České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového Inženýrství

Bakalářská práce

Knihovna pro fyzikální engine založený na pružinovém modelu

Martin Labuť

Vedoucí práce: Ing. František Štampach

16. května 2013

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 16. května 2013 .....

Odkaz na tuto práci

Labuť, Martin. *Knihovna pro fyzikální engine založený na pružinovém modelu*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

This paper's purpose is design and implementation of a physics engine, that supports deformable and destructible bodies, while using a very simple model. Difficult problems, like calculating cracks in the body, or rotation around multiple axes, is approximated using a large number of simple calculations. This paper also contains various tests of this model's behavior.

Keywords physics simulation, spring, damper, point mass

Abstrakt

Tato práce se zabývá vytvořením fyzikálního enginu, který pracuje s deformovatelnými a zničitelnými tělesy, při použití velmi jednoduchého modelu. Tím se poměrně složitá problematika (rotace kolem více os, praskání tělesa) aproximuje velkým množstvím jednoduchých výpočtů. Součástí práce jsou i rozsáhlé testy chování modelu.

Klíčová slova fyzikální simulace, pružina, tlumič, hmotný bod

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Před nějakou dobou jsem pracoval na počítačové hře, kde proti sobě hráči bojovali v jimi navržených vesmírných lodích. Hráč měl možnost loď libovolně vytvarovat (pro zjednodušení se loď skládala ze čtvercových dílů) a osadit technikou, včetně motorů. Pohybové možnosti lodi následně vyplývaly z jejích fyzikálních vlastností (váha, poloha těžiště, moment setrvačnosti) a umístění motorů. To dávalo hře velkou variabilitu a hráčům možnost plně uplatnit svojí kreativitu.

Ve hře jsem používal vlastní fyzikální engine, který byl založen na fyzice tuhých těles. Engine byl celkem jednoduchý, ale bohužel použitím tuhých těles jsem přišel o některé možnosti, které by hra určitě využila, především to byla nedeformovatelnost lodí. Jediný způsob změny tvaru lodi byla ztráta některých dílů, která fungovala zcela nefyzikálně (vycházela z pravidel hry). Zároveň na lodi nemohly působit žádné vnitřní síly (reálně by se např. při použití příliš silných motorů mohla loď roztrhnout jejich působením nebo by odstředivé síly při příliš rychlé rotaci mohly odrthout část lodi).

Tím jsem se dostal k nápadu vytvoření enginu, který by fungoval na základě jednodušších interakcí. Kolize v předchozím enginu jsem řešil pomocí pružin, a tak jsem tuto technologii použil i dále. Jednotlivé díly lodi byly vzájemně pospojované pružinami, všechny síly působily v daném dílu a dále se přenášely pomocí interakce pružin. Při jednoduchých testech tohoto systému se zdálo, že engine celkem funguje, tak jsem se rozhodl vytvořit engine, který je založen na tomto modelu, a otestovat, jak se v něm budou složitější struktury chovat.

Kapitola 1

Popis řešeného problému

1.1 Obsah

Cílem této práce je vytvořit fyzikální engine, který bude splňovat následující vlastnosti:

- Všechny objekty v simulovaném světě budou definovány pomocí soustavy hmotných bodů propojených pružinami.
- Hmotné body ovlivňují ostatní hmotné body jen prostřednictvím pružin, jejichž působení je definováno jednoduchým vzorcem (viz sekce 2.1.1).
- Pohyb hmotných bodů podléhá zákonům Newtonovské dynamiky.

1.2 Forma

Engine bude implementován jako knihovna v C++. Snahou bude dosáhnout co nejnižší závislosti na externích knihovnách, ideálně pouze na základních, platformě nezávislých, knihovnách C++. Ke knihovně budou dodány všechny zdrojové kódy, ale knihovna samotná bude předkompilována pro okamžité použití.

Kapitola 2

Analýza a návrh

2.1 Existující metody

S myšlenou deformovatelných těles už se pracuje, tento obor simulace se nazývá „Soft body physics“. Už nějakou dobu se tyto modely používají ve filmech a high-end hrách [6], ale začínají se objevovat i ve volně dostupných enginech, jako je např. Bullet [1]. Způsobů implementace je hned několik.

2.1.1 Mass-Spring Systems

Popsaná technologie se velmi podobá existující technologii nazývané „Mass-Spring Systems“ (zmíněná např. v [5]). Model je zde tvořen pravidelnou sítí bodů, mezi kterými jsou umístěné spoje. Spoj se chová jako dvě primitivní fyzikální struktury: pružina a tlumič [7]. Síla, kterou působí pružina, odpovídá funkci

$$\overrightarrow{F_s} = k_{01}\overrightarrow{d_{01}}(|\overrightarrow{x_1} - \overrightarrow{x_0}| - l_{01}),$$

kde $\overrightarrow{x_0}$ a $\overrightarrow{x_1}$ jsou polohové vektory dvou bodů, k_{01} je tuhost pružiny a l_{01} její klidová délka. $\overrightarrow{d_{01}}$ je normalizovaný vektor směru spoje, viz (2.1).

Síla působená tlumičem odpovídá

$$\overrightarrow{F_d} = s_{01}\overrightarrow{d_{01}}(\overrightarrow{v_1} - \overrightarrow{v_0}, \overrightarrow{d_{01}})$$

kde $\overrightarrow{v_0}$ a $\overrightarrow{v_1}$ jsou vektory rychlostí bodů a s_{01} je koeficient útlumu.

$\overrightarrow{d_{01}}$ je normalizovaný vektor rozdílu polohových vektorů, představující směr, ve kterém spoj leží. Jelikož vzorce obou sil jsou zjednodušeně $\overrightarrow{d_{01}}$

vynásobený skalárem, tak obě síly mají nutně stejný směr jako spoj. Vzorec je tedy

$$\overrightarrow{d_{01}} = \frac{\overrightarrow{x_1} - \overrightarrow{x_0}}{|\overrightarrow{x_1} - \overrightarrow{x_0}|} \tag{2.1}$$

Parametry s_{01} a k_{01} se mohou lišit pro každý jednotlivý spoj a svými hodnotami výrazně ovlivňují chování struktury.

2.1.2 Metoda konečných prvků

Metoda konečných prvků je postup pro řešení diferenciálních rovnic, ale stejný název se používá i pro označení způsobu výpočtu deformací, ve kterém se právě touto metodou řeší diferenciální rovnice. Ty se sestaví podle modelu objektu.

2.2 Existující řešení

Na trhu je samozřejmě již dostupné velké množství různých fyzikálních enginů. Ne všechny jsou zdarma, nicméně lze najít i zdarma dostupné enginy vysoké kvality. Pokusím se zde stručně popsat pár enginů, které jsem před zahájením vývoje zkoumal.

2.2.1 Havok

Havok je obsáhlá sada nástrojů pro vývoj her a fyzikální simulace. Obsahuje nástroje pro animace, simulace destrukce a, co se nejvíce týká této práce, nástroj pro simulaci látek a vlasů, který také používá některé z technologií soft body fyziky. Havok je v současnosti uvolněn zdarma pro menší projekty.

2.2.2 DMM

Digital Molecular Matter je engine zaměřený na simulaci destrukce. Je používán především ve filmech (například v Avatarovi [6]) a vzácně ve hrách (např. Force Unleashed[4]). Nicméně v případě her se používá pouze pro vizuální dojem ze hry, ale nikdy neovlivňuje herní svět.

2.2.3 Bullet

Bullet je herní engine, který nabízí rozsáhlé možnosti pro simulace. Umožňuje simulaci jak tuhých těles, tak deformovatelných těles (soft body) a

jejich vzájemnou interakci. Také nabízí detekci kolizí pro deformovatelná tělesa. Už ale např. neumožňuje deformovatelná tělesa za běhu dělit [1].

2.3 Použité řešení

Z existujících technologií navrženému řešení nejlépe odpovídá Mass-Spring System. Použiji tedy tuto metodu, včetně vzorců uvedených v 2.1.1. K řešení diferenciálních rovnic se použije Eulerova metoda, která sice není úplně přesná, ale na druhou stranu není výpočetně náročná. Simulovaný svět se upravuje po malých časových krocích, při kterých se aplikují síly a rychlost bodu,

$$\overrightarrow{v_0}(t+s) = \overrightarrow{v_0}(t) + \overrightarrow{F_0}(t)s/m_0, \tag{2.2}$$

$$\overrightarrow{x_0}(t+s) = \overrightarrow{x_0}(t) + \overrightarrow{v_0}(t)s, \tag{2.3}$$

kde $\overrightarrow{x_0}$ je polohový vektor bodu, $\overrightarrow{v_0}$ je rychlostní vektor bodu, $\overrightarrow{F_0}$ je celková síla působící na bod, m_0 je hmotnost bodu a s je délka časového kroku.

Přesnost výpočtu silně závisí na volbě časového kroku, resp. na poměru velikosti sil, působících v systému, a časového kroku (tzn. čím větší síly působí v systému, tím menší časový krok je třeba zvolit)

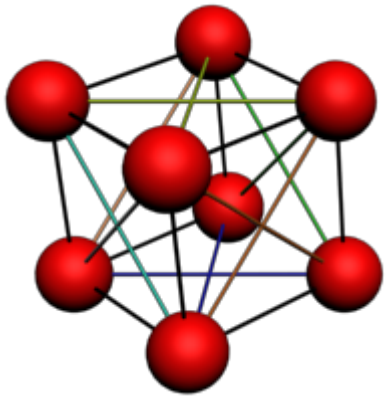
Na rozdíl od existujicích řešení je třeba tento engine nastavit tak, aby se výsledky fyzikálního působení projevovaly do herního světa, nikoli pouze do grafického výstupu. K tomu bude potřeba mít nějakou metodu, jak získávat údaje o stavu simulovaného světa zpět do nadřazené části programu. V praxi to pak bude znamenat, že fyzikální jevy, které engine vypočítá, budou mít přímý vliv na hráčův dojem ze hry.

V tomto simulovaném světě se budou pohybovat shluky pospojovaných bodů, nazývané struktury. Ty mohou mít zcela libovolný tvar, nicméně pro simulaci pevných těles se budou nejčastěji využívat pravidelné mřížkové struktury (viz. obr. 2.3), které zajistí uchování tvaru.

2.4 Požadavky na systém

Z daných závěrů vyplývá několik jasných požadavků, které musí systém splňovat.

Obrázek 2.1: Nejjednodušší díl struktury



2.4.1 Modularita

Jedním z největších problémů u existujících enginů je nemožnost doimplementování vlastních součástí. Pokud chce programátor použít jakoukoli nestandartní technologii (například voxelový terén), výběr použitelných enginů se prudce sníží. Proto bude nejdůležitější vlastností tohoto enginu modularita. Veškeré funkce, kromě úplně základní kostry, budou napsány jako moduly, které bude možno podle libosti přidávat nebo vyměňovat za vlastní verze. V základní verzi budou k dispozici veškeré moduly potřebné pro základní nasazení enginu. Zároveň tyto moduly budou sloužit uživateli jako návod pro tvorbu jeho vlastních modulů.

2.4.2 Správa struktur

Simulovaný svět bude sestávat z množství struktur, které představují tělesa v daném světě. Každá struktura sestává z bodů a pružin, ale je možné, že některé moduly budou požadovat další atributy, které musí struktura nést. V základním nastavení ponese struktura pouze údaje o hmotných bodech. Obvykle bude použit modul, který přidá i údaje o pružinách. Nicméně mohou nastat případy, kdy se pružiny dají dopočítat (u pravidelných struktur). V takovém případě uživatel přidá jiný modul než jen prostý seznam pružin.

2.4.3 Správa bodů

Podobně jako struktury také body musí nést všechny údaje, které budou vybrané modely vyžadovat. Přestože jakékoli použití enginu bude vyžado-

vat, aby body nesly údaje o svých fyzikálních vlastnostech (pozice, rychlost, zrychlení, hmotnost), i tyto údaje budou přidány modulem. V základním nastavení tedy bude bod prázdný, dokud moduly nepřidají nějaké sledované údaje.

2.4.4 Dynamika

Nejjednodušší funkcí systému bude simulace zákonů dynamiky. To znamená vypočítat pozici bodu v závislosti na rychlosti, původní pozici a časovém kroku, a rychlost v závislosti na silách, původní rychlosti a časovém kroku. To se dá dělat několika způsoby, obvykle je vyšší přesnost dosažena za cenu vyšší výpočetní náročnosti. Tato funkce tedy bude pokryta modulem, popřípadě více moduly, které budou různými způsoby počítat pohyb.

2.4.5 Pružinová interakce

Druhou fyzikální funkcí systému je výpočet působení sil, způsobených pružinovou interakcí. I zde je více možností, od jednoduchého krokování po diferenciální rovnice. Proto i tato funkce bude pokryta moduly, které budou síly různými způsoby počítat. Knihovna musí obsahovat minimálně základní modul, který dokáže aplikovat síly jednoduchým způsobem.

2.4.6 Vnější síly

Jelikož engine má být uzpůsoben pro vývoj počítačových her, musí existovat možnost, jak bude hráč ovlivňovat herní svět. Tedy musí být možnost, jak v simulovaném světě působit silou, která pochází z vnějšku.

2.4.7 Detekce kolizí

Jelikož má být engine využitelný ve hrách, je potřeba, aby nabízel detekci kolizí. Vnitřní interakce ve struktuře budou modelovat předdefinované pružiny, ale jakákoli interakce mezi strukturami musí být dynamická. Jelikož kolize budou zahrnovat obrovské množství bodů, je třeba, aby byly optimalizovány. Jelikož existuje několik metod, jak detekovat kolize na této struktuře, bude detekce kolizí modulární, a to nejlépe vícenásobně, aby se dala požadovaná metoda sestavit na přání (oddělené by určitě měly být moduly pro detekci a pro řešení kolizí, neboť i to lze provádět více způsoby).

2.4.8 Ničení a tvoření

Je možné, že struktury se budou ve světě utvářet postupně až v průběhu simulace. Je tudíž potřeba umožnit strukturám se za běhu slučovat. Zároveň je ale možné, že se struktury budou naopak rozpadat, ať už následkem srážky, nebo uměle vnějším zásahem. Pak je nutné, aby existoval mechanizmus, který bude moci zrušit libovolné pružiny a i celé body. Po zrušení bodů a pružin musí být možné automaticky zkontrolovat celistvost struktury.

2.4.9 Determinismus

Pokud má být engine použit ve hrách, je nutné, aby byl synchronizovatelný. Ani v běžných hrách není praktické, aby se synchronizovala poloha všech herních objektů. Už vůbec není možné, aby se synchronizovala poloha všech hmotných bodů, kterých mohou být tisíce. Aby se množství synchronizovaných dat omezilo, je potřeba, aby engine byl plně deterministický. Poté lze synchronizovat pouze akce uživatelů a jakýkoli další vývoj situace se dopočítá u všech účastníků hry stejně.

2.4.10 Modelování struktur

Struktury obsahují množství bodů a pružin a bylo by silně nepraktické, kdyby bylo nutno všechna tyto data ručně plnit. V knihovně musí být k dispozici nástroje pro tvoření pravidelných struktur (krychlové mřížky jako na obr. 2.3), pro otáčení a spojování takových struktur.

2.5 Funkce

Z navržených požadavků vyplývají následující funkce, které musí systém poskytovat.

2.5.1 Vytvoření simulace

Vytvoření simulace je prvním krokem k použití enginu. V průběhu vytváření uživatel určí velikost časového kroku a všechny moduly, které chce použít. Jakmile jsou určeny moduly, lze zjistit, která všechna data se musí u bodů a struktur ukládat, a tím se otevře možnost vkládat do simulace struktury.

2.5.2 Správa dat

Systém se musí postarat o to, aby byla všechna data někde uložena. Jakmile jsou určeny všechny moduly, je jasné, kolik místa v paměti zabere každý uzel a každá struktura. Při vytváření nových struktur se tedy zohlední tyto informace a prostor v paměti se rovnou připraví tak, aby se tam žádaná data vešla.

2.5.3 Vytvoření struktury

Při vytvoření struktury je zapotřebí zkontrolvat všechny moduly a inicializovat jejich vlastní data. Totéž se pak děje při mazání struktury, všechny moduly po sobě musí "uklidit".

Po obsahové stránce lze strukturu vytvořit několika způsoby:

- Pomocí sady nástrojů se vygenerují základní tvary.
- Spojí se již existující struktury, případně body uvnitř jedné struktury.

2.5.4 Manipulace se strukturou

Po vytvoření obsahu struktury je možné, že struktura není na správném místě ve světe. To lze vyřešit za pomoci další sady nástrojů, které budou umožňovat:

- zvolené body, nebo celé struktury, otáčet a posunovat.
- vyrovnat délky pružin (Při úpravě tvaru struktury bude možné strukturu zafixovat v tomto novém tvaru).

2.5.5 Běh simulace

Uživatel musí mít možnost řídit běh simulace. Jelikož délka časového kroku je pevně dána pro celou simulaci, může v podstatě jen určit, kdy se provede vyhodnocení celého simulačního kroku. Vyhodnocování by nemělo být přerušeno ani data po dobu vyhodnocování měněna.

2.5.6 Přístup k výsledkům

Uživatel bude potřebovat nějak se dostat k výsledkům simulace. Pokud potřebuje všechna data, například pro obsáhlejší vykreslování, budou k dis-

pozici iterátory nad celými datovými sadami. Pro přístup ke konkrétním bodům může použít pohledy (viz níže).

2.5.7 Pohledy

Jelikož body nemají v základním nastavení žádné identifikační atributy, nelze úplně dobře najít jeden konkrétní bod (například ve hře může být na některé body navázána speciální herní funkce). Aby to bylo možné, bude mít uživatel k dispozici pohledy, které představují odkazy na konkrétní body. Veškeré manipulace se strukturami, které nějak výrazně zasahují do paměťového úložiště bodů, budou počítat s existencí pohledů, a případně jejich vnitřní strukturu upraví tak, aby stále odkazovaly na stejné body.

2.5.8 Detekce kolizí

Detekce kolizí bude probíhat automaticky v každém kroku simulace. Uživatel nebude muset nijak do detekce zasahovat, kromě jejího zařazení do aktivních modulů, nicméně bude mít možnost si zaregistrovat posluchač pro událost kolize. Bude také mít možnost pomocí dalšího modulu využít automatické destrukce.

2.6 Algoritmy

V navrhovaných funkcích systému je hned několik míst, kde je definice funkce velmi volná a není přesně jasné, jak se bude systém v daném místě chovat. Všechny tyto postupy a algoritmy se pokusím rozebrat v této sekci.

2.6.1 Správa modulů

Pro každou simulaci bude existovat globální úložiště modulů, kam je potřeba všechny požadované moduly před zahájím simulace uložit a odtud bude možnost získat odkaz na vyžadovaný modul. Funkce na vyhledání modulu ve správci bude rozlišená podle tříd, takže bude možné vyhledávat i podle abstraktních tříd. Díky tomu mohou moduly být závislé pouze na rozhraních, a ne na konkrétní implementaci, čímž lze dosáhnout vyšší modularity celého systému.

2.6.2 Ukládání dat

Jak již bylo několikrát zmíněno, obsah dat pro body a struktury není předem znám, ale vytváří se až podle vybraných modulů. V praxi to bude

vypadat tak, že se při vložení modulů do úložiště zjistí jejich paměťové požadavky a podle toho se rozvrhne místo v paměti. Při alokaci paměti pro body a struktury se pak použije souhrnná velikost, a poté budou jednotlivé moduly informovány, kde se v paměti každého bodu nachází jejich atributy. Při přesunu bodu (např. při rozpadu struktur) lze jednoduše přenést celý úsek paměti bodu bez nutnosti znát jeho vnitřní strukturu.

Bod sám o sobě neponese žádné informace, má tedy nulovou velikost. Základní parametry, jako např. poloha, se přidají jedním z modulů. Přestože vytvořit simulaci bez tohoto modulu nebude mít valný smysl, jsou tyto parametry umístěny v seperátním modulu, aby se chovaly stejně jako jakákoli jiná data bodů.

Stejný postup se použije i u stuktur s tím rozdílem, že struktura má některé atributy přirozeně, bez nutnosti přidávat moduly (např. odkaz na svoje úložiště bodů). Na rozdíl od bodů tedy u struktur existují dvě části úložiště, přirozené a modulové. Jelikož se ale k přirozenému úložišti příliš často nepřistupuje a i s úložištěm bodů se pracuje pomocí příslušných modulů, tak tato podvojnost nebude příliš rušit.

2.6.3 Generované struktury

Nástroj pro generování struktury bude umět jen základní mřížku, což bude velmi jednoduchá pravidelná struktura bodů, pospojovaná dvěma možnými způsoby. Tím vznikne soustava krychlí. V každém případě budou spojeny body bezprostředně sousední, tedy pružiny povedou po hranách krychle. Ovšem k udržení stability struktury jsou potřeba ještě tzv. strukturální pružiny (popis nejjednodušší struktury viz [5]). Ty mohou být buď úhlopříčky stran krychle (6 stran po 2 úhlopříčkách), nebo úhlopříčky krychle samotné (4 úhlopříčky celkem).

2.6.4 Detekce kolizí

Pro jednoduchost jsem zvolil metodu detekovat kolize mezi body, pružiny při tom zcela ignorovat. Tento způsob detekce kolizí má určité výhody, například jednoduchost a rychlost, oproti např. detekci kolizí na trojúhelnících (což mj. umoňuje engine Bullet [1]), na druhou stranu omezuje možné tvary struktur (je potřeba celý objem tělesa pokrýt body s dostatečnou tloušťkou). I přes zdánlivou jednoduchost prověření kolizí všech bodů se všemi by mělo složitost $O(n^2)$, což je příliš. Naštěstí lze detekci optimalizovat.

Zvolenou metodou optimalizaceí je použití oktalového stromu, do kterého lze body vkládat uspořádaně podle pozice a velikosti, čímž se složitost sníží na ideálně $O(nlog_8 n)$. Reálná složitost silně závisí na rozložení bodů ve světě, nicméně pokud se svět bude skládat z pravidelných struktur, což pravděpodobně bude (kromě momentů, kdy zrovna dochází k deformacím), tak se reálná složitost může té ideální celkem přiblížit.

V případě, že ke kolizi dojde, existuje několik možností, jak ji vyřešit. Často používanou metodou je vrácení bodů do takové polohy, kdy ke kolizi nedojde. To je však zde celkem nevhodné kvůli vysoké hustotě bodů (kdy vyřešením jedné kolize lze snadno způsobit další). Navíc engine je tvořen pro deformovatelná tělesa, a byla by škoda tato tělesa vytvořit ze shluku nedeformovatelných bodů. Proto byly vybrány pružné kolize - v případě srážky se bude silové působení řešit stejným vzorcem, jako interakce pružin, ovšem tyto pomyslné pružiny budou mít o něco vyšší tuhost, než pružiny strukturální.

Kapitola 3

Realizace

3.1 Balíčky a třídy

Po zvážení všech požadavků a funkcí jsem vytvořil následující strukturu balíčků a tříd. Balíčky jsou navržené tak, aby se závislosti vrstvily, a ne cyklily. Při nasazení knihovny bude mít program, který ji využívá, stejné závislosti, jako má balíček testů (což jsou v podstatě programy, využívající knihovnu).

3.1.1 Jádro (core)

Jádro obsahuje nezbytné třídy pro běh simulace, na kterých závisí naprostá většina ostatních tříd.

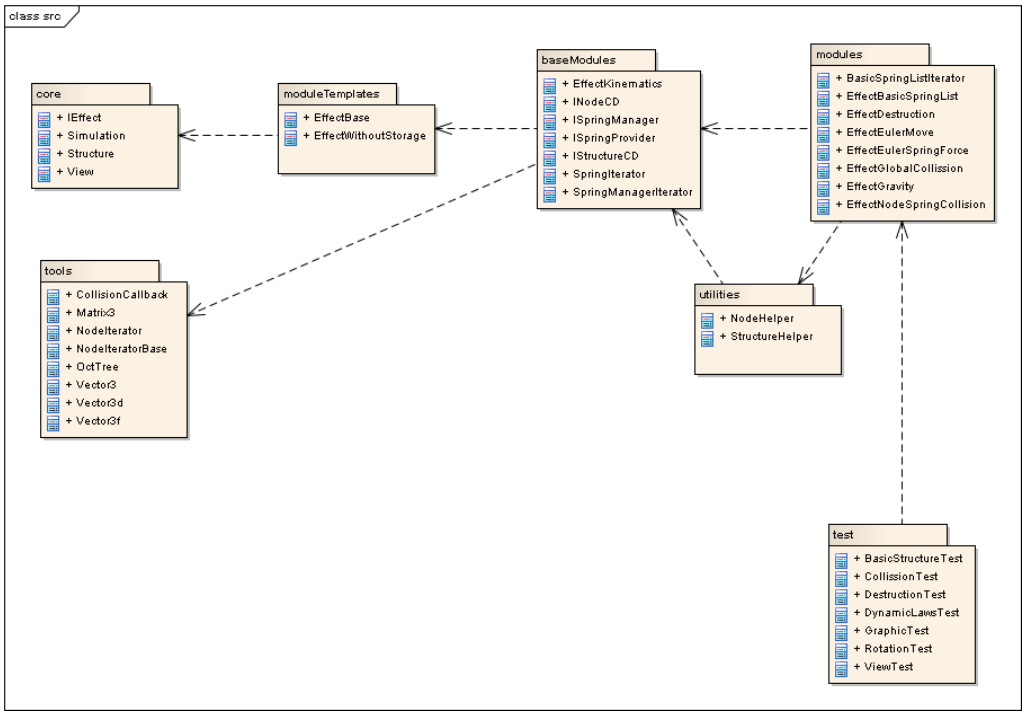
3.1.1.1 Simulation

Třída Simulation je srdcem celého enginu. Jakékoli použití enginu ze strany uživatele začne vytvořením instance Simulace. Ta spravuje seznam efektů (IEffect 3.1.1.3), které tvoří danou simulaci, a později také seznam struktur, které se v simulovaném světě nacházejí. Simulace dále obstarává vytváření a mazání struktur a organizuje výpočet kroku simulace.

3.1.1.2 Structure

Structure představuje jednu strukturu v herním světě. Je vytvářena a mazána svojí nadřazenou třídou simulace, která se zároveň stará o uložení všech atributů, které jsou vyžadovány efekty.

Obrázek 3.1: Diagram závislosti balíčků



3.1.1.3 IEffect

Základní rozhraní pro efekty, které představují jednotlivé volitelné moduly. IEffect obsahuje jak správu dat pro body a struktury, tak několik metod, které jsou volány při počítání kroku simulace.

3.1.1.4 View

Pohled (View) je pouze jednoduchým odkazem na vybrané uzly ve struktuře. Seznam pohledů na své uzly si udržuje každá struktura a při jakémkoli zásahu do struktury uzlů je potřeba ověřit a popřípadě opravit tyto pohledy.

3.1.2 Základní moduly (baseModules)

Balíček základních modulů poskytuje ty nejjednodušší moduly, které většinou jen poskytují úložiště pro data, nebo případně nabízejí jednu konkrétní službu. Většina modulů je zde jen jako abstraktní třída, konkrétní implementace jsou až v balíčku modules.

3.1.2.1 ISpringProvider

ISpringProvider je rozhraním pro efekty, které nějakým způsobem ukládají nebo vytváří pružiny ve struktuře. K pružinám se bude přistupovat pomocí speciálního iterátoru, který si každá implementace ISpringProvider vytvoří.

3.1.2.2 ISpringManager

ISpringManager je rozšířením pro ISpringProvider, které navíc obsahuje metody pro modifikaci pružin. To je nezbytné, pokud se mají provádět jakékoli automatické úpravy struktury za běhu (např. rozpad struktury vlivem kolize).

3.1.2.3 EffectKinematics

EffectKinematics je základním efektem, který ukládá kinematické parametry bodu, což jsou rychlost, pozice, působící síly, a hmotnost. Naprostá většina efektů ho používá a pravděpodobně nemá valný smysl vytvářet pro něj alternativu. Nicméně pro konzistenci systému byly i tyto základní atributy vloženy pomocí efektu. Působící síly se ukládají, jelikož některé jevy (např. útlum u pružin) jsou závislé na rychlosti, a nelze proto rychlost měnit v průběhu vyhodnocování ostatních modulů.

3.1.2.4 INodeCD

INodeCD je rozhraním pro efekt, který dokáže vyřešit srážku dvou bodů. Toto rozhraní je používáno detektory kolizí, jsou mu předávány všechny detekované kolize.

3.1.3 Nástroje (tools)

Balíček tools obsahuje technické pomůcky, které jsou používány jinými balíčky. Jsou to velmi obecné třídy, které by mohl uživatel uplatnit i v místě, která nesouvisí s tímto enginem.

3.1.3.1 msd::Vector3

Vector3 je jednoduchý trojrozměrný vektor, na kterém je implementováno velké množství základních aritmetických operací. Je uložen v jmenným prostoru msd, protože jeho název je velmi obecný a mohl by se překrývat s jinou uživatelovou implementací.

3.1.3.2 msd::Matrix3

Matrix3 je omezená implementace matice 3x3. Obsahuje jen ty nejnutnější metody, které engine využívá. Stejně jako Vector3, i Matrix3 je uložena ve jmenném prostoru.

3.1.3.3 NodeIterator

NodeIterator je speciální iterátor, který iteruje přes předem neznámé datové úložiště bodů. Instancuje se pro strukturu a efekt a vždy ukazuje na začátek atributů patřících danému efektu. Pomocí šablony ho lze vytvořit pro libovolný typ úložiště Většina efektů bude mít definovanou strukturu (ve smyslu datové struktury, neboli struct), která představuje jejich datové úložiště a lze jí se použít jako šablonu.

3.1.3.4 OctTree

OctTree je jednoduchá implementace oktalového stromu, který se používá pro detekci kolizí. Jelikož není nijak závislý na žádné části enginu, lze ho samozřejmě použít i kdekoli jinde.

3.1.4 Vzory (moduleTemplates)

Tento balíček obsahuje dvě pomocné abstraktní třídy, které částečně implementují rozhraní IEffect.

3.1.4.1 EffectBase

EffectBase je abstraktní implementace IEffect pro efekty, které potřebují ukládat data pro body nebo struktury. Zbylé metody IEffectu, které se netýkají ukládání dat, neimplementuje.

3.1.4.2 EffectWithoutStorage

EffectWithoutStorage je abstraktní implementace IEffect pro efekty, které nepotřebují ukládat data. Stejně jako EffectBase přepisuje metody pro ukládání a správu dat v podstatě prázdnými implementacemi, zbylé metody neimplementuje.

3.1.5 Pomůcky (utilities)

Tento balíček obsahuje pomocné třídy, které dokáží provádět obvyklé operace na strukturách a lze je využít ke konstrukci struktur.

3.1.6 NodeHelper

NodeHelper je jednodušší třída, obsahuje metody, které nepracují s pružinami. Veškeré operace, jako přesuny, rotace, aplikace sil, se dají jednoduše provádět přes tuto třídu.

3.1.7 StructureHelper

StructureHelper obsahuje nástroje, které vyžadují práci s pružinami. Některé jeho metody jsou využívány i v pokročilých modulech. Dokáže tvořit struktury, spojovat je, detekovat jejich rozpad a rozložit strukturu na více struktur.

3.1.8 Moduly(modules)

Balíček modulů obsahuje všechny efekty (moduly), které jsou v základní implementaci dostupné. Některé efekty nemají alternativu (ale lze ji doprogramovat), jiné jsou již v základu výlučné, a tak je možné vybrat jen jeden z nich.

3.1.8.1 EffectBasicSpringList

EffectBasicSpringList je nejjednodušší metodou uložení pružin do struktury. Vytvoří v paměti jednoduchý seznam, který lze postupně naplnit parametry pružin.

3.1.8.2 EffectEulerMove

EffectEulerMove je efektem pro nejjednodušší výpočet pohybu bodů. Používá k řešení pohybu Eulerovu metodu (jako v [2]), která může být ale značně nepřesná, zvláště v extrémech (pokud se bude působící síla prudce měnit v závislosti na poloze).

3.1.8.3 EffectEulerSpringForce

Podobně jako předchozí efekt, EffectEulerSpringForce používá jednoduchou Eulerovu metodu pro výpočet sil působících na body. Počítá s tím, že na body bude po celou dobu diskrétního časového kroku působit stejná síla. Stejně jako u pohybu, v extrémních případech může vzniknout značná nepřesnost. Nicméně to je cenou za extrémní jednoduchost, a tudíž vysokou výpočetní rychlost, této metody.

3.1.8.4 EffectGlobalCollission

EffectGlobalCollission poskytuje základní detekci kolizí za pomoci oktalového stromu. Používá pouze tuto optimalizaci, tzn. postupně vloží všechny body do stromu, a přitom detekuje kolize. K vyřešení kolizí vyžaduje potomka INodeCD, kterému zasílá všechny kolize.

3.1.8.5 EffectNodeSpringCollision

EffectNodeSpringCollision je nejjednodušší implementací INodeCD, tedy efektu k řešení kolizí mezi body. Jak napovídá název, kolize řeší stejným vzorcem, jako se řeší interakce pružin.

3.1.8.6 EffectGravity

EffectGravity je jednoduchým efektem pro vytvoření gravitace ve světě. Umožňuje gravitační působení libovolným směrem, a také ukotvení struktur - některé struktury mohou být z působení gravitace vyňaté.

3.1.8.7 EffectDestruction

EffectDestruction je poměrně jedoduchým efekte, obstarávájícím destrukci. K detekci destrukce je použit triviální algoritmus: když se pružina natáhne nad určitou délku (resp. protáhne na určitý násobek své klidové délky), přetrhne se. Tato prahová hodnota je samozřejmě konfigurovatelná.

3.1.9 Testy (test)

Balíček testů obsahuje abstraktní třídu pro tvorbu testů a všechny testy, které jsou uvedené v sekci 3.2. Podrobný popis všech testů je v příslušné sekci. Balíček testů se kompiluje zvlášť, protože vytváří závislosti na grafických a jiných vstupně-výstupních knihovnách, které jsou nežádoucí pro samotnou knihovnu.

3.1.9.1 GraphicTest

GraphicTest je abstraktní třídou pro tvorbu testů. Obstarává vytvoření okna, inicializaci opengl, základní ovládání kamery, ukončení i kroky simulace. Potomci musí jen vytvořit a naplnit simulaci, případně vykreslovat obsah simulace (k čemuž GraphicTest dodává množství pomocných funkcí).

3.2 Testy

Jelikož celý fyzikální model je pokusný, a jeho vlastnosti nejsou předem známy, tak bylo třebachování důkladně otestovat. Provedenými testy jsem se snažil prověřit jak základní chování systému, tak funkčnost všech implementovaných funkcí. Testy může zároveň uživatel využít jako návod k použití enginu.

Následující testy zkoumají především model a jeho chování, čímž ověřují vyšší funkce systému. Před započetím těchto pokusů byly provedeny elementární testy všech tříd, které prověřili stabilitu a funkčnost metod. Zde uvedené pokusy na ně navazují a testují komplexní chování systému.

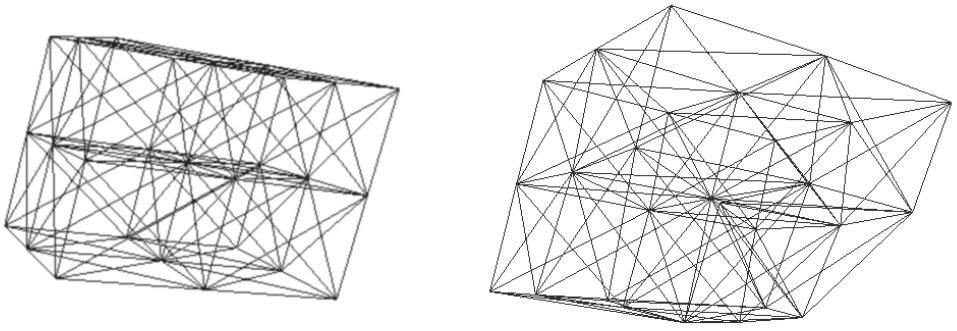
3.2.1 Test základního chování

Prvním z testů je test chování systému. Zajímá nás především, zda navržený model vytvoří stabilní struktury (tzn. existuje klidový stav, do kterého se lze dostat) a jak rychle se stabilizuje.

3.2.1.1 Svět

V testovaném světě bude existovat pouze jediná struktura, krychle tvořená $3 \times 3 \times 3$ body, navzájem pospojovanými. Na tuto krychli budou vyvíjeny postupně se zvyšující síly (v daných časových okamžicích bude síla aplikována na jeden z bodů).

Obrázek 3.2: Krychle před a po deformaci



3.2.1.2 Požadovaný výsledek

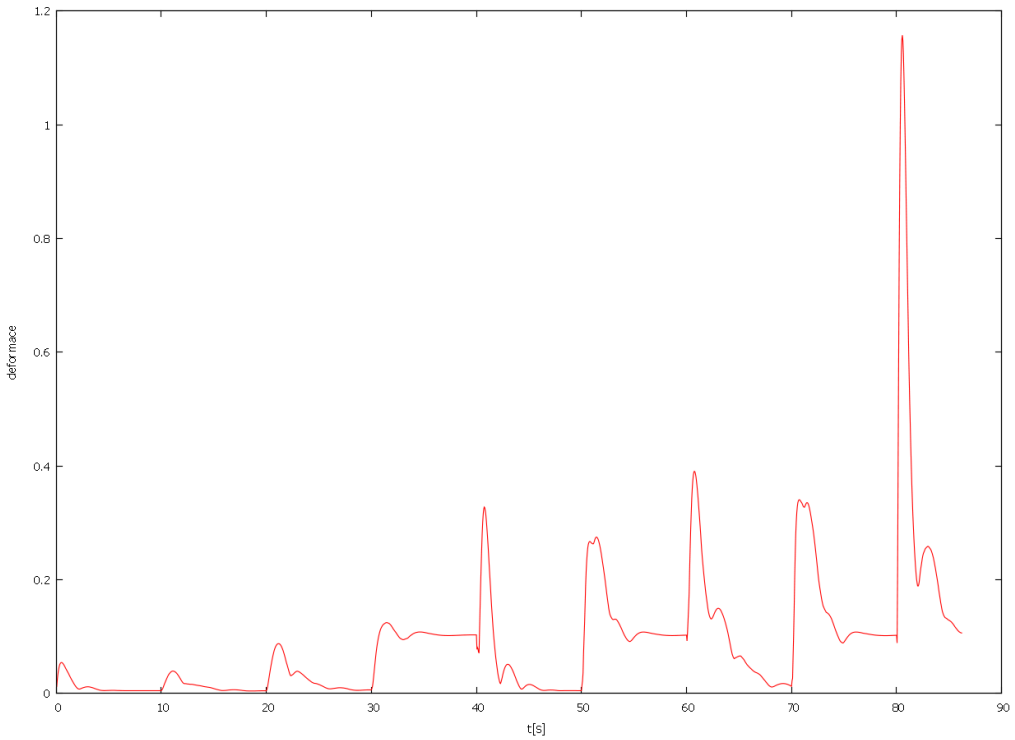
V testu budeme měřit míru deformace, kterou jsem definoval takto:

$$d = \frac{1}{|U|} \sum_{s \in U} \frac{||\overrightarrow{x_{s0}} - \overrightarrow{x_{s1}}| - l_s|}{l_s},$$

s je index pružiny, U je možina všech pružin, $\overrightarrow{x_{s0}}$ a $\overrightarrow{x_{s1}}$ jsou polohy obou bodů, které pružina s spojuje a l_s je klidová délka pružiny. Zjednodušeně by se dalo říct, že jde o průměrnou relativní výchylku délky pružiny.

V momentě, kdy síla zapůsobí, se míra deformace zvýší. V ideálním případě by se měla opět časem snížit, jak budou pružiny působit silou směrem ke své klidové poloze.

Obrázek 3.3: Míra deformace v čase



3.2.1.3 Naměřené hodnoty

Jak je vidět z grafu 3.3, míra deformace se téměř vždy vrací k nule (skutečná hodnota se v tu chvíli pohybuje okolo 0.005). V některých případech

se ale ustálí na hodnotě mnohem vyšší. Co se v takovém případě stalo, ukazuje obrázek 3.2. Pravý dolní roh se v tomto případě zaseknul za prostředním bodem z opačné strany, čímž natrvalo zdeformoval krychli (resp. do dalšího silového působení, kdy se zase bod uvolnil).

Tady se ukazuje první velká slabina systému. Rovnice silového působení nezaručuje, že body neprojdou skrz sebe navzájem. Pokud má model představovat hmotu, tak by rozhodně nemělo být možné, aby k tomu došlo.

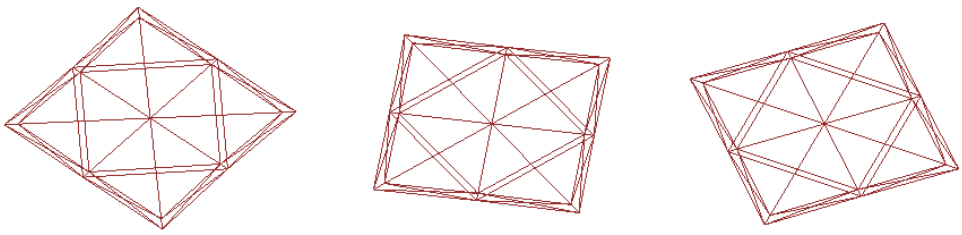
3.2.2 Test zákonů dynamiky

Tento test prověřuje platnost základních zákonů dynamiky v systému. Chceme ověřit, zda silové působení mimo těžiště dá vzniknout rotačnímu pohybu, kolem jaké osy se bude těleso otáčet a zda budou platit základní zákony zachování.

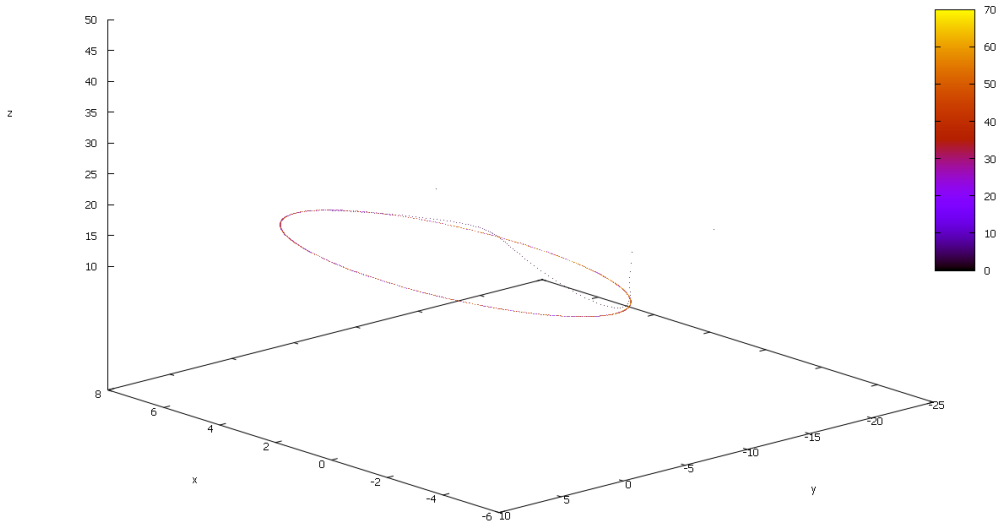
3.2.2.1 Svět

V testu se využije svět s jednou vygenerovanou strukturou, ve tvaru kvádru. Stěna, kterou prochází předpokládná osa otáčení je tvořena mřížkou 3×3 bodů, a struktura má 2 vrsty (tzn. 18 bodů celkem). Body jsou pospojované dvěma typy pružin: základní pružiny spojují sousední body v řádcích, ve sloupcích a ve vrstvách, strukturální pružiny spojují body v úhlopříčkách čtverců. Většina bodů má nulovou počáteční rychlost, kromě dvou protilehlých hran, které mají počáteční rychlost vzájemně opačnou (tzn. počáteční vektorový součet všech hybností je nulový).

Obrázek 3.4: Ukázka rotace struktury



Obrázek 3.5: Hodnoty momentu hybnosti



3.2.2.2 Požadovaný výsledek

Struktura bodů by se měla začít otáčet okolo osy, procházející středem širší strany. Zároveň by měly platit zákony zachování energie, hybnosti a momentu hybnosti, takže po chvíli by se struktura měla ustálit v pravidelném pohybu. Energie a hybost bodu jsou definovány jako

$$E_k = \frac{1}{2}mv^2,$$

$$\overrightarrow{p} = m\overrightarrow{v},$$

kde m je hmotnost a $\overrightarrow{v}$ vektor rychlosti.

Moment hybnosti pro hmotný bod s polohovým vektorem $\overrightarrow{r}$, rychlostí $\overrightarrow{v}$ a hmotností m je definován jako

$$\overrightarrow{L} = \overrightarrow{r} \times m\overrightarrow{v}.$$

Moment hybnosti se zachovává vůči pevně zvolenému bodu, což je v tomto případě počátek soustavy souřadnic.

Všechny tři veličiny sledujeme v průběhu pokusu, a pokud se nebudou měnit, tedy budou platit zákony jejich zachování, tak model dosáhne požadované podobnosti s modelem tuhých těles. Jelikož ale výpočty probíhají s

Tabulka 3.1: Naměřené hodnoty

čas	$\overrightarrow{p}$	E_k	$\overrightarrow{L}$
0.05	(0, 0, 0)	200	(0, -25, 50)
0.1	(0, 0, 0)	200	(0, -25, 50)
0.15	(1.90e-006, -4.76e-007, 0)	114.76	(6.20, -12.43, 13.69)
0.2	(1.90e-006, 4.76e-007, 0)	69.62	(-5.52, -8.39, 25.81)
0.25	(3.8e-006, 1.43e-006, 0)	38.40	(-3.16, -7.11, 19.38)
0.3	(3.8e-006, 1.43e-006, -2.98e-008)	36.62	(-3.45, -6.37, 18.33)
0.35	(3.33e-006, 1.90e-006, -2.98e-008)	35.79	(-3.64, -5.87, 17.62)
0.4	(4.29e-006, 9.53e-007, -2.98e-008)	35.24	(-3.80, -5.48, 17.05)
0.45	(4.29e-006, 1.43e-006, -2.98e-008)	34.84	(-3.95, -5.15, 16.57)
0.5	(3.8e-006, 1.19e-006, -2.98e-008)	34.55	(-4.08, -4.86, 16.14)
1	(4.29e-006, 1.43e-006, -7.45e-008)	33.94	(-4.19, -3.64, 13.58)
1.5	(3.57e-006, 1.54e-006, -5.96e-008)	33.52	(-2.68, -4.05, 13.28)
2	(4.52e-006, 2.35e-006, -2.42e-008)	32.72	(-0.92, -4.83, 14.39)
2.5	(5.00e-006, 2.92e-006, -1.49e-008)	32.30	(0.36, -5.42, 15.83)
3	(5.24e-006, 1.43e-006, -2.04e-008)	32.27	(1.08, -5.72, 16.84)
3.5	(7.15e-007, 7.7e-007, -2.04e-008)	32.30	(1.72, -5.70, 17.24)
4	(3.57e-006, 1.07e-006, -3.72e-009)	32.27	(2.54, -5.32, 17.21)
4.5	(2.14e-006, 1.3e-006, -9.31e-010)	32.24	(3.47, -4.62, 16.98)
20	(8.82e-006, -5.96e-007, -2.57e-008)	32.23	(-3.35, -4.63, 16.74)
40	(5.72e-006, -2.38e-007, 5.50e-008)	32.23	(-2.56, -5.13, 16.74)
60	(9.53e-006, -5.96e-008, 7.19e-008)	32.23	(-1.66, -5.47, 16.74)

omezenou přesností, tak se velmi hodnoty nezachovají přesně, ale s určitou odchylkou, na hranici přesnosti výpočtů.

3.2.2.3 Měření

Otáčení je zřejmé z obrázku 3.4, po úvodní deformaci skutečně vznikne rotační pohyb okolo správné osy. Pro posouzení zákonů zachování fyzikálních veličin poslouží naměřené hodnoty v tabulce 3.1.

Zákon zachování hybnosti Zákon zachování hybnosti platí, v mezích daných přesností výpočtů. Zpočátku je hybnost zcela rovná nulovému vektoru, později se všechny tři složky pohybují okolo 10^{-6}, což je zcela zanedbatelná hodnota.

Zákon zachování kinetické energie Energie se v první části simulace silně ztrácí. Je to zároveň doba, kdy dochází k deformacím tělesa. Jakmile

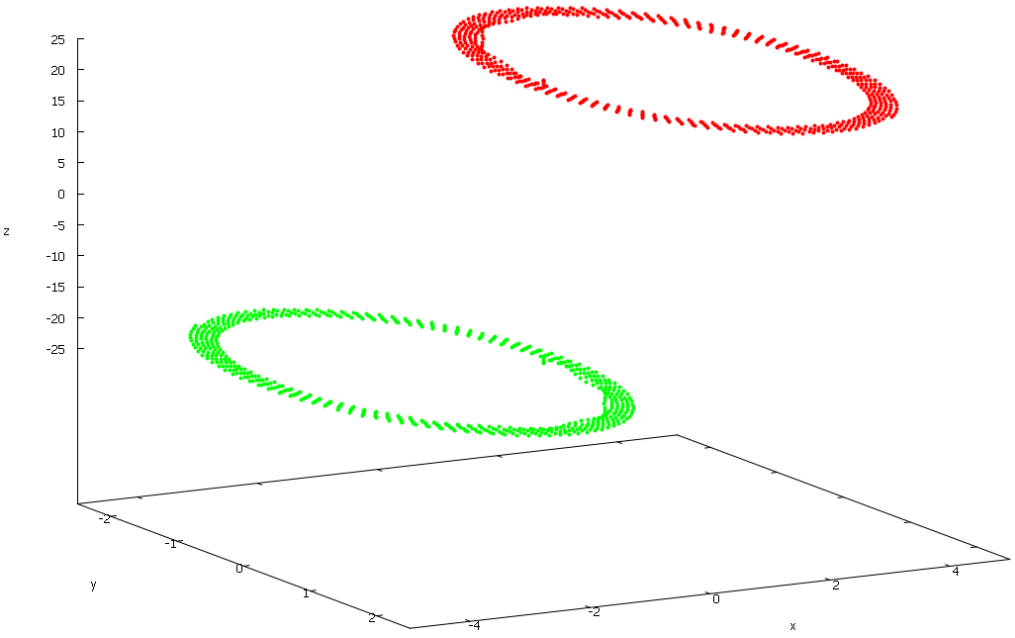
se tvar tělesa ustálí (okolo 0.5 s), energie se udržuje na stále stejné (resp. podobné) hodnotě.

Zákon zachování momentu hybnosti Moment hybnosti se chová nejzvláštněji ze všech měřených hodnot. Na první pohled se nezachovává, nicméně nějaká pravidelnost v jeho hodnotách je. Když jsem vykreslil jeho hodnoty do grafu 3.5, ukazuje se, že po úvodním ustálení hodnot se vektor momentu hybnosti pohybuje po elipse.

3.2.3 Test rotace

Z prvního testu je patrné, že v modelu vzniká rotační pohyb. Ovšem ve třech rozměrech je rotační pohyb fyzikálně celkem komplikovaný, pokud se těleso neotáčí podle některé ze svých os souměrnosti. Těleso, které se totiž otáčí kolem osy souměrnosti, se bude chovat jako setrvačník a bude bránit změně směru osy otáčení.

Obrázek 3.6: Polohy okrajových bodů



3.2.3.1 Svět

Svět bude podobný jako v prvním testu, struktura ale bude delší (3×3×10 bodů). V počátku budou opět protilehlé hrany rozpohybovány a po 1 vteřině

(20 výpočetních kroků), kdy už by měl být pohyb ustálen, bude aplikována síla na konce struktury tak, aby vytvořila rotační pohyb kolem osy kolmé k aktuální ose otáčení. Aby se efekt projevil, síla první, způsobující úvodní rotaci, bude mnohem vyšší než síla druhá, „destabilizační“.

3.2.3.2 Požadovaný výsledek

Ideálním výsledkem testu by měla být situace, kdy úvodní rotace po délce kvádru stabilizuje objekt, takže kolmá síla poté nezpůsobí otočení objektu. K analýze výsledků budeme sledovat polohu středů stran, ležících na původní ose otáčení.

3.2.3.3 Naměřené hodnoty

Jak je patrné z grafu 3.6, tak druhá síla, působící kolmo na původní osu otáčení, způsobila pouze rotaci primární osy otáčení.

3.2.4 Test kolizí a determinizmu

Další z vyšších funkcí systému je detekce kolizí. Prestože je zabudovaná detekce kolizí velmi jednoduchá, vytváří velké množství interakcí. Tím je tento test ideální i pro vyzkoušení determinizmu - všechny interakce a silová působení musí být deterministické, aby při případném současném běhu více instancí (např. několik klientů v počítačové hře) dopadly výpočty naprosto stejně.

3.2.4.1 Svět

Svět zde tvoří plocha, ukotvená vůči gravitaci (tzn. gravitace na ni nepůsobí), a krychle, která padá na její okraj. Plocha má navíc obrovskou hmotnost, tudíž srážka s krychlí s ní pohne zcela zanedbatelně.

Svět je navíc při každém spuštění náhodně upraven - ovšem pouze po technické stránce. Rozmístění bodů a struktur je stále stejné, ale pořadí struktur v simulaci a pořadí bodů v krychli se náhodně mění.

3.2.4.2 Požadovaný výsledek

Od kolize čekáme, že zastaví jednu z hran padající krychle, čímž krychli roztočí podle její osy. Plocha by se neměla téměř vůbec pohnout.

Tabulka 3.2: Výsledné pozice a rychlosti v závislosti na náhodné masce ve floatu

maska	$\overrightarrow{x}$	$\overrightarrow{v}$
6	(-7.93459, -12.0873, -1.04082)	(-1.52207, -2.06628, -0.928203)
7	(-7.93458, -12.0873, -1.04082)	(-1.52207, -2.06628, -0.928203)
11	(-7.93459, -12.0873, -1.04082)	(-1.52207, -2.06628, -0.928203)
2	(-7.93458, -12.0873, -1.04082)	(-1.52207, -2.06628, -0.928203)
0	(-7.93459, -12.0873, -1.04082)	(-1.52207, -2.06628, -0.928204)
6	(-7.93459, -12.0873, -1.04082)	(-1.52207, -2.06628, -0.928203)

Zároveň budeme v několika pokusech v určitém časovém bodu (po 180 krocích výpočtu) měřit polohu a rychlost jednoho (pokaždé stejného) bodu v krychli. Pokud je engine naimplementován správně, tak by měla hodnota vyjít v každém testu stejná. Tato podmínka je navíc ztížena náhodností pořadí bodů a struktur, která tím testuje i nezávislost na pořadí (nezávislost na pořadí sice není pro synchronizaci důležitá, jelikož pořadí bodů i struktur bude ve všech instancích stejné. Ale pokud by výsledek na pořadí závislý byl, tak se jedná o chybu implementace fyzikálního modelu, jelikož pořadí bodů nemá žádný fyzikální význam, a přesto by ovlivnilo výsledek.

Celý test provedeme ve floatu i doublu, protože nám jde o absolutní přesnost.

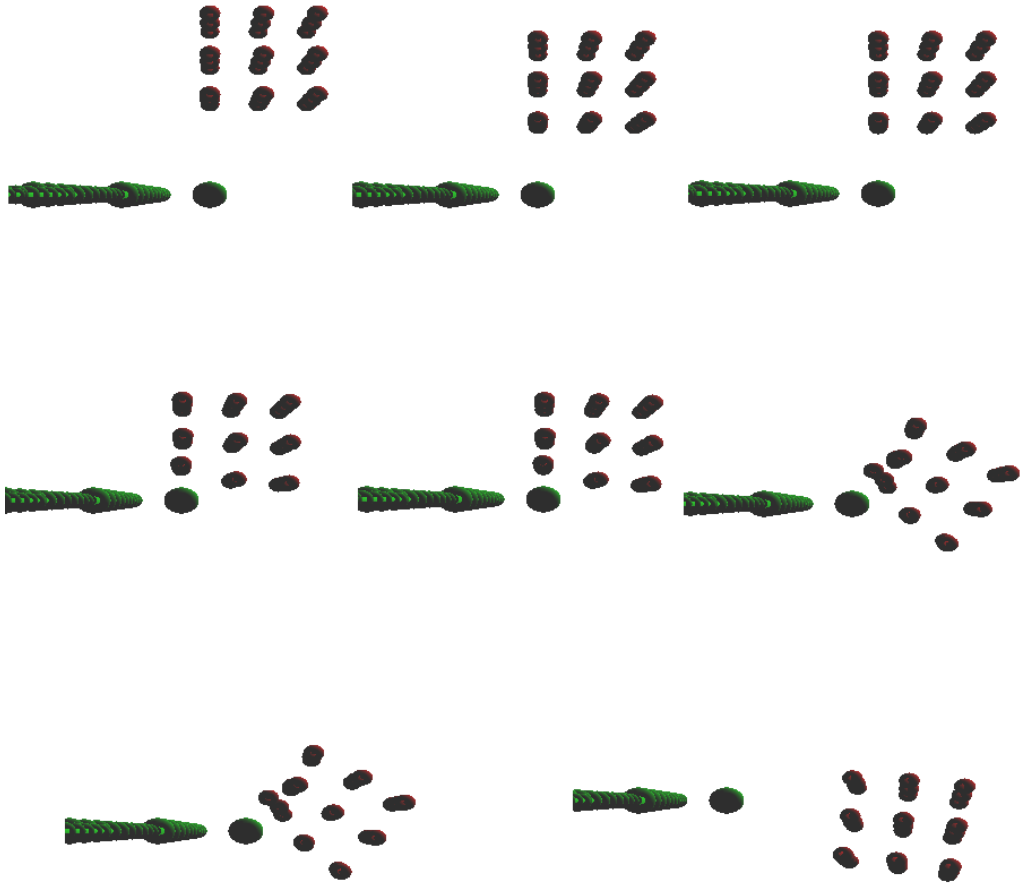
3.2.4.3 Naměřené hodnoty

Tabulky 3.2, 3.3 a 3.4 ukazují naměřené hodnoty v závislosti na „masce“, která ukazuje, které z náhodných úprav pořadí bodů a struktur byly provedeny. Testy se stejnou maskou tedy začínají s identickým datovým vzorkem, jiná maska naznačuje permutovaný datový vzorek.

Tabulka 3.2 ukazuje závislost výsledných poloh a rychlostí ve floatu (hodnoty jsou zaokrouhlené na dostatečnou přesnost, aby byl vidět rozdíl). Na posledních pozicích už jsou vidět mírné rozdíly. Ačkoli pro výsledek je rozdíl zanedbatelný, v praxi by se mohlo stát, že se bude rozdíl načítat, až v jednu chvíli vzroste natolik, že již zásadním způsobem ovlivní stav světa. Zároveň je vidět, že pro stejnou masku (test s maskou 6 proběhl dvakrát) je výsledek totožný (a je totožný i v celém rozsahu přesnosti).

V tabulkách 3.3 a 3.4 můžeme vidět stejné hodnoty, tentokrát ze simulace počítané v doublech. Vidíme, že nepřesnost také nastává, ale mnohem nižší

Obrázek 3.7: Průběh testu kolize



Tabulka 3.3: Výsledné pozice v závislosti na náhodné masce v doublu

maska	$\overrightarrow{x}$
5	(-7.93457389926414, -12.087266960654, -1.04082707627609)
6	(-7.93457389926415, -12.087266960654, -1.04082707627609)
0	(-7.93457389926412, -12.087266960654, -1.04082707627608)
1	(-7.93457389926412, -12.087266960654, -1.04082707627608)

Tabulka 3.4: Výsledné rychlosti v závislosti na náhodné masce v doublu

maska	$\overrightarrow{v}$
5	(-1.52206587908737, -2.06628217721569, -0.928207569394508)
6	(-1.52206587908738, -2.06628217721569, -0.928207569394509)
0	(-1.52206587908735, -2.06628217721569, -0.928207569394505)
1	(-1.52206587908735, -2.06628217721568, -0.928207569394504)

- kolem 14. desetiného místa, ovšem po pouhých 180 simulovaných krocích, což odpovídá 9 vteřinám běhu. Je možné, že simulace poběží i několik hodin.

Jelikož velikost nepřesnosti je závislá na použítém typu proměnných, tak lze usuzovat, že není způsobena chybou v implementaci, ale nepřesností výpočtů s pohyblivou desetinou čárkou (resp. neasociativností jejich sčítání - tento problém je znám[3]). Tuto nepřesnost bohužel nelze nijak odstranit, a je potřeba s ní počítat při návrhu programů využívajících tuto knihovnu.

Náhled na průběh testu lze vidět na obr. 3.7.

3.2.5 Test destrukce

Závěřečný test se bude ze všech testů nejvíce blížit skutečnému nasazení enginu. Jednak vyzkouší pokročilejší metody konstukce struktur, ale hlavně vyzkouší ničivou srážku dvou objektů. Jelikož model destrukce je extrémně jednoduchý, bez jakéhokoli fyzikálního základu, tak nebudeme zkoumat přesnost výsledků, ale spíše vizuální dojem.

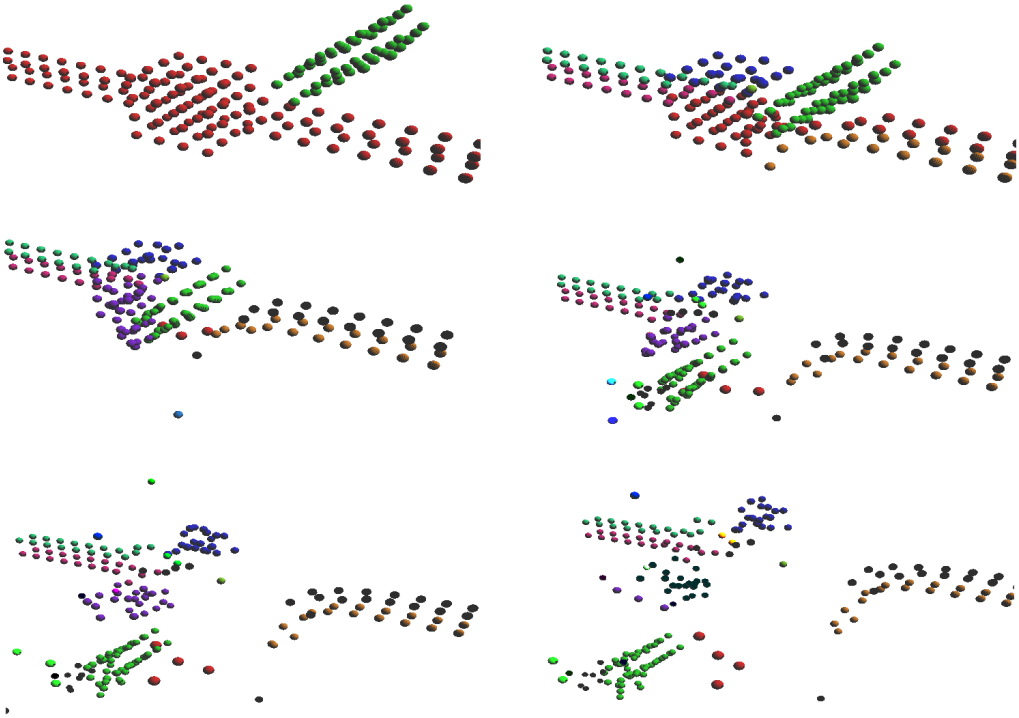
3.2.5.1 Svět

V centru simulace stojí struktura složená ze třech menších kvádrů (dále nazývána „družice“, jelikož ji má připomínat). Tato družice je vytvořena pomocí dodaných nástrojů. Nedaleko od ní je umístěna deska z mnohem tvrdšího (tuhost pružin je vyšší) a těžšího materiálu. Ta začíná v pohybu směrem ke středu družice. Po chvíli dojde k destruktivní srážce.

3.2.5.2 Požadovaný výsledek

Při srážce obou těles musí být správně detekovány kolize a vyvozeny následky. Deska by, vzhledem ke své tvrdosti, rychlosti a váze, měla proletět skrz družici a rozpůlit ji. Systém musí správně rozpoznat oddělení částí a rozdělit družici na několik struktur.

Obrázek 3.8: Průběh testu



3.2.5.3 Pozorování

Po srážce skutečně dojde k rozdělení těles, navíc s vymrštěním změti odštěpků do prostoru, což je ještě lepší výsledek, než byl původně očekáván. Jak je patrno z obrázku 3.8, struktury jsou odděleny (všechny body jedné struktury mají stejnou barvu, každá struktura má unikátní barvu). Lze zpozorovat větší množství malých, často jednobodových, struktur.

3.2.6 Nasazení

Přiložená knihovna je připravená k okamžitému použití. Kompilovaná je pod MinGW 4.7 na operačním systému Windows, a to jako statická i dynamická knihovna (pochopitelně uživatel použije jednu z nich). Tato předkompilovaná verze obsahuje kompletně všechny třídy popsané v sekci Balíčky a třídy 3.1. Drtivá většina tříd je poskytovaná jako šablony, jejichž parametr je datový typ s pohyblivou desetinou čárkou. Takové třídy jsou všechny předkompilované pro double a float.

Knihovnu lze zkompilovat pro libovolnou platformu, je závislá pouze na standardních knihovnách C++. K tomu je přiložen CMakeLists.txt, který umožňuje kompilaci pomocí hojně rozšířeného programu CMake. Knihovnu lze i jednoduše připojit k vlastnímu projektu a kompilovat dohromady, zdrojové kódy celé knihovny jsou také samozřejmě přiloženy.

Ke knihovně ja taktéž dodána uživatelská dokumentace v anglickém jazyce, která obsahuje dokumentaci všech tříd, popis testů a základní úvod do použití knihovny.

Závěr

Podařilo se implementovat knihovnu pro zadanou fyzikální simulaci. Zvolená metoda implementace byla zaměřena na rozšiřitelnost za cenu nižší efektivity. Knihovna je vybavena všemi potřebnými funkcemi pro běh fyzikální simulace, včetně detekce kolizí a destrukce objektů.

Fyzikální model použitý jako základ této knihovny, se ukázal být značně problematický. V testech vyšlo najevo, že kromě očekávaného stabilního stavu existují i jiné stabilní stavy, které představují značnou deformaci oproti úvodnímu stavu, a struktura se do nich může dostat během běžné simulace. Je to způsobeno nevhodnou volbou modelu pro silové působení pružin. Vhodnější by byl model, který nedovolí bodům přiblížit se příliš blízko (nejlépe takový, který v nulové vzdálenosti působí nekonečnou silou). Zároveň, k zajištění funkčnosti takového modelu, by bylo potřeba implementovat lepší metoda pro řešení diferenciálních rovnic než je zvolená Eulerova metoda.

Nicméně i přes tyto nevýhody nabízí engine některé zajímavé funkce, především tedy destrukci, které rozhodně mohou najít uplatnění, zvláště při tvorbě počítačových her.

Literatura

[1] Coumans, E.: Bullet Physics Manual. http://bullet.googlecode.com/svn/trunk/Bullet_User_Manual.pdf, [27/4/2013].

[2] Fiedler, G.: Game Physics: Integration basics. http://gafferongames.com/game-physics/integration-basics/, [10/5/2013].

[3] Garzarán, M. J.: Floating Point Arithmetic. http://www.cs.uiuc.edu/class/fa07/cs498mjg/notes/floating-point.pdf, [10/5/2013].

[4] LucasArts: The New Technology of The Force Unleashed. http://www.lucasarts.com/games/theforceunleashed/gameinfo/news/technology.html, [27/4/2013].

[5] Nealen, A.; Müller, M.; Keiser, R.; aj.: Physically based deformable models in computer graphics. In *Computer Graphics Forum*, ročník 25, Wiley Online Library, 2006, s. 809–836. Dostupné z: `http://www.matthiasmueller.info/publications/egstar2005.pdf`

[6] Seymour, M.: DMM: FEA for VFX. http://www.fxguide.com/featured/dmm-fea-for-vfx/, [27/4/2013].

[7] Štoll, I.; Tolar, J.: *Teoretická fyzika*. Česká technika - nakladatelství ČVUT, 2008.

Příloha **A**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `bin` ..... adresář s kompilovanou knihovnou a testy
- `doc` ..... adresář s dokumentací knihovny
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Labut_Martin_2013.pdf` ..... text práce ve formátu PDF