitola popisuje některé zajímavé a důležité části implementace.

Architektura simulátoru je založena na návrhovém vzoru `Model-View-Controller`. `Model` slouží jako úložiště simulovaných objektů a zahrnuje funkci `PopulateWorld`, která obsahuje programovou definici herního světa (počáteční pozice, tvary a další vlastnosti objektů). `View` se stará o vykreslování objektů a práci s kamerou. `Controller` řídí simulaci, zahrnuje algoritmus životního cyklu simulátoru a propojuje `Model` a `View`.

4.1 Numerická integrace

Simulátor používá pro numerickou integraci pohybových rovnic metodu Runge-Kutta čtvrtého řádu. Pro implementaci metody RK4 vytvoříme třídy `State` a `Derivative` pro stav a derivace jeho veličin. Dále budeme potřebovat funkci `GetDerivative`, která na základě předaného stavu a jeho derivací zjistí nový stav, z něhož určí síly pomocí jejich metod `Evaluate`, a vrátí nové derivace. Samotná funkce `RK4_Step` pro jeden krok metody RK4 bude opakovaně počítat nové derivace pro pomocné body přes funkci `GetDerivative` a z váženého průměru těchto derivací určí konečný stav.

Algoritmus 6 zachycuje jeden krok metody RK4 pro lineární pohyb (úhlové veličiny jsou vypuštěny, neboť se integrují identicky). Na řádcích 16-17 se pro dané těleso určuje suma všech sil, z níž se na řádku 20 určí hodnota zrychlení (derivace rychlosti) pro nový stav. Řádky 25-31 zachycují postupný výpočet pomocných bodů a jejich váženého průměru tak, jak bylo uvedeno v soustavě rovnic 3.1.

V simulátoru je implementována i Eulerova metoda především pro testovací a porovnávací účely. Eulerova metoda zaostává i pro jednoduché simulace jako např. simulace volného pádu, kterou znázorňuje tabulka 4.1. V tabulce si zároveň můžeme všimnout zaokrouhlovacích chyb v posledních cifrách pozic y.

4.2 Uživatelské rozhraní

Uživatelské rozhraní simulátoru je jednoduché a má hlavně demonstrovat možnost zasahovat do simulace za běhu. Herní svět (model), který se zobrazí po spuštění aplikace, lze zadat programově. Následně může uživatel za běhu přidávat či mazat objekty, měnit statické objekty na dynamické (a naopak) a působit na tělesa silami pomocí myši. Uživatel má možnost simulaci pozastavit, přičemž během této pauzy lze stále zasahovat do modelu. V simulátoru lze posouvat kameru a pracovat na rozsáhlém prostoru.

Algoritmus 6: Metoda Runge-Kutta čtvrtého řádu

```
1  struct State {
2    Vector r;                                                    // poloha
3    Vector v;                                                   // rychlost
4    double m;                                                   // hmotnost
5  };
6  struct Derivative {
7    Vector dr;                                                  // rychlost
8    Vector dv;                                                 // zrychleni
9  };
10 Derivative GetDerivative(State state, Derivative deriv, double dt) {
11   State newState;
12   newState.r = state.r + deriv.dr * dt;
13   newState.v = state.v + deriv.dv * dt;
14   newState.m = state.m;

15   Vector forceSum(0.0, 0.0);
16   for(ForceIterator i = forces.begin(); i != forces.end(); ++i)
17     forceSum += (*i)->Evaluate(newState);

18   Derivative result;
19   result.dr = newState.v;
20   result.dv = forceSum / newState.m;
21   return result;
22 }
23 void RK4_Step(double dt, State &state) {
24   // výpočet pomocných bodů
25   Derivative k1 = GetDerivative(state, CurrentDerivative(), 0.0);
26   Derivative k2 = GetDerivative(state, k1, 0.5 * dt);
27   Derivative k3 = GetDerivative(state, k2, 0.5 * dt);
28   Derivative k4 = GetDerivative(state, k3, dt)

29   // celkové změny = vážený průměr pomocných bodů
30   Vector dr = (k1.dr + (k2.dr + k3.dr) * 2.0 + k4.dr) / 6.0 * dt;
31   Vector dv = (k1.dv + (k2.dv + k3.dv) * 2.0 + k4.dv) / 6.0 * dt;

32   // uplatnění změn => přechod do nového stavu
33   state.r += dr;
34   state.v += dv;
35 }
```

Metoda	Krok [s]	Výsledná pozice y [m]
analyticky	–	-490,33250000000000
RK4	1	-490,33249999999998
RK4	0,1	-490,33250000000027
Eulerova	0,1	-485,42917500000027
Eulerova	0,01	-489,84216750000547
Eulerova	0,001	-490,28346675005253

Tabulka 4.1: Simulace 10-ti sekund volného pádu s počáteční pozicí $y_0 = 0$

Uživatelské vstupy (a další události operačního systému) jsou zachytávány pomocí funkce `SDL_PollEvent` knihovny SDL. Pomocí funkce `SDL_PollEvent` čteme v cyklu události třídy `SDL_Event`, které můžeme podle jejich typu rozčlenit a zpracovávat. Simulátor reaguje na následující typy událostí:

- `SDL_QUIT`: Žádost o ukončení aplikace.
- `SDL_MOUSEBUTTONDOWN`: Stisknutí tlačítka myši.
- `SDL_MOUSEBUTTONUP`: Uvolnění tlačítka myši.
- `SDL_MOUSEMOTION`: Pohyb myši.
- `SDL_KEYDOWN`: Stisk klávesy.
- `SDL_KEYUP`: Uvolnění klávesy.
- `SDL_VIDEORESIZE`: Změna velikosti okna.

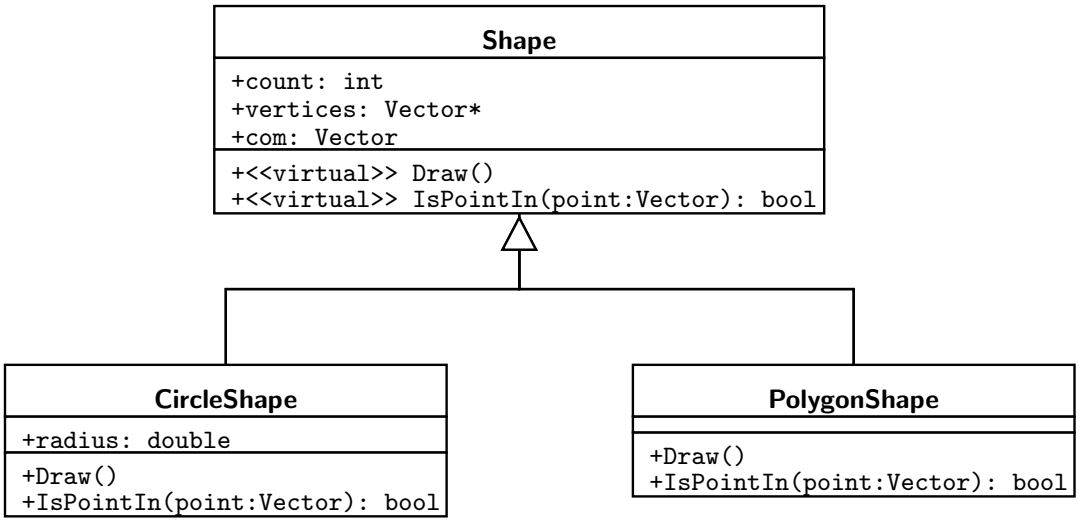
4.3 Vykreslování těles

Pro projekci těles do dvourozměrného prostoru (plochy) využijeme funkci knihovny GLU (nadstavba nad OpenGL) – `gluOrtho2D`. Funkce `glViewport` se stará o převod mezi souřadnicemi OpenGL a pixely okna. Pomocí funkce `gluOrtho2D` můžeme namapovat souřadnice OpenGL na nové jednotky, kterými budou metry. Jelikož vstupy uživatelského okna přijímáme v pixelech zavedeme převodní systém z pixelů na metry. Nezávisle na velikosti okna stanovíme, že jeho výška bude rovna konstantě V v metrech. Pokud je H výška klientské části okna v pixelech, určíme hodnotu jednoho metru jako $\frac{H}{V}$ pixelů. Šířku zobrazované plochy v metrech dopočteme tak, aby byl zachován poměr výšky a šířky okna.

Dle sekce 3.4 existují v simulátoru dva typy těles. Základ pro oba druhy těles tvoří abstraktní třída `Shape`, kterou rozšiřuje třída pro kruh `CircleShape` a třída pro n-úhelník `PolygonShape` viz obrázek 4.1.

Metoda `Draw` slouží k vykreslování těles. N-úhelník je vykreslen pomocí OpenGL funkce `glBegin(GL_POLYGON)`, přičemž jednotlivé vrcholy jsou zadány funkcí `glVertex2d`. Tvar kruhu lze zobrazit například jako sekvenci trojúhelníků skládaných do vějíře přes funkci `glBegin(GL_TRIANGLE_FAN)`.

Funkce `IsPointIn` třídy `Shape` slouží ke zjištění, zda se určitý bod P nachází uvnitř tělesa `Shape`. Předaný bod P má polohu zadanou v metrech. Metoda `IsPointIn` slouží



Obrázek 4.1: Diagram tříd zachycující vztah mezi jednotlivými druhy těles

například pro vstup z uživatelského rozhraní, kdy se po stisku tlačítka myši projdou všechny objekty a zjistí se, na které těleso uživatel klikl. Kliknutím na těleso lze například aplikovat sílu `ForceMouse`.

Bod P se nachází v kruhu s těžištěm T a poloměrem R, pokud platí:

$$|\overrightarrow{TP}| \leq R$$

Máme-li vrcholy konvexního n-úhelníku seřazeny v protisměru hodinových ručiček, lze pro určení polohy bodu P použít vektorový součin. Pokud pro každý vrchol A a jeho následující vrchol B platí:

$$\overrightarrow{AB} \times \overrightarrow{AP} > 0$$

pak se bod P nachází uvnitř n-úhelníku, neboť vektorový součin hrany AB a spojnice AP je kladný, pouze pokud je bod P v levé polorovině hrany AB. Jestliže vrcholy procházíme v protisměru hodinových ručiček, průnik jejich levých polorovin vyplní právě obsah celého n-úhelníku.

4.4 Kolize a kontakty těles

Na konci simulačního kroku (funkce `Simulate`) dochází ke zpracování kolizí. Kolize a kontakty těles jsou detekovány v metodě `DetectCollisions`. Funkce `DetectCollisions` prochází všechny dvojice těles. Pro každou dvojici nejprve zkontroluje, zda-li nejsou obě tělesa ve dvojici statická (příznak `isDynamic` má hodnotu `false`). Jestliže jsou obě tělesa statická, není potřeba se kolizí těchto těles dále zabývat. Podle typů těles je pak volána konkretní funkce pro detekci kolizí:

- `CircleCircleCollision` pro kolizi dvou kruhů,
- `PolygonCircleCollision` pro srážku n-úhelníku a kruhu,
- `PolygonPolygonCollision` pro kolize dvou n-úhelníků.

Systém detekce kolizí využívá pomocnou funkci `ClosestEdgePoint` pro určení bodu Q, který je nejblíže bodu P a zároveň leží na hraně AB. Bod Q najdeme jako:

$$k = \min\left(\max\left(\frac{\overrightarrow{PB}\cdot\overrightarrow{AB}}{\overrightarrow{AB}\cdot\overrightarrow{AB}}, 1\right), 0\right)$$
$$\boldsymbol{Q} = k\boldsymbol{A} + (1-k)\boldsymbol{B}$$

Nejsložitější je zjištění kolize dvou n-úhelníků, protože se n-úhelníky mohou dotýkat bodem nebo úsečkou (viz sekce 3.6.2), a proto je funkce `PolygonPolygonCollision` rozdělena na dvě části. Metodou `EdgeEdgeCollision` detekujeme dotyk úsečkou viz obrázky 3.4b a 3.4c. K dotyku úsečkou dojde, pokud pro alespoň jednu kombinaci hran obou n-úhelníků platí, že jsou rovnoběžné a že se dva libovolné vrcholy z těchto hran nacházejí uvnitř druhého tělesa. Funkce `PointInPolygonDetection` odhaluje dotyk jedním vrcholem viz obrázek 3.4a.

Nalezené kolize zpracovává funkce `ResolveCollisions`. Informace o jednotlivých kolizích reprezentuje třída `Contact`, která obsahuje:

- `first`, `second` – Ukazatele na obě tělesa.
- `point` – Bod dotyku.
- `normal` – Vektor normály.
- `penetration` – Hloubka penetrace.
- `restitution` – Hodnota součinitele restituce.
- `desiredVelocity` – Předpokládaná relativní rychlost těles po srážce v_k (viz sekce 3.6.4).
- `Prepare` – Metoda pro výpočet rychlosti `desiredVelocity`.
- `Resolve` – Metoda pro výpočet normálového a třecího impulzu (J_n a J_t).
- `RemovePenetration` – Metoda pro odstranění penetrace lineárním posunem těles ve směru normály.

Kapitola 5

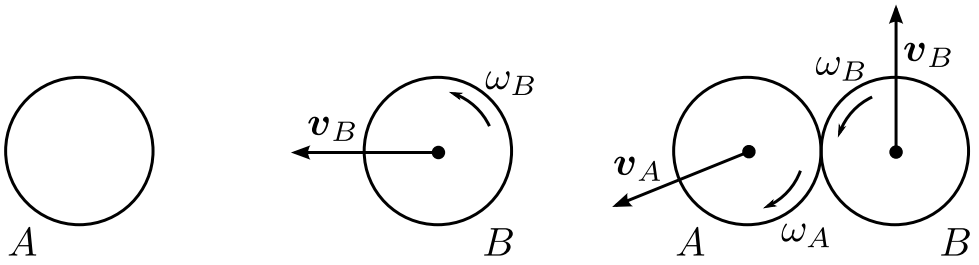
Ukázky z implementovaného simulátoru

Tato kapitola prezentuje sadu demonstračních modelů, na kterých jsou předvedeny vlastnosti simulátoru. Jednotlivé ukázky jsou dostupné na přiloženém CD formou programu i videonahrávky.

5.1 Experiment 1

Cílem tohoto experimentu je prokázat správnost chování kolizí – tedy detekce kolizí a správné rozřešení srážky ve směru normály i tečny (tření). Videozáznam experimentu najdeme v souboru `experiment1.avi`.

Model obsahuje dva kruhy s hmotnostmi $M_A = M_B = 1$ kg a poloměry $R_A = R_B = 2$ m. V čase t_0 bude kruh A v klidu a kruh B bude mít lineární rychlost $\boldsymbol{v}_B = (-1; 0)$ m a úhlovou rychlost $\omega_B = 2$ rad.s^{-1}. Pro ilustraci nebudeme uvažovat gravitační sílu. Srážka proběhne s restitucí $k = 1$ a koeficientem dynamického tření $f_d = 0,5$. Počáteční situaci znázorňuje obrázek 5.1a.



(a) Před srážkou (b) Moment srážky

Obrázek 5.1: Experiment 1 – Srážka dvou kruhů

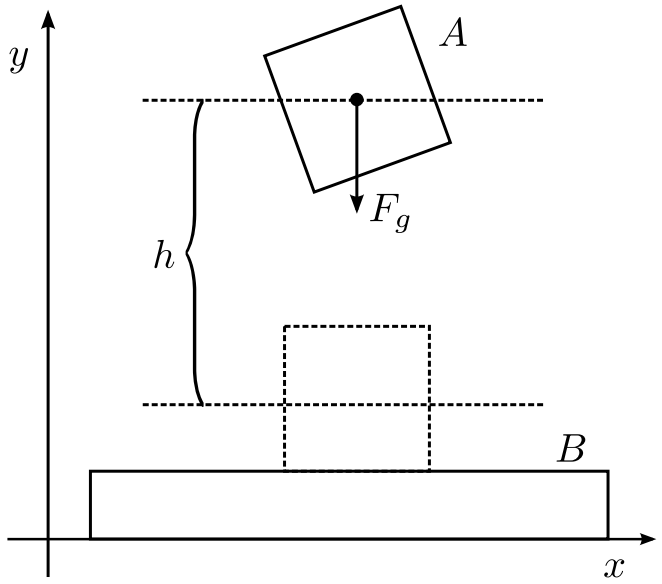
Jelikož je restituce $k = 1$ a obě tělesa mají stejnou hmotnost dojde k úplnému předání hybnosti a těleso A získá celou x-ovou složku rychlosti tělesa B, přičemž těleso B zastaví. Pokud bychom ignorovali tření, nevznikl by v tečné složce (y) žádný pohyb nezávisle na tom, jakou úhlovou rychlost by měla obě tělesa. Zahrneme-li tření, rotace tělesa B proti směru hodinových ručiček pří nárazu způsobí rotaci tělesa A ve směru hodinových ručiček. Dynamické tření závisí na normálovém impulzu a s koeficientem dynamického tření $f_d = 0,5$

vyvolá poloviční změnu rychlostí než tomu bylo v x-ové složce. Těleso A bude mít po srážce rychlost $\boldsymbol{v}'_A = (-1; -0,5)$ m a těleso B bude mít rychlost $\boldsymbol{v}'_B = (0; 0,5)$ m. Přesně tyto hodnoty jsem naměřil i v simulátoru. Směr pohybu po srážce znázorňuje obrázek 5.1b.

5.2 Experiment 2

Druhá ukázka demonstruje zachování energie a přeměny lineární rychlosti na úhlovou rychlost (a naopak) při srážkách. Video experimentu je v souboru `experiment2.avi`.

Zachování energie demonstrujeme na čtverci A, který upustíme z výšky h na vodorovnou podložku B. Nebudeme uvažovat tření, aby se čtverec pohyboval pouze ve směru osy y. Pokud srážka proběhne s restitucí $k = 1$ a čtverec upustíme tak, aby se srážkou neroztočil, odrazí se do původní výšky h a odrážel by se do nekonečna při zachování energie (perpetuum mobile). Jelikož chceme zároveň demonstrovat, že se při srážkách přeměňují lineární a úhlové veličiny, vychýlíme čtverec tak, aby při se při odrazu roztočil. Počáteční stav modelu zachycuje obrázek 5.2.



Obrázek 5.2: Experiment 2 – Opakovaná srážka čtverce A s podložkou B

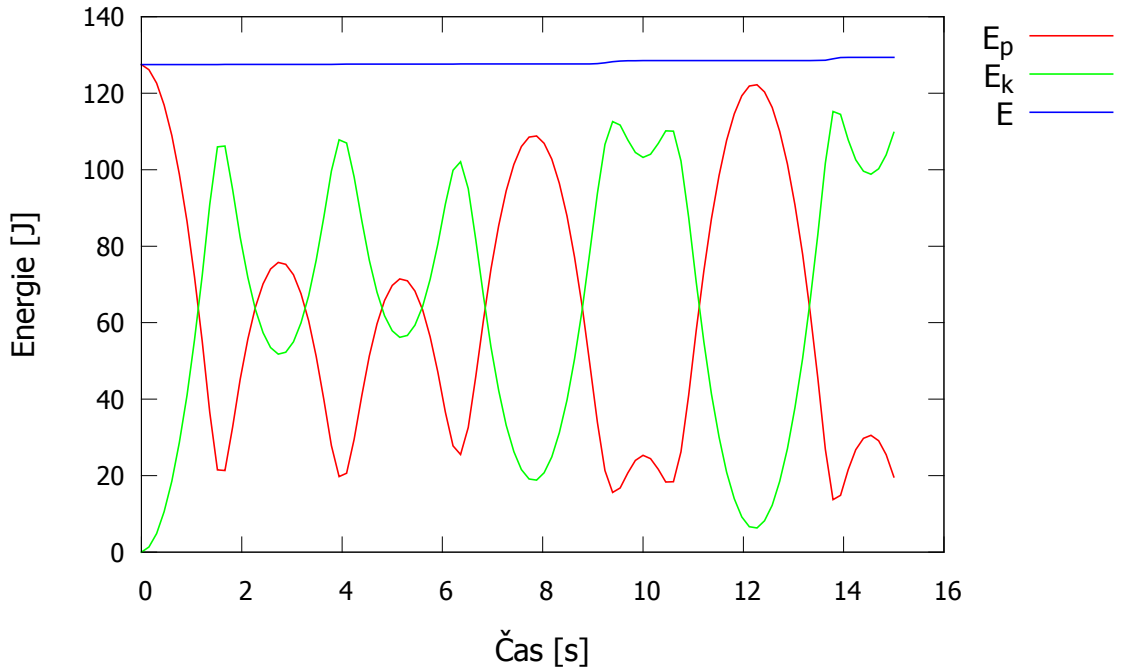
Celkovou energii E určíme jako součet potenciální energie E_p a kinetické energie E_k, které určíme podle vzorců:

$$E_p = mgh$$
$$E_k = \frac{1}{2}\left(mv^2 + I\omega^2\right)$$
$$E = E_p + E_k$$

Při srážkách bude docházet k přeměnám úhlové rychlosti na lineární a naopak. Pokud se těleso A při odrazu roztočí (vzroste mu ω), odrazí se do menší výšky než ta, ze které těleso padalo. Pokud se při odrazu zpomalí jeho otáčivý pohyb, odrazí se výše. V obou případech se sice bude lišit hodnota E_p a E_k, ale celková energie E bude totožná.

Vzhledem k tomu, jak je v simulátoru řešena penetrace (lineární posun viz 3.6.4), bude docházet k drobnému skokovému přírůstku potenciální energie při každém nárazu. Pokud

v simulátoru zakážeme odstranění penetrace, naměřené výsledky budou přesné a dojde pouze k zaokrouhlovacím chybám v řádu 10^{-13} J. Časový průběh experimentu (s odstraněním penetrace) vystihuje obrázek 5.3. Výška h je určena jako poloha těžiště vůči jeho klidové poloze na podložce. Těleso A se do této klidové polohy téměř nikdy nedostane, neboť se odrazí některým rohem dřív než dosáhne nulové výšky, a proto v grafu ani jedna z energií neklesne až na nulovou hodnotu.



Obrázek 5.3: Experiment 2 – Graf jednotlivých energií

5.3 Experiment 3

Třetí ukázka demonstruje rozdílné koeficienty dynamického tření. Na nakloněnou plošinu upustíme 4 čtverce seřazené podle hodnot koeficientů dynamického tření f_d. Plošina bude klesat ve směru osy x.

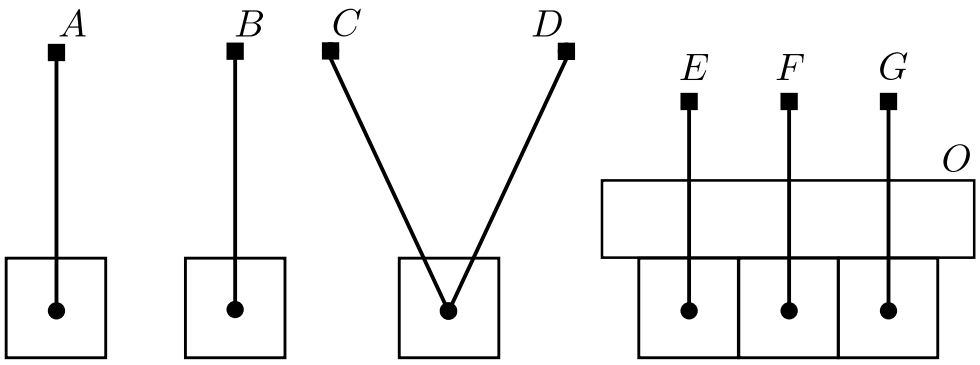
Čtverec s nejnižší hodnotou dynamického tření se po dopadu na plošinu začne pohybovat ve směru osy x nejrychleji. Čtverec s nejvyšší hodnotou f_d urazí krátkou vzdálenost a zastaví. Soubor `experiment3.avi` obsahuje záznam experimentu.

5.4 Experiment 4

Čtvrtá ukázka se zaměřuje na některé nadefinované síly, především pro pružiny. Video čtvrté ukázky obsahuje soubor `experiment4.avi`. Model je znázorněn na obrázku 5.4 a obsahuje:

- netlumenou pružinu A,
- tlumenou pružinu B,
- dvě netlumené pružiny C a D nesoucí stejné závaží,

- tři tlumené pružiny E, F a G, na jejichž závaží je položen obdélník O.



Obrázek 5.4: Experiment 4 – Různé použití pružin

Závaží všech pružin má hmotnost 3 kg. Na každé závaží působí gravitační síla z rovnice 2.12 a pružinová síla 2.13. U tlumených pružin navíc působí brzdná síla 2.14. V počátku simulace jsou závaží pružin vychýlena z jejich klidové pozice, a proto začnou kmitat. Závaží tlumených pružin B, E, F a G postupně zastaví (stejně jako obdélník O, který nesou).

Kapitola 6

Závěr

Cílem této práce bylo prostudovat problematiku tvorby fyzikálních simulátorů mechaniky těles a navrhnout a implementovat simulátor zabývající se stejnou tématikou.

V této práci byly popsány některé postupy, které se uplatňují v současných real-time simulátorech mechaniky tuhých těles. V rámci práce byly navrženy a implementovány všechny potřebné komponenty simulátoru včetně systémů pro numerickou integraci, detekci kolizí a rozřešení kolizí. Pro demonstraci dosažených výsledků a prezentaci vlastností simulátoru byla vytvořena sada experimentů, které byly podrobně vysvětleny. Simulátor byl stavěn tak, aby šel rozšířit například přidáním dalších sil nebo typů těles.

Vytvořený simulátor dosahuje velmi přesných výsledků a ve většině situací nemá problém se stabilitou. Největší problém má simulátor v oblasti detekce kolizí. Jelikož není detekován přesný čas srážek, dochází k částečné penetraci těles a simulace probíhá nepřesně pro více těles stavěných na sebe. Systém spojité detekce kolizí popsaný v kapitole 3.3.1 by odstranil většinu nepřesností a zajistil by i odhalení kolizí, které současný simulátor nenajde.

Pokud bychom měli k dispozici dostatečný výkon a chtěli bychom dosáhnout maximální přesnosti, mohli bychom rozšířit algoritmus simulace jednoho kroku tak, aby zahrnul v mezikrocích metody RK4 detekci a zpracování kolizí.

Vývoj by také mohl směřovat na oblast grafiky a uživatelského rozhraní. Simulátor lze například rozšířit pro práci v trojrozměrném prostoru nebo by mohl být vytvořen detailní editor modelů.

Literatura

[1] Bender, J. *Impulse-based dynamic simulation* [online]. [cit. 2012-04-29]. Dostupné na: <http://www.impulse-based.de/>.

[2] Catto, E. *Box2D: A 2D Physics Engine for Games* [online]. [cit. 2012-04-29]. Dostupné na: <http://box2d.org/>.

[3] Catto, E. *Fast and Simple Physics using Sequential Impulses* [online]. 2006 [cit. 2012-05-5]. Dostupné na: <http://code.google.com/p/box2d/downloads/detail?name=GDC2006_ErinCatto.zip>.

[4] Catto, E. *Numerical Integration* [online]. 2009 [cit. 2012-04-29]. Dostupné na: <http://code.google.com/p/box2d/downloads/detail?name=GDC2009_ErinCatto.zip>.

[5] Coumans, E. *Bullet Physics Library* [online]. [cit. 2012-04-29]. Dostupné na: <http://bulletphysics.org/wordpress/>.

[6] Coumans, E. *Continuous Collision Detection and Physics* [online]. Draft revision 5. 2005 [cit. 2012-04-10]. Dostupné na: <http://www.continuousphysics.com/BulletContinuousCollisionDetection.pdf>.

[7] Drumwright, E. A Fast and Stable Penalty Method for Rigid Body Simulation. *IEEE Transactions on Visualization and Computer Graphics*. Leden 2008, roč. 14, č. 1. S. 231–240. Dostupné na: <www-robotics.usc.edu/~drumwrig/pubs/tvcg.pdf>. ISSN 1077-2626.

[8] Ericson, C. *Real-time Collision Detection*. 1. vyd. Boston: Morgan Kaufmann Publishers, 2005. 591 s. ISBN 15-586-0732-3.

[9] Halliday, D., Resnick, R. a Walker, J. *Fyzika: vysokoškolská učebnice obecné fyziky*. 1. vyd. Brno: VUTIUM, 2000. ISBN 80-214-1868-0.

[10] Hecker, C. *Physics, Part 3: Collision Response* [online]. 1997 [cit. 2012-04-17]. Dostupné na: <http://chrishecker.com/images/e/e7/Gdmphys3.pdf>.

[11] Hecker, C. *Physics, Part 4: The Third Dimension* [online]. 1997 [cit. 2012-04-17]. Dostupné na: <http://chrishecker.com/images/b/bb/Gdmphys4.pdf>.

[12] Jakobsen, T. *Advanced Character Physics* [online]. 2003 [cit. 2012-04-15]. Dostupné na: <www.pagines.ma1.upc.edu/~susin/contingut/AdvancedCharacterPhysics.pdf>.

[13] Lantinga, S. *Simple DirectMedia Layer* [online]. [cit. 2012-03-20]. Dostupné na: <http://www.libsdl.org/>.

[14] Lembcke, S. *Chipmunk Physics* [online]. [cit. 2012-04-29]. Dostupné na: <http://chipmunk-physics.net/>.

[15] Millington, I. *Game Physics Engine Development.* Boston: Morgan Kaufmann Publishers, 2007. 456 s. ISBN 0-12-369471-X.

[16] Tassa, Y. *CapSim – the MATLAB physics engine* [online]. [cit. 2012-03-25]. Dostupné na: <http://alice.nc.huji.ac.il/~tassa/pmwiki.php?n=Main.Code>.

[17] Witters, K. *DeWiTTERS Game Loop* [online]. 2009-07-13 [cit. 2012-04-29]. Dostupné na: <http://www.koonsolo.com/news/dewitters-gameloop/>.

Příloha A

Obsah CD

/source/ Adresář se zdrojovými kódy programu včetně `solution` pro Visual Studio a `Makefile` pro Linux.

/text/ Adresář s textovou částí bakalářské práce ve formátu `pdf` a jejím zdrojovým kódem v LaTeXu.

/text/fig/ Adresář s obrázky a grafy použitými v textové části práce.

/video/ Adresář s videozáznamy demonstračních experimentů.

readme.pdf Návod pro instalaci a použití programu ve formátu `pdf`.

readme.txt Textový formát souboru s návodem pro instalaci a použití programu .