VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

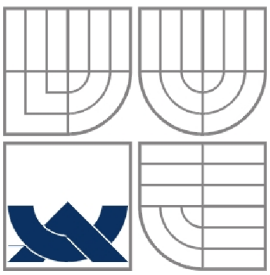
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D APLIKACE PRO MOBILNÍ TELEFONY

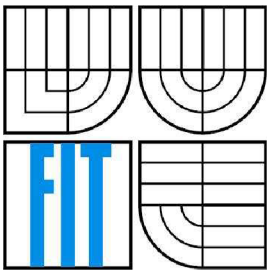
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MAREK VYORAL
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS ANF MULTIMEDIA

3D APLIKACE PRO MOBILNÍ TELEFONY

3D APPLICATION FOR MOBILE PHONES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MAREK VYORAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá 3D grafickými rozhraními M3G a MascotCapsule navrženými pro platformu J2ME (Java 2 Micro Edition). Nejprve poskytuje základní informace o samotné platformě a poté o obou rozhraních. Dále tato rozhraní porovnává z pohledu implementace, následuje porovnání jejich výkonnosti na základě testování na reálných mobilních zařízeních. Práce pak dále popisuje implementací demonstrační aplikace, která využívá rozhraní M3G.

Abstract

This thesis considers with 3D graphics interfaces M3G and MascotCapsule, both designed for platform J2ME (Java 2 Micro Edition). First of all it provides basic information about platform and then about both interfaces. Afterwards, interfaces are compared from implementation point of view. It is followed by comparision of performance based on testing on real mobile devices. Then this thesis describes implementation of demonstration application, which uses M3G interface.

Klíčová slova

J2ME, mobilní aplikace, mobilní telefon, M3G, Mobile 3D Graphics API, JSR 184, MascotCapsule

Keywords

J2ME, mobile application, mobile phone, M3G, Mobile 3D Graphics API , JSR 184, MascotCapsule

Citace

Vyoral Marek: 3D aplikace pro mobilní telefony, bakalářská práce, Brno, FIT VUT v Brně, 2009

3D aplikace pro mobilní telefony

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Aleše Láníka.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Vyoral
18.5.2009

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce Ing. Aleši Láníkovi za spolupráci, cenné rady a užitečné návrhy.

Obsah

1 Úvod

Není tomu tak dávno, kdy mobilní telefony sloužily výhradně k uskutečňování telefonních hovorů. Vzhledem k vysoké pořizovací ceně a menší dostupnosti byla tato zařízení používána především jako nástroj ke zrychlení a zefektivnění komunikace v pracovním prostředí. Přibývání další funkcionality bylo tomuto podřízeno, takže vývoj se zabýval především technologiemi využitelnými pro pracovní nasazení. Příkladem mohou být textové zprávy, datové přenosy nebo přístup k e-mailu.

Postupem času se ovšem mobilní telefony stávaly levnějšími a tím pádem dostupnějšími pro širší vrstvy populace. Zde už se situace výrazně mění. Mobilní telefon už není jen nástrojem pro práci, ale stává se i jakousi hračkou, poskytující svému uživateli zábavu, kdykoliv potřebuje ukrátit nudnou chvilku. Výrobci mobilních telefonů si toto uvědomili a postupně začali přidávat i funkce určené výhradně pro zábavu.

Nejtypičtějším příkladem jsou mobilní hry. Ty se začaly postupně objevovat už na modelech, které měly pouze jednoduchý černobílý displej a z dnešního pohledu velmi slabý výpočetní výkon. Takovéto hry byly samozřejmě velmi jednoduché a po grafické stránce velmi omezené, ovšem od té doby si mobilní telefon bez her už jen málokdo dokáže představit. První hry byly také pevně implementovány v zařízení a nebylo možné jakkoliv instalovat a přidávat hry nové. Průkopníkem v této oblasti byla společnost Nokia, která jako první přišla s možností přidávání her do svých modelů telefonů.

Absolutní revolucí v oblasti mobilních her a aplikací bylo vyvinutí platformy J2ME a zavedení její podpory na mobilních telefonech. To umožnilo do telefonů nejen přidávat nové aplikace a hry, ale také je jednoduše a efektivně vyvíjet, což zvládne prakticky každý programátor, který je schopen programovat v jazyce Java.

Postupem času se mobilní telefony stále zdokonalovaly, rostl jejich výpočetní výkon, velikost pamětí a zlepšovaly se další hardwarové parametry. Vývojem prošly také displeje, černobílé s malým rozlišením byly postupně nahrazovány barevnými s čím dál tím větším rozlišením a barevnou hloubkou. Tento fakt se samozřejmě odrazil také v oblasti mobilních her. Podobně jak tomu bylo u osobních počítačů, postupně rostla jejich složitost, velikost a především úroveň grafického zpracování. Není tedy překvapivé, že evoluce mobilních her velmi podobně kopírovala vývoj her počítačových. Ze začátku to byly především graficky a výpočetně nenáročné logické hry. Se zdokonalováním mobilního hardwaru přicházely stále propracovanější typy her, ovšem pořád byly omezeny dvourozměrnou grafikou. A tak bylo pouze otázkou času, kdy se na mobilních telefonech objeví první 3D hry.

A právě touto oblasti, 3D grafikou na mobilních zařízeních, se budu v této bakalářské práci zabývat. Konkrétně dvěma 3D rozhraními, která byla vyvinuta pro J2ME za účelem jednoduššího programování vykreslování třírozměrné grafiky v reálném čase. Tím prvním je *Mobile 3D Graphics API for J2ME* a tím druhým *MascotCapsule*. Nejprve tato práce podává základní informace o samotné platformě J2ME a o obou rozhraních. Dále pak rozhraní srovnává z pohledu programování a přístupu k některým oblastem. Následuje srovnání z pohledu výkonnosti založené na praktickém testování na různých mobilních zařízeních. Nakonec tato práce popisuje demonstrační aplikaci, která využívá M3G API a zdůvodnění volby právě tohoto rozhraní.

2 J2ME

Kapitola poskytuje základní informace týkající se *Java 2 Micro Edition* (dále J2ME) [1] a zasazuje následně probíraná témata do širšího kontextu.

2.1 Programovací jazyk Java

Programovací jazyk Java [2] vyvinula společnost Sun Microsystems. Byl představen 23. května roku 1995 a brzy se zařadil mezi nejpoužívanější programovací jazyky na světě. Jeho syntaxe vychází z programovacího jazyka C++ a odstraňuje některé problémové konstrukce. Jedná se o jazyk objektově orientovaný, tedy kromě několika primitivních datových typů jsou všechny ostatní prvky objekty.

Na rozdíl od staticky kompilovaného C++ je Java jazykem interpretovaným. To s sebou přináší jednu z největších výhod jazyka Java, a tou je snadná přenositelnost programů na různá zařízení a architektury. Nutnou podmínkou je ovšem existence interpretu (tzv. JVM – Java Virtual Machine) na daném zařízení. Java ovšem není čistě interpretovaným jazykem, zdrojový kód musí být nejdříve zkompilován do mezikódu, tzv. *bytecode*. Tento kód je poté možné distribuovat a pomocí interpretu JVM spouštět na konkrétních zařízeních. Skutečnost, že se jedná o jazyk interpretovaný s sebou přináší také jisté nevýhody. Jednou z nich může být například nižší rychlost a delší doba nutná ke spuštění aplikace. Je zde ale snaha tyto problémy eliminovat zaváděním technologií jako například *just-in-time kompilace* (některé části kódu jsou překládány přímo do nativního strojového kódu).

Další možná nevýhoda interpretovaného jazyka se projeví v podobě velké paměťové náročnosti u malých aplikaci. Ty totiž potřebují mít v paměti načteno také celé běhové prostředí.

2.2 Java platformy

Pro vývoj a běh programů v jazyce Java vznikly čtyři základní platformy [3]:

- **JavaCard** – použití na jednoduchých zařízeních jako platební a čipové karty nebo SIM karty mobilních operátorů. Jedná se o nejjednodušší edici poskytující především funkcionalitu danou cílovým zaměřením, jako šifrování a zabezpečení.
- **Java ME** – Micro Edition – použití na mobilních zařízeních jako PDA a mobilní telefony. Je známa také pod zkratkou J2ME a právě jí se zabývá tato bakalářská práce.
- **Java SE** – Standard Edition – určena pro standardní stolní počítače.
- **Java EE** – Enterprise Edition – aplikace pro rozsáhlé informační a distribuované systémy.

Aplikace napsané v jazyce Java jsou přenositelné v rámci stejné platformy. Přenositelnost mezi různými platformami je už ovšem komplikovaná, protože cílová platforma nemusí poskytovat všechnu potřebnou funkcionalitu.

2.3 Java 2 Micro Edition

Verze J2ME [4] je jakousi odlehčenou a modifikovanou edicí odvozenou od *Java Standard Edition* a upravenou pro potřeby mobilních zařízení. Mezi jednotlivými zařízeními jsou však značné rozdíly, proto vznikly ještě tzv. konfigurace. Ty specifikují minimální požadavky na zařízení, jako typ a velikost dostupné paměti, použitý procesor a jeho frekvenci nebo typ síťového připojení atd.

Existují dvě základní konfigurace [1] [5]:

> **Connected Limited Device Configuration** (CLDC) – Typickým zástupcem této kategorie jsou mobilní telefony a různé organizéry. Tato konfigurace stanovuje jako minimální hardwarové požadavky 160 kB (pro CLDC 1.1 192 kB) stálé paměti a 32 kB volné paměti typu RAM pro běh virtuálního stroje JVM. Konfigurace CLDC podporuje podmnožinu modifikovaných funkcí z některých knihoven Java Standard Edition, konkrétně *java.lang*, *java.io* a *java.util*. Přináší také zcela novou knihovnu pro vstup a výstup uzpůsobenou mobilním zařízením *javax.microedition.io*.
>
> **Connected Device Configuration** (CDC) – Tato konfigurace odpovídá zařízením, která jsou někde mezi jednoduchými zařízeními typu CLDC a stolními počítači. Typickými zástupci jsou chytré telefony, výkonnější PDA a různé kapesní počítače. Hardwarovými požadavky jsou 32 bitový procesor, 2 MB paměti RAM a 2,5 MB stálé paměti. Tato konfigurace se již na rozdíl od CLDC více *přibližuje Java Standard Edition*.

Každá z konfigurací je dále doplněna tzv. profilem. Ten přináší další třídy a funkce, které odpovídají specifickému použití a potřebám daných zařízení. Pro obě konfigurace existuje několik profilů. Z pohledu této bakalářské práce je důležitý *Mobile Information Device profil* (MIDP) [6]. Právě ten rozšiřuje konfiguraci CLDC a je určen primárně pro mobilní telefony. K hardwarovým požadavkům na zařízení přidává displej o velikosti minimálně 96 krát 54 pixelů a možnost ovládat zařízení klávesami, případně pomocí dotykové obrazovky. MIDP 1.0 doplňuje konfiguraci CLDC o tyto knihovny:

- *javax.microedition.io* – vstup/výstupní funkce upravené pro potřeby mobilních telefonů
- *javax.microedition.lcdiu* – funkce pro práci s uživatelským rozhraním
- *javax.microedition.rms* – poskytuje perzistentní úložiště dat
- *javax.microedition.midlet* – základní třídy pro běh mobilních aplikací

Časem vznikla na nových výkonnějších modelech mobilních telefonů potřeba práce např. s multimédii, proto verze profilu MIDP 2.0 přináší navíc další knihovny:

- *javax.microedition.media* – obsahuje třídy pro práci s multimédii
- *javax.microedition.lcdui.game* – knihovna pro jednoduché vytváření 2D her
- *javax.microedition.pki* – poskytuje zabezpečené připojení

2.4 Java Specification Request

Jak jednotlivé konfigurace, tak profily vznikly na základě procesu *Java Community Process* (JCP) [5]. Na tomto procesu se podílejí odborníci ze společností, které působí v oblasti počítačů, elektroniky a softwaru. Cílem je dojít ke společné dohodě a návrhu tak, aby vznikaly specifikace, podle kterých budou výrobci navrhovat své produkty. Právě díky tomu je zajištěna podpora J2ME na široké škále zařízení od různých výrobců.

Důležitým prvkem procesu vývoje je tzv. *Java Specification Request* (JSR) [7], který popisuje jakési zadání, co je potřeba vyvinout a udělat. Na základě požadavku JSR je sestavena skupina odborníků, která danou specifikaci vytvoří, schválí finální verzi a výrobci zařízení pak podle této

specifikace mohou navrhovat svá zařízení. Takovýchto JSR jsou pro celou platformu Java stovky, pro edici J2ME desítky.

Nové požadavky na specifikace se objevují neustále, proto jsou jednotlivá JSR v různých stádiích vývoje, od zadání, přes rozpracované, až po finální kompletně hotové verze. Každý požadavek *Java Specification Request* je označen číslem.

A právě jedním z těchto požadavků je také JSR 184 s názvem **Mobile 3D Graphics API for J2ME** (dále M3G) [8]. Tato specifikace vznikla na základě potřeby rozhraní pro práci s 3D grafikou na mobilních zařízeních. Nicméně nezávisle na Java Community Process vzniklo samostatně také konkurenční rozhraní se stejným zaměřením nazvané **MascotCapsule** [9], podporované především společností Sony Ericsson.

2.4.1 Přehled vybraných JSR

Následující přehled zmiňuje některá JSR [7], která se týkají aplikací na mobilních telefonech. Jsou zde uvedena především ta, jenž zasahují do oblasti grafiky a mobilních her.

JSR 30 - *Connected Limited Device Configuration*
: Specifikace CLDC konfigurace, pod kterou spadají mobilní telefony.

JSR 37 - *Mobile Information Device Profile*
: Specifikace MIDP profilu, který rozšiřuje právě CLDC konfiguraci.

JSR 118 - *Mobile Information Device Profile 2.0*
: Druhá upravená verze specifikace profilu MIDP.

JSR 135 - *Mobile Media API*
: Rozhraní pro práci s multimédii, videem a audiem na mobilních zařízeních.

JSR 139 - *Connected Limited Device Configuration 1.1*
: Novější verze konfigurace CLDC.

JSR 178 - *Mobile Game API*
: Rozhraní pro snadnější a rychlejší vývoj her.

JSR 184 - *Mobile 3D Graphics API*
: Rozhraní pro vykreslování 3D grafiky v reálném čase.

JSR 226 - *Scalable 2D Vector Graphics API*
: Specifikace API pro práci a vykreslování 2D vektorové grafiky a podporu grafického formátu W3C Scalable Vector Graphics (SVG).

JSR 271 - *Mobile Information Device Profile 3*
: Nejnovější verze profilu MIDP přinášející novou funkcionalitu.

JSR 297 - *Mobile 3D Graphics API 2.0*
: Nástupce JSR 184 přinášející podporu nových funkcí a optimalizaci funkcí stávajících.

3 M3G API (JSR 184)

Specifikace *Mobile 3D Graphics API* [8] byla vyvinuta na základě žádosti JSR 184 (Java Specification Request) jako součást *Java Cummunity Process*. Toto rozhraní je určeno především pro zařízení konfigurace CLDC s profily MIDP 1.0 nebo MIDP 2.0, což jsou typicky zařízení s nízkým výpočetním výkonem, malou velikostí paměti a bez hardwarové podpory vykreslovaní 3D grafiky. Nicméně specifikace nebrání implementaci tohoto rozhraní i na složitějších zařízeních konfigurace CDC nebo jakékoliv jiné Java platformě.

Vývojem se zabývala skupina odborníků, tzv. Mobile *3D Graphics API Expert Group*, složená ze zástupců mnoha firem působících v oblasti mobilních zařízení a vestavěných systémů (např. Nokia, Sony Ericsson, Siemens, Motorola, Sun Microsystems, Intel, Texas Instruments). Zapojení takto širokého spektra výrobců přineslo rozšíření tohoto rozhraní prakticky na většinu mobilních telefonů s podporou J2ME na současném trhu.

3.1 Vytváření modelů

Kromě vytváření modelů přímo pomocí příkazů programovacího jazyka, což může být velmi složité, nepřehledné a časově náročně, umožňuje M3G API použít jednodušší postup. Modely lze vytvořit pomocí externích 3D editorů, exportovat je do souboru formátu *.m3g* a poté načíst v mobilní aplikaci pomocí třídy Loader. Toto přináší především zrychlení a zjednodušení vývoje. Nicméně možnost vytvářet modely pomocí programového kódu zůstává zachována, neboť v některých aplikacích může být nezbytná.

Plug-in moduly umožňující export modelů do *.m3g* souborů existují pro všechny běžně rozšířené 3D editory (např. 3ds Max, Blender, Maya, Cinema 4D). Náležitosti formátu, datové typy a struktura souboru jsou důkladně popsány ve specifikaci API.

3.2 Třídy balíčku javax.microedition.m3g

M3G API je obsaženo v balíčku **javax.microedition.m3g** [8]. V této podkapitole je uveden seznam všech tříd, které tento balíček obsahuje. U každé třídy je pak uveden stručný popis. Jde pouze o přehled základních údajů, detaily některých vybraných tříd jsou pak rozebírány v dalších kapitolách.

AnimationController uchovává informace o rychlosti a pozici animační sekvence.
AnimationTrack spojuje KeyframeSequence a AnimationController a umožňuje tak animaci.
Appearance seskupuje objekty tříd, které určují vzhled modelu a další vlastnosti ovlivňující jeho vykreslení, například Material a Texture2D.
Background odpovídá pozadí scény. Umožňuje nastavit barvu nebo obrázek pozadí záběru.
Camera reprezentuje bod, ze kterého je scéna pozorována. Umožňuje nastavit jak paralelní, tak perspektivní projekci.
CompositingMode je komponentou třídy Appearance zapouzdřující vlastnosti týkající se vykreslování pixelů.
Fog je taktéž komponentou třídy Appearance, definuje osvětlení v závislosti na vzdálenosti objektu.
Graphics3D reprezentuje grafický kontext. Může mít pouze jedinou instanci. Veškeré vykreslování

v aplikaci probíhá pomocí metody *render* právě této třídy.

Group je uzlem v grafu scény, umožňuje seskupovat více objektů do skupiny.

Image2D uchovává dvourozměrný obrázek, který může být použit jako textura, pozadí scény nebo sprite.

IndexBuffer je komponenta třídy Mesh definující submesh.

KeyframeSequence reprezentuje posloupnost klíčových snímků animace.

Light je zdrojem světla ve scéně. Lze mu nastavit vlastnosti jako barva, intenzita nebo typ osvětlení.

Loader poskytuje metody k importu a deserializaci modelů a grafu scény z externích *.m3g* souborů. Také poskytuje metodu pro načtení textury ze souboru *.png*.

Material je jedna z komponent třídy Appearance definující vzhled objektu.

Mesh reprezentuje 3D objekt ve scéně jako povrch složený z polygonů a přiřazuje mu vzhled typu Appearance.

MorphingMesh se liší od běžného tělesa Mesh tím, že může obsahovat více objektů VertexBuffer. Každý z nich může mít dále zadánu svou váhu. Výsledný tvar tělesa je poté dán kombinací všech VertexBufferů podle jejich váhy.

Node je abstraktní třída, která reprezentuje uzel v grafu scény. Z této třídy dědí třídy Camera, Mesh, Sprite3D, Light a Group. Díky tomu mohou být uzly v grafu scény.

Od třídy **Object3D** je odvozeno téměř všechno, co se ve 3D scéně vyskytuje. Výjimkou jsou pouze třídy Loader, Transform, RayIntersection a Graphics3D.

PolygonMode je komponentou třídy Appearance, uchovává informace týkající se vykreslování polygonů (viz. kapitola 5.6 Vzhled objektů).

RayIntersection je třída, kterou využívá API výhradně při vykreslování pro uložení průsečíků.

SkinnedMesh je vhodnou třídou pro objekty animované pomocí kostry (skeletally animated).

Sprite3D odpovídá dvourozměrnému obrázku, který má třírozměrnou pozici ve scéně.

Texture2D je jednou z komponent Appearance, uchovává texturu typu Image2D a další informace specifikující umístění textury na povrch tělesa, souřadnice a typ filtrování.

Transform odpovídá transformační matici. Má rozměry 4 řádky a 4 sloupce čísel typu *float*. Poskytuje metody pro rotace, inverzi, násobení matic a podobně.

Transformable je nadřazenou třídou pro objekty typu Node a Texture2D. Poskytuje metody pro transformaci 3D objektů a textur, jako například rotace, posunutí nebo změnu měřítka

VertexArray je pole určené pro uchování souřadnic vrcholů, jejich barev a normál.

VertexBuffer je složen z několika VertexArray. Reprezentuje vrcholy, které dohromady tvoří povrch 3D objektu.

World je kořenovým uzlem celého grafu scény.

4 MascotCapsule

Toto rozhraní, podobně jako M3G bylo vyvinuto jako knihovna pro vykreslování 3D grafiky v reálném čase pro vestavěná zařízení, mobilní telefony a PDA podporující J2ME. Vývojem tohoto API se zabývá japonská firma HI Corporation. MascotCapsule vyžaduje jako minimální prostředí pro korektní běh zařízení s procesorem srovnatelným s ARM9 100Mhz, zabírá 100 kB paměti a vyžaduje alespoň 800 kB volného místa na haldě [9].

Rozhraní prochází postupným vývojem, proto je potřeba uvést, že tato bakalářská práce pojednává o třetí verzi, tj. MascotCpasule V3. V současné době již existuje verze V4, která přináší novou funkcionalitu a plně odpovídá specifikaci M3G 1.1. Z tohoto důvodu zde již zásadní odlišnosti MascotCapsule oproti M3G prakticky zanikají. V současné době je stále ale ještě používána převážně verze 3, proto se tedy tato práce zabývá MascotCapsule V3.

4.1 Vytváření modelů

Modely pro MascotCapsule lze stejně jako v MG3 API vytvářet jak v kódu, tak pomocí běžných komerčních či volně šiřitelných 3D editorů. Podporovány jsou například nástroje 3ds Max, LightWave3D, Maya, Softimage 3D a Blender.

Po vytvoření modelu následuje export do souborů formátu *.bac* a *.tra* [10]. Prvně zmíněný soubor obsahuje samotný model, druhý soubor obsahuje animační data. Tyto soubory je následně potřeba zkonvertovat do formátu *.mbac*, respektive *.mtra*. K tomu slouží nástroj *M3DConverter*, který je k dispozici zdarma na stránkách společnosti HI Corporation.

Dalšími užitečnými nástroji, které jsou volně k dispozici, jsou jednoduchý editor *.bac*/*.tra* souborů *PAC* a prohlížeč *.mbac*/*.mtra* souborů *PVMicro*. Ten je možné použít pro prvotní náhled, jak přibližně bude výsledný model vypadat na mobilním zařízení. Stejně jako v M3G API je pak modely uložené ve vyexportovaných souborech možné načíst do aplikace.

4.2 Třídy balíčku com.mascotcapsule.micro3d.v3

Rozhraní MascotCapsule je obsaženo v balíčku **com.mascotcapsule.micro3d.v3** [11]. Na rozdíl od M3G je MacotCapsule na počet tříd značně chudší, je jich pouze přibližně třetina.

ActionTable uchovává data o pohybu objektů a umožňuje tak jejich animaci.
AffineTrans je transformační matice o rozměru 3 řádky a 4 sloupce. Slouží k manipulaci s objektem, například k rotaci.
Effect3D obsahuje informace pro vykreslení objektu, typ stínování, osvětlení, průhlednost atd.
Figure reprezentuje 3D objekt ve scéně a umožňuje načíst model z externího souboru.
FigureLayout obsahuje další data nutná pro vykreslení objektu, především pozici, orientaci a velikost.
Graphics3D je stěžejní třídou celého API, zajišťuje totiž vykreslování a poskytuje k tomu potřebné metody.
Light reprezentuje osvětlení scény a uchovává nastavení jeho parametrů.
Texture uchovává texturu a umožňuje její načtení z externího souboru.
Util3D poskytuje matematické operace odmocnina, sinus a cosinus.
Vector3D je vektor tří čísel typu *int*, který je často používán jinými třídami.

5 Srovnání obou API

Tato kapitola srovnává obě rozhraní z implementačního pohledu. Zabývá se několika oblastmi, ve kterých se obě API od sebe značně liší. Tato kapitola čerpá ze zdrojů [8], [9], [10] a [11].

5.1 Načítání modelů do aplikace

Jak bylo v předchozích kapitolách uvedeno, modely lze vytvářet v různých 3D editorech a poté je exportovat do souborů odpovídajících formátů. Nyní se podíváme, jakým způsobem lze s modely v těchto souborech uložených, pracovat přímo v aplikaci.

K načtení modelu do aplikace v M3G API slouží speciální třída Loader. Ta umí deserializovat data z *.m3g* souboru a nalézt v nich instance tříd odvozených od Object3D, tj. World, Group, Camera, Light a dále také Material, Appearance a Texture2D. Z této třídy nelze vytvářet instance a obsahuje pouze dvě statické metody `load` (rozlišeny pomocí parametrů). Ty vrací pole nalezených objektů typu Object3D. Toto pole však neobsahuje všechny objekty grafu scény, obsahuje pouze objekty (uzly), na které není reference ze žádných jiných uzlů. Tyto vrcholové uzly lze dále prohledávat metodou `load` pro nalezení dalších objektů, a takto rekurzivně pokračovat. Příklad načtení scény:

```
World world = null;

Object3D[] buffer = Loader.load("/myScene.m3g");
for(int i = 0; i < buffer.length; i++) {
 if(buffer[i] instanceof World) {
  world = (World)buffer[i];
  break;
 }
}
```

Statická metoda *load* třídy Loader vrátí do pole `buffer` nalezené instance typu Object3D, které nalezne v souboru *myScene.m3g*. Následuje cyklus, který toto pole prohledá a získá referenci na objekt typu World. Ten je kořenem grafu scény, který obsahuje všechny modely a objekty. Díky tomu můžeme získat reference na další objekty ve scéně.

Při exportu scény do *.m3g* souboru je potřeba nastavit jednu kameru jako aktivní. Tu lze pak získat jednoduše pomocí metody třídy World *getActiveCamera().* Při exportu do souboru je také vygenerován HTML dokument, který obsahuje tabulku všech objektů scény a jejich identifikátor (celé číslo). Reference na jednotlivé objekty scény lze získat právě pomocí těchto jejich ID. Třída World obsahuje metodu *find(int ID),* která toto provede, nicméně je třeba ještě provést přetypování na odpovídající typ. V případě 3D objektu je to tedy přetypování na typ Mesh apod.

MascotCapsule používá zcela odlišný přístup k načítání modelů do aplikace. Předchozí způsob není možný už jen kvůli absenci implementace grafu scény v tomto API. Jak již zde bylo zmíněno dříve, MascotCapsule chápe jeden vyexportovaný soubor *.mbac* jako jeden 3D model objektu, ne jako celou scénu, jak tomu je u M3G.

Objekt je zde reprezentován třídou Figure. K načtení modelu ze souboru slouží konstruktor, který jako parametr přijímá řetězec s cestou k danému souboru s modelem:

```
Figure box = new Figure("/box.mbac");
```

Soubor *.mbac* obsahuje pouze objekt samotný, nikoliv navíc kamery, světla, textury a materiály, jak tomu může být v M3G. Tyto objekty je třeba deklarovat, nastavit a umístit do celé scény dodatečně v kódu.

5.2 Objekty ve scéně a manipulace s nimi

Každý objekt v třírozměrné scéně je reprezentován instancí odpovídající třídy. V M3G se jedná o třídu Mesh. Nejdůležitějším prvkem této třídy je *VertexBuffer*, který obsahuje souřadnice všech vrcholů tělesa a normály. Dále pak tato třída obsahuje pole *IndexBuffer*, ve kterém je možné pomocí indexování vrcholů definovat submesh. Další důležitou součástí třídy Mesh je pole objektů typu Appearance, které definují vzhled celého objektu, případně každého submesh.

S objekty ve scéně je často potřeba manipulovat, rotovat jimi a měnit jejich velikost. To je umožněno díky tomu, že třída Mesh dědí ze třídy Transformable. A právě ona poskytuje metody pro výše zmíněné manipulace. Tyto metody jsou pak volány přímo nad konkrétní instancí třídy Mesh.

Metoda *setTranslation(float x, float y, float z)* umístí objekt přesně na zadané souřadnice ve scéně. Metoda *translate(float x, float y, float z)* přičte zadané hodnoty k aktuální pozici objektu, tj. jedná se o posunutí objektu.

Podobně je tomu i u rotací. Existuje zde metoda na nastavení orientace objektu *setOrientation(float angle, float x, float y, float z).* Prvním parametrem je úhel natočení udaný ve stupních po směru hodinových ručiček, další tři parametry udávají osu rotace. K rotaci o určitý počet stupňů podle zadané osy slouží metody *preRotate(float angle, float x, float y, float z)* a *postRotate(float angle, float x, float y, float z),* které se liší pouze pořadím násobení transformačních matic.

Poslední důležitou manipulací s objektem je změna jeho velikosti. Jako u předchozích operací, i zde je k dispozici metoda, která přímo nastaví velikost objektu *setScale(float x, float y, float z).* Samozřejmě nechybí ani metoda pro změnu velikosti *scale(float x, float y, float z),* která aktuální velikost vynásobí zadanými hodnotami.

Další možností manipulace je přímá úprava transformační matice. Ta je prvkem v již zmíněné nadřazené třídě Transformable jako objekt třídy Transformation. Jedná se o transformační matici čísel typu float o rozměru 4 krát 4. Třída Transformation poskytuje metody jako inverze, rotace a násobení jinou maticí.

V MascotCapsule je 3D objekt ve scéně reprezentován třídou Figure. Ta obsahuje taktéž prvky jako souřadnice vrcholů a normály, navíc na rozdíl od M3G třídy jí lze přímo nastavit texturu (případně více textur) typu Texture.

Zásadní rozdíl ovšem nastává v přístupu k manipulacím s objekty. V MascotCapsule nelze jednoduše volat nad objektem třídy Figure metody zajišťující manipulaci, jak tomu je v M3G. Zde je situace zcela odlišná. Informace o objektu jako pozice, rotace, měřítko a další data nutná pro vykreslení jsou uloženy ve třídě FigureLayout. K propojení konkrétního 3D objektu ve scéně s konkrétní instancí FigureLayout dojde až při volání vykreslovací metody. Třída FigureLayout již ale poskytuje metody pro nastavení umístění a měřítka.

Ovšem neimplementuje přímo metody pro rotace s objektem. Obsahuje ale transformační matici typu AffineTrans (případně pole takovýchto matic), která již poskytuje spoustu metod pro práci s transformační maticí. Mezi nimi jsou právě také metody pro rotace podle jednotlivých os.

Porovnáme-li přístupy obou API, vidíme, že M3G přináší navíc jakési zjednodušení v podobě metod, které lze volat přímo nad konkrétními objekty a které vykonají požadovanou transformaci.

5.3 Textury

Další oblastí, ve které se obě API značně liší, je práce s texturami.

V M3G není maximální velikost textury nijak omezena, pouze množstvím volné paměti, která je k dispozici. Nicméně zde jisté omezení existuje, rozměry textury musí být mocninami čísla 2. Textury mohou být uloženy přímo v souboru *.m3g* společně s modely, nebo pomocí metody statické třídy Loader načteny z externího souboru (ve formátu.*png)*. Barvy jsou uloženy ve formátu RGB, případně RGBA, jeden byte pro každou složku.

Samotný obrázek textury je reprezentována objektem třídy Image2D. Tento objekt může mít nastavenu vlastnost *muttable*, tzn. může být měněn kdykoliv za běhu aplikace a změna se projeví okamžitě ve scéně. Image2D může být ale taky nastaven jako *immutable*, tzn. po vytvoření se již měnit nebude. Tato informace umožňuje API provést optimalizaci, kompresi, případně redukci barevné hloubky za účelem zvýšení rychlosti vykreslování a úspory paměti.

Objekt třídy Image2D může být použit také jako sprite nebo obrázek na pozadí scény. V tomto případě ovšem mohou být rozměry libovolné, omezení na mocniny čísla 2 zde neplatí.

Texturu jako takovou reprezentuje třída Texture2D, ovšem samotná bitmapa, jak již bylo zmíněno, je uložena v objektu typu Image2D. Třída Texture2D uchovává navíc také další vlastnosti textury, jako například typ filtrování, nastavení barvy nebo blending. Na rozdíl od MascotCapsule, texturu nelze konkrétnímu 3D objektu (Mesh) přiřadit přímo. Je zde ještě jeden mezičlánek, a tím je třída Appearance. Ta kromě jiného může obsahovat objekt typu Texture2D. Objekt typu Appearance lze pak již přímo přiřadit danému objektu Mesh, případně více objektům.

MascotCapsule umožňuje použít textury o maximálním rozměru 256 krát 256 pixelů s 8 bitovou barevnou hloubkou, tj. 256 barev. V porovnání s M3G toto omezení barevné hloubky přináší výrazné snížení vizuální kvality vykreslovaných modelů. Textury nejsou uloženy v souborech *.mbac* spolu s modely, ale načítány z bitmapových souborů *.bmp*.

Texturu reprezentuje třída Texture. Na rozdíl od M3G API je zde situace značně jednodušší. Existuje zde přímo konstruktor, který dokáže načíst texturu z externího bitmapového souboru. Je zde také implementován druhý konstruktor, který načte texturu z bajtového pole a metoda *dispose()*, která texturu zruší. To je ovšem vše, žádné další vlastnosti ani nastavení, jako tomu je v MG3, tato třída neposkytuje. Výhodou je, že texturu lze již přímo přiřadit konkrétnímu modelu nebo modelům.

Výhodou MascotCapsule v práci s texturami je tedy především jednoduchost, nicméně to s sebou přináší značná omezení. Naopak lehce složitější přístup k texturám na straně M3G je vyvážen podporou dalších nastavení (blending, filtering, wrapping) a toto API poskytuje subjektivně kvalitnější texturování modelů.

5.4 Graf scény

Jedním z velkých rozdílů mezi oběma rozhraními je podpora grafu scény. Ten je implementován pouze v M3G API, v MascotCapsule V3 zcela chybí.

Celá scéna je tedy reprezentována stromem jednotlivých uzlů. Uzly v grafu scény mohou být objekty tříd, které dědí z třídy Node, tj. Camera, Group, Light, Mesh a Sprite3D. Třída Camera

reprezentuje bod, ze kterého je celá scéna pozorována. Třída Group je neuspořádanou skupinou několika jiných uzlů grafu. Speciální podtřídou třídy Group je třída World, což je kořenový uzel celého grafu. Třída Light reprezentuje světlo, třída Mesh 3D objekt scény a třída Sprite3D dvourozměrný obrázek umístěný v prostoru.

Při vytváření scén v externích 3D editorech a jejich exportu do souborů formátu *.m3g* se struktura grafu scény vytvoří automaticky, např. v 3ds Max jsou všechny jednotlivé 3D objekty potomky vrcholového uzlu World. Nicméně při exportu je možné si tento strom upravit dle vlastních potřeb. Samozřejmostí je možnost manipulace a vytváření grafu scény přímo v programu. K tomu slouží především metody třídy Group, jako *addChild()* a *removeChild()*.

5.5 Osvětlení scény

Velmi důležitým prvkem ve scéně je osvětlení, bez něj by nebylo ve scéně vidět naprosto žádné objekty.

V M3G je světelný zdroj reprezentován třídou Light. Ta dědí ze třídy Node, světlo může být tudíž také součástí grafu scény. Třída Node zase dědí ze třídy Transformable, proto může být se světlem manipulováno jako s kterýmkoliv jiným 3D objektem, tzn. měnit jeho pozici ve scéně a orientaci.
Světelný zdroj může být jedním ze čtyř typů, které M3G podporuje:

- **Ambient:** světlo tohoto typu osvětluje všechny objekty ve scéně ze všech směrů a jeho intenzita je stejná v jakékoliv vzdálenosti. Proto jeho pozice ve scéně ani nasměrování nemají na výsledné osvětlení objektů žádný vliv. Tento typ světla je výpočetně nejjednodušší.
- **Directional:** tento světelný zdroj taktéž osvětluje všechny objekty ve scéně se stejnou intenzitou nezávislou na vzdálenosti. Roli zde už ovšem hraje jeho nasměrování.
- **Omnidirectional:** jedná se o bodový zdroj světla, které z daného bodu vychází všemi směry. Umožňuje nastavit ubývání intenzity se vzdáleností od zdroje. U tohoto typu světelného zdroje hraje roli pouze jeho pozice, nasměrování nemá žádný vliv.
- **Spot:** Tento výpočetně nejnáročnější typ světla vrhá světelný kužel, roli zde proto hraje jak jeho pozice, tak jeho nasměrování. Taktéž umožňuje nastavit ubývání intenzity s rostoucí vzdáleností a exponent intenzity světla uvnitř kuželu.

U světelného zdroje lze kdykoliv libovolně jeho typ změnit. U typů, kde lze nastavit ubývání intenzity s rostoucí vzdáleností (Omnidirectional a Spot) lze zvolit buďto konstantní, lineární nebo kvadratické ubývání. U každého zdroje světla je možné také nastavit jeho barvu zadáním její RGB hodnoty a intenzitu.

Jak je tomu i v jiných oblastech, také v možnostech osvětlení scény je MascotCapsule značně jednodušší. Nelze zde vytvářet různé typy zdrojů osvětlení a ty pak libovolně umísťovat do scény, jako by to byly 3D objekty. Třída Light reprezentuje jen soubor nastavení, které pak při vykreslování používá Effect3D, svázaný s konkrétním objektem ve scéně. U objektu třídy Light je možné nastavit směrový vektor, intenzitu paralelního světla a intenzitu ambietního světla. Jak je vidět, ve scéně jsou jakoby dva zdroje světla (paralelní a ambientní). Jejich vliv na objekty lze pak proto nastavovat právě pomocí zadání jejich intenzit. Na rozdíl od M3G zde nelze nastavit barvu světeleného zdroje.

Srovnáme-li tedy obě API z pohledu osvětlení, M3G nabízí zcela jistě bohatší a komplexnější možnosti. Je ovšem zřejmé, že za cenu vyšších nároků na hardware mobilního zařízení.

5.6 Vzhled objektů

Tato podkapitola pojednává o tom, jakým způsobem obě API přistupují k uchování informací o vzhledu objektů a jejich nastavování.

V M3G k tomuto účelu slouží třída Appearance. Ta je složena z několika dalších komponent: Material, PolygonMode, CompositingMode, Fog a Texture2D. Každá tato komponenta má vlastní třídu stejného názvu:

- **Material** uchovává informace pro výpočet osvětlení, tj. hodnoty barev Ambient, Diffuse, Specular a Emmisive.
- **PolygonMode** uchovává nastavení týkající se vykreslování polygonů.
  - *Winding* – specifikuje, která strana polygonů bude přední (front face).
  - *Culling* – udává, zda-li budou vykresleny přední, zadní nebo obě strany polygonů.
  - *Shading* – stínování může být nastaveno jako FLAT nebo SMOOTH.
- **CompositingMode** udržuje nastavení týkající se pixelů (depth offset a blending).
- **Fog** definuje osvětlení v závislosti na vzdálenosti objektu.
- **Texture2D** uchovává texturu a k ní odpovídající nastavení.

Appearance obsahuje pouze reference na instance těchto tříd. To přináší především úsporu paměti, na několik objektů ve scéně tak může být například aplikován stejný Material a k tomu postačí pouze jediná instance třídy Material.

Po vytvoření nové instance třídy Appearance jsou všechny její komponenty nastaveny na hodnotu *null*. Změnit to lze pomocí odpovídajících metod, příkladem může být *setMaterial()*. Není potřeba, aby byly nastaveny všechny komponenty. Pokud má Texture2D hodnotu *null*, textura prostě nebude aplikována. V případě CompositingMode a PolygonMode rovných *null* bude při vykreslení použito výchozích hodnot. Když je komponenta Material nastavena na *null*, nebude počítáno osvětlení.

Ještě je zde vhodné zmínit, že referencí na Texture2D může mít Appearance i více než jednu, a to podle počtu dostupných texturovacích jednotek.

Třída uchovávající nastavení vzhledu objektů ve scéně byla v rozhraní MascotCapsule nazvána Effect3D. Není překvapením, že na rozdíl od Appearance v M3G, je značně jednodušší a neumožňuje tolik možných nastavení. V první řadě umožňuje nastavit typ stínování, buďto NORMAL nebo TOON. Při použití stínování typu TOON vypadají vykreslené objekty jakoby byly pouze dvourozměrné.

Dále pak třída Effect3D obsahuje referenci na objekt typu Light, který uchovává data o osvětlení (viz. Podkapitola 5.5 Osvětlení scény), možnost nastavení průhlednosti a enviromental mapping. Konkrétní instance Effect3D je spojena s modelem až při volání metody *renderFigure* pro vykreslení daného objektu.

5.7 Vykreslení scény (rendering)

V M3G existuje třída Graphics3D, která reprezentuje grafický kontext a zajišťuje vykreslení scény a objektů. Tato třída umožňuje vytvořit pouze jedinou instanci, jedná se totiž o tzv. singleton. Tuto instanci je třeba získat metodou *getInstance*. V případě, že aplikace potřebuje provést překreslení obrazovky, zavolá metodu *paint*. V této metodě je nejprve potřeba získat instanci Graphics3D a svázat ji s objektem typu Graphics z balíčku `javax.microedition.lcdui` metodou *bindTarget*. Tento objekt se tak stane cílem, do něhož bude vykreslení provedeno. Cílem může být ale také objekt typu Image2D. Poté je možné voláním metody *render* scénu vykreslit a nakonec je třeba ještě grafický kontext uvolnit od daného cíle metodou *releaseTarget*. Ukázka výše zmíněné třídy *paint*:

```
protected void paint(Graphics g) {
    Graphics3D g3d = Graphics3D.getInstance();
    g3d.bindTarget(g);
    g3d.render(world); //world reprezentuje kořen grafu scény
    g3d.releaseTarget();
}
```

Grafický kontext Graphics3D může být v jednom okamžiku svázán pouze s jedním cílovým objektem. Proto po zavolání metody *bindTarget* bude veškeré vykreslování prováděno do color bufferu daného cíle, až do zavolání metody *releaseTarget.*

M3G rozlišuje dva přístupy k vykreslování. Prvním je takzvaný **retained mode**, kdy je naráz vykreslována celá scéna, tj. volá se metoda *render* s parametrem udávajícím kořenový uzel grafu.

Druhým přístupem je **immediate mode**, kdy vykreslování probíhá zvlášť po jednotlivých uzlech grafu scény, skupinách nebo jednotlivých objektech.

U vykreslování lze také nastavit některé volby, ovlivňující výslednou kvalitu. Mezi ně patří antialiasing, dithering a zda-li použít true color barevnou hloubku. Tyto funkce s sebou ale pochopitelně přinášejí vyšší výpočetní nároky.

V MascotCapsule je situace velice podobná. Vykreslovací metody poskytuje taktéž třída Graphics3D. Vykreslovaní ale není voláno automaticky aplikací, muže být proto prováděno v libovolné metodě. Je ovšem potřeba zajistit periodické volání této metody, například z nějaké nekonečné smyčky. Ve vykreslovací metodě je také potřeba svázat grafický kontext s cílem, do kterého bude vykreslení provedeno. Taktéž to může být objekt typu Graphics z balíčku `javax.microedition.lcdui` případně objekt typu Image z téhož balíčku.

V MascotCapsule není implementován graf scény, proto nelze volat jedinou metodu nad uzlem grafu scény pro vykreslení všech objektů. To je třeba provést pro každý objekt ve scéně zvlášť. Pro vykreslení modelu typu Figure, načteného z externího souboru, je zde metoda *renderFigure*. Existuje zde také metoda *renderPrimitives,* která umí vykreslit model vytvořený nebo vygenerovaný přímo v programu, pokud jsou metodě předány potřebné parametry, jako pole souřadnic vrcholů, normály, textura atd.

Tyto metody ovšem neprovedou vykreslení ihned, pouze ho zaregistrují. Samotné vykreslení takto zaregistrovaných objektů proběhne až při zavolání metody *flush*. Tento postup ale obchází metoda *drawFigure,* která vykreslení provede ihned bez odkladu.

Na rozdíl od M3G nelze nastavovat funkce jako antialiasing, dithering nebo true color barevnou hloubku. Jinak je ale princip vykreslování u obou API velice podobný.

6 Srovnávací aplikace

Za účelem porovnání výkonu a rychlosti vykreslování obou API jsem vytvořil srovnávací aplikaci, jak ve verzi pro rozhraní MG3, tak pro MascotCapsule. Tato aplikace je jakýsi benchmark, skládající se z deseti testů. Každý z testů probíhá 10 sekund a v průběhu tohoto času je spočítáno, kolikrát došlo k překreslení obrazovky. Cílem bylo vytvořit rozmanité testy, například mnoho jednoduchých objektů ve scéně, naopak pouze jeden jednoduchý objekt, složitý objekt, více zdrojů světla, použití textur atd. Během testu je zobrazován aktuální počet FPS (snímků za sekundu) a čas, do kterého se nepočítá doba načítání modelů mezi jednotlivými testy. Nakonec jsou zobrazeny výsledky, tj. názvy jednotlivých testů a počet snímků, které byly reálně vykresleny.

Modely pro jednotlivé testy jsem vytvářel v 3D editoru Autodesk 3ds Max 9. Základem je vždy scéna s jednou kamerou, jedním nebo více světly a objektem, případně více objekty.Modely jsem poté exportoval do souborů formátu *.m3g* pro verzi aplikace testující M3G API a *.bac* pro verzi MascotCapsule.

Nejprve jsem vytvořil konečnou verzi této srovnávací aplikace pro M3G a následně ji přepracoval do verze pro MascotCapsule. Kvůli různým omezením a odlišnostem mezi oběma API nejsou tyto dvě verze aplikace naprosto a přesně totožné. Jednotlivé zásadní odlišnosti jsou popsány níže v popisech jednotlivých testů.

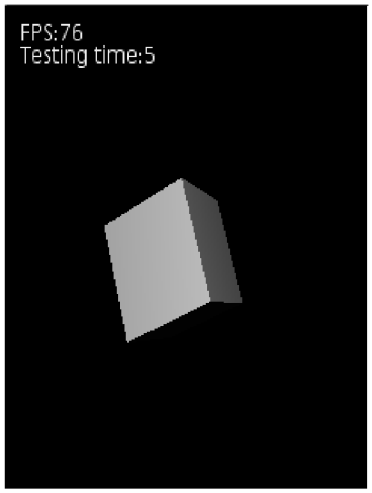
6.1 Popis jednotlivých testů

Test 1:

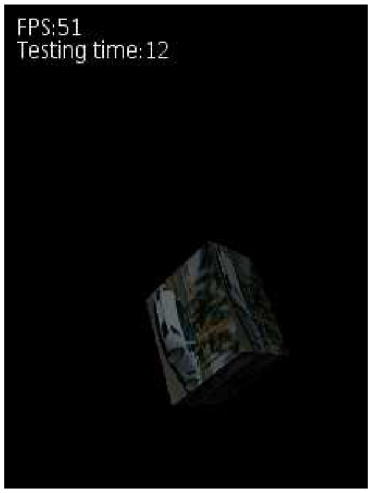
Základní scéna s kamerou, jedním světlem a krychlí, která rotuje.

Test 2:

Stejná scéna jako v testu 1, ovšem na rotující krychli je aplikována textura. Ve verzi pro M3G je textura uložena přímo v souboru s celou scénou. Ve verzi MascotCapsule nejsou textury uloženy v souborech společně s modely, proto je textura v samostatném bitmapovém souboru *.bmp*. MascotCapsule navíc podporuje pouze textury s 8 bitovou barevnou hloubkou.



Obrázek 6.1: Test 1



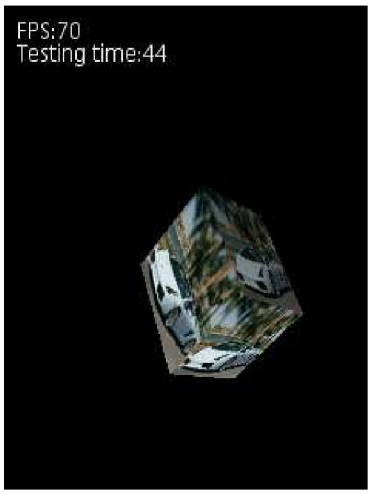
Obrázek 6.2: Test 2

Test 3:

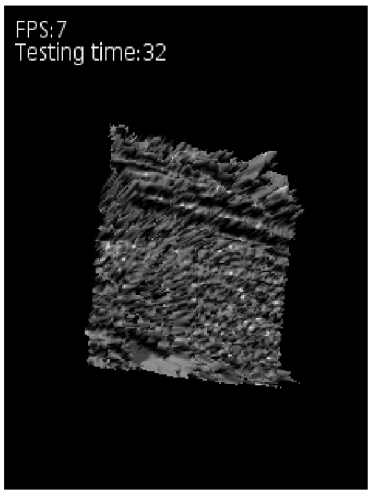
Scéna jako v předchozím testu, na rozdíl od něj obsahuje 4 bodová světla typu Omnidirectional. Tento test je ovšem použit pouze v M3G verzi. MacsotCapsule neumožňuje použít 4 světla, proto je místo testu 3 zopakován test 2.

Test 4:

Základní scéna s rotujícím složitým modelem, který má 10201 vrcholů. Byl vytvořen ze základního objektu rovina (v 3ds Max nazvaným Plane) aplikací modifikátoru Displace, který jej podle bitmapy zdeformoval do složitého tvaru. Zde jsem opět narazil na omezení MascotCapsule, a to v podobě maximálního počtu vrcholů, které může model mít, konkrétně 21854. Původní model v tomto testu tuto hranici zdaleka přesahoval, proto jsem jej musel zjednodušit a zmenšit počet vrcholů, těsně pod tuto hranici. Zde jsem ale narazil na další problém. Tím je nedostatek paměti u starších modelů mobilních telefonů, pro které byl i tento zjednodušený model příliš paměťově náročný. Abych umožnil spuštění tohoto testu i na starších telefonech, zmenšil jsem tedy složitost modelu na konečných 10201 vrcholů.



Obrázek 6.3: Test 3



Obrázek 6.4: Test 4

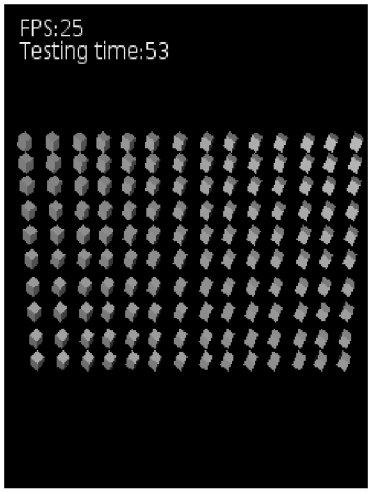
Test 5:

Tento test je totožný s testem 4, na model je navíc aplikována textura.

Test 6:

Scéna obsahuje 140 rotujících krychlí bez textury. Ve verzi M3G obsahuje scéna 140 samostatných krychlí, ke kterým se přistupuje pomocí pole objektů typu Mesh. Při každém vykreslení nového snímku se v cyklu polem prochází a pro každou krychli je volána metoda provádějící rotaci. Tento přístup ovšem není možný ve verzi pro MascotCapsule. Zde nelze ze souboru *.mbac* získat reference na jednotlivé části (krychle) scény a tyto si poté uložit do pole či kolekce. Modelem lze proto rotovat pouze jako celkem, ne jen jednotlivými krychlemi zvlášť. Použil jsem proto model samostatné krychle z testu 1, který je v každém snímku příslušně rotován, a poté ve dvou vnořených cyklech duplikován a vykreslen do požadovaného rozmístění (14 sloupců, 10 řádků).

Obrázek 6.5: Test 5



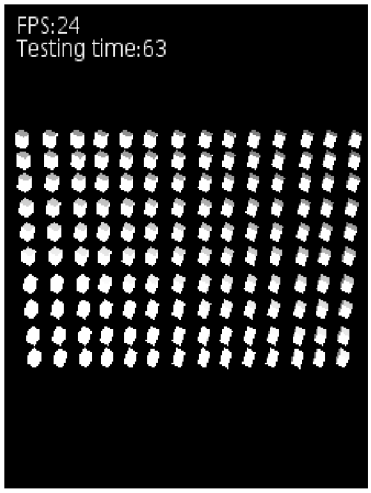
Obrázek 6.6: Test 6

Test 7:

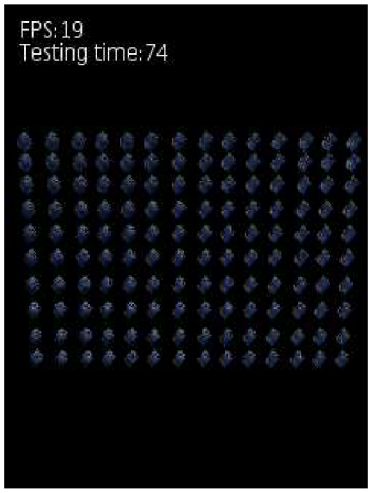
Stejný jako test 6, na rozdíl od něj je v M3G verzi nasvícena scéna čtyřmi světly. Ve verzi MascotCapsule je podobně jako v testu 2 a ze stejného důvodu zopakován předchozí test.

Test 8:

Opět scéna se 140 rotujícími krychlemi. Na všechny krychle je navíc aplikována textura.



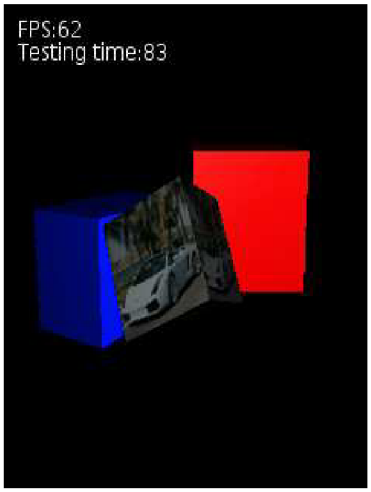
Obrázek 6.7: Test 7



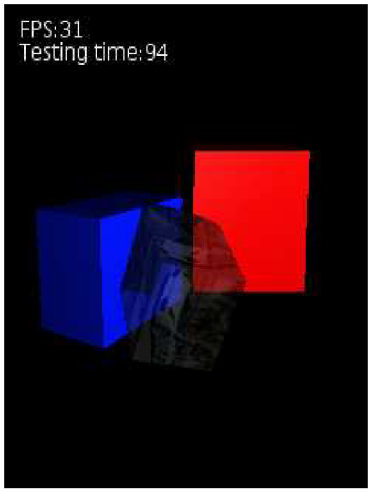
Obrázek 6.8: Test 8

Testy 9 a 10:

Poslední dva testy jsou zaměřeny na průhlednost objektů. Scéna je složena z rotující rychle s texturou a dvěma statickými krychlemi v pozadí. V testu 9 není použita průhlednost, slouží pro porovnání výsledků s testem 10, kde je rotující krychli s texturou nastavena 55% průhlednost. Ve verzi MascotCapsule opět není celá scéna v jednom souboru. Modely statických krychlí v pozadí a krychle s texturou jsou zvlášť ve dvou souborech.



Obrázek 6.9: Test 9



Obrázek 6.10: Test 10

6.2 Výsledky

Po dokončení obou verzí a otestování na softwarových emulátorech jsem srovnávací aplikaci spouštěl na několika mobilních telefonech a sbíral výsledky testů. Ideální pro srovnání abou API jsou samozřejmě telefony značky Sony Ericsson, které podporují jak M3G, tak MascotCapsule. Nicméně testy jsem prováděl i na telefonech jiných značek, které podporují pouze MG3 API.

Kromě reálných mobilních telefonů jsem tyto testovací aplikace také pro zajímavost spouštěl na softwarových emulátorech a taktéž sbíral výsledky. Emulátory jsem využíval při vývoji a testování daleko častěji, než testování aplikace přímo na fyzickém mobilním telefonu. Při implementaci testů pro M3G jsem používal nejčastěji emulátor od společnosti Sun Microsystems, který je součástí balíku nástrojů *J2ME Development Kit*. Občas jsem využíval také emulátor zařízení s operačním systémem Symbian S60, ten je součástí *Nokia S60 3rd Edition SDK*. Oba emulátory ovšem nepodporují rozhraní MascotCapsule. Proto jsem při implementaci srovnávací aplikace pro toto API využíval softwarové emulátory od společnosti Sony Ericsson. Za zmínku stojí, že vývojový balíček S*ony Ericsson SDK* obsahuje skutečně pestrou škálu emulátorů různých modelů.

Následující tabulka přináší přehled získaných výsledků (viz. Tabulka 6.1). U každého modelu mobilního telefonu je uvedeno, která verze srovnávací aplikace a pro jaké API na něm byla spuštěna. Čísla u jednotlivých testů vyjadřují počet snímků, které byly během každého testu za 10 sekund vykresleny. Zvláště na starších telefonech se stávalo, že některé testy nebyly vůbec spuštěny z důvodu nedostatku volné paměti. V těchto případech je v tabulce uvedeno číslo 0.

Počet vykreslených snímků v jednotlivých testech

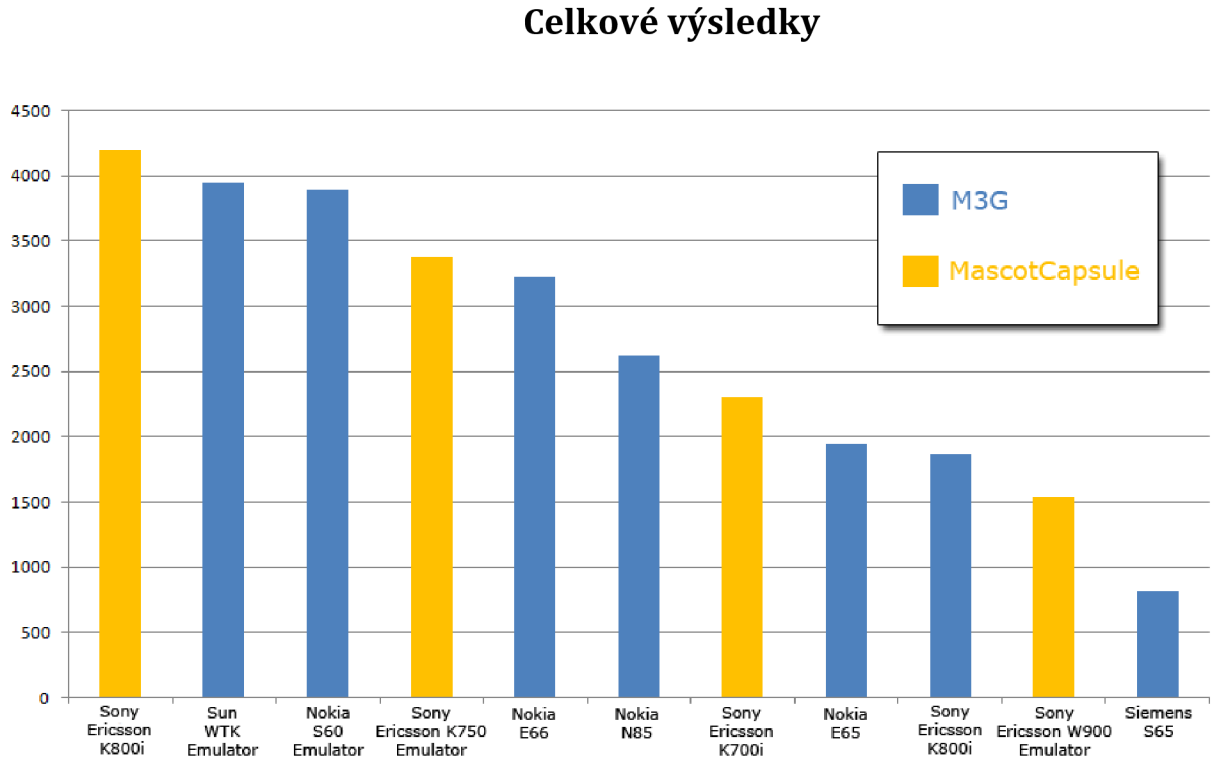
Telefon	API	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Nokia E65	M3G	506	365	355	21	18	68	61	50	245	246
Nokia E66	M3G	827	571	571	36	29	126	129	91	446	390
Nokia N85	M3G	565	451	452	31	27	156	133	100	367	339
Sony Ericsson K800i	M3G	543	376	365	10	7	80	54	45	209	176
Sony Ericsson K800i	MC	724	716	719	110	99	143	144	127	709	710
Sony Ericsson K700i	MC	479	443	449	12	10	63	63	61	389	330
Nokia S60 Emulator	M3G	739	606	691	57	62	233	241	185	620	452
Sun WTK Emulator	M3G	613	613	455	116	90	413	378	328	465	470
Sony Ericsson K750 Emulator	MC	314	325	324	441	432	301	316	315	287	323
Sony Ericsson W900 Emulator	MC	116	149	167	147	161	167	152	170	153	153
Siemens S65	M3G	226	181	182	8	0	0	0	0	104	108

Tabulka 6.1: Výsledky jednotlivých testů

Vysvětlivky:

- *M3G – výsledky testů aplikace využívající JSR 184: Mobile 3D Graphics API*
- *MC – výsledky testů aplikace využívající rozhraní MascotCapsule V3*
- *Čísla udávají, kolik snímků bylo reálně vykresleno za dobu trvání testu, tj. 10 sekund*

Po sečtení výsledků všech testů a seřazení získáme graf porovnávající výkonnost jednotlivých modelů mobilních telefonů:



Následující tabulka (viz. Tabulka 6.2) přináší přehled několika vybraných údajů o testovaných mobilních telefonech [12]. Důležitým sloupcem (Hardware) je údaj o použitém procesoru, případně velikosti dostupné RAM. Na tomto údaji lze sledovat, jaký vliv mají právě tyto parametry na rychlost vykreslování 3D grafiky. Podle roku uvedení na trh lze také sledovat postupný vývoj jak hardwaru, tak nárůstu výpočetní výkonnosti. Posledním údajem je informace o verzi použitého operačního systému, pokud jím daný model disponuje. Operační systém mobilního zařízení totiž může mít pochopitelně na rychlost vykreslování také jistý vliv.

Výrobce	Model	Rok uvedení na trh	Hardware	Operační systém
Sony Ericsson	K800i	2006	210Mhz	Bez OS
Sony Ericsson	K700i	2004	110Mhz	Bez OS
Nokia	E65	2007	220MHz, 20MB	Symbian S60 v9.1
Nokia	E66	2008	369MHz, 128MB	Symbian S60 v9.2
Nokia	N85	2008	369MHz, 128MB	Symbian S60 v9.3
Siemens	S65	2004	100MHz	Bez OS

Tabulka 6.2: Technické parametry a další informace o testovaných mobilních telefonech

6.3 Zhodnocení výsledků

Podívejme se nejprve na výsledky testů provedených na modelu Sony Ericsson K800i, na kterém bylo možné spustit verze testů pro obě API. Zde MascotCapsule dosahuje jednoznačně lepších výsledků ve všech testech. Zejména u testů číslo 4 a 5 (rotace složitého modelu) je počet vykreslených snímků až 10 krát větší ve prospěch MascotCapsule. Nicméně je třeba podotknout, že MascotCapsule podporuje pouze 8bitovou barevnou hloubku textury, což pochopitelně přináší výhodu z pohledu výkonu, ale na druhou stanu značně horší vizuální kvalitu zobrazení. Také u ostatních testů je vizuální kvalita vykreslených modelů subjektivně nižší v neprospěch MascotCapsule.

Pokud porovnáme model K800i s výkonnostně podobným modelem E65 od společnosti Nokia, zjistíme, že výsledky v M3G jsou de facto srovnatelné, s drobnými rozdíly v jednotlivých testech. Na novějších modelech E66 a N85 značky Nokia je vidět, jak velkou roli hraje použitý procesor a větší RAM paměť. Nárůst výkonu a počtu vykresleních snímků oproti modelu E65 je místy téměř dvojnásobný.

Velice zajímavá situace ovšem nastává u testů 4 a 5 (rotace složitého modelu). I když jsou výsledky poněkud lepší, než u staršího modelu E65, stále dosahují velmi nízkých hodnot. V porovnání s MascotCapsule výsledky modelu Sony Ericsson K800i byla vykreslena pouze asi jen přibližně třetina snímků. Také ostatní výsledky testů na tomto telefonu jsou srovnatelné a místy i předčí výkon modelů Nokia E66 a N85. A to jsou tyto telefony o 2 roky novější, s rychlejším a výkonnějším procesorem, mají operační systém a spadají do kategorie chytrých telefonů.

K podobnému zjištění dojdeme i při porovnání staršího modelu Sony Ericsson K700i z roku 2004 s chytrým telefonem Nokia E65 z roku 2007. MascotCapsule test na starším z modelů dosahuje relativně podobných výsledků jako M3G verze na novější Nokii. Nicméně je třeba dodat, že K700i disponuje displejem s menším rozlišením a schopností zobrazit nižší počet barev.

Zajímavé je také srovnání modelu K700i s telefonem Siemens S65. Oba byly uvedeny na trh v roce 2004 a používají podobné procesory. Hodnoty výsledků Siemensu S65 podporujícího M3G jsou velmi nízké. Některé testy z důvodu přílišné paměťové náročnosti na tomto modelu ani nebyly provedeny. Ovšem mobilní telefon Sony Ericsson K700i dosahuje v MascotCapsule testech velmi solidních výsledků a všechny testy proběhly korektně.

Podívejme se nyní na porovnání výsledků mezi jednotlivými testy. Test číslo 1 obsahuje rotující krychli, v testu 2 je krychli přidána textura. V M3G verzi toto způsobí pochopitelně pokles počtu vykreslených snímků a můžeme si všimnout, že celkem znatelný. Ovšem zajímavé je, že ve verzi MascotCapsule je pokles výkonu výrazně menší. Test číslo 3 přidává další 3 světla (pouze v M3G), ale na pokles výkonu to překvapivě nemá skoro žádný vliv. Co ale není překvapením je prudký pokles počtu vykreslených snímků v testech 4 a 5, což je pochopitelné vzhledem ke složitosti modelu. V testech 6, 7 a 8, kde je základem skupina rotujících krychlí, nastává situace, kdy M3G výsledky nových Nokií E66 a N85 přibližně odpovídají výsledkům MascotCapsule telefonu K800i. Taktéž starší Nokia E65 dosahuje výsledků Sony Ericssonu K700i. Mohlo by se na první pohled zdát, že v případě velkého počtu jednoduchých objektů ve scéně se výkon obou API vyrovnává. Nesmíme ale opomenout fakt, že ve v obou případech mají Sony Ericssony mnohem slabší procesory. Podobně jako u prvních tří testů, i zde je situace podobná, přidání osvětlení má vliv jen minimální, přidání textur se již projeví. Z posledních dvou testů s průhledností zjistíme, že podle očekávání použití průhlednosti jistý vliv má, ovšem ne příliš výrazný.

Z výsledků jednoznačně vyplívá, že z pohledu výkonnosti jasně vítězí MascotCapsule. Toto rozhraní umožnilo provozovat už na starších a výpočetně slabších modelech relativně plynule běžící 3D hry a aplikace. Nevýhodou je ovšem podpora jen ze strany modelů značky Sony Ericsson. Navíc v dnešní době výkonnost mobilních telefonů natolik vzrostla, že už i M3G aplikace podávají použitelný výkon.

7 Demonstrační aplikace

7.1 Návrh a volba rozhraní

Jako demonstrační aplikaci jsem se rozhodnul implementovat jednoduchou 3D hru. Cílem této hry je procházet s postavičkou místnostmi, které jsou vzájemně propojeny dveřmi a dohromady tvoří jakési prostorové bludiště. Několik různě propojených místností tvoří dohromady level, k jeho úspěšnému dokončení je třeba projít speciálními finálními dveřmi. Spletitost levelů a počet místností pochopitelně postupně narůstá.

Celé bludiště je obsaženo jakoby v jedné krychli. Pohledem na přední stěnu této krychle je vidět uvnitř právě jedna místnost. Pokud postavička projde dveřmi do další místnosti, krychle se otočí a pohledem na jinou stěnu uvidíme místnost, do které postavička nově vešla. Tato hra občas cíleně popírá fyzikální a optické zákony, proto orientace nové místnosti může být libovolná.

Jednotlivé místnosti jsou barevně rozlišeny, obsahují dveře do jiných místností, patra a případně schody. Místnosti jsou pokaždé generovány náhodně, to znamená, že při každém spuštění vypadají zcela jinak.

Ovládání hry je velice jednoduché. Pomocí kurzorových kláves nebo joysticku mobilního telefonu lze celou krychlí naklánět. Postavička na to poté reaguje a rozejde se směrem, kterým je krychle nakloněna. Pokud postavička vejde před dveře, automaticky se krychle pootočí a postavička vejde do další místnosti. Pokud použije stejné dveře, kterými přišla, vrátí se zpět do předchozí místnosti. Tato hra tedy neslouží pouze k zábavě, nutí také hráče používat prostorovou představivost a vizuální paměť.

Pro implementaci bylo možné samozřejmě použít jak rozhraní M3G, tak MascotCapsule. Má volba padla na M3G především díky jeho široké podpoře ze strany mobilních telefonů, aplikace není tedy omezena pouze na telefony značky Sony Ericsson. Navíc sám vlastním přístroj značky Nokia, což mi umožnilo hru testovat nejen na softwarovém emulátoru, ale také přímo na fyzickém zařízení. M3G také poskytuje bohatší funkcionalitu.

7.2 Implementace

7.2.1 Požité nástroje

Při vývoji aplikace jsem používal následující nástroje:

- **NetBeans IDE 6.1** – vývojové prostředí pro psaní samotného kódu
- **Java Platform Micro Edition Software Development Kit 3.0** – obsahuje nutné nástroje pro vývoj mobilních aplikaci včetně softwarového emulátoru.
- **Nokia S60 3rd Edition SDK for MIDP** – nástroje pro vývoj Java aplikací pro zařízení s operačním systémem Symbian.
- **Autodesk 3ds Max 9** – 3D editor, ve kterém byly vytvořeny všechny modely.
- **Adobe Photoshop CS** – grafický editor použitý pro vytvoření menu.

7.2.2 Postup implementace

Úplně prvním krokem při tvorbě této demonstrační aplikace bylo vytvoření modelu první jednoduché místnosti, import do aplikace a zajištění jejího naklánění při stisku odpovídající klávesy. Poté bylo

potřeba vymyslet systém, jakým se bude postavička v místnosti pohybovat. Nesmí procházet stěnami, naopak pokud vykročí z vyvýšeného místa, musí vlivem gravitace spadnout dolů. Také musí být schopna používat korektně schody, jak směrem nahoru, tak směrem dolů. Jednoduchým řešením tohoto problému bylo použití matice o rozměru 10 krát 10, která je jakousi mapou celé místnosti. Rozděluje ji na 100 segmentů a k nim uchovává odpovídající hodnotu. Například číslo 3 v matici udává, že na dané pozici je úroveň podlahy ve výšce 30 (rozměry místnosti ve všech směrech jsou 100). Pokud by byl v místnosti sloup až po strop, na dané pozici v matici by bylo číslo 10, atd.

Aby se odlišily schody od běžné podlahy, jejich výška je zadána záporným číslem, tedy například -1 udává segment schodů vedoucí do výšky 10.

Díky této matici bylo tedy možno implementovat pohyb postavičky (zatím pouze jednoduchého kvádru) po místnosti. Dále jsem začal modelovat další místnosti, což se ovšem ukázalo jako velice pracné a časové náročné. Navíc vyvstal problém s paměťovou náročností, pokud by totiž každá místnost měla být samostatným modelem v novém souboru, aplikace by byla příliš velká už při malém počtu místností. Rozhodl jsem se proto pro automatické náhodné generování místností. K tomu se ideálně hodí již dříve zmíněná matice popisující místnost. Nejenže umožňuje pohyb postavičky, ale tuto matici lze relativně jednoduše vygenerovat a pak už jen stačí vytvořit funkci, která danou místnost podle zadané matice vytvoří a poskládá.

Při generování matice je třeba dbát především na to, aby byla vzniklá místnost smysluplná. Schody musí navazovat, musí být možné dostat se ke všem dveřím a postavička nemůže nikde uváznout. Před vytvořením funkce, která celou místnost sestaví, bylo potřeba vytvořit v 3D editoru novu scénu. Ta obsahuje prázdnou místnost a jednotlivé elementy, ze kterých je celá místnost následně poskládána (viz. Obrázek 7.4). Sestavovací funkce potom tyto elementy duplikuje a rozmísťuje na požadovanou pozici podle vygenerované matice. Díky tomu tedy postačuje pouze jeden *.m3g* soubor a výsledná velikost aplikace je tak razantně zmenšena.

Dalším krokem bylo vytvoření samostatné třídy reprezentující jednu místnost. Ta uchovává již zmíněnou matici, barvu místnosti, dále také pozici a počet dveří a informace o tom, do kterých dalších místností dané dveře vedou. Do této třídy bylo přesunuto také generování. Nyní již proto bylo možné vytvářet náhodně generované místnosti i s dveřmi, a ty pak vzájemně propojit a vytvořit tak celý level. Samotný přechod do další místnosti probíhá tak, že postavička vejde do dveří, celá krychle se pootočí odpovídajícím směrem, uvnitř krychle se vymění její obsah (místnost) a rotace krychle se dokončí.

Aby byly jednotlivé místnosti snadněji rozpoznatelné, každá z nich má nejen jiné rozmístění schodů a dveří, ale také jinou barvu stěn. Původní záměr byl generovat barvu místnosti zcela náhodně, to s sebou ale přináší značný problém. Náhodná barva může být často nevhodná, například příliš tmavá nebo příliš křiklavá a podobně. Proto jsem se rozhodl vytvořit další třídu, která reprezentuje paletu několika použitelných barev. Z té jsou pak náhodně vybírány barvy a přiřazovány jednotlivým místnostem.

V této fázi vývoje aplikace nastal čas začít se věnovat také vizuální stránce. To znamená především použití textur, vymodelování postavičky a jednoduché animace její chůze.

Důležitou součástí her i aplikací obecně je ovládací menu. V této aplikaci umožňuje spustit novou hru, zobrazit nápovědu, změnit nastavení a celou aplikaci ukončit. Zobrazení jednoduché nápovědy je dle mého názoru velmi důležité, zvláště pak u her. Nastavení původně u této hry plánováno nebylo, ovšem zejména po přidání textur značně vzrostla výpočetní náročnost, proto jsem se rozhodnul tuto položku do menu přidat. V ní lze totiž mimo jiné použití textur vypnout.

Následujícím krokem bylo vytvoření dalších obrazovek, oznamujících konec levelu, dokončení hry a přerušení hry v jejím průběhu. Závěrečná fáze vývoje znamenala přidání počítadla FPS (vykreslených snímků za sekundu), možnost zobrazení času, dolaďování detailů, zlepšování vizuální stránky a přidávání dalších nových levelů.

7.2.3 Třídy

Celá demonstrační aplikace je obsažena v balíčku *cube3D*, ten obsahuje následující třídy:

- **main** – hlavní třída, která dědí ze třídy MIDlet a implementuje rozhraní CommandListener. Je nezbytná pro spuštění, musí ji obsahovat každá mobilní aplikace profilu MIDP.
- **scene3D**- třída zajišťující vykreslování grafiky, běh aplikace, ovládání i samotnou herní logiku. Dědí ze třídy Canvas a implementuje rozraní Runnable.
- **room** – objekt této třídy reprezentuje jednu místnost. Obsahuje matici, která místnost popisuje a další vlastnosti, jako pozice dveří, kam vedou a barvu stěn. Obsahuje metody umožňující tyto vlastnosti nastavovat a získávat. Obsahuje také metodu pro náhodné vygenerování matice.
- **colorPool** – tato třída je paletou barev (materiálů). Obsahuje pole prvků typu Material a metodu, která vrátí Material z daného indexu.

7.2.4 Metody třídy scene3D

Jak již bylo zmíněno, jedná se o stěžejní třídu celé aplikace. Obsahuje následující metody (metody jsou veřejné a nevrací žádnou hodnotu, proto jsou *public viod*)*:*

switchLevel(int number) – Vytvoří level s číslem zadaným jako parametr. Nejprve je vytvořeno pole objektů typu room. Tento typ používá konstruktor, který vygeneruje místnost s požadovaným počtem dveří. Následně jsou místnosti navzájem propojeny pomocí metod *setLink(int n)* třídy room. Po propojení místností je každé z nich náhodně přiřazen materiál získaný z palety materiálů (objekt typu colorPool). Důležitým krokem je také nastavení proměnné `finalLevel`, která označuje číslo finální místnosti. Do té budou potom umístěny cílové dveře, do kterých je cílem postavičky dostat se. Jak je vidět, při každém spuštění levelu vypadají jednotlivé místnosti různě. Co je ovšem pokaždé stejné, je tedy pouze jejich vzájemné propojení. Přidání nového levelu do hry je velice snadné, stačí pouze přidat další větev do příkazu *switch*, vytvořit pole objektů room a vzájemně je propojit.

start() – Tato metoda obsahuje počáteční incializaci aplikace, tj. vytvoření a nastavení potřebných proměnných, načtení obrázků z externích souborů atd.

buildRom(int level) – Podle zadaného čísla sestaví odpovídající místnost levelu. K tomu využívá matici popisující každou místnost. Tu postupně prochází a na pozice označené číslem 3 umístí kvádr o výšce 30. Ten získá duplikací kvádru steps3, který je obsažen v souboru se scénou. Dále pak následuje rozmístění schodů. Schody jsou v matici označeny čísly -1, -2 a -3, k nim existují v souboru se scénou odpovídající vzory, které jsou taktéž duplikovány a rozmísťovány. U schodů je ovšem ještě potřeba nastavit správnou orientaci.

Sestavení místnosti pokračuje rozmístěním dveří. Každá místnost může mít maximálně 4 dveře, kterým odpovídají 4 proměnné, udávající jejich pozici. Pokud není tato proměnná hodnoty null, jsou na danou pozici dveře umístěny. Ty mají v souboru se scénou 3 vzory: plné dveře, průhledné dveře a finální dveře (označující konec levelu). Většina dveří vznikne duplikací z plného vzoru, výjimku tvoří pouze ty, které jsou umístěny na přední stěně krychle, u těch jsou vzorem dveře průhledné. Nakonec je testováno, zda-li je aktuální místnost závěrečnou místností levelu. Pokud ano, jsou do místnosti přidány finální dveře.

switchStartRoom() – Metoda volaná při spuštění každého levelu. Zajistí načtení scény a objektů ze souboru *.m3g* a získá reference na jednotlivé objekty pomocí jejich ID. Podle nastavení aplikace přiřadí modelům vzhled a barvy, nebo ponechá původní textury. Nastaví taktéž typ osvětlení a provede další potřebné akce pro spuštění levelu.

switchRoom(int level) – Podle zadaného parametru přepne a zobrazí požadovanou místnost. Nejprve vyčistí vnitřek krychle tím, že odstraní ze scény všechny elementy vygenerované v předchozí místnosti. Poté nastaví pozici postavičky tak, aby stála přede dveřmi, kterými do nové místnosti vstoupila. Dále zajistí inicializace potřebných proměnných a volá metodu *buildLevel*, která danou místnost sestaví.

Metody *rotateLeft()*, *rotateRight()*, *rotateUp()* a *rotateDown()* zajišťují animace rotace krychle, v případě kdy postavička přechází z jedné místnosti do jiné. Jsou využívány metodou *rotateCube*.

Metody *fadeIn()* a *fadeOut()* jsou volány na začátku, resp. po dokončení levelu. Postupně přidávají nebo ubírají intenzitu osvětlení a první z nich navíc vynuluje počítadlo času.

run() – Základem metody je nekonečná smyčka, která zajišťuje běh aplikace. V první řade obstarává pohyb postavičky po místnosti. Pokud je zjištěno, že je krychle nakloněna, postavička je posunována příslušným směrem. I zde se využívá matice popisující místnost, podle ní je postavička umísťována do patřičné výšky. Je zde také zajištěno, aby nebylo možné vejít do zdi, aby z vyvýšeného místa spadla postavička dolů a také aby šla po schodech postupně, a ne třeba jedním velkým skokem. Je zde ovládána také jednoduchá animace naznačující chůzi.

V nekonečné smyčce se také neustále kontroluje, zda-li není postavička na pozici dveří. Pokud ano, zavolá se metoda rotující krychlí v požadovaném směru, získá se informace o čísle místnosti, do které se má vstoupit a tato místnost je poté zobrazena. V případě, že je aktuálně zobrazená místnost také místností finální, kontroluje se, zda-li postavička nevstoupila do finálních dveří. Pokud ano, level je dokončen a zobrazí se patřičná obrazovka.

rotateCube(int smer, int link) – Tato metoda zajistí přechod mezi dvěma místnostmi. Jednak zajistí animaci rotace krychle, ale také přepnutí místnosti v průběhu rotace. Díky této metodě se tedy po vstoupení do dveří krychle otočí do jiné místnosti. Parametry udávají směr rotace a číslo místnosti, do které má být přepnuto.

loadWorld(String path) – Z cesty zadané parametrem načte soubor *.m3g* a vyhledá v něm referenci na kořenový uzel grafu scény typu World.

drawMenu(Graphics g) – Vykreslí hlavní menu aplikace. To je tvořeno obrázky načtenými z externích .png souborů. Tento postup zajišťuje, že menu bude na všech telefonech vypadat stejně. Pokud by totiž bylo menu vykreslováno přímo metodami třídy Graphics, především u textů nastává problém. Každé zařízení může totiž texty vykreslovat jinak. Obrázky mají rozměry 176 krát 220 pixelů a jsou centrovány na střed displeje. Proto nejmenším rozlišením, na kterém bude aplikace vypadat korektně je právě tento rozměr.
Podobně je tomu u dalších menu a obrazovek. Pro vykreslení nastavení aplikace slouží metoda *drawSettings*, pro nápovědu *drawHelp*, pro přerušení hry *drawQuit* a pro dokončení levelu *drawLevelComplete*.

Ukázka obrazovek vykreslených předchozími metodami

Obrázek 7.1: drawMenu

Obrázek 7.2: drawQuit

Obrázek 7.3: drawLevelComplete

paint(Graphics g) – Metoda zajišťuje rendering a je volána automaticky v případě, kdy aplikace požaduje překreslit obrazovku. V této metodě se také podle řídicí proměnné vybírá, zda-li vykreslit 3D scénu, menu nebo některou z dalších obrazovek.

keyPressed(int keyCode) – V případě, že je stisknuta některá klávesa na mobilním telefonu, aplikace zavolá tuto metodu. Podle kódu lze poté rozlišit, o jakou klávesu se jedná a patřičně na její stisk reagovat. Například pokud je aktuálně zobrazena 3D scéna s místností, metoda reaguje na zmáčknutí kurzorové klávesy nakloněním krychle v daném směru a uvede do pohybu postavičku. Pokud je zobrazeno menu, metoda zajistí odpovídající pohyb kurzoru atd. Tato metoda je *protected*.

keyReleased(int keyCode) – Metoda je volána při uvolnění klávesy a slouží k navrácení původní orientace krychle a zastavení animace postavičky. Tato metoda je *protected.*

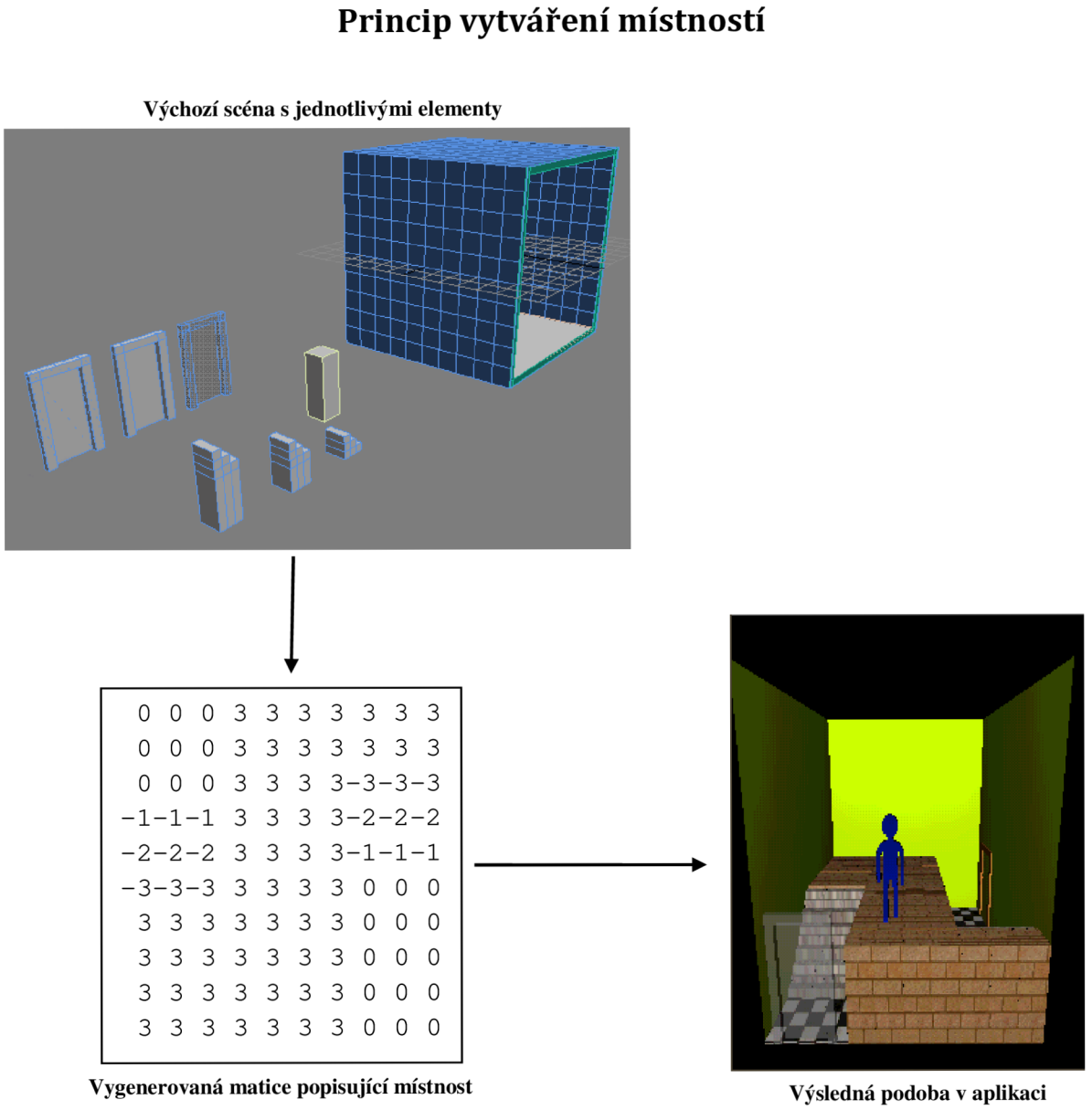
7.2.5 Generování místností

Za zmínku stojí ještě detailněji rozebrat princip generování místnosti. Ta je reprezentována objektem třídy room. Tato třída obsahuje konstruktor *room(int number)*, který vytvoří instanci místnosti s počtem dveří zadaným parametrem `number`. K samotnému vygenerování matice, která místnost popisuje, slouží metoda *generateRoom(int doorCount).*

Náhodná čísla jsou získávána pomocí pseudonáhodného generátoru třídy Random z balíčku `java.util`. S jejich pomocí pak vzniká výsledná podoba. Místnost je rozdělena na 3 pruhy vodorovné se směrem kamery o šířkách 30, 40 a 30. Krajní dva obsahují schody. Pozice schodů je vygenerována náhodně, taktéž jejich orientace (směrem k přední nebo zadní stěně krychle). V prostředním pruhu schody nejsou, pouze se náhodně vybere, zda bude podlaha na úrovni spodní stěny krychle, nebo vyvýšena do úrovně schodů. Dále jsou do místnosti umístěny dveře. Jejich počet může být 1 až 4, tento počet udává parametr `doorCount`. Dveře jsou umístěny tak, aby logicky zapadaly do místnosti, tj. pokud je místnost vyvýšena, dveře jsou umístěny pod schody a podobně. Pozice a počet dveří nejsou nijak obsaženy v matici. K tomu to účelu existují čtyři třídní proměnné nazvané `doorPositionA` až `doorPositionD`. Jsou to pole o dvou celočíselných prvcích, udávajících pozici dveří v místnosti a jejich výchozí hodnotou je *null*. Podle požadovaného počtu dveří v místnosti jsou však postupně nastavovány. Pokud má mít tedy daná místnost pouze jedny

dveře, pozici má nastavenu proměnná `doorPositionA`, ostatní jsou *null*. Pokud má mít místnost dvoje dveře, hodnotu pozice mají nastavenu pouze `doorPositonA` a `doorPositionB` atd.

Jak již bylo zmíněno dříve, samotné rozmístění elementů a sestavení místnosti je uskutečněno v metodě *buildRoom* třídy *s*cene3D.



Obrázek 7.4: Znázornění principu, jakým jsou ve hře vytvářeny jednotlivé místnosti

7.3 Návrhy na další vylepšení aplikace

Jako každou aplikaci, i tuto by bylo samozřejmě možné dále vylepšovat a vyvíjet. Zcela jistě by šlo relativně jednoduše vytvořit další nové levely. K tomu by postačilo vymyslet propojení místností a přidat jej jako další větev konstrukce *switch* do metody *switchLevel()* třídy scene3D.

Také by bylo zajímavé rozšířit automatické generování tak, aby byly generovány mnohem rozmanitější místnosti. To znamená vytvořit další elementy, různé typy schodů, podlah, sloupů, překážek atd. To by znamenalo především rozšíření metody *generateRoom* třídy room. V případě přidání nových elementů do místnosti by je bylo nutné nejprve vytvořit ve 3D editoru a vyexportovat nový .m3g soubor. Také by to vyžadovalo zásah do metody *switchStartRoom* za účelem získání referencí na tyto nově vytvořené objekty.

Možným vylepšením by bylo také zdokonalení vzhledu aplikace, tj. nejen generovat náhodně rozložení místností, ale také materiály, barvy a textury jednotlivých elementů. Pokud by byla aplikace provozována na výkonných zařízeních, ty by zcela jistě zvládly také dokonalejší a plynulejší animaci chůze postavičky.

Velmi zajímavým vylepšením aplikace aby byla také implementace automatického generování nejen místností, ale také celých levelů, tzn. propojení mezi místnostmi. Levely by už tedy nebyly staticky propojeny ručně, ale pokaždé by vypadaly zcela odlišně.

Ke zvýšení herní obtížnosti by mohlo přispět vytvoření takového vzájemného propojení místností, kdy by také záleželo na směru průchodu dveřmi. Při průchodu by dveře z jedné strany vedly do úplně jiné místnosti, než ze které těmito dveřmi postavička vešla. Tím by hra daleko více popírala prostorové zákonitosti a daleko víc by nutila hráče přemýšlet.

Velkým zpestřením by také bylo implementovat podporu pohybových senzorů, které se v mobilních telefonech stále častěji objevují. Krychle by tak mohla reagovat na náklon telefonu a podle něj by se postavička pohybovala.

To ovšem nejsou zcela jistě všechna vylepšení, která by se dala realizovat. Určitě je zde ještě velký prostor k dalším inovacím.

8 Závěr

Tato práce nejprve stručně seznámila čtenáře s programovacím jazykem a platformou Java. Dále se detailněji věnovala edici Java 2 Micro Edition a stručně popsala proces vývoje dalších balíčků a specifikací. Tento úvod zasadil následují kapitoly do širšího kontextu.

Dále se práce již zabývala zadaným problémem, a to srovnáním rozhraní pro vykreslování 3D grafiky na mobilních zařízeních, konkrétně *Mobile 3D Graphisc API* a *MascotCapsule*. Nejprve poskytla základní informace o obou rozhraních a třídách, které poskytují. V následujících kapitolách byly popsány hlavní rozdíly mezi oběma API, především z implementačního pohledu, ale také z pohledu podporovaných grafických funkcí.

Práce se věnovala také srovnání z hlediska reálné výkonnosti a poskytla výsledky získané měřením výkonu na reálných mobilních telefonech. Za tímto účelem jsem vytvořil testovací aplikaci, která byla v práci taktéž popsána. Nakonec se práce zabývá popisem demonstrační aplikace (hry), jejíž implementace byla také součástí zadání.

Díky této bakalářské práci jsem se seznámil z platformou J2ME a hlavně poměrně důkladně s tvorbou 3D aplikací v tomto prostředí. Získal jsem jak teoretické znalosti, tak praktické zkušenosti s oběma API. Při tvorbě testovací aplikace jsem si vyzkoušel implementaci programů jak v M3G, tak v MascotCapsule. Díky práci na demonstrační aplikaci jsem se pak ještě důkladněji a hlouběji seznámil s rozhraním M3G.

Literatura

[1] Bittnerová, Lucie Rút. *Co vás zajímá o J2ME, ale báli jste se zeptat* [online]. Aktualizován 2002-10-14 [cit. 2009-05-10]. Dostupný z WWW: <http://interval.cz/clanky/co-vas-zajima-o-j2me-ale-bali-jste-se-zeptat/>.

[2] *Java (programming language)* [online]. Aktualizován 2009-05-12 [cit. 2009-05-10]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Java_(programming_language)>.

[3] *Java (software platform)* [online]. Aktualizován 2009-04-30 [cit. 2009-05-10]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Java_(software_platform)>.

[4] Sun Microsytems. *Sun Mobile Device Technology - Introduction to Mobility Java Technology* [online]. [cit. 2009-05-10]. Dostupný z WWW: <http://developers.sun.com/mobility/getstart/>.

[5] TOPLEY, Kim. *J2ME v kostce : pohotová referenční příručka*. 1. vyd. Praha : Grada Publishing, 2004. 536 s. ISBN 80-247-0426-9.

[6] *Mobile Information Device Profile* [online]. Ver. 2.0 [cit. 2009-05-10]. Dostupný z WWW: <http://java.sun.com/javame/reference/apis/jsr118/>.

[7] Sun Microsytems. *JSRs: Java Specification Requests* [online]. [cit. 2009-05-10]. Dostupný z WWW: < http://jcp.org/en/jsr/tech?listBy=1&listByType=platform>.

[8] *Mobile 3D Graphics API Technical Specification* [online]. Ver. 1.1 [cit. 2009-05-10]. Dostupný z WWW: <http://jcp.org/aboutJava/communityprocess/mrel/jsr184/index.html>.

[9] *MascotCapsule Programming for com.mascotcapsule* [online]. Ver. 1.0.2 [cit. 2009-05-10]. Dostupný z WWW: <http://www.mascotcapsule.com/en/download/comdot_doc.php>.

[10] *MascotCapsule V2/V3 User's Manual* [online]. Ver. 1.0.1 [cit. 2009-05-10]. Dostupný z WWW: <http://www.mascotcapsule.com/en/download/comdot_doc.php>.

[11] *API reference of com.mascotcapsule package for Sony Ericsson* [online]. Aktualizován: 2007-04-07 [cit. 2009-05-10]. Dostupný z WWW: <http://www.mascotcapsule.com/en/download/comdot_doc.php>.

[12] *Comparisons, specifications, and reviews of latest cell phones* [online]. [cit. 2009-05-10]. Dostupný z WWW: <http://www.phonegg.com/>.

Seznam příloh

Přiložené DVD obsahuje následující adresáře:

- **bin** – zkompilované balíčky .jar, které lze spustit přímo na mobilním telefonu
- **doc** – dokumentace k aplikacím
- **project** – adresář s projekty pro vývojové prostředí NetBeans
- **src** – zdrojové soubory

Adresáře jsou dále rozděleny do podadresářů podle aplikace:

- **cube3D** – demonstrační aplikace
- **benchmarkM3G** – testovací aplikace ve verzi pro M3G API
- **benchmarkMC** – testovací aplikace ve verzi pro MascotCapsule

DVD také obsahuje soubor *cube3D.jpg* s plakátem a *readme.txt* s popisem obsahu disku a návodem na spuštění aplikací.