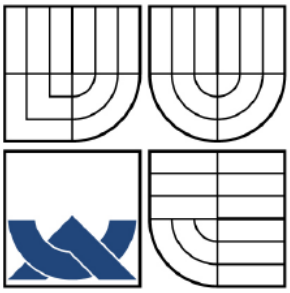


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH AUTOMATICKÉHO GENERÁTORU PROSTŘEDÍ PRO MOBILNÍ ROBOT

AUTOMATIC ENVIRONMENT GENERATOR DESIGN FOR MOBILE ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR SCHREIBER
AUTHOR

VEDOUCÍ PRÁCE — Ing. STANISLAV VĚCHET, Ph.D.
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2007/08

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Schreiber Petr

který/která studuje v **bakalářském studijním programu**

obor: **Aplikovaná informatika a řízení (3902R001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh automatického generátoru prostředí pro mobilní robot

v anglickém jazyce:

Automatic environment generator design for mobile robot

Stručná charakteristika problematiky úkolu:

Cílem práce je návrh a realizace automatického generátoru prostředí pro mobilní robot. Generátor má vygenerovat kompletní virtuální prostředí, ve kterém se následně pohybuje autonomní robot. Virtuální prostředí bude uloženo ve zvoleném formátu pro pozdější potřebu simulací. Simulace budou probíhat v prostředí Open-Dynamic-Engine (ODE), které poskytuje široké možnosti pro simulace umělých světů, které jsou vhodné pro simulační ověřování navigačních metod v robotice.

Cíle bakalářské práce:

Prostudujte problematiku simulací pomocí ODE
Navrhněte strukturu formátu pro ukládání virtuálního prostředí
Navrhněte strukturu generátoru prostředí
Generátor prostředí realizujte
Ověřte funkci generátoru prostředí v součinnosti s dalšími importovanými dynamickými prvky (robot)

Seznam odborné literatury:

Kosei D.: Robot programming with Open Dynamics Engine, Morikita Publishing Co, Ltd., Tokyo, 2007, ISBN:978-4627846913

www.ode.org

Vedoucí bakalářské práce: Ing. Stanislav Věchet, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2007/08.

V Brně, dne 15. 01. 2008

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ DĚKANÁT

Šeda

doc. RNDr. Ing. Miloš Šeda, Ph.D.
Ředitel ústavu

v.z. K. Kuběnka

doc. RNDr. Miroslav Doupovec, CSc.
Děkan fakulty

LICENČNÍ SMLOUVA

(na tomto místě je vloženo licenční ujednání)

ABSTRAKT

Tato práce se zabývá problematikou návrhu a realizace automatického generátoru virtuálního prostředí pro účely simulace autonomního robotu. Autor popisuje různé přístupy k implementaci generátoru interiéru a jejich vazbu na knihovnu Open Dynamics Engine a formát XODE. Součástí práce je i ověření funkce generátoru v součinnosti s virtuálním mobilním robotem.

ABSTRACT

This work deals with design and realization of automatic environment generator, serving for autonomous robot simulations. The author describes various approaches to implementation of interior environment generator and their relation to Open Dynamics Engine physics library and XODE file format. This work also includes verification of generator functionality with participation of virtual mobile robot.

KLÍČOVÁ SLOVA

Simulace, generátor, interiér, Open Dynamics Engine

KEYWORDS

Simulation, generator, interior, Open Dynamics Engine

PODĚKOVÁNÍ

Za pomoc s vypracováním práce, připomínky a zvláště pravidelné konzultace děkuji Ing. Stanislavu Věchetovi, PhD a Ing. Vítu Ondrouškovi.

Dále bych chtěl poděkovat Jaroslavu Miklendovi, Richardu Serišovi a Petru Zetkovi za týmovou spolupráci při zkoumání Open Dynamics Engine.

OBSAH

ČÁST A – TEORETICKÁ ČÁST

1 ÚVOD

Návrh robotu a jeho testování je myšlenkově, finančně i časově náročnou činností. Proto je třeba v této oblasti hledat nové postupy, vedoucí ke snížení vlivu zmíněných kritických faktorů.

Počítačový návrh produktů proniká do všech oblastí lidské tvorby. Je logické, že se tato metoda ukazuje jako vhodná i pro návrh mechanismů a robotů.

V reálném světě je možné testovat schopnosti robotu umístěním do libovolného prostředí jako např. místnost, parkoviště a další. Přestože se tento postup jeví jako velice vhodný, má několik zásadních nevýhod.

Vyžaduje fyzickou manipulaci s robotem, hlavně pak časově náročné cestování na místo, kde má být proveden test. V případě, že na místě zjistíme nedostatky v chování robotu, nezbývá než se vrátit zpět a odstranit je, případně se pokusit o přeprogramování na místě pomocí přenosného počítače, což vyžaduje přítomnost zdroje elektřiny či dobrou výdrž baterie.

Další komplikací je nebezpečí poškození robotu. V etapě návrhu mechanismu se často uchylujeme k odlehčeným, nekapotovaným konstrukcím, které mohou být poškozeny vlivem počasí či nárazem.

Ze zmíněných důvodů vychází metoda počítačové simulace jako časově nenáročné a bezpečné řešení. V případě dobrých výsledků virtuálních testů robotu je pak možno přistoupit k jeho fyzické konstrukci s mnohem menším rizikem neúspěchu. S výhodou lze tohoto postupu využít i při simulaci již existujícího stroje.

Faktem je, že i při pokusu o zavedení počítačového návrhu mobilního robotu se člověk setkává s řadou překážek. První je samotný způsob reprezentace stroje a jeho chování. Přesná simulace robotu v plném rozsahu vyžaduje jak důkladnou analýzu mechanické podstaty robotu, tak i případné sledování jeho chování, chceme-li simulovat již existující zařízení. Takto získané informace je pak možno přenést do simulačního modelu.

Dobrou zprávou je, že tvorba fyzikálních simulací je dnes mnohem jednodušší než před deseti lety. Po roce 2000 se objevilo velké množství freeware či dokonce Open Source knihoven, které umožňují minimálně práci s pevnými tělesy. Zatímco počítání dynamiky je velmi jednoduché i bez těchto knihoven, detekce kolizí již nikoliv.

Sestavením virtuálního robotu však práce nekončí, je třeba pozorovat jeho reakce při pohybu ve virtuálním prostředí. Mobilní robot se většinou sestává z několika desítek součástí, prostředí ale může nabývat mnohem komplexnějších tvarů. V případě lidských výtvorů, jako jsou interiéry budov, jde většinou o relativně jednoduché a pravidelné objekty. Složitější situace nastává u venkovních prostor, zvláště pak terénních nerovností a rostlin.

Pokud by měl člověk toto prostředí tvořit sám, pomocí plnohodnotných modelovacích programů, musel by čas, který za těchto podmínek nebylo třeba věnovat cestování, nahradit dobou strávenou u počítače při pečlivé tvorbě testovací lokace.

Na následujících stranách budete seznámeni s postupy, které umožní jednoduše a rychle vytvářet komplexní virtuální prostředí s možností parametrizace výstupu. Poslední kapitola je věnována umístění robotu do připraveného prostředí, a provedení kompletních testů přímo na PC.

2 CÍLE PRÁCE

Zadaná úloha klade několik požadavků, které je nutno splnit. Těmi jsou:

- Nastudování problematiky simulací pomocí Open Dynamics Engine
- Volba formátu pro ukládání prostředí
- Návrh a realizace automatického generátoru prostředí
- Ověření funkce generátoru prostředí zavedením dynamického prvku - robotu

Open Dynamics Engine (dále ODE) je jednou z knihoven, navrženou pro výpočty simulací dynamiky a kolize pevných těles, proto je v dalším textu porovnána s konkurenčními produkty. Z volby simulační knihovny vyplývá i výběr vhodného formátu souboru, který by mohl data připravená pro zpracování uchovat.

Těžištěm této práce je vyhotovení řešení, které by zajistilo časově nenáročnou tvorbu lokací, a které v budoucnu poslouží pro testy lokalizace robotů VUT. Součástí práce je i ověření možnosti importovat mobilní robot do vytvořeného prostředí.

3 SOUČASNÝ STAV TECHNOLOGIÍ

Cílů nastíněných v předchozí kapitole je možno dosáhnout několika cestami. Rozsah této práce vyžaduje použití různých systémů, od simulační knihovny přes programovací prostředí až k vizualizačním prvkům.

Současná nabídka software je naštěstí poměrně široká. Řešení, která byla dříve dostupná jen obtížně nebo za cenu velkých nákladů, jsou postupně více či méně úspěšně doplňována nástroji nezatíženými komplikovanými licenčními podmínkami. Proto se následující přehled věnuje jak tradičním, tak i relativně novým aplikacím a knihovnám, jejichž použití by vypracování úkolu usnadnilo.

3.1 Řešení pro výpočet fyzikální simulace

Simulace dynamicky stabilní chůze robotu a počítání jeho interakcí s okolím je ve většině případů dost náročné, proto bylo rozhodnuto o použití knihovny třetí strany. Na tu byly kladeny následující požadavky:

- Počítání dynamiky pevných těles (možnost aplikovat síly a momenty)
- Definice vazeb
- Zpracování detekce kolizí
- Formát souboru, ze kterého by bylo možné data načítat

Jak již bylo zmíněno v úvodu, dostupnost takovýchto řešení je nyní poměrně bezproblémová a proto bylo možné si vybrat. Většina zmíněných modulů byla původně orientována na ne zcela nutně přesné použití v počítačových a konzolových hrách, posledních 5 let lze však již sledovat trend postupného zpřesňování.

Zadáním byl doporučen Open Dynamics Engine, ten však není jediným možným prostředkem pro fyzikální výpočty. V dalších podkapitolách proto budou představeny i jiné produkty, které by bylo možno použít.

Bohužel stejný trend již nelze sledovat v oblasti ukládání simulačních dat. Většina programátorů se uchyluje k tvorbě vlastního formátu souboru a obecných řešení bohužel není mnoho.

Drtivá většina pak překvapivě spoléhá na dokumenty vycházející ze schématu XML. To je výhodné pro svou modulární strukturu a podobnost popisu jazykem HTML, se kterým má většina lidí již nějakou zkušenost. Velkou nevýhodu však představují extrémní nároky na diskový prostor, které díky jednoduché nekomprimované podobě rychle rostou s množstvím uložených dat.

3.1.1 Havok Physics

Jednou z prvních vlaštovek v oblasti fyzikálních výpočtů v reálném čase se stal pokročilý systém Havok Physics. Ten byl až do května 2008 dostupný pouze pod drahou komerční licencí. Vzhledem k tomu, že od zmíněného data je postupně uvolňován i pro nekomerční aplikace zdarma, je vhodné jej zmínit podrobněji, přestože pro tuto práci již z časových důvodů nemohl být použit.

Obr. 1 *Logo fy Havok.[1]*

Havok Physics nabízí stabilní počítání dynamiky i detekce kolizí, které bylo odladěno v průběhu let v mnoha (zejména herních) aplikacích. Knihovna umožňuje používat všechna standardní geometrická primitiva, jako jsou krychle, válec a další. Navíc přidává i libovolnou geometrii reprezentovanou polem trojúhelníků.

Velmi zajímavá je možnost integrovat tento systém s profesionálními grafickými programy jako jsou Autodesk 3Ds Max, Maya či Avid Softimage XSI, a provádět tak návrh těles a jejich vazeb vizuálně. Nevýhodou je ovšem vysoká pořizovací cena zmiňovaných nástrojů.

Z dalších pozoruhodných schopností je třeba zmínit možnost provádění paralelních výpočtů a integraci s dalšími produkty firmy Havok, které nabízejí i pokročilejší funkce jako jsou simulace látek a rozbíjení těles.

Na první pohled velmi propracovaný nástroj má bohužel i několik nevýhod. Původně podporoval počítání fyziky i na 3D akcelerátorech ATi a NVIDIA, po odkoupení firmou Intel se však o tomto řešení spíše mlčí. Havok tak bude patrně do budoucna více sledovat zájmy svého nového majitele.

3.1.2 Ageia PhysX a NVDIA CUDA

Velmi zajímavé je API[1] firmy Ageia, které nabízí obdobné funkce jako předchozí řešení, firma však podpořila výkon i vyvinutím speciálního hardware, karty s PPU (z angl. Physics Processing Unit). Ta je volitelná a není nutné ji mít v PC pro spuštění programu s podporou PhysX.

Obr. 2 *Vzhled loga Ageia PhysX před převzetím firmou NVIDIA.* [2]

Pro získání dokumentace a potřebných knihoven je nutné se zaregistrovat na stránkách výrobce. Podobně jako Havok, i Ageia byla odkoupena velkým výrobcem hardware, a to firmou NVIDIA. Je tedy pravděpodobné, že funkčnost fyzikálních akcelerátorů bude přenesena na grafické karty této firmy. Bude zajímavé sledovat, nakolik bude dodržen slib o zpětné kompatibilitě.

[1] Z angl. *Application Programming Interface*, tedy rozhraní pro programování aplikací

Oficiální materiály naznačují, že bude možné použít 2 karty NVIDIA stejného typu, přičemž jedna bude počítat grafický výstup a druhá fungovat jako fyzikální akcelerátor. Teoreticky by se mohla objevit i karta se dvěma čipy na jedné desce, což by přineslo zjednodušení prostorově i z hlediska napájení složitého modelu dvou karet. PhysX jako takový se pak stane součástí technologie NVIDIA CUDA, která slouží k použití grafických karet pro obecné, nejen grafické výpočty.

Tento krok je přijímán odbornou veřejností jak pozitivně tak i s četnými výhradami. John Carmack, známý průkopník v oblasti 3D aplikací, vyjádřil o výhodnosti sloučení CUDA a PhysX velké pochybnosti [3].

Přestože je toto řešení nováčkem na poli fyzikálních simulací, navzdory jednostranně negativním kritikám zejména v odborných i herních časopisech, je PhysX nasazováno ve velkém množství aplikací a her, kde se v počtu instalací již blíží zavedenému Havok Physics.

Vysoké číslo je možná dáno díky faktu, že Ageia staví na základech staršího simulačního software Novodex.

3.1.3 Newton Game Dynamics

Tato knihovna je vyvíjena od roku 2003. Dle výrobce je určena pro počítání fyziky v reálném čase nejen pro herní aplikace. Obsahuje unikátní deterministickou metodu pro výpočty a tudíž je dle autorů vhodná i pro použití v aplikacích vyžadujících vysokou přesnost.

Obr. 3 *Logo Newton Game Dynamics.* [4]

Knihovna je distribuována bez zdrojových kódů, a proto není možné správnost způsobu výpočtu nijak posoudit. Systém Newton Game Dynamics byl použit v řadě freeware produktů i několika komerčních programech.

3.1.4 Open Dynamics Engine

ODE je fyzikální systém pro zpracování pevných těles, vazeb a detekci kolizí. Je udržován ve vývoji poměrně aktivní komunitou již od počátku 21. století. Je poměrně běžně nasazován v simulačních aplikacích od čtyřkolých vozidel[6] přes letadla[7] až po aplikace v robotice[8].

Obr. 4 *Logo Open Dynamics Engine.* [5]

Knihovna je též hojně integrována do víceúčelových enginů jako jsou Ogre3D a Irrlicht, a tudíž je její použití dlouhodobě ověřeno v praxi.

Výhodou je existence souborového formátu XODE, který slouží pro ukládání objektů pro simulaci. Přestože tento formát není oficiálním řešením, byl vyvinut firmou TankSoftware[9] a je užíván komunitou okolo Open Dynamics Engine.

3.2 Knihovny pro 3D grafiku

Díky prudkému vývoji v oblasti hardwarově akcelerované grafiky je dnes možné na běžných PC provádět názorné vizualizace i velice komplexních scén. Vykreslování probíhá díky optimalizovaným funkcím dnešních 3D karet, a to jak těch integrovaných, tak i externích akcelerátorů do AGP či PCI-Express slotů.

Všechny dnešní běžně dostupné karty používají metodu rasterizace. To znamená, že promítají vektorová data do bitmapy či na obrazovku počítače. Pro každý pixel se ukládá hloubka a na základě relativně jednoduchých testů tak lze skládat výsledný obraz. Podrobný popis této metody je mimo rozsah této práce, více informací lze nalézt ve zdrojích [10][11][12].

Firma Intel již dlouho ohlašuje vývoj vlastního akcelerátoru, který funguje na zcela jiném principu – raytracingu. Zatím však nejsou k dispozici žádné testovací vzorky, proto na tuto možnost nebyl brán ohled. Raytracing[2] počítaný CPU je i dnes velmi pomalý, a pro zobrazení simulace v reálném čase se tak nehodí. Zmiňované řešení Intelu, „Larabee“[13], však bude podporovat i rasterizační API, proto se zdá využití rasterizace jako vhodné.

Chceme-li využít 3D akcelerace, musíme použít API, které nám tyto funkce zpřístupní. Následující přehled představuje dva v současnosti nejpoužívanější systémy. Kdysi oblíbené Glide3D pro karty firmy 3Dfx je dnes již zcela nepoužitelné, neexistuje nový hardware, který by pro něj měl ovladač, a proto nebylo bráno v potaz.

3.2.1 Direct3D

Tento produkt firmy Microsoft je velmi populární v herním průmyslu, jako nejpoužívanější část balíku DirectX. Umožňuje plnohodnotné využití možností grafických karet, včetně programování grafického výstupu pomocí všech typů shaderů[3].

Obr. 5 *Logo DirectX 9.* [14]

Direct3D je v drtivé většině používán pro „nevážné“ použití zejména v herním průmyslu. Existují však i výjimky, názorným příkladem je projekt 3D NURBS modeláře MoI[15].

Pomocí tohoto API lze renderovat libovolná grafická primitiva, přiřadit jim barvu, textury a další vlastnosti. Nasvětlení je realizováno pomocí per-vertex přístupu, který pro běžné použití stačí, pro kvalitní vizualizace je však vhodnější nahradit fragment shaderem který operuje na úrovni pixelů a světlo je tak rozloženo mnohem plynuleji.

[2] Raytracing je doslova “metodou sledování paprsku”, dosud tento postup nacházel uplatnění hlavně ve fotorealistickém vykreslování statických scén
[3] *Shader* je program prováděný přímo na grafické kartě.

Jistou výhodou je bezesporu načítání vlastního formátu „X“, který je exportován několika modelovacími aplikacemi. Tento formát má však velmi stručnou dokumentaci a tak je třeba spoléhat se na řešení exportu od již zavedených firem.

Nevýhodou tohoto API je omezení poslední verze 10 pro běh pouze pod systémem MS Windows Vista. Starší verze 9 (viz obr. 5) je pak možno provozovat na MS Windows XP, MS Windows Mobile a herní konzoli XBOX 360.

3.2.2 OpenGL

Tato knihovna je nástupcem řešení IrisGL původně vyvíjeném firmou Sillicon Graphics. V druhé polovině devadesátých let vývoj přešel pod křídla OpenGL Architecture Review Board a nyní je API udržováno společností Khronos Group.

Obr. 6 *Logo OpenGL.* [16]

OpenGL ve své poslední verzi 2.1 nabízí stejné možnosti jako Direct3D, což je dáno do značné míry faktem, že obě API pracují nad stejným typem hardware. Shodné je i omezení současně zapnutých světelných zdrojů na 8. OpenGL má však oproti svému konkurentovi výhodu v širší podpoře operačních systémů a zařízení. Navíc je snadněji rozšiřitelné díky mechanismu extenzí, takže každý výrobce hardware může předložit nové vlastnosti hardware vývojářům mnohem jednodušeji a nezávisle na výrobci operačního systému.

Návrh syntaxe tohoto rozhraní je dobře odstupňován tak, že začátečník může rychle začít tvořit vlastní aplikace a postupně využívat složitějších instrukcí pro vylepšování výkonu.

Zajímavá je zejména možnost provozovat OpenGL aplikace na libovolných MS Windows či systémech Linux. Pod MS Windows je OpenGL dostupné ve verzi 1.1 už v základní instalaci, jedná se však o nepříliš kvalitní implementaci, a proto je vhodné nainstalovat ovladač grafické karty, který správu OpenGL převezme. Většina moderních karet tak podporuje nejnovější standard 2.1, výjimkou jsou karty Intel, které nabízejí OpenGL 1.3 – 1.5.

OpenGL nemá vlastní formát souboru pro geometrii, ale díky nekomplikované interní struktuře je možné použít vlastní rutiny pro načítání dat ze standardních formátů souborů jako je OBJ a další. [16]

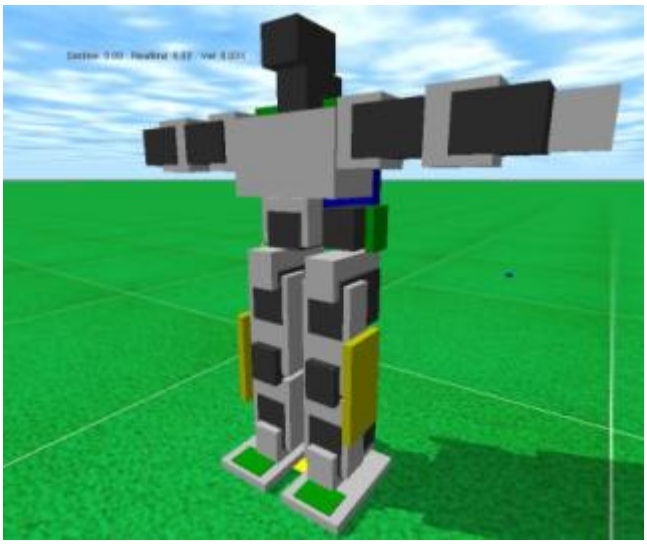
4 POUŽITÁ ŘEŠENÍ

Pro zpracování projektu byla vybrána vždy jedna technologie z příslušných oblastí, zmíněných v minulé kapitole. Při výběru byla upřednostňována levnější řešení a systémy, se kterými již autor měl nějakou zkušenost.

4.1 Volba fyzikální knihovny

Velice nesnadný byl výběr fyzikálního engine, a to hlavně díky velmi bouřlivým změnám, které ve světě simulací během vyhotovování práce probíhaly. Na samém počátku byl Open Dynamics Engine zcela jasnou volbou. Díky své otevřené koncepci, dlouhodobému používání a snadné dostupnosti měl v podstatě jediného konkurenta v podobě řešení Ageia PhysX.

Nepříliš pozitivní recenze PhysX v tištěných i online magazínech však od tohoto API zrazovaly. Při vědecké práci je však třeba být i trochu nedůvěřivý, zvláště vůči extrémně pozitivním či negativním reakcím, které se v tomto případě vyskytovaly. Na rozdíl od ODE, získání PhysX SDK vyžaduje registraci, která však nemusí nutně vyústit v získání SDK. Tento přístup je patrně běžný v oblasti komerčních řešení, přesto však zanechává poněkud negativní dojem, že daná firma spíše než o co největší rozšíření svého řešení usiluje o „lukrativní“ uživatele. Po úspěšné registraci je již možno stáhnout SDK, bohužel dokumentace je velice kusá, což se překvapivě ukázalo jako typický rys většiny fyzikálních knihoven.



Obr. 7 *Simulace humanoidního robotu za použití ODE.*[8]

Volba tedy padla na Open Dynamics Engine, který nabízí své API pro libovolné použití pod operačními systémy MS Windows a Linux. Navíc je ODE kromě počítačových her již v zahraničí používáno i pro simulace robotů(viz obr. 7). Dokumentace základních funkcí je mírně rozporuplná – kompletní manuál existuje pouze pro starší verzi. Oficiálně je podporována i dedikovaná Wiki aktivně udržována komunitou, která již nabízí ucelený pohled i na nejnovější vlastnosti.

Více se o této knihovně dozvíte v kapitole 5.

4.2 Grafická reprezentace simulace

Volba grafické knihovny pro vizualizaci výstupu simulace již byla o něco méně komplikovaná. S ohledem na budoucí rozvoj fyzikálních simulací na VUT a bezproblémovost portování grafického API na vyšší verze bylo zvoleno OpenGL, se kterým má autor již delší, pozitivní zkušenost. Direct3D od firmy Microsoft je taktéž velmi schopný nástroj, bohužel však s každou verzí prochází několika změnami, které udržování aplikace dost komplikují. Omezení Direct3D 10 na provoz pod Windows Vista je poměrně typickou ukázkou takového přístupu. Použití staršího Direct3D 9 by bylo možné, ovšem nenabízelo by žádné podstatné výhody.

OpenGL je dnes již standardně akcelerováno na grafických kartách nejrozšířenějších výrobců, jakými jsou ATi, NVIDIA a Intel. Při konstrukci vizualizační složky byl brán ohled na co nejširší kompatibilitu s různým grafickým hardware. OpenGL je také upřednostňováno pro drtivou většinu CAD, modelovacích a vizualizačních programů, což pouze potvrzuje vhodnost této volby.

Dokumentace OpenGL je snadno dostupná, jak v knižní podobě[17] (a to dokonce česky) tak i ve formě nesčetných tutoriálů na internetu [18][19].

K dispozici je i online referenční příručka s podrobným popisem programového rozhraní. To spolu s kompletní specifikací umožňuje orientovat se v dané knihovně bez velkých problémů velice rychle.

Výhodou akcelerovaných řešení je, při dodržení doporučovaných postupů, jejich minimální požadavek na rychlost CPU, které pak může většinu svého výkonu použít na provádění výpočtů. Z tohoto důvodu bylo již předem vyloučeno jakékoli „softwarové“ vykreslování na bázi GDI či GDI+, které by zbytečně brzdilo chod simulace.

Pro tvorbu a správu OpenGL oken byl použit starší systém GLUT, který ač již není nejmodernějším řešením, stále poskytuje dostatečnou funkčnost a hlavně dostupnost napříč různými operačními systémy.

4.3 Programovací jazyk Python

Python je programovací jazyk, který byl vyvinut v 80. letech minulého století, masového rozšíření však nabyl až během konce 90. let. Jedná se o jazyk objektově orientovaný, multiplatformní a interpretovaný, s kompilací do byte kódu.

Obr. 8 *Logo programovacího jazyka Python.* [18]

Python je zaměřen na vysokou produktivitu programování, ve smyslu snadné a rychlé tvorby programů, nejedná se tedy o primárně silové řešení určené pro tvorbu vysoce optimalizovaných či kompaktních aplikací jako je tomu v případě jazyků C, Delphi, PowerBASIC, MASM a dalších.

Programátora Python například nenutí deklarovat typ proměnných. Do těch lze přiřazovat cokoli dle situace, což se trochu podobá datovému typu VARIANT z jiných programovacích jazyků.

Hlavní předností Pythonu je možnost provozovat jej jak na Windows XP tak i různých distribucích Linuxu. Tato univerzálnost však znamená i ne zcela plné využití možností těchto dvou odlišných platforem, což je zejména patrné na správě oken.

Vybrané fyzikální a grafické technologie jsou do Pythonu implementovány pomocí modulů. V případě OpenGL je syntaxe většinou shodná s tou použitou v jiných jazycích, funkce standardně vracející hodnoty odkazem či ukazatelem však mají zpravidla pozměněnou podobu.

Procedurální ODE je zpřístupněno pod objektovým rozhraním, pomocí modulu PyODE. To přináší mnoho výhod i některá úskalí, jako je například odlišnost názvů procedur a eliminace handlů na jednotlivé elementy simulace, což mírně komplikuje přenos kódu z jiných jazyků, které nejčastěji používají ono procedurální rozhraní.

Dokumentace jazyka samotného je na dobré úrovni, v České Republice vyšly již tři knihy [21][22][23], které popisují základy jazyka, na internetu je pak možno najít příručky v angličtině, španělštině a dalších světových jazycích.

Horší situace nastává s popisem PyODE. Dokumentaci představují 3 názorné tutoriály, ale samotný popis funkcí je již velmi slabý. Běžné je i shrnutí funkčnosti dané metody jedinou větou, což je zčásti způsobeno generováním dokumentace z komentářů ve zdrojovém kódu. To však nelze brát jako omluvu. Při současném studiu ze zdrojů originálního ODE a verze pro jazyk Python však již lze aplikace s jistou mírou úsilí vytvářet. Jistým problémem je vyčerpanost vývojového týmu, který již nemá s modulem větší plány, zato je ochoten předat je do rukou jiného vývojáře pod Open Source licencí[14].

Volba tohoto jazyka umožnila i návaznost na další bakalářské projekty, které se zabývají regulací dvoukolého robotu, chůzí šestinohého mechanismu či regulací kuličky na houpačce.

5 ARCHITEKTURA OPEN DYNAMICS ENGINE

Open Dynamics Engine je knihovna pro simulaci pevných těles. Fyzikálně aktivní těleso se v ODE skládá ze dvou celků, proměnných typu *body* a *geom*. Typ body sám o sobě je v podstatě hmotným bodem, který je charakterizován svou pozicí, hmotností a silovými účinky. Typ geom reprezentuje tvar tělesa. Spojením proměnných typu body a geom tak získáváme objekt, který má fyzikální vlastnosti i tvar, který je používán při detekci kolizí. Pro objekt reprezentovaný pomocí body a geom budu pro zjednodušení v dalším textu této kapitoly používat označení „těleso“.

Na těleso je možné působit silovými účinky a momenty, případně nastavovat přímo vektory rychlostí a podobně. Aktuální pozici a natočení objektu lze získat ve formě matice, která je pak přímo použitelná pro vizualizace pomocí OpenGL, ale i jinými systémy.

Zatímco použití body samotného nemá pro složitější simulace význam, geom může existovat samostatně, a sloužit tak jako pevná překážka pro pohybující se tělesa.

Tělesa mohou být navzájem propojena různými typy vazeb, nejpoužívanější jsou vedena v následujícím seznamu:

- Sférická vazba
- Rotační vazba
- Vetknutí
- Posuvná vazba

Přítomnost vazeb umožňuje sestavovat virtuální modely i poměrně složitých konstrukcí, jako jsou roboty, automobily či různé mechanismy.

5.1 Způsob simulace

Simulace chování těles je v dané knihovně realizována integrací po časových krocích. Ta je prováděna dvěma způsoby – přesným způsobem nebo zjednodušeným způsobem (tzv. rychlý krok). Druhý způsob je díky nižší přesnosti pro účely simulace zcela nevhodný, a jeho použití je tak možné hlavně v počítačových hrách či jiných aplikacích, kde není kladen důraz na korektnost výsledku.

Integrace může být prováděna po libovolných časových krocích, autoři však doporučují konstantní velikost kroku. Stabilita simulace roste s klesající velikostí simulačního kroku, zvláště v případě použití jeho konstantní velikosti.

Chování simulace je možno upravit mimo jiné i nastavením specifických parametrů, které ošetřují chování vazeb. Tyto parametry lze většinou nastavit jak globálně, tak i pro každou vazbu zvlášť. Dohromady slouží k exaktnímu nastavení chování vazeb

5.1.1 ERP

Error Reduction Parameter je opravný koeficient, který se aplikuje v případě silového působení ve vazbě. Tato hodnota může nabývat hodnot v intervalu <0, 1>, ovšem mezní hodnoty není autory doporučeno používat. Je-li ERP nastaven na hodnotu 0, není prováděna žádna korekce. S rostoucí hodnotou je na vazbu vyvíjeno dodatečné silové působení přímo úměrné velikosti koeficientu. [24]

5.1.2 CFM

Constraint Force Mixing je koeficient, který upravuje pevnost vazeb. Jeho nastavení má zásadní vliv na stabilitu simulace. Nastavení hodnoty parametru na nulu má za následek vytvoření ideálně pevné vazby či omezení, díky jisté míře nepřesnosti uvnitř ODE však tento efekt není zcela spolehlivý. Nastavení vyšší hodnoty CFM má pozitivní vliv na stabilitu simulace, představuje však i zvýšenou míru tolerance pohybu v rámci vazby. [24]

5.2 Detekce kolizí

ODE nabízí možnost integrace dynamické simulace a počítání detekce kolizí jak pomocí interního řešiče, tak i externí knihovnou. V dalším textu je předpokládána možnost první.

Na detekci kolizí v ODE mají vliv jak již zmíněné parametry CFM a ERP, tak i některé další koeficienty. Těmi jsou *bounciness* a *Mu*. První z nich určuje, do jaké míry se budou tělesa navzájem odrážet. Druhý parametr již má přesnější fyzikální význam, a to koeficient tření, který se aplikuje, dojde li k doteku dvou a více těles.

Zajímavostí ODE je jeho způsob výběru kolizních bodů. Ten je svým způsobem nedeterministický. Například při kontaktu dvou ploch si knihovna náhodně vybere kolizní body (kterých by v reálném světě bylo nekonečno) a s nimi dále pracuje. Toto chování je probráno v článku[26].

Dojde-li ke kolizi těles, ODE automaticky vyvolá programátorem definovanou tzv. *callback* funkci, která může chování při doteku těles upravit v závislosti na typu objektů či jiných podmínkách.

5.3 Přístup pro získání vhodného nastavení simulace

Nastavení Open Dynamics Engine tak, aby co nejvíce odpovídal potřebám simulace, je poměrně komplikované. Ve většině případů je nutné provedení experimentů na jednodušších systémech.

Získání co nejvhodnějších parametrů bylo provedeno při spolupráci s kolegy, jejichž práce se Open Dynamics Engine také týkala. Na doporučení školitele bylo rozhodnuto o použití krychle na nakloněné rovině, jako modelové situace. Výhodou tohoto přístupu je jeho snadná ověřitelnost. Optimální nastavení parametrů simulace pomocí ODE bylo prováděno na problému pohybu tělesa po nakloněné rovině. Tato úloha byla spočtena analyticky a simulována pomocí toolboxu MatLabu Simulink, v součinnosti s [31][32][33].

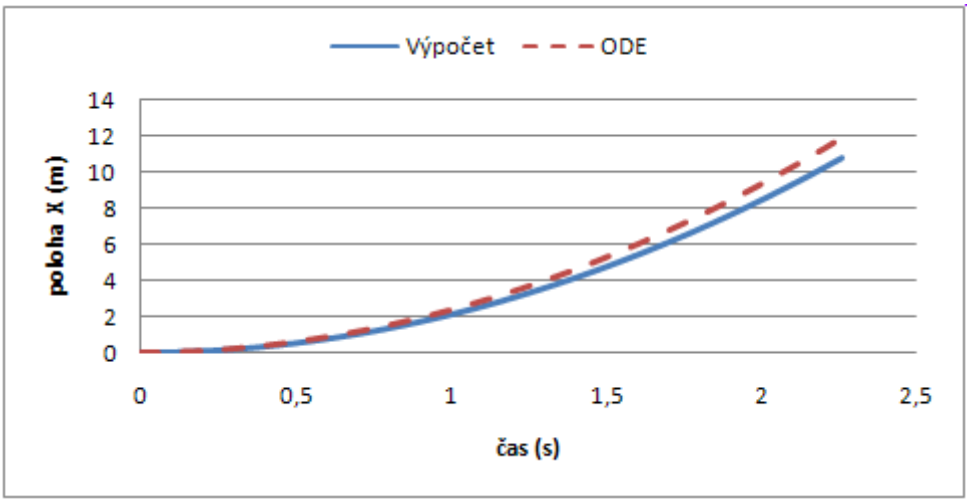
Při tomto pokusu jsme dospěli k následujícímu nastavení Open Dynamics Engine:

- ERP 0.1
- CFM 10^{-5}
- Bounciness 0.2
- Mu 0.1

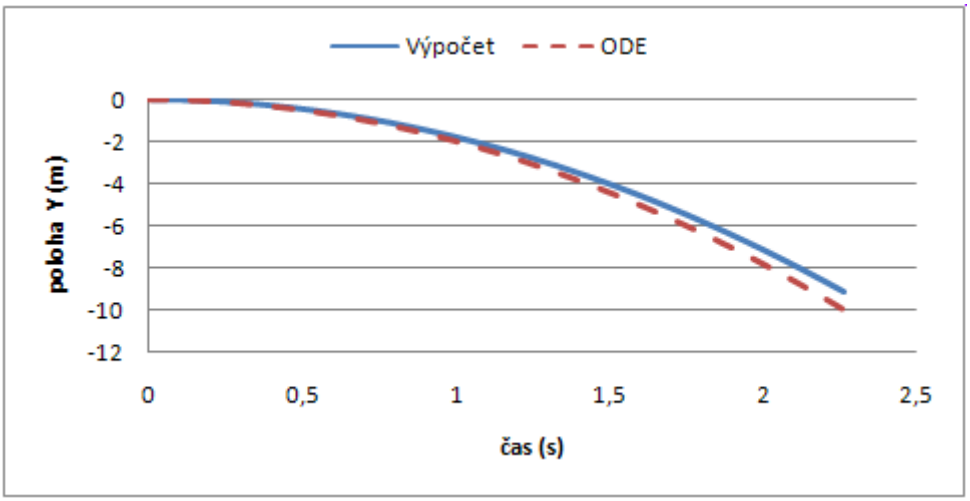
Koeficient tření odpovídá kontaktu ocel-ocel[25], ostatní parametry byly získány empiricky tak, aby se výsledek v ODE co nejvíce přiblížil výsledku vypočtenému ostatními postupy. Z obr. 9, obr. 10 a obr. 11 je patrné srovnání dosažených výsledků simulace zmíněného elementárního problému pomocí ODE a analytického výpočtu pro výše uvedené parametry simulace.

Jak je patrné z grafů, k plné shodě výsledků nedošlo. To je však dáno faktem, že analytický model nepočítal s možností pohybu tělesa v jiném směru než rovnoběžném s nakloněnou rovinou.

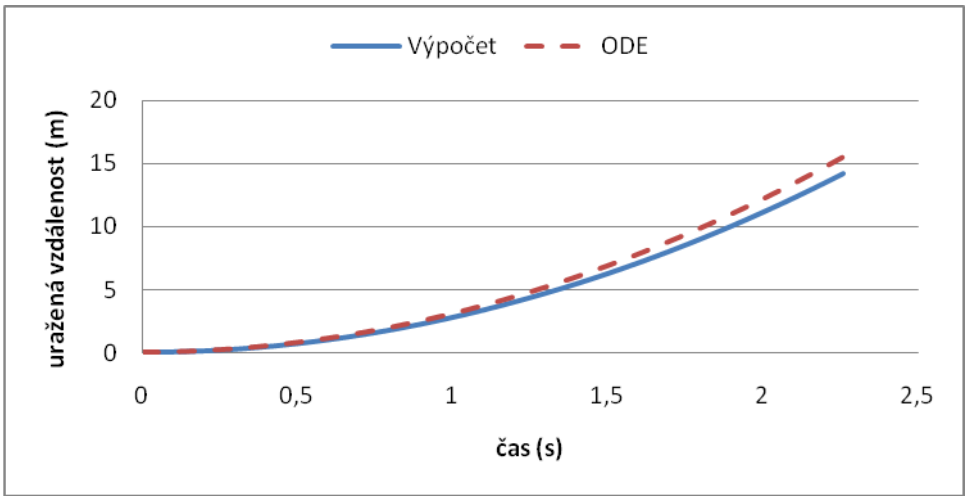
Zjištěné hodnoty byly později využity při testu importu robotu do prostředí vytvořeného generátorem. Veškeré výpočty Open Dynamics Engine byly prováděny prostřednictvím modulu PyODE 1.2, který využívá ODE ve verzi 0.8, v nastavení přesnosti „single“.



Obr. 9 Průběh polohy na ose X pro těleso na nakloněné rovině (40°)



Obr. 10 Průběh polohy na ose Y pro těleso na nakloněné rovině (40°)



Obr. 11 Průběh uražené dráhy pro těleso na nakloněné rovině (40°)

6 FORMÁT XODE

Pro ukládání zdrojových dat simulace byl zvolen XML formát XODE, a to hlavně pro svou schopnost přesně reprezentovat vnitřní strukturu objektů Open Dynamics Engine. Pro lepší představu zde uvádím příklad souboru v souladu s poslední specifikací[9], který definuje krychli o hraně 1 metr a hustotě 1000 $kg.m^{-3}$:

```
<?xml version="1.0" encoding="UTF-8"?>

<xode version="1.0" name="VzorovaKrychle"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://tankammo.com/xode/1.0r17/xo
       de.xsd">

    <world transformstyle="vector">

      <body name="bKrychle">

              <transform>
                    <position x="0" y="0" z="0" />
               </transform>

              <mass>
                    <mass_shape density="1000">
                          <box sizex=“1“ sizey=“1“ sizez=“1“ />
                    </mass_shape>
              </mass>

              <geom name="gKrychle">
                    <box sizex=“1“ sizey=“1“ sizez=“1“ />
              </geom>

        </body>
    </world>
</xode>
```

Modul jazyka Python pro práci s ODE nabízí možnost čtení těchto souborů pomocí jednoduchých metod, z čehož bylo vyvozeno, že v tomto směru bude způsob práce s XODE vyřešen. To se však ukázalo jako jen částečně pravdivé tvrzení, a to z následujících základních důvodů, které nebyly zpočátku zřejmé:

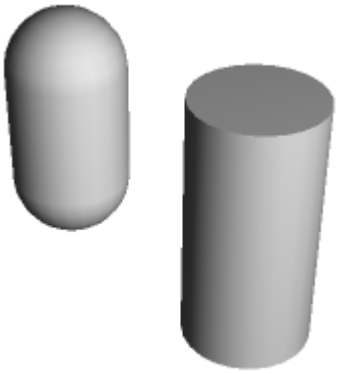
- PyODE mírně upravuje některá pravidla vzhledu souboru XODE oproti specifikaci
- Některé části XODE specifikace nejsou implementovány
- Dokumentace nevystihuje současný stav některých částí modulu
- Modul neobsahuje funkce pro zápis XODE souborů, pouze jejich čtení

První zmíněný problém se týká drobných detailů, jako je nutnost specifikovat jméno pro značky world a zavedení nové značky space. Vzorový příklad uvedený výše se tak změní na následující:

```
<?xml version="1.0" encoding="UTF-8"?>

<xode version="1.0" name="VzorovaKrychle"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://tankammo.com/xode/1.0r17/xo
       de.xsd">

    <world name=“world“ transformstyle="vector">
        <space name=“space“>
              <body name="bKrychle">

                    <transform>
                    <position x="0" y="0" z="0" />
                </transform>

                <mass>
                    <mass_shape density="1000">
                        <box sizex=“1“ sizey=“1“ sizez=“1“ />
                    </mass_shape>
                </mass>

                <geom name="gKrychle">
                    <box sizex=“1“ sizey=“1“ sizez=“1“ />
                </geom>

              </body>
        </space>
    </world>
</xode>
```

Tato změna oproti standardu však není samoúčelná a má své opodstatnění. Modul PyODE obsahuje tzv. *parser* (nástroj pro čtení jednotlivých elementů souboru). Tento parser XODE souborů tak díky zmiňované změně může ke všem objektům v souboru přistupovat na základě jejich názvu. Do budoucna nabízí i schopnost definice více světů, které by pak bylo možno načítat dle jejich unikátního jména.

Druhý problém, a to mezery v implementaci formátu, se ukázal jako závažnější. Navzdory přítomnosti objektu válec v metodách ODE, parser tento typ objektu vůbec nerozpoznával. Řešení této situace byla dvojí - modifikace zdrojových kódů modulu nebo provádění konverze z příbuzného objektu – kapsle (viz obr. 12).



Obr. 12 Kapsle a válec, odlišná jsou pouze zakončení

Během práce na načítání XODE byly zvažovány a implementovány oba přístupy, nakonec byla zvolena cesta úpravy zdrojových kódů modulu tak, aby daný objekt a jeho vlastnosti rozpoznal. Důvodem tohoto rozhodnutí byly pozitivní dopady modifikace na vzhled a srozumitelnost kódu pro načítání XODE, který se oproti druhé variantě podstatně zkrátil. Při tomto procesu byly podniknuty i kroky k informování programátora o stále chybějících vlastností modulu, které původní verze v některých případech nijak neošetřovala, a uživatel tak mohl nabýt dojmu, že jsou podporovány.

Tento zásah do PyODE sice nevyřešil všechny problémy které tento modul má, ale bylo jím dosaženo stavu použitelnosti, který umožnil další práce na zobrazování výstupu generátoru i XODE souborů obecně bez větších omezení.

Poslední dvě komplikace - nedokonalost dokumentace a absence zápisu XODE, nepředstavovaly pro vypracování projektu zásadní překážku.

XODE jako takové neposkytuje možnost definice vizuálních vlastností geometrických těles, což z hlediska simulace problém nepředstavuje. Pro snazší optickou rozlišitelnost objektů však bylo v rámci této práce přistoupeno k jednoduchému způsobu uložení informace o barvě či průhlednosti tělesa. Tyto údaje jsou připisovány ke jménu každého objektu typu Geom. Tento nenáročný způsob zajistí, že vytvořené XODE soubory si zachovají vnější vzhled souboru dle specifikace, a proto nehrozí problém s nekompatibilitou při práci v jiných programech.

ČÁST B – PRAKTICKÁ ČÁST

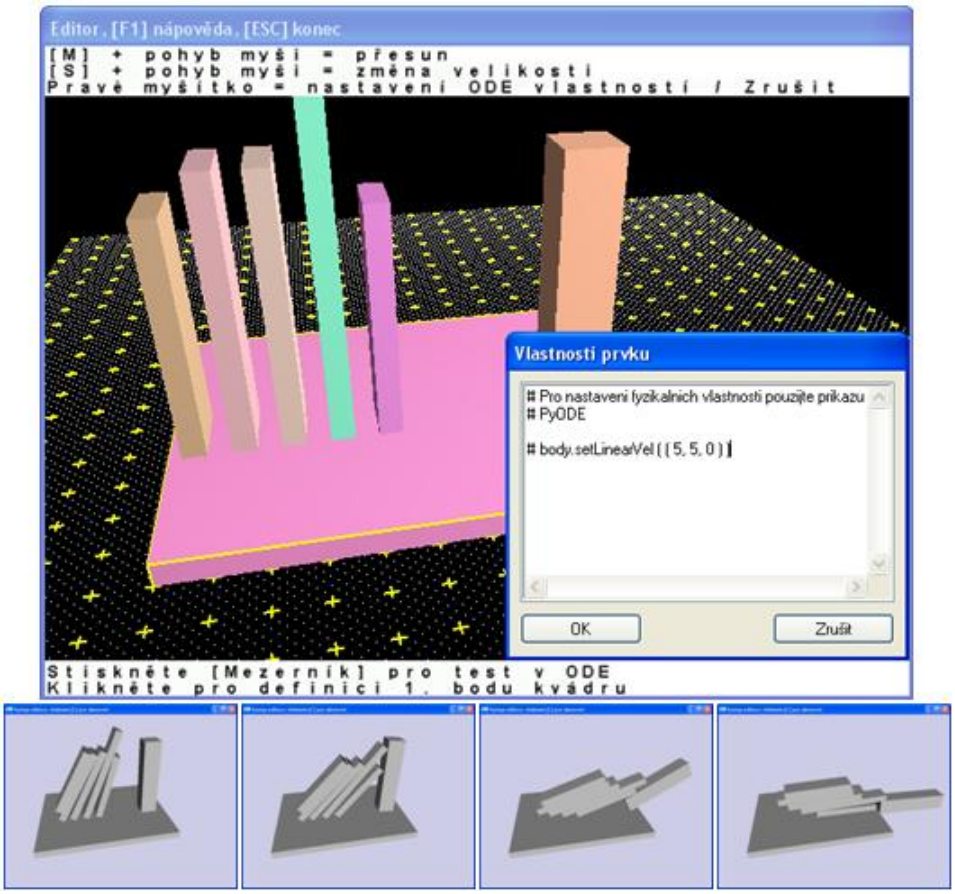
7 ROZBOR PŘÍSTUPŮ PRO TVORBU PROSTŘEDÍ

Dříve než započaly samotné práce, bylo třeba se rozhodnout pro specifický typ prostředí. Vzhledem k účelu použití byl po dohodě s vedoucím práce vybrán interiér budov. Přestože dnes je možné, díky široké nabídce již hotových řešení a dostupnosti podkladů, generovat rozsáhlé, atraktivně působící exteriéry, pro test lokalizace se jako vhodnější jeví vnitřní prostory budov.

Tvorba prostředí pro testy mobilního robotu může být realizována několika způsoby. Proto bylo navrženo několik prototypů generátorů. K těmto počátečním testům byl použit programovací jazyk thinBASIC[27], který autor dobře zná, a tudíž návrh konceptu generátoru mohl probíhat velmi rychle a bez rizika velkého zdržení.

7.1 Koncept interaktivního editoru

Původní idea počítala s vizuálním návrhem prostředí, ve kterém by měl tvůrce plnou kontrolu nad vzhledem scény. Definice scény probíhala pomocí myši a klávesnice, za použití jednoduchých nástrojů pro přidávání kvádrů i složitějších objektů jako je např. schodiště.



Obr. 13 Vzhled editoru a náhled simulace

Fyzikální vlastnosti bylo možno nastavit pro každý objekt, a to ve formě krátkého kódu v jazyce Python. Z editoru bylo možné kdykoli spustit automaticky generovaný Python skript a pozorovat tak chování scény (viz obr. 13). Přestože tento systém fungoval poměrně dobře, nebyl nakonec použit. Jako hlavní problém se ukázala malá efektivita práce, ruční definice objektů zabírala příliš mnoho času.

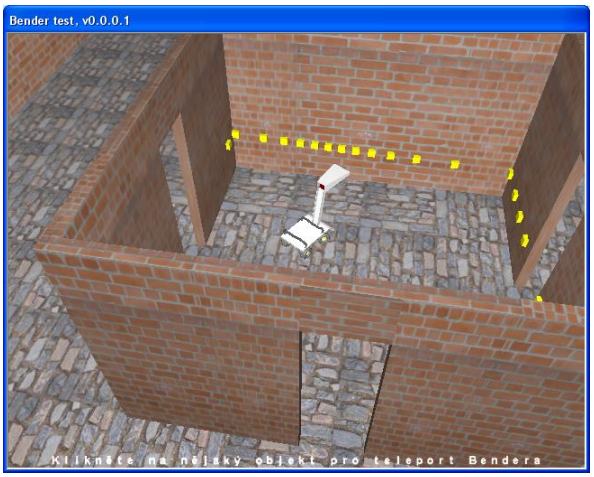
7.2 Koncept dlaždicového prostředí

Po předchozím experimentu byl zvolen zcela odlišný přístup. Byla vytvořena knihovna prvků o fixní velikosti, které sloužily jako 3D dlaždice a byly umisťovány do prostoru na mřížku o pevné velikosti. Jednoduchým skládáním dlaždic vedle sebe bylo možno vytvořit statické trojrozměrné prostory, nevýhodou tohoto řešení byla však absence možnosti jakékoli parametrizace.

Obr. 14 Model robotu Bender

Při těchto testech byl použit velmi jednoduchý model robotu Bender (viz obr. 14). Počítačový model sice přesně neodpovídal charakteristikám své reálné předlohy, ale sloužil jako nosič senzorů vzdálenosti a barev. Simulace senzorů vzdálenosti byla realizována pomocí knihovny OpenGL. Vzdálenost překážky od senzoru byla vždy stanovena na základě zjištění hloubky určitého pixelu pomocí hodnoty z-bufferu grafické karty. Díky tomuto přístupu nebylo nutné provádět výpočet kolize parsku senzoru a detekované plochy, který by byl výpočetně náročný.

Obr. 15 ukazuje schopnost simulovaného senzoru detekovat překážky v rozsahu 180°, jedná se o 2D scan okolí robotu ve stanovené výšce nad povrchem. Kolizní body jsou symbolizovány žlutými krychlemi. Senzor zjistil u každého z těchto bodů jeho vzdálenost, polohu v prostoru (souřadnice) a RGB barvu.



Obr. 15 Ukázka detekce nejbližších bodů

Vzhledem k výhodám, které modul pro 3D grafiku v jazyce thinBASIC nabízí, bylo možné nastavit kamery na libovolná místa objektu a tak sledovat okolí z pohledu robotu. Obr. 160 ukazuje kromě krychlí, indikujících blízké body, i kouli stejné barvy, jako barva fragmentu ve středu obrazu. Koule sloužila k ověření funkčnosti senzoru.

Obr. 16 Detekce nejbližších bodů i barvy středu obrazu

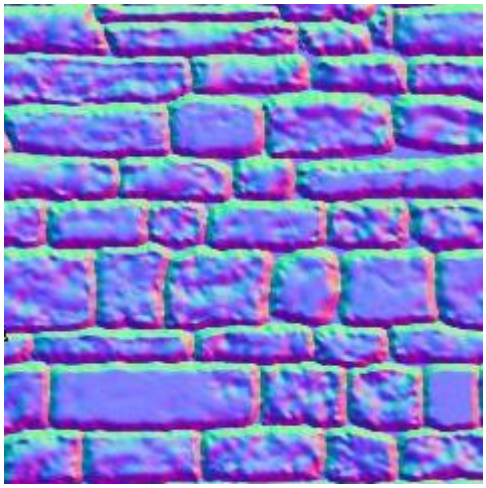
Jak již bylo zmíněno, hlavní nevýhodou tohoto řešení byla nedostatečná variabilita prostředí a stále poměrně vysoké požadavky na jeho obsluhu a tvorbu.

Přestože u tohoto konceptu byly použity textury pro vylepšení vzhledu i vizuální věrnosti, v dalších pokusech se od nich upustilo. Pro kvalitní grafické zobrazení prostředí, lepší zpracování světla a hlavně realističtější rozptyl paprsků od povrchů, by bylo nutné vytvořit i sadu normálových map.

Jejich tvorba je však poměrně komplikovaná. Vyžaduje vytvoření komplexního povrchu, který textura reprezentuje a současně je relativně náročná na grafický hardware. Novější grafické karty optimalizované pro vykreslování náročných scén herních aplikací

zvládají zpracování této techniky bez větších problémů, to však bohužel zatím nelze jednoznačně tvrdit i o integrovaných grafických kartách.

Normálová mapa jako taková je vlastně bitmapou, barvy jednotlivých pixelů reprezentují normálové vektory (viz obr. 17). Objekty používající výpočet světla pomocí této techniky tak mohou mít jednoduchou geometrickou reprezentaci (např. zeď definovaná jedním obdélníkem), ale díky údajům z bitmapy se jeví jako komplexnější objekty.



Obr. 17 Normálová mapa pro kamennou zeď

7.3 Závěry vyvozené z výsledků experimentů

Ze zmíněných předběžných pokusů, které sloužily k nalezení co nejlepšího možného řešení generování prostředí, bylo vyvozeno několik závěrů. Předně bylo upuštěno od vizuálního návrháře, který sice dává největší kreativní možnosti, bohužel je práce s ním vždy časově náročná. Koncepce dlaždic se sice plně neosvědčila, jisté prvky z ní však později byly použity ve finálním řešení, jež bude podrobněji popsáno v následující kapitole.

8 NÁVRH GENERÁTORU PROSTŘEDÍ

Na základě poznatků získaných při přípravných pracích byl zvolen nový přístup ke generátoru prostředí. Ten spočíval v návrhu API, které by nabízelo možnost definice prostředí na několika úrovních složitosti a jejich přímém zápisu do XODE souborů. Toto rozhraní je rozvrstveno do několika úrovní, které umožňují definici a zápis následujících:

- Vrstva 0 - Hlavička a patička souboru, obecný zápis
- Vrstva 1 - Elementární geometrické prvky a jejich transformace
- Vrstva 2 - Složené objekty, reprezentující části interiéru
- Vrstva 3 - Obecné lokace

Elementárními geometrickými prvky jsou myšlena geometrická primitiva jakými jsou kvádr či koule. Tyto objekty lze do souboru zapisovat jak v podobě statických překážek, tak i objektů s fyzikálními vlastnostmi typu hmotnost, zrychlení a dalšími. Výjimkou z pravidla je speciální případ krychle (zeď) a tzv. trimesh. Jedná se o geometrickou reprezentaci objektu, založenou na popisu trojúhelníkovou sítí.

Složené objekty představují nábytek. Ten je v generátoru přítomen ve třech typech - židle, stůl a skříň. Tyto objekty jsou implementovány pomocí zmíněných elementárních prvků, stejnou cestou by bylo možno typy nábytku v budoucnu rozšířit.

Nejvyšší vrstva pak nabízí funkce, jejichž výstupem je ucelené prostředí.

Následující text vás podrobně seznámí se schopnostmi funkcí jednotlivých vrstev. Pro podrobný popis každé funkce doporučuji k nahlédnutí referenční příručku v elektronické příloze.

8.1 Vrstva pro základní práci s XODE souborem

Nejnižší úroveň funkcí využívaných generátorem je představována metodami, které zapouzdřují práci se souborem. Nabízí tak to nejzákladnější pro zápis XODE, včetně hlavičky a patičky dokumentu.

8.2 Vrstva transformací a jednoduchých objektů

Tato úroveň poskytuje funkce dvojího typu – základní transformace pozice objektů a zápis grafických primitiv.

Transformační funkce slouží k určení místa, kam bude objekt vložen. Práce z pozicí je velice jednoduchá, a obsahuje i jednoduchý zásobník, pomocí kterého je možné pozice zálohovat a znovu obnovovat.

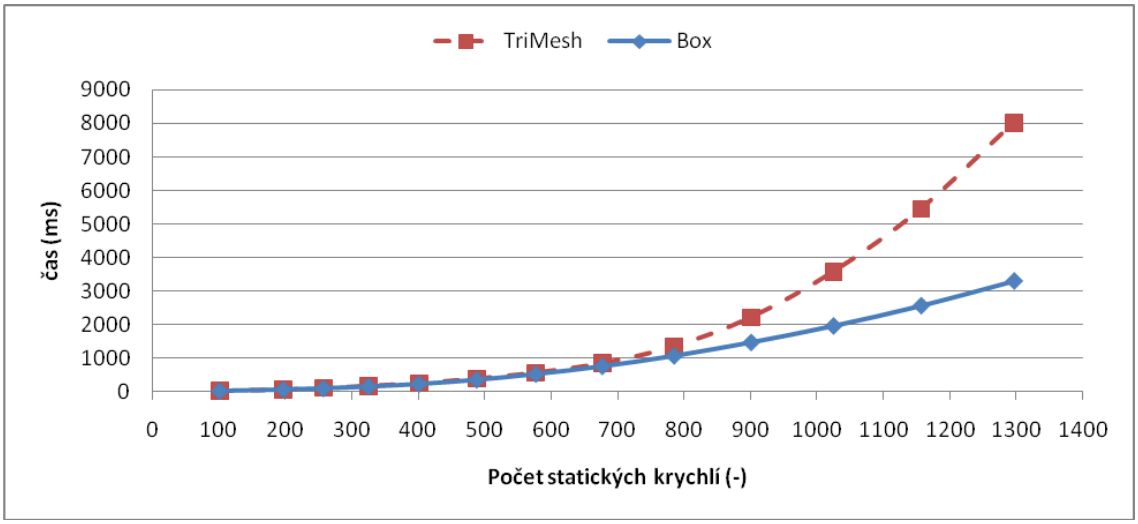
Objekty zapisované touto vrstvou jsou trojího typu:

- základní geometrická primitiva
- zdi, sloupy
- trimesh

Podporovaná primitiva jsou kvádr, koule, krychle a obecná plocha. Termín zeď a sloup zde označuje speciální případ krychle, která není zadána svým středem ale podstavou. Navíc jsou vkládány jako pevná, nepohyblivá tělesa.

Zvláštní pozornost si zaslouží zmíněný trimesh (složenina anglických slov triangle, trojúhelník a mesh, síť), tedy objekt definovaný sítí trojúhelníků. Jeho největší výhodou je možnost popisu tělesa libovolného tvaru. Pro tvorbu trimeshů byl zvolen postup konverze z formátu Wavefront OBJ. Tento formát je pro tento účel vhodný z několika důvodů. Vnitřní strukturou je téměř shodný se způsobem reprezentace trimeshe v XODE souboru. Navíc je formát OBJ exportován z velkého množství zavedených modelovacích programů jako jsou Rhinoceros, Blender, 3D Studio a další.

V určité fázi návrhu generátoru se uvažovalo o nahrazení geometrických primitiv trimesh modely, výpočetní náročnost by však podstatně vzrostla, jak ukazuje graf na obr. 18.



Obr. 18 Výpočetní náročnost krychlí definovaných obecně a trimeshem

8.3 Vrstva pro tvorbu složitých objektů

Jak židle, tak i stůl a skříň nejsou reprezentovány objekty o fixní velikosti, naopak je možno je parametrizovat tak, aby bylo dosaženo co největší tvarové a rozměrové pestrosti. Míra parametrizace je nejvyšší u objektu skříně, na jejímž stručném popisu vám bude přiblížen přístup, který byl při tvorbě nábytku zvolen.

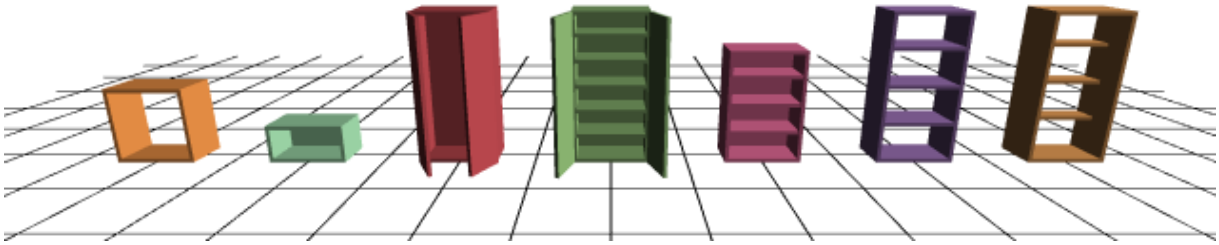
8.3.1 Ukázka možností parametrizace

Většina místností v domech obsahuje nějaký větší nábytek charakteru skříně. Aby bylo možné vkládat skříň jednoduše, a přitom zachovat možnost vlastní úpravy, má tento objekt následující nastavitelné charakteristiky:

- Šířka
- Výška
- Hloubka
- Zadní přepážka (ano/ne)
- Horní přepážka (ano/ne)
- Dolní přepážka (ano/ne)
- Počet poliček (0 - n)
- Zanoření poliček ve skříni

- Dveře (ano/ne)
- Počáteční úhel levého křídla dveří (0 – 180°)
- Počáteční úhel pravého křídla dveří (0 – 180°)

Jednoduchou úpravou těchto parametrů tak dostáváme velmi rozličné podoby, jak ukazuje obr. 19.



Obr. 19 Některé z možných výstupů funkce pro tvorbu skříně

Zobrazeny jsou vybrané typy skříní, v pořadí zleva doprava se jedná o následující varianty:

- Jednoduchá
- Jednoduchá se zadní příčkou
- Jednoduchá se zadní příčkou a pootevřenými dveřmi
- Jednoduchá se zadní příčkou, pootevřenými dveřmi a poličkami
- Jednoduchá se zadní příčkou a poličkami
- Jednoduchá bez zadní příčky a s poličkami
- Jednoduchá bez zadní příčky a s poličkami posunutými vzad

Tato tvarová pestrost nabízí snadnou cestu jak upravit výsledný ráz místnosti. Důležitým prvkem jsou pak pootevřené dveře, které mohou robot mást a představovat překážku v cestě.

8.4 Vrstva pro generování lokací

Nejvyšší úroveň nabízí úzce specializované funkce pro tvorbu rozsáhlých trojrozměrných lokací, a představují tak samotné jádro generátoru. Pro větší flexibilitu tato vrstva nabízí hned dva přístupy k tvorbě prostředí.

8.4.1 Metoda tvorby pravoúhlé sítě místností

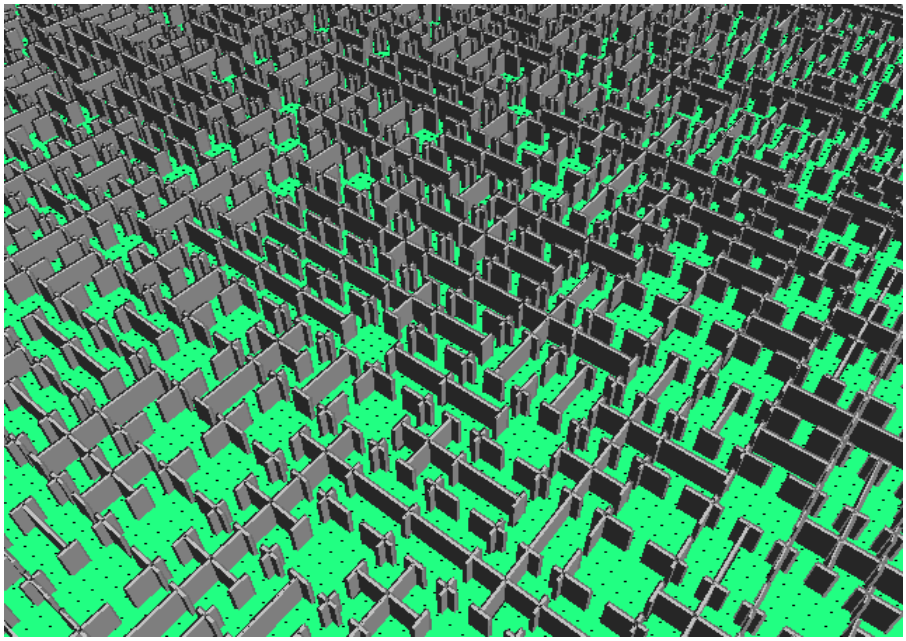
První přístup umožňuje zaplnit vymezenou plochu místnostmi. Funkce rozdělí danou plochu na definovaný počet podoblastí, které pak generátor na základě daných pravidel může rozdělit. Jednotlivé místnosti jsou navzájem spojeny průchody, přičemž platí, že mezi každými dvěma místnostmi, které mají dveře, existuje cesta.

V implementované verzi generátoru existují 4 základní způsoby, jak ze zmíněných podoblastí vytvořit místnosti:

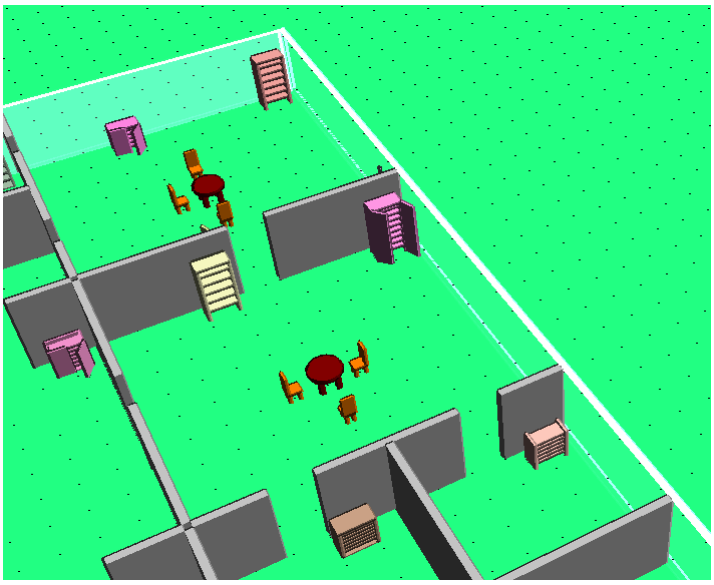
- Příčným dělením
- Horizontálním dělením
- Rozdělením na čtyři
- Bez dělení

Tyto místnosti mohou být zaplněny nábytkem – židlemi, stoly a skříněmi. Tyto elementy se do prostoru umisťují dle jednoduchých pravidel tak, aby co nejvíce odpovídaly běžnému rozložení v reálných budovách. Stoly jsou tak umisťovány doprostřed velkých místností a mohou být automaticky obklopeny židlemi. Skříně jsou logicky kladeny k některým stěnám místnosti, aby nepřekážely ve dveřích. Zároveň jsou vždy otočeny do místnosti tak, aby do nich bylo vidět.

Automatické umisťování jednotlivých typů nábytku je možné i vynechat (viz obr. 20), a provést dle vlastních představ pomocí metod předchozí vrstvy. Pokud však potřebujeme rychle vytvořit kompletní prostředí, je doporučováno automatické zaplnění nábytkem povolit.



Obr. 1 *Obr. 20 Výstup generátoru bez nábytku*



Obr. 21 *Výstup generátoru s nábytkem*

Jak je patrné z obr. 21, pro větší přehlednost jsou obvodové zdi poloprůhledné, což umožňuje základní vizuální kontrolu nad simulovanými ději v prostorách místností.

8.4.2 Využití upraveného Lindenmayerova systému

Postup zmiňovaný v minulé kapitole nabízí dostatečnou míru variability, přesto si mohou být generované lokace svým rázem do jisté míry podobné. Pro případ, kdy by tato vlastnost znamenala problém, a testy mobilního robotu by vyžadovaly odlišný ráz prostředí, byl do nejvyšší vrstvy zahrnut i jednoduchý generátor využívající pro tvorbu prostředí Lindenmayerovy systémy, též známé jako L-systémy.

Ty jsou matematickým formalismem, původně navrženým pro modelování procesů v přírodě[12]. Typů L-systémů existuje několik, v následujícím textu však bude uvažována pouze základní, tzv. deterministická bezkontextová varianta.

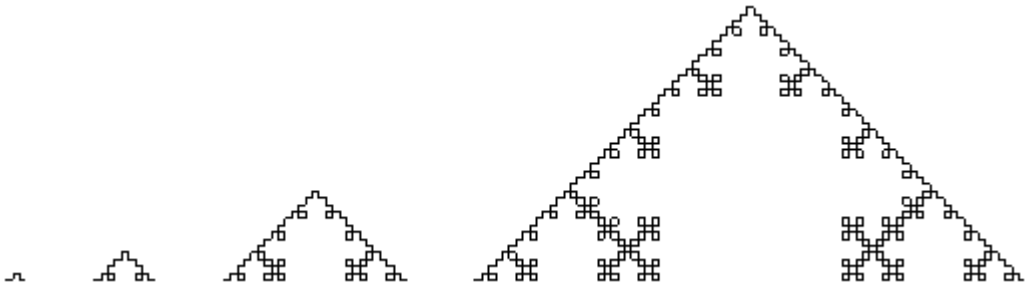
L-systémy fungují na principu paralelního přepisování řetězců, které obsahují symboly později použitelné pro grafickou interpretaci. Tyto řetězce mohou obsahovat dva typy symbolů, které budu pro zjednodušení nazývat dle konvence uvedené ve zdroji[28]:

- Proměnné
- Konstanty

Proměnnými nazýváme takové symboly, které mají přiřazeno pravidlo pro vlastní přepsání. Konstanty jsou symboly, které se při iteracích systému měnit nebudou. Pro představu uvedu známý příklad systému, který slouží ke generování Kochovy křivky:

- Proměnné: F → F+F-F-F+F
- Konstanty: +, -
- Výchozí stav: F

Ke grafické interpretaci L-systémů bývá používáno tzv. *želví grafiky*. Jedná se o jednoduchý princip, který pracuje s konceptem „kreslící želvy“. Imaginární želva se může pohybovat pouze vpřed a zatáčet, při tom za sebou zanechává stopu. V tomto případě se symbol F bere jako povel „vpřed“ a konstanty + a – jako příkaz k zatočení o 90°. Jednoduchou grafickou reprezentaci ukazuje obr. 22.

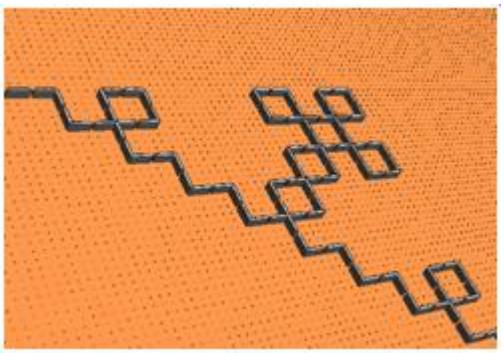


Obr. 22 První 4 iterace Kochovy křivky

Jak je patrné z tohoto přikladu, L-systémy jsou vhodné nejen pro simulaci růstu rostlin či buněčných struktur, ale v podstatě pro reprezentaci libovolných tvarů, které při použití otáčení o pravý úhel mohou do jisté míry dokonce připomínat interiér budov.

Pro účel generátoru tak slouží D0L systémy jako přístup pro tvorbu zdí s průchody. Díky značné míře možností podoby výstupu bylo u tohoto generátoru upuštěno od automatické definice jakéhokoli nábytku. L-systémem generovaný komplex tvořený zdmi dokáže být dostatečně komplexní i bez těchto prvků.

Pro přiblížení uvádím v příloze 3 kód, který vytvoří labyrint vycházející s Kochovy křivky. Výstup programu je patrný na obr. 23.



Obr. 23 *Část geometrie vygenerované interpretací L-systému*

Tento krátký program může sloužit, zvláště při vyšším množství iterací, ke tvorbě velmi rozsáhlých oblastí, které pak může robot procházet.

8.5 Souhrn schopností navrženého generátoru

Nejpodstatnější součástí vytvořeného generátoru je bezesporu nejvyšší vrstva funkcí, která slouží k tvorbě kompletních lokací. Možnost zaplnit libovolně velkou oblast sítí místností šetří práci, která by byla strávena manuální definicí těchto prvků. Spletitá síť chodeb, kterou

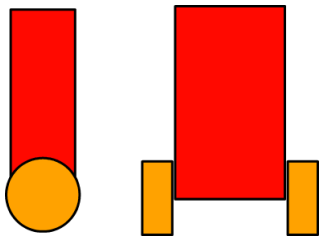
generátor navrhne, je sama o sobě do značné míry složitá. Přidá-li se k této prostorové pestrosti navíc i prvek nábytku, zvláště skříní s pootevřenými dveřmi, vzniká tak virtuální prostor, který je pro navigační či řídící systémy robotu dostatečnou výzvou.

Druhý generátor nejvyšší vrstvy, založený na interpretaci výstupu L-systému, je pak alternativou pro případ, kdy potřebujeme pro robot vytvořit prostředí jiného typu, než místnosti a chodby.

Nižší vrstvy generátoru, které obstarávají základní práci s tělesy, mohou posloužit jako základ pro jiné typy generátorů prostředí, dokonce i jako pomocné funkce pro vizuální návrháře, který bude předmětem zájmu diplomové práce autora. Použití nižších vrstev je tak vhodné v případech, kdy chceme vytvořit prostředí, které nemá charakteristickou stavbu, či pro obohacení výstupu již zmíněných generátorů složitých lokací.

9 OVĚŘENÍ FUNKCE GENERÁTORU V SOUČINNOSTI S IMPORTOVANÝM DYNAMICKÝM PRVKEM

Po navržení a realizaci generátoru lokací (viz kap. 8.4) a vhodném nastavení Open Dynamics Engine (viz kap. 5.3) bylo nutné provést ověření vhodnosti vytvořeného řešení. Jelikož bude výstup generátoru v budoucnu využit pro testy robotů, bylo nutné do něj umístit virtuální mobilní robot. Práce[32] se zabývá balancujícím robotem „Pierot“. Model vyhotovený jejím autorem se tak stal objektem pro test. Robot „Pierot“ se sestává ze dvou kol připojených k vysokému tělu, viz obr. 24.



Obr. 24 *Pohled na robot zboku a zepředu*

9.1 Návrh struktury simulačního programu

Pro účely experimentu byl navržen a naprogramován koncept struktury programu zajišťující simulaci. Program je objektově orientován a při jeho návrhu byl kladen důraz na zachování obecnosti v maximální možné míře. Vytvořený program je tak možné využít pro různé simulace virtuálních robotů v rozličných prostředích, viz [32][33]. Za přínosné tak lze považovat zachování unifikovaného vzhledu zdrojového kódu.

Program tvoří několik samostatných modulů, které slouží k řešení jednotlivých podproblémů simulace a jejího provádění:

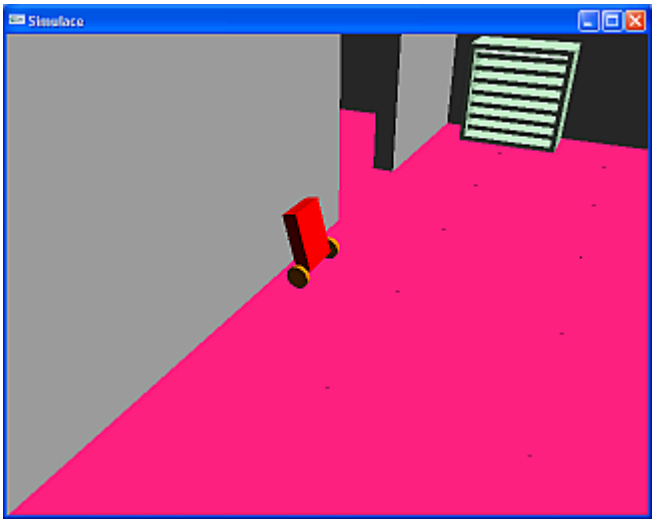
- sSim
- sTaskToSim
- sCreateObjects
- sPaintSim

První modul (sSim) obstarává samotný chod simulace, druhý (sTaskToSim) slouží k nastavení chování objektů v průběhu simulace. Modul sCreateObjects slouží k zjednodušené definici objektů, jako jsou válec, krychle či koule. Objekty či celé scény však mohou být do simulace importovány i pomocí navržené funkce pro import XODE. Poslední modul programu (sPaintSim) nabízí možnost automatické vizualizace objektů, bez nutnosti znalosti OpenGL.

9.2 Provedení testu

Autor[32] využívá pro vytvoření robotu pouze příkazů PyODE, samotný robot se nachází na rovině, kde se nevyskytují žádné překážky. Pro test robotu a generátoru by však byly tyto podmínky nevhodné. Generátorem tak byla vytvořena síť místností. Pro jednoduchost a přehlednost byl „Pierot“ přímo součástí XODE souboru s vygenerovanou lokací. Jednotlivé části robotu byly pro snadnější rozlišení od elementů prostředí označeny jmény Pierot_Kolo1, Pierot_Kolo2, Pierot_Telo (viz. zdrojový kód v příloze 4).

Díky tomu, že simulace robotu v práci[32] byla napsána v jazyce Python, bylo začlenění jejich prvků do zmiňovaného prototypu programu bez jakýchkoli komplikací. Kód pro regulaci polohy robotu byl jednoduše vložen do metody modulu sTaskToSim, jednotlivé části robotu byly načteny ze souboru a identifikovány na základě již zmiňovaných jmen.



Obr. 25 *Robot v průběhu testu*

Aby bylo možné zjistit, zda je robot schopen kontaktu s prostředím, byl robot umístěn do blízkého okolí tělesa, reprezentujícího zeď místnosti (viz obr. 25), taktéž získané z XODE souboru. Na počátku simulace byl zavedením momentu do vazeb robot uveden do nestabilní polohy. Díky výchylce, kterou způsobil akční zásah, došlo ke kontaktu robotu s překážkou a tedy i silovému působení na robot. Navržený PID regulátor natočením kol zabránil pádu robotu a ustálil jej do vodorovné podoby.

Provedený experiment tak potvrdil možnost součinnosti statického prostředí, vytvořeného generátorem, a mobilního robotu.

10 ZÁVĚR

V této práci jsem mimo jiné prozkoumal možnosti vybraných fyzikálních knihoven, které jsou v současné době k dispozici, viz kap. 3.1. Některé z nich jsem měl možnost posoudit pouze dle dostupných materiálů, s některými jsem však měl možnost se seznámit blíže, a tak lépe zhodnotit jejich schopnosti. I přes vysokou propracovanost Ageia PhysX či snadno pochopitelnou syntaxi Newton Game Dynamics je dle mého názoru Open Dynamics Engine velmi dobrým řešením pro simulaci fyzikálních dějů. A to jak pro svoji architekturu, tak i pro aktivní komunitu, která tuto knihovnu používá pro široké spektrum aplikací. ODE nachází uplatnění jak v populární oblasti počítačových her, tak i simulátorech vozidel, letadel a v neposlední řadě i robotů.

Velké množství dostupných tutoriálů, přehledná příručka i udržovaná Wiki umožňuje pochopit princip ODE velmi rychle, a práce s ním je tak velmi intuitivní. Výše zmíněné platí do jisté míry i pro modul Pythonu PyODE. Ten sice zpočátku některými svými vlastnostmi kladl jisté překážky pro zhotovení práce, i tento problém byl záhy překonán vlastní úpravou tohoto modulu, která byla možná jak díky otevřeným zdrojovým kódům, tak i jejich snadné pochopitelnosti a logické struktuře (viz kap. 6). Nevýhodou tohoto řešení však může být nekompatibilita s budoucí verzí PyODE. Oproti ODE nabízí nástavba PyODE navíc i výhodu v podobě srozumitelnější objektové syntaxe.

Pro vykreslování výstupu simulace byl použit modul pro zpřístupnění OpenGL v jazyce Python, viz kap. 4.2. OpenGL je grafické API, které umožňuje kvalitní a rychlou vizuální prezentaci výsledků simulací. Tato knihovna se navíc s velkou pravděpodobností dočká během podzimu roku 2008 specifikace nové verze, stále však zpětně kompatibilní s tou stávající. To jistě otevře nové cesty pro budoucí, realitě věrnější grafickou prezentaci simulací.

XODE, formát souboru stavějící na koncepci XML, se ukázal jako vhodné řešení, jak pro svou schopnost reprezentovat přesně strukturu ODE, tak i pro jednoduchá pravidla která k tvorbě tohoto typu souboru postačují, viz kap. 6. V rámci této práce byla rozšířena škála informací, které tento soubor může nést o barvu a průhlednost objektů, což ve spojení s vykreslením pomocí již zmíněného OpenGL znamená pohodlnou vizuální kontrolu nad průběhem simulace, viz kap. 8.4.1.

Těžiště této práce, generátor lokací, je složen ze 4 základních vrstev, viz kap. 8. Tyto vrstvy poskytují programátorovi nejen možnost základní práce se soubory XODE, ale i možnost definovat základní i složená tělesa, např. nábytek, který může být vkládán do prostředí (viz kap. 8.3). Složené prvky mohou být parametrizovány a tvořené prostředí tak může jednoduše nabýt velmi proměnlivého rázu. Nejvyšší vrstvy generátoru, které poskytují hned dva přístupy ke generování kompletních trojrozměrných lokací (viz kap. 8.4), pak umožňují tvořit testovací prostředí pro robot ve velmi krátkém čase. Je možné vytvořit blok místností o rozloze panelového bytu či rozsáhlou oblast pokrývající plochu mnohonásobně větší. Systém pro tvorbu sítě (volitelně plně vybavených) místností spolu s konfigurovatelným generátorem prostor na bázi Lindenmayerova systému tak v budoucnu poslouží i dalším projektům a aplikacím v robotice.

Možnost testu chování robotu ve vytvořeném prostředí byla ověřena při experimentu s dvoukolým robotem „Pierot“(viz kap. 9). Import robotu nebyl obtížným úkolem, a to díky do značné míry shodným technologiím, které tato i zmiňovaná práce využívaly. V budoucnu tak lze předpokládat relativně bezproblémovou tvorbu nových projektů, které na funkcích generátoru vyvinutém v rámci této práce budou stavět. Pro tento účel byl ve spolupráci se

školitelem a Ing. Vítem Ondrouškem navržen prototyp programu, který integruje simulaci v PyODE a vizualizaci pomocí OpenGL do jednoho celku (viz kap 9.1).

Za přínos této práce tak považuji:

- Získané znalosti o Open Dynamics Engine
- Řešení pro zápis i načítání XODE souborů
- Realizované programové rozhraní generátoru
- Vytvoření struktury simulačního programu
- Ověření schopnosti použití generátoru a robotu

V řešení problematiky generátoru budu pokračovat v rámci diplomové práce. Některé nepoužité přístupy objevené při práci na tomto projektu (jako jsou např. normálové mapy, viz kap. 7.2) tak v budoucnu ještě mohou najít uplatnění.

SEZNAM LITERATURY

[1] Havok. *Logo* [online].[cit 2007-12-20].
Dostupné z:<http://www.havok.com/>

[2] Ageia. *Logo* [online].
[cit 2007-12-20].Dostupné z:<http://www.ageia.com/>

[3] Shrout, R.*John Carmack on id Tech 6, Ray Tracing, Consoles, Physics and more* [online]. 2008, 12.03.2008 [cit. 20.03.2008].
Dostupné z:<http://www.pcper.com/article.php?aid=532&type=expert&pid=1>

[4] Newton Game Dynamics. *Logo.* [online].[cit 2007-12-20].
Dostupné z:<http://www.newtondynamics.com/ >

[5] SMITH, Russel. *Open Dynamics Engine*. [online].[cit 2007-12-20].
Dostupné z:<http://www.ode.org/>

[6] GAAL, Rud van; Dolphinity B.V., *Racer car and racing simulator* [online].
[cit 2008-02-10]
Dostupné z:<http://www.racer.nl/>

[7] Arenalogic.com,*Viper Arena*, [online]. [cit 2008-03-17]
Dostupné z:<http://www.arenalogic.com/>

[8] AT Humboldt, *Project Simloid*. [online]. [cit 2008-04-04]
Dostupné z:<http://www.robocup.de/AT-Humboldt/simloid.shtml>

[9] DENISS, William. *XODE specification*. [online]. [cit 2007-09-12]
Dostupné z:<http://tankammo.com/xode/xode.txt>

[10] SHOAFF, William D. *Rasterization* [online] *2002-04-04* [cit 2008-05-17]
Dostupné z:
<http://www.cs.fit.edu/~wds/classes/graphics/Rasterize/rasterize/rasterize.html>

[11] BLYTHE, David. *Rasterization* [online] *Sat Mar 29 02:23:21 PST 1997*
[cit 2008-05-17]
Dostupné z:
<http://www.opengl.org/documentation/specs/version1.1/glspec1.1/node41.html>

[12] ŽÁRA, Jiří; Beneš Bedřich; Sochor Jiří; Felkel Petr. *Moderní počítačová grafika*.2. vyd. Brno. Computer press. 609 s. ISBN 80-251-0454-0

[13] VALICH, Theo. *Intel Touts Larrabee At GDC*.[online] *2008-02-21*
[cit 2008-05-17]
Dostupné z:
<http://www.tomshardware.com/news/intel-touts-larrabee-gdc,4832.html>

[14] Wikipedia. *DirectX*. [online]. [cit 2008-05-17]
Dostupné z:< http://en.wikipedia.org/wiki/Directx>

[15] Triple Squid Software Design. *MoI, 3D modeling for designers and artists*. [online].
[cit 2008-01-22]
Dostupné z:<http://moi3d.com/>

[16] Dostupné z:<http://www.opengl.org/>

[17] SHREINER, Dave; Woo Mason; Neider Jackie; Davis Tom. *OpenGL – Průvodce programátora*.1. vyd. Brno. Computer press. 680 s. ISBN: 80-251-1275-6

[18] GameDev.net. *NeHe Productions* [online].[cit 2007-12-20].
Dostupné z:< http://nehe.gamedev.net/>

[19] Lighthouse3D. *OpenGL tutorials* [online].[cit 2007-12-20].
Dostupné z:< http://lighthouse3d.com/opengl/tutorials.shtml>

[20] Python Software Foundation. *Python Programming Language*. [online].
[cit 2008-11-05].Dostupné z:<http://www.python.org/>

[21] BEAZLEY, David M. *Python – podrobná referenční příručka*.2003. Neocortex. 445 s. ISBN: 80-86330-05-2

[22] LUTZ, Mark. *Naučte se Python*. Grada, 2003. 360 s. ISBN: 80-247-0367-X

[23] HARMS, Daryl; McDonald Kenneth. *Začínáme programovat v jazyce Python*. Computer Press, 2003. 476 s. ISBN: 80-722-6799-X
[24] SMITH, Russel. *Open Dynamics Engine User Guide*. [online].[cit 2008-03-03]. Dostupné z:< http://www.ode.org/ode-latest-userguide.html>
[25] Wikipedia. *Tření*. [online]. [cit 2008-04-05] Dostupné z:< http://cs.wikipedia.org/wiki/T%C5%99en%C3%AD>
[26] POPE, Mark. *ODE determinism*. [online]. [cit 2008-02-02] Dostupné z:< http://gamecreator.blogspot.com/2007/03/ode-determinism.html>
[27] OLMI, Eros; Bianchi, Roberto. *thinBASIC programming language* [online]. [cit 2007-09-03] Dostupné z:< http://www.thinbasic.com/>
[28] Wikipedia. *L-system*. [online]. [cit 2008-04-05] Dostupné z:< http http://en.wikipedia.org/wiki/L-system>
[29] Canonical Ltd. Ubuntu. *7.10 More info*. [online]. [cit 2008-05-21] Dostupné z:< http://www.ubuntu.com/getubuntu/releasenotes/710tour>
[30] BASS, Matthias, *osobní sdělení*, [cit 2008-03-16]
[31] MIKLENDA, Jaroslav, *osobní sdělení*, [cit 2008-04-16]
[32] ZETKA, Petr: *Realizace jednoduchého dynamického modelu pomocí ODE*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2008. Vedoucí bakalářské práce Ing. Stanislav Věchet, Ph.D.
[33] SERIŠ, Richard: *Využití ODE pro sestavení dynamického modelu čtyřnohého robotu*. Brno:Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2008. Vedoucí bakalářské práce Ing. Vít Ondroušek.

SEZNAM PŘÍLOH

Příloha č. 1 – Zprovoznění generátoru na Windows XP
Příloha č. 2 – Zprovoznění generátoru na Ubuntu 7.10
Příloha č. 3 – Ukázka kódu pro generování Kochovy křivky
Příloha č. 4 – Přiložené CD-R médium s následujícím obsahem:

- Knihovny funkcí pro práci s XODE a Lindenmayerovými systémy
  /Software/Generator/Moduly/

- Generátor interiéru
  /Software/Generator/Interier/VytvorLokaci_Mistnost.py

- Generátor prostředí založený na Lindenmayerových systémech
  /Software/ Generator/Lindenmayer/VytvorLokaci_LSystem.py

- Dokumentace rozhraní pro generování XODE souboru
  /Software/Generator/Moduly/Dokumentace/XODEGenerator.chm

- Test robotu Pierot
  /Software/TestRobotu/XODE_Pierot.py

- Jednoduchý prohlížeč XODE souborů
  /Software/XODE_Viewer/XODE_Viewer.py

- Modifikace PyODE 1.2
  /Software/UpravaPyODE/

PŘÍLOHA 1 – ZPROVOZNĚNÍ GENERÁTORU NA WINDOWS XP

Pro použití programů vytvořených v rámci této práce je nutné nainstalovat Python a jeho moduly, a aplikovat modifikaci pro práci s XODE soubory.

Následující přehled představuje kroky, které je nutno ke splnění výše uvedeného podniknout.

Instalace Python 2.5

Pro instalaci Pythonu je třeba navštívit stránky programu na adrese http://www.python.org/ a ze sekce „Downloads“ stáhnout verzi Pythonu 2.5 ve verzi pro Windows.

Instalace samotná pak probíhá podobně jako v případě jiných aplikací pro MS Windows.

Instalace dodatečných modulů

Některé programy vypracované v rámci této práce vyžadují instalaci modulů třetích stran, které lze získat z následujících odkazů:

- PyODE
  http://downloads.sourceforge.net/pyode/PyODE-1.2.0.win32-py2.5.exe?use_mirror=osdn

- PyOpenGL 2.0.2.01 + Numeric 24
  http://www.develer.com/~rasky/PyOpenGL-2.0.2.01.py2.5-numpy24.exe

- NumPy 1.0.4
  http://downloads.sourceforge.net/numpy/numpy-1.0.4.win32-py2.5.msi?modtime=1194536709&big_mirror=0

- Matplotlib-0.91.1
  http://downloads.sourceforge.net/matplotlib/matplotlib-0.91.1.win32-py2.5.exe?modtime=1196759584&big_mirror=0

Pro plnou funkčnost generátoru je nutné aplikovat „Modifikaci PyODE“ z elektronické přílohy práce. Soubory modifikace jsou určeny k přepsání původních souborů v podadresáři Pythonu, které se typicky nacházejí v následujícím umístění:

```
C:\Python25\Lib\site-packages\xode\
```

PŘÍLOHA 2 – ZPROVOZNĚNÍ GENERÁTORU NA UBUNTU 7.10

Díky volbě jazyka Python, je možno generátor provozovat jako pod MS Windows tak i pod Linuxem. Pro ověření tohoto faktu bylo přistoupeno k pokusu o spuštění programu pod distribucí Ubuntu 7.10 „Gutsy Gibbon“. Tento systém je dostupný zdarma ke stažení[29].

Pro test bylo použito PC o následujících technických parametrech:

- Procesor Intel Pentium II, 400 MHz
- 192 MB RAM, 100 MHz
- Grafická karta NVIDIA GeForce 2 MX400 64MB AGP 4x

Po instalaci systému není možné generátor prostředí spustit ihned, je třeba doinstalovat některé balíčky a proto je nutné připojení k internetu. Samotné získání softwaru je pak velmi jednoduché, stačí zadat následující příkazy do terminálu:

```
sudo apt-get install python-setuptools

sudo easy_install pyrex

sudo easy_install PyOpenGL
```

K instalaci je samozřejmě nutné heslo uživatele root. Instalace PyODE může být dost komplikovaná, doporučuji proto stáhnout balíčky *libode0debian1 0.8.dfsg-3 i386.deb* a *python-pyode 1.2.0-2 i386.deb* ze stránky:

http://www.sixthfloorlabs.com//posts/2007/11/08/ODE_debs/index.html

Pro plnou funkčnost je nutno aplikovat i „Modifikaci PyODE“ z elektronické přílohy – soubory stačí nakopírovat do podadresáře Pythonu:

```
usr\lib\python2.5\site-packages\xode\
```

Po úspěšné instalaci je možné skripty generátoru pouštět z prostředí IDLE a pracovat s nimi podobně jako pod MS Windows.

PŘÍLOHA 3 – UKÁZKA KÓDU PRO GENEROVÁNÍ KOCHOVY KŘIVKY

```
# -*- coding: cp1250 -*-
# Knihovny funkci vytvorene pro tento projekt
# Generator samotny a obecne pouzitelna trida pro tvorbu
# L-systemu
from XODEGenerator_L3 import *
from LSystem_Functions import TLindenmayerSystem

# Vytvorime instanci nejvyssi, 3. vrstvy generatoru
test = XODEGenerator_L3()

# Otevreme novy soubor XODE, hlavicka se automaticky vytvori
test.fileOpenNew('KochovaKrivka.xode')
test.fileWriteLn('<!-- Jednoduchy L-System -->')

# Nastavime barvu pro podlahu a pridame ji do souboru
test.clrSet([255, 128, 64])
test.putFloor()

# Nadefinuje L-System pro tvorbu Kochovy krivky,
# zadame vychozi stav
l = TLindenmayerSystem("F")

# Pridani promenne a jeji transformace
l.addVariable('F', 'F+F-F-F+F')

# Pridani konstant
l.addConstant('+')
l.addConstant('-')

# Provedeme 3 iterace
l.iterate(3)

# Nastavime odlisnou barvu pro novou geometrii
test.clrSet([192, 192, 192])

# Vytvorime geometrii
# ( 5 metru dlouhe useky s pauzou pro dvere, otaceni po 90°)
test.addLSystemWalls(l.LString, 5, 0.5, 90)

# Zapiseme paticku souboru a zavreme ho
test.fileClose()
```