VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

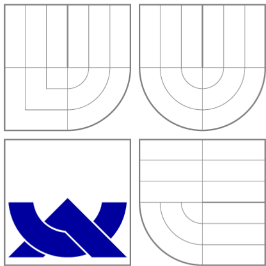
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ POHYBU OBJEKTU V 3D PROSTORU

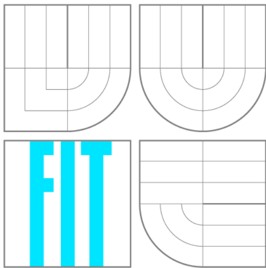
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. RADEK SASÝN
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ POHYBU OBJEKTU V 3D PROSTORU

PATH PLANNING IN 3D SPACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. RADEK SASÝN
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV ROZMAN, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce popisuje hledání cest mezi překážkami ve 3D prostoru pomocí pravděpodobnostních algoritmů. Uživatelé si mohou v uživatelském prostředí vytvořit scénu – přidat objekty, definovat startovní objekt, překážky a cílovou pozici a spustit pravděpodobnostní algoritmus. Nalezenou cestu lze poté vizualizovat. Práce popisuje pravděpodobnostní algoritmy, detekce kolizí a základy práce s 3D grafikou. Dále popisuje návrh a implementaci vytvořené aplikace.

Abstract

This work describes path finding among obstacles in 3D space using probabilistic algorithms. Users can create scene in application GUI – define start object, obstacles, goal position and run probabilistic algorithm. The finding path is visualized. The work describes probabilistic algorithm, collision detection and the basics of 3D graphics and shows design and implementation of an application created.

Klíčová slova

pravděpodobnostní algoritmy, plánování pohybu, detekce kolizí, 3D prostor, OpenGL, C++

Keywords

probabilistic algorithm, path planning, collision detection, 3D space, OpenGL, C++

Citace

Radek Sasýn: Plánování pohybu objektu v 3D prostoru, diplomová práce, Brno, FIT VUT v Brně, 2013

Plánování pohybu objektu v 3D prostoru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana Ph.d.

....
Radek Sasýn
22. 5. 2013

Poděkování

Děkuji vedoucímu práce Ing. Jaroslavu Rozmanovi Ph.D za vedení, konzultování a pomoc při tvorbě práce.

© Radek Sasýn, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod 4

1.1 Skladba práce . . . 5

2 Zobrazení 6

2.1 Tvorba 3D grafiky . . . 6

2.1.1 DirectX – Direct3D . . . 6

2.1.2 OpenGL . . . 7

2.2 Správa okna – knihovna SDL . . . 7

2.3 Modelování 3D objektů . . . 8

2.3.1 Reprezentace modelů . . . 8

2.3.2 Nástroj pro tvorbu modelů – Blender . . . 11

3 Detekce kolizí ve scéně 12

3.1 Rozdělení scény . . . 12

3.1.1 Pravidelná mřížka . . . 12

3.1.2 Oktalový strom (Octree) . . . 13

3.2 Obalová tělesa . . . 14

3.2.1 Osově zarovnaný box (AABB) . . . 15

3.2.2 Obalová koule (Bounding Sphere) . . . 15

3.2.3 Orientovaný obalový box (OBB) . . . 16

3.2.4 Hierarchie obalových těles . . . 16

3.3 Knihovny pro detekce kolizí . . . 17

3.3.1 CollDet . . . 17

3.3.2 OZCollide . . . 17

3.3.3 FreeSOLID . . . 17

4 Plánování cesty v prostoru – pravděpodobnostní algoritmy 18

4.1 Základní pojmy a techniky . . . 19

4.1.1 Vzorkování náhodných bodů z konfiguračního prostoru . . . 19

4.1.2 Spojování dvou konfigurací . . . 20

4.1.3 Zkracování a vyhlazování cesty . . . 21

4.2 PRM algoritmus . . . 22

4.2.1 Tvorba grafu konfiguračního prostoru . . . 22

4.2.2 Vyhledávání cesty . . . 23

4.3 EST algoritmus . . . 25

4.3.1 Tvorba stromu konfiguračního prostoru . . . 25

4.3.2 Napojení stromu . . . 26

4.4 RRT algoritmus . . . 26

4.4.1 Tvorba stromů konfiguračního prostoru 26
4.4.2 Spojení stromů 28
4.5 SRT algoritmus 29
4.5.1 Tvorba grafu konfiguračního prostředí 29
4.5.2 Vyhledávání cesty 29

5 Návrh aplikace **31**
5.1 Konfigurační soubor 31
5.2 Uživatelské rozhraní 32
5.2.1 Menu 32
5.2.2 Scéna 34
5.2.3 Animace výsledné cesty 34
5.3 Formát souborů s objekty 35
5.4 Ovládání aplikace 35
5.5 Implementační jazyk, použité knihovny a algoritmy 36
5.6 Diagram tříd 36

6 Implementace **38**
6.1 Zobrazení 38
6.1.1 Aplikační okno 38
6.1.2 Objekty 39
6.1.3 Scéna 40
6.1.4 Menu 40
6.1.5 Animace cesty 41
6.2 Detekce kolizí – knihovna ColDet 42
6.3 Pravděpodobnostní algoritmus 42
6.3.1 Datové struktury 43
6.3.2 Vzdálenostní a spojovací funkce 45
6.3.3 Fáze 1 – generování kořenů stromů 46
6.3.4 Fáze 2 – expanze uzlů vygenerovaných stromů 46
6.3.5 Fáze 3 – propojení stromů do výsledného grafu 47
6.3.6 Fáze 4 – hledání cesty ve výsledném grafu 49
6.3.7 Fáze 5 – zkracování nalezené cesty 50
6.4 Běh aplikace 51

7 Testování **53**
7.1 Ovládání aplikace 53
7.2 Funkčnost plánovacího algoritmu 53
7.3 Výkonnost plánovacího algoritmu 54

8 Závěr **57**
8.1 Možná rozšíření 58

A Obsah DVD **61**
A.1 Adresářová struktura 61

B Stručný návod k použití **62**

Seznam obrázků

2.1 Ukázka objektů reprezentovaných pomocí sítě trojúhelníků. Obrázky jsou převzaty z [22]. 9
2.2 Jednoduchý model reprezentovaný hranicemi. Obrázek je převzat z [1]. . . . 9
2.3 Nevyrobitelné objekty. První obrázek obsahuje nonmanifold vrchol, druhý nonmanifold hranu a třetí nonmanifold plochu. Obrázky jsou převzaty z [3]. 10
2.4 Ukázka CSG stromu. Obrázek je převzat z [7]. 11

3.1 Pravidelné rozdělení scény. 13
3.2 Rozdělení prostoru na osminy v oktalovém stromu. Celková krychle charakterizuje otcovský uzel oktalového stromu, menší (číslované) krychle jsou potomci tohoto uzlu. 14
3.3 Objekt obalený v osově zarovnaném boxu. 15
3.4 Objekty obalené v obalové kouli. Obrázek byl převzat v [4]. 15
3.5 Objekt obalený v orientovaném obalovém boxu. 16
3.6 Hierarchické obalení tělesa pomocí OBB. Obrázek je převzat z [2]. 16

4.1 Příklad úzkého průchodu. Objekt se musí přesunout z bodu A do bodu B. . 20
4.2 Příklad zkrácení cesty. Červené body značí start a cíl cesty, zelené body jsou nalezené uzly cesty, černé šipky značí hrany mezi uzly v cestě a modrá šipka je hrana, která zkracuje cestu. 21
4.3 Příklad vyhlazení cesty. Červené body značí start a cíl cesty, zelené body jsou nalezené uzly cesty, černé šipky značí hrany mezi uzly v cestě (původní trasu) a modrá šipka je vyhlazení cesty (nová trasa). 22

5.1 Zjednodušený diagram tříd. 37

6.1 Ukázka uživatelského prostředí. 39
6.2 Blokové schéma plánovacího algoritmu. 43
6.3 Scéna, na které bude předváděn algoritmus. 43
6.4 Scéna po simulaci generování kořenů stromů. 47
6.5 Scéna po expanzi uzlů ve stromech. 48
6.6 Scéna po propojení stromů. 49
6.7 Scéna po nalezení a zkrácení cesty. 50

7.1 Ukázky testovacích zdí. Zeď vlevo má velikost 110 %, zeď uprostřed má velikost 107 % a zeď vpravo má velikost 105 %. 54
7.2 Hlavolam „Ježek v kleci“. 55
7.3 Hlavolam „Past na brouka“.Vlevo pohled zepředu, uprostřed pohled z boku a vpravo pohled zezadu. 55

Kapitola 1

Úvod

S plánováním pohybu se člověk setkává denně a všude. Chce-li se člověk přesunout například po místnosti, nebo pokud potřebuje přestěhovat kus nábytku, musí si naplánovat trasu. Lidé plánování cesty při chůzi provádějí většinou intuitivně, což ovšem nemusí platit u zmíněného stěhování nábytku. Pro zjednodušení takových úkonů lze zapojit počítač, kdy člověk definuje problém a počítač vrátí funkční řešení.

Počítače ovšem většinou nejsou schopny řešit složité problémy naivně (kontrola všech možností), kvůli časové složitosti takových algoritmů. Proto bylo navrženo mnoho způsobů, jak navrhnout algoritmy vracející dobré výsledky s menší časovou složitostí. Jedním takovým způsobem je použití pravděpodobnostního hledání pomocí pravděpodobnostních algoritmů, které náhodně volí určitý počet zkoumaných možností. Rozsah vygenerovaných možností (v rámci stavového prostoru řešeného problému) pak přímo ovlivňuje časovou náročnost algoritmu a kvalitu nalezeného řešení.

Cílem diplomové práce je seznámit se s knihovnami DirectX a OpenGL, nastudovat způsoby modelování jednoduchých geometrických objektů v programech jako, je například Blender. Dále nastudovat metody pro detekce kolizí a stěžejní část sestává z nastudování pravděpodobnostních algoritmů pro plánování pohybu. Následně je potřeba provést návrh aplikace splňující všechny požadavky – musí obsahovat uživatelské rozhraní umožňující uživateli vytvořit scénu z dostupných objektů, definovat překážky, startovní objekt a cílovou pozici, nastavit pravděpodobnostní algoritmus a spustit plánování. Aplikace musí být schopna nalezenou cestu vizualizovat. Po provedení návrhu následuje implementace aplikace ve zvoleném programovacím jazyce. K implementaci lze využít volně dostupných knihoven. Nakonec je nutno provést testování implementované aplikace zahrnující tvorbu objektů vhodné pro zvolené experimenty.

Pravděpodobnostní algoritmy jsou algoritmy schopné vyhledávat cestu i ve spojitém prostoru díky náhodnému vzorkování bodů. Oproti jiným přístupům vzorkování (například rozdělení prostoru na pravidelnou mřížku) může tato technika výrazně urychlit vyhledávání cesty, jelikož díky náhodnému vzorkování (například s rovnoměrných rozložením) je možné zvolit počet vzorků v prostoru a pokusit se hledat cestu pouze s těmito vzorky. Budou-li vzorky vhodně rozmístěné (což je zatíženo náhodou), lze nalézt cestu mezi startovní a cílovou pozicí mnohem rychleji, než při procházení velkého množství pravidelně navzorkovaných bodů. Pravděpodobnostní algoritmy tedy přináší možnost hledání cest v kratším čase, ovšem se sníženou pravděpodobností nalezení cesty (záleží na vhodnosti generování bodů v prostoru při vzorkování).

1.1 Skladba práce

Kapitola 2 popisuje základní principy tvorby 3D grafiky, použitelné knihovny (DirectX, OpenGL, SDL) a základní prvky modelování 3D objektů. V kapitole 3 lze nalézt metody detekcí kolizí a opět stručný popis použitelných knihoven. Kapitola 4 obsahuje základní techniky pro pravděpodobnostní plánování cesty objektu v prostoru a popis vybraných pravděpodobnostních algoritmů (konkrétně algoritmy PRM, EST, RRT a SRT). Kapitola 5 obsahuje návrh implementace aplikace, která splňuje zadané požadavky. V kapitole 6 je proveden popis implementace navržené aplikace s podrobným popisem implementace pravděpodobnostního algoritmu použitého pro plánování cesty. Kapitola 7 pak obsahuje navržené experimenty pro testování uživatelského rozhraní aplikace, funkčnosti a výkonnosti implementovaného algoritmu. Dále lze v této kapitole nalézt výsledky provedených experimentů a zhodnocení výsledků.

Kapitola 2

Zobrazení

Pro splnění zadaného úkolu bude potřeba vytvořit uživatelské rozhraní, které bude schopno zobrazit nalezený výsledek, popřípadě průběh výpočtu algoritmu. Je tedy nutné diskutovat možnosti a zvolit vhodné nástroje pro vývoj 3D grafiky a s tím spojené vytvoření aplikačního okna. Tato kapitola tedy obsahuje popis knihoven pro tvorbu grafiky, možnosti vytvoření aplikačního okna a také tvorbu 3D modelů.

2.1 Tvorba 3D grafiky

Následující sekce obsahuje popis knihoven pro tvorbu grafiky. Nejprve je popsána sada knihoven DirectX (se zaměřením na Direct3D) a následně knihovna OpenGL, což jsou nejpoužívanější knihovny pro tvorbu grafiky.

2.1.1 DirectX – Direct3D

DirectX je kolekce softwarových nástrojů pro tvorbu multimediálních a herních aplikací. Tvůrcem DirectX je společnost Microsoft. První verze DirectX byla vydána v roce 1994 jako součást operačního systému Microsoft Windows 95. Od vzniku do současnosti vzniklo mnoho verzí, aktuální verze je DirectX 11 [19]. Kromě posledních dvou verzí (DirectX 10 a DirectX 11) jsou všechny verze zpětně kompatibilní. Jak již bylo zmíněno, DirectX se skládá z několika částí rozdělených podle účelu. Část DirectX Graphics obsahuje knihovny pro práci s grafikou, obsahuje například knihovnu DirectDraw pro 2D rastrovou grafiku, Direct3D pro 3D grafiku a jiné. Další části DirectX slouží například pro přehrávání a záznam zvuku, pro komunikaci přes počítačové sítě, pro využití GPU (graphics processing unit) pro výpočty a podobně [21]. Následující text se zaměřuje na knihovnu Direct3D.

Direct3D, komponenta DirectX, slouží jako rozhraní pro práci s 3D grafikou. Tato knihovna disponuje velkým množstvím operací. Jedná se o operace k vykreslení libovolných objektů (body, čáry, polygony, komplexní objekty, . . .), k úpravě rozlišení výsledného obrazu, pro práci s osvětlením scény, ke změně materiálových vlastností objektů (barva, barevná škála, textura, . . .), pro změny vlastností kamery (rotace, posun, . . .) nebo také operace aplikující algoritmy, které vylepšují vzhled scény (efekt mlhy, antialiasing, . . .). Knihovna Direct3D funguje nezávisle na hardware počítače, nezávislost je zajištěna díky DDI (Device Driver Interface), což jsou ovladače grafické karty nainstalované v systému. Knihovna Direct3D umožňuje urychlovat některé grafické operace pomocí hardware grafické karty, jaké operace lze takto urychlit závisí na konkrétní grafické kartě [21].

Pro tvorbu programů s využitím DirectX (popřípadě Direct3D) je vhodné použít vývojové prostředí Microsoft Visual C++. Volání funkcí z DirectX ovšem předchází správné nastavení vývojového prostředí a vytvoření základní Win32 aplikace. Aby bylo možné spouštět DirectX aplikace, musí být v systému nainstalovaný DirectX *end-user runtime*, jehož verze musí odpovídat verzi použitého DirectX [21].

2.1.2 OpenGL

OpenGL (Open Graphics Library) je knihovna pro práci s 2D a 3D grafikou a slouží jako jednotné API (Application Programming Interface) mezi grafickým hardware a programem. Knihovnu vytvořila společnost SGI (Silicon Graphics Inc.) v roce 1992. Programy vytvořené pomocí OpenGL jsou nezávislé na cílové platformě a použitém programovacím jazyce [20]. OpenGL je otevřeným standardem. Autor specifikace tedy netvoří každou realizaci na určité platformě, pouze dohlíží na dodržování této specifikace. Tuto roli plní konsorcium ARB (Architecture Review Board), složené z vůdčích firem v oblasti vývoje software a hardware. Implementace připadá na výrobce grafických zařízení. Označení OpenGL dostanou pouze implementace, které úspěšně projdou sérií testů, které provádí rovněž sdružení ARB [25].

OpenGL obsahuje asi 250 funkcí, pomocí kterých lze specifikovat objekty ve 2D nebo 3D. Takové objekty poté lze dále zpracovávat. Tvůrce programu, tvořící grafickou scénu pomocí OpenGL, nemusí znát konfiguraci počítače, pouze využívá funkce knihovny OpenGL. Pokud programátor volá funkci nepodporovanou grafickou kartou, provede se softwarová realizace. OpenGL má široké renderovací schopnosti – od zobrazování jednoduchých drátových modelů těles, přes ploché stínování až po komplexní osvětlené scény s mlhou, lesklými či průhlednými stěnami objektů a tak dále. Objekty jsou charakterizovány pomocí takzvané hraniční reprezentace (boundary representation), tedy plošnými útvary na povrchu objektu, ne jeho objemem. Tato reprezentace je výhodnější pro rychlé vykreslování [20].

Pro práci s OpenGL pro C/C++ slouží knihovna *opengl.lib* a hlavičkové soubory *glut.h*, *opengl.h*, *glu.h*, popřípadě *glaux.h* [20]. Pro základní množinu funkcí, datových typů a maker postačuje pouze *opengl.h*, ostatní slouží jako knihovní nadstavby (pomocné knihovny) a obsahují funkce rozšiřující možnosti práce s OpenGL.

Shrnutí

DirectX i OpenGL poskytují širokou škálu možností pro tvorbu 3D grafiky a v obou případech lze vytvořit obdobné grafické scény. Stejně tak lze v obou případech tvořit grafiku, aniž by bylo nutno znát grafický hardware počítače, na kterém bude program spouštěn. Rozdíl je ovšem v možnostech použití výsledného programu na různých operačních systémech. Zatímco DirectX podporuje pouze operační systém Windows, OpenGL umožňuje tvorbu programů pro operační systémy Windows, Linux, Mac OS a další. Hlavně díky své multiplatformnosti bude v této práci k zobrazení grafických scén použita knihovna OpenGL. Další text v této kapitole bude tedy uvažovat pouze nástroje, které jsou schopny pracovat s OpenGL (popřípadě naopak).

2.2 Správa okna – knihovna SDL

Aby bylo možné zobrazovat grafickou scénu, bude potřeba vytvořit aplikační okno. K vytvoření aplikačního okna lze použít na každém operačním systému zabudovaný správce oken (například knihovna WinAPI na Windows). Pro multiplatformní aplikace by ovšem tato

možnost nesla nemalé komplikace, jelikož každý operační systém má odlišný způsob tvorby aplikačního okna. Tento problém řeší knihovna SDL (Simple DirectMedia Layer).

Knihovna SDL umožňuje vytváření multiplatformních aplikačních oken (podporuje operační systémy Windows, Linux, MacOS, ...) [18], dále obsahuje funkce k přehrávání zvuku a videa, ke zpracování vstupů z klávesnice, myši a joysticku, lze také pomocí ní pracovat s 3D grafikou přes OpenGL. K hlavní knihovně SDL existuje několik podpůrných knihoven, které umožňují například načítání různých formátů obrázků (knihovna SDL_image), komplexní funkce pro práci se zvukem (knihovna SDL_mixer), síťové služby (SDL_net) a další [16]. Knihovna SDL ve spojení s OpenGL umožňuje tvorbu multimediálních aplikací a her s grafickým rozhraním.

2.3 Modelování 3D objektů

Modely reprezentují 3D objekty pomocí soustav bodů, které jsou propojeny do geometrických útvarů jako trojúhelníky, úsečky, křivky v prostoru a podobně. Body, ze kterých se objekt skládá, lze rozdělit na body vnitřní a body hraniční. Uvedené geometrické útvary slouží k popisu množiny bodů hraničních. Pro popis vnitřních bodů je potřeba užití trojrozměrných geometrických útvarů, jako krychle, kvádr, válec a podobně [27].

Tato sekce obsahuje informace o vybraných reprezentacích modelů. Podrobněji se zaměřuje na hraniční reprezentaci, jelikož je výhodná z hlediska dalšího zpracování a zobrazování – grafické procesory jsou schopny snadno zobrazit objekty s hraniční reprezentací [27]. Dále lze v této sekci nalézt stručný popis nástroje Blender.

2.3.1 Reprezentace modelů

Následuje popis reprezentace modelů pomocí sítí trojúhelníků, hraniční reprezentace a konstruktivní geometrie těles.

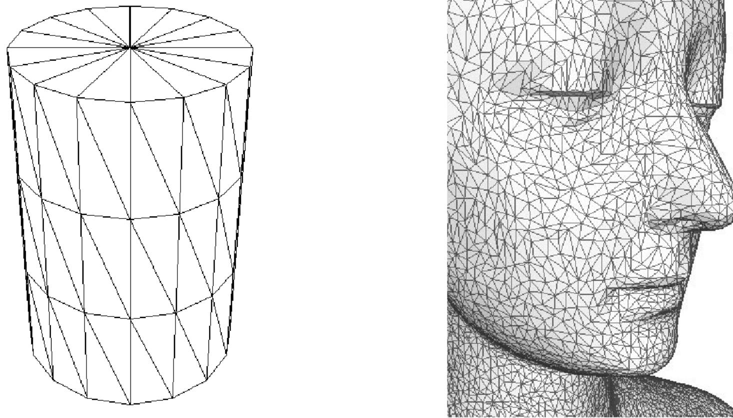
Síť trojúhelníků

Trojúhelník je výhodný geometrický útvar pro popis objektu, jelikož má velmi dobré vlastnosti – všechny jeho vrcholy leží v jedné rovině a je vždy konvexní. Mnoho geometrických výpočtů nad trojúhelníkem lze optimalizovat a jeho zobrazování podporuje grafický hardware [27].

Síť trojúhelníků je množina trojúhelníků sdílejících své hrany. Datový popis sítě trojúhelníků lze rozdělit na dvě logické části – geometrická část a topologická část. Geometrická část obsahuje souřadnice vrcholů trojúhelníků a topologická část zaznamenává informace o tom, které vrcholy tvoří trojúhelník, popřípadě o tom, které trojúhelníky spolu sousedí. Toto rozdělení je výhodné při některých operacích se sítí, kdy postačí použít informace pouze z jedné části popisu. V některých případech, jako například při zpracování dat v grafickém procesoru, není oddělení geometrických a topologických dat výhodné [27].

Na obrázku 2.1 lze vidět příklady objektů reprezentované pomocí sítě trojúhelníků. Mezi výhody sítě trojúhelníků patří:

- lze vytvořit libovolný objekt,
- snadno reprezentovatelná jako soubor vrcholů,
- jednoduše transformovatelná do jiných reprezentací,



Obrázek 2.1: Ukázka objektů reprezentovaných pomocí sítě trojúhelníků. Obrázky jsou převzaty z [22].

- snadné vykreslení na počítači.

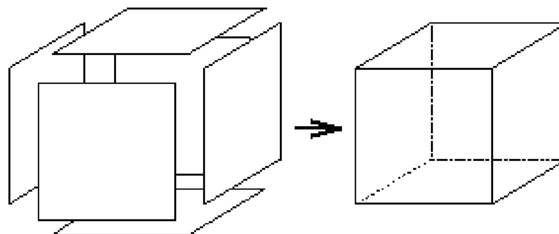
Naopak její nevýhody jsou:

- zakřivené plochy lze pouze aproximovat,
- náročná simulace některých objektů (například vlasy, tekutiny, ...),
- nesnadné mapování textur.

Podrobnější popis a další informace lze získat například v [22, 27].

Hraniční reprezentace

Popis hranice (boundary representation, B-rep) je jeden z nejběžnějších způsobů reprezentace objektů. Jedná se o popis množiny hraničních bodů, vnitřní body nejsou v této reprezentaci uchovávány (popřípadě je lze odvodit z popisu hranice) [27]. Obrázek 2.2 ukazuje příklad hraniční reprezentace krychle, která je složena z šesti čtverců.

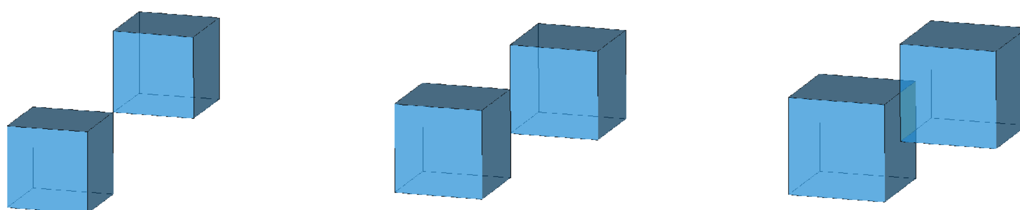


Obrázek 2.2: Jednoduchý model reprezentovaný hranicemi. Obrázek je převzat z [1].

Hranice jsou definovány pomocí takzvaných plošek (strukturovaná plošková reprezentace), které mohou být rovinné, kvadratické, toroidní nebo tvarované povrchy. Objekt je reprezentován jako množina plošek spolu s topologickými informacemi, které definují vzájemné propojení plošek. Propojení plošek u hraniční reprezentace splňuje, že výsledný objekt je uzavřený v 3D prostoru (tedy že plošky zahrnují všechny hraniční body objektu) [1].

Definicí objektu, jakožto složeninu vnitřních a hraničních bodů, lze s výhodami použít například při modelování tvaru tělesa v systémech CAD. Z praktického hlediska však lze popsat objekty, které není možné vyrobit. Z tohoto důvodu se zavádí pojem **manifold**. Manifold je takový model objektu, který odpovídá nějakému skutečnému objektu (lze jej tedy vyrobit) [27]. Naopak pojem **nonmanifold** reprezentuje nevyrobitelný objekt. Na obrázku 2.3 lze vidět příklady nevyrobitelných objektů – nonmanifoldů. Objekt nelze vyrobit, pokud obsahuje:

- Vrchol, který spojuje dvě části tělesa.
- Hranu, která náleží více než dvěma plochám.
- Plochy, které se navzájem dotýkají a jiné [3, 27].

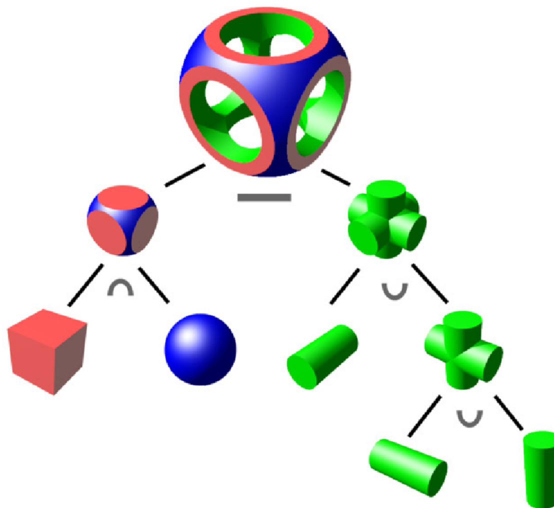


Obrázek 2.3: Nevyrobitelné objekty. První obrázek obsahuje nonmanifold vrchol, druhý nonmanifold hranu a třetí nonmanifold plochu. Obrázky jsou převzaty z [3].

K popisu hraniční reprezentace lze kromě strukturované ploškové reprezentace využít také popis pomocí základních prostorových prvků, jako jsou body, úsečky a části rovinných ploch (jako vrcholy, hrany a stěny). Při popisu pomocí vrcholů, hran a stěn se musí uchovávat různé příznaky a prvky musí být uspořádány do hierarchických struktur, to vše kvůli požadavkům zobrazovacích algoritmů [27]. Další způsob popisu povrchu tělesa (nejstarší a nejjednodušší) je hranová reprezentace, kdy k popisu slouží pouze hrany objektu. Objekty popsané hranovou reprezentací připomínají prostorové drátové modely, proto jsou někdy jako drátové modely označovány. Hranová reprezentace není jednoznačná, jelikož popis neobsahuje dostatek topologických informací (pouze seznam vrcholů a seznam hran). Další způsoby popisu (bodová reprezentace, jednoduchá plošková reprezentace . . .) nejsou v této práci uvedeny, lze je nalézt například v [27].

Konstruktivní geometrie těles

Metoda konstruktivní geometrie těles (CSG, Constructive Solid Geometry) odráží postupy prováděné konstruktérem při návrhu tvaru tělesa, především v CAD systémech. Jedná se o reprezentaci tělesa založenou na stromové struktuře (CSG stromu), ve kterém jsou uchovány dílčí kroky tvorby tělesa. Na obrázku 2.4 lze vidět ukázku CSG stromu. Listové uzly stromu obsahují takzvané CSG primitiva, což jsou základní geometrické objekty (kvádr, koule, válec, kužel, popřípadě i složitější entity jako plocha NURBS a podobně). Nelistové uzly stromu charakterizují množinové operace a prostorové transformace prováděné nad zmíněnými objekty. Používané množinové operace jsou sjednocení, průnik a rozdíl, prostorovými transformace se myslí úprava velikosti primitiv, jejich natočení a podobně [27].



Obrázek 2.4: Ukázka CSG stromu. Obrázek je převzat z [7].

Tato reprezentace byla vytvořena především pro fázi vytváření a tvarování tělesa. Zobrazování objektu v CSG reprezentaci není jednoduché, jelikož nejsou k dispozici přímo vykreslitelné geometrické prvky (hrany, plochy, . . .). Z hlediska rychlosti běhu aplikace je výhodnější těleso převést do jiné reprezentace (například hraniční reprezentace), kde zobrazení není tak časově náročné. Další informace o CSG reprezentaci lze nalézt v [27].

2.3.2 Nástroj pro tvorbu modelů – Blender

Blender je integrovaná sada nástrojů umožňující tvořit v široké oblasti 3D grafiky. Jedná se o open-source software, jehož zdrojové kódy spadají pod licenci GNU GPL. Blender umožňuje uživatelům pracovat na všech majoritních operačních systémech a k opětovnému spuštění vytvořeného obsahu není potřeba přítomnost Blenderu na počítači. Blender obsahuje rozhraní založené na OpenGL. Instalační soubory je možné stáhnout z oficiálních webových stránek programu Blender [6] a také zde lze získat další užitečné informace.

Klíčovými rysy Blenderu jsou:

- Široká paleta nástrojů pro práci s 3D grafikou – modelování, animování, rendering, simulování, tvorbu her a další.
- Multiplatformní aplikace – umožňuje práci na Windows, Linux, OSX, FreeBSD a další.
- Podpora zdarma na webových stránkách [6].
- Celosvětová komunita uživatelů.

Kapitola 3

Detekce kolizí ve scéně

Detekce kolizí slouží k nalezení kolidujících těles ve scéně. Scéna se skládá ze statických a dynamických těles. Pomocí detekce kolizí a reakce na kolizi lze zajistit, že dynamické tělesa neprotínají žádné statické tělesa, popřípadě nekolidují dvě nebo více dynamických těles. Pokud je kolize mezi dvěma tělesy detekována, spustí se systém reakce na kolizi. Detekce kolizí je obecně velmi náročný problém. Čím více těles scéna obsahuje, tím delší výpočet detekce kolizí zahrnuje [12]. Detekce kolizí musí být co nejrychlejší, jelikož je spouštěna před každým vykreslením scény (rychlost vykreslování scény je přímo závislá na rychlosti detekce kolizí).

Detekce kolizí probíhá ve dvou fázích. V první fázi se zjišťují dvojice objektů, které spolu mohou kolidovat. V této fázi se scéna rozdělí na části za použití struktur k tomu určeným (pravidelná mřížka, segmentový nebo intervalový strom a podobně). Ke zjednodušení těles (mohou mít velmi složité tvary) se používají takzvané obálky. Pomocí obálek lze tvar tělesa zjednodušit a tím urychlit hledání kolizí [12, 26].

Ve druhé fázi se provádí hledání objektů (za použití dvojic objektů nalezených v první fázi), které spolu kolidují. Jelikož se v této fázi zjišťují už pouze kolize dvou objektů, provádí se test trojúhelník vůči trojúhelníku [12, 26].

V této kapitole jsou popsány některé struktury pro rozdělení scény a také různé druhy obalových těles. Nakonec jsou uvedeny existující knihovny, které lze použít k implementaci detekci kolizí.

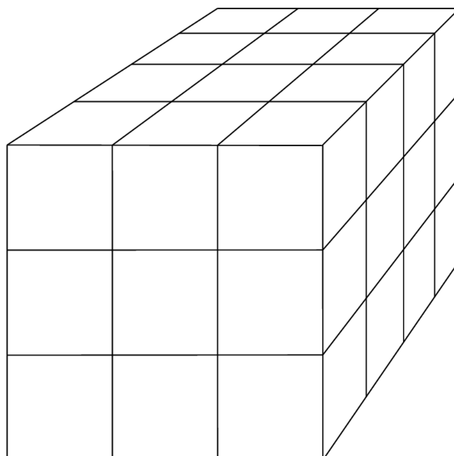
3.1 Rozdělení scény

Jedná se o systematické rozdělení scény, které umožňuje zrychlit následnou detekci. Zrychlení spočívá ve vyloučení takových dvojic objektů, které nemohou kolidovat. Zpravidla se zkoumá poloha objektů v prostoru.

Tato sekce popisuje některé možné postupy rozdělení scény, pro podrobnější popis lze použít například [12].

3.1.1 Pravidelná mřížka

Rozdělení scény na pravidelnou mřížku je jedna z nejjednodušších metod. Jedná se o rozdělení prostoru na pravidelné krychle (ukázka na obrázku 3.1) a to jak horizontálně, tak vertikálně. Jemnost dělení určuje počet jednotlivých krychlí, na které bude scéna rozdělena. Do paměti jsou uloženy všechny krychle a do krychlí se vkládají objekty přítomné v daném prostoru (prostoru, který náleží krychli).



Obrázek 3.1: Pravidelné rozdělení scény.

Tato metoda není efektivní z hlediska velikosti potřebné paměti, jelikož jsou ukládány i prázdné krychle. Rozdělení scény na pravidelnou mřížku lze použít spíše v případě použití scén s rovnoměrným rozložením objektů v celém prostoru. V případě nerovnoměrného rozložení objektů ve scéně může nastat situace, kdy se většina objektů nachází v jednom místě a jsou tedy přiřazeny do jedné krychle. V tomto případě ztrácí rozdělení scény smysl, jelikož by pro další fázi detekce byla vybrána většina objektů [26].

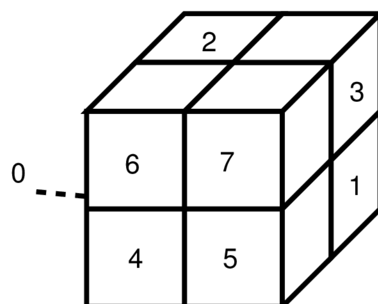
Dalším problémem této metody je práce s objekty s velkými rozdíly ve velikosti. Dynamický objekt s mnohem větší velikostí než velikost buňky mřížky bude zařazen do mnoha buněk a následně bude náročné aktualizovat stav mřížky pro jeho novou polohu. Jako řešení lze použít zarovnání velikosti buňky na velikost největšího objektu, ovšem v tomto případě klesá efektivita, jelikož malé objekty budou rozmístěny v méně buňkách a budou tedy častěji zkoumány i v případě, kdy jsou od sebe dostatečně vzdáleny (při menší velikosti buňky by byly od sebe separovány). Tento problém řeší takzvané hierarchické mřížky. Hierarchické mřížky používají několik úrovní velikostí buněk a jsou svou strukturou podobné stromu [12].

3.1.2 Oktalový strom (Octree)

Jedná se o typický způsob rozdělení prostoru s použitím stromu. K dělení prostoru slouží hierarchicky uspořádané osově zarovnané krychle. Kořen stromu charakterizuje celou scénu (největší krychli) a má osm potomků (nazývané buňky, osminy a podobně), kteří charakterizují vždy osminu velikosti kořenové krychle (viz obrázek 3.2). Stejným způsobem jsou děleny i synovské uzly kořene a rekurzivně také synovské uzly těchto uzlů a tak dále. Toto dělení pokračuje do předem zadané maximální hloubky stromu nebo předem zadané minimální velikosti krychle [12].

Zjednodušením oktalového stromu pro dva rozměry je takzvaný quadtree. Charakteristika quadtree odpovídá octree, liší se pouze počet potomků každého uzlu (u quadtree jde o 4 potomky) [12].

K rozdělení scény existuje mnoho různých metod, ať už stromových nebo jiných. Popsané metody patří mezi nejpoužívanější z důvodu jejich jednoduchosti, další metody v této práci nejsou popisovaný, ale lze je nalézt v literatuře [12].



Obrázek 3.2: Rozdělení prostoru na osminy v oktalovém stromu. Celková krychle charakterizuje otcovský uzel oktalového stromu, menší (číslované) krychle jsou potomci tohoto uzlu.

3.2 Obalová tělesa

Přímé testování kolize dvou objektů ve scéně by mohlo být velmi náročné, jelikož tělesa mohou být velmi složitá – skládající se z mnoha polygonů. Z tohoto důvodu se používají takzvaná obalová tělesa (obálky), která zjednoduší tvar objektu (obalí ho) a tím se významně urychlí výpočet. Při detekci kolizí se nejdříve zkontrolují kolize obálek objektů a až když kolidují dvě obálky, započne kontrola kolize skutečného tvaru objektu. Tento způsob výrazně urychluje detekci kolizí především u objektů, které nekolidují. Může nastat situace, kdy ačkoliv spolu objekty nekolidují, jejich obálky v kolizi jsou. V tomto případě nedochází k urychlení kontroly kolizí, jelikož se musí zkontrolovat i kolize skutečných tvarů objektů. Ke zmenšení počtu takových situací lze použít lepších obalových těles, které lépe vystihnou tvar objektu [12].

Jako obalová tělesa se používají různá geometrická tělesa. Ne všechna tělesa jsou k tomu vhodná, požadované vlastnosti jsou:

- snadno zjistitelný test průniku,
- těsně přiléhající,
- snadné pro výpočty,
- nenáročné na místo v paměti.

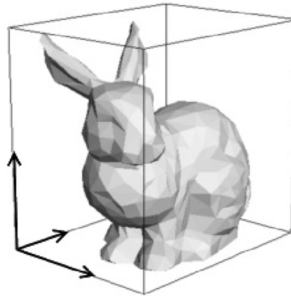
Testování kolizí obalových těles (tedy testování průniku těchto těles) probíhá v předzpracování samotnému hledání kolizí mezi objekty, vyžaduje se tedy co nejrychlejší provedení. Rychlost testu na průnik musí být co nejmenší. Tuto rychlost ovšem přímo ovlivňuje požadavek na těsnou přiléhavost, jelikož čím lépe obálka přiléhá na objekt, tím je geometricky složitější a roste tedy i rychlost zjištění průniku mezi obálkami [12]. Je nutné nalézt kompromis mezi rychlostí nalezení průniku a kvality přiléhavosti obálky.

U dynamických těles probíhá přepočítávání pozice, popřípadě tvaru obálky v každém zobrazovacím cyklu programu. Čím delší tento výpočet bude, tím se snižuje zobrazovací rychlost programu. Je tedy potřeba vybírat tvary obálek, které umožňují rychlé výpočty při tvorbě obálky (při obalování objektu). Velikost paměťového místa, které je potřeba pro uložení obálky do paměti, závisí rovněž na složitosti geometrického tvaru obálky [12]. Je tedy nutné volit vhodné geometrické tvary obálek i z důvodu šetření paměti.

Následuje popis nejpoužívanějších obalových těles, nejprve popis osově zarovnaného boxu, poté obalové koule a nakonec orientovaného boxu. Další druhy obalových těles, popřípadě detailnější popis zde zmíněných, lze nalézt v [12].

3.2.1 Osově zarovnaný box (AABB)

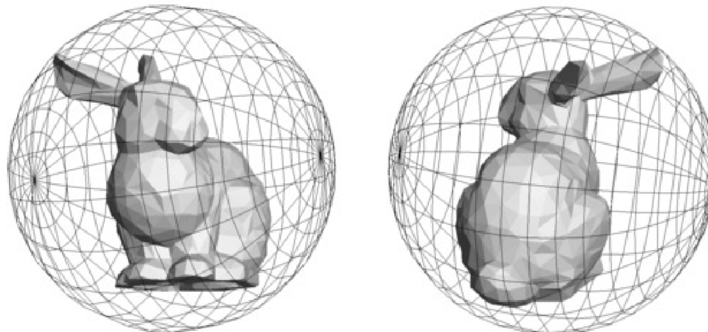
Osově zarovnaný box používá k obalení objektu kvádr (ve 2D prostoru obdélník), který má hrany rovnoběžné se souřadnými osami (viz obrázek 3.3). Jedná se o jeden z nejpoužívanějších obalových těles. Výhodou osově zarovnaného boxu je jednoduchost testování kolize mezi dvěma kvádry, lze provést pouze šest porovnání [12].



Obrázek 3.3: Objekt obalený v osově zarovnaném boxu.

3.2.2 Obalová koule (Bounding Sphere)

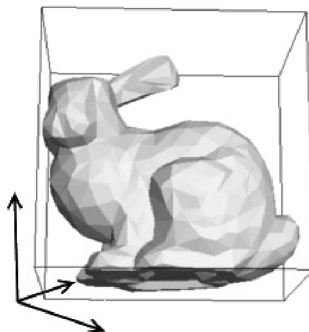
K obalení objektu je využita koule (viz obrázek 3.4) – velmi jednoduchý způsob, kdy k reprezentaci stačí pouze střed koule a její poloměr. Obalová koule je další velmi často používané obalové těleso. Stejně jako osově zarovnaný box má obalová koule velmi snadný test na průnik – porovnáním vzdálenosti středů koulí se součtem jejich poloměrů. Obalová koule je navíc invariantní vůči rotaci, což usnadňuje přepočítávání obálky pro dynamické objekty. Naproti tomu má obalová koule velký volný prostor (obepíná těleso volně) [12].



Obrázek 3.4: Objekty obalené v obalové kouli. Obrázek byl převzat v [4].

3.2.3 Orientovaný obalový box (OBB)

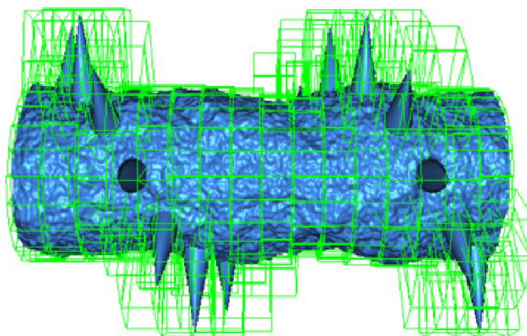
Další obalové těleso, které používá k obalení kvádr. Na rozdíl od osově zarovnaného boxu nemá orientovaný obalový box hrany rovnoběžné se souřadnými osami, ale je natočený podle objektu, který obaluje (viz obrázek 3.5). Orientované obalové těleso aproximuje objekty lépe, než předešlá obalová tělesa. Lze jej reprezentovat několika způsoby (například šest ploch, osm hran a podobně). Rychlost výpočtu testu průniku je závislá na reprezentaci, obecně je však časově náročnější, než u osově zarovnaného boxu nebo obalové koule, jelikož je potřeba určit směry os obaleného objektu [12].



Obrázek 3.5: Objekt obalený v orientovaném obalovém boxu.

3.2.4 Hierarchie obalových těles

U složitějších objektů se používají hierarchie obalových těles, což jsou například stromy jednoduchých obalových těles (AABB, OBB, ...). Tyto stromy obsahují v listových uzlech základní obalová tělesa a nelistové uzly charakterizují sjednocení obalových těles v synovských uzlech. Kořen stromu pak obsahuje kompletní obalové těleso daného objektu. Příklad hierarchického obalení tělesa je vidět na obrázku 3.6. Algoritmy k vytvoření hierarchií a jejich podrobnější popis lze nalézt v [12].



Obrázek 3.6: Hierarchické obalení tělesa pomocí OBB. Obrázek je převzat z [2].

3.3 Knihovny pro detekce kolizí

Pro implementaci detekce kolizí existuje množství knihoven. Následuje popis několika vybraných. Všechny níže uvedené knihovny jsou implementovány v jazyce C++ a jsou volně dostupné k použití.

3.3.1 CollDet

Knihovna CollDet implementuje detekce kolizí mezi dvěma polyhedry[1]. Hlavní účel této knihovny je použití ve 3D hrách, ve kterých je potřeba přesně hledat kolize mezi dvěma složitými objekty. Knihovnu lze stáhnout na webové stránce [24], kde lze nalézt i dokumentaci.

3.3.2 OZCollide

Jedná se o rychlou knihovnu implementující detekce kolizí, která nabízí jednoduchou práci s API, jehož výkonnost a funkcionalitu lze použít zejména na aplikace v reálném čase. Knihovna OZCollide nabízí multiplatformní nástroj, který umožňuje detekovat kolize mezi různými druhy entit (například koule, elipsoidy, AABB, OBB, trojúhelník a podobně). Kromě binárního výstupu lze požadovat také úplný seznam kolidujících objektů. Knihovnu lze stáhnout na webové stránce [13], kde lze nalézt i dokumentaci a další informace.

3.3.3 FreeSOLID

Knihovna pro detekci kolizí 3D objektů, které se mohou pohybovat nebo deformovat. FreeSOLID byla navržena pro použití v interaktivních grafických aplikacích. Knihovna obsahuje mnoho funkcí, mezi nimi například umožňuje práci se stejným maticovým popisem pohybu, jako tomu je v případě knihovny OpenGL (matice 4×4 obsahující posuv, rotaci a přiblížení objektu). Knihovnu lze stáhnout na webové stránce [9], kde lze nalézt i dokumentaci a další informace.

[1] Polyhedra jsou geometrická tělesa složená z rovinných ploch s rovnými hranami.

Kapitola 4

Plánování cesty v prostoru – pravděpodobnostní algoritmy

Plánováním cesty, ať už obecně objektu v rámci této práce nebo například robota v nějakém prostředí, se rozumí nalezení posloupnosti kroků (pohybů, rotací), kterými lze daný objekt bezkolizně dopravit ze startovní pozice na pozici cílovou. Jak už bylo zmíněno, plánování cesty probíhá v nějakém prostředí. Toto prostředí může být spojité nebo diskrétní. V případě diskrétního prostředí lze nalézt cestu poměrně snadno pomocí deterministických algoritmů (diskrétní prostředí lze popsat grafem). V druhém případě (u spojitých prostředí), již není hledání cesty nijak snadným problémem, jako vhodný postup se nabízí převést prostředí na diskrétní (navzorkovat) a poté použít algoritmy nad grafem.

Pravděpodobnostní algoritmy pracují se spojitým prostředím, které nejprve navzorkují a poté hledají cestu nad grafem diskrétního prostředí. Vzorkování probíhá pomocí generování náhodných bodů v prostoru. Nagenerované body jsou pospojovány do výsledného grafu, ve kterém následně probíhá hledání cesty. Aby bylo možné hledat cestu ve spojitém prostředí pomocí pravděpodobnostních algoritmů, musí být toto prostředí předem známé, tedy existuje nějaký popis tohoto prostředí (model prostředí). Pokud tato podmínka není splněna, nelze pravděpodobnostních algoritmů pro hledání cesty použít [17]. Pomocí pravděpodobnostních algoritmů je možné hledat cesty objektů i ve vícedimenzionálním prostoru.

Pravděpodobnostní algoritmy lze rozdělit na (podle možnosti opakovaného využití vytvořeného grafu prostředí na různé startovní a cílové body):

- *Vícedotazové* – vytvořený graf prostředí lze použít opakovaně. Lze tedy hledat cesty mezi libovolnými dvěma body bez opětovného generování grafu prostředí.
- *Jednodotazové* – nelze použít vygenerovaný graf vícekrát, jelikož grafem je strom vytvořený mezi startovním a cílovým bodem.
- *Kombinované* – kombinuje oba předchozí přístupy. Ke generování grafu jsou využity i jednokrokové algoritmy, avšak vygenerovaný graf lze znova využít pro hledání cesty mezi různými body.

Jednodotazové pravděpodobnostní algoritmy jsou obecně rychlejší, ovšem pouze pro hledání jedné cesty v prostoru. Vícedotazové pravděpodobnostní algoritmy mají své využití v případě, že je potřeba hledat více cest v jednom prostředí, jelikož jim generování grafu

zabere více času. Při opakovaném hledání cest (mezi libovolnými body v grafu prostředí) je potom rychlost lepší, než u jednodotazových algoritmů [10].

Následuje popis používaných pojmů a technik a dále popis několika pravděpodobnostních algoritmů. Konkrétně algoritmu PRM (Probabilistic roadmaps), EST (Expansive-Spaces Trees), RRT (Rapidly-Exploring Random Trees) a SRT (Sampling-Based Roadmap of Trees). Další informace o zmíněných, popřípadě dalších pravděpodobnostních algoritmech, lze nalézt v literatuře [10, 14, 17] nebo na webové stránce [15], kde se nachází také ukázky funkčnosti zmíněných pravděpodobnostních algoritmů.

4.1 Základní pojmy a techniky

Tato sekce obsahuje popis důležitých pojmů při popisu pravděpodobnostních algoritmů a používaných technik při tvorbě a práci s grafem prostředí. Zmíněné pojmy jsou definovány vzhledem k použití při popisu pravděpodobnostních algoritmů.

Konfigurace objektu – stav objektu, který popisuje nějaké umístění objektu v prostoru a jeho natočení vzhledem k souřadným osám. Konfigurace objektu tedy obsahuje tři souřadnice určující polohu v prostoru a tři úhly, definující natočení objektu.

Volná konfigurace – konfigurace, ve které objekt nekoliduje s žádnou překážkou.

Konfigurační prostor C – všechny možné konfigurace objektu.

Volný konfigurační prostor C_{free} – všechny volné konfigurace objektu v C.

Cesta – posloupnost volných konfigurací, spojujících startovní a cílovou pozici.

Vzdálenostní funkce $dist : C \times C \to \mathcal{R}_0^+$ – funkce určující vzdálenost mezi dvěma uzly grafu konfiguračního prostředí. Základní vzdálenostní funkce počítá euklidovskou vzdálenost mezi dvěma uzly v prostoru, tedy

$$dist(q_1, q_2) = |emb(q_1) - emb(q_2)|$$

kde $emb()$ je euklidovská vzdálenost bodu od počátku souřadné soustavy [10].

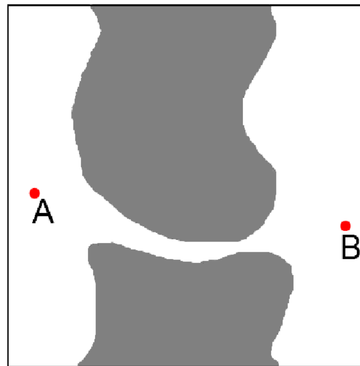
Spojovací funkce Δ – funkce tvořící hrany grafu konfiguračního prostředí spojováním dvou konfigurací. Vstupem jsou dvě konfigurace z volného konfiguračního prostoru a vrací NIL nebo nalezenou cestu mezi zadanými konfiguracemi, která nekoliduje s žádnou překážkou v prostoru. Spojovací funkce může spojit dva uzly například úsečkou nebo křivkou. Vlastnosti spojovacích funkcí jsou popsány v sekci 4.1.2.

4.1.1 Vzorkování náhodných bodů z konfiguračního prostoru

Jedná se o první fázi tvorby grafu konfiguračního prostředí, konkrétně o výběr uzlů grafu. Každý uzel grafu konfiguračního prostředí je jedna konfigurace. Na výběru uzlů závisí následné pokrytí prostoru – možnosti, kam se může objekt přesunout, popřípadě jak se může natočit.

Konfigurace lze náhodně generovat například s rovnoměrným rozložením pravděpodobnosti. Pro každou dimenzi (pro všechny pozice na souřadných osách i pro všechny směry rotace) musí být generována právě jedna hodnota v povoleném rozsahu. Rozsah generování omezují okraje prostoru (nelze pracovat s neomezenými prostory), popřípadě limity na rotace objektu. Vzorkování s rovnoměrným rozložením pravděpodobnosti je použitelné na

všechny prostory, ovšem lze nalézt prostory, kde při použití rovnoměrného rozložení pravděpodobnosti nebude vzorkování příliš efektivní. Jedná se o prostory obsahující takzvané úzké průchody – oblasti mezi překážkami, ve kterých je málo místa vzhledem k velikosti přesouvaného objektu (viz obrázek 4.1). Pokud musí objekt pro dosažení cíle takovým místem projít, může být rovnoměrné rozložení pravděpodobnosti nedostatečné, jelikož nemusí nalézt vhodné uzly, aby šly vytvořit vhodné hrany grafu (hrany, které nekolidují s překážkami a zároveň přesunou objekt úzkým průchodem). [14].



Obrázek 4.1: Příklad úzkého průchodu. Objekt se musí přesunout z bodu A do bodu B.

Pro řešení neefektivnosti vzorkování rovnoměrným rozložením pravděpodobnosti se používají filtrovací nebo retrakční algoritmy. Filtrovací algoritmy zachycují volný prostor náhodnými vzorky v okolí úzkých průchodů pomocí různých technik. Retrakční algoritmy zmenšují překážky, tím zvětšují průchody – zlepšují pravděpodobnost navzorkování míst mezi překážkami a po navzorkování upraví graf prostředí podle reálné velikosti překážek [10, 14].

4.1.2 Spojování dvou konfigurací

Jedná se o druhou fázi tvorby grafu konfiguračního prostředí, konkrétně tvorbu hran mezi uzly vybranými v první fázi. Vytvořené hrany nesmí kolidovat s překážkami, jelikož následně mohou sloužit jako část cesty. Musí být možné, aby objekt mohl uskutečnit bezkolizní posun po trase, kterou každá hrana charakterizuje.

Pro vytvoření hran je potřeba nejprve zjistit ke každému uzlu n nejbližších uzlů (uzlů s nejmenší vzdáleností). K určení vzdálenosti mezi dvěma uzly slouží vzdálenostní funkce. Pro nalezení nejbližších bodů lze použít:

- *Prosté měření vzdáleností* – zjištění vzdálenosti od každého uzlu do všech ostatních uzlů a ukládání n nejbližších bodů. Tato metoda je velmi neefektivní pro větší počty uzlů, popřípadě větší počty dimenzí.

- *Kd-strom* – sestavení stromu dělením grafu konfiguračního prostředí na poloviny se stejným počtem uzlů, dokud není rozdělen na jednotlivé uzly. Výsledkem je binární graf, jehož listové uzly jsou uzly grafu a nelistové uzly charakterizují dělící plochy v prostoru. Kd-strom se po sestavení uloží do paměti a následně lze provádět vyhledávání nejbližších uzlů pro každý uzel grafu konfiguračního prostředí [10, 14].

Po nalezení nejbližších uzlů lze tvořit hrany mezi těmito uzly. K tomu slouží spojovací funkce. Spojovací funkce musí být výkonná, tedy schopná účinně nalézt cestu mezi dvěmi

konfiguracemi, ale také efektivní. Použití úsečky pro spojení dvou konfigurací je časově efektivní, ale ne příliš výkonné. V případě použití křivek lze konstatovat dobrou výkonnost za cenu horší efektivity [10]. Je tedy nutné nalézt kompromis mezi těmito vlastnostmi na základě konkrétního problému.

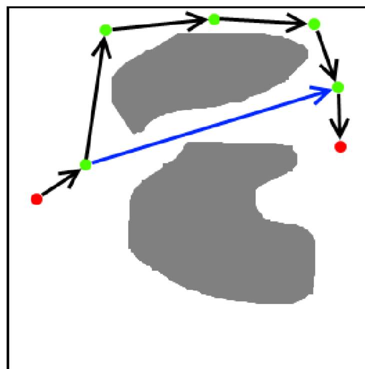
Následuje popis dalších vlastností spojovacích funkcí. Symetrická spojovací funkce zaručuje, že najde stejnou cestu z bodu A do bodu B, jako z bodu B do bodu A. Deterministická spojovací funkce nalezne pro každé hledání cesty stejnou cestu mezi dvěma body. Spojovací funkce interpoluje všechny konfigurace – vzorkuje spojnici mezi konfiguracemi a kontroluje, zda koliduje s nějakou překážkou. Lze použít různé způsoby implementace:

- *Inkrementální interpolace* – spojnice je rozdělena na části podle zadaného kroku a hranice mezi částmi jsou kontrolovány na kolize s překážkami.
- *Interpolace rekurzivním dělením* – spojnice je rozdělena na dvě části. Bod uprostřed spojnice je zkontrolován na kolize s překážkami, pokud nekoliduje, rozdělí se obě části znova na poloviny a vše se rekurzivně opakuje, dokud se nenarazí na kolizi nebo rozdělené části nejsou menší, než hodnota zadaného kroku.

Důležitým faktorem je volba délky kroku, který by měl být co nejmenší, ovšem s ohledem na výpočetní náročnost algoritmu. Velká délka kroku umožňuje rychlé provádění algoritmu, ale hrozí nedetekování kolize mezi překážkou a spojnicí vzorkovanou větší délkou kroku, než je velikost překážky [10]. Opět je nutné hledat kompromis mezi výkonem a minimální velikostí překážek, se kterými dokáže algoritmus pracovat.

4.1.3 Zkracování a vyhlazování cesty

Po vytvoření grafu konfiguračního prostředí a nalezení cesty od startovního do cílového uzlu lze tuto cestu dále upravovat (zlepšovat). Úprava cesty zpomaluje samotný pravděpodobnostní algoritmus, je tedy vhodnější provádět úpravy až po skončení běhu pravděpodobnostního algoritmu, než přímo při jeho běhu.

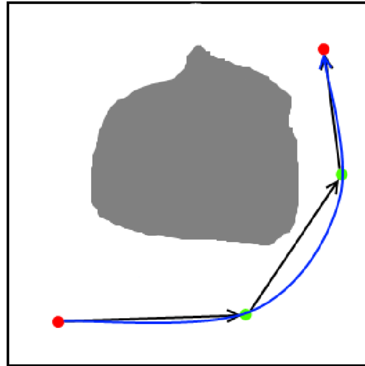


Obrázek 4.2: Příklad zkrácení cesty. Červené body značí start a cíl cesty, zelené body jsou nalezené uzly cesty, černé šipky značí hrany mezi uzly v cestě a modrá šipka je hrana, která zkracuje cestu.

Po nalezení cesty lze zkoumat, zda nejde zkrátit. Zkrácením se chápe spojení dvou uzlů (pouze uzlů, které obsahuje cesta), které je možné spojit (spojnice nekoliduje s žádnou překážkou), ale v průběhu algoritmu spojeny nebyly, jelikož nebyly klasifikovány jako pro sebe navzájem nejbližší uzly (ukázka viz obrázek 4.2). Ke zkracování lze použít *hladový přístup*.

Hladový přístup funguje následovně – kontroluje, zda je možné spojit startovní a cílový bod přímo, pokud nelze, zkontroluje startovní s předposledním (prvním před cílovým) a tak dále. Pokud nelze spojit počáteční uzel s žádným uzlem na cestě, pokračuje na sousední uzel (druhý od začátku cesty) a opakuje hledání spojnic od cílového uzlu. Takto projde celou cestu. Pokud nalezne možnost zkrácení cesty, provede propojení uzlů, kterých se to týká a odstraní z cesty všechny uzly mezi spojovanými uzly [10].

Další možností je nalezenou cestu vyhladit (ukázka viz obrázek 4.3). V tomto případě lze použít uzly cesty jako řídící body pro spline a poté nalezenou křivku testovat na kolize s překážkami v prostoru.



Obrázek 4.3: Příklad vyhlazení cesty. Červené body značí start a cíl cesty, zelené body jsou nalezené uzly cesty, černé šipky značí hrany mezi uzly v cestě (původní trasu) a modrá šipka je vyhlazení cesty (nová trasa).

4.2 PRM algoritmus

PRM algoritmus je jedním ze základních pravděpodobnostních algoritmů pro hledání cesty ve vícedimenzionálním prostoru. Jedná se o vícedotazový pravděpodobnostní algoritmus, který pracuje ve dvou krocích. V prvním kroku se buduje graf prostředí (roadmapa). Ve druhém kroku lze poté vyhledávat cesty mezi libovolnými uzly grafu konfiguračního prostoru, který byl vytvořen v prvním kroku.

Algoritmus PRM byl nastudován z [10] (kapitola 7.1) a z [14] (kapitola 5).

4.2.1 Tvorba grafu konfiguračního prostoru

V první fázi algoritmu PRM se vytváří neorientovaný graf $G = (V, E)$ volného konfiguračního prostoru. V je konečná množina uzlů, které charakterizují volné konfigurace objektu v daném prostoru. Množina E obsahuje dvojice (u, v), charakterizující hrany v grafu a platí $u, v \in V$. Uzly jsou generovány náhodně z volného konfiguračního prostoru. Hrany jsou tvořeny z vygenerovaných uzlů pomocí spojovací funkce Δ (viz sekce 4.1), která ke spojování využívá úsečky.

K vytvoření grafu lze použít algoritmus 4.1. Vstupem algoritmu jsou dvě hodnoty, počet uzlů výsledného grafu n a počet nejbližších uzlů k, které budou hledány pro každý uzel. Nejprve jsou nainicializovány množiny V a E. Následuje vygenerování n volných konfigurací (uzlů), generování probíhá náhodně. Dále jsou tvořeny hrany mezi nagenerovanými uzly a to nalezením k nejbližších sousedů s každým uzlem (za použití vzdálenostní funkce) a spojením

uzlů se svými sousedy pomocí spojovací funkce. Spojovací funkce spojí dva uzly úsečkou (pokud nekoliduje s žádnou překážkou).

Algoritmus 4.1 Algoritmus pro vytvoření grafu volného konfiguračního prostoru.

Vstup: dvojice (n, k), kde
  n – počet uzlů výsledného grafu
  k – počet ukládaných nejbližších sousedů každého uzlu
Výstup: neorientovaný graf $G = (V, E)$

1. $V \leftarrow \emptyset$
2. $E \leftarrow \emptyset$
3. **while** $|V| < n$ **do**
4.   **repeat**
5. $q \leftarrow$ náhodná konfigurace z C
6.   **until** q je volná konfigurace
7. $V \leftarrow V \cup \{q\}$
8. **end while**
9. **for all** $q \in V$ **do**
10. $N_q \leftarrow k$ nejbližších sousedů podle vzdálenostní funkce $dist$
11.   **for all** $q' \in N_q$ **do**
12.   **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq NIL$ **then**
13. $E \leftarrow E \cup \{(q, q')\}$
14.   **end if**
15.   **end for**
16. **end for**

Vstupní parametry je potřeba volit vhodně s ohledem na rychlost výpočtu a úspěšnost nalezení cesty. S malým počtem uzlů v grafu n nemusí být volný konfigurační prostor dostatečně pokryt, aby bylo možné nalézt cestu mezi startovním a cílovým uzlem, na druhou stranu příliš mnoho uzlů v grafu může zpomalit tvorbu grafu. Obdobné je to i s počtem sousedů každého uzlu k.

4.2.2 Vyhledávání cesty

V druhé fázi algoritmu, po vytvoření grafu konfiguračního prostoru, probíhá vyhledávání cesty. Vyhledávání lze spouštět opakovaně z různých startovních do různých cílových uzlů.

Před vyhledáváním cesty je potřeba nejprve napojit startovní a cílovou konfiguraci do grafu volného konfiguračního prostoru, tedy přidat dva uzly a dvě hrany. Uzly jsou charakterizovány startovní a cílovou konfigurací, hrany je potřeba nalézt. Nalezení hran probíhá podobně jako při tvorbě grafu – vyhledá se určitý počet nejbližších sousedů startovního a cílového uzlu a hledá se hrana k nejbližšímu z nich. Není-li nalezena (koliduje s překážkou), pokračuje hledání u dalšího nejbližšího uzlu. Rozdíl oproti tvorbě grafu tkví v ukončení hledání hran v případě, že už byla jakákoliv jedna hrana nalezena a přidána. Po napojení hran je spuštěno samotné hledání nejkratší cesty mezi startovním a cílovým uzlem pomocí k tomu určenému algoritmu, například lze použít Dijkstrův algoritmus. Algoritmus 4.2 ukazuje postup napojení startovní a cílové konfigurace a spuštění hledání nejkratší cesty mezi nimi.

Algoritmus PRM nezaručuje nalezení cesty mezi dvěma konfiguracemi v konfiguračním prostoru ani v případě úspěšného napojení startovního a cílového bodu do grafu. Důvodem nenalezení cesty může být přítomnost více komponent v grafu volného konfiguračního

Algoritmus 4.2 Algoritmus pro nalezení cesty mezi startovní a cílovou konfigurací.

Vstup: čtveřice $(q_{init}, q_{goal}, k, G)$, kde
q_{init} – startovní konfigurace
q_{goal} – cílová konfigurace
k – počet ukládaných nejbližších sousedů každého uzlu
$G = (V, E)$ – neorientovaný graf volného konfiguračního prostoru (vytvořen pomocí algoritmu 4.1)

Výstup: cesta P z q_{init} do q_{goal} v G

1. $N_{q_{init}} \leftarrow k$ nejbližších sousedů startovní konfigurace podle vzdálenostní funkce *dist*
2. $N_{q_{goal}} \leftarrow k$ nejbližších sousedů cílové konfigurace podle vzdálenostní funkce *dist*
3. $V \leftarrow V \cup \{q_{init}, q_{goal}\}$
4. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{init} vybraný z $N_{q_{init}}$
5. **repeat**
6.   **if** $\Delta(q_{init}, q') \neq NIL$ **then**
7. $E \leftarrow E \cup \{(q_{init}, q')\}$
8.   **else**
9. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{init} vybraný z $N_{q_{init}}$
10.   **end if**
11. **until** uzel q_{init} byl úspěšně napojen **or** $N_{q_{init}} = \emptyset$
12. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{goal} vybraný z $N_{q_{goal}}$
13. **repeat**
14.   **if** $\Delta(q_{goal}, q') \neq NIL$ **then**
15. $E \leftarrow E \cup \{(q_{goal}, q')\}$
16.   **else**
17. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{goal} vybraný z $N_{q_{goal}}$
18.   **end if**
19. **until** uzel q_{goal} byl úspěšně napojen **or** $N_{q_{goal}} = \emptyset$
20. $P \leftarrow$ nejkratší cesta z q_{init} do g_{goal} v G
21. **if** $P \neq \emptyset$ **then**
22.   **return** P
23. **else**
24.   **return** $failure$
25. **end if**

prostoru a napojení startovního a cílového uzlu k různým komponentám. Přítomnost více komponent v grafu může být následkem nedostatečného pokrytí volného konfiguračního prostoru (málo uzlů) nebo rozdělení konfiguračního prostoru na části překážkou (nelze najít nekolidující hranu mezi uzly různých komponent). Obsahuje-li graf více komponent, existují dvě možnosti, jak cestu nalézt (pokud tedy nějaká cesta existuje). Tyto možnosti jsou:

- Přidání dalších uzlů do grafu – vytvoří se tak nové hrany a existuje pravděpodobnost, že se komponenty spojí.
- Napojit startovní a cílový uzel ke stejné komponentě – jsou-li dva uzly součástí jedné komponenty, vždy mezi nimi existuje cesta.

4.3 EST algoritmus

EST algoritmus je jednodotazový pravděpodobnostní algoritmus. Pracuje podobně jako PRM algoritmus s tím rozdílem, že graf volného konfiguračního prostředí je strom, který má startovní uzel jako kořen. Existuje i dvoustromová verze EST algoritmu, kdy druhý strom má jako kořen cílový uzel. EST algoritmus pracuje ve dvou fázích:

- Tvorba stromu – generuje se strom volného konfiguračního prostoru, začíná se ve startovním uzlu. V případě dvou stromů se začíná ve startovním i cílovém uzlu a cílem je spojit oba stromy hranou.
- Nalezení cesty – dojde-li k vygenerování stromu od startovního uzlu a k napojení cílového uzlu, cesta musí existovat a je nenáročné ji najít. V případě dvou stromů cesta musí existovat, pokud se stromy spojí hranou do jednoho stromu.

Algoritmus EST byl nastudován z [10] (kapitola 7.2) a z [14] (kapitola 6).

4.3.1 Tvorba stromu konfiguračního prostoru

EST algoritmus vytváří buď jeden strom nebo dva stromy volného konfiguračního prostředí. Přidávání nových uzlů do stromu řeší algoritmus 4.3. Tento algoritmus provádí po inicializaci n pokusů o rozšíření stromu o jeden uzel. Při rozšiřování (expanzi) stromu je nejdříve vybrán uzel ze stromu, který bude expandován. Tento uzel je vybírán pomocí takzvané pravděpodobnostní funkce π_T, která bude popsána níže. Dále se generuje konfigurace z volného konfiguračního prostoru v okolí vybraného uzlu a pokud hrana (úsečka) spojující obě konfigurace nekoliduje s překážkami, je konfigurace přidána do stromu jako nový uzel.

Algoritmus 4.3 Algoritmus konstruuje EST strom volného konfiguračního prostředí.

Vstup: dvojice (q_0, n), kde
q_0 – kořen stromu (startovní uzel)
n – maximální počet pokusů o rozšíření stromu
Výstup: strom $T = (V, E)$ s kořenem q_0

1. $V \leftarrow \{q_0\}$
2. $E \leftarrow \emptyset$
3. **for** $i = 1$ to n **do**
4. $\quad q_{rand} \leftarrow$ uzel z V vybrán s pravděpodobností $\pi_T(q_{rand})$
5. $\quad q_{new} \leftarrow$ náhodně vybraná konfigurace v okolí konfigurace q_{rand}
6. $\quad$ **if** $\Delta(q_{rand}, q_{new}) \neq NIL$ **then**
7. $\quad\quad V \leftarrow V \cup \{q_{new}\}$
8. $\quad\quad E \leftarrow E \cup \{(q_{rand}, q_{new})\}$
9. $\quad$ **end if**
10. **end for**
11. **return** T

Nejdůležitější část tvorby EST stromu je výběr **pravděpodobnostní funkce** π_T. Pomocí této funkce jsou vybírány uzly stromu, které budou expandovány. Pravděpodobnostní funkce by měla generovat uzly co nejvhodněji s ohledem na pokrytí volného konfiguračního prostoru (jedná se o uzly v prostoru méně pokrytého stromem). Čím vhodněji jsou generovány uzly k expanzi, tím efektivněji algoritmus EST pracuje.

Existuje mnoho možností, jak pravděpodobnostní funkci π_T implementovat. Jeden z nejpoužívanějších způsobů uvažuje uchovávání informací o hustotě okolních bodů pro každý uzel stromu. U této možnosti je potřeba definovat okolí uzlu, ve kterém jsou počítány blízké uzly. Okolí lze definovat například jako mřížku, která rozděluje prostor na buňky a každá buňka obsahuje informaci o počtu uzlů v ní obsažených. Pravděpodobnostní funkce π_T pak vloží uzel do jedné z buněk mřížky, přičemž buňky s menším počtem uzlů jsou vybrány s větší pravděpodobností. Po přidání nového uzlu se mřížka aktualizuje.

4.3.2 Napojení stromu

Jednostromová verze algoritmu EST nalezne cestu mezi startovním a cílovým uzlem v případě, že lze cílový bod napojit nějakou nekolidující hranou s libovolným uzlem stromu. Vytváří-li algoritmus EST dva stromy (jeden s kořenem ve startovním uzlu, druhý s kořenem v cílovém uzlu) je potřeba nalézt hranu, která tyto dva stromy propojí. Pokud se povede stromy propojit, lze s jistotou tvrdit, že cesta mezi startovním a cílovým uzlem existuje.

Propojování stromů probíhá následovně. Pomocí pravděpodobnostní funkce π_T je vybrán uzel z prvního stromu. K tomuto uzlu algoritmus nalezne nejbližší uzel z druhého stromu a pomocí spojovací funkce Δ se pokusí uzly spojit hranou. Pokud je spojení úspěšné, stromy jsou propojeny a lze snadno nalézt cestu mezi startovním a cílovým uzlem. V případě neúspěchu je postup aplikován obdobně na druhý strom a hledá hranu s některým uzlem ve stromu prvním.

Urychlením EST algoritmu vznikl SBL (*Single-query, Bi-directional, Lazy-collision checking*) algoritmus, který taktéž pracuje se dvěma stromy s kořeny ve startovním a cílovém uzlu. SBL algoritmus využívá stejných principů k expanzi uzlu, pouze nekontroluje, zda hrana mezi expandovaným uzlem a nově nalezeným blízkým uzlem koliduje s nějakou překážkou. Takto jsou sestaveny oba stromy, poté jsou spojeny hranou a standardně je nalezena cesta od startovního k cílovému uzlu. Nalezená cesta projde kontrolou na legitimitu hran, tedy všechny hrany mezi uzly cesty jsou zkontrolovány, zdali nekolidují s překážkami. Urychlení algoritmu spočívá v nekontrolování kolizí s překážkami všech hran, ale pouze hran v cestě.

4.4 RRT algoritmus

RRT algoritmus je dalším jednodotazovým pravděpodobnostním algoritmem, který k popisu volného konfiguračního prostoru používá dva stromy – první s kořenem ve startovní konfiguraci, druhý s kořenem v cílové konfiguraci (lze použít i jeden strom podobně jako u EST algoritmu, viz sekce 4.3). Podobně jako EST algoritmus tvoří RRT algoritmus postupně oba stromy, které se poté snaží spojit, čímž zaručí nalezení cesty mezi startovním a cílovým uzlem.

Algoritmus RRT byl nastudován z [10] (kapitola 7.2) a z [14] (kapitola 7).

4.4.1 Tvorba stromů konfiguračního prostoru

Budují se dva stromy, T_{init} s kořenem ve startovním uzlu a T_{goal} s kořenem v cílovém uzlu. Oba stromy jsou tvořeny současně, vždy přidáním jednoho uzlu do T_{init} a jednoho uzlu do T_{goal}. Nalezení určitého počtu nových uzlů a přidání těchto uzlů do stromu lze pomocí algoritmu 4.4 v případě obou stromů.

Algoritmus 4.4 Algoritmus konstruuje RRT strom volného konfiguračního prostředí.

Vstup: trojice $(q_0, n, step_size)$, kde
q_0 – kořen stromu (startovní uzel v případě T_{init}, cílový uzel v případě T_{goal})
n – maximální počet pokusů o rozšíření stromu
$step_size$ – vzdálenost nového uzlu od expandovaného uzlu (pouze v případě konstantní délky kroku)

Výstup: strom $T = (V, E)$ s kořenem q_0

1. $V \leftarrow \{q_0\}$
2. $E \leftarrow \emptyset$
3. **for** $i = 1$ to n **do**
4. $q_{rand} \leftarrow$ náhodně vybraná volná konfigurace
5. $q_{near} \leftarrow$ nejbližší uzel z V ke konfiguraci q_{rand}
6. $q_{new} \leftarrow$ bod na úsečce mezi uzly q_{rand} a q_{near} ve vzdálenosti $step_size$ od q_{near}
7. **if** $\Delta(q_{near}, q_{new}) \neq NIL$ **then**
8. $V \leftarrow V \cup \{q_{new}\}$
9. $E \leftarrow E \cup \{(q_{near}, q_{new})\}$
10. **end if**
11. **end for**
12. **return** T

Algoritmus přijímá tři parametry – kořenový uzel stromu q_0, počet pokusů o přidání uzlu ke stromu n a vzdálenost nového uzlu od expandovaného uzlu $step_size$ a vrací vytvořený strom T (konkrétně T_{init}, popřípadě T_{goal} v závislosti na kořenovém uzlu). Po inicializaci tedy algoritmus provede n iterací obsahující:

1. Náhodné vygenerování konfigurace (uzlu) z volného konfiguračního prostoru (q_{rand}).

2. Nalezení nejbližšího sousedního uzlu z dosud sestaveného stromu (q_{near}).

3. Určení nového uzlu q_{new}, který se nachází ve vzdálenosti délky kroku ($step_size$) od uzlu q_{near} a nacházejícího se na úsečce mezi uzly q_{near} a q_{rand}.

4. Spojení uzlů q_{near} a q_{new} úsečkou pomocí spojovací funkce Δ. V případě úspěšného spojení dojde k přidání uzlu q_{new} a hrany (q_{near}, q_{new}) do stromu T.

Délka kroku $step_size$ je velmi důležitý parametr, který ovlivňuje efektivitu RRT algoritmu. Lze ho volit konstantně pro celý běh algoritmu (předpoklad u algoritmu 4.4) nebo ho upravovat podle konkrétní situace. V případě volby konstantní délky kroku $step_size$ je nutno volit vhodně s ohledem na výkonnost algoritmu. Je-li zvolena délka kroku příliš malá, bude algoritmus tvořit mnoho uzlů s větší hustotou, což může zpomalit výpočet algoritmu. Naopak příliš velká délka kroku může zapříčinit problémové tvoření stromu v konfiguračním prostoru s mnoha překážkami, kde bude problém v kolidování nových hran (hrany (q_{near}, q_{new})) s překážkami. V případě dynamicky volené délky kroku lze aktuální délku určit například podle vzdálenosti uzlů q_{near} a q_{new}. Je-li vzdálenost mezi těmito uzly větší, bude uzel q_{new} od uzlu q_{near} dále.

Pro zlepšení pravděpodobnosti nalezení cesty lze použít upravený RRT algoritmus. Úprava spočívá ve změně přístupu k tvoření nového uzlu – není tvořen pouze jeden volný uzel, ale po vytvoření jednoho uzlu se pokračuje po úsečce q_{near}, q_{new} po délce kroku $step_size$ a přidává další uzly, dokud nepřekoná velikost úsečky. Tento způsob lze ještě

vylepšit nepřidáváním všech uzlů, ale pouze jednoho nejvzdálenějšího uzlu od uzlu q_{near}, což sníží paměťové i výpočetní nároky na algoritmus.

4.4.2 Spojení stromů

RRT algoritmus končí s vytvářením stromů ve chvíli, kdy se mu podaří tyto dva stromy spojit (tím nalézt cestu od startovního do cílového uzlu). Pokus o spojení stromů lze spustit po každém přidání uzlu do obou stromů. V takovém případě je možné zkontrolovat zda, nově přidaný uzel ve stromě nelze propojit s kterýmkoliv uzlem ve druhém stromu.

Další možností je využití algoritmu 4.5, který se pokouší stromy propojit pomocí nového uzlu, vzdáleného o délku kroku od náhodně generovaného uzlu. Pokud se mu podaří spojit nalezený uzel s oběma stromy, je spojení úspěšné. V případě neúspěchu lze stromy vyměnit a algoritmus opakovat, ovšem není vhodné provádět příliš mnoho pokusů z důvodu zpomalení běhu algoritmu.

Algoritmus 4.5 Algoritmus spojující RRT stromy pomocí nového uzlu.

Vstup: čtveřice $(T_1, T_2, l, step_size)$, kde
$T_1 = (V_1, E_1)$ – první RRT strom
$T_2 = (V_2, E_2)$ – druhý RRT strom
l – maximální počet pokusů o propojení stromů T_1 a T_2
$step_size$ – vzdálenost nového uzlu od expandovaného uzlu (pouze v případě konstantní délky kroku)

Výstup: $merged$ pokud je spojení úspěšné, jinak $failure$

1. **for** $i = 1$ **to** l **do**
2. $q_{rand} \leftarrow$ náhodně vybraná volná konfigurace
3. $q_{near1} \leftarrow$ nejbližší uzel z V_1 ke konfiguraci q_{rand}
4. $q_{new1} \leftarrow$ bod na úsečce mezi uzly q_{rand} a q_{near1} ve vzdálenosti $step_size$ od q_{near1}
5. **if** $\Delta(q_{near1}, q_{new1}) \neq NIL$ **then**
6. $V_1 \leftarrow V_1 \cup \{q_{new1}\}$
7. $E_1 \leftarrow E_1 \cup \{(q_{near1}, q_{new1})\}$
8. $q_{near2} \leftarrow$ nejbližší uzel z V_2 ke konfiguraci q_{new1}
9. $q_{new2} \leftarrow$ bod na úsečce mezi uzly q_{new1} a q_{near2} ve vzdálenosti $step_size$ od q_{near2}
10. **if** $\Delta(q_{near2}, q_{new2}) \neq NIL$ **then**
11. $V_2 \leftarrow V_2 \cup \{q_{new2}\}$
12. $E_2 \leftarrow E_2 \cup \{(q_{near2}, q_{new2})\}$
13. **if** $q_{new1} = q_{new2}$ **then**
14. **return** $merged$
15. **end if**
16. $Swap(T_1, T_2)$
17. **end if**
18. **end if**
19. **end for**
20. **return** $failure$

Algoritmus 4.5 je možné zlepšit pomocí úprav zmíněných v sekci 4.4.1. Tedy tvorbou co nejvzdálenějších nových uzlů (ve vzdálenosti násobku délky kroku od expandovaného uzlu, popřípadě ve vzdálenosti náhodně vygenerovaného uzlu).

4.5 SRT algoritmus

SRT algoritmus kombinuje jednodotazový a vícedotazový přístup pravděpodobnostních algoritmů. Vícedotazový je z hlediska tvorby grafu volného konfiguračního prostoru (roadmapy), který tvoří pomocí některého z jednodotazových pravděpodobnostních algoritmů. Průběh SRT algoritmu lze rozdělit na dvě fáze, v první fázi se vytváří graf volného konfiguračního prostoru a v druhé fázi se vyhledávají cesty mezi libovolnými body ve volném konfiguračním prostoru.

Algoritmus SRT byl nastudován z [10] (kapitola 7.3) a z [14] (kapitola 8).

4.5.1 Tvorba grafu konfiguračního prostředí

Graf volného konfiguračního prostoru G_T je tvořen pomocí jednodotazových pravděpodobnostních algoritmů takto:

1. Pomocí vzorkovací funkce s například rovnoměrným rozložením pravděpodobnosti je vygenerováno n volných konfigurací v prostoru.
2. Z n vygenerovaných volných konfigurací jsou vytvořeny kořeny stromů $T_1, \ldots, T_n$, které jsou přidány do grafu G_T.
3. U každého stromu $T_1, \ldots, T_n$ provede některý z jednodotazových pravděpodobnostních algoritmů (například EST viz sekce 4.3 nebo RRT viz sekce 4.4) l pokusů o rozšíření.
4. Výsledné stromy $T_1, \ldots, T_n$ v G_T jsou vzájemně propojeny.

Propojení stromů se provádí pomocí algoritmu 4.6. Vstupem algoritmu jsou stromy vytvořené jednodotazovými pravděpodobnostními algoritmy uložené do seznamu V_T, počet nejbližších sousedních stromů k ke kterým bude daný strom napojován a počet náhodných stromů r, ke kterým bude daný strom napojován. Výstupem je výsledný graf $G_T = (V_T, E_T)$ volného konfiguračního prostoru. Algoritmus vybírá pro každý strom $k + r$ jiných stromů a postupně se je snaží napojit nejprve pomocí spojovací funkce Δ úsečkami. V případě neúspěchu napojení spojovací funkcí Δ uplatní algoritmus sofistikovanější postupy pro spojení dvou stromů (v algoritmu 4.6 zobrazeno jako funkce $MergeTrees(T_i, T_j)$), která stromy spojí například křivkou.

K určení vzdálenosti mezi dvěma stromy lze použít zprůměrování pozic uzlů pro získání globální pozice stromu. S globálními pozicemi stromů se následně pracuje jako s klasickými uzly.

4.5.2 Vyhledávání cesty

Výsledný graf G_T volného konfiguračního prostoru má v ideálním případě pouze jednu komponentu, složenou z pospojovaných stromů. Před samotným hledáním cesty je potřeba k této komponentě napojit startovní a cílový uzel. Napojení lze provést obdobně jako vytvoření bodu do grafu, tedy určit q_{init} a q_{goal} jako kořeny stromů a ty rozšířit o určený počet uzlů a následně je připojit do grafu G_T. Po napojení startovního a cílového uzlu lze cestu vyhledat pomocí algoritmů určených k hledání cesty v grafu (například Dijkstrův algoritmus).

SRT algoritmus lze pomocí určitých nastavení parametrů degradovat na algoritmy PRM, EST nebo RRT. PRM algoritmus lze získat nastavením nulového počtu expanzí při tvorbě

Algoritmus 4.6 Algoritmus propojující jednotlivé stromy v SRT grafu.

Vstup: trojice (V_T, k, r), kde
  V_T – seznam všech stromů v G_T
  k – počet nejbližších sousedů stromů určených k napojení
  r – počet náhodných stromů určených k napojení

Výstup: graf $G_T = (V_T, E_T)$ volného konfiguračního prostoru

1. $E_T \leftarrow \emptyset$
2. **for all** $T_i \in V_T$ **do**
3. $\quad N_{T_i} \leftarrow r$ náhodných a k sousedních stromů stromu T_i z V_T
4. $\quad$ **for all** $T_j \in N_{T_i}$ **do**
5. $\quad\quad$ **if** T_i a T_j nejsou součástí stejné komponenty grafu G_T **then**
6. $\quad\quad\quad merged \leftarrow FALSE$
7. $\quad\quad\quad S_i \leftarrow$ seznam náhodně vybraných uzlů ze stromu T_i
8. $\quad\quad\quad$ **for all** $q_i \in S_i$ **and** $merged = FALSE$ **do**
9. $\quad\quad\quad\quad q_j \leftarrow$ uzel ze stromu T_j nejbližší k uzlu q_i
10. $\quad\quad\quad\quad$ **if** $\Delta(q_i, q_j) \neq NIL$ **then**
11. $\quad\quad\quad\quad\quad E_T \leftarrow E_T \cup \{(q_i, q_j)\}$
12. $\quad\quad\quad\quad\quad merged \leftarrow TRUE$
13. $\quad\quad\quad\quad$ **end if**
14. $\quad\quad\quad$ **end for**
15. $\quad\quad\quad$ **if** $merged = FALSE$ **and** $MergeTrees(T_i, T_j)$ **then**
16. $\quad\quad\quad\quad E_T \leftarrow E_T \cup \{(T_i, T_j)\}$
17. $\quad\quad\quad$ **end if**
18. $\quad\quad$ **end if**
19. $\quad$ **end for**
20. **end for**

stromů v grafu volného konfiguračního prostoru (zbudou pouze kořeny) a zadáním nulového počtu opakování funkce $MergeTrees()$ (uzly budou spojovány pouze úsečkami). Je-li nastaven počet generovaných stromů na nulu (zbudou pouze startovní a cílový), přejde se při výpočtu rovnou na druhou fázi, kdy bude k nalezení cesty použit některý z jednodotazových pravděpodobnostních algoritmů, jako například EST nebo RRT.

SRT algoritmus je velmi výkonný a vhodný pro paralelizaci (výhodné pro použití pro vícedimenzionální úlohy). Paralelizovat lze tvorbu grafu, kdy jsou stromy tvořeny najednou, bez vzájemné závislosti.

Kapitola 5

Návrh aplikace

Tato kapitola obsahuje návrh aplikace nazvané *PaPla3D*, která bude schopna nalézt cestu objektu ze startovního do cílového bodu v 3D prostoru s překážkami. Aplikace bude obsahovat uživatelské rozhraní umožňující vytvoření úlohy a vizualizaci nalezené cesty.

V rámci návrhu bylo potřeba vyřešit následující:

- konfigurační soubor,
- uživatelské rozhraní,
- formát souborů s objekty,
- ovládací prvky aplikace a
- implementační jazyk a použité knihovny.

5.1 Konfigurační soubor

Konfigurační soubor slouží k nastavení volitelných parametrů aplikace. Konfiguračním souborem aplikace *PaPla3D* je soubor `PaPla3D.config`. Tento soubor obsahuje volby vztahující se k velikosti okna a velikosti scény. Konkrétní klíčová slova, které může konfigurační soubor obsahovat jsou:

- `winWidth` – Šířka aplikačního okna v pixelech. Minimální hodnota – 800. Maximální hodnota – 1920. Implicitní hodnota – 800.
- `winHeight` – Výška aplikačního okna v pixelech. Minimální hodnota – 600. Maximální hodnota – 1080. Implicitní hodnota – 600.
- `sceneWidth` – Šířka scény. Minimální hodnota – 5. Maximální hodnota – 50. Implicitní hodnota – 10.
- `sceneHeight` – Výška scény. Minimální hodnota – 5. Maximální hodnota – 50. Implicitní hodnota – 10.
- `sceneDepth` – Hloubka scény. Minimální hodnota – 5. Maximální hodnota – 50. Implicitní hodnota – 10.

Správný formát položky konfiguračního souboru a komentáře:

```
<klíčové slovo>=hodnota
#komentář
```

V případě chybějících položek (pokud nejsou povinné) se použije implicitní hodnota. Aplikaci nelze spustit bez přítomnosti konfiguračního souboru ve správném formátu (pokus o takové spuštění končí chybou).

5.2 Uživatelské rozhraní

Uživatelské rozhraní slouží k interakci uživatele s aplikací, proto musí být přehledné, snadno ovladatelné a vzhledově přívětivé. Uživatelské rozhraní je situováno do aplikačního okna, jehož velikost lze ještě před spuštěním aplikace nastavit v konfiguračním souboru (viz sekce 5.1). Za běhu lze velikost okna změnit také klasickým kliknutím na okraj a tažením ve směru změny.

Cílem uživatelského rozhraní je umožnit uživateli vytvořit si vlastní scénu z nabízených objektů, poté nastavit parametry algoritmu pro vyhledávání cesty a v neposlední řadě také vizualizovat nalezenou cestu. Aplikační okno sestává ze scény a uživatelských menu, kterými lze ovládat aplikaci. Po provedení algoritmu pak existuje možnost spuštění animace nalezené cesty.

V horní části aplikačního okna se při běhu aplikace zobrazují informační (zelenou barvou) a chybové (červenou barvou) hlášení.

Následuje popis jednotlivých částí uživatelského rozhraní.

5.2.1 Menu

Aplikace *PaPla3D* využívá dvě menu. První z nich (označme ho jako levé) slouží k výběru objektů do scény, nastavení vlastností a spuštění vyhledávacího algoritmu a následné vizualizace cesty. Dále také obecné funkce jako smazání scény nebo ukončení aplikace. Druhé menu (označme ho jako pravé) slouží jako seznam objektů vložených do scény. Každému vloženému objektu lze v tomto menu nastavit stav (aktivní, startovní) nebo ho smazat. Možnosti v obou menu jsou seskupeny podle jejich příslušnosti k dané části aplikace, popřípadě k dané akci.

Levé menu obsahuje následující skupiny a jejich možnosti:

- `Objects` – skupina obsahující objekty s odpovídajícím formátem (*.obj*) načtené z adresáře *objects/*. V této skupině se zobrazují názvy objektů podle názvů souborů. Ukázka načtených objektů ze souborů *sphere.obj* a *cylinder.obj*:
  - `sphere`
  - `cylinder`
- `Planning algorithm` – vlastnosti a spuštění vyhledávacího algoritmu.
  - `Trees` – počet generovaných kořenů stromů v prostoru.
  - `Extensions` – počet expanzí každého stromu.
  - `Distance range` – vzdálenostní rozsah nového uzlu od expandovaného uzlu. (Rozsah 1 je základní velikost objektu `sphere`, zadá-li tedy uživatel hodnotu 2, budou se generovat uzly ve vzdálenosti dvou průměrů objektu `sphere`).

* `Angle range` – úhlový rozsah nového uzlu od expandovaného uzlu. Zadaná hodnota je rozsahem do kladného i záporného směru (rozsah 20 znamená generování úhlů mezi úhlem expandovaného uzlu mínus 20 stupňů a úhlem expandovaného uzlu plus 20 stupňů).
  * `Connection step` – velikost kroku spojovací funkce. Jedná se o minimální vzdálenost mezi dvěma body na hraně, které jsou testovány na kolize s ostatními prvky scény.
  * `Nearest trees` – počet nejbližších stromů, které se spolu s daným počtem náhodných stromů algoritmus pokusí propojit s každým konkrétním stromem.
  * `Random trees` – počet náhodných stromů, které se spolu s daným počtem nejbližších stromů algoritmus pokusí spojit s každým konkrétním stromem.
  * `Nodes in tree` – počet uzlů prvního spojovaného stromu, které se algoritmus pokusí napojit s nejbližším uzlem nalezeným ve druhém spojovaném stromu.
  * `Incomplete graph` – indikace pokračování algoritmu i za předpokladu, že se ve fázi spojování stromů nezdařilo propojit všechny stromy do jedné komponenty (cesta mezi startem a cílem však může existovat).
  * `Show nodes` – zobrazení všech nagenerovaných uzlů. Zobrazuje se startovní objekt zmenšený na desetinu. Tuto možnost není vhodné používat při velkém množství nagenerovaných uzlů.
  * `Show edges` – zobrazení všech hran vzniklých při hledání cesty.
  * `Show path` – zobrazení výsledné cesty. Zobrazují se jak uzly, tak hrany nalezené i zkrácené cesty.
  * `Run` – spouštěcí tlačítko plánovacího algoritmu.

- `Animation` – vlastnosti a spuštění animace nalezené cesty.
  * `Animation speed` – rychlost animace. Zadaná hodnota charakterizuje čas, po který bude objekt cestovat po jedné hraně. (hodnota 1 znamená hrana za sekundu, 0.1 znamená hrana za 10 sekundu a 10 znamená hrana za 0.1 sekundu)
  * `Shortening path` – určuje, která cesta bude animována. Je-li zaškrtnuto, bude animována zkrácená cesta, jinak nezkrácená cesta.
  * `Animate path` – spustí animaci cesty.
  * `Init animation` – inicializuje cestu do počáteční podoby (startovní objekt na původní pozici).

- `Clear Scene` – vyčistí scénu do inicializační podoby (smaže všechny objekty vyjma cílového, inicializuje algoritmus).

- `Quit PaPla3D` – ukončí aplikaci.

Pravé menu obsahuje speciální skupinu pro cílový objekt a dále skupinu pro každý objekt vložený do scény. Příklad pravého menu, když je ve scéně zobrazen s cílovým objektem pouze jeden objekt načtený ze souboru *sphere.obj*:

- `#0 GOAL` – speciální skupina pro cílový objekt.
  * `Active` – indikace aktivity cílového objektu.

- `#1 sphere`
  - * `Active` – indikace aktivity objektu `sphere`.
  - * `Start` – je-li zatrženo, je objekt `sphere` startovním objektem.
  - * `Delete` – odstranění objektu ze scény.

5.2.2 Scéna

Scéna je hlavní částí aplikačního okna, její velikost lze nastavit v konfiguračním souboru (viz sekce 5.1) a za běhu programu již není možné jí měnit. Ovládací prvky, pomocí nichž lze se scénou a s objekty ve scéně manipulovat, jsou popsány v sekci 5.4. Scéna v inicializačním stavu obsahuje pouze cílový objekt, který má nejprve podobu neutrálního objektu. Jakmile bude některý z přidaných objektů ve scéně nastaven jako startovní, změní se podoba cílového objektu na podobu tohoto startovního objektu. Pokud není objekt definován jako startovní nebo cílový objekt, pak se jedná o překážku. Manipulovat lze pouze s aktivním objektem – objektem, který má v pravém menu nastavenu vlastnost `Active` (viz sekce 5.2.1). Aktivita objektu je v rámci scény výlučná vlastnost, nelze tedy aktivovat více než jeden objekt. To stejné platí i pro nastavení startovního objektu.

Objekty jsou podle svých vlastností odlišeny barvou. Při zobrazení vygenerovaných elementů (uzly, hrany, nalezená cesta) jsou rovněž využity barevné odlišení. Použité barvy jsou uvedeny v tabulce 5.1.

Element scény	**Barva**
cílový objekt	zelená
startovní objekt	modrá
aktivní objekt	červená
překážka	hnědá
vygenerované uzly	žlutá
vygenerované hrany	žlutá
nezkrácená cesta	světle modrá
zkrácená cesta	oranžová

Tabulka 5.1: Tabulka barev jednotlivých elementů ve scéně.

5.2.3 Animace výsledné cesty

Animace cesty slouží k vizualizaci výsledku algoritmu, nelze ji tedy spustit před provedením algoritmu a ani v případě, že bude algoritmus neúspěšný. Uživatel může díky animaci zhlédnout celou nalezenou cestu startovního objektu do cílového objektu. Startovní objekt postupně cestuje po na sebe navazujících hranách, dokud nedorazí do cílového uzlu.

Před spuštěním animace může uživatel zvolit, zdali se bude animovat zkrácená nebo nezkrácená cesta. V průběhu animace je možné animaci pozastavit (viz sekce 5.4) a opět spustit. Pokud uživatel po pozastavení animace změní cestu, která se má animovat, bude animace po spuštění inicializována do startovní pozice.

5.3 Formát souborů s objekty

Objekty zobrazované ve scéně jsou reprezentovány pomocí seznamu trojúhelníků. Díky této reprezentaci byl zvolen formát souboru pro objekty *.obj*, jelikož jeho skladba odpovídá právě této reprezentaci. Formát *obj* je velmi univerzální geometrickou definicí nejen objektů, ale i animací. Obsahuje definici každého bodu jako jeho souřadnice, mapování textury, normály a plochy.

Následuje ukázka formátu *.obj* se stručným popisem (# značí komentář):

```
# Seznam vrcholů
v 0.1 0.2 0.3
v ...
...

# Mapování textur
vt 0.500 1
vt ...
...

# Normály
vn 0.6 0.5 0.6
vn ...
...

# Plochy (definované čtyřmi možnými způsoby, popsáno níže)
f 1 2 3
f 1/2 2/1 3/3
f 1/2/3 2/1/2 3/3/1
f 1//2 2//3 3//1
f ...
...
```

Popis: Seznam vrcholů sestává ze třech souřadnic, pro každou osu jedna. Mapování textur je složeno z bodů na textuře (tedy opět souřadnice). Normály jsou definovány pomocí normálových vektorů. Plochy prvního typu (`f v1 v1 v3`) jsou definovány pouze třemi vrcholy, přičemž čísla uvádějí, o který vrchol v pořadí se jedná. Plochy druhého typu (`f v1/vt1 v2/vt2 v3/vt3`) oproti prvnímu typu obsahují navíc mapování textur na jednotlivé vrcholy. Plochy třetího typu (`f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3`) oproti druhému typu obsahují navíc normály pro každý vrchol. Plochy čtvrtého typu (`f v1//vn1 v2//vn2 v3//vn3`) obsahují kromě vrcholů pouze normály [8]. Aplikace *PaPla3D* umožňuje použití všech možností zápisu ploch (ovšem textury nezobrazuje).

Dále je možné do *obj* souboru přiřadit i soubor s materiály, ale tato funkce není v této práci potřebná, takže podrobný popis bude vynechán.

5.4 Ovládání aplikace

Aplikace *PaPla3D* musí uživateli pomocí ovládacích prvků umožnit manipulovat se scénou a objekty v ní, popřípadě řídit běh aplikace. Použité ovládací prvky jsou myš a klávesnice.

Pomocí myši lze spouštět akce ze zobrazených menu. Další funkce myši a klávesnice jsou popsány v tabulce 5.2.

Komponenta	**Akce**	**Funkce**
pravé tlačítko myši	stisk a tažení	posun scény po ose X a ose Y
kolečko myši	skrolování	posun scény po ose Z
levé tlačítko myši	stisk a tažení	rotace scény po ose X a ose Y
Z + levé tlačítko myši	stisk a tažení (vertikálně)	rotace scény po ose Z
CTRL + vše výše	viz výše	akce s aktivním objektem
Q	stisk	inicializace kamery
S	stisk při animaci	pozastavení animace
ESC	stisk	ukončení aplikace

Tabulka 5.2: Tabulka ovládacích prvků aplikace.

5.5 Implementační jazyk, použité knihovny a algoritmy

Pro implementaci aplikace *PaPla3D* byl zvolen programovací jazyk *C++*. Jazyk *C++* je velmi rozšířený a umožňuje tvořit v mnoha programovacích stylech (procedurální programování, objektově orientované programování a generické programování). Aplikace *PaPla3D* je vytvořena objektově orientovaným programovacím stylem.

Existuje velké množství knihoven implementovaných v jazyce *C++*, kterými si lze usnadnit tvorbu aplikace. V rámci implementace aplikace *PaPla3D* bylo několik takových knihoven využito. Konkrétní knihovny a jejich funkci lze vidět v tabulce 5.3.

Název knihovny	**Funkce**
AntTweakBar	tvorba uživatelského menu
ColDet	detekce kolizí mezi objekty
OpenGL (viz sekce 2.1.2)	zobrazení, práce s grafikou
SDL (viz sekce 2.2)	tvorba aplikačního okna

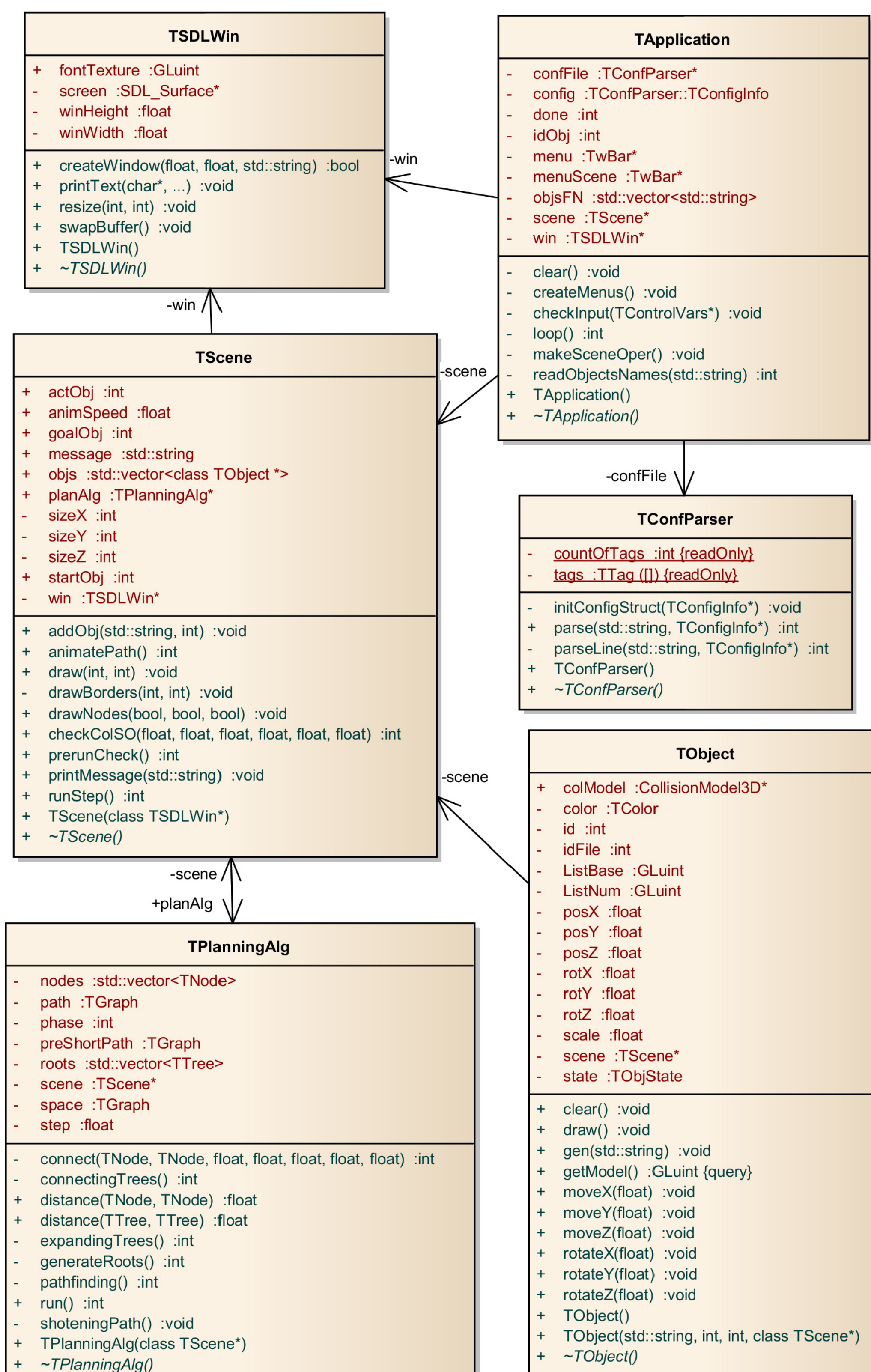
Tabulka 5.3: Tabulka použitých knihoven.

Všechny využité knihovny jsou volně dostupné k použití.

Jako vyhledávací algoritmus byl zvolen algoritmus SRT (viz sekce 4.5) pro jeho efektivitu a výkonnost a v neposlední řadě také díky jeho nastavitelnosti (pomocí parametrů lze degradovat na algoritmy PRM a EST nebo RRT, podle zvolené implementace expanze stromů).

5.6 Diagram tříd

Použité třídy a vztahy mezi nimi jsou znázorněny v diagramu tříd. Z důvodu velikosti kompletního diagramu tříd práce obsahuje (obrázek 5.1) pouze zmenšený diagram, který neobsahuje všechny atributy a metody, pouze ty důležitější z pohledu funkce jednotlivých tříd a celkové aplikace.



Obrázek 5.1: Zjednodušený diagram tříd.

Kapitola 6

Implementace

Tato kapitola obsahuje popis implementace aplikace. Tedy řešení jednotlivých částí tvorby aplikace (zobrazení, detekce kolizí mezi objekty a další), použité datové struktury a techniky. Součástí této kapitoly je také podrobný popis implementace zvoleného prohledávacího algoritmu (algoritmus SRT, viz sekce 4.5) a popis práce s některými použitými knihovnami (AntTweakBar, ColDet).

Aplikace *PaPla3D* byla implementována v programovacím jazyce *C++* (viz sekce 5.5). Základní stavební bloky jazyka *C++* byly nabyty při studiu, popřípadě v knize [23] a informace o standardních třídách jazyka byly čerpány z [5].

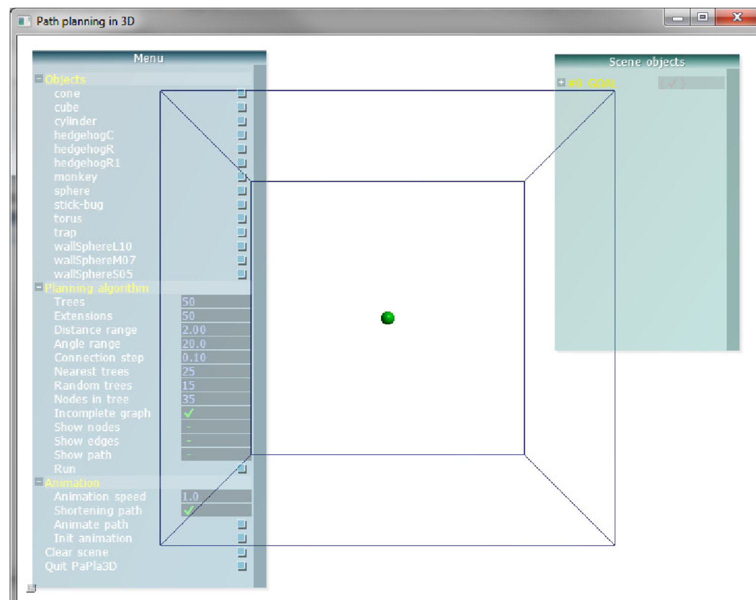
6.1 Zobrazení

Grafické prostředí tvoří 3D scéna tvořená ze dvou menu a scény, ve které jsou zobrazovány objekty (ukázka vzhledu uživatelského prostředí lze vidět na obrázku 6.1). Aplikace *PaPla3D* obsahuje také animaci nalezené cesty probíhající také ve scéně. Tato sekce popisuje implementaci zmíněných částí. Nejprve bude popsána implementace vytvoření aplikačního okna, dále implementace samostatného objektu, poté scény, menu a nakonec implementace animace nalezené cesty.

6.1.1 Aplikační okno

Implementace tvorby aplikačního okna se nachází ve třídě `TSDLWin`. Aplikační okno lze vytvořit pomocí metody `bool createWindow(float width, float height, std::string caption)`, kde parametry `width` a `height` značí šířku a výšku vytvořeného okna a `caption` definuje titulek okna. Metoda při úspěšném vytvoření okna vrací hodnotu `true`, jinak hodnotu `false`.

Další důležitou funkcí třídy `TSDLWin` je psaní textu do aplikačního okna. Před psaním textu musí být nejprve vytvořen font z textury přiložené u aplikace. K tomu slouží metoda `bool createFont(std::string path)` vracející hodnotu `true` při úspěchu a hodnotu `false` jinak. Parametr `path` značí cestu k souboru s texturou fontu. Po vytvoření fontu lze vypisovat text do scény pomocí metody `void printText(const char *fmt, ...)`, jejíž parametry jsou řídící řetězec (`fmt`) a dále proměnné použité v řídícím řetězci.



Obrázek 6.1: Ukázka uživatelského prostředí.

6.1.2 Objekty

Objekt charakterizuje třída `TObject`. Tato třída obsahuje všechny potřebné vlastnosti objektu jako je aktuální pozice ve scéně, aktuální rotace, velikost, barvu, identifikační číslo v rámci scény, identifikaci souboru, ze kterého byl načten a v neposlední řadě obsahuje zobrazovací a kolizní model objektu.

Pro práci s třídou `TObject` slouží metody, pomocí kterých lze pohybovat, popřípadě rotovat s objektem nebo nastavovat jeho velikost a podobně. Při pohybu objektu je hlídána pozice objektu, aby objekt nepřekročil hranice scény. Při rotaci jsou úhly normalizovány do intervalu od 0 do 360 stupňů.

Velmi důležitou metodou této třídy je metoda `void gen(std::string filePath)`, která ze zadané cesty k souboru s formátem *obj* (parametr `filePath`) načte objekt a vygeneruje pro něj zobrazovací a kolizní model. Třída `TObject` má k dispozici kromě bezparametrového konstruktoru také konstruktor, který díky zadaným hodnotám (cesta k souboru, identifikační číslo objektu v rámci scény, identifikace souboru a ukazatel na rodičovský objekt – scénu) vygeneruje oba modely a nastaví potřebné vlastnosti již při vytvoření instance třídy. Další stěžejní metodou je `void draw()`. Tato metoda objekt vykreslí do scény podle vlastností (pozice, rotace, velikost, barva), které má aktuálně nastaven.

Každý objekt si udržuje strukturu `state` se stavem sebe sama. Informace v této struktuře jsou aktualizovány pomocí pravého menu (viz sekce 5.2.1). Jedná se o strukturu typu (komentáře říkají, které možnosti v pravém menu ovlivňují konkrétní proměnnou):

```
typedef struct TObjState
{
  bool focus;   //Active
  bool start;   //Start
  bool goal;
  bool hide;    //Delete
}TObjState;
```

Proměnná `focus` indikuje aktivitu objektu v rámci scény (s aktivním objektem lze manipulovat). Proměnná `start/goal` indikuje, zda je objekt startovním/cílovým objektem. A proměnná `hide` značí smazání objektu ze scény. Cílový objekt je vytvořen automaticky při startu aplikace a nelze tedy měnit proměnnou `goal` pomocí menu. Cílový objekt lze vidět na obrázku 6.1 (zelená koule uprostřed scény).

6.1.3 Scéna

Implementaci scény zapouzdřuje třída `TScene`. V rámci aplikace má tato třída za úkol obstarat práci s objekty, zprostředkovat práci plánovacího algoritmu, vykreslovat sebe i všechny objekty a tvořit animaci výsledné cesty (animace výsledné cesty bude popsána v sekci 6.1.5).

Třída `TScene` spravuje všechny objekty ve scéně. Objekt lze do scény přidat pomocí metody `void addObj(std::string fileName, int idFile)`, parametry této metody jsou cesta souboru (`fileName`) a identifikace souboru (`idFile`). Po vytvoření je objekt přidán do vektoru (datová struktura `vector`) ukazatelů na objekty `objs`. Pokud je objekt smazán ze scény (pomocí `Delete` v menu, viz sekce 5.2.1), pak aplikace odstraní objekt pomocí destruktoru třídy `TObject` a do vektoru `objs` pouze uloží na místo ukazatele na smazaný objekt hodnotu `NULL`. Z vektoru `objs` se položky neodstraňují z důvodu ponechání identifikačních čísel nesmazaných objektů ve scéně (objekty jsou ve vektoru `objs` indexovány pomocí svého identifikačního čísla). Na první pozici ve vektoru `objs` se vždy nachází cílový objekt, jelikož je vytvořen již při spuštění aplikace. Scéna si udržuje aktuální informace o identifikačních číslech startovního objektu (proměnná `startObj`) a aktivního objektu (proměnná `actObj`).

Po spuštění algoritmu (`Run` v menu, viz sekce 5.2.1) je nejprve volána metoda `int prerunCheck()`, která zkontroluje kolize startovního a cílového objektu s překážkami. Pokud existuje nějaká taková kolize, plánovací algoritmus není spuštěn, vypíše se příslušné chybové hlášení a uživatel má možnost opravit nedostatky ve scéně a znovu spustit. V případě, že startovní, ani cílový objekt s překážkami nekolidují je volána metoda `int runStep()`, která nejprve pomocí metody `void initPlanAlg()` vytvoří a nainicializuje instanci třídy `TPlanningAlg` (třída implementující plánovací algoritmus, viz sekce 6.3) a poté spustí krok algoritmu (pomocí metody `int run()` třídy `TPlanningAlg`, bude popsána v sekci 6.3). Po každém kroku algoritmu je metoda `int runStep()` volána znovu pro provedení dalšího kroku, dokud plánovací algoritmus neskončí.

Třída `TScene` zajišťuje vykreslování objektů při vykreslování svých hranic. Hranice i objekty lze vykreslit pomocí metody `void draw(int winWidth, int winHeight)`, kde parametry charakterizují šířku (`winWidth`) a výšku `winHeight` aplikačního okna. Dále pomocí metody `void drawNodes(bool drawN, bool drawE, bool drawP)` lze vykreslovat uzly (`drawN=true`), hrany (`drawE=true`) a cesty (`drawP=true`) vygenerované, popřípadě nalezené plánovacím algoritmem. Parametry této metody jsou přímo závislé na stavu indikátorů vykreslení objektů, hran a cest v menu (viz sekce 5.2.1). Třída `TScene` se dále stará o vykreslování chybových a informačních hlášení.

6.1.4 Menu

Menu v aplikaci *PaPla3D* byly vytvořeny pomocí knihovny AntTweakBar [11]. Tato knihovna umožňuje snadno a rychle vytvořit menu obsahující tlačítka, zatrhávací tlačítka, textové nebo číselné editovatelné proměnné a další. Tato sekce popisuje funkce potřebné pro práci (v rámci této práce) s knihovnou AntTweakBar.

Nejprve je potřeba knihovnu inicializovat, k tomu slouží funkce `int TwInit(TwGraphAPI graphAPI, void *device)`. Parametr `graphApi` určuje v jakém grafickém prostředí se bude pracovat (pro OpenGL zadáno `TW_OPENGL`) a parametr `device` odkazuje na grafické zařízení a pro OpenGL je potřeba zadat hodnotu `NULL`. Návratová hodnota 0 značí chybu a hodnota 1 úspěch. V rámci inicializace je potřeba dále nastavit velikost grafického okna (zobrazovací plochy) pomocí funkce `int TwWindowSize(int width, int height)`, kde `width` značí šířku okna a `height` výšku okna. Návratové hodnoty jsou opět 0 při chybě a 1 při úspěchu.

Po inicializaci již lze vytvořit nové menu. K tomu slouží funkce `TwBar *TwNewBar(const char *name)` vracející ukazatel na nové menu s názvem zadaným do parametru `name`. Nově vytvořenému menu lze upravit mnoho vlastností (jako velikost, umístění, barvy, popisek, viditelnost a mnoho dalších) použitím funkce `int TwDefine(const char *def)`. Parametr `def` obsahuje řetězec s jedním nebo více nastaveními parametrů a také názvem menu, kterému se bude zadaná vlastnost nastavovat. Návratová hodnota 0 opět značí chybu a hodnota 1 úspěch.

Přidání proměnných do menu se provádí pomocí funkcí `TwAddButton`, `TwAddVarRW`, `TwAddVarRO` a další. V rámci projektu byly použity pouze dvě možnosti a to tlačítka a proměnné pro zápis i čtení. Tlačítko je možné přidat pomocí funkce `int TwAddButton(TwBar *bar, const char *name, TwButtonCallback callback, void *clientData, const char *def)`, kde `bar` určuje menu, `name` název nového tlačítka, `callback` je odkaz na návratovou funkci, která bude volána při stisknutí tlačítka, `clientData` je parametr návratové funkce a `def` je textový řetězec obsahující nastavení parametrů tlačítka (podobné jako u funkce `TwDefine`, jen se jedná o vlastnosti tlačítka). Pokud funkce vrátí hodnotu 1, tak bylo tlačítko úspěšně přidáno, pokud hodnotu 0, tak se stala chyba. Vložení proměnné pro zápis i čtení je obdobné, jedná se o funkci `int TwAddVarRW(TwBar *bar, const char *name, TwType type, void *var, const char *def)`, kde parametry `bar`, `name` a `def` jsou stejné jako u funkce `TwAddButton`. Parametr `type` určuje datový typ, který bude daná proměnná zpracovávat (k dispozici jsou veškeré základní datové typy jako `bool`, `int`, `float`, `double`, `char` a některé další). Návratové hodnoty jsou shodné s funkcemi výše.

Pro vykreslení menu je potřeba v každém cyklu programu volat funkci `int TwDraw()`. Tato funkce musí být podle specifikace knihovny volána vždy po zobrazení všech ostatních zobrazovaných prvků. Před skončením aplikace je dále potřeba knihovnu ukončit pomocí funkce `int TwTerminate()`.

Knihovna AntTweakBar dále umožňuje zpracovávat události z klávesnice a myši, popřípadě kontrolovat změny velikosti okna. Tyto schopnosti knihovny v rámci této práce ovšem nebyly využity, jelikož k tomu posloužila knihovna SDL.

6.1.5 Animace cesty

Animaci nalezené cesty lze spustit pomocí metody `int animatePath()` třídy `TScene`. Metoda vrací hodnotu 0, pokud animace může dále pokračovat nebo hodnotu 1, pokud je hotovo nebo se stala chyba (například při spuštění animace před nalezením cesty).

Rychlost animace lze nastavit v menu (viz sekce 5.2.1). Rozsah rychlosti je od hodnoty 0.1 do hodnoty 10, přičemž hodnota 0.1 značí pomalý běh a objekt cestuje po jedné hraně 10 sekund a hodnota 10 je naopak vysoká rychlost, kdy objekt po hraně přejde za 0.1 sekundy (kompromisem nechť je hodnota 1 – objekt přejde přes hranu za 1 sekundu).

Animace nalezené cesty probíhá v následujících krocích:

1. Kontrola existence cesty.

2. Nastavení animace na první hranu a uzly na této hraně v případě prvního volání metody `animatePath`. Nebo nastavení další hrany v pořadí na cestě a úprava aktuálních uzlů na uzly této hrany v případě dalších volání metody `animatePath`. Pokud další hrana neexistuje (poslední hrana již byla animována) skok na krok 5.

3. Pohyb startovního objektu po aktuální hraně ve směru k cílovému objektu. Vzdálenost uražená v jednom kroku je závislá na zadané rychlosti podělené o hodnotu 50, což značí počet *frame per second*, tedy počet překreslení obrazu za vteřinu.

4. Pokud startovní objekt nedorazil do koncového uzlu aktuální hrany, zpět na krok 3. Jinak skok na krok 2.

5. Animace dokončena, inicializace startovního objektu do původní pozice a rotace.

Nutno podotknout, že objekt cestující po hranách je kromě posouvání také rotován (závislosti určení úhlu rotace v jednom kroku jsou identické jako u posuvu).

V průběhu spuštěné animace se nezobrazují menu (z důvodu neomezování viditelnosti) a je zamezen přístup k ovládacím prvkům (obsažených v menu), avšak stále lze manipulovat se scénou pomocí myši a klávesnice (viz sekce 5.4). Animaci lze pozastavit v libovolném bodě a poté znovu spustit (například i se změněnou rychlostí animace).

6.2 Detekce kolizí – knihovna ColDet

Jak již bylo zmíněno v sekci 6.1.2, pro každý objekt přidaný do scény je vytvořena instance třídy `TObject`. Tato třída mimo jiné obsahuje i kolizní model objektu, který se tvoří při načítání reprezentace objektu ze souboru.

Třída, která charakterizuje kolizní model, je součástí knihovny ColDet a nese název `CollisionModel3D`. Pro vytvoření instance této třídy slouží metoda `CollisionModel3D *newCollisionModel3D()`. Po vytvoření instance kolizního modelu objektu lze přidávat jednotlivé trojúhelníky objektu pomocí metody `void addTriangle(float v1[3], float v2[3], float v3[3])`, parametry `v1`, `v2`, `v3` jsou trojice souřadnic jednotlivých vrcholů trojúhelníku. Po přidání všech trojúhelníků je před použitím potřeba kolizní model uzavřít, to lze provést pomocí metody `void finalize()`. Uzavřením je ukončena tvorba kolizního modelu.

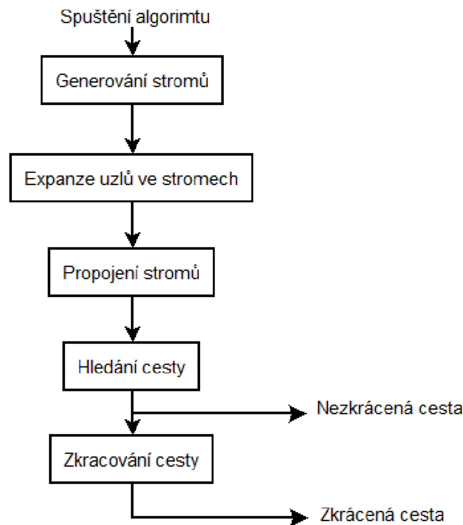
S dokončeným kolizním modelem již lze testovat kolize s jinými objekty. K tomu slouží metoda `bool collision(CollisionModel3D *other)`, kde `other` značí kolizní model objektu, se kterým probíhá test a výsledkem je hodnota `true` při kolizi, `false` jinak.

V případě potřeby zkoumat kolize pohybujících se objektů (nebo pouze objektů, které změnily svou pozici nebo rotaci oproti koliznímu modelu po uzavření) existuje metoda `void setTransform(float m[16])`, která na kolizním modelu provede transformace provedené na zobrazovacím modelu objektu. Parametr `m` je aktuální projekční matice zobrazovacího modelu.

Knihovna ColDet obsahuje mnoho jiných možností, ovšem v rámci této práce postačuje popis předešlých součástí.

6.3 Pravděpodobnostní algoritmus

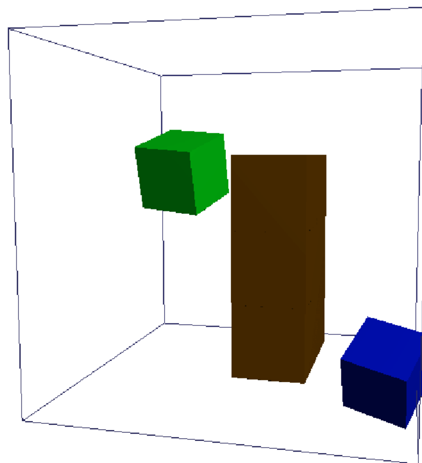
Aplikace *PaPla3D* obsahuje implementaci pravděpodobnostního algoritmu SRT (popis viz sekce 4.5). Pravděpodobnostní algoritmus zapouzdřuje třída `TPlanningAlg`. V následující



Obrázek 6.2: Blokové schéma plánovacího algoritmu.

sekci lze nalézt řešení všech částí algoritmu, včetně implementačních detailů. Nejprve budou popsány datové struktury, ve kterých jsou uloženy potřebné informace, dále vzdálenostní funkce a spojovací funkce a nakonec následuje podrobný popis jednotlivých fází implementovaného algoritmu. Zmiňované fáze jsou zobrazeny v blokovém schématu na obrázku 6.2.

Na obrázku 6.3 je zobrazena scéna, na které budou ukazovány jednotlivé fáze algoritmu. Modrá krychle slouží jako startovní objekt, zelená krychle jako cílový a hnědá zeď postavená z šesti krychlí slouží jako překážka.



Obrázek 6.3: Scéna, na které bude předváděn algoritmus.

6.3.1 Datové struktury

V této sekci jsou popsány všechny datové struktury a uložiště potřebné pro plánovací algoritmus. U každé struktury je uveden její popis a obecná funkce, podrobnější funkčnost v rámci plánovacího algoritmu bude obsažena v následujících sekcích.

Nejzákladnější datovou strukturou je uzel. Uzel je charakterizován strukturou:

```
typedef struct TNode
{
  float pX;
  float pY;
  float pZ;
  float aX;
  float aY;
  float aZ;
  int tree;
  int parent;
}TNode;
```

Položky `pX`, `pY`, `pZ` jsou souřadnice uzlu na všech osách. Položky `aX`, `aY`, `aZ` jsou úhly natočení uzlu na všech osách. Parametr `tree` určuje, ke kterému stromu v rámci scény uzel patří. A nakonec parametr `parent` určuje otcovský uzel ve stromu, ke kterému uzel patří. Je-li parametr `parent` roven hodnotě -1, jedná se o kořenový uzel stromu `tree`.

Všechny uzly ve scéně jsou ukládány do vektoru `std::vector<TNode> nodes`, což je jediné místo, které fyzicky obsahuje uložení uzlů, všechny ostatní struktury obsahující uzly obsahují pouze indexy (sloužící jako odkazy) na uzly do tohoto vektoru.

Dalším potřebným prvkem je hrana (spojnice dvou uzlů). Hrana je charakterizována strukturou:

```
typedef struct TEdge
{
  int v1;
  int v2;
  float diffAngX;
  float diffAngY;
  float diffAngZ;
}TEdge;
```

Položky `v1` a `v2` jsou indexy do vektoru `nodes` a charakterizují oba uzly na hraně. Položky `diffAngX`, `diffAngY`, `diffAngZ` značí rozdíl mezi úhly natočení na všech osách obsažených uzlů. Je-li rozdíl kladné číslo, rotuje objekt přesouvající se po této hraně kladným směrem, je-li záporné, tak záporným směrem. Při zjišťování těchto hodnot jsou zjištěny rozdíly úhlů do kladných i záporných směrů a uložen je menší úhel s příslušným znaménkem charakterizující jeho směr.

Pomocí hran a uzlů lze vytvořit grafy a stromy. Nejprve struktura pro strom:

```
typedef struct TTree
{
  std::vector<int> vrts;
  std::vector<TEdge> edges;
  float pX;
  float pY;
  float pZ;
}TTree;
```

Vektor `vrts` obsahuje indexy do vektoru `nodes` na všechny uzly ve stromu. Uzel na první pozici ve vektoru `vrts` je kořenovým uzlem stromu. Vektor `edges` obsahuje všechny hrany

stromu (typu TEdge, viz popis výše). Další tři položky, tedy pX, pY, pZ slouží jako globální pozice stromu. Globální pozice je spočtena jako těžiště pozic všech uzlů obsažených ve stromu. Struktura TTree obsahuje globální pozici z optimalizačních důvodů, jelikož je možné ji takto počítat již v průběhu tvorby stromu. Všechny stromy obsažené scéně jsou uloženy do vektoru std::vector<TTree> roots.

Obecný graf je charakterizován strukturou:

```
typedef struct TGraph
{
  std::vector<int> vrts;
  std::vector<TEdge> edges;
  void clear()
  {
    vrts.clear();
    edges.clear();
  }
}TGraph;
```

Obsažené vektory vrts a edges jsou definovány shodně jako u stromů. Struktura TGraph obsahuje navíc funkci clear(), pomocí které lze strukturu inicializovat (vymazat všechny složky grafu).

Ve scéně jsou potřeba tři uložiště typu TGraph. Jedná se o tyto uložiště:

```
TGraph space;
TGraph path;
TGraph preShortPath;
```

Uložiště space slouží k uložení kompletního grafu konfiguračního prostoru, path slouží k uložení nalezené a zkrácené cesty a preShortPath slouží k uložení nalezené cesty před zkrácením.

6.3.2 Vzdálenostní a spojovací funkce

Vzdálenostní funkce slouží k určení vzdálenosti mezi dvěma uzly. Implementována byla funkce počítající euklidovskou vzdálenost (viz sekce 4.1) s vynechanou odmocninou pro zrychlení výpočtu (není potřeba, jelikož algoritmus používá měření vzdálenosti pouze k nalezení nejbližšího uzlu). Implementovaná vzdálenostní funkce float distance(TNode n1, TNode n2) přijímá jako parametry dva uzly (n1, n2) a vrací vypočtenou vzdálenost. Vzdálenostní funkce je přetížená pro výpočet vzdáleností mezi stromy, funkce je obdobná, pouze přijímá jako parametry dva stromy a vrací vzdálenost mezi nimi, tedy mezi jejich těžišti (jejich globálními pozicemi).

Spojovací funkce slouží k nekoliznímu propojení dvou uzlů úsečkou. Její implementaci obsahuje metoda int connect(TNode n1, TNode n2, float from, float to, float angX, float angY, float angZ). Parametry n1 a n2 jsou uzly mezi kterými se metoda pokusí vytvořit hranu ve tvaru úsečky, parametry from a to určují, která část úsečky je momentálně zpracována (viz níže) a nakonec parametry angX, angY a angZ určují, o jaký úhel se objekt na dané části úsečky rotuje a podle znaménka také určuje směr rotace. Návratová hodnota 0 značí úspěch, tedy uzly lze spojit bezkolizní hranou a v případě hodnoty 1 nelze spojit.

Spojovací funkce `connect` je rekurzivní funkce (implementuje interpolaci rekurzivním dělením – viz sekce 4.1.2), která dělí potenciální hranu postupně na poloviny (a v každém dělení testuje prostřední bod na kolizi), dokud nedosáhne úsečky o velikosti kroku. Krok lze zadat v menu (možnost `Connection step`, viz sekce 5.2.1). Parametry `from` a `to` tedy vymezují právě zkoumanou a dělenou část úsečky (stejně tak s úhly). Při volání funkce je tedy potřeba zadat do `from` hodnotu 0 a do `to` vzdálenost mezi zadanými body (tedy celou délku úsečky mezi uzly).

Implementovaná spojitá funkce je symetrická a deterministická (viz sekce 4.1.2).

6.3.3 Fáze 1 – generování kořenů stromů

První fází implementace plánovacího algoritmu je generování kořenů stromů do prostoru (scény). Ve třídě `TPlanningAlg` obsahuje tuto funkčnost metoda `int generateRoots()`. Tato metoda vrací hodnotu 0 při úspěchu a hodnotu 1 při chybě. Počet kořenů stromů, který bude vygenerován lze nastavit před spuštěním algoritmu v levém menu (viz sekce 5.2.1) v položce `Trees`.

Generování stromů probíhá následovně:

1. Kontrola zadaného počtu stromů (méně než 2, končí s chybou).
2. Přidání startovního a cílového uzlu jako první dva kořeny stromů do úložiště `roots` (viz sekce 6.3.1).
3. Pokud je vytvořen dostatečný počet kořenů stromů, skok na krok 5., jinak pokračuje na krok 4.
4. Generování náhodně umístěného uzlu s náhodnými rotacemi do prostoru scény. Kontrola kolizí generovaného uzlu (tedy startovního objektu na pozici a s úhly natočení vygenerovaného uzlu) s překážkami scény. Nekolidující uzel je přidán do úložiště pro uzly (`nodes`) a také je vytvořen nový strom, kterému je tento uzel kořenem. Nový strom je přidán do úložiště `roots`. Pokračování na kroku 3.
5. Úspěšné ukončení fáze.

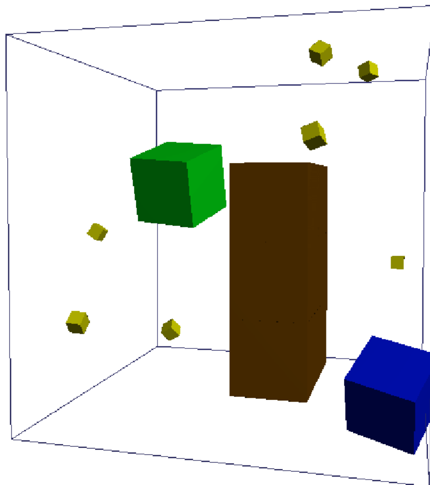
Na obrázku 6.4 lze vidět scénu po simulaci provedení první fáze algoritmu. Zadaný počet stromů je 10, přičemž dva z nich jsou startovní a cílový uzel a zbylých 8 je rozmístěno náhodně po scéně (žluté objekty).

6.3.4 Fáze 2 – expanze uzlů vygenerovaných stromů

Další fází plánovacího algoritmu je expanze uzlů ve stromech. Třída `TPlanningAlg` obsahuje metodu `int expandingTrees()`, která tuto funkčnost implementuje. Tato metoda vrací hodnotu 0 při úspěchu a hodnotu 1 při chybě. Počet expanzí v každém stromě lze nastavit v menu (viz sekce 5.2.1) v položce `Extensions`. Dalším parametrem nastavitelným z menu je dosah expanze ve všech dimenzích, tedy maximální vzdálenost a maximální rozdíl rotace mezi expandovaným a novým uzlem (položka `Distance range` pro maximální vzdálenost ve všech osách a položka `Angle range` pro maximální rozdíl mezi úhly natočení ve všech osách).

Následuje popis expanze uzlů v krocích:

1. Kontrola zadaného počtu expanzí (musí být větší než 0, jinak chyba).



Obrázek 6.4: Scéna po simulaci generování kořenů stromů.

2. Pokud byly zpracovány všechny stromy, skok na krok 5. Jinak zvolit další (na začátku první) strom.

3. Pokud byl ve stromu expandován požadovaný počet uzlů, skok na 2. Jinak náhodně určit uzel v aktuálním stromu, který bude expandován.

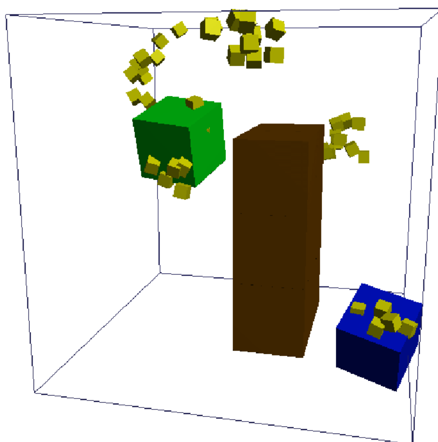
4. Generování nového uzlu v požadovaném rozsahu vzdálenosti a natočení. Kontrola kolizí tohoto uzlu s překážkami ve scéně. V případě úspěchu (uzel nekoliduje s překážkami) se provede pokus o spojení expandovaného a nového uzlu hranou (pomocí spojovací funkce, viz sekce 6.3.2). V případě, že existuje nekolizní hrana, je nový uzel přidán do uložiště `nodes` a pomocí nalezené hrany je připojen do aktuálního stromu (přidán odkaz na tento uzel a vytvořená hrana do struktury stromu). Při přidávání nového uzlu do stromu je také přepočtena globální pozice stromu (jeho těžiště, viz sekce 6.3.1). Skok na krok 3. (i v případě neúspěchu některé části v tomto kroku).

5. Úspěšné ukončení fáze.

Na obrázku 6.5 lze vidět scénu po vygenerování 5 kořenů stromů v první fázi (viz předešlá sekce) a po provedení 10 expanzí na každém stromu (nové uzly byly generovány ve vzdálenosti maximálně 1 a rozdíl úhlu nového uzlu vůči expandovanému uzlu byl nastaven na maximálně 20 stupňů do kladného i záporného směru).

6.3.5 Fáze 3 – propojení stromů do výsledného grafu

Po vytvoření stromů pomocí dvou předchozích fází přichází na řadu jejich propojení. Ve třídě `TPlanningAlg` k tomu slouží metoda `int connectingTrees()`. Tato metoda vrací hodnotu 0 při úspěchu (tedy všechny stromy se zdařilo propojit do jedné komponenty) a hodnotu 1 při neúspěchu. Nastavitelných parametrů v menu pro tuto fázi je několik (viz také sekce 5.2.1). Lze nastavit počet nejbližších stromů, se kterými bude algoritmus zkoušet propojit každý strom, pomocí položky `Nearest trees`, dále počet dalších náhodných stromů (pro případ, kdy se nepodaří propojit s nejbližšími stromy) pomocí položky `Random trees`. Lze také nastavit počet náhodně zvolených uzlů stromu, které bude algoritmus zkoušet propojit s nejbližším uzlem v připojovaném stromu, pomocí položky `Nodes in tree`.



Obrázek 6.5: Scéna po expanzi uzlů ve stromech.

Nakonec menu umožňuje rozhodnout, zda pokračovat v algoritmu i v případě neúspěchu při propojení všech stromů do jednoho grafu, jedná se o položku `Incomplete graph`.

Postup propojování stromů je následující:

1. Kontrola zadaných parametrů. Pokud součet počtu nejbližších stromů k propojení a počtu náhodně zvolených stromů k propojení přesáhne celkový počet stromů, bude krok 3. a krok 4. přeskočen a algoritmus bude zkoušet propojit aktuální strom postupně se všemi ostatními stromy, aniž by zjišťoval jejich vzdálenost nebo je náhodně volil.

2. Pokud již byly propojeny (nebo byl proveden pokus o propojení) všechny stromy, skok na krok 10. Jinak zvolit další strom (na začátku zvolit první strom).

3. Určení zadaného (`Nearest trees`) počtu nejbližších stromů.

4. Určení zadaného (`Random trees`) počtu náhodných stromů.

5. Pokud se propojení zdařilo nebo byl proveden pokus o propojení s vybranými stromy (z kroku 3. a 4.), popřípadě se všemi stromy (viz krok 1.), skok na krok 2. Jinak zvolit další z vybraných stromů, který neleží ve stejné komponentě jako aktuální strom a pokračovat.

6. Určení zadaného (hodnota `Nodes in tree`) počtu uzlů z propojovaného stromu (strom zvolený v kroku 2.).

7. Pokud se propojení zdařilo nebo byl zpracován každý z vybraných uzlů v kroku 6., skok na krok 5. Jinak zvolit další z vybraných uzlů a pokračovat.

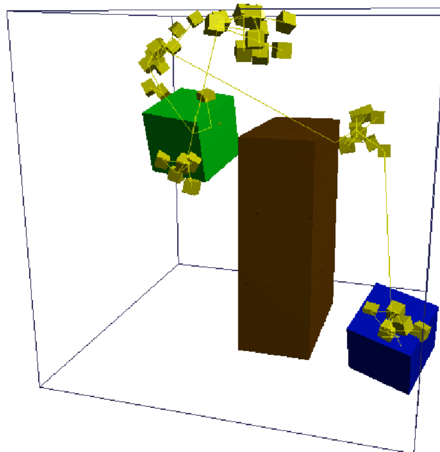
8. Nalezení nejbližšího uzlu z vybraného stromu (zvolen v kroku 6.) k vybranému uzlu (zvolen v kroku 7.).

9. Pokus o propojení obou uzlů (pomocí spojovací funkce, viz sekce 6.3.2). Pokud funkce propojí uzly úspěšně, přidá se vytvořená hrana do grafu `space` (viz sekce 6.3.1) a stromům jsou upraveny příznaky komponenty, do které nyní náleží. Skok na krok 7.

10. Ukončení propojovací fáze algoritmu.

Po ukončení propojovací fáze algoritmu se dále kontroluje počet komponent výsledného grafu. Pokud je graf složen z jedné komponenty, propojení bylo úspěšné a cesta ze startovního do cílového uzlu zřejmě existuje. Pokud bylo komponent více, pokračuje algoritmus hledáním cesty (další fáze algoritmu) pouze v případě zaškrtnuté možnosti `Incomplete graph` v levém menu. Cesta ze startovního do cílového uzlu může existovat i pokud nebyly stromy propojeny do jediné komponenty.

Na obrázku 6.6 lze vidět scénu se zobrazenými hranami po provedení propojení všech stromů do jednoho grafu. Propojení proběhlo úspěšně a výsledkem tedy je graf s jedinou komponentou, do kterého je každý strom napojen pomocí právě jedné hrany.



Obrázek 6.6: Scéna po propojení stromů.

6.3.6 Fáze 4 – hledání cesty ve výsledném grafu

V propojeném grafu (ať už zcela nebo jen zčásti) probíhá vyhledávání cesty ze startovního do cílového uzlu (tyto uzly charakterizují startovní a cílový objekt). Tuto funkčnost třídy `TPlanningAlg` implementuje metoda `int pathfinding()`, která vrací hodnotu 0 při nalezení cesty a hodnotu 1, pokud cesta neexistuje.

Vyhledávání cesty probíhá následovně:

1. Na grafu `space` obsahující propojené stromy je spuštěno prohledávání algoritmem BFS (Breadth-first search) ze startovního stromu. V rámci provedení algoritmu BFS jsou stromy, ze kterých je `space` složen, zobecněny na uzly (tedy každý strom charakterizuje jeden uzel zpracovávaného grafu) a hrany, které jsou spojnicemi mezi těmito stromy. Úspěšné provedení algoritmu BFS vrací strom předchůdců s kořenem ve startovním stromu.

2. Pokud existuje předchůdce ve stromu předchůdců pro cílový strom, cesta určitě existuje – pokračovat dále sestavením cesty. Jinak fáze neúspěšně končí (návratová hodnota 1).

3. Do cesty `path` vložit kořen cílového stromu a nastavit tento strom jako aktuální. Skok na krok 5.

4. Pokud byl poslední předchůdce aktuálního uzlu startovní uzel, skok na krok 9. Jinak zvolit za nový aktuální uzel předchůdce ve stromu předchůdců právě aktuálního uzlu.

5. Nalézt hranu mezi aktuálním uzlem a předchůdcem aktuálního uzlu v grafu `space`.

6. Do cesty `path` vložit v reverzním pořadí všechny uzly a hrany na cestě z uzlu nalezené hrany v aktuálním stromu do kořenu aktuálního stromu.

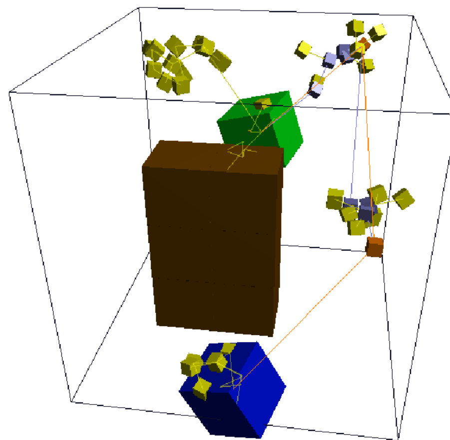
7. Do cesty `path` vložit nalezenou hranu mezi aktuálním stromem a jeho předchůdcem ve stromu předchůdců (hranu z kroku 5.).

8. Do cesty `path` vložit v přímém pořadí všechny uzly a hrany na cestě z uzlu nalezené hrany v předchůdci aktuálního stromu do kořene předchůdce aktuálního stromu. Skok na krok 4.

9. Úspěšné ukončení fáze.

Při budování cesty algoritmus kontroluje směry všech hran. Hrany, které nesměrují z cílového uzlu do startovního uzlu, jsou převráceny do požadovaného směru. Kontrola a případná úprava směru hran se provádí kvůli zjednodušení animace, která běží v reálném čase.

Na obrázku 6.7 lze vidět již vyřešený problém se zobrazenými vygenerovanými prvky i s nalezenou cestou (objekty a hrany ve světlé modré barvě). Na obrázku je znázorněna již i zkrácená cesta (prvky v oranžové barvě), která v některých úsecích převzala hrany nezkrácené cesty, proto jsou světle modré jenom části cesty.



Obrázek 6.7: Scéna po nalezení a zkrácení cesty.

6.3.7 Fáze 5 – zkracování nalezené cesty

Poslední fází algoritmu je zkrácení nalezené cesty. Tato fáze není pro chod algoritmu nutná a provádí se pouze z důvodu vyhlazení (plynulosti) výsledné cesty. Zkracování cesty provádí ve třídě `TPlanningAlg` metoda `void shoteningPath()`. Tato metoda nemá žádnou návratovou metodu, jelikož při ní již nemůže nastat chyba v algoritmu.

Zkrácení cesty probíhá v těchto krocích:

1. Aktuálně nalezenou cestu (`path`) překopírovat do nové struktury `preShortPath` (viz sekce 6.3.1).

2. Nastavit uzel x na cílový uzel. Skok na krok 4.

3. Pokud je uzel x roven startovnímu uzlu, pak skok na krok 8. Jinak nastavit x na následující uzel na cestě ve směru ke startovnímu uzlu.
4. Nastavit uzel y na startovní uzel. Skok na krok 6.
5. Pokud je uzel y roven uzlu x, skok na krok 3. Jinak nastavit uzel y na předešlý uzel na cestě.
6. Pokus o propojení uzlů x a y hranou pomocí spojovací funkce (viz sekce 6.3.2). V případě úspěšného propojení pokračovat, jinak skok na 5.
7. Ze struktury `path` odstranit všechny uzly a hrany mezi uzly x a y (samotné uzly x a y však ponechat) a vložit vytvořenou hranu na příslušné místo v cestě. Skok na krok 3.
8. Konec zkracování cesty.

V předešlém postupu je potřeba při provádění jednotlivých kroků dbát na případné ubírání uzlů a hran po úspěšném propojení některých uzlů. Kroky 3. a 5. jsou totiž prováděny vždy na již změněné cestě (mohou se změnit následníci a předchůdci některých uzlů).

Na obrázku 6.7 lze vidět zkrácenou cestu (oranžová barva) po úspěšném provedení plánovacího algoritmu.

6.4 Běh aplikace

Jako základní stavební kámen aplikace *PaPla3D* slouží třída `TApplication`. Tato třída má na starosti spuštění a běh aplikace, řídí tedy tvorbu instancí ostatních tříd. Funkci třídy `TApplication` lze rozdělit na dvě části, na inicializaci aplikace při spuštění a na aplikační smyčku.

Pomocí metody `int start()` lze spustit inicializaci aplikace. Tato metoda vrací hodnotu 0 při úspěšném provedení inicializace a úspěšném ukončení aplikační smyčky (viz níže) a hodnotu 1 při chybě. Konkrétní funkce této metody jsou:

- Tvorba aplikačního okna vytvořením instance třídy `TSDLWin` (viz sekce 6.1.1) a voláním příslušných metod této třídy.
- Tvorba parseru konfiguračního souboru (instance třídy `TConfParser`) a zprostředkování samotného parsování.
- Tvorba a inicializace scény (instance třídy `TScene`, viz 6.1.3).
- Rozšíření konfiguračních informací do instancí příslušných tříd.
- Čtení adresáře se soubory s objekty a kontrola validity jejich jmen.
- Vytvoření obou menu (viz sekce 6.1.4).
- Spuštění aplikační smyčky.

Aplikační smyčku implementuje metoda `int loop()` vracející hodnotu 0 při úspěšném ukončení smyčky, jinak hodnotu 1. V rámci aplikační smyčky probíhají následující akce:

- Kontrola vstupů z ovládacích prvků pomocí metody `void checkInput(TControlVars *vars)`, kde parametr `vars` je instance struktury obsahující informace o rychlosti pohybu a rotace scény a její aktuální pozici a rotaci na všech osách.
- Nastavení kamery do aktuální pozice a rotace scény.
- Vykreslení scény a všech objektů v ní (viz sekce 6.1.3).
- Kontrola času, dokročení na 50 fps (frame per second).
- Inicializace aplikace metodou `void clear()` v případě kliku na položku `Clear scene` v levém menu (viz sekce 5.2.1).
- Spuštění inicializace animace v případě kliku na položku `Init animation` v levém menu (viz sekce 6.1.5).
- Zprostředkování kroku animace, pokud je zrovna animace prováděna.
- Kontrola práce se scénou pomocí metody `void makeSceneOper()`. Tato metoda kontroluje přidávání objektů do scény a změny jejich stavu (viz sekce 6.1.2).
- Zprostředkování kroku algoritmu, pokud je zrovna spuštěn.
- Vykreslení menu (viz sekce 6.1.4).
- Vypsání chybové, popřípadě informační hlášky.

Po skončení aplikační smyčky jsou ukončeny běžící knihovny a uvolněny datové uložiště a aplikace končí.

Kapitola 7

Testování

Aplikace *PaPla3D* byla podrobena několika testům, které měly za úkol zjistit použitelnost, funkčnost a především výkonnost aplikace. Provedené testy a jejich výsledky jsou prezentovány v této kapitole. Nejprve následuje popis testování tvorby problému a ovládání aplikace a poté test funkčnosti a výkonnosti implementovaného algoritmu.

7.1 Ovládání aplikace

Cílem tohoto testu bylo zjistit, zda uživatelské rozhraní splňuje zadané požadavky a zda je pro uživatele snadno ovladatelné (uživatelsky přívětivé).

Ovládání aplikace bylo testováno na skupině 10 uživatelů. Každý z uživatelů měl k dispozici manuál programu a měl za úkol vytvořit libovolnou scénu z dostupných objektů a spustit algoritmus. Poté v případě neúspěchu upravovat scénu a spouštět algoritmus, dokud cestu nenalezne a spustit vizualizaci cesty. Uživatelé měli dále možnost provádět i úpravy parametrů plánovacího algoritmu, ovšem jen pokud byli seznámeni s postupem práce pravděpodobnostního algoritmu SRT použitého v aplikaci. Úspěšnost testovaných uživatelů byla hodnocena vizuálně (pozorováním).

Výsledky a zhodnocení

Všichni testovaní byli schopni vytvořit scénu pomocí přidávání objektů, problém také nedělalo rozmístění objektů a určení startovního uzlu. Vyhledání tlačítka pro spuštění plánovacího algoritmu u většiny nebyl problém, po prohlédnutí manuálu již u nikoho. Valná většina vytvořených scén byla jednoduchá a cesta byla nalezena okamžitě, pouze u dvou testovaných musela být provedena úprava scény. Jeden z nich po úpravě scény (odmazání překážky) a opětovném spuštění cestu nalezl, druhý úspěchu docílil pomocí úprav parametrů algoritmu (přidáním stromů a počtu expanzí). Spuštění vizualizace cesty opět nebyl žádný problém.

Z výsledků dosažených testery lze konstatovat, že uživatelské rozhraní umožňuje uživateli vytvořit scénu, spustit plánovací algoritmus a nalezenou cestu vizualizovat s minimálním úsilím (prostudováním manuálu aplikace).

7.2 Funkčnost plánovacího algoritmu

Funkčnost plánovacího algoritmu v rámci implementace byla testována pro každou jeho fázi (viz sekce 6.3). Funkčnost plánovacího algoritmu jako celku byla otestována na množství

různých scén, kde bylo z pozorování patrné, zda cesta existuje nebo neexistuje.

Výsledky a zhodnocení

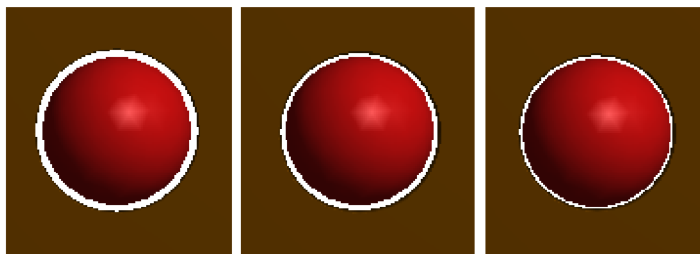
Testy prováděné v průběhu implementace na jednotlivých fázích algoritmu nebyly vždy úspěšné, ovšem nedostatky vyplývající z testů byly okamžitě odstraňovány a implementace každé fáze byla dokončena až když testy probíhaly úspěšně. Po dokončení implementace byly tedy všechny tyto testy úspěšné.

Výsledkem testování základní funkčnosti implementovaného plánovacího algoritmu jako celku bylo ve většině případů úspěšné nalezení cesty mezi startovním a cílovým objektem v případě, že tato cesta zřejmě existovala (v případě, že nalezena nebyla, postačovalo opětovně spustit plánovací algoritmus, popřípadě upravit parametry plánovacího algoritmu a opětovně spustit). Pokud cesta zřejmě neexistovala, v žádném z pokusů nebyl plánovací algoritmus úspěšně ukončen, ani po úpravě parametrů plánovacího algoritmu. Lze tedy konstatovat správnou funkčnost implementovaného plánovacího algoritmu.

7.3 Výkonnost plánovacího algoritmu

Výkonnost plánovacího algoritmu je chápána jako schopnost řešit náročné scény, tedy čím náročnější scény dokáže plánovací algoritmus vyřešit (najít cestu mezi startovním a cílovým objektem), tím lepší výkonností disponuje.

Test navržený k zjištění výkonnosti plánovacího algoritmu spočíval ve vytvoření objektu ve tvaru zdi s dírou ve tvaru koule. Díra byla větší, než samotný objekt koule a cílem bylo testovat, jak moc se může díra zmenšit, aby byl algoritmus schopný ještě nalézt cestu z jedné strany zdi na druhou. Byly vytvořeny zdi s těmito velikostmi děr (100 % značí velikost objektu koule): 110 %, 107 %, 105 %, 103 % a 101 %. (Na obrázku 7.1 lze vidět ukázky zdí s dírami o velikosti 110 %, 107 % a 105 %).



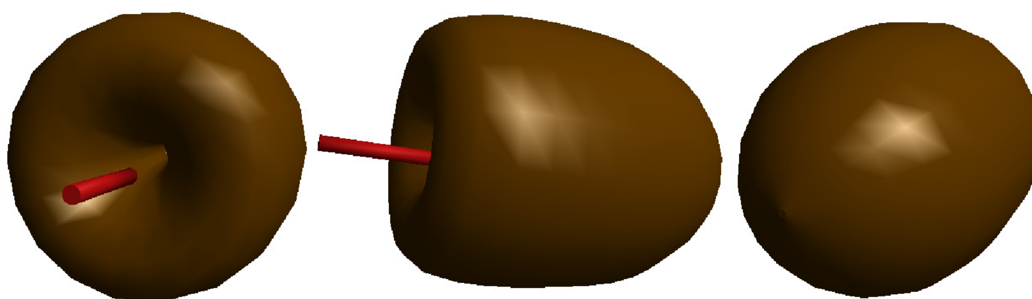
Obrázek 7.1: Ukázky testovacích zdí. Zeď vlevo má velikost 110 %, zeď uprostřed má velikost 107 % a zeď vpravo má velikost 105 %.

Další testy byly provedeny na objektech na obrázcích 7.2 a 7.3. První úkol je hlavolam zvaný „Ježek v kleci“ a cílem je dostat objekt zvaný ježek (kulatá část s ostny) z klece ven. Druhý úkol je nazvaný „Past na brouka“ a cílem je dostat brouka (objekt ve tvaru tyče) umístěného dovnitř pasti ven.

Výsledky a zhodnocení

Výsledky testu výkonnosti lze vidět v tabulce 7.1. Pro všechny spuštění byly nastaveny stejné parametry: 200 stromů rozšířených o 50 expanzí, stromy aplikace zkoušela propojit

Obrázek 7.2: Hlavolam „Ježek v kleci“.



Obrázek 7.3: Hlavolam „Past na brouka“.Vlevo pohled zepředu, uprostřed pohled z boku a vpravo pohled zezadu.

se všemi ostatními stromy a všemi uzly ve stromu (úspora strojového času, jelikož není potřeba vyhledávat nejbližší stromy, popřípadě generovat náhodné stromy ani uzly, vše se zpracuje sekvenčně). Pro každou zeď bylo provedeno 10 spuštění algoritmu. Tabulka 7.1 pak obsahuje počet úspěšných dokončení algoritmu, tedy nalezení cesty skrz konkrétní díru.

Velikost díry	Počet úspěšných nalezení cesty
110 %	10
107 %	8
105 %	4
103 %	1
101 %	0

Tabulka 7.1: Tabulka výsledků výkonnostního testu, .

Z výsledků je vidět, že pro plánovací algoritmus není problém nalézt cestu (tedy najde v každém případě), pokud má startovní objekt v nejužším průchodu rezervu alespoň 10 % své velikosti. Se zmenšováním průchodů klesá úspěšnost nalezení cesty, přičemž pro díru o velikosti 101 % (tedy s rezervou 1 % na průchod startovního objektu) již plánovací algoritmus cestu nenalezne. Díra o velikosti 101 % je tedy příliš malá – pozorováním nelze najisto určit, zda startovní objekt vůbec může touto dírou projít (kvůli reprezentaci objektu, kdy objekt koule charakterizuje mnohostěn a díra je složena z obdélníků, tedy hrozí, že 1 % rezerva nepokryje dostatečně velký prostor, aby se pokryly nerovnosti kvůli této

reprezentaci). Pro díru o velikosti 103 % velikosti objektu koule byla cesta nalezena pouze jednou. To znamená, že startovní objekt touto dírou projít může (rezerva je dostačující), ovšem nalezení cesty není pro plánovací algoritmus snadné. V závislosti na velikosti rezervy potřebuje plánovací algoritmus nalézt body na obou stranách díry přesněji, což ovlivňuje generování uzlů. Pokud existuje možnost průchodu startovního objektu dírou, lze nalézt cestu přes tuto díru, ovšem pro menší díry musí plánovací algoritmus vygenerovat uzel v mnohem menším prostoru a snižuje se tedy úspěšnost nalezení cesty.

Čas provádění algoritmu nebyl úmyslně zahrnut do tabulky výsledků, jelikož je silně závislý na náhodě (jelikož plánovací algoritmus pracuje s náhodou v téměř každé fázi provádění). Výsledné doby hledání cesty v tomto testu se lišily i v několika řádech (nalezení cesty ve 3 sekundách, ani v 500 sekundách nebylo výjimkou), průměrování těchto hodnot tedy nemá statistický význam.

Testy provedené na objektech charakterizující hlavolamy „Ježek v kleci“ (viz obrázek 7.2) a „Past na brouka“ (viz obrázek 7.3) nebyly úspěšné v žádném z pokusů a s žádným nastavením. Náročnost vyřešení těchto úkolů je příliš velká a aplikace *PaPla3D* je není schopna vyřešit. K zvýšení pravděpodobnosti nalezení řešení i u těchto úkolů by mohly pomoci některé z vylepšení plánovacího algoritmu (řízené vzorkování blízko překážek a jiné).

Kapitola 8

Závěr

Cílem práce bylo vytvořit aplikaci schopnou hledat cesty mezi startovním objektem a cílovou pozicí v 3D prostoru s překážkami. K hledání cesty měl sloužit některý z pravděpodobnostních algoritmů. Součástí aplikace mělo být uživatelské prostředí umožňující uživatelům tvořit vlastní scény přidáváním objektů a definováním startovního objektu, cílové pozice a překážek. Pomocí uživatelského rozhraní muselo být možné nastavit parametry pravděpodobnostního algoritmu pro plánování cesty a spustit tento algoritmus. V případě nalezení cesty mělo uživatelské rozhraní obsahovat možnost cestu vizualizovat.

Tyto cíle byly splněny a v tuto chvíli lze pomocí aplikace tvořit libovolné scény tvořené z dostupných objektů se snadnou definicí typu objektu (startovní, cílový, překážka) a snadnou manipulací s objekty v rámci scény. Výsledná aplikace obsahuje implementaci pravděpodobnostního algoritmu SRT, který se vyznačuje svou efektivitou a výkonností. Parametry algoritmu je možné nastavit v menu uživatelského prostředí a ve stejném místě je možné spustit plánování pomocí uvedeného algoritmu. V případě nalezení cesty ve vytvořené scéně lze cestu vizualizovat pomocí animace. Animaci lze v průběhu pozastavit v kterémkoliv bodě a opětovně spustit nebo inicializovat. Objekty přidávané do scény si může uživatel libovolně upravit (externě v kterémkoliv nástroji pro tvorbu objektů pracující s formátem *obj*, například Blender) a přidat do aplikace (do příslušného adresáře), což rozšiřuje možnosti tvorby scény téměř neomezeně.

Testování aplikace ověřilo správnou funkci implementovaného pravděpodobnostního algoritmu pro plánování cesty. Testování také ověřilo intuitivitu ovládání aplikace na několika uživatelích, kteří byli bez problémů schopni s uživatelským prostředím pracovat a plnit zadané úkoly. Testy výkonnosti aplikace proběhly na objektech charakterizujících zdi s otvory různých velikostí. Výsledky výkonnostních testů jsou uspokojivé (pokud cesta zřejmě existuje, lze ji s vhodným nastavením parametrů pravděpodobnostního algoritmu nalézt téměř vždy) a jsou prezentovány v kapitole 7. Příliš náročné úlohy (v nichž je potřeba nalézt příliš přesnou cestu, což je pro náhodné vzorkování prostoru téměř nemožné) jako jsou hlavolamy na obrázcích 7.2 a 7.3 však aplikace nebyla schopna vyřešit s žádným nastavením parametrů pravděpodobnostního algoritmu.

Pro vypracování této práce bylo potřeba nastudovat základní práci s 3D grafikou a s detekcemi kolizí objektů v 3D prostoru. Dále bylo potřeba detailně nastudovat a analyzovat pravděpodobnostní algoritmy. Před implementací aplikace bylo potřeba navrhnout strukturu programu podle zadaných požadavků (byl navržen diagram tříd, konfigurační soubor, vzhled a funkce uživatelského rozhraní a tak dále). Po návrhu následovala implementace aplikace *PaPla3D*. Implementace byla provedena v několika fázích, nejprve bylo vytvořeno aplikační okno a v něm uživatelské prostředí, následně byl implementován pravděpodob-

nostní algoritmus pro plánování cesty a nakonec byla vytvořena animace nalezené cesty. Po finalizaci aplikace byly provedeny testy na použitelnost uživatelského rozhraní a funkčnost a výkonnost plánovacího algoritmu. Z provedených testů byly vyhodnoceny závěry.

Aplikace běží na operačním systému *Microsoft Windows*.

Diplomová práce byla prezentována na konferenci Student EEICT 2013 v kategorii Inteligentní systémy a umístila se na 1. místě.

8.1 Možná rozšíření

V této sekci následuje seznam možných rozšíření aplikace, které nebyly v rámci diplomové práce z časových a jiných důvodů implementovány. Těmito možnými rozšířeními jsou:

- Implementace řízeného vzorkování náhodných bodů (například blízko k překážkám), aby byla aplikace schopna nalézt cesty i v hodně úzkých průchodech.
- Implementace řízení hustoty generovaných uzlů při vzorkování prostoru. (Uzly se budou generovat rovnoměrněji po celém prostoru – lepší pokrytí prostoru).
- Optimalizace algoritmu ve fázi propojování stromů (časově nejnáročnější fáze).
- Možnost ukládání a načítání scén a parametrů plánovacího algoritmu.
- Implementace optimálnější vzdálenostní funkce.
- Implementace sofistikovanější spojovací funkce (například spojující dva uzly nejen úsečkou, ale po neúspěchu ještě křivkou).
- Rozšířit implementaci aplikace, aby byla schopná pracovat i na jiných operačních systémech (například *Linux*).

Literatura

[1] Boundary representation [online]. Dostupné z http://cadd.web.cern.ch/cadd/cad_geant_int/thesis/node23.html, 1996 [cit. 2012-10-20].

[2] Efficient Bounding of Displaced Bézier Patches [online]. Dostupné z http://fileadmin.cs.lth.se/graphics/research/papers/2010/bezbound/, 2010 [cit. 2012-12-27].

[3] Manifold and Non-manifold Objects [online]. Dostupné z http://doc.spatial.com/index.php/Manifold_and_Non-manifold_Objects, 2011 [cit. 2012-10-22].

[4] Bounding Volumes [online]. Dostupné z http://mathforum.org/mathimages/index.php/Bounding_Volumes, 2011 [cit. 2012-11-2].

[5] C++ Reference [online]. Dostupné z http://www.cplusplus.com/reference/, 2011 [cit. 2013-4-16].

[6] blender.org - Home [online]. Dostupné z http://www.blender.org/, 2012 [cit. 2012-10-24].

[7] Constructive_solid_geometry - Wikipedia, the free encyclopedia [online]. Dostupné z http://en.wikipedia.org/wiki/Constructive_solid_geometry, 2012 [cit. 2012-10-24].

[8] Wavefront .obj file [online]. Dostupné z http://en.wikipedia.org/wiki/Wavefront_.obj_file, 2013 [cit. 2013-4-18].

[9] van den Bergen, G.: Software Library for Interference Detection [online]. Dostupné z http://www.win.tue.nl/~gino/solid/, 2004 [cit. 2012-12-26].

[10] Choset, H.: *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Mit Press, 2005, ISBN 9780262033275.

[11] Decaudin, P.: AntTweakBar GUI library [online]. Dostupné z http://anttweakbar.sourceforge.net/doc/, 2013 [cit. 2013-5-2].

[12] Ericson, C.: *Real-Time Collision Detection*. Elsevier, 2005, ISBN 9781558607323.

[13] Kravtchenko, I.: OZCollide - Collision detection library [online]. Dostupné z http://www.tsarevitch.org/ozcollide/, 2012 [cit. 2012-12-26].

[14] Kvasnica, M.: *Vizualizace hledání cesty pro robota.* Bakalářská práce, Vysoké učení technické v Brně, 2008.

[15] Kvasnica, M.: Pravděpodobnostní algoritmy – bakalářská práce [online]. Dostupné z http://www.stud.fit.vutbr.cz/~xkvasn03/bakalarka/index.php, 2008 [cit. 2012-11-9].

[16] Lantinga, S.: Simple Direct Media Layer [online]. Dostupné z http://www.libsdl.org/, 2012 [cit. 2012-10-18].

[17] LaValle, S.: *Planning Algorithms.* Cambridge University Press, 2006, ISBN 9780521862059.

[18] Lidický, B.: CZ NeHe OpenGL – Vytvoření SDL okna [online]. Dostupné z http://nehe.ceske-hry.cz/cl_sdl_okno.php, 2003 [cit. 2012-10-18].

[19] Microsoft Corporation: DirectX 11 [online]. Dostupné z http://windows.microsoft.com/cs-CZ/windows7/products/features/directx-11, 2012 [cit. 2012-10-16].

[20] OpenGL Architecture Review Board; Shreiner, D.; Woo, M.; aj.: *OpenGL programming guide: the official guide to learning OpenGL, version 2.* Addison-Wesley, páté vydání, 2006, ISBN 0-321-33573-2, 838 s.

[21] Pokorný, P.: *DirectX – začínáme programovat.* Grada, 2008, ISBN 9788024722542.

[22] Power, K.: Polygon meshes [online]. Dostupné z http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/meshes/polygon_meshes.html, 2011 [cit. 2012-10-20].

[23] Prata, S.: *Mistrovství v C++.* Computer Press, 2001, ISBN 80-7226-339-0, 966 s.

[24] SourceForge: ColDet 3D Collision Detection [online]. Dostupné z http://sourceforge.net/projects/coldet/, 2012 [cit. 2012-12-26].

[25] The Khronos Group: About the OpenGL ARB [online]. Dostupné z http://www.opengl.org/archives/about/arb/, 2012 [cit. 2012-10-16].

[26] Vašíček, J.: *Detekce kolizí.* Diplomová práce, Západočeská univerzita v Plzni, 2008.

[27] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika.* Computer Press, 2004, ISBN 9788025104545.

Příloha A

Obsah DVD

Přiložené DVD obsahuje zdrojové soubory aplikace, knihovny, potřebné pro chod aplikace, konfigurační soubor, soubory s objekty, soubor *README* se stručným manuálem k ovládání aplikace, technickou zprávu ve formátu *pdf* a zdrojové soubory technické zprávy pro program LaTeX. DVD obsahuje také spustitelné binární soubory přeložené na systému *Microsoft Windows 7 64bit*. Pro pro překlad obsahuje DVD vytvořený projekt do vývojového prostředí *Codeblocks*.

A.1 Adresářová struktura

`Aplikace PaPla3D` – zdrojové a spustitelné soubory aplikace *PaPla3D*.

- `images/` – obrázky textur.
- `objects/` – soubory s objekty pro tvorbu scén.
- `src/` – zdrojové soubory.
- `PaPla3D.cbp` – projekt do programu *Codeblocks*.
- `PaPla3D.config` – konfigurační soubor.
- `PaPla3D.exe` – binární soubor získaný překladem na systému *Microsoft Windows 7 64bit*
- `README` – návod na ovládání aplikace.
- Další soubory (potřebné knihovny a podobně).

`Technická zpráva/`

- `latex/` – zdrojové soubory a obrázky pro překlad technické zprávy.
- `zprava.pdf`

`Ukázka PaPla3D/`

- `PaPla3D.avi` – video-ukázka tvorby jednoduché scény, spuštění algoritmu a vizualizace.

Příloha B

Stručný návod k použití

Tato příloha obsahuje stručný návod k ovládání aplikace v několika krocích pro prvotní seznámení s aplikací *PaPla3D*. Návod obsahuje odkazy na příslušné místa obsahující podrobnější informace v rámci této práce. Podrobnější informace lze také nalézt v souboru *README*, který je přiložen spolu s implementací aplikace na DVD (viz příloha A).

Doporučený postup po spuštění aplikace je následující:

1. Vytvořit scénu. To lze pomocí naklikávání objektů ze seznamu v levém menu a jejich manipulací (viz ovládání aplikace v sekci 5.4) ve scéně. Před spuštěním algoritmu je potřeba nastavit v pravém menu nějakému objektu příznak, že je startovním objektem.
2. Nastavit parametry plánovacího algoritmu a spustit plánování. To lze pomocí možností v levém menu (konkrétně viz sekce popisující menu 5.2.1). Pro začátečníky nebo uživatele, kteří nejsou seznámeni s algoritmem SRT (viz sekce 4.5), se doporučuje nejprve ponechat inicializační hodnoty, popřípadě nastudovat zmíněný algoritmus a návaznost parametrů na jeho funkčnost. Plánování lze spustit rovněž v levém menu klikem na položku `Run`.
3. Pokud cesta nebyla nalezena, skok na krok 5., jinak pokračovat krokem 4.
4. Vizualizovat nalezenou cestu. Animaci lze spustit klikem na možnost `Animate path` v levém menu.
5. V této chvíli existuje několik možností. Buď lze aplikaci ukončit (možnost `Quit PaPla3D`) nebo aktuální scénu upravit a pokračovat krokem 2. nebo pokračovat na krok 2. bez úprav scény (bude se hledat cesta ve stejném prostředí, v případě, že na první pokus nebyla cesta nalezena, může plánovací algoritmus nalézt cestu nyní) nebo pomocí možnosti `Clear scene` v levém menu scénu inicializovat a začít krokem 1.