Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Tomáš Stráník

Fyzikální simulátor pohybu robota

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Jaroslav Semančík
Studijní program: Informatika, Programování

2006

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčením práce.

V Praze dne 26.7.2006 Tomáš Stráník

Název práce: Fyzikální simulátor pohybu robota
Autor: Tomáš Stráník
Katedra: Kabinet software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Jaroslav Semančík
e-mail vedoucího: jaroslav_semancik@yahoo.com

Abstrakt: Předložená práce se zabývá simulací pohybu robota složeného z dvojrozměrných tuhých těles. Práce seznamuje čtenáře se základy fyzikální simulace tuhých těles v 2D systému. V prvních kapitolách jsou popsány algoritmy potřebné k pochopení správné simulace tuhých těles a dále potom odvození rovnic potřebných pro realizaci algoritmů. V dalších kapitolách je popsána implementace fyzikální knihovny pro simulaci tuhých těles. Součástí práce je i vytvoření demonstrativní aplikace pro simulování pohybu robota, která používá vytvořenou fyzikální knihovnu. Aplikace obsahuje příklady pohybů robota ve 2D, které jsou v dobrém souladu s opravdovým 2D pohybem.

Klíčová slova: 2D simulace tuhých těles

Title: Physical simulator of a robot motion
Author: Tomáš Stráník
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Jaroslav Semančík
Supervisor's e-mail address: jaroslav_semancik@yahoo.com

Abstract: The aim of this thesis is a simulation of the motion a robot consisting from rigid bodies in two dimensions. The work, in general, introduces the reader with the basic principles of the simulation. In the first part, algorithms used in the simulation are described and explained in details. Furthermore, equations describing the algorithms are derived. Next, the description of the library for the simulation of the robot's motion is presented. Finally, a demonstrative program using the library is introduced. This application contains few examples of the robot's motion in the 2D, which are in good agreement with real 2D motion.

Keywords: 2D rigid body's simulation

Kapitola 1 – Úvod

Cílem této bakalářské práce je vytvoření aplikace, která bude určena k vytváření jednoduchých dvourozměrných robotů a k fyzikálnímu simulování jejich pohybu ve dvourozměrném světě. Pohyb bude simulován na zemi, tzn. nikoliv ve vodě a ve vzduchu.

Uživatel dostane do rukou nástroj, s jehož pomocí vytvoří jednoduchý model robota z dvourozměrných polygonů, které spojí klouby a nadefinuje, jak se má robot v čase pohybovat.

Aplikace musí umět věrně fyzikálně simulovat zadaný pohyb robota, aby uživatel viděl, jak vytvořený robot funguje.

Dalším cílem této bakalářské práce je vytvoření univerzální knihovny pro simulaci tuhých těles, která bude v samotné aplikaci dále využita.

Motivace pro vytvoření této fyzikální knihovny je skutečnost, že není snadné nalézt volně šiřitelnou knihovnu pro 2D simulaci tuhých těles. Je nutno podotknout, že existuje mnoho komerčních i volně stažitelných knihoven pro 3D.

Tato práce seznámí čtenáře se základy fyzikální simulace v 2D systému. V práci je kladen důraz na matematické odvození rovnic pro fyzikální simulaci v 2D systému, jelikož drtivá většina prací, které se tomuto tématu věnují, je věnována 3D. V práci jsou odvozené rovnice pro 2D systém, které jsem nenalezl v žádných mě dostupných materiálech.

Předpokládají se základní znalosti lineární algebry a mechaniky.

1.1 Členění kapitol

V druhé kapitole se čtenář seznámí se základy simulace tuhých těles na počítači. Čtenář se dále dozví, co je vše nutné udělat, aby se tělesa ve scéně správně pohybovala pod vlivem sil, které na tělesa působí. Také je uvedeno jaké fyzikální modely se používají k simulaci vzájemných odrazů těles a jaké fyzikální modely pro simulaci tření. Dále je uvedeno co je nutno udělat, aby tělesa zůstala spojena klouby.

Ve třetí kapitole se čtenář seznámí s konkrétní implementaci vytvořené fyzikální knihovny libPE. Je zde popsáno, jaké fyzikální modely byly použity pro implementaci knihovny a jaké byly důvody k takovému výběru. V programátorské příručce je popsán interface k používání knihovny, takže po jejím prostudování, bude čtenář schopný

sestrojit např. aplikaci, ve které budou jeho tělesa správně rotovat i poté, co do nich narazí jiné těleso atd…

Ve čtvrté kapitole je popsána samotná aplikace pro simulaci robotů. Uživatel bude mít v rukou ukázkové použití fyzikální knihovny libPE. V uživatelské příručce je popsáno, jak pomocí aplikace uživatel vytvoří nová tělesa, jak nastaví jejich vlastnosti (hmotnost, moment setrvačnosti), jak vytvoří roboty, které se bude snažit rozpohybovat. Také je zde uvedeno vše potřebné pro spuštění aplikace. Na aplikaci byl hlavně kladen důraz na přenositelnost a platformové nezávislosti.

Kapitola 2 – Fyzikální simulace

V této kapitole bude čtenář seznámen se základy fyzikální simulace v počítači. Nejdříve rozebereme jednoduchý příklad simulace jedné částice (hmotného bodu) v 2D prostoru. Dále bude příklad rozšířen na celá tělesa. Chtěl bych upozornit, že ve značení budu někdy vynechávat šipky pro značení vektorů. V zápisu by se totiž pletly.

2.1 Simulace částice

Předpokládejme, že chceme simulovat pohyb hmotného bodu v 2D prostoru. Hmotný bod je v každém čase jednoznačně definován svou polohou a rychlostí.

Zaveďme proto značení:

$p(t)$	*poloha hmotného bodu v čase t*
$v(t) = \dot{p}(t) = \frac{d}{dt} p(t)$	*rychlost hmotného bodu v čase t*
$\mathbf{X}(t) = \begin{pmatrix} p(t) \\ v(t) \end{pmatrix}$	*stav hmotného bodu v čase t*

Abychom mohli částici animovat v čase, musíme znát jaká je derivace $\mathbf{X}(t)$ v čase *t* a jaký je počáteční stav $\mathbf{X}(t_0)$ bodu v čase t_0. Abychom zavedli $\dot{\mathbf{X}}(t)$, musíme znát sílu, která působí na těleso v čase *t*.

Musíme proto dále zavést:

m	*hmotnost částice*
$F(t)$	*součet všech sil, které působí na částici v čase t*
$\dot{v}(t) = a(t) = \frac{F(t)}{m}$	*zrychlení hmotného bodu v čase t*
$\dot{\mathbf{X}}(t) = \begin{pmatrix} \dot{p}(t) \\ \dot{v}(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ a(t) \end{pmatrix}$	*změna stavu bodu v čase t*
$\mathbf{X}(t_0) = \begin{pmatrix} p(t_0) \\ v(t_0) \end{pmatrix}$	*počáteční stav bodu v čase* t_0

Nyní, když známe $\dot{\mathbf{X}}(t)$ a $\mathbf{X}(t_0)$ můžeme vypočítat stav našeho hmotného bodu v libovolném čase *t* a pomocí jednoduchého iterativního algoritmu:

$t_0 = 0;$
$h = malé\ číslo;$
$while(t_0 < t)$
{
$\mathbf{X}(t_0 + h) = \mathbf{X}(t_0) + h \cdot \dot{\mathbf{X}}(t_0);$
$t_0 += h;$
}

Popsaný iterativní algoritmus se nazývá Eulorova metoda pro řešení běžných diferenciálních rovnic (ODE) $\dot{\mathbf{X}} = f(\mathbf{X}, t)$. Tato metoda je kriticky závislá na volbě parametru h. Čím větší je h, tím větší chyba se akumuluje v $\mathbf{X}$. Další, mnohem přesnější metody pro řešení ODE jsou např. v [1].

Výše popsaný algoritmus nám umožňuje simulovat naši částici po celý sledovaný časový úsek.

Celý simulační algoritmus pro simulaci hmotných bodů ve světě vypadá potom následovně:

nastav počáteční podmínky pro částice;
t = 0;
h = malé;
while (simuluji) {
foreach (částice ve scéně) {
spočti výslednou sílu $F(t)$ *působící na částici;*
$těleso.a = \frac{F(t)}{m};$
$těleso.v += h \cdot těleso.a;$
$těleso.p += h \cdot těleso.v;$
}
$t += h;$
nakresli tělesa na nových pozicích;
}

2.2 Tuhé těleso a jeho veličiny

Nyní, když víme, jak simulovat částice v prostoru, rozšíříme naší simulaci na celá tělesa. Při simulování v počítači musíme zavést určité fyzikální modely, abychom simulaci zjednodušili. Je např. dosti nepředstavitelné, že bychom při simulaci počítali s každým atomem tělesa. Počítat, jak atomy tělesa do sebe narážejí při deformaci tělesa po nárazu na tvrdou podložku, je dosti obtížné a pro plynulou simulaci nevhodné.

Pro real-time simulování pohybu těles se obvykle používá model ideálního tuhého tělesa. Jedná se o těleso, které má tu ideální vlastnost, že libovolná síla působící na těleso nezpůsobí deformaci tělesa, ale má pouze pohybový účinek.

Na rozdíl od částic, tuhým tělesem můžeme také otáčet. Těleso zaujímá ve scéně nějaký prostor, protože má nějaký objem. Musíme proto zavést orientaci tuhého tělesa. Ve 3D se jedná o matici o rozměrech 3x3, která udává, jak je těleso otočeno vzhledem svému počátku kolem os x, y, z. Ve 2D je situace jednodušší, protože tělesem můžeme otáčet pouze kolem jedné osy. Orientace tělesa je proto skalár.

Počátek tělesa si můžeme jednoduše představit tak, že se jedná o bod o souřadnici $\vec{0}$ v souřadným systému, který má počátek v těžišti tělesa. Tento souřadný systém se výstižně anglicky nazývá *body space*. Každé tělesa z *body space* umisťujeme do scény, která má také svůj souřadnicový systém, jenž je pevně zvolen. Tento systém se anglicky nazývá (také výstižně) *world space*.

Pokud máme orientaci tělesa zadanou v jeho *body space*, můžeme naše těleso umístit do *world space* (do kterého umisťujeme všechna tělesa) tak, že nejdříve těleso otočíme podle jeho orientace v jeho *body space* a následně těleso posuneme podle jeho pozice do *world space*. Toto je obecný princip, který se používá jak v 2D, tak v 3D systému.

Pokud má těleso svou orientaci, tak se určitě může tato orientace tělesa v průběhu simulace měnit. Derivace orientace tělesa podle času se nazývá úhlová rychlost a udává jak rychle se mění orientace našeho tělesa. Jinými slovy, jak rychle se naše těleso otáčí.

Nyní, když víme, jak je definována orientace a pozice tělesa v prostoru, je dobré si uvědomit, co ovlivňuje tyto dvě vlastnosti tělesa. Podle 1. Newtonova zákona těleso setrvává v klidu nebo v pohybu rovnoměrném přímočarém, jestliže na něj nepůsobí jiná tělesa silou nebo síly působící na těleso jsou v rovnováze. Je dobré si uvědomit, že když na těleso začne působit nějaká síla, tak se v tom okamžiku nezmění rychlost tělesa ani jeho úhlová rychlost natož pak jeho pozice či orientace.

To, co síla, která působí na těleso, ovlivní, je změna rychlosti tělesa, neboli jeho zrychlení. Jelikož jsme si rychlost tělesa rozdělili na dvě složky a to lineární rychlost (derivace pozice podle času) a úhlovou rychlost (derivace orientace podle času), zavedeme proto ještě lineární zrychlení (2. derivace pozice podle času) a úhlové zrychlení (2. derivace orientace podle času).

Nyní máme skoro všechny fyzikální veličiny tuhého tělesa, které potřebujeme pro jeho simulaci.

Zaveďme proto značení:

$p(t)$	*poloha tělesa v čase t*
$v(t) = \dot{p}(t) = \frac{d}{dt} p(t)$	*rychlost tělesa v čase t*
$a(t) = \dot{v}(t) = \frac{d}{dt} v(t)$	*(lineární) zrychlení tělesa v čase t*
$\Omega(t)$	*orientace tělesa v čase t*
$\omega(t) = \dot{\Omega}(t) = \frac{d}{dt} \Omega(t)$	*úhlová rychlost tělesa v čase t*
$\alpha(t) = \dot{\omega}(t) = \frac{d}{dt} \omega(t)$	*úhlové zrychlení tělesa v čase t*

Nyní se zaměřme, jak působící síla ovlivní zrychlení (jak lineární tak úhlové) tělesa. Pokud síla působí v těžišti, tak se těleso po čase začne pohybovat ve směru působení síly, ale nezačne se otáčet. Představte si tužku položenou na stole. Pokud do ní cvrkneme v jejím středu, tužka se začne pohybovat, ale ne rotovat. Pokud do ní necvrnkneme v jejím středu, ale někde poblíž, tužka se začne pohybovat i rotovat. Čím větší bude vzdálenost působení síly od těžiště, tím více se bude tužka na stole otáčet.

Lineární složku zrychlení tělesa $a(t)$ síla $F(t)$ podle 2. Newtonova zákonu ovlivňuje vztahem $a(t) = \frac{F(t)}{m}$, kde *m* je hmotnost tělesa.

U úhlové složky zrychlení $\alpha(t)$ je situace o něco složitější. Podle příkladu s tužkou jsme viděli, že otáčení tužky ovlivňuje místo působení síly. Síla $F(t)$ totiž vytváří moment síly $\tau(t)$.

Moment síly $\tau(t)$ je fyzikální veličina, která vyjadřuje míru otáčivého účinku síly. Velikost momentu síly závisí na velikosti síly a na vzdálenosti od osy otáčení (čím dále, tím větší moment síly). Osa otáčení je v 2d systému, jak už bylo řečeno, jen jedna.

Moment síly $\tau(t)$ je v 3D dán vztahem $\tau(t) = r \times F(t)$, kde $\vec{r}$ je působiště síly v *body space* tělesa. Ve 2D vektorový součin není a je nahrazen jinou operací tzv. *PerpDotProduct*(), kterou budu značit symbolem $\otimes$ a je definována

$$\vec{a} \otimes \vec{b} = \vec{a}_{\perp} \cdot \vec{b} \quad \textit{kde } \vec{a}_{\perp} \textit{ je kolmý vektor k } \vec{a} \textit{ a napravo rovnosti je skalární součin}$$

$$\tau(t) = r \otimes F(t) = r_{\perp} \cdot F(t)$$

Operace $\otimes$ má stejný význam pro 2D systémy jako vektorový součin pro 3D systémy. Bližší informace v [2].

Nyní, když známe moment síly $\tau(t)$, který síla $F(t)$ vytvoří na těleso, můžeme se více přiblížit k zjištění, jak velkou změnu úhlového zrychlení síla $F(t)$ způsobí. Stejně jako hmotnost tělesa ovlivňuje lineární zrychlení tělesa (čím těžší těleso je, tím hůře se roztlačuje), tak existuje fyzikální veličina moment setrvačnosti I, která vyjadřuje míru setrvačnosti tělesa při otáčivém pohybu. Její velikost závisí na rozložení hmoty v tělese vzhledem k ose otáčení. Body tělesa s větší hmotností a umístěné dál od osy mají větší moment setrvačnosti.

Moment síly $\tau(t)$ a moment setrvačnosti I jsou ve vztahu $\tau(t) = \alpha(t) \cdot I$. Čím větší je tedy moment setrvačnosti I tělesa, tím hůře se nám ho bude roztáčet. Pokud bude $I = \infty$, potom těleso neroztočí jakákoliv velká síla, stejně jako jakákoliv velká síla nerozpohybuje těleso s nekonečnou hmotností.

Z výše popsaného musíme proto zavést pro tuhé těleso ještě dvě veličiny a to:

m	*hmotnost tělesa*
I	*moment setrvačnosti, který je v 2D skalár*

Pro bližší informace např. v [3] a [2].

Nyní, když víme, jak působící síla ovlivňuje lineární a úhlové zrychlení tělesa můžeme přepsat algoritmus pro simulování částic na algoritmus, který bude simulovat celá tělesa:

```
nastav počáteční podmínky pro tělesa;
spočti hmotnost a moment setrvačnosti pro tělesa;
t = 0;
h = malé;
while ( simuluji ) {
    foreach ( těleso  ve scéně ) {
            F(t) = 0⃗; // výsledná síla působící na těleso v těžišti
            foreach ( síla f(t)  působící na těleso) {
                    r =  místo působiště síly v body space tělesa;
                    těleso.  τ+ = r ⊗ f(t);
                    F(t)+ = f(t);
            }
            // integrování  lineární složky
            těleso. a = F(t) / m;
            těleso. v+ = h · těleso.a;
            těleso. p+ = h · těleso.v;
            // integrování úhlové složky
```

```
            těleso.α = τ(t) / I;
            těleso.ω+ = h · těleso.α ;
            těleso.Ω+ = h · těleso.ω ;
        }
        t+ = h;
        nakresli tělesa na nových pozicích;
    }
```

2.3 Detekce kolizí

Při simulování pohybu tuhých těles musíme zajistit, aby tělesa do sebe při nárazu nepronikala. Simulujeme přece pohyb ideálních nedeformovatelných tuhých těles.

K tomuto účelu slouží detektor kolizí. Detektor kolizí má za úkol, jak už jeho název napovídá, detekovat kontakty těles ve scéně. Dále jeho podstatná úloha spočívá ve smysluplným informováním o tomto jevu. Pokud výstupem detektoru bude pouze informace, že některá dvě tělesa ve scéně jsou v kontaktu a nic víc, tak těžko budeme schopni zabránit tomu, aby tato tělesa v příštím simulačním kroku do sebe přestala narážet, jinými slovy, aby se od sebe odrazila.

Další vlastnost dobrého detektoru kolizí spočívá v predikci kolizí. Musíme si totiž uvědomit, že tělesa ve scéně simulujeme po krátkých časových úsecích o velikosti *h*. Může se proto stát, že v čase *t* ještě kolize není detekována, ale v čase *t*+*h*, již tělesa do sebe pronikají. Samotný kontakt totiž nastal v časovém intervalu $(t, t+h)$.

Proto výstupem detektoru kolizí, který umí předpovídat kolize, je buď fakt, že kolize nastala, nebo že nastane v kolizním čase t_K, pro který platí, že $t < t_K < t+h$, a nebo že kolize nenastala a do příštího simulačního kroku nenastane.

Další vlastnost, který by dobrý detektor kolizí mohl umět, je schopnost vypořádat se situací, kdy tělesa do sebe již pronikají. Tato situace nastává hlavně pro ty detektory, které neumějí předpovídat kolize (o kolizi se většinou dozvědí, až když tělesa do sebe pronikají). Detektor potom musí tělesa od sebe odsunout takovým způsobem, aby simulace mohla pokračovat v co nejvěrnější podobě. Na druhou stranu je dobré vědět, že některé simulační modely s pronikáním těles v simulaci počítají. Dokážou tělesa poté zastavit, aby dále do sebe nepronikala. Pro běžnou simulaci, hlavně v počítačových hrách, je takovéto chování tolerováno.

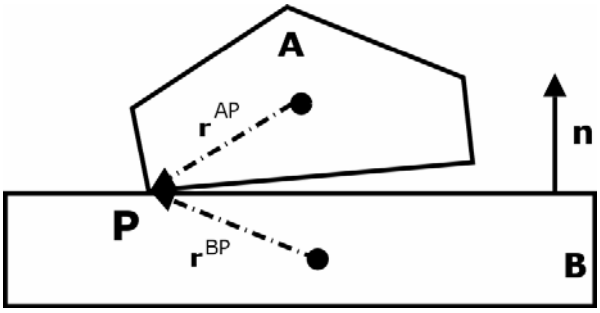
Jak už bylo řečeno, detektor kolizí nás musí srozumitelně informovat o vzniklém kontaktu. Tuto informaci poté předá modulu, který se stará o odrazy. Tento odrazový

modul poté začne působit na tělesa, která se v kolizi vyskytla, silami tak, aby přestala do sebe pronikat.

2.4 Odrazy

V této kapitole se seznámíme, co všechno je potřeba udělat, aby tělesa při kontaktu do sebe přestala pronikat.

Představme si tělesa *A* a *B*, která jsou v kontaktu a dotýkají se v bodě *P*. Vzdálenost těžiště tělesa *A* od kontaktního bodu *P* budeme značit $\vec{r}^{AP}$ a vzdálenost těžiště tělesa *B* od kontaktního bodu *P* budeme značit $\vec{r}^{BP}$. Pro lepší představu se můžeme prohlédnout obrázek (1).



obr (1)

Abychom věděli, zda tělesa A a B do sebe opravdu narážejí, musíme znát rychlosti bodu P na tělese A i na tělese B. Ačkoliv souřadnice bodu P na tělese A a souřadnice bodu P na tělese B jsou v okamžiku kontaktu totožné (nastal přeci kontakt), rychlosti obou bodů mohou být značně rozdílné - např. pokud těleso A je v klidu a těleso B na těleso A padá. Potom rychlost bodu P na tělese A je $\vec{0}$, protože celé těleso A je v klidu, a rychlost bodu P na tělese B je určitě nenulová (těleso se přeci pohybuje).

Rychlost libovolného bodu $\vec{x}$ na tělese T můžeme ve 2d systému spočítat pomocí jeho lineární a úhlové rychlosti pomocí vztahu:

$$v_{TX}(t) = v_T(t) + r_{\perp}^{TX} \cdot \omega_T(t) \qquad \text{, kde } r_{\perp}^{TX} \text{ je kolmý vektor k vektoru } r^{TX} \quad (1)$$

V tomto vztahu jde dobře vidět, že body, které jsou dále od osy otáčení, se pohybují větší úhlovou rychlostí než body, které jsou blíže k ose otáčení. Jak už bylo řečeno v 2d je pouze jedna osa otáčení, která většinou prochází těžištěm tělesa. Ve vztahu jde potom dobře vidět, že rychlost těžiště nezávisí na úhlové rychlosti tělesa, ale pouze na lineární

rychlosti (pokud osa otáčení prochází těžištěm). Pro bližší informace o odvození vztahu viz [2].

Zaveďme proto značení:

$$\begin{aligned} v_{AP}(t) &= v_A(t) + r_{\perp}^{AP} \cdot \omega_A(t) \\ v_{BP}(t) &= v_B(t) + r_{\perp}^{BP} \cdot \omega_B(t) \end{aligned} \qquad (2)$$

Relativní rychlost obou bodů na kolizních tělesech poté můžeme vyjádřit pomocí vztahu:

$$v_{AB}(t) = v_{AP}(t) - v_{BP}(t) \qquad (3)$$

Abychom mohli dobře usoudit, zda kolize opravdu nastala, potřebujeme znát "kolizní normálu" pro kolizi. „Kolizní normála" je normála hrany, do které se narazilo. Ve 3D se jedná o normálu plochy, do které se narazilo. Tuto důležitou informaci nám musí sdělit detektor kolizí. Kolizní normálu budeme značit *n*, viz obr (1). Při kontaktu vrchol-vrchol je výběr kolizní normály nedeterministický.

Nyní můžeme zavést relativní normálovou rychlost kolidující bodů:

$$v_n(t) = v_{AB}(t) \cdot n = \left(v_{AP}(t) - v_{BP}(t)\right) \cdot n \qquad (4)$$

Pomocí *(4)* můžeme konečně vyslovit podmínku pro výskyt kolize:

Pokud $v_n(t) < 0$, *tak nastává kolize těles, které jsou v kontaktu.*

Pokud je výše zmíněná podmínka splněná, znamená to, že se tělesa do sebe v příštím simulačním kroku ještě více zanoří, jelikož se pohybují proti sobě.

Pokud je $v_n(t) > 0$, znamená to, že kontaktní body obou těles se od sebe vzdalují a není třeba se o kolizi starat. V příštím simulačním kroku sám kontakt zmizí.

Pokud je $v_n(t) = 0$, znamená to, že kontaktní body se do sebe nezanořují, ani se od sebe nevzdalují. Ačkoliv je $v_n(t) = 0$ neznamená to ještě, že $v_{AB}(t)$ je nula. Pokud $v_n(t) = 0$ a $v_{AB}(t) \neq \vec{0}$, potom takováto situace znamená, že se tělesa v kontaktním bodu dotýkají a v příštím simulačním kroku se dále dotýkat budou a že se jedno z těles po druhém smýká.

Jestliže zjistíme, že kontakt je kolidující, musíme na tělesa začít působit silami tak, aby v příštím simulačním kroku do sebe přestala pronikat.

2.4.1 Impulsy

Mohlo by se zdát, že situace je nyní jasná. Detekovali jsme kolizní kontakt, a proto v místě kontaktu zapůsobíme na tělesa dostatečně velkými silami opačného směru (podle 3. Newtonova zákona akce a reakce), tak aby tělesa v příštím simulačním kroku do sebe přestala pronikat.

Jak už jsme si ale řekli, to co síla ovlivňuje je zrychlení a nikoliv rychlost. Proto ať už by síla, která na těleso v místě kontaktu působí, byla jakkoliv veliká, bude potřebovat nějaký čas, aby ovlivnila rychlost a následně pozici tělesa (síla ovlivní zrychlení tělesa, které po integraci změní rychlost).

Jelikož jsou ale tělesa v kolizním kontaktu a my potřebujeme, aby tělesa již v příštím simulačním kroku do sebe přestala pronikat, nemáme žádný čas na to, abychom čekali, než síla ovlivní rychlost. Musíme proto ovlivnit přímo rychlost tělesa.

K tomuto účelu slouží fyzikální veličina impuls síly $\vec{\psi}(t)$, která ovlivňuje hybnost tělesa.

Impuls síly si můžeme představit jako nekonečně velkou sílu působící v nekonečně krátkém časovém úseku.

Impuls $\vec{\psi}(t)$ aplikované na těleso A ve vzdálenosti $\vec{r}$, od osy otáčení (těžiště), způsobí změnu lineární a úhlové rychlosti tělesa vztahem:

$$v'_A(t) = v_A(t) + \frac{\psi(t)}{m_A} \qquad (5)$$

$$\omega'_A(t) = \omega_A(t) + \frac{r \otimes \psi(t)}{I_A} = \omega_A(t) + \frac{r_\perp \cdot \psi(t)}{I_A} \qquad (6)$$

V místě kolize budeme tedy působit impulsy nikoliv silami.

2.4.2 Model odrazu

Je dobré si uvědomit, co se ve skutečném světě při nárazu děje. Při nárazu těles se atomy jednotlivých těles začnou dotýkat a těleso se začne deformovat. Při deformaci se nějaké množství pohybové energie uvolní v podobě tepla. Tudíž množství energie se po

srážce v systému sníží a proto součet rychlostí těles před srážkou se nerovná součtu rychlostí těles po srážce.

Toto chování se v simulací může napodobit pomocí modelu s anglickým názvem *Newton's Law of Restitution*. Tento model dává do vztahu relativní normálovou rychlost kontaktního bodu před kolizí a po kolizi pomocí vztahu:

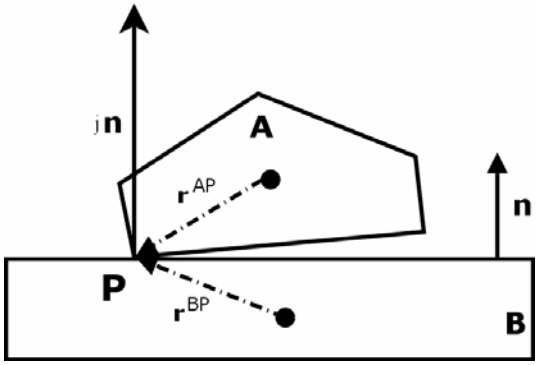
$$v_n'(t) = -e \cdot v_n(t) \quad , kde \ e \in < 0,1 > \qquad (7)$$

kde $v_n'(t)$ je relativní normálová rychlost bodu po kolizi a e je konstanta, která je zavedena tímto modelem. Anglicky se konstanta e nazývá *coefficient of restitution* a udává poměr mezi normálovou rychlostí bodu před kolizí a po kolizi.

Pokud je $e = 0$, znamená to, že se body po odrazu od sebe neodrazí. Jedná se např. o bláto spadlé na zem. Pokud je $e = 1$ jedná se o ideální těleso. Hodnota konstanty e závisí na materiálech těles, která se v kolizi ocitnou.

2.4.3 Systém bez tření

Pokud v systému není žádné tření mezi tělesy v místě kontaktu, potom impuls působí ve směru kolizní normály mezi kolidujícími tělesy. V jednom směru na těleso A a opačným impulsem na těleso B. viz obr *(2)*.



obr (2)

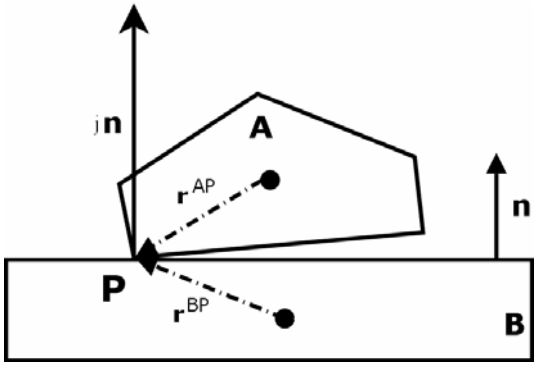
Úkolem je tedy spočítat velikost j impulsu $\vec{\psi}_N(t) = j \cdot \vec{n}(t)$, tak aby platila rovnost *(7)*.

Po spočtení j, aplikujeme impuls $\vec{\psi}_N(t) = j \cdot \vec{n}(t)$ na těleso A a opačný impuls $-\vec{\psi}_N(t) = -j \cdot \vec{n}(t)$ na těleso B. Po správném aplikování impulsu se tělesa od sebe odrazí.

2.4.4 Systém s třením

Pro simulování tření v systému se obvykle používá *Coloumbův model tření*.

Jeho funkce je následující. Pokud v systému definujeme tření, potom v místě kontaktu působí impuls, který můžeme rozložit na dva impulsy. Kromě normálové složky, která má stejný směr jako v systému bez tření, zde působí ještě tečná složka.



obr (3)

Tečná složka výsledného impulsu, jak už název napovídá, má směr tečný ke kolizní hraně a vytváří tření v systému. Zatímco normálová složka se stará o zabránění pronikání těles do sebe. Viz obr *(3)*.

Tření může být buď dynamické či statické. Abychom mohli rozhodnout, jaká třecí síla má v místě kontaktu P působit mezi tělesy A a B, musíme určit relativní tečnou rychlost $v_t(t)$ kolidujících bodů:

$$v_t(t) = v_{AB}(t) \cdot n_\perp = \left(v_{AP}(t) - v_{BP}(t)\right) \cdot n_\perp \quad (8)$$

Pokud je $v_t(t) = 0$, potom v místě kontaktu působí statická třecí síla (impuls). Pokud je $v_t(t) \neq 0$, působí dynamická třecí síla (impuls).

Pokud v místě kontaktu působí dynamická třecí síla, potom třecí impuls $\vec{\psi}_T(t)$ je ve vztahu s $\vec{\psi}_N(t)$ dán pomocí rovnice:

$$\vec{\psi}_T(t) = \mu_d \cdot \vec{\psi}_N(t) \qquad \textit{kde } \mu_d \textit{ je koeficient dynamického tření. (9)}$$

Pokud v místě kontaktu působí statická třecí síla, potom třecí impuls $\psi_T(t)$ je ve vztahu s $\psi_N(t)$ dán pomocí nerovnice:

$$\psi_T(t) \le \mu_s \cdot \psi_N(t) \qquad \textit{kde } \mu_s \textit{ je koeficient dynamického tření. (10)}$$

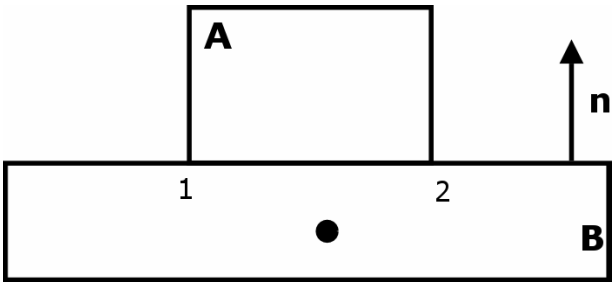
Statické tření má za úkol „udržet“ místo kontaktu v tečném směru na své pozici. Jinými slovy, zabraňuje, aby se těleso začalo pohybovat. Třecí síla $\psi_T(t)$ je tak veliká, aby relativní tečná rychlost bodů po aplikování impulsu výsledného $\psi_N(t) + \psi_T(t)$ zůstala $v_t'(t) = 0$.

Pokud je $v_t'(t) < 0$, potom $\vec{\psi}_T(t) = \mu_s \cdot \vec{\psi}_N(t)$. Statická třecí síla se co nejvíce snaží, aby se kolizní bod nepohyboval.

Naopak pokud je $v_t'(t) > 0$, potom $\vec{\psi}_T(t) = -\mu_s \cdot \vec{\psi}_N(t)$.

2.4.5 Separované odrazy

Pokud v implementaci vypočítáváme kolizní impulsy sériově (ať už v systému s třením či bez tření), mluvíme o implementaci se separovanými odrazy. Implementace vypadá tak, že vezmeme jeden kolizní kontakt a spočteme impuls, který aplikujeme na tělesa.



obr (4)

Takovéto implementace jsou snadné, ale obvykle málo přesné. Způsobuje nestabilitu a „roztřesení“ systému.

Důvod si ukážeme na malé ukázce: Představme si čtverec, který dopadl na desku. Vzniklá situace má dva kolizní body 1,2 (viz obr 4.). Po vypočtení a aplikování impulsu na první kolizní bod, ovlivníme rychlost druhého kolizního bodu. Výsledkem bude, že náš čtverec začne rotovat a bude poskakovat na desce z jednoho rohu na druhý. Chyba bude zvlášť názorná, pokud v kolizích budeme počítat s konstantou $e = 1$. Chyba je v tom, že impulsy o sobě „nevědí“.

V takovéto implementaci se také těžko implementuje statické tření.

2.4.6 Simultánní odrazy

Jak už bylo řečeno v předešlé kapitole, je nutné, aby při výpočtu velikosti impulsu, byl brán zřetel na ostatní impulsy. U našeho příkladu z minulé kapitoly je třeba, aby na kolizní body 1, 2 byly aplikovány impulsy stejné velikosti, které budou mít za následek, že se čtverec odrazí směrem nahoru a nezačne rotovat.

Pro výpočet impulsů se používají analytické metody. Jedná se o řešení tzv. *Linear complementary Programming problem* (LCP).

Pomocí LCP lze dobře simulovat statické tření.

2.5 Klouby

V simulacích často chceme, aby některé dva objekty byly spojeny klouby. Scéna je poté mnohem zajímavější a pro pozorovatele zábavnější. Ve 2d systému je pouze jeden druh kloubu, máme zde jeden stupeň volnosti.

Pro kloub, který spojuje tělesa *A* a *B* v čase *t*, zaveďme toto značení:

$\vec{P}(t)$	*pozice kloubu ve world space v čase t*
$\vec{r}_A(t) = \vec{P}(t) - \vec{p}_A(t)$	*vzdálenost kloubu od těžiště tělesa A*
$\vec{r}_B(t) = \vec{P}(t) - \vec{p}_B(t)$	*vzdálenost kloubu od těžiště tělesa B*
$\vec{v}_{AP}(t)$	*rychlost bodu* $\vec{r}_A(t)$ *tělesa A*
$\vec{v}_{BP}(t)$	*rychlost bodu* $\vec{r}_B(t)$ *tělesa B*

Vektor $\vec{v}_{AP}(t)$ tedy udává rychlost bodu tělesa *A* v místě kloubu. Podobně vektor $\vec{v}_{BP}(t)$ udává rychlost bodu tělesa *B* v místě kloubu.

To co musí kloub udělat, aby se tělesa *A* a *B* držela pohromadě, je udržovat po celý simulovaný čas hodnotu $\vec{v}_{AP}(t) - \vec{v}_{BP}(t)$ na nule. Neboli:

$$\vec{v}_{AP}(t) - \vec{v}_{BP}(t) = \vec{0} \qquad (11)$$

Pokud relativní rychlost obou bodů na tělesech spojená klouby bude $\vec{0}$, potom se tělesa od sebe nikdy nevzdálí. Samozřejmě budou moci kolem kloubu rotovat.

Možností jak klouby implementovat je hned několik. Některé jsou více úspěšné a některé méně. Klouby se mohou implementovat pomocí pružin (méně úspěšná metoda), analyticky podobně jako simultánní odrazy (více úspěšná metoda, ale někdy málo

robustní), nebo metodami založené na aplikování impulsů, které bývají obvykle robustní, nicméně nejsou tak pružné jako analytické.

2.6 Možné přístupy

Nyní víme co je vše potřebné udělat pro správnou simulaci tuhých těles. Metod, jak dosáhnout požadavků, které byly popsány v předešlých kapitolách, je hned několik. Krátce zde popíšu nejvíce známé metody.

2.6.1 Penalty Methods (simulování pomocí pružin)

Princip těchto metod je založen na vytvoření pružiny v místě kontaktu a nebo v místě kloubu tak, aby síla pružiny tělesa opět dostala do správné polohy a zabránila jejich pronikání do sebe. Pokud už tělesa nedrží pohromadě, tak pružina zaniká. Tata metoda je relativně snadná na implementování, ale mnohem těžší je zvolit správně parametry pružin. Tuto metodu ve své práci nepoužívám**.**

2.6.2 Metody založené na impulsech

V této metodě jsou všechna omezení na pohyb brány jako kolize, na kterých je potřeba aplikovat impulsy. Řešení kolizí je lokální a neberou se v úvahu simultánní kolize. V těchto metodách je těžké věrně simulovat statické tření. Na druhou stranu tyto metody jsou relativně snadné a robustní.

2.6.3 Analytické metody

V těchto metodách se berou všechny kolize či dokonce i klouby v úvahu naráz. Jedná se o komplexní řešení, ve kterém se dobře simulují odrazy, klouby a věrně vypadá i statické tření. Tuto metodu používám pro simulování simultánních odrazů s třením, takže se o ní čtenář více dozví v následujících kapitolách.

Kapitola 3 – Knihovna libPE

V této kapitole se čtenář seznámí s praktickou implementací 2D fyzikálního simulátoru. Knihovna libPE byla vyvinuta pro účely bakalářské práce. Z toho vyplývá, že zvolené implementace v simulátoru, jsou směřovány pro účely simulace robota. Nicméně knihovna libPE je napsaná tak obecně, že lze použít i pro jiné praktické účely.

V příštích kapitolách bude čtenář seznámen s implementací každé části fyzikálního simulátoru, počínaje detekcí kolizí až po implementaci kloubů.

3.1 Modul pro detekci kolizí

V knihovně jsem implementoval detektor kolizí, který se umí vypořádat se situací, kdy tělesa jsou již v sobě zanořená. Detektor je založený na algoritmu, kterému se říká *metoda separovaných os.* Algoritmus je velmi robustní a dokáže být snadno rozšířen na detektor, který dokáže i kolize předpovídat.

Jediná jeho nevýhoda je, že funguje pouze pro konvexní polygony. Konvexní polygon ve 2D je definován tak, že každý jeho vnitřní úhel je menší než 180 stupňů. Toto omezení není tak omezující, protože každý nekonvexní polygon lze rozložit na trojúhelníky, které již konvexní jsou. Nicméně triangulace polygonů v knihovně není implementovaná.

3.1.1 Metoda separujících os

Metoda separujících os je založena na myšlence, že dva konvexní polygony se neprotínají, právě tehdy když lze rovinu, v které polygony leží, rozdělit přímkou na dvě poloroviny tak, že obě tělesa leží v různých polorovinách. Této přímce se potom říká *separující osa* těchto polygonů.

Počet separovaných os může být značné množství. Nicméně lze dokázat, že stačí otestovat pouze ty přímky, které procházejí hranami těles. Pokud alespoň u jedné přímky zjistíme, že je separující osou, potom řekneme, že se polygony neprotínají.

Test, zda je přímka separující osa testovaných těles, probíhá tak, že se obě tělesa promítnou na přímku kolmou na testovanou osu. Pokud se intervaly obou promítaných těles na kolmici testované přímky neprotínají, potom testovaná přímka je separující osa a tělesa se neprotínají.

Pokud se tělesa protínají, potom ani na jedné ose nejsou promítnuté intervaly těles disjunktní. Jako kolizní hrana se bere ta hrana tělesa, na jejíž ose se intervaly protínají nejméně. Tato minimální délka společného intervalu určuje vzdálenost, o kterou je potřeba tělesa od sebe posunout ve směru kolizní normály, aby se neprotínala, ale pouze dotýkala. Tato informace umožňuje detektoru tělesa od sebe odsunout.

Bližší informace o algoritmu jsou uvedeny v [4].

3.2 Modul pro řešení kolizí

Modul pro výpočtu impulsů pro odrazy je v knihovně libPE implementován na základě analytických metod. Tato metoda byla zvolena, protože umožňuje řešení simultánních odrazů a dobrého řešení statického tření, které je pro simulování robota podstatné. Bez dobrého tření by se robot ve scéně nedokázal věrně pohybovat.

Při vysvětlování postupu pro výpočet impulsů, které je potřeba aplikovat na místech odrazů ve 2D systému, se nejdříve zaměřím na případ separovaného odrazu. Poté příklad rozšířím na výpočet impulsů pro simultánní odrazy a nakonec přidám tření.

Je dobré podotknout, že existují články, které se touto problematikou zabývají (jedná se např. o [5] ,[6] nebo [7]), tyto články jsou ale napsány pro 3D systémy.

3.2.1 Nejprve jeden odraz bez tření

Jak už bylo napsáno, při kolizním kontaktu tělesa bez tření, je třeba na tělesa aplikovat impuls $\vec{\psi}_N(t) = j \cdot \vec{n}(t)$. Místo i směr aplikování impulsu známe. Jde nám tedy o to vypočítat velikost impulsu, tedy skalár j.

Kvůli přehlednosti budu v následujících výpočtech místo $\vec{v}(t), \vec{\psi}_N(t), \vec{n}(t)$ psát pouze v, ψ_N, n atd…

Pro přehlednost zopakuji ještě jednou značení, které k výpočtu budeme potřebovat:

n	*kolizní normála*
P	*kolizní bod ve world space*
r^{AP}, r^{BP}	*vzdálenost ' kolizní bod – těžiště tělesa '*
$v_n = v_{AB} \cdot n = (v_{AP} - v_{BP}) \cdot n$	*relativní normálová rychlost bodů*
$v'_A = v_A + \frac{\psi}{m_A}$	*změna lineární rychlosti po aplikaci impulsu* ψ

$\omega'_A = \omega_A + \frac{r_\perp \cdot \psi}{I_A}$ *změna úhlové rychlosti po aplikaci impulsu* ψ

$v_{AP}, v'_{AP}, v_{BP}, v'_{BP}$ *rychlost bodu kontaktu na tělesa A(B) před a po aplikování impulsu* ψ

Rychlost kolizního bodu na tělese *A* a *B* před aplikováním impulsu je:

$$v_{AP} = v_A + r_\perp^{AP} \cdot \omega_A \qquad v_{BP} = v_B + r_\perp^{BP} \cdot \omega_B$$

Rychlost kolizního bodu na tělese A po aplikování impulsu $\psi = j \cdot n$ bude:

$$v'_{AP} = v'_A + r_\perp^{AP} \cdot \omega'_A = v_A + \frac{j \cdot n}{m_A} + r_\perp^{AP} \cdot \left(\omega_A + \frac{r_\perp^{AP}(j \cdot n)}{I_A} \right)$$

$$v'_{AP} = v_A + r_\perp^{AP} \cdot \omega_A + \frac{j \cdot n}{m_A} + r_\perp^{AP} \cdot \left(r_\perp^{AP}(j \cdot n) \right)$$

$$v'_{AP} = v_{AP} + j \cdot \left(\frac{n}{m_A} + \frac{r_\perp^{AP}\left(r_\perp^{AP} \cdot n\right)}{I_A} \right) = v_{AP} + j \cdot C_A \text{ ,kde } C_A \text{ je konstantní vektor (12)}$$

Stejným způsobem se určí rychlost kolizního bodu na tělese B po aplikování impulsu $-\psi = -j \cdot n$ (zákon akce a reakce):

$$v'_{BP} = v_{BP} - j \cdot \left(\frac{n}{m_B} + \frac{r_\perp^{BP}\left(r_\perp^{BP} \cdot n\right)}{I_B} \right) = v_{BP} - j \cdot C_B \text{ ,kde } C_B \text{ je konstantní vektor (13)}$$

Relativní normálová rychlost kolizních bodů po aplikování impulsu je dána vztahem:

$$v'_n = n \cdot \left(v'_{AP} - v'_{BP}\right) = n \cdot \left(v_{AP} + j \cdot C_A - v_{BP} + j \cdot C_B\right) = n \cdot \left(v_{AP} - v_{BP}\right) + n \cdot \left(j \cdot C_A + j \cdot C_B\right)$$

$$v'_n = v_n + j \cdot n\left(C_A + C_B\right) \qquad (14)$$

Nyní podle našeho modelu *Coefficient of restitution* má pro relativní normálové rychlosti platit vztah *(7)*, pokud do něj dosadíme z rovnice *(14)*:

$$v'_n = -e \cdot v_n$$

$$v_n + j \cdot \left[n \cdot \left(C_A + C_B\right)\right] = -e \cdot v_n$$

$$j = \frac{-e \cdot v_n - v_n}{n \cdot \left(C_A + C_B\right)} = \frac{-\left(1+e\right) \cdot v_n}{n \cdot \left(C_A + C_B\right)}$$

Po dosazeních konstantních vektorů C_A a C_B, vyjde velikost impulsu následovně:

$$j = \frac{-\left(1+e\right) \cdot v_n}{\left(n \cdot n\right) \cdot \left(\frac{1}{m_A} + \frac{1}{m_B} \right) + \frac{\left(r_\perp^{AP} \cdot n\right)^2}{I_A} + \frac{\left(r_\perp^{BP} \cdot n\right)^2}{I_B}} \qquad (15)$$

Nyní stačí aplikovat impuls $\vec{\psi}_N(t) = j \cdot \vec{n}(t)$ na těleso A a impuls $-\vec{\psi}_N(t) = -j \cdot \vec{n}(t)$ na těleso B.

3.2.2 Simultánní odrazy bez tření

V této kapitole definujeme úlohu, pomocí které vypočteme velikosti impulsů, které je třeba aplikovat v místech kontaktů.

Vztah *(14)* z minulé kapitoly ukazuje vztah mezi relativní normálovou rychlostí po a před aplikováním impulsu v místě kontaktu. Představme si, že ve scéně je nyní ***k* kolizí**. ***Rychlost tělesa (a tedy i kolizního bodu) ovlivňují všechny impulsy, které působí na všech kolizních bodech onoho tělesa***. Proto relativní normálovou rychlost *i-tého* kontaktu ovlivňují všechny impulsy generované ostatními kolizemi na tělese.

Vztah *(14)* musíme proto pro *i-tý* kolizní kontakt přepsat do tvaru:

$$v'_{n_i} = v_{n_i} + \sum_{l=1}^{K} j_l \cdot n_i \left(C_{A_{il}} + C_{B_{il}} \right) \qquad (16)$$

kde K je počet impulsů, které ovlivňují rychlost kontaktu i. $K \leq k$

Vektory $C_{A_{il}}, C_{B_{il}}$ udávají změnu rychlosti *i-tého* bodu po aplikování impulsu o velikosti 1 na místě *l-tého* kontaktu. Jedná se o stejné konstanty jako ve vztahu *(14)*. Rozdíl je pouze v tom, že nyní rozlišujeme místo aplikování impulsu. Ve vztahu *(14)* byly tyto konstanty bez indexu a určovaly změnu rychlosti bodu po aplikování impulsu v tomto bodě.

Proto nyní změna rychlosti *x-tého* kolizního bodu tělesa T po aplikování jednotkového impulsu na místě *y-tého* kolizního bodu bude:

$$C_{T\,xy} = \frac{n_y}{m_T} + \frac{r_{\perp}^{AP_x} \left(r_{\perp}^{AP_y} \cdot n_y \right)}{I_T} \quad \text{,kde } P_i \text{ je pozice i-tého kontaktu. (17)}$$

Vztah *(16)* můžeme zapsat do tvaru:

$$v'_{n_i} = v_{n_i} + \sum_{l=1}^{k} j_l \cdot \mathbf{A}_{il} \quad \text{,kde A je matice } A \in \Re^{nxn} \quad (18)$$

Vztah *(7)* podle modelu *Coefficient of restitution*, musíme změnit na nerovnost, jelikož se může stát, že nějaký jiný mnohem silnější impuls může ovlivnit rychlost tělesa. Proto pro *i-tý kolizní* bod musí platit nerovnost:

$$v'_{n_i}(t) \geq -e \cdot v_{n_i}(t) \text{ , kde } e \in <0,1> \qquad (19)$$

Pokud pro i-tý bod bude platit ostrá nerovnost $v'_{n_i}(t) > -e \cdot v_{n_i}(t)$, potom to znamená, že nějaký silnější impuls překonal účinky impulsu na *i-tém* kontaktu. Tato situace implikuje, že velikost impulsu na pozici *i-tého* kontaktu musí být nulová. Tato podmínka se může zapsat ve tvaru:

$$j_i \cdot \left(v'_{n_i} + e \cdot v_{n_i}\right) = 0 \qquad (20)$$

Dosadíme-li do nerovnice *(19)* pomocí *(18)* dostaneme:

$$v'_{n_i} \geq -e \cdot v_{n_i}$$

$$v_{n_i} + \sum_{l=1}^{k} j_l \cdot A_{il} \geq -e \cdot v_{n_i}(t)$$

$$\sum_{l=1}^{k} j_l \cdot \mathbf{A}_{il} + v_{n_i} \cdot (1+e) \geq 0$$

$$\mathbf{A} \cdot \mathbf{x} + \mathbf{b} \geq \mathbf{0} \quad ,kde\ \mathbf{A} \in \Re^{kxk}, \mathbf{x} \in \Re^k,\ \mathbf{b} \in \Re^k \qquad (21)$$

Složky vektoru $\mathbf{x}$ určují velikosti impulsů, které se mají aplikovat v místech kontaktů.

Pro matici $\mathbf{A}$ tedy platí:

$\mathbf{A}_{ij} = 0$ *pokud j-tý impuls neovlivňuje rychlost i-tého kontaktu.*

$\mathbf{A}_{ij} = n_i \cdot \left(C_{A_{ij}} + C_{B_{ij}}\right)$ *pokud j-tý impuls ovlivňuje rychlost i-tého kontaktu.*

Pro vektor $\mathbf{b}$ platí:

$\mathbf{b}_i = v_{n_i} \cdot (1+e)$

Dále pro nerovnost *(21)* musí platit podmínky:

$\mathbf{x} \geq \mathbf{0}$ *velikost každého impulsu musí být nezáporná*

$\mathbf{x}^T \cdot (\mathbf{A} \cdot \mathbf{x} + \mathbf{b}) = \mathbf{0}$ *viz podmínka (20).*

Úloha *(21)* s výše uvedenými podmínkami se nazývá *Linear Complementarity Problem (LCP).* V knihovně libPE je naimplementován iterační algoritmus pro řešení LCP. O detailech implementace je napsáno v příslušné kapitole.

Po vyřešení úlohy *(21)* známe velikosti impulsů, které je třeba aplikovat v místech kontaktů.

3.2.3 Simultánní odrazy s třením

Nyní rozšíříme úlohu o statické tření, které je implementováno v knihovně libPE. V místě kontaktu budou nyní působit dva impulsy. Jeden impuls $\vec{\psi}_N(t) = j \cdot \vec{n}(t)$ ve

směru kolizní normály a druhý impuls $\vec{\psi}_T(t) = t \cdot \vec{n}_\perp(t)$ ve směru tečném s kolizní hranou. Viz *obr. (3)* v kapitole 2.4.4.

Úloha, kterou odvodíme, bude podobná úloze *(21)*. Hlavní rozdíl je v tom, že nyní musíme zkoumat pro každý bod jak jeho relativní normálovou rychlost v_{n_i}, tak relativní tečnou normálovou rychlost v_{t_i}. Budeme postupovat podobným způsobem.

Rychlost kolizního bodu *P* na tělese *A*, po aplikování normálového impulsu $\vec{\psi}_N(t)$ a současně třecího impulsu $\vec{\psi}_T(t)$ v tomto bodě, bude:

$$v'_{AP} = v'_A + r_\perp^{AP} \cdot \omega'_A = v_A + \frac{\psi_N + \psi_T}{m_A} + r_\perp^{AP} \cdot \left(\omega_A + \frac{r_\perp^{AP}(\psi_N + \psi_T)}{I_A} \right)$$

$$v'_{AP} = v_{AP} + \frac{\psi_N}{m_A} + r_\perp^{AP} \cdot \left(\frac{r_\perp^{AP}(\psi_N)}{I_A} \right) + \frac{\psi_T}{m_A} + r_\perp^{AP} \cdot \left(\frac{r_\perp^{AP}(\psi_T)}{I_A} \right)$$

$$v'_{AP} = v_{AP} + \frac{j \cdot n}{m_A} + r_\perp^{AP} \cdot \left(\frac{r_\perp^{AP}(j \cdot n)}{I_A} \right) + \frac{t \cdot n_\perp}{m_A} + r_\perp^{AP} \cdot \left(\frac{r_\perp^{AP}(t \cdot n_\perp)}{I_A} \right)$$

$$v'_{AP} = v_{AP} + j \cdot \left[\frac{n}{m_A} + \frac{r_\perp^{AP} \cdot (r_\perp^{AP} \cdot n)}{I_A} \right] + t \cdot \left[\frac{n_\perp}{m_A} + \frac{r_\perp^{AP} \cdot (r_\perp^{AP} \cdot n_\perp)}{I_A} \right]$$

$$v'_{AP} = v_{AP} + j \cdot C_A + t \cdot T_A \qquad (22)$$

Analogicky rychlost kolizního bodu *P* na tělese *B* po aplikování normálového impulsu $-\vec{\psi}_N(t) = -j \cdot \vec{n}(t)$ a současně třecího impulsu $-\vec{\psi}_T(t) = -t \cdot \vec{n}_\perp(t)$ v tomto bodě bude:

$$v'_{BP} = v_{BP} - j \cdot C_B - t \cdot T_B \qquad (23)$$

kde v obou případech C_A, T_A, C_B, C_B *jsou konstantní vektory.*

Relativní **normálová** rychlost kolizního bodu po aplikování impulsů je potom dána vztahem:

$$v'_n = n \cdot (v'_{AP} - v'_{BP}) = n \cdot (v_{AP} + j \cdot C_A + t \cdot T_A - v_{BP} + j \cdot C_B + t \cdot T_B)$$

$$v'_n = v_n + n(j \cdot C_A + j \cdot C_B + t \cdot T_A + t \cdot T_B) \qquad (24)$$

Relativní **tečná** rychlost kolizního bodu po aplikování impulsů je podobně dána vztahem:

$$v'_t = n_\perp \cdot (v'_{AP} - v'_{BP}) = n_\perp \cdot (v_{AP} + j \cdot C_A + t \cdot T_A - v_{BP} + j \cdot C_B + t \cdot T_B)$$

$$v'_t = v_t + n_\perp \cdot (j \cdot C_A + j \cdot C_B + t \cdot T_A + t \cdot T_B) \qquad (25)$$

Nyní, stejně jako v předešlém případě, kdy rychlost kolizních bodů ovlivňují i jiné impulsy, musíme indexovat konstanty C_A, T_A, C_B, T_B.

Proto nyní změna rychlosti *x-tého* kolizního bodu tělesa *T* po aplikování jednotkového **normálového** impulsu na místě *y-tého* kolizního bodu je:

$$C_{T\,xy} = \frac{n_y}{m_T} + \frac{r_{\perp}^{AP_x}\left(r_{\perp}^{AP_y} \cdot n_y\right)}{I_T} \qquad (26)$$

kde P_i je pozice i-tého kontaktu a n_y je kolizní normála y-tého bodu.

A podobně změna rychlosti *x-tého* kolizního bodu tělesa *T* po aplikování jednotkového **tečného** impulsu na místě *y-tého* kolizního bodu je:

$$T_{T\,xy} = \frac{\left(n_y\right)_{\perp}}{m_T} + \frac{r_{\perp}^{AP_x}\left(r_{\perp}^{AP_y} \cdot \left(n_y\right)_{\perp}\right)}{I_T} \qquad (27)$$

kde P_i je pozice i-tého kontaktu a n_y je kolizní normála y-tého bodu.

Máme-li *K* tečných a normálových impulsů, které ovlivňují relativní rychlost *i-tého* kolizního bodu, potom **relativní normálová rychlost** tohoto bodu po aplikování všech impulsů, které ovlivňují jeho rychlost, bude:

$$\begin{aligned} v'_{n_i} &= v_{n_i} + n_i\left[\sum_{l=1}^{K}\left(j_l \cdot C_{A_{il}} + j_l \cdot C_{B_{il}}\right) + \sum_{l=1}^{K}\left(t_l \cdot T_{A_{il}} + t_l \cdot T_{B_{il}}\right)\right] \\ v'_{n_i} &= v_{n_i} + n_i\left[\sum_{l=1}^{K} j_l \cdot \left(C_{A_{il}} + C_{B_{il}}\right) + \sum_{l=1}^{K} t_l \cdot \left(T_{A_{il}} + T_{B_{il}}\right)\right] \end{aligned} \qquad (28)$$

Analogicky pro **relativní tečnou rychlost** i-tého kolizního bodu platí:

$$v'_{t_i} = v_{t_i} + \left(n_i\right)_{\perp} \cdot \left[\sum_{l=1}^{K} j_l \cdot \left(C_{A_{il}} + C_{B_{il}}\right) + \sum_{l=1}^{K} t_l \cdot \left(T_{A_{il}} + T_{B_{il}}\right)\right] \qquad (29)$$

Soustavy *(28) a (29)* můžeme zapsat v maticovém tvaru následovně:

$$\mathbf{v}' = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}' \qquad (30)$$

$$\mathbf{v}' = \left(v'_{n_1},\ v'_{t_1},\ \ldots,\ v'_{n_k},\ v'_{t_k}\right) \in \Re^{2kx2k}$$

$$\mathbf{b} = \left(v_{n_1},\ v_{t_1},\ \ldots,\ v_{n_k},\ v_{t_k}\right) \in \Re^{2k}$$

$$\mathbf{x} = \left(j_1,\ t_1,\ \ldots,\ j_k,\ t_k\right) \in \Re^{2k}$$

$$\mathbf{A} \in \Re^{2kx2k}$$

Pro normálové rychlosti musí opět platit *(19),* takže pokud *(28)* dosadíme do *(19)* normálové složky v_{n_i} vektoru **b** se změní na $(1+e)\cdot v_{n_i}$.

Úloha *(30)* se musí řešit pod podmínkami, které zaručí, že velikosti normálových a tečných impulsů budou splňovat *Coloumbův model tření* z kapitoly 2.4.4.

Proto se úloha *(30)* musí řešit s podmínkami:

$$|t_i| \le \mu \cdot j_i, \quad v_{t_i} \cdot t_i < 0 \text{ a zároveň } v'_{t_i}\left(|t_i| - \mu \cdot j_i\right) = 0 \text{ ,kde } \mu \text{ je koef. stat. tření} \quad (31)$$

První podmínka omezuje velikost třecí síly. První část druhé podmínky říká, že třecí síla musí působit v opačném směru, než v jakém se těleso pohybovalo (má ho zpomalovat). Druhá část podmínky popisuje poslední vlastnost statického tření z kapitoly 2.4.4.

Po vyřešení úlohy *(30)* budeme znát velikosti normálových a třecích (tečných) impulsů, které je nutné aplikovat v místech kolizí.

Problémem je, že úloha *(30)* řešená pod podmínkami *(31)* není LCP problémem. Nicméně přibližné řešení je nastíněno v příští kapitole.

3.2.4 Řešení LCP

Obecný LCP problém je definován:

Pro danou symetrickou matici $\mathbf{A} \in \Re^{n\times n}$, *vektor* $\mathbf{b} \in \Re^n$, *limitní vektory* $\mathbf{lo}, \mathbf{hi} \in \Re^n$ *takové, že* $\mathbf{lo} \le \mathbf{0}$ *a* $\mathbf{hi} \ge \mathbf{0}$, *nalezni vektor* $\mathbf{x} \in \Re^n$ *a* $\mathbf{w} \in \Re^n$, *tak aby platilo:*

$$\begin{aligned} \mathbf{w} &= \mathbf{A}\cdot\mathbf{x} + \mathbf{b} \\ \mathbf{lo} &\le \mathbf{x} \le \mathbf{hi} \end{aligned} \qquad (32)$$

a dále pro každé i=1,…,n byla splněna alespoň jedna z podmínek

$$\begin{aligned} x_i &= lo_i, w_i \ge 0 \\ x_i &= hi_i, w_i \le 0 \\ lo_i &< x_i < hi_i, w_i = 0 \end{aligned}$$

Pokud je $\mathbf{lo} = \mathbf{0}$ a $\mathbf{hi} \approx \infty$, potom podmínky pro $\mathbf{x}$ vypadají takto:

$$\begin{aligned} x_i &= 0, w_i \ge 0 \\ x_i &\approx \infty, w_i \le 0 \\ 0 &< x_i, w_i = 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} &\mathbf{w} = \mathbf{A}\cdot\mathbf{x} + \mathbf{b} \ge \mathbf{0} \\ &\mathbf{0} \le \mathbf{x} \\ &\mathbf{x}^{\mathrm{T}} \cdot \mathbf{w} = \mathbf{0} \end{aligned} \qquad (33)$$

Takto zadaná úloha *(33)* je matematicky ekvivalentní s úlohou *(21)* pro řešení simultánních odrazů bez tření.

Iterativní algoritmus pro řešení takto zadané úlohy lze nalézt v [8]. Ve stejném článku je i stručný popis rozšíření tohoto algoritmu pro řešení úlohy *(30)* pro třecí síly. Nicméně Baraff nedokázal konečnost tohoto rozšířeného algoritmu. Tvrdí však, že při testování nenarazil na žádnou konfiguraci těles ve scéně, tak aby se algoritmus zacyklil.

Já jsem v knihovne libPE rozšířil Baraffův základní algoritmus pro řešení úlohy *(21)* na řešení obecného LCP problému podle definice *(32).*

Tento rozšířený algoritmus jsem poté použil pro přibližné řešení úlohy s třením *(30)*.

Myšlenka přibližného řešení je následující:

Nejdříve vypočtu velikost normálových impulsů pro systém bez tření. Poté změním podmínky úlohy *(30)* na podmínky úlohy *(32)* tak, že velikosti třecích sil omezím podle velikostí spočtených normálových impulsů a koeficientu tření. Takto zadanou úlohu *(30)* už umím řešit.

Algoritmus nastíním na ukázce. Mějme v systému *k* kolizí.

1. *Vypočti matici* $\mathbf{A} \in \Re^{k \times k}$ *a vektor* $\mathbf{b} \in \Re^{k}$ *pro úlohu (21)*
2. *Vypočti vektor* $\mathbf{x} \in \Re^{k}$ *velikosti impulsů z úlohy (21)*
3. *Vypočti matici* $\mathbf{A} \in \Re^{2k \times 2k}$ *a vektor* $\mathbf{b} \in \Re^{2k}$ *pro úlohu (30)*
4. *Omezení velikosti třecích impulsů (nastavení vektorů* **lo**, **hi***):*

   for (i=0; i<k; i++)

   *lo[2*i] = 0; hi[2*i] = ∞;* *// jedná se o normálovou složku*

   *hi[2*i+1] =* μ **x[i];* *// omezíme velikosti třecích sil*

   *lo[2*i+1] = -* μ **x[i];*
5. *Vypočti úlohu (30) s podmínkami úlohy(32) (použij spočtené vektory* **lo**, **hi***)*
6. *Aplikuj impulsy*

Takto vypočtené impulsy se liší od správného řešení úlohy *(30)* jen minimálně. Velikosti třecích impulsů jsou pravděpodobně o něco větší než by měly být, jelikož se omezily velikostí normálových impulsů bez tření. Nicméně tento algoritmus pro přibližné řešení úlohy *(30)* je konečný, právě tehdy, když jsou konečné alogritmy pro řešení úloh *(32)* a *(33)* - a ty jsou.

3.3 Modul pro řešení kloubů

V knihovně libPE jsem zvolil pro implementaci kloubů metodu založenou na aplikování impulsů. Tuto metodu jsem zvolil, protože je velmi robustní a pro simulování robota je dostačující. Tato metoda dokáže udržovat polygony spojené klouby pohromadě po celý simulovaný čas a dokonce dokáže opravit totálně porušené klouby. Tím se myslí, že dokáže úchyty kloubů obou těles za nějaký čas opět spojit. Toto většinou analytické metody nedokážou.

3.3.1 Metoda založená na impulsech

Algoritmus pro implementaci kloubů v 3d systému založenou na aplikování impulsů lze nalézt v [9]. Já zde vysvětlím princip fungování algoritmu a odvození rovnic pro 2D systém.

Princip algoritmu ukážu na příkladu. Představme si kloub, k němuž jsou přichyceny tělesa *A* a *B*. Zavedu značení:

$\vec{P}_A(t)$	*pozice bodu na A, který je součástí kloubu, ve world space*
$\vec{P}_B(t)$	*pozice bodu na B, který je součástí kloubu, ve world space*
$\vec{v}_{AP}(t)$	*rychlost bodu* $\vec{P}_A(t)$ *tělesa A*
$\vec{v}_{BP}(t)$	*rychlost bodu* $\vec{P}_B(t)$ *tělesa B*
$\vec{P}_A(t) = \vec{P}_B(t)$	*požadovaný konzistentní stav kloubu*

Algoritmus je založen na aplikování opravných impulsů na tělesa *A* a *B* v čase *t* v místech kloubů a předpovídání pozic bodů kloubů v čase *t+h*. Pokud platí, že v příštím simulačním kroku bude kloub v konzistentním stavu, poté není třeba aplikovat korektní impuls. Pokud ale kloub v konzistentním stavu v čase *t+h* nebude, neboli by platilo $\vec{P}_A(t+h) \neq \vec{P}_B(t+h)$, potom je ještě třeba v čase *t* aplikovat správně spočtený korektní impuls v místech kloubů tak, aby v příštím simulačním kroku byl kloub v konzistentním stavu – tak aby platilo: $\vec{P}_A(t+h) = \vec{P}_B(t+h)$.

Nechť tedy v čase *t+h* jsou body $\vec{P}_A(t+h)$ a $\vec{P}_B(t+h)$ od sebe vzdálené $\vec{d} = \vec{P}_A(t+h) - \vec{P}_B(t+h)$. Musíme proto v čase *t* v bodech $\vec{P}_A(t)$ a $\vec{P}_B(t)$ aplikovat impulsy $\vec{\psi}(t)$ a $-\vec{\psi}(t)$, tak aby platilo $\vec{0} = \vec{P}_A(t+h) - \vec{P}_B(t+h)$.

Známe: $\vec{P}_A(t)$, $\vec{P}_B(t)$, h.

Pozice bodu $\vec{P}_A(t+h)$ v čase $t+h$ po aplikování impulsu $\vec{\psi}(t)$ v čase t bude:

$$\begin{aligned}
\vec{P}'_A(t+h) &= \vec{P}_A(t) + h\cdot\left(v_{AP_A}(t) + \Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right)\right)\\
\vec{P}'_A(t+h) &= \vec{P}_A(t) + h\cdot v_{AP_A}(t) + h\cdot\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right) \qquad (34)\\
\vec{P}'_A(t+h) &= \vec{P}_A(t+h) + h\cdot\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right)
\end{aligned}$$

kde $\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right)$ *je změna rychlosti bodu* $\vec{P}_A(t)$ *po aplikování impulsu* $\vec{\psi}(t)$ *ve stejném bodě.*

Stejně pozice bodu $\vec{P}_B(t+h)$ v čase $t+h$ po aplikování impulsu $-\vec{\psi}(t)$ v čase t bude:

$$\vec{P}'_B(t+h) = \vec{P}_B(t+h) + h\cdot\Delta v\left(\vec{P}_B(t),-\vec{\psi}(t)\right) \qquad (35)$$

kde $\Delta v\left(\vec{P}_B(t),-\vec{\psi}(t)\right)$ *je změna rychlosti bodu* $\vec{P}_B(t)$ *po aplikování impulsu* $-\vec{\psi}(t)$ *ve stejném bodě.*

Nyní chceme, aby $\vec{P}'_A(t+h) - \vec{P}'_B(t+h) = \vec{0}$, dosazením *(34)* a *(35)* dostaneme:

$$\begin{aligned}
&\vec{P}'_A(t+h) - \vec{P}'_B(t+h) = \vec{0}\\
&\vec{P}_A(t+h) + h\cdot\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right) - \vec{P}_B(t+h) - h\cdot\Delta v\left(\vec{P}_B(t),-\vec{\psi}(t)\right) = \vec{0}\\
&\vec{d} + h\cdot\left(\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right) - h\cdot\Delta v\left(\vec{P}_B(t),-\vec{\psi}(t)\right)\right) = \vec{0} \qquad (36)\\
&\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right) - \Delta v\left(\vec{P}_B(t),-\vec{\psi}(t)\right) = -\frac{\vec{d}}{h}
\end{aligned}$$

Nyní odvodíme změnu rychlosti bodu $\vec{P}_A(t)$ po aplikování impulsu $\vec{\psi}(t)$ v bodě $\vec{P}_A(t)$. V odvození nebudu kvůli přehlednosti používat proměnné jako funkce času, jelikož se všechny vztahují k času t.

$$\begin{aligned}
\Delta v\left(\vec{P}_A,\vec{\psi}\right) &= \frac{\vec{\psi}}{m_A} + \frac{r_\perp\cdot\left(r_\perp\cdot\vec{\psi}\right)}{I_A} = \frac{\vec{\psi}}{m_A} + \frac{1}{I_A}\cdot\left(-r_y\psi_x + r_x\psi_y\right)\cdot\begin{pmatrix}-r_y\\ r_x\end{pmatrix}\\
\Delta v\left(\vec{P}_A,\vec{\psi}\right) &= \frac{\vec{\psi}}{m_A} + \frac{1}{I_A}\cdot\begin{pmatrix}r_y^2\psi_x - r_x\psi_y r_y\\ -r_y\psi_x r_x + r_x^2\psi_y\end{pmatrix}\\
\Delta v\left(\vec{P}_A,\vec{\psi}\right) &= \frac{\vec{\psi}}{m_A} + \frac{1}{I_A}\cdot\begin{bmatrix}r_y^2 & -r_x r_y\\ -r_x r_y & r_x^2\end{bmatrix}\cdot\begin{pmatrix}\psi_x\\ \psi_y\end{pmatrix} \qquad (37)\\
\Delta v\left(\vec{P}_A,\vec{\psi}\right) &= \frac{\vec{\psi}}{m_A} + \frac{1}{I_A}\cdot\mathbf{C}_A\cdot\vec{\psi} = \left(\frac{1}{m_A}\mathbf{E} + \frac{1}{I_A}\mathbf{C}_A\right)\cdot\vec{\psi}
\end{aligned}$$

kde $\vec{r}(t)$ *je vzdálenost* $\vec{P}_A(t)$ *od těžiště A a matice* $\mathbf{C}_A(t)$ *je konstanta pro těleso A.*

Obdobně pro změnu rychlosti bodu $\vec{P}_B(t)$ po aplikování impulsu $-\vec{\psi}(t)$ v bodě $\vec{P}_B(t)$ platí:

$$\Delta v\left(\vec{P}_B,-\vec{\psi}\right)=\frac{-\vec{\psi}}{m_B}+\frac{1}{I_B}\cdot\mathbf{C}_B\cdot\left(-\vec{\psi}\right)=\left(\frac{1}{m_B}\mathbf{E}+\frac{1}{I_B}\mathbf{C}_B\right)\cdot\left(-\vec{\psi}\right) \qquad (38)$$

Po dosazení *(38)* a *(37)* do *(36)* dostaneme:

$$\begin{aligned}
&\Delta v\left(\vec{P}_A(t),\vec{\psi}(t)\right)-\Delta v\left(\vec{P}_B(t),-\vec{\psi}(t)\right)=-\frac{\vec{d}}{h}\\
&\left(\frac{1}{m_A}\mathbf{E}+\frac{1}{I_A}\mathbf{C}_A\right)\cdot\vec{\psi}-\left(\frac{1}{m_B}\mathbf{E}+\frac{1}{I_B}\mathbf{C}_B\right)\cdot\left(-\vec{\psi}\right)=-\frac{\vec{d}}{h}\\
&\left(\frac{1}{m_A}\mathbf{E}+\frac{1}{I_A}\mathbf{C}_A+\frac{1}{m_B}\mathbf{E}+\frac{1}{I_B}\mathbf{C}_B\right)\cdot\vec{\psi}=-\frac{\vec{d}}{h}\\
&\left(\left(\frac{1}{m_A}+\frac{1}{m_B}\right)\mathbf{E}+\frac{1}{I_A}\mathbf{C}_A+\frac{1}{I_B}\mathbf{C}_B\right)\cdot\vec{\psi}=-\frac{\vec{d}}{h}\\
&\mathbf{D}\cdot\vec{\psi}=-\frac{\vec{d}}{h}\\
&\vec{\psi}=-\frac{\vec{d}}{h}\cdot\mathbf{D}^{-1}
\end{aligned} \qquad (39)$$

Rovnice *(39)* nám ukazuje jak vypočítat korektní impuls $\vec{\psi}(t)$, který je třeba aplikovat na tělesa, aby kloub byl v příštím simulačním kroku v konzistentním stavu.

Pokud je v systému více jak jeden kloub, potom korekční impulsy musejí být spočteny pro každý kloub a po jejich aplikování musí pokračovat testování zda jsou již všechny v konzistentním stavu. Pokud ne, aplikační impulsy je třeba znovu spočítat a aplikovat. Pomocí tohoto iterativního algoritmu se dosáhne konzistentnosti všech kloubů.

Po aplikování impulsů v čase *t* je sice kloub v konzistentním stavu, ale relativní rychlost bodů na kloubu je nenulová. Proto se na začátku simulačního kroku, který následuje po kroku v němž byly aplikovány korektní impulsy, aplikují ještě další korekční impulsy, které způsobí, že relativní rychlost bodů na kloubech bude nula.

Výpočet těchto impulsů je podobný výpočtu impulsů na úpravu pozice. Nyní jsme na začátku dalšího simulačního kroku po aplikování impulsů na korekturu pozice, tedy v čase *t*+*h*:

Víme: $\vec{v}_{AP}(t+h)-\vec{v}_{BP}(t+h)=\vec{d}$

Chceme, aby po aplikování impulsů byla relativní rychlost bodů nulová, tedy:

$$\begin{aligned}
&\vec{v}'_{AP} - \vec{v}'_{BP} = \vec{0} \\
&\vec{v}_{AP} + \Delta v(\vec{P}_A, \vec{\psi}) - \vec{v}_{BP} - \Delta v(\vec{P}_B, -\vec{\psi}) = \vec{0} \\
&\vec{d} + \Delta v(\vec{P}_A, \vec{\psi}) - \Delta v(\vec{P}_B, -\vec{\psi}) = \vec{0} \qquad (40) \\
&\Delta v(\vec{P}_A, \vec{\psi}) - \Delta v(\vec{P}_B, -\vec{\psi}) = -\vec{d} \\
&\vec{\psi} = -\vec{d} \cdot \mathbf{D}^{-1}
\end{aligned}$$

kde matice $\mathbf{D}(t+h)$ *je stejná konstanta jako ve vztahu (39), jen v čase t+h.*

3.4 Programátorská dokumentace

Na přiloženém cd-disku je programátorská dokumentace pro knihovnu libPE, která obsahuje seznam všech souborů, ze kterých je knihovna vytvořena. Také obsahuje popis všech tříd v knihovně libPE. Vysvětlení hlavních principů a algoritmů pro simulaci tuhých těles je však ukázáno v této práci.

Kapitola 4 – Aplikace Simulování Robota

V této kapitole seznámím čtenáře se samotnou aplikací pro simulování pohybu robota. Jak už bylo řečeno - aplikace používá pro fyzikální simulaci výše popsanou knihovnu libPE.

Aplikace, stejně jako knihovna libPE, je napsána v programovacím jazyce C++. Na vytvoření grafického uživatelského prostředí jsem použil na platformě nezávislou knihovnu wxWidgets. Aplikace proto lze spustit jak na operačním systému Windows, tak systémech Linux/UNIX i na strojích se systémem MacOS (viz kapitola wxWidgets).

V příštích kapitolách se zmíním o tom, jaká úskalí návrh aplikace přinášel. Popíši, jak definuji roboty v aplikaci, jak se zadává pohyb pro roboty, jaké datové struktury jsem pro aplikaci definoval apod.

4.1 Tělesa a Roboti

Těleso je v aplikaci definováno stejně jako tuhé těleso v knihovně libPE. Jeho tvar a objem je určen množinou vrcholů. Hmotnost a moment setrvačnosti tělesa jsou vypočítány pomocí zvolené hustoty a tvaru tělesa, který je dán jeho vrcholy. Poslední parametr tělesa je *coeficient of resitution*, který je popsán v kapitole 2.4.2.

Robot je v aplikaci definován jako množina těles a množina kloubů, pro které platí, že nemůže nastat situace, že některé těleso není k ostatním tělesům připojeno kloubem a dále platí, že každý kloub z množiny kloubů spojuje jenom tělesa z množiny těles robota.

Pro robota se definují motory, pomocí nichž se může robot pohybovat. Motory se umisťují na jednotlivá tělesa robota. K tělesu robota může být umístěn libovolný počet motorů - tedy i žádný. Motor umístěný na tělesu, má za úkol udržovat jeho úhlovou rychlost na zvolené velikosti po daný časový interval. Interval nám tedy vlastně říká, kdy se má motor zapnout a kdy vypnout. Motor udržuje úhlovou rychlost tělesa na dané hodnotě pomocí sil, které vytvářejí pouze moment síly, tzn., že tělesem pouze otáčejí a nikoliv posouvají. Motor si tedy můžeme představit jako dvojici vzájemně opačných sil, které působí v určité vzdálenosti od těžiště tak, že vytvářejí požadovaný moment síly, který udržuje úhlovou rychlost tělesa na požadované velikosti.

Aby model motorů byl více reálný, zadává se pro každý motor jeho maximální výkon - tj. jak velký moment síly může motor vyvinout. Může se proto stát, že

vložíme-li do nějakého tělesa v robotu příliš slabý motor, který nebude mít dost síly na to, aby tělesem otočil, tak se robot nezačne pohybovat.

Tento model motorů jsem zvolil, protože nejvíce zapadá do celé koncepce simulace v knihovně libPE. Do simulovaného systému vkládáme síly a simulátor poté sám vypočte jak se tělesa budou pohybovat. Také tento model dokáže dobře zadávat pohyb, který konstruktér robota požaduje.

4.2 Scéna

Scéna je prostor, kde se robot pohybuje. Scéna obsahuje množinu robotů a těles. Tělesa mohou ve scéně být nepohyblivá - tj. mají nekonečnou hmotnost a nekonečně velký moment setrvačnosti. Takovými tělesy potom nelze ve scéně pohybovat (žádná síla nedokáže přeci s nimi pohnout) a používají se na vytváření prostředí, ve kterém se robot bude pohybovat. Pro označení takovýchto nepohyblivých těles se v aplikaci nastavuje jejich hustota na nulu.

Pro scénu je také důležité nastavit velikost gravitace a koeficient tření.

Po vytvoření scény stačí začít scénu simulovat pomocí knihovny libPE, která udělá všechnu fyzikální práci za aplikaci.

4.3 Databáze

Ukládání scén, robotů a těles jsem se rozhodl ukládat do databáze, která se skládá ze tří souborů ve formátu xml. V souboru *./data/bodies.xml* jsou uložena pouze tělesa.

Těleso tvaru obdélníku s hustotou 1 a *coeficient of resitution* 0.2 je v souboru uložen v následujícím formátu:

```
<body name="obdelnik" density="1" cor="0.2">
        <vertex x="20" y="-10"/>
        <vertex x="20" y="10"/>
        <vertex x="-20" y="10"/>
        <vertex x="-20" y="-10"/>
</body>
```

Roboti jsou uloženy v souboru *./data/robots.xml*. Jak už bylo řečeno, robot je dán množinou těles a kloubů. Tělesa se v robotovi neukládají přímo, ale odkazuje se na ně do souboru s tělesy pomocí jejich jmen.

Příklad uložení robota s názvem *housenka*, který se skládá ze třech obdélníků, které jsou spojeny dvěma klouby je následující:

```
<robot name="housenka">
<body alias="h_0" source="obdelnik" x="-33" y="0" orientation="0">
        <force time0="0" time1="1000" angular="0.3" startTorque="1e+006"
                maxTorque="2e+006" increase="0.001"/>
  </body>
  <body alias="h_1" source="obdelnik" x="-1" y="0" orientation="0">
  </body>
  <body alias="h_2" source="obdelnik" x="33" y="0" orientation="0">
  </body>
  <joint bodyA="h_0" bodyB="h_1" anchorAx="16" anchorAy="0"/>
  <joint bodyA="h_1" bodyB="h_2" anchorAx="17" anchorAy="-1"/>
</robot>
```

V příkladu je také vidět motor, který je připojen na první těleso, který má za úkol od kroku 0 do kroku 1000 udržovat úhlovou rychlost tohoto tělesa na hodnotě 0.3 radiánů za krok. Úhlovou rychlost se snaží udržet pomocí síly, která vyvine maximální moment síly $2 \cdot 10^6 Nm$. Motor začne působit na těleso momentem o velikosti $1 \cdot 10^6 Nm$ a v každém simulačním kroku, kdy se úhlová rychlost nedostala na požadovanou velikost, zvýší působící moment o *0.001* násobek *startovacího momentu. M*aximálně však do hodnoty $2 \cdot 10^6 Nm$.

Podobně jako uložení roboti v databázi odkazují na tělesa, tak uložená scéna, místo ukládání celých objektů, odkazuje na roboty a tělesa.

Ukládaný formát je proto podobný formátu, ve kterém se ukládají roboti. Musí se ale uložit ještě informace o velikosti gravitace a tření.

Příklad uložení malé scény s robotem *housenka* a s jedním dalším tělesem vypadá následovně:

```
<scene name="testovaci scena" gravitation="9.1" friction="0.7" >
  <robot source=" housenka" x="368.25" y="132.334" orientation="0"/>
  <body alias="o_1" source="obdelnik" x="192" y="262" orientation="0"/>
</scene>
```

Výhoda takovéto databáze spočívá v tom, že pokud vytvoříme jakéhokoliv robota nebo těleso, můžeme tento objekt použít v jakékoliv další scéně. U výše zmíněného robota *housenka*, můžeme testovat jeho pohyb v různých scénách, aniž bychom museli robota znovu vytvářet. Dále méně podstatná výhoda spočívá v ušetření kapacity disku.

Nevýhoda takového způsobu uložení objektů v databázi vyjde najevo pokud se z takovéto databáze objekty mažou. Může nastat situace, že v robotovi odkazujeme na těleso, které již v databázi není. Stejná situace může nastat s uloženým robotem ve scéně.

Nicméně se domnívám, že výhody takového uložení převažují nad nevýhodami a proto jsem zvolil právě takovýto způsob ukládání objektů.

Pro práci s xml jsem naprogramoval knihovnu s názvem *libXmlTool*, která umožňuje parsování, přidávání a mazání záznamů v souboru ve formátu xml.

4.3.1 Knihovna libXmlTool

Knihovna libXmlTool je napsaná v jazyku C a C++. Zdrojový kód pro parsování jazyku xml je vygenerován pomocí programu *flex/lex*. Knihovnu jsem vyvinul pro snadnou práci s xml. Knihovna nepodporuje všechny vlastnosti formátu xml. Chtěl jsem, abych mohl co nejjednodušeji do xml souboru záznamy přidávat a mazat. To jsem zajistil pomocí rozhraní, které musí implementovat každý objekt, který chce být schopen se uložit nebo se smazat z xml souboru (databáze).

Vývoj této knihovny nebyl pro bakalářský projekt podstatný a proto jej zde nebudu více rozebírat.

4.4 Simulace

Samotnou simulaci pohybu robota zajišťuje knihovna libPE. V samotné aplikaci poté musíme v každém simulačním kroku vypočítat jakou silou má motor působit na těleso v robotovi. Klouby zajistí, aby tělesa, z kterých se robot skládá, zůstala spojena.

Algoritmus pro výpočet aktuálního momentu síly, který je třeba na těleso aplikovat, je snadný. Vychází totiž přímo z návrhu motorů pro roboty:

```
foreach ( motor m )
        if ( t > m.time0 && t < m.time1)       // zda je motor zapnutý.
                if ( |uhlová rychlost| < m.rychlost)
                        if (m.aktualniMoment == 0)
                                m.aktualniMoment = m.startovaciMoment;
                        else
                                m.aktualniMoment += m.increase*m.startTorque;
                        if (m.aktualniMoment > m.maxMoment)
                                m.aktualniMoment = m.maxMoment;
                        AplikujMomentSílyNaTěleso(m.aktualniMoment);
                else
                        m.aktualniMoment = 0;
```

4.5 Knihovna wxWidgets

Pro vytvoření grafického prostředí na simulování robota byla použita knihovna wxWidgets. Jedná se o multiplatformní knihovnu napsanou v C++. Umožňuje psaní aplikací pro platformy Linux, Windows (16bit, 32bit), Mac OS, embed zařízení, ...

Pod daným operačním systémem se knihovna snaží využívat co nejvíce nativních prostředků dané platformy. To činí knihovnu přijatelně rychlou a spolehlivou. Další výhodou wxWidgets je velmi podrobná programátorská dokumentace.

Kvůli všem těmto výhodám, jsem pro programování grafického uživatelského prostředí zvolil právě tuto knihovnu.

4.6 Uživatelská příručka

Uživatelská příručka pro ovládání programu pro Simulování Robota je na přiloženém cd-disku. Je v ní všechno potřebné, aby uživatel mohl sám konstruovat nové roboty a simulovat jejich pohyb ve scéně, kterou si sám vytvoří.

Ovládání samotného programu není nikterak složité. Ovládání je víceméně intuitivní. Jsou zde popsány všechny nutné postupy pro provedení úspěšné simulace robota.

4.7 Programátorská dokumentace

Dále je na přiloženém cd-disku programátorská dokumentace, která obsahuje seznam všech souborů v aplikaci. Také obsahuje popis všech tříd v aplikaci. Celá jádro aplikace pro simulování robota vychází z knihovny libPE. Velká část kódu v aplikaci zajišťuje správné chování uživatelského grafické prostředí.

Kapitola 5 – Závěr a budoucí práce

V rámci předložené práce jsem vytvořil funkční knihovnu pro simulování tuhých těles a seznámil čtenáře s hlavními principy fyzikálního simulování tuhých těles ve 2D systému v počítači. V práci jsem kladl důraz na matematické odvození rovnic potřebných pro simulaci 2D systému, které jsem nemohl v pro mě dostupných materiálech nalézt. Zjistil jsem, že mnoho prací zabývající se touto problematikou je zaměřeno většinou na 3D systémy. Pro 2D systémy jsem žádnou ucelenou publikaci nenalezl.

V knihovně jsem naimplementoval algoritmus na detekci kolizí, algoritmus pro řešení simultánních odrazů se statickým třením a algoritmus pro řešení kloubů ve scéně. Funkčnost mé fyzikální knihovny jsem předvedl na aplikaci, pomocí které může uživatel simulovat pohyb robota ve 2D scéně.

V případě dalšího vývoje fyzikální knihovny libPE je možno se zaměřit na:

- Vyzkoušet Baraffův rozšířený algoritmus na řešení odrazů s třením.
- Rozšířit detektor kolizí na detektor, který umí kolize předpovídat
- Zkusit použití analytické metody pro řešení problematiky kloubů.
- Zvolit lepší integrační metodu než je metoda Eulerova.

V případě dalšího vývoje aplikace pro pohyb robota je možno se zaměřit na:

- Lepší zadávání vlastností motorů pro robotova tělesa
- Zajistit stálou časovou plynulost průběhu simulace

Literatura

[1] Andrew Witkin and David Baraff, „Differential Equation Basics” SIGGRAPH 2001

[2] Chris Hecker, Behind The Screen: part 2 www.d6.com/users/checker/pdfs/gdmphys2.pdf 1995-1997

[3] Chris Hecker, Behind The Screen: part 1 www.d6.com/users/checker/pdfs/gdmphys1.pdf 1995-1997

[4] Olivier Rebellion, internetové stránky, http://uk.geocities.com/olivier_rebellion/

[5] T. Gaing, G.Bradshaw, C.O.Sullivan, Complementarity Based Multiple Point Collision Resolution, Fourth Irish Workshop on Computer Graphics 2003

[6] David Baraff, Analytical methods for dynamic simulation of non-penetrating rigid bodies, Computer Graphics Proceedings, 1989, strany 223-232

[7] K. Kawachi, H. Suzuki, F. Kimura, Simulation of Rigid Body Motion with Impulsive Friction Force, Proceedings of IEEE International Symposium on Assembly and Task Planning, 1997, strany 182-187

[8] David Baraff, Fast Contact Force Computation For Non-Penetrating Rigid Bodies, Computer Graphics Proceedings, 1994, strany 23-34

[9] Alfred A. Schmidt, internetové stránky, http://i31www.ira.uka.de/projekte/mechanik/publications/details2003.php?publicationNumber=1, 2006