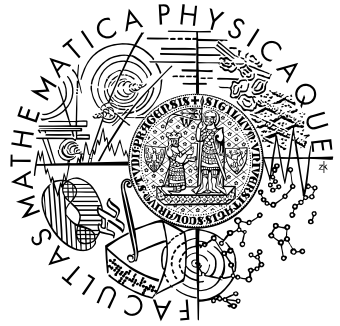
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Svoboda

Virtuální hřiště

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Zuzana Vlčková

Studijní program: Informatika, programování

2008

Děkuji Mgr. Zuzaně Vlčkové za rady a čas, který mi během vypracování této práce věnovala.

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

V Praze dne Tomáš Svoboda

Obsah

Název práce: Virtuální hřiště

Autor: Tomáš Svoboda

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Zuzana Vlčková

e-mail vedoucího: zuzana.vlckova@gmail.com

Abstrakt: Tato práce slouží jako doprovodný dokument k programu Virtuální hřiště. Tento program měl za úkol simulovat pohyb a interakci dokonale tuhých těles v reálném prostředí s gravitací. Měl být optimalizován tak, aby byl schopen na dnešních PC simulovat řádově desítky těles v reálném čase. To bylo bohužel mimo jiné i na úkor fyzikální přesnosti, ale myslím, že se povedlo vytvořit uvěřitelný model chování objektů. Do programu je také zakomponovaná simulace vozidla. Pokusím se zde nastínit problémy, kterým jsem musel při programování čelit a provedu srovnání s jinými algoritmy, které se používají k řešení podobných problémů. Na konci práce uvedu strukturu programu a uživatelskou příručku k programu.

Klíčová slova: fyzikální simulace tuhých těles, fyzika auta

Title: Virtual playground

Author: Tomáš Svoboda

Department: Department of Software Engineering

Supervisor: Mgr. Zuzana Vlčková

Supervisor's e-mail address: zuzana.vlckova@gmail.com

Abstract: Present document describes algorithms used in program Virtual playground. This program was designed to simulate movement and interaction of perfectly rigid bodies in real environment with gravity. It had to be optimised to be able to simulate tens of bodies on contemporary PCs in real-time. It was in fact also to the prejudice of physical precision but I think that the result is physically plausible. There is also vehicle simulation added to the program. I will try to introduce difficulties I had during programming and I will do a comparsion with other algorithms that are used to solve similar kind of problems. In the end, I will describe program structure and provide user guide for the program.

Keywords: physical simulation, rigid bodies, vehicle simulation

Kapitola 1

Úvod

Již několik desítek let se lidstvo snaží v počítači vytvářet simulaci reálného světa. Bezpochyby největší cestu má v tomto směru za sebou počítačová grafika. Bylo vyvinuto bezpočet vykreslovacích technik, které jsou schopny vytvářet fotorealistické scény na obrazovkách monitorů. Hardware běžných dnešních PC je připraven pro práci s pokročilou grafikou a je schopen počítat desítky snímků za sekundu. Jenže se stále zlepšující se kvalitou vykreslovaných obrázků bylo zřejmé, že pro dokonalou iluzi je nutné, aby se realisticky vykreslené objekty ve scéně také reálně chovaly. K tomu bylo potřeba vyvinout algoritmy, které by byly schopny vypočítat dostatečně fyzikálně přesné chování vykreslovaných objektů.

Tyto algoritmy se začaly významněji vyvíjet až v osmdesátých letech. Dřívější hardware nedovoloval provádět složité fyzikální a geometrické výpočty dostatečně rychle pro využití v reálném čase (real-time) a výpočet jednoho snímku trval spíše minuty než zlomky sekund. Nicméně i algoritmy, které nejsou real-time, našly svoje uplatnění. Kromě využití v grafice je možné pomocí nich odhadovat výsledky některých nákladných pokusů. Navíc lze u simulovaných pokusů lépe sledovat jejich průběh. Asi nejlepším příkladem jsou simulace srážek vozidel - crashtestů. Díky počítačům jsou automobilky schopny navrhnout bezpečná auta a neutrácejí zbytečně mnoho peněz za skutečné crashtesty.

Hardware, který by byl schopný počítat simulaci v reálném čase, byl vyvinut teprve nedávno a s ním i optimalizované algoritmy, které jsou schopny ho využít. Real-time fyzikální simulace našly uplatnění také v počítačových hrách. Software, který je schopen rychle zpracovávat grafické informace v simultánním čase interakce uživatele s prostředím, se nazývá engine. Jedna z prvních her, která implementovala pokročilý fyzikální engine byl Half-life 2,

vyvinutý v roce 2004 společností Valve [16]. Tento krok se ukázal jako správný a nemalé náklady na vývoj enginu se bohatě vyplatily. Od té doby vzniklo několik fyzikálních enginů a většina dnešních her má některý z nich v sobě zakomponováno. Také se začal vyvíjet hardware určený primárně pro výpočet fyzikálních simulací, ale dodnes není v běžných PC moc rozšířený - zdaleka ne tak, jako grafické akcelerátory. Některé návrhy počítají s tím, že se grafický akcelerátor bude starat i o fyzikální stránku.

Další příklady využití simulace v reálném čase:

- Trenažéry – významně pomáhají například pilotům při tréningu. Na trenažérech je možno cvičit krizové situace, které by byly v reálu příliš nebezpečné.
- Robotika – roboti mohou simulovat výsledky proveditelných akcí a díky tomu lépe plánovat své akce.

Virtuální hřiště implementuje algoritmy pro real-time simulaci. Pro optimalizaci se simulují pouze dokonale tuhá tělesa (rigid bodies). To jsou tělesa, jejichž tvar ani rozložení váhy se nemění. Například dutá koule, ve které je voda, není dokonale tuhé těleso, i když zvenku nemění svůj tvar.

Kapitola 2 se zabývá detekcí kolizí. Jsou uvedeny základní problémy při výpočtu detekce kolizí a příklady jejich řešení včetně detailního popisu algoritmu použitého v programu Virtuální hřiště. V kapitole 3 je podobným stylem rozebírána fyzikální simulace. Kapitola 4 popisuje implementaci programu. Kapitola 5 obsahuje uživatelskou příručku a kapitola 6 tvoří závěrečné zhodnocení.

Kapitola 2

Detekce kolizí

Detekce kolizí je nezbytná součást každého programu počítajícího fyzikální simulaci, která umožňuje vzájemnou interakci těles formou kolizí. Před každým pohybem těles v časovém kroku je nutné zjistit jestli - a popřípadě za jakých podmínek - tělesa kolidují. Toto je vážný problém pro real-time počítané algoritmy, které musí kontrolovat kolize až několiksetkrát za sekundu, aby byla hloubka zanoření těles do sebe minimální.

Byly vyvinuty i techniky, které dokáží vypočítat přesný čas kolize. Nicméně tyto algoritmy se ukázaly jako NP-těžké, musí se řešit pomocí heuristik a pro komplexní real-time simulace jsou nevhodné.

Další způsob, jak se s tímto problémem vypořádat, je počítat kolize méně často a v případě nálezu kolize se vracet v čase a hledat přesnější dobu, kdy ke kolizi došlo. To ale přináší spoustu dalších problémů, které je potřeba řešit. Například při velkém počtu kolidujících těles se může program velmi zpomalit, a to je v real-time nepřípustné.

Cesta, kterou jsem zvolil já, se zdá být nejjednodušší. V programu Virtuální hřiště se kolize počítají v řádově desítkách až stovkách testů za sekundu. Toto číslo lze v programu nastavit jako parametr a jako optimální bylo zvoleno číslo 128, které se ukázalo být rozumným kompromisem mezi rychlostí a přesností simulace. Nicméně i menší hodnoty se dají použít bez větších potíží. Pro dobrý výsledek je nutné kolize dobře zakomponovat do fyzikálních simulací, o kterých budu hovořit v další kapitole.

2.1 Broad phase

První krok v detekci kolizí je tzv. broad phase. Během této fáze se určí, které dvojice těles mohou být potenciálně v kolizi. Tyto dvojice se poté pošlou do další fáze, která finálně rozhodne, zda jsou tělesa v kolizi a případně najde přesné místo kolize a normálový vektor. Pokud by se měly testovat všechny dvojice, tak by byla asymptotická složitost $O(n^2)$. S použitím vhodných algoritmů se dá tato složitost snížit na $O(n \log n)$, a to je při větším počtu těles velmi výrazné zrychlení.

Existuje několik algoritmů pro broad phase detekci kolizí. Například algoritmus SAP (Sweep and Prune [9]), který si ukládá seřazené horní a dolní hodnoty souřadnic těles, podle kterých rychle určuje překrývající se tělesa. Dále se používají různé stromové struktury (Sphere trees [10], AABB trees [11], BSP [12]).

Já jsem zvolil pro Virtuální hřiště algoritmus využívající Octree [13], což je zástupce AABB stromů. Octree je stromová struktura, která rozděluje prostor na osově orientované krychle (AABB – axis aligned boundnig box). Každý uzel stromu může mít až 8 potomků. Jako kořen je krychle, která obsahuje celý simulovaný prostor a její potomci dále zjemňují prostor do menších krychlí. Krychle, které jsou v listech stromu, obsahují seznam těles, které do nich zasahují. Ukončující podmínkou pro zjemňování je většinou určitý počet těles v krychli. Tato tělesa musí být ale nepohyblivá, jinak by se musela struktura při každém posunutí těles celá přepočítat a přealokovat paměť, a to by bylo příliš nákladné. Nicméně u pohyblivých těles je nutné, aby byly přidávány a ubírány dynamicky. Tímto se vytvoří skupiny těles, které se budou testovat v další fázi detekce kolizí.

Jako celkem nepříjemná nevýhoda se jeví to, že se musí v každém kroku u pohyblivých těles přepočítat, do kterých listových krychlí náleží. To není časově úplně levná záležitost, i když je několikanásobně levnější, než kdyby se musely testovat kolize mezi všemi tělesy. Zde je ještě prostor pro další optimalizaci.

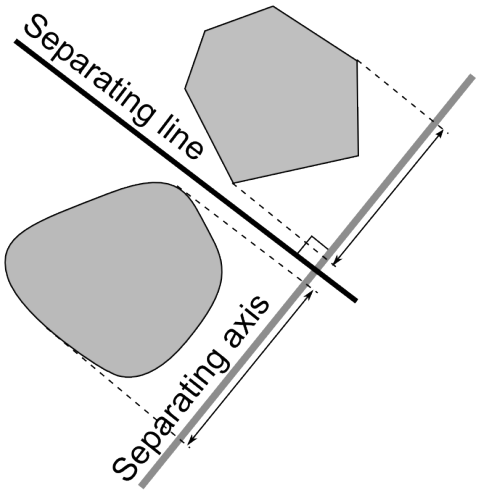
2.2 Narrow phase

Během této fáze se určuje, zda konkrétní dvojice těles mezi sebou kolidují. Je také nutné případně spočítat přesnou pozici kolize, normálový vektor a hloubku zanoření těles do sebe. Tyto hodnoty jsou potřeba pro výpočet reakce na kolizi. Kolizí mezi dvěma tělesy může být víc.

Toto je velmi časově náročné a je to počítáno velmi často, proto je pro real-time simulaci nezbytně nutná velká míra optimalizace. Kontrolovat složitě tvarovaná tělesa je velmi těžké, a proto se obalují do tzv. kolizních primitiv. Tato primitiva jsou v zásadě konvexní a lehce geometricky reprezentovatelná. Mezi nejpoužívanější patří krychle, koule a válec. Popřípadě mohou být tělesa obalena do tzv. „low poly model“. To je model složený z malého množství mnohoúhelníků (polygonů).

V programu Virtuální hřiště je implementována detekce kolizí krychlí mezi sebou a krychlí s trojúhelníkovým modelem.

Bohužel se mi nepodařilo najít vhodný algoritmus, který by tento úkol řešil. Musel jsem vyvinout vlastní algoritmus založený na SAT (Separating axis theorem [8]). Tento teorém říká, že pokud se dvě nekonvexní tělesa neprotínají, tak existuje osa, na kterou se provede projekce těles, taková, že se projekce nepřekrývají (viz obr.1). Tato osa se nazývá separující (separating axis).



Obr.1: Separating Axis Theorem

Pro polygonový model jsou jako potenciální separující osy použity normály polygonů a dále vektorové součiny mezi normálami z rozdílných těles. Je vidět, že počet těchto os roste se čtvercem počtu polygonů v modelu testovaných těles, proto je potřeba, aby bylo těchto polygonů co možná nejméně. Například pro testování kolize dvou krychlí (každá má 6 stran - polygonů) je potřeba 15 os. 3 osy jako normály polygonů pro každou krychli (rovnoběžné osy se mohou sloučit). Dále 9 os jako vektorové součiny normál rozdílných krychlí.

SAT má tu výhodu, že k tomu, aby se rozhodlo, že se tělesa neprotínají stačí, aby se našla jakákoli separující osa. Tímto se vcelku rychle vyřeší případy, kdy tělesa nekolidují a těchto případů je většina. Pokud se tělesa protínají, tak se sice musí zkontrolovat všechny osy, ale najde se alespoň hloubka zanoření a normálový vektor kolize. Hloubka zanoření je nejmenší hodnota z délek překrývajících se částí projekcí těles na osu a normálový vektor je tato osa.

Nicméně algoritmus nedokáže určit přesnou pozici kolize a ta je nezbytně nutná pro následnou reakci na kolizi. Pokud tedy SAT označí dvojici těles jako kolidující, je ještě potřeba spočítat přesné pozice kolizí.

V programu je to řešeno tak, že se nejprve zkontroluje, zda neleží některý vrchol jedné krychle v druhé krychli. Poté se otestuje, jestli neprotíná některá hrana první krychle nějakou ze stran druhé krychle a případně se vypočítá

průsečík. Je vidět, že je algoritmus nejvíc efektivní, pokud jsou tělesa daleko od sebe, ale s ubývající vzdáleností je nutné kontrolovat více os a v případě kolize je ještě nutné spočítat průsečíky.

Kapitola 3

Fyzikální simulace

Fyzikální simulace v programu Virtuální hřiště je založena na publikaci [1] Bridson a kol. (2003). V této publikaci autoři představují algoritmus pro simulaci tuhých těles, ale nezaměřují se na real-time. Abych dosáhl dostatečné rychlosti simulace, musel jsem algoritmus upravit, nicméně idea zůstala stejná.

3.1 Numerická integrace

Počítač nedokáže počítat se spojitými veličinami, a tak ani čas v simulaci nemůže být kontinuum. Proto to, co se od fyzikálního simulátoru obecně požaduje, je výpočet pozic těles v jednotlivých časových krocích tak, aby se dala tělesa vykreslit.

K výpočtu pozice, je třeba nejdřív určit, jaké síly působí na těleso. Ty jsou většinou dány (gravitace) a není nutné je složitě počítat. Ze sil působících na těleso se vypočítá zrychlení pomocí rovnice a = F/m.

Dále:

$$\frac{dv}{dt} = a$$

$$\frac{dx}{dt} = v$$

To znamená, že pokud jsou známy působící síly, tak k nalezení pozice a rychlosti stačí integrovat tyto rovnice. Avšak ve většině případů nelze řešit integraci algebraicky, a proto se dostane ke slovu numerická integrace, která dokáže výsledek odhadnout. V tomto případě funguje tak, že začne

s počátečními hodnotami pozice a rychlosti a poté po malých časových krocích zjišťuje, jak se tyto hodnoty změní.

Existuje několik způsobů jak provádět numerickou integrací. Základní – nejjednodušší - ale také nejhorší je Eulerova integrace [14]. Nechť *dt* je délka časového kroku, v_0 původní rychlost a x_0 původní pozice, potom Eulerova integrace vypadá následovně:

$$v = v_0 + a \cdot dt$$

$$x = x_0 + v \cdot dt$$

Toto ale funguje přesně, jen v případě, že je změna integrované veličiny konstantní. To platí jen u rychlosti, protože na tělesa působí konstantní síla (gravitace). U výpočtu pozice se ale integruje rychlost a ta konstantní není. U kol vozidla není přesný ani výpočet rychlosti, protože na kolo působí pružina proměnnou silou.

Například pro časový krok o velikosti jedné sekundy:

Těleso padá 10 sekund volným pádem se zrychlením $a = 10\,m/s$ a chceme spočítat dráhu, kterou urazí. Na to existuje jednoduchý fyzikální vzorec $x = \int a \cdot t\,dt = \frac{a \cdot t^2}{2}$. Po dosazení $x = 500\,m$. Při použití Eulerova integrátoru dostáváme:

$$t = 0 : x = 0\,m, v = 0\,m/s$$

$$t = 1 : x = 10\,m, v = 10\,m/s$$

$$t = 2 : x = 30\,m, v = 20\,m/s$$

$$t = 3 : x = 60\,m, v = 30\,m/s$$

$$t = 4 : x = 100\,m, v = 40\,m/s$$

$$t = 5 : x = 150\,m, v = 50\,m/s$$

$$t = 6 : x = 210\,m, v = 60\,m/s$$

$t = 7 : x = 280\,m, v = 70\,m/s$

$t = 8 : x = 360\,m, v = 80\,m/s$

$t = 9 : x = 450\,m, v = 90\,m/s$

$t = 10 : x = 550\,m, v = 100\,m/s$

Z toho vyplívá, že za 10 sekund se nakumulovala nepřesnost 50m a stále roste. Pro takto dlouhý časový krok je tedy Eulerova metoda nepoužitelná. V simulaci se ale používá délka kroku v řádu milisekund a v tomto případě se dá označit nepřesnost za zanedbatelnou. Ovšem ne pro komplexní simulace, kde se velmi rychle mění působící síly. Například pro simulaci pružiny (zejména s velkou tuhostí) je Eulerova integrace nevhodná a nepřesnost může snadno dosáhnout kritických hodnot, kdy začne pružina kmitat stále rychleji až nakonec „vybuchne“.

V programu Virtuální hřiště se i přes to, že to na první pohled nevypadá jako dobrý nápad, používá právě Eulerova integrace, a to z toho důvodu, že je nejjednodušší a časový krok je dostatečně malý (kvůli kolizím). Původně v programu nebyly simulovány pružiny, ale ty přibyly se simulací vozidla. U těchto pružin lze pozorovat problémy, pokud je časový krok delší než 10-15 milisekund.

Existuje spousta typů simulací, kde je Eulerův algoritmus nepoužitelný. Například letecký simulátor, kde působí proměnné aerodynamické síly potřebuje o hodně robustnější integrátor. Mezi často používané patří tzv. RK4 (Runge-Kutta čtvrtého řádu [15]). RK4 v podstatě dělá to, že ohodnotí derivaci ve čtyřech bodech a pomocí těchto hodnot určí zakřivení, aby získal dobrý odhad derivace. Ale ani tento integrátor není dokonale robustní a i s jeho použitím může dojít k „výbuchu“ simulace, i když se to nestává zdaleka tak často jako u Eulerovy metody.

3.2 Průběh simulace

Akce, které je nutno provést v každém kroku simulace, jsou:

- výpočet nové pozice/rychlosti
- detekce kolizí
- reakce na kolize
- reakce na kontakty

To, co je na algoritmu publikovaném v [1] Bridson a kol. (2003) zajímavé, je delikátní propojení ošetření kolizí/kontaktů a numerického integrátoru.

Většina ostatních simulátorů postupuje tak, že vypočítají pomocí numerické integrace novou rychlost a pozici, poté zjistí kolize a případně je ošetří. To má ale svoje nedostatky. Například běžný případ, kdy je krychle v klidu na podložce a působí na ní gravitace. Nejdříve se vypočítá nová rychlost a pozice. To způsobí, že se krychle propadne pod zem, to následná detekce kolizí odhalí, ale protože krychle má nenulovou rychlost, tak se odrazí zpět s nenulovou rychlostí. Dále ke kolizi nedochází, protože se krychle pohybuje ven z podložky, takže fáze řešení kontaktů nic neudělá. Tím vzniká „poskakování“, které je nutné dále řešit. Jedno z možných řešení je použití tzv. rychlostních prahů (krychle se považuje za stacionární, pokud se pohybuje dostatečně malou rychlostí). Ale ani to nedává v některých situacích uspokojivé výsledky (například pokud se krychle pohybuje na nakloněné rovině).

Ve Virtuálním hřišti postupuje simulace tak, že se kolize odhadují krok dopředu. Integrátor zjistí novou pozici x a rychlost v (v_0 původní rychlost a x_0 původní pozice) a s těmito hodnotami detekuje kolize. Zjistí se kolizní páry, ale kolize se ošetří s původní rychlostí v_0. Poté se vypočítají nové rychlosti/pozice a ošetří se kontakty (ty se počítají jako dokonale nepružné odrazy).

Tímto se zamezí poskakování krychle na podložce z předchozího příkladu. Nejprve se detekují kolize s hodnotami v a x, ale kolize se řeší s rychlostí v_0 a ta je nulová, proto se tento případ v ošetřování kolizí neřeší. Řeší se až ve fázi ošetřování kontaktů, po přepočítání pozice/rychlosti. Kontakt se počítá jako nepružný odraz, tudíž se krychle neodrazí a rychlost zůstane nulová.

V reálném případě může nastat víc kolizí. Ošetření jedné kolize může ovlivnit jinou kolizi (změnou rychlosti). Bohužel není zřejmé, která kolize nastala jako první, a proto se kolize ošetřují najednou. Vzhledem k tomu, že mohou vzniknout i nové kolize, je nutné proceduru několikrát opakovat. V programu je pomocí GUI Virtuálního hřiště možné měnit, kolikrát se tato procedura opakuje pomocí parametru „*Resolve iterations*“.

Obdobná je i situace u ošetřování kontaktů. Tady může dojít k ještě horším případům. Například pokud leží na sobě několik krychlí, tak se jednotlivé kolize velmi silně ovlivňují. Nepřesnosti jsou u kontaktů více viditelné, a proto se v programu používá dvojnásobný počet iterací ošetřování než u kolizí. Simulování „zdí“ postavených z krychlí je obecně velmi časově náročné a citlivé na přesnost. V těchto případech je složitá jak detekce, tak i ošetření kolizí.

Optimalizace simulace Virtuálního hřiště oproti [1] Bridson a kol. (2003) spočívá zejména v tom, že se nepočítají znovu kolize pro ošetření kontaktů, a také v tom, že pořadí vyhodnocování kolizí (a kontaktů) je náhodné. V [1] Bridson a kol. (2003) mají přednost kolize s větší hloubkou. Nicméně i přesto simulace funguje vcelku věrohodně.

3.3 Simulace vozidla

Vozidlo je ve Virtuálním hřišti simulováno jako těleso, ke kterému jsou připojena další čtyři tělesa – kola. Ty jsou připojena přes pružinu s tlumičem. Tuhost pneumatik se zanedbává.

Jako kolizní primitivum pro detekci kolizí kol se pro zjednodušení používá místo válce paprsek. To je mezi real-time simulátory vozidel velmi rozšířené a i když mohou nastat případy, kdy to působí nereálně (například najetí na obrubník), tak se tato optimalizace vyplatí.

Impulsy, které vznikají při kolizi kola, se musí projevit i na vozidle. Konkrétně impulsy, které jsou kolmé na směr působení pružiny jsou automaticky aplikovány i na vozidlo. Impulsy rovnoběžné s pružinou se na vozidlo neaplikují přímo, ale skrz působení pružiny. To způsobí „zhoupnutí“ vozidla.

Motor simulován není, působí jen konstantní silou na poháněná kola. Stejně tak není simulován diferenciál, rozložení brzdného účinku, přítlak spojlerů, ABS, ESP a další technologie používané v dnešních autech, proto ovládání vozidla působí poněkud nestabilně. Pro rozumně fungující simulaci auta by bylo nutné tyto prvky dodělat, ale to nebylo cílem při tvorbě Virtuálního hřiště.

Kapitola 4

Struktura programu

Program byl naprogramován v prostředí Microsoft Visual Studio 8 a je primárně určen pro operační systém Windows. Pro přenositelnost na jiné systémy by mělo stačit poupravit třídu *TTimer*, která pro měření času používá funkci *QueryPerformanceCounter*, která je definována ve *windows.h*.

Pro práci s grafikou program využívá grafickou knihovnu OpenGL [2]. Jako API byl zvolen GLUT [3], který se stará o vytvoření okna, komunikaci s operačním systémem, ošetřování vstupů z klávesnice a myši atd.

Pro GUI (Graphic User Interface) je využita knihovna GLUI [4], díky níž lze snadno vytvořit různé ovládací prvky, pomocí kterých může uživatel za použití myši (popřípadě i klávesnice) modifikovat parametry simulace. Nicméně GLUI má i pár nevýhod. Například je složité (někdy i nemožné) měnit pozice a velikosti ovládacích prvků. Asi největší problém je to, že pokud je některý prvek aktivní, tak přebere focus klávesnici a již nejsou vysílány signály od klávesnice GLUTu a program nemůže reagovat na stisknutí tlačítka. Proto je nutné po použití některého ovládacího prvku překliknout zpět do vykreslované části, aby se vrátila kontrola nad klávesnicí GLUTu.

Součástí práce jsou i zdrojové kódy a projekt, který je možno zkompilovat v prostředí Microsoft Visual Studio 8. Jsou na přiloženém CD zabaleny v souboru „*Virtual Playground source.zip*“.

4.1 Datové struktury

Zde jsou představeny základní třídy používané v programu.

- **TGlobal** – Tato třída obsahuje různé globálně využitelné proměnné. Například pozici/směr kamery, nastavení světel ve scéně, ukazatel na označené těleso atd. Také obsahuje ukazatel na třídu *TPhysics*, která se stará o fyziku.
- **TPhysics** – Obsahuje proměnné a metody týkající se fyziky. Součástí je i vektor těles (*TRigidBody)* a struktura Octree (*TOctree*).
- **TRigidBody** – Abstraktní třída představující těleso. Potomci této třídy jsou *TBox* – krychle, *TVehicle* – vozidlo a *TVehicleWheel* – kolo. Obsahuje vlastnosti tělesa jako pozice, rychlost, orientace a také metody pro jejich vhodnou modifikaci například formou impulsů.
- **TOctree –** Stromová struktura Octree i s metodami pro její naplnění.
- **TGui** – Obsahuje pomocné proměnné pro GUI.
- **TVector3D**, **TMatrix3D** – Struktury s metodami a přetíženými operátory pro snadnější práci s trojrozměrnými vektory a maticemi.
- **TModel** – Struktura 3D modelu složeného z trojúhelníku. Obsahuje metody pro načtení modelu z Wavefront .obj formátu a pro výpočet normálových vektorů u vrcholů.
- **TTimer** – Počítá uběhlý čas a FPS.

4.2 Běh programu

Běh programu začíná inicializací základních struktur (*TGlobal*, *TPhysics*, *TOctree*), načtením modelu z .obj souboru (*TModel*). Dále se model přidá do Octree, vytvoří se GUI (*TGui*) a spustí se hlavní programová smyčka

(*GlutIdle)*. Tam se pokaždé spočítá, za jak dlouho se dokončil minulý krok smyčky (pomocí *TTimer*) a podle toho se určí kolikrát proběhne simulace (záleží na zvolené přesnosti simulace – parametr *simulation precision*).

Simulace probíhá podle kapitoly 3.2 v metodách třídy *TPhysics* (modul *physics.cpp*). Spustí se metoda *DoTimestep*, která provede následující akce:

- Přemístí tělesa do nových listů v Octree.
- Spustí metodu *GetAllExternalForces*. Ta na každém tělesu zavolá virtuální metodu *AddExternalForces* třídy *TRigidBody* přidávající externí síly (gravitaci).
- Spustí metodu *HandleAllCollisions*, která se následně stará o vyřešení kolizí:
  - Vypočte nové pozice/rychlosti pro všechna tělesa (virtuální metody třídy *TRigidBody* – *UpdateVelocity*, *UpdatePosition*), přitom si uloží původní pozice/rychlosti (*SaveTempValues*).
  - Detekuje kolize (viz kapitola 2). Broad phase probíhá v metodě třídy *TOctree* – *OctCollisions*. Následná Narrow phase probíhá ve funkcích z modulu *collisions.cpp*:
    - *DetectBodyCollision* – detekce kolizí mezi tělesy.
    - *DetectGroundCollisions* – detekce kolizí mezi tělesem a zemí.
    - *BoxTriangleCollision* – detekce kolizí mezi tělesem a modelem hřiště.
    - Nalezené kolize jsou ukládány do vektoru (*collisions*).
  - Vrátí tělesům původní rychlosti (*ReloadVelocities*).
  - Několikrát provede reakci na kolize (*ResolveCollision*, popřípadě *ResolveWheelCollision,* pokud koliduje kolo vozidla). Směr

postupu při procházení vektoru kolizí se v každé iteraci mění (může to urychlit řešení – podobný princip jako Shake sort).

- Vrátí tělesům původní pozici (*ReloadPositions*).

- Vypočte novou rychlost, již ovlivněnou kolizemi.
- Vypočítá reakce na kontakty. Probíhá stejně jako reakce na kolize, ale použije se nulová pružnost těles. I zde je nutné provést výpočet několikrát.
- Následuje proces mražení těles:
  - Nejprve se zkontroluje, jestli se těleso pohybuje nezanedbatelnou rychlostí. Pokud ano – resetuje se čítač inaktivity, pokud ne – k čítači se přičte aktuální časový krok (dt).
  - Poté se zkontroluje, jestli čítač dosáhl prahové hodnoty (threshold). Pokud ano, těleso se zmrazí.
  - Další krok je aktivace "visících" těles. Každé těleso má uložený seznam těles, se kterými koliduje a svoji pozici v čase kolize. V tomto kroku se kontroluje rozdíl aktuální pozice a pozice z doby kolize. Pokud je rozdíl větší než prahová hodnota, tak se odmrazí všechna tělesa ze seznamu kolidujících těles. Tímto se ošetří případy, kdy na sobě leží dvě zmražená tělesa a spodní těleso se z nějakého důvodu odsune. Algoritmus tento případ rozezná a vrchní těleso odmrazí.
- Nakonec se ze získaných rychlostí vypočítají pozice těles.

Po skončení iterace simulace se vykreslí scéna (*GlutDisplay)*.

Takto program pokračuje, dokud není ukončen. Mezitím se ošetřují vstupy z klávesnice (*GlutKeyboard*, *releaseSpecialKey*), myši (*GlutMouse*), změna velikosti okna (*GlutReshape*) a vstupy z GUI (*control_cb*).

Kapitola 5

Uživatelská příručka

Program je určen pro operační systém Windows. Po rozbalení souboru „*Virtual Playground executable.zip*“ je možné bez další instalace spustit soubor „*Virtual Playground.exe“*.

Po spuštění programu se zobrazí dvě okna. První (konzolové) není moc důležité, vypisuje se do něj pouze průběh inicializace programu. Druhé okno již graficky znázorňuje hřiště a na okraji okna jsou ovládací panely.

Program se ukončuje buď zavřením jednoho z oken nebo pomocí klávesy *Esc*.

5.1 Ovládání kamery

Kamera se ovládá pomocí šipek v levém dolním rohu obrazovky. Stačí kliknout, podržet a posunout myš chtěným směrem a kamera se patřičně posune. K dispozici jsou čtyři módy pohybu kamery:

- *Camera dir* – Mění směr pohledu.
- *Camera pos* – Mění horizontální pozici kamery.
- *UP/DOWN* – Mění pozici kamery ve vertikálním směru.
- *(FOR/BACK)ward* – Posunuje kameru ve směru pohledu.

Pro velmi jemné posuny je možno přidržet klávesu *Ctrl*, naopak pro rychlejší posuny přidržet *Shift*.

Kamera může fungovat ve třech režimech:

- *Attach camera to selected object* – Kamera je „přichycena“ k označenému tělesu a není možné s ní pohybovat ani označovat jiná tělesa. Pokud není označené žádné těleso, je kamera volná.
- *Free camera, object selection* – Kamera je volná a lze vybírat tělesa kliknutím myši. Označené těleso je ohraničeno modrou barvou.
- *Free camera, push/stop/pull* – Kamera je volná a pomocí myši je možné ovlivňovat tělesa. Kliknutím levého tlačítka myši se do tělesa strčí, kliknutím pravého tlačítka se těleso přitáhne a prostředním tlačítkem se těleso zastaví.

5.2 Označování tělesa

Označit těleso lze pouze pokud je volná kamera (není připojena k tělesu). V takovém případě stačí kliknout jakýmkoli tlačítkem myši na těleso vyobrazené ve scéně. Aktuálně označené těleso je ohraničeno modrou barvou.

5.3 Obecné nastavení

Obecné nastavení je možné provádět pomocí položky *Global values*, která se nachází na pravé straně obrazovky. Kliknutím na ní se zobrazí možnosti nastavení.

- *Draw wireframe* – Zapne mód vykreslování těles formou drátěných modelů.
- *Lights* – Obsahuje nastavení světel ve scéně. K dispozici jsou dvě světla. Je možné:
    - Zapínat/vypínat světlo (*Enabled*).

- Měnit barvu pomocí posuvníků (první posuvník je červená složka barvy, druhý je zelená, třetí modrá).
- Měnit směr, ze kterého přichází světlo, kliknutím a tažením myši (*Direction*).

5.4 Nastavení simulace

K nastavení simulace slouží položka *Physics values*. Kliknutím se položka rozbalí (případně zabalí). Umožňuje měnit následující parametry:

- *Gravity* – Nastavuje gravitaci ve třech dimenzích (Z je nahoru). Je nutné potvrdit pomocí tlačítka *Commit*.
- *Show collisions* – Graficky znázorní kolize. Je vhodné používat současně s „*Draw wireframe*“ (viz 5.3), aby byly kolize lépe vidět.
- *Simulation precision* – Určuje minimální počet kroků simulace za sekundu. Čím větší, tím je simulace přesnější, ale zpomaluje to chod programu.
- *Octree* – Datový strom, do kterého se ukládají tělesa. Vhodným nastavením *Octree depth* (hloubky stromu) je možné zrychlit simulaci. *Show octree* zapíná vykreslování uzlů stromu.
- *Freezing enabled* – Zapne (resp. vypne) zmrazování neaktivních těles, které se poté nesimulují. Tím se ušetří výpočetní kapacita a program je rychlejší. Zmražené objekty zešednou.
- *Resolve iterations* – Nastavuje počet iterací ošetřování kolizí a kontaktů. Čím vyšší číslo, tím přesnější, ale pomalejší simulace.

5.5 Vytváření a modifikace těles

Vytvářet a modifikovat tělesa je možné v položce *Object values*.

Obyčejné těleso se přidává kliknutím na tlačítko *Add new*. Vozidlo se přidává pomocí tlačítka *Add new vehicle*. Po kliknutí na tlačítko je ještě nutné těleso umístit do hřiště. Stačí kliknout do místa, kam má být těleso umístěno a těleso se tam vytvoří.

Pro odstranění tělesa je nutné těleso nejprve označit (viz 5.2) a poté kliknout na tlačítko *Remove*. K odstranění všech těles slouží tlačítko *Remove all*.

Kromě toho je možné modifikovat označené těleso. Je k disposici celá řada parametrů:

- *Vehicle* – Je aktivní, pokud je vybrané vozidlo nebo se vozidlo vytváří.
    - *Engine power* – Síla motoru.
    - *Brake power* – Síla brzd (působí na každé kolo stejně).
    - *Wheels* – Zde lze vybrat kola, která budou modifikována. Když se vytváří nové vozidlo, tak se nastaví všechna kola stejně.
    - *Wheel size* – Velikost kola.
    - *Power gain* – Podíl síly motoru, kterým je kolo poháněno.
    - *Spring stiffness* – Tuhost pružiny u kola.
    - *Damper coef* – Účinnost tlumičů pérování.
    - *Dynamic friction* – Koeficient tření, pokud se kolo pohybuje.
    - *Static friction* – Koeficient tření, pokud je kolo v klidu.
    - *Spring length* – Délka pružiny u kola.
- *Density* – Hustota tělesa.

- *Mass* – Hmotnost tělesa (při vytváření tělesa se používá pro určení hmotnosti hustota a velikost).
- *Elasticity* – Pružnost tělesa.
- *Dynamic friction* – Koeficient tření, pokud se těleso pohybuje.
- *Static friction* – Koeficient tření, pokud je těleso v klidu.
- *Immovable* – Pokud je zapnuto, tak je těleso nepohyblivé.
- *Allow freezing* – Povoluje mrazení tělesa.

Po nastavení parametrů je ještě nutné potvrdit modifikaci stisknutím tlačítka *Commit*.

5.6 Klávesnice

Označené těleso je možné ovládat pomocí klávesnice. Pokud je kurzor myši na ovládacím panelu nebo je vybrán ovládací prvek, neslouží klávesnice k ovládaní vybraného tělesa, ale k obsluze vybraného ovládacího prvku. Pro ovládání tělesa je nutné kliknout kamkoliv do vykreslovaného prostoru.

- Pomocí kláves *Num 8,4,2,6,7,9* je možné přidat tělesu rotaci.
- Šipky a klávesy *Home* a *End* přidávají tělesu rychlost v různých směrech.
- Pokud je označené vozidlo, slouží šipky k ovládání vozidla.
- Klávesy *Page Up*, *Page Down* zrychlují resp. zpomalují krok simulace.
- Klávesa *Esc* ukončuje program.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit program, který by simuloval chování dokonale tuhých těles v reálném prostředí s gravitací. Do této simulace měla být ještě zakomponována simulace vozidla. Důraz byl kladen na rychlost výpočtů, přitom měla být simulace dostatečně přesná, aby bylo chování těles uvěřitelné.

Výsledný program zvládá simulovat řádově desítky až stovky těles v reálném čase při zachování dostatečné přesnosti (testováno na procesoru AMD Athlon 64 X2 Dual 4000+). Poměr mezi rychlostí a přesností simulace je možné měnit pomocí parametrů.

Dalším cílem bylo naprogramovat grafické prostředí, které by znázorňovalo scénu s tělesy. V neposlední řadě měl program vytvořit uživatelské rozhraní, pomocí kterého by mohl uživatel přidávat, ubírat a modifikovat tělesa a měnit parametry simulace za běhu programu.

Struktura programu umožňuje snadno přidávat další typy těles (například koule, válec, letadlo, vrtulník). Stačí přidat potomka třídy *TRigidBody* a doprogramovat příslušné abstraktní funkce.

6.1 Další budoucnost programu

Program by mohl být vylepšen a rozšířen o mnoho nových vlastností:

- Optimalizace použitých algoritmů.
- Přidání dalších typů těles.
- Možnost spojovat tělesa do celku a vytvářet tak složitější tělesa.

- Implementace simulace vody.
- Možnost uložit do souboru aktuální scénu a následně ji nahrát.
- Rozšíření možností ovládání.
- Oddělení vykreslovaného modelu tělesa a kolizního primitiva.
- Vylepšení grafického znázornění, zejména použitím textur.

Literatura

[1] Bridson R., Fedkiw R., Guendelman E.: *Nonconvex Rigid Bodies with Stacking, ACM Transactions on Graphics (Proc. SIGGRAPH 2003)*

[2] Grafická knihovna OpenGL: http://www.opengl.org

[3] API GLUT 3.7 (2000): http://www.opengl.org/resources/libraries/glut/

[4] GUI knihovna GLUI 2.35 (2006): http://www.cs.unc.edu/~rademach/glui/

[5] Fiedler G.: Integration Basics http://www.gaffer.org/game-physics/integration-basics

[6] Hacker C.: Physics, The Next Frontier, *Game Developer Magazine Oct/Nov 96*

[7] Chapman D.: Jiggle - http://www.rowlhouse.co.uk/jiggle/

[8] Separating Axis Theorem: http://en.wikipedia.org/wiki/Separating_axis_theorem

[9] SAP (Sweep and Prune): http://bulletphysics.com/Bullet/phpBB3/download.php?id=12

[10] Sphere trees: https://www.cs.tcd.ie/publications/tech-reports/reports.99/TCD-CS-1999-10.pdf

[11] AABB trees: http://www.win.tue.nl/~gino/solid/jgt98deform.ps

[12] BSP trees: http://www.inf.ufrgs.br/~comba/papers/2005/sabsp-i3d05.pdf

[13] Octree: http://en.wikipedia.org/wiki/Octree

[14] Eulerova metoda numerické integrace:
http://en.wikipedia.org/wiki/Euler's_method

[15] Numerická integrace RK4 (Rugne-Kutta čtvrtého řádu):
http://en.wikipedia.org/wiki/Runge-Kutta_method

[16] Valve, Half-life 2 (2004): http://orange.half-life2.com/hl2.html

Přílohy

Obsah přiloženého CD:

- Kopie této práce ve formátu PDF a .odt (OpenOffice).
- Spustitelná verze programu Virtuální hřiště zabalená v souboru „*Virtual Playground executable.zip*“.
- Zdrojové kódy a projekt připravený pro kompilaci v prostředí Microsoft Visual Studio 8 v souboru „*Virtual Playground source.zip*“.