TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

BAKALÁŘSKÁ PRÁCE

Liberec 2007 **Luboš Martínek**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektrotechnika a informatika

Studijní obor: 2612R011 – Elektrotechnické informační a řídicí systémy

3D aplikace v J2ME s využitím JSR-184

3D application in J2ME with JSR-184 support

Bakalářská práce

Autor: **Luboš Martínek**

Vedoucí práce: Ing. Roman Špánek

Konzultant: Ing. Pavel Pirkl

V Liberci 10.5.2007

ORIGINÁL ZADÁNÍ

BAKALÁŘSKÉ PRÁCE

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum:

Podpis:

Abstrakt

Tato bakalářská práce se zabývá zobrazováním prostorových dat na mobilních telefonech pomocí aplikačního rozhraní J2ME s využitím Java mobile 3D graphics (JSR-184). Jsou zde vysvětleny procesy zobrazování trojrozměrných dat v počítačových systémech a některé speciální technologie, dále se práce věnuje platformě J2ME (Java2 Micro Edition), následuje nejpodstatnější část, která obsahuje popis metod a tříd JSR-184 nebo-li podpory 3D na mobilních zařízeních.

Součástí bakalářské práce je sada programů, ze kterých je snadné pochopit způsob přístupu ke grafickým problémům trojrozměrné scény. Ukázky obsahují různé typy transformací kamery a modelu krychle, skládání objektů (compositing mode), práci se scénou (scene graph), příklady světel, mlhy, animací, ale nejdůležitější částí je import scény z profesionálního modelovacího softwaru a její ovládání na klávesnici telefonu.

Abstract

This bachelor thesis deals with the imaging of three-dimensional data on mobile phones by the support of application interface J2ME with usage of Java mobile 3D graphics (JSR - 184). Processes of imaging three-dimensional data in computer systems and some special techniques are explained. Further, the thesis describes platform J2ME (Java2 Micro Edition), followed by the most important section containing specification of the methods and classes available in JSR- 184, which bring support for 3D on mobile devices.

An integral part of the bachelor thesis is a program package, from which one can easy understand various access techniques to graphics problems of three-dimensional scenes. Nevertheless the package contains various types transformations of the camera and model cube, composition objects, work with scene, examples of lights, fog, animation, the most important part is ability to import a scene from a professional modeling software and its further control by the use of the keyboard of the mobile phone.

Obsah

Seznam obrázků

Úvod

Software pro mobilní telefony byl dlouhou dobu kolekce statických aplikací implementovaných pro specifický operační systém. S Javou pro mobilní zařízení získali vývojáři postupem času možnost vytvářet aplikace, které už nejsou vyhrazeny pro omezené množství zařízení bez nutnosti rekompilace a modifikace zdrojového kódu. Podobnost celosvětově rozšířené Javy pro desktopy, J2EE a J2SE, poskytuje velký lidský potenciál pro vývoj softwaru v oblasti mobilních telefonů podporujících Javu. Před několika lety se začala také rychle rozvíjet oblast 3D grafiky. Přirozeným krokem v evoluci Javy bylo vytvoření specifikace jak ovládat 3D scény. Nokia sjednotila tým profesionálů, který koncem roku 2003 vypracoval finální verzi specifikace dnes známou jako JSR-184.

Cílem zadané práce je seznámit čtenáře s problematikou zobrazování prostorových dat na mobilních telefonech s podporou JSR-184, vysvětlit proces vytváření trojrozměrného obsahu v několika fázích a nastínit některé speciální techniky využívané pro efekty, metody simulující reálná prostředí.

Práce je dělena do tří základních kapitol. V první kapitole je rozebrána problematika zobrazování trojrozměrných dat v počítačových systémech. Vysvětluje se nejdříve teoretická část lineární algebry spolu s výpočtem osvětlení a mlhy, následuje specifikace konstrukce trojrozměrné scény a další zobrazovací techniky geometrického zpracování a rastrování. Druhá kapitola je věnována průřezu aplikačního rozhraní J2ME. Jsou zde představeny konfigurace, profily a popis specifikací CLDC a MIDP. V poslední kapitole se popisuje design rozhraní JSR-184. Specifikuje se datová struktura 3D scény, třídy, metody a funkce prostředí. Součástí je projekt vytvořený v Netbeans IDE obsahující několik vzorových příkladů v podobě zdrojového kódu.

1 Zobrazování prostorových dat

1.1 Fáze grafického zpracování

I když zatím neznáme podrobnosti o datech ani grafických algoritmech, je vhodné si uvést přehled typického 3D zobrazovacího řetězce (bez ohledu na to, zda je podporován hardwarem). Na následujícím diagramu je hrubé rozdělení zobrazovacího řetězce.

Jednotlivé etapy zpracování mohou být samozřejmě velmi složité a často v sobě obsahují zřetězení mnoha dílčích kroků (jako v případě geometrického zpracování) nebo dokonce paralelní výpočet (jak bývá zvykem u konečné rasterizace tzn. převodu na pixely).

Fáze *aplikace* typicky obsahuje prezentaci 3D dat a jejich jakékoli aplikačně závislé zpracování: pohyby těles, fyzikální simulace virtuálního světa, rozhodování o tom, které části scény jsou důležitější (a které se tedy budou kreslit nejpřesněji – Level of Detail, LoD), interakce mezi dynamickými předměty, umělou inteligenci (zejména důležitá je ve videohrách, ale též v ostatních realistických simulátorech), apod. Implementace je výhradně na straně software.

Fáze *geometrie* je zodpovědná za většinu operací prováděných nad jednotlivými ploškami (obecně polygony, často jen trojúhelníky) a jejich vrcholy: geometrické transformace v 3D, osvětlení, projekce, ořezávání, transformace a ořezávání ve 2D. Tato fáze je často implementována pomocí jednoho dlouhého řetězce (pipeline) a pokud je alespoň některá část přenesena na hardware (hardware T&L), uplatňuje se zde i paralelismus (nezávislé zpracování).

Finální fáze *rasterizace* má za úkol převést geometrická data (primitiva – nejčastěji body, čáry a trojúhelníky) do rastrové reprezentace, vykreslit 2D/3D objekty na rastrový displej. Nejdůležitějšími technikami jsou: určování viditelnosti (Z-buffer), mapování textur, interpolace barev (stínování), zpracování průhledných objektů, mlha, práce s šablonou (stencil), apod.

1.2 Teorie

Nejdůležitější pojmy z matematiky (lineární algebry, geometrie) a optiky, které 3D počítačová grafika používá:

1.2.1 Lineární algebra a geometrie

Při popisu souřadnic 3D objektů se (kromě obyčejných kartézských souřadnic) používají tzv. homogenní souřadnice. Je to čtveřice $[x, y, z, w]$, kde poslední složka *w* je nulová pro vektory (směry) a nenulová pro vlastní body trojrozměrného prostoru. Je-li homogenní složka nenulová, ale různá od 1, můžeme spočítat normalizované souřadnice $\left[\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right]$. V normalizovaném tvaru $[x, y, z, 1]$ jsou první tři složky identické s „obyčejnými" nehomogenními souřadnicemi bodu.

Používání homogenních souřadnic spolu s homogenními transformačními maticemi (rozměrů 4×4) nám umožňuje jednotně prezentovat nejen rotace, protažení (změnu měřítka), zkosení, apod., ale též posunutí. Díky tomu je snadné např. libovolně umístit osu otáčení, střed protažení nebo zkosení.

Transformační matice budeme zásadně chápat jako homogenní, jejich aplikace na souřadnice bodu spočívá v násobení řádkového vektoru souřadnic maticí zprava:

$$[x, y, z, w] \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = [x', y', z', w']$$

Nejběžnější transformační matice mají speciální tvar posledního sloupce – proto se také někdy ukládají jen jako třísloupcové:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix}$$

Zde levá horní submatice a_{11} až a_{33} vyjadřuje rozměr a orientaci, vektor $[t_1, t_2, t_3]$ posunutí (translaci) a jednotkový čtvrtý sloupec naznačuje, že se jedná o afinní

transformaci (zachovávající rovnoběžky). Matice perspektivní projekce má naopak netriviální poslední sloupec.

Několik ukázek elementárních transformačních matic – rotace kolem osy *z* o úhel α, natažení objektu ve směru osy *x* s koeficientem 1,5 a posunutí o vektor $[2,1,-3]$:

$$\begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1{,}5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & -3 & 1 \end{bmatrix}$$

Díky asociativitě maticového násobení lze posloupnost několika navazujících maticových transformací spočítat jako násobení jedinou (složenou, kompozitní) maticí:

$$((([x,y,z,w]\cdot M_1)\cdot M_2)\cdot M_3) = [x,y,z,w]\cdot(M_1\cdot M_2\cdot M_3)$$

Pomocí skládání maticových transformací lze snadno spočítat matice pro otáčení kolem libovolné osy, protažení podle libovolného vektoru, apod.

Významnou roli mezi transformacemi hrají tzv. *transformace tuhého tělesa* (rigid-body transform), složené pouze z otáčení a posunutí. Pomocí jedné translace a tří otočení lze například sestrojit matici, která mezi sebou převádí dvě souřadné soustavy – ztotožňuje dva uživatelské systémy definované kartézskými osami (x,y,z) a (v,u,l). Při označení druhého systému bylo použito notace souřadného systému spojeného s polohou pozorovatele, kde *v* je směr pohledu (view), *u* vektor mířící vzhůru (up – temeno hlavy pozorovatele) a *l* vektor levé upažené ruky (left hand).

Dále se budu zabývat *projekčními transformacemi*. Pro jednoduchost se předpokládá, že uživatel je umístěn v počátku souřadnic a dívá se ve směru osy *z*. Rovnoběžné promítání (odpovídající pohledu z nekonečna) se implementuje pouhým „zapomenutím" třetí souřadné složky (složky *z*). *Perspektivní projekce* musí navíc zohlednit vzdálenost pozorovaného objektu – čím je předmět dál, tím se zobrazí jako menší. Jednoduchý vzorec pro perspektivní projekci by mohl vypadat: $\left[\frac{x}{z},\frac{y}{z}\right]$ nebo se použije následující homogenní transformační matice (*d* je vzdálenost projekční roviny od pozorovatele, přičemž pozorovatel je v počátku souřadnic a dívá se ve směru kladné poloosy *z*).

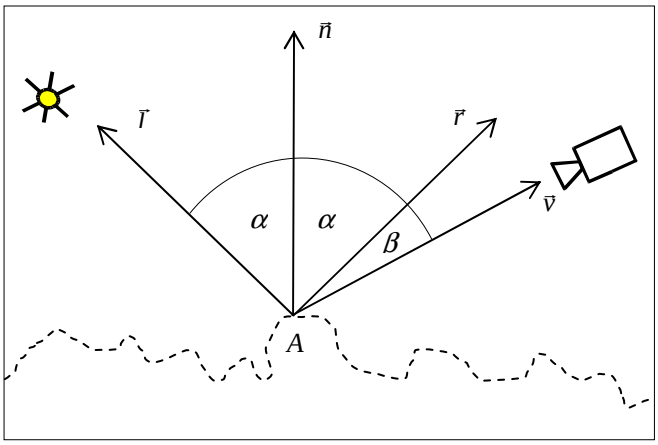
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/d \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

V praxi se používají komplikovanější matice, které transformují celý objem zorného úhlu (view frustum) do kvádru nebo krychle, nejčastěji umístěných kolem počátku souřadnic. Například v systému Direct3D se jedná o kvádr $[-1,-1,0]$ až $[1,1,1]$.

Kompletní projekční transformace se tedy skládá z převodu souřadných soustav (souřadná soustava spojená s pozorovatelem se převede do světového systému souřadnic) následovaného některou z matic perspektivní projekce.

1.2.2 Osvětlení

Aby zobrazení trojrozměrných těles bylo více plastické, používají se v počítačové grafice různé osvětlovací modely. V nejjednodušší formě to jsou vztahy popisující odraz světla na povrchu tělesa podle orientace plochy vzhledem ke světelnému zdroji v čemž spočívá barevný odstín a pozorovatel tak získá lepší představu o prostoru. Do „osvětlení" se obyčejně nezahrnuje výpočet stínů (které části předmětu jsou skutečně zdrojem osvětleny a na které vrhne stín nějaká překážka).



Obrázek 1: odraz světla

Na obrázku je znázorněna situace, kdy na povrch tělesa do bodu A svítí ze směru $\vec{l}$ světelný zdroj. Normálový vektor je označen $\vec{n}$, směr k pozorovateli $\vec{v}$ a vektor

dokonalého (zrcadlového) odrazu $\vec{r}$. Vektory $\vec{l}$, $\vec{n}$ a $\vec{r}$ svírají úhel α, vektory $\vec{r}$ a $\vec{v}$ úhel β.

Jedním z nejjednodušších světelných modelů je Phongův model. Světlo odrážející-se do směru $\vec{v}$ se skládá ze tří složek: okolní nebo-li zbytkové světlo (ambient light) L_a, difusní odraz (diffuse) L_d a lesklý odraz (specular) L_s. Zjednodušené vzorce pro výpočet těchto tří složek:

$$L_a = C \cdot k_a$$

$$L_d = (C \cdot C_l) \cdot k_d \cdot \cos\alpha$$

$$L_s = C_l \cdot k_s \cdot \cos^h \alpha$$

Kde C je vlastní barva povrchu tělesa, C_l barva a intenzita zdroje, konstanty k_a, k_d, k_s slouží k nastavení vlastností materiálu (lesklý, matný, apod.) a exponent h ovlivňuje ostrost zrcadlového odlesku (součin dvou barevných vektorů se počítá po složkách). Zjednodušení spočívají v tom, že barva zrcadlového odlesku je totožná s barvou světla a dále v tom, že k_s považujeme za konstantu. Přestože není tento elementární model fyzikálně věrný, pro použití v rychlém zobrazování většinou postačí.

Kosinus úhlu, který svírají dva jednotkové vektory, se počítá velmi jednoduše jako jejich skalární součin. To znamená, že ve 3D stačí tři násobení a dvě sčítání, některé architektury (MMX SSE, 3DNow! nebo assemblery GPU) obsahují pro tyto operace speciální instrukce.

Pokud se ve scéně vyskytuje více zdrojů světla, jejich L_d a L_s se sčítají (L_a se započítá pouze jedenkrát). Je-li zdroj světla relativně blízko ve scéně (není v nekonečnu), měl by se započítat útlum jeho intenzity podle převrácené hodnoty čtverce jeho vzdálenosti od bodu *A*. V praxi se však tento vztah nahrazuje slabším vzorcem, kde se vzdálenost objevuje v menší mocnině.

1.2.3 Mlha

Při průchodu světla okolním prostředím (atmosférou) dochází ve skutečnosti k mnoha jevům, z nich se v počítačové grafice nejčastěji počítá pohlcení/rozptyl paprsku v mlze. Barva mlhy C_f (obvykle bílá) se mísí s vypočtenou barvou povrchu tělesa C_s podle vzorce:

$$C = f \cdot C_s + (1 - f) \cdot C_f$$

Kde f je koeficient vypočtený podle některého z následujících vzorců (první je jednodušší lineární mlha, druhý vzorec popisuje fyzikálně věrnější exponenciální mlhu):

$$f = \frac{z_{end} - z}{z_{end} - z_{start}} \qquad f = e^{-D \cdot z}$$

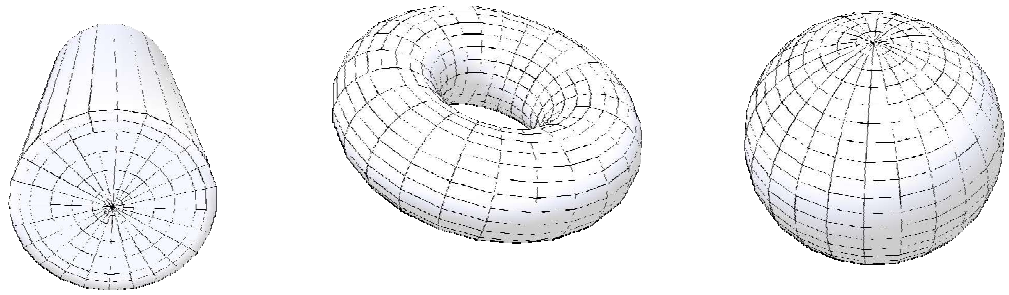
kde z_{end} a z_{start} udávají vzdálenosti začátku a konce mlhy od pozorovatele, z je vzdálenost vykreslovaného bodu od pozorovatele a D definuje hustotu mlhy.

1.3 Reprezentace 3D scény

V hardwarově podporované 3D grafice se používá v podstatě jenom jedna reprezentace scény – všechny objekty jsou definovány pomocí svého povrchu (proto se této reprezentaci někdy říká povrchová, *B-rep* jako *boundary representation*). Jelikož je potřeba povrch těles definovat co nejúsporněji, používáme k tomu síť mnohoúhelníků (polygonů) – všechna naše tělesa jsou tedy mnohostěny. I když se při modelování někdy používají i složitější plošky, do grafického vykreslovacího systému se většinou už posílají jenom trojúhelníky.

Je třeba si uvědomit, že mnohostěn nemůže dokonale nahradit hladká tělesa a proto i u tak jednoduchého tělesa, jako je třeba koule, narážíme při aproximaci mnohostěnem na mnohé problémy. Obecně platí – čím přesnější (věrnější) aproximaci chceme dostat, tím větší počet trojúhelníků musíme použít a tím pomalejší bude vykreslování modelu.

V paměti počítače se síť trojúhelníků pamatuje snadno, stačí mít uložené 3D souřadnice všech vrcholů sítě a každý trojúhelník si pak jen uloží, které tři vrcholy mu patří (vrcholy se mezi jednotlivými trojúhelníky sdílejí a ušetří se tak paměť i výkon systému). Vrcholy tělesa mohou nést ještě další informace jako například barvu, normálový vektor (pro stínování) a souřadnici textury. Běžné dnešní modely obsahují stovky (menší tělesa) až milióny vrcholů (ohromné objekty, celá scéna). Na následujícím obrázku jsou vidět tělesa aproximované sítí čtyřúhelníků.



Obrázek 2: sítě modelů

1.3.1 Level of Detail

Moderní grafické akcelerátory umějí zobrazit kolem 10^7 až 10^8 trojúhelníků za vteřinu. To ale v případě, kdy naše scéna v plné přesnosti má 10^7 nebo více trojúhelníků, nestačí pro plynulý pohyb. Nastupují techniky souhrnně nazývané *Level of Detail* (LoD). Využívají toho, že pozorovatel si může dobře prohlížet pouze objekty, ke kterým má blízko. Vzdálené předměty se mohou kreslit zjednodušeně nebo se dokonce zcela eliminují. Systém LoD má za úkol automaticky připravovat různé stupně přesnosti modelu podle toho, jak blízko je pozorovatel k danému objektu. Primitivní algoritmy umějí pouze přepínat mezi předem připravenými modely (pokud není systém dobře vyladěn, můžeme pozorovat rušivé přeskakování), dokonalejší řešení obsahuje metody, jak adaptivně přepočítávat složitost trojúhelníkové sítě podle vzdálenosti k pozorovateli. Pokud se pozorovatel posune o malý kousek, bývá i změna sítě malá. Kvalitní algoritmy pro LoD jsou poměrně komplikované a v rámci tohoto dokumentu se jimi nemůžeme podrobněji zabývat.

1.4 Geometrické zpracování

Je-li povrchový model 3D scény připraven k zobrazení, musí nejprve projít geometrickým zpracováním. Tam se pracuje s celými trojúhelníky, resp. jejich vrcholy. Souřadnice vrcholů se transformují ze světových souřadnic (ve kterých je má uložena aplikace) nebo z lokálních relativních souřadnic (Hierarchické transformace) do takové souřadné soustavy, ve které se s nimi bude pohodlně pracovat. To bývá obvykle soustava spojená s pozorovatelem, kdy směr pohledu leží na ose *z*. Souřadnice *x* a *y* se ještě transformují (škálují) tak, aby kresba vyplnila požadovaný výřez na obrazovce a pak se již dvojice $[x, y]$ dají použít přímo k vykreslení a zbylá složka *z* poslouží při výpočtu

viditelnosti jako vzdálenost od pozorovatele (resp. vzdálenost od přední ořezávací roviny).

1.4.1 Stínování

Podle zadaného zdroje světla se pro všechny plošky spočítají vektory $\vec{r}$, $\vec{l}$ a $\vec{v}$ (jsou-li zdroj i pozorovatel ve značné vzdálenosti, může se variabilita vektorů $\vec{l}$ a $\vec{v}$ zanedbat). V praxi se používá jedna ze tří metod interpolace:

- *Konstantní stínování* (Flat shading) je nejjednodušší a nejrychlejší, bohužel však dává pro aproximace hladkých těles nejhorší výsledky. Osvětlení se spočítá jednou pro každý trojúhelník a celá plocha se vybarví daným odstínem
- *Gouraudova interpolace barvy* (Gouraud shading) odstraňuje nepříjemně ostré přechody mezi jednotlivými ploškami a hodí se proto na objekty, které jsou aproximacemi hladkých těles. Osvětlení se počítá v každém vrcholu mnohostěnu a uvnitř trojúhelníka se odstín dopočítává lineární interpolací. Takovou interpolaci zvládnou běžné grafické akcelerátory. Při použití ostrých odlesků (velmi lesklé materiály) nedává ani interpolace barvy uspokojivé výsledky.
- *Phongova interpolace normály* (Phong shading) je nejdokonalejší, ale také nejpomalejší metoda. Do každého kresleného pixelu se interpoluje normálový vektor (z daných normál ve vrcholech mnohostěnu). V každém pixelu se potom počítá osvětlení podle požadovaného modelu. Protože postup předpokládá schopnost hardware vykonávat v jednotlivých pixelech netriviální operace, je Phongova interpolace dostupná jen u nejmodernějších GPU (a jen pro omezenou množinu světelných zdrojů).

1.4.2 Projekce a ořezávání

Poslední fáze geometrického zpracování začíná výpočtem skutečné polohy objektů (vrcholů) v rovině *průmětny*. Pro rovnoběžnou projekci je to triviální operace, kdy perspektivu vyřeší některá z transformačních matic uvedených dříve (převod komolého zorného jehlanu do viditelného kvádru/krychle – anglicky view frustum).

Aby se následující fáze zpracování (rasterizace) nezatěžovaly zbytečně daty, která nemohou podstatně přispět k výslednému obrázku (a aby se při promítání nemuselo počítat se zvláštními případy), aplikuje se na všechna grafická data algoritmus ořezávání (clipping). To znamená, že jsou ze zpracování okamžitě odstraněny všechny objekty ležící celé mimo zorný jehlan (frustum) a objekty jen částečně zasahující do zorného jehlanu jsou oříznuty. Protože se jedná o body, úsečky nebo trojúhelníky, není výpočet příliš složitý.

1.4.3 Hierarchické transformace

Prezentace virtuálního světa obsahuje nezřídka hierarchické principy – scéna se skládá z objektů (často jsou objekty instancemi vytvořenými podle daných vzorů), samotné objekty se skládají z komponent, ty zase z menších dílů, atd. Z mnoha praktických důvodů je vhodné, aby se v takovém modelu světa používaly relativní transformační matice. Žádný objekt nemá transformační matici ze své souřadné soustavy (local coordinate system) do světového systému souřadnic (world coordinate system), ale pouze relativní matici transformace mezi svou lokální soustavou a soustavou spojenou s bezprostředním nadřízeným systémem.

1.5 Výpočet viditelnosti

Součástí poslední fáze grafického zpracování jsou algoritmy určující přesný vzhled výsledku. Jednou z důležitých komponent je metoda výpočtu viditelnosti. V grafickém hardware se už dlouhá léta používá výhradně jeden algoritmus: Z-buffer. Idea je jednoduchá – zobrazované grafické objekty (čáry, trojúhelníky, apod.) se rozdělí na jednotlivé body (pixely) a pro každý pixel se jeho viditelnost určí samostatně. Protože se jedná o algoritmus typu „hrubá síla”, je výpočet podpořen dvojrozměrným polem velikosti obrazovky. Pro každý pixel obrazovky se zaznamená vzdálenost toho bodu 3D scény, který je v tomto pixelu zobrazen (nejbližší ze všech). Dosud nepokreslené části obrazovky mají v poli uloženou nějakou velkou hodnotu (nekonečno) a touto hodnotou se pole na začátku musí inicializovat.

1.6 Textury

Vzhled nakreslených 3D objektů by bez textur byl velmi chudý, povrch objektů by byl velmi hladký, nerealistický. Techniky mapování textur mají největší zásluhu na

tom, jak realisticky dnes počítačem generované obrázky vypadají (i když ještě dlouho zřejmě nebudou nerozeznatelné od přirozených snímků).

Definovat pojem textury není úplně jednoduché: musíme se spokojit s obecným vyjádřením, že textura je technika modifikující pixel po pixelu vzhled kresleného povrchu. Nejčastěji se textura aplikuje jako barevná tapeta, kterou lepíme na povrch tělesa. Ale mohou být i textury ovlivňující optické vlastnosti materiálu (odrazivost), zavádějící hrbolatost (bump-texture) nebo nahrazující některý z obtížných algoritmických postupů (např. Phongovo stínování nebo zrcadlový odraz na hladkém povrchu tělesa).

1.7 Průhlednost

Mnoho (zejména geometrických) efektů se implementuje velmi snadno a elegantně, máme-li k dispozici poloprůhledné trojúhelníky. Dokonce je možné mapovat poloprůhledné textury – v některých místech jsou pak objekty (plochy) průhledné zatím co jinde ne.

Průhlednost se v počítačové grafice obyčejně definuje pomocí parametru *alpha*. Je to desetinné číslo mezi 0 a 1 udávající míru neprůhlednosti objektu – 0 znamená úplně průhledný, 1 zcela neprůhledný. Často se přidává jako čtvrtá složka k barevné trojici RGB, pak mluvíme o poloprůhledné barvě RGBA.

1.8 Šablony

Inspirováni Z-bufferem vymysleli vývojáři grafických čipů další silný nástroj: buffer šablon (stencil buffer). Jako se do Z-bufferu ukládá informace o hloubce nakresleného 3D bodu, stencil buffer slouží k uložení libovolné informace (atributu) daného pixelu. Typicky je programátorovi k dispozici (pro každý pixel) několik bitů v bufferu šablony. Grafický zobrazovací řetězec je navíc upraven tak, aby se do šablony dalo zapisovat (dokonce pomocí libovolné booleovské operace) a naopak – před vlastním vykreslením (nebo Z-testem) je GPU připraven šablonu otestovat a podle výsledku kreslení provést či nikoli.

Tak dokážeme elegantně vykreslovat pouze do některých částí obrazovky (jako náhradu za poloprůhlednost tam, kde by nebyla dostatečně efektivní), selektivně vyřadit Z-testování, počítat vržené stíny, atd. Nejvíce triků umožňují šablony spolu s víceprůchodovými zobrazovacími algoritmy.

2 J2ME

2.1 Historie

Internet tvoří ohromné množství počítačů s různými mikroprocesory a různými operačními systémy. To, co kdysi bylo sice znervózňujícím, ale ne příliš významným problémem, přerostlo v kritický aspekt při vývoji síťových aplikací.

V roce 1993 začínalo být autorům programovacího jazyka Java zřejmé, že problém přenositelnosti programů pro různá elektronická zařízení je velmi podobný problému přenositelnosti aplikací napsaných pro síť Internet. Proto se zaměřili na takové rozšíření jazyka, aby jej bylo možné použít i pro Internet.

2.2 Přehled technologie

J2ME je verze produktu Java společnosti Sun Microsystems, určená pro trh s elektronickými zařízeními, jakými jsou např. mobilní telefony, pagery, osobní digitální asistenti (PDA), přídavná zařízení a další malá zařízení. Od doby jeho vzniku se k vývoji J2ME připojilo více než 600 společností, např. Palm, Nokia, Motorola a RIM. Směr, jímž se J2ME vyvíjí, však nemá být uchováván v tajnosti za zavřenými dveřmi velkých korporací. Vývoj J2ME je zajištěn projektem zvaným Java Community Process (JCP).

J2ME poskytuje kompletní řešení nejmodernějších síťových aplikací pro malá zařízení. Výrobcům, poskytovatelům a vývojářům těchto zařízení také slibuje možnost vyvíjet nově aplikace pro své zákazníky. Tato možnost však neznamená, že by byly obětovány základní vlastnosti Javy, jejichž význam v současnosti stále roste, jmenovitě kompatibilita různých prostředí a zabezpečení.

Verze J2ME definuje skupinu knihoven a API, které lze spustit na každém virtuálním stroji; říká se jim konfigurace a profily.

2.3 Konfigurace

Mobilní telefony, pagery, organizery a další malá zařízení se liší svojí formou, funkcemi a vlastnostmi. Často však používají podobné procesory a mají podobné množství paměti. Proto autoři J2ME vytvořili konfigurace. Konfiguracemi se definuje horizontální členění produktů založené na dostupném množství paměti a výkonu procesoru

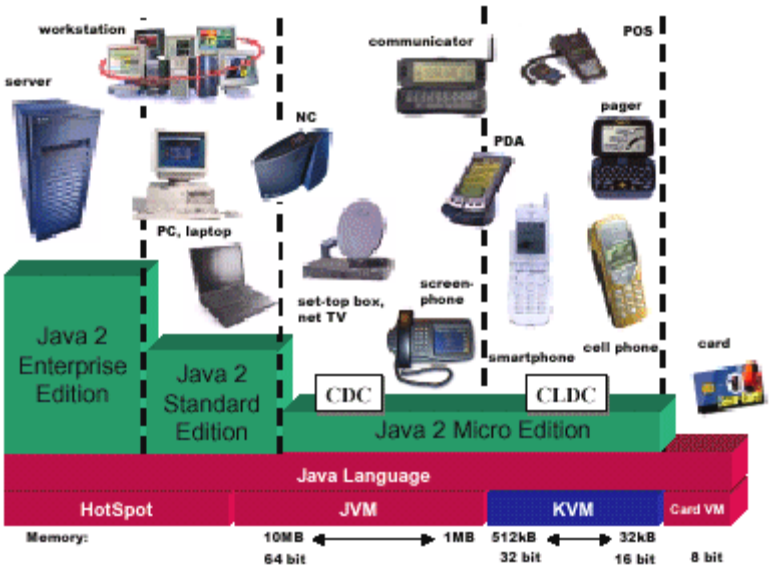
jednotlivých zařízení. Jakmile má tyto informace k dispozici, konfigurace zjistí následující údaje:

- podporované rysy programovacího jazyka Java
- podporované rysy virtuálního stroje Javy
- podporované základní java knihovny a API

V současnosti existují pro J2ME dvě standardní konfigurace: CLDC (Connected Limited Device Configuration) a CDC (Connected Device Configuration).

Konfigurace *CDC* je určena pro výkonná zařízení, která jsou občas připojena k síti. Patří sem přídavná zařízení, televize po Internetu, domácí spotřebiče a navigační systémy pro vozidla. CDC obsahuje plnou verzi virtuálního stroje Javy podobného tomu, který se dnes používá u J2SE. Rozdíl spočívá v paměti příslušného zařízení a v zobrazovací schopnosti.

Druhý typ konfigurace, *CLDC*, je pro J2ME obvyklejší. Tato konfigurace udává mnohem menší požadavky na zabudovaná zařízení než CDC. Konfigurace CLDC byla poprvé vydána v říjnu 1999 se záměrem vytvořit „nejnižšího společného jmenovatele“ v prostředí Java pro zabudovaná zařízení, dvousměrné pagery, osobní digitální asistenty (PDA) a osobní organizery.



Obrázek 3: rozdělení konfigurací

2.4 Profily

J2ME umožňuje definovat java prostředí pro různé produkty stejné úrovně tím, že zavádí profily. Profil je vlastně sada programových rozhraní (API) tvořících nadstavbu konfigurace. Profil nabízí programu přístup k vlastnostem specifickým pro dané zařízení. Následuje několik příkladů profilů, které jsou v současnosti k dispozici pro J2ME:

MIDP je navržen pro práci s CLDC a poskytuje sadu programových rozhraní použitelných v mobilních zařízeních, jako jsou mobilní telefony a obousměrné pagery. MIDP obsahuje třídy pro uživatelská rozhraní, trvalé ukládání a práci v síti. Dále obsahuje standardizované pracovní prostřední, které umožňuje „nahrávat“ do koncového zařízení nové aplikace. Malým aplikacím, které běží pod MIDP, se říká Midlety.

Profil PDAP je založen na CLDC a poskytuje API pro uživatelské rozhraní (mají být podskupinou AWT) a API pro ukládání dat v příručních zařízeních. V době vzniku této práce se na profilu PDAP ještě pracovalo a nebyly dostupné žádné popisy jeho implementace.

Základní profil rozšiřuje programové rozhraní, které poskytuje CDC, ale nedodává žádné API pro uživatelské rozhraní. Jak naznačuje název „základní“ (Foundation), tento profil má sloužit jako základní pro další profily, například pro Osobní profil a RMI profil.

Osobní profil rozšiřuje možnosti Základního profilu o grafické rozhraní (GUI), na němž lze spustit applety pro Java Web. Jelikož se PersonalJava mění v Osobní profil, bude zpětně kompatibilní s aplikacemi pro PersonalJava 1.1 a1.2.

RMI profil se používá ke vzdálenému volání metod (Remote Method Invocation - RMI). Je to mechanismus jazyka Java, umožňující volání metod vzdálených objektů v distribuovaných aplikacích. RMI profil rozšiřuje možnosti Základního profilu o RMI pro daná zařízení. Protože navazuje na Základní profil, je RMI profil určen pro CDC/Základní profil, a ne pro CLDC/MIDP. RMI profil je kompatibilní s J2SE RMI API 1.2x a vyššími.

2.5 CLDC

Podle specifikace mají zařízení, na nichž se uplatňuje CLDC, následující vlastnosti:

- *160 kB až 512 kB celkové paměti* – Zařízení s CLDC by mělo mít minimálně 128 kB paměti ROM pro virtuální stroj Javy a pro CLDC knihovny a minimálně 32 kB paměti RAM využitelné pro práci virtuálního stroje, to vše nezávisle na jiných aplikacích.
- *16bitový nebo 32bitový procesor* s minimální taktovací frekvencí 25 MHz – Tyto typy procesorů jsou v současnosti pro přenosná zařízení typická.
- *Připojitelnost k některému druhu sítě* – V případě CLDC se často jedná o dvousměrné dvoudrátové připojení s omezenou šířkou pásma.
- *Nízká spotřeba energie* – Zařízení s CLDC často pracují na energii z baterií. Proto mají velmi nízkou spotřebu energie.

Tomuto popisu vyhovují zařízení všech tvarů a velikostí. Okamžitě se nám vybaví mobilní telefony a pagery, ale stejně dobře bychom mohli Javu instalovat i na čtečky čárového kódu, video a audio vybavení, navigační systémy a další bezdrátová zařízení, která se ještě objeví. V podstatě lze očekávat, že společně se změnami na trhu těchto zařízení se budou měnit i specifikace pro CLDC.

S danými, výše uvedenými omezeními poskytuje v současnosti CLDC zařízením tyto funkce:

- soubor rysů Javy a virtuálního stroje
- soubor základních knihoven Javy (java.lang a java.util)
- základní vstup/výstup (java.io)
- základní podporu sítí (javax.microedition.io)
- zabezpečení

CLDC nezasahuje do průběhu aplikací, do uživatelských rozhraní, ani do správy událostí, či do interakce mezi uživatelem a aplikací. Tyto rysy totiž spadají do sféry profilů, jako je MIDP, které jsou uváděny na vrcholu CLDC a přispívají k jeho funkčnosti.

2.6 MIDP

Standard MIDP definuje mobilní informační zařízení jako zařízení s následujícími minimálními vlastnostmi:

- *Displej* – Velikost obrazovky minimálně 96 x 54 bodů s minimální hloubkou barev 1 bit.
- *Vstupní zařízení* – Klávesnice pro jednu nebo dvě ruce, případně dotyková obrazovka.
- *Paměť* – 32 kB paměti RAM pro práci Javy; 128 kB paměti ROM pro komponenty MIDP a 8 kB stálé paměti pro dlouhodobé ukládání dat z aplikací.
- Práce v síti – Přerušované obousměrné spojení, obvykle bezdrátové s omezenou šířkou pásma.

Protože je MIDP vrcholem CLDC, zaměřuje se na následující oblasti, které jsou v CLDC neřešené:

- *Správa průběhu aplikací* – MIDP obsahuje balík javax.microedition.midlet s třídami a metodami pro zahájení, přerušení a vymazání aplikací z hostitelského prostředí.
- *Uživatelské rozhraní a události* – MIDP také poskytuje balíky javax.microedition.lcdui, které zahrnují třídy a rozhraní k tvorbě komponent grafického uživatelského rozhraní v aplikacích.
- *Připojitelnost k síti* – MIDP rozšiřuje rozhraní ContentConnection obecného připojovacího systému (Generic Connection Framework) tím, že poskytuje rozhraní HttpConnection a současně instalační sadu pro http protokol.
- *Ukládání dat v zařízení* – MIDP také poskytuje balík javax.microedition.rms, který zavádí databázový záznamový systém. Ten umožňuje aplikacím ukládat data v zařízení.

2.7 IDE pro Javu

Nástroje IDE (Integrated Development Enviroment) jsou GUI aplikace, sdružující v sobě několik programů. Jde o editor kódu, kompilátor, debugger, systém nápověd a další vývojářské nástroje. Nástroje IDE svou koncepcí „vše v jednom“ usnadňují vývojářskou práci. Dále je třeba uvést některé z klíčových vlastností prostřední IDE:

- sdružování do projektů – nastavíme cesty ke knihovnám, souborům nápovědy, verzi kompilátoru a další vlastnosti projektu
- automatické zálohování
- možnost vyvolání nápovědy přímo k vybranému objektu
- automatické formátování, vizuální zvýraznění syntaxe
- zobrazení chyb při kompilaci – kliknutím na chybu se přeneseme přímo do chybné pasáže kódu
- kódový asistent – nabízí například třídy z balíků nebo ukazuje typy parametrů metod, počet parametrů, atd.

Některé nejznámější IDE nástroje pro jazyk Java: JBuilder (Borland – www.codegear.com/products/jbuilder), NetBeans (Sun – www.netbeans.org), Eclipse (IBM – www.eclipse.org), WebLogic Workshop (BEA – dev2dev.bea.com/workshop), JDeveloper (Oracle – www.oracle.com/technology/products/jdev), Visual J++ (Microsoft – msdn2.microsoft.com/en-us/vjsharp), Intelli J (JetBrains – www.jetbrains.com/idea), CodeGuide (OmniCore – www.omnicore.com/en).

3 JSR-184

3.1 Základní prvky

K API můžeme přistupovat ve dvou odlišných režimech: *immediate* a *retained*. Režim immediate lze chápat podobně jako nízkoúrovňovou funkci OpenGL a podobných 3D API. Dovoluje programátorovi definovat téměř každý detail renderovacího procesu. Režim retained kompletně skrývá většinu nízkoúrovňových funkcí, ale dovoluje programátorovi nahrávat a zobrazovat animovaná 3D data v několika řádkách kódu.

Tyto dva režimy mají hlavní vliv na implementaci vnitřní renderovací posloupnosti (rendering pipeline), jejich rozdíly v programátorském přístupu nejsou dramaticky odlišné. Je možné nahrávat, zobrazovat a měnit 3D grafická data velmi efektivně v obou režimech. Formát 3D dat podporovaný API je *.m3g*. Je velmi kompaktní a současně také přizpůsobitelný a umožňuje ukládat data téměř libovolné složitosti do jediného souboru.

3.1.1 Scéna

Stromová struktura, ve které každý list definuje druh fyzického nebo abstraktního prvku – uzlu (node) v trojrozměrném prostoru (kamera, světlo, modely), je obvykle nazývána scéna (scene graph). Některé scény mohou obsahovat další vlastnosti jako materiál nebo aplikačně závislá metadata.

V JSR-184 může scéna obsahovat jakoukoliv třídu rozšířenou z *javax.microedition.m3g.Object3D*. „Držitelem místa" pro scénu je objekt *javax.microedition.m3g.World*. Je možné nahrát celou scénu z jediného *.m3g* souboru. Uložený svět lze přímo renderovat na obrazovku (retained mode) nebo když obsahuje něco jednoduššího, jako třeba jediný model, lze k němu přistupovat bezprostředně (immediate mode).

Pro udržení nízké paměťové náročnosti scény je výhodné sdílet data mezi objekty.

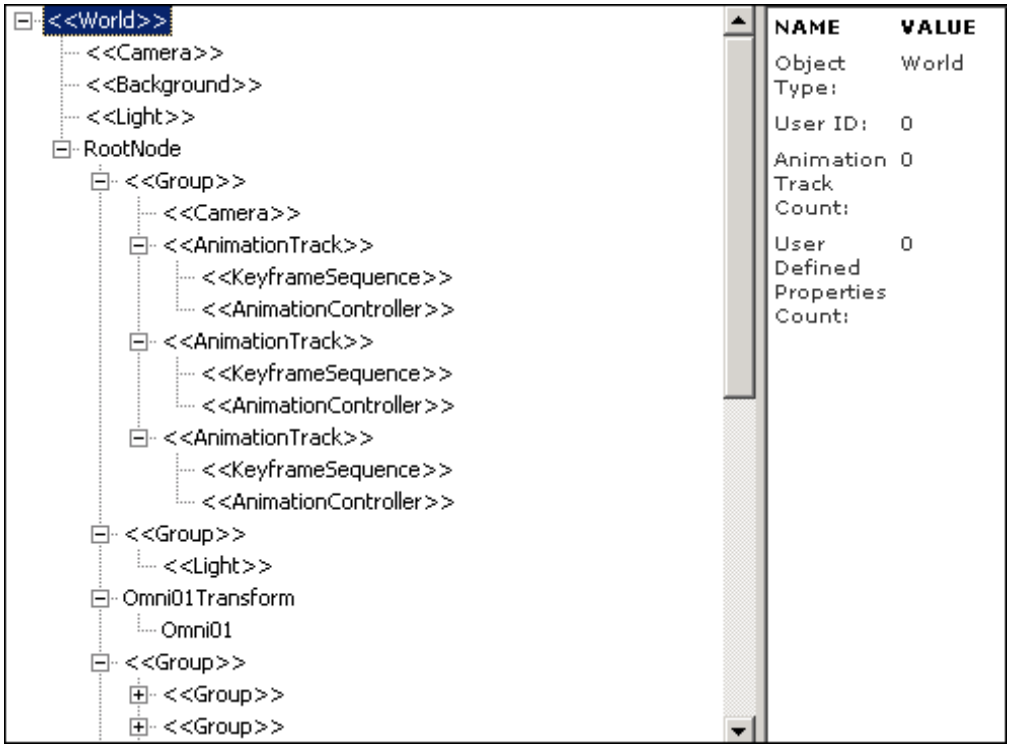
3.1.2 World

Třída World (*javax.microedition.m3g.World*) poskytuje snadnou cestu jak udržovat všechny informace o 3D scéně (kompletní scene graph) v jednom uspořádaném balíčku. Jedna instance třídy World může teoreticky obsahovat libovolné množství modelů, kamer, světel apod., každý z nich může mít několik parametrů pro animaci.

World lze chápat jako nejvyšší část scény. Na rozdíl od obvyklého prvku není třída World schopna mít předka a její transformace jsou ignorovány během renderovacího procesu. Například programátoři týmu Swerve automaticky přiřazují oddělenou kořenovou skupinu (RootNode) k vrcholu scény přímo pod World a díky tomu lze celou scénu transformovat ačkoli to třída World nedovoluje.

Stejně jako samotný World objekt, tak i .m3g formát souboru, je schopen obsahovat cokoliv od jednoduchého 3D modelu až po komplexní animovanou 3D scénu s několika kamerami, světly, apod. World může mít vlastní statické pozadí definované ve třídě *javax.microedition.m3g.background*.

Spojování prvků a dalších dat mezi několika World objekty je možné, ale tento proces je náročný na paměť.



Obrázek 4: struktura scény (scene graph)

Při procesu nahrávání objekt Loader rozbalí obsah .m3g souboru a vytvoří všechny důležité Java objekty během doby spouštění. Vytvoří se ovladač animace (animation controller), nastaví se rodičovská třída a skupiny, inicializují se světla a kamery atd. a to vše jedním řádkem v programu.

I relativně jednoduchá scéna s několika objekty se může změnit v komplexní mnoha prvkovou. Protože každý objekt (nezáleží na velikosti ani vzhledu) obsazuje určité množství paměti, která je v mobilních zařízeních značně omezená, je velmi důležité odstranění všech nepotřebných dat už ve fázi modelování autorským nástrojem (např. 3ds max).

3.1.3 Loader

Třída Loader (*javax.microedition.m3g.Loader*), implementována v základu platformy, je navržena pro streaming *.m3g* obsahu. Programátoři nejsou nuceni vyvíjet vlastní formáty ani psát komplexní a náročný kód Loaderu (je to možné, ale potřebné pouze ve speciálních případech). Když je *.m3g* soubor zavedený v paměti, všechny třídy rozšířené z Object3D, jsou automaticky převedeny Loaderem na pole objektů. Následuje ukázka kódu:

```
Object3D[] o = null;
try
{
o = Loader.load(name);
}
catch (Exception e)
{
}
World loadedWorld = (World) o[0];
```

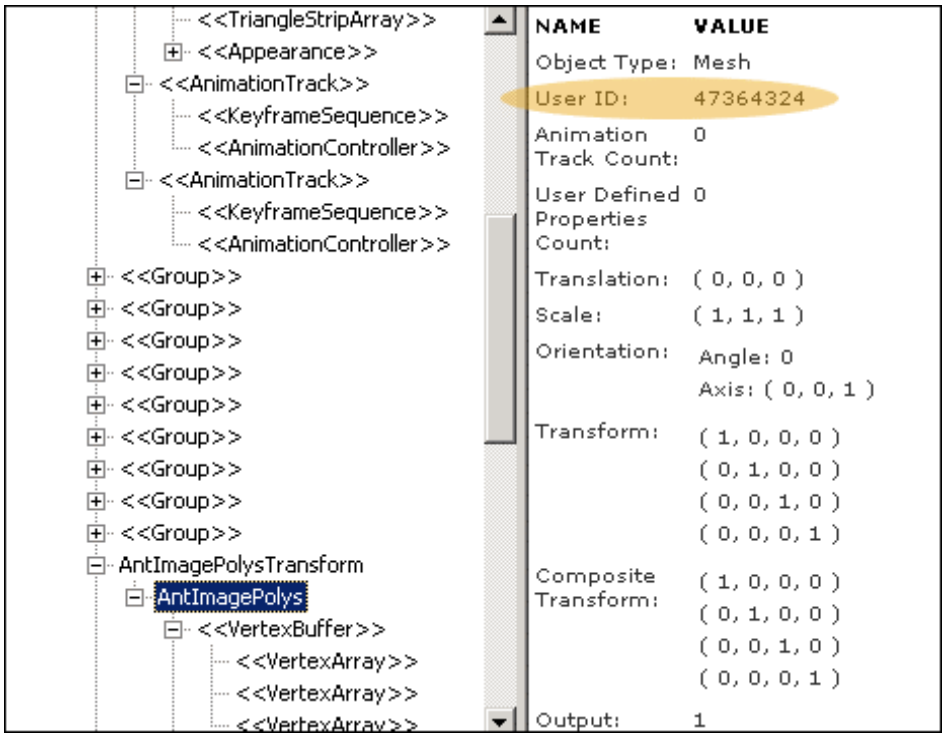
Nahrávání dat rovnou z *.m3g* souboru je nejvýhodnější cestou jak přenést 3D obsah do aplikace. Stojí to velké množství paměti, speciálně pokud je každý objekt animovaný. Obsah lze také animovat manuálně v programovém kódu. Například náhodné sněhové vločky je rozumnější animovat manuálně pomocí kódu, protože animovaná data z autorského nástroje by obsadila značnou část paměti.

Obrázek 5: simulace náhodných vloček (forum.nokia.com)

3.1.4 Object3D

Skoro všechny třídy v knihovně *.m3g* (kromě Loader, Transform, RayIntersection a Graphics3D) jsou odvozené z *javax.microedition.m3g.Object3D*. Třída Object3D poskytuje základní sadu funkcí společnou všem třídám, jako je serializace/deserializace objektů, ovládání animace, uživatelsky definovaný identifikátor, hodnoty parametrů a duplikační metody.

Použitím těchto metod může být jakákoliv třída odvozená od *Object3D* jednoduše animována s přiřazeným ovladačem animace (animation controller). Mohou být streamovány z *.m3g* souborů, klonovány a je možné jim přiřadit uživatelské parametry a identifikátory (lze to také manuálně použitím autorského nástroje). Tato informace může být využita k identifikaci objektu ve scéně. Na následujícím obrázku jsou vidět vlastnosti a identifikátor objektu třídy mesh spolu s částí stormové struktury scény.



Obrázek 6: uživatelský identifikátor

3.1.5 AnimationTrack, AnimationController, KeyframeSequence

Objekty a jejich animovatelné parametry (intenzita světla, transformace kamery, viditelnost, apod.) mohou být přednastaveny sekvencí snímků, kterou lze animovat použitím metody *animate()* zděděné ze třídy Object3D.

Třída *javax.microedition.m3g.KeyframeSequence* definuje sadu klíčových hodnot, jejich příslušné časy a interpolační metodu k určení aktuální hodnoty ve vybraném čase. Každý *KeyframeSequence* může být nastaven buď na určitý časový úsek nebo na opakování ve smyčce. Je dokonce možné použít více klíčových snímků pro jednu pozici v čase, což umožní kamerové střihy nebo další nesouvislosti v animační posloupnosti.

Aktuální hodnota klíčového snímku sekvence ve zvoleném čase může být získána interpolační metodou (linear nebo Catmull-Rom spline interpolace) nebo krok za krokem, kde je každý snímek konstantní dokud nedosáhne animace další klíčové hodnoty.

Sekvence klíčových snímků může být cíleně spuštěna pro zobrazení vybrané scény jako parametr *javax.microedition.m3g.AnimationTrack.*

Rychlost animace, klíčové pozice a případné míchání několika animačních úseků jsou ovládány ve třídě *javax.microedition.m3g.AnimationController*, která může být přiřazena několika úsekům animace.

3.1.6 Graphics3D

Třída *javax.microedition.m3g.Graphics3D* obsluhuje renderovaní trojrozměrné scény podobně jako ve 2D grafice třída *Graphics*.

Celý objekt World nebo jen část(i) scény mohou být renderovány voláním Graphics3D s nastavenou kamerou. Renderovací kamera může být buď vybrána manuálně, nebo lze použít výchozí.

Před renderováním musí být kontext *Graphic3D* vázaný na *MIDP Graphics* objekt, který vykresluje zbytek zobrazovacího pole. Po renderování je třeba *Graphic3D* uvolnit.

```
// bind to graphics
g3d.bindTarget(g);

// get currently displayed World
World scn = m_scenes[m_currentScene];
scn.animate(m_currentTime);

// render the scene
g3d.render(scn);

// release g3d
g3d.releaseTarget();
```

API nabízí čtyři odlišné renderovací režimy. První mód se používá když je renderována celá scéna najednou (retained mode). Aktivní renderovací kamera a světla jsou v tomto režimu nastaveny třídou World. Ve zbylých třech režimech (immediate modes) jsou aktivní renderovací kamera a skupina aktivních světel definovány ve třídě *Graphics3D*. Druhý režim je užívaný pro renderování vybrané části scény (scene graph node) včetně vybraných částí skupin. Třetí a čtvrtý jsou určené pro renderování jednotlivých částí modelů. Tyto režimy mohou být velmi užitečné při tvorbě rozmanitých grafických triků, jako je například renderování na textuře. Aplikace, které uplatňují jakýkoliv druh algoritmů na optimalizaci viditelnosti (portal a exit rendering) nebo techniku level of detail, mají profit z renderování modelů těmito metodami.

Též je možné určit kvalitu renderování v *Graphics3D*. Nastavení mění poměr kvality a rychlosti, jako například celoobrazovkový antialiasing (využívaný na vyhlaze-

ní hran polygonů), dithering (využívaný u displejů s nízkou barevnou hloubkou pro kvalitnější barvy) a true color renderování (používané u zařízení s vyšší barvenou hloubkou pro věrnější obraz). Předešlá nastavení obrazu nejsou nutná pro správnou funkci API, jedná se pouze o doplňková nastavení.

Velikost displeje, maximální velikost textury, počet světel a maximální počet textur lze zjistit u každého zařízení s podporou JSR-184 metodou *getProperties()*. Limity zařízení by se měly vždy dodržovat, protože nastavení mimo rozsah podporovaných hodnot může způsobit chybování renderingu aplikace nebo dokonce její ukončení.

3.1.7 Background

Třída Background (javax.microedition.m3g.Background) definuje jak bude uvolněn renderovací buffer. Background také podporuje nastavitelné ořezávání (clipping) trojúhelníků.

3.2 Transformace a typy uzlů

3.2.1 Transformace

Abstraktní třída *javax.microedition.m3g.Transformable* definuje posun, změnu měřítka, rotaci a volnou transformační matici pro uzel (*javax.microedition.m3g.Node*) nebo texturu (*javax.microedition.m3g.Texture2D*). Transformační matici bodu p $[x, y, z, w]$ reprezentují buď souřadnice vrcholu nebo textury definované relativně k rodičovskému souřadnému systému uzlu $p' = T \cdot R \cdot S \cdot M \cdot p$, kde T – určuje matici translace, R – rotaci, S – měřítko a M – všeobecnou homogenní matici velikosti 4×4.

Třída Transformable obsahuje metody pro nastavení všech transformací manuálně (*setTranslation(), setScale(), atd.*).

3.2.2 Node

Node (uzel – *javax.microedition.m3g.Node*), odvozená z abstraktní třídy *javax.microedition.m3g.Transformable*, reprezentuje všechny možné typy objektů scény jako jsou světla, kamery, modely, Sprite3D a jejich skupiny.

Uzel definuje místní systém souřadnic, který se transformuje vzhledem k rodičovskému systému souřadnic. Uzel nemůže být sdílen mezi World ani Group z důvodu omezení třídy *javax.microedition.m3g.Group*. Uzel může být přiřazen

k referenčnímu uzlu, v tomto případě je matice R – rotace zaměněna se zarovnávací maticí A následujícím způsobem: $p' = T \cdot A \cdot S \cdot M \cdot p$. Dva referenční uzly se dají nastavit pokud se použije nejdříve metoda *setAlignment()* pak *align()* nebo se přiřadí jako parametr metody *align()*. Například kamery nebo světla mohou být přiřazené k vybranému uzlu, ale také mohou být použity pro objekty typu billboard, které se automaticky otáčí čelní stranou ke kameře.

Další užitečná vlastnost pro uzel je Scope. Scope (rozsah) může být použit pro různé účely, ale nejčastěji se používá pro třídění viditelnosti objektů. Představme si velký trojrozměrný svět rozdělený na několik částí, ve kterém žádná z částí není viditelná z ostatních. Nastavíme pro každou část světa různou masku Scope a změníme kameru tak aby korespondovala s aktuální pozicí pozorovatele. Pokud jsou kamera a Scope různé, nejsou uzly renderované což ve výsledku ušetří drahocenný čas CPU.

Použitím Scope lze zvýšit rychlost propočtů světel. Obvykle mají světla v herním světě předem určený dosah (range of influence - ROI) nastavený typem světla a intenzitou. Přiřazením odlišných Scope pro světla a modely vzhledem k jejich vzdálenosti se může definovat, zda světlo ovlivní model nebo ne. Díky tomu je možné si dovolit více světel, která znatelně neovlivní čas výpočtu scény.

3.2.3 Group

Svazek uzlů může být seskupený pro snadnější manipulaci použitím *javax.microedition.m3g.Group*. Seskupování je užitečné v situacích, kde je zapotřebí ovládat několik objektů najednou. Častý příklad použití *Group*: automobil s kabinou a čtyřmi koly. Určíme-li vůz jako *Group*, je snadné pohybovat celým autem bez nutnosti pohybu kol a kabiny odděleně.

Skupiny mohou být použity i tam, kde chceme nastavit viditelnost velkého počtu objektů najednou. Využívá se toho nejčastěji při viditelnosti speciálních efektů jako jsou exploze, nebo je lze efektivně použít v kombinaci se systémem třídění viditelnosti pro skrývání uzlů potomků (pokud je rodičovská skupina neviditelná).

3.2.4 Camera

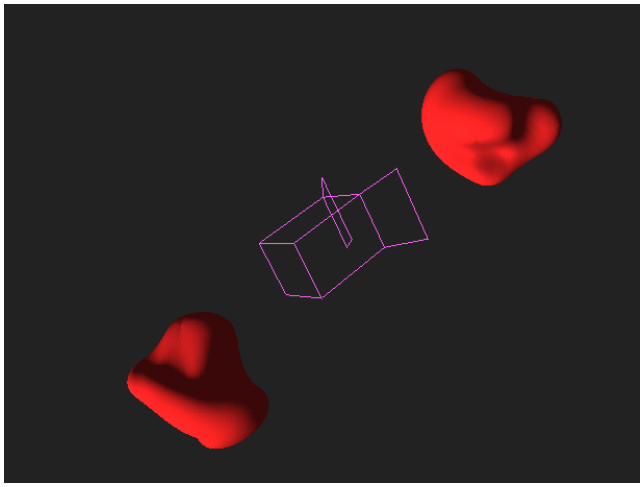
Camera (*javax.microedition.m3g.Camera*) je třída, která transformuje souřadnice z 3D prostoru na souřadnice obrazovky. Kamera používá OpenGL ořezávání a projekci s výjimkou uživatelsky definovaných řezů.

Je možné nastavit více kamer v režimech immediate a retained. Teoreticky je povoleno jakékoliv množství kamer, což znamená, že lze vytvořit skvělou trajektorii a rozmanité úhly pohledu stejně dobře jako s využitím speciálních efektů pomocí rotace a parametrů jedné kamery. Kamera nastavená pro renderování může být vybrána odděleně, lze ji zaměřit na objekty (propojením na zarovnávající uzel) a změnit parametr úhlu pohledu.

3.2.5 Light

Specifikace JSR-184 podporuje čtyři různé typy světel, každý s rozdílným výpočetním algoritmem. Výpočty splňují standardy OpenGL. Typy světel jsou:

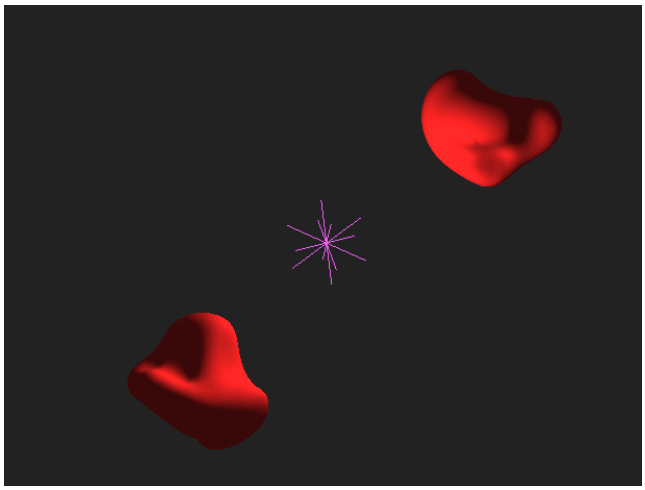
- *Ambient* – určuje obecnou intenzitu světla na objekty scény. Světlo ambient osvětluje celou scénu stejnou intenzitou, pozice a směry jsou ignorovány během výpočtu, což snižuje zátěž CPU.
- *Directional* (směrové) – definuje směr, z něhož se světlo šíří prostorem. Pozice a vzdálenost světla od objektů nemá žádný vliv, takže může být volně umístěno ve scéně. Směrové světlo je dobré pro simulaci vzdálených světelných zdrojů jako třeba denní světlo a mělo by zatížit CPU o trochu méně než Omni a Spot typ světla. Na obrázku je umístěno světlo mezi dva objekty a přesto paprsek osvětluje obě tělesa stále z levého dolního rohu. Ambient světlo s intenzitou 25 % je také přidáno do scény.



Obrázek 7: directional light (forum.nokia.com)

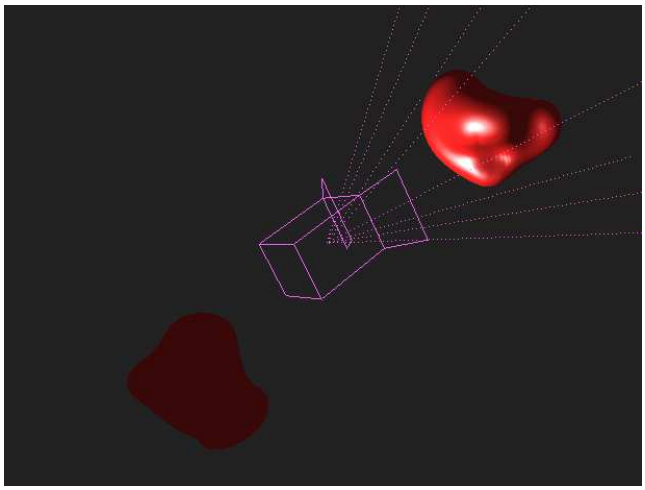
- *Omni* – určuje bod zdroje světla. Ovlivňuje objekty ve všech směrech okolo zdroje. Může být nastavena útlumová křivka, kdy světlo ztrácí svoji intenzitu

se vzdáleností. Jsou k dispozici dva různé typy útlumových charakteristik lineární a kvadratická.



Obrázek 8: omni light (forum.nokia.com)

- *Spot* – definuje směr, pozici a úhel dopadu paprsků. Kužel směřující ve smyslu záporné osy *z* značí osvětlenou oblast. Spot neovlivňuje objekty, které přesahují kuželem vytyčenou oblast, jak je vidět z další ukázky. Výpočetní náročnost je mnohem vyšší než u předcházejících typů světel. Pro spot světlo je možné definovat parametr, který ovlivní ostrost osvětlené oblasti.



Obrázek 9: spot light (forum.nokia.com)

Výpočty světel mohou ve velkých scénách vyžadovat značnou část procesorového času a proto je žádoucí vybírat korektní typy světel, používat Scope nebo se dokonce úplně vyhnout použití světel a místo nich zvolit předrenderované světlo v textuře. Ačkoli každé světlo má RGB barvu a hodnotu intenzity, je přesný styl jak světlo působí na renderování povrchu modelu je definován ve vlastnosti *Material*. Kombinací proměnli-

vých parametrů světel a materiálů lze dosáhnout emulace široké škály různých typů povrchů. Také je možné vytvářet zajímavé efekty deklarováním záporných intenzit světel, což způsobuje ztemňování povrchů.

3.2.6 Mesh

Mesh je v podstatě svazek souřadnic vrcholů s popisem jejich propojení tak, aby se vytvořil 3D model s povrchem odpovídající struktuře objektu. Součástí je i popis druhu materiálu použitý pro vytvoření povrchu tělesa. *Javax.microedition.m3g.Mesh* je třída zapouzdřující *vertex buffer*, *index buffer* a vhodné třídy definující vzhled modelu, které určují tvar a povrch objektu.

V nejjednodušší formě musí Mesh obsahovat nejméně tři vrcholy spojené do trojúhelníku než se může Mesh renderovat. V praxi to znamená, že pro třídu Mesh je nutné definovat alespoň jedno *VertexArray* (pole vrcholů) a nejméně jeden platný *IndexBuffer* (propojení vrcholů).

Polygony mohou být teoreticky libovolného tvaru a mohou mít každý libovolné množství vrcholů, v praxi je ale dostupná pouze jediná implementace třídy *IndexBuffer* (*javax.microedition.m3g.TriangleStripArray*), která vyžaduje trojúhelníkový tvar polygonu. Není to problém, protože většina současného 3D modelovacího softwaru exportuje modely jako svazek trojúhelníků.

Základní implementace Mesh je rigid body (pevné tělo), což znamená že vrcholy nemohou být animovány (kromě přímého přepisování dat sítě vrcholů před renderováním snímku).

Vícenásobné podsítě se dají definovat uvnitř Mesh, každá s vlastním vzhledem. To znamená, že jeden Mesh může obsahovat několik různých typů povrchů. Submeshe (podsítě) sdílejí stejný *VertexBuffer*.

Jsou-li použity pouze neprůhledné polygony, nemá pořadí renderování z-buffered trojúhelníků žádný efekt na výsledek. Neprůhledné povrchy by měly být vždy vykreslovány před transparentními, aby nedocházelo ke špatnému řazení při vykreslování scény.

Z tohoto důvodu je většina pravidel pro pořadí renderování určena ve třídě *Appearance*. Číslo pořadí renderovací vrstvy lze nastavit metodou *setLayer()* v *Appearance*. Modely s nižším číslem vrstvy jsou vždy renderovány dříve než objekty s vyšším číslem vrstvy. To dovoluje manuální nastavení pořadí renderování modelů scény. Nutno

doplnit, že všechny neprůhledné polygony jsou vždy ve stejné vrstvě renderovány před transparentními.

3.2.7 VertexBuffer, VertexArray, IndexBuffer, TriangleStripArray

Javax.microedition.m3g.VertexBuffer, javax.*microedition.m3g.VertexArray* a *javax.microedition.m3g.TriangleStripArray* tvoří základ trojrozměrného modelu (Mesh).

Javax.microedition.m3g.VertexBuffer je třída, která uchovává informace o vrcholech v podobě datové struktury pole. V implementaci jsou dostupné čtyři typy této datové struktury:

- *Coordinate position array* – pole pozic souřadnic vrcholů (definující prostor modelu). Toto pole musí být vždy součástí implementace Mesh, protože jinak by nebylo co zobrazovat.
- *Vertex normals array* – pole normál uchovává informace o normálových vektorech vrcholů pro pozdější výpočet odrazu světla. Toto pole musí být součástí modelu pokud se využije osvětlení (v jiných případech není vyžadováno).
- *Vertex color array* – pole barev slouží pro tónování modelu pomocí barev vrcholů.
- *Texture coordinate array* – určuje souřadnice textury pro každou aktivní texturovací mapu.

Využití těchto tříd je nutné v případě, kdy se objekty vytváří manuálně pomocí kódu.

3.3 Mesh - vlastnosti povrchu

Tyto třídy definují vzhled povrchu trojrozměrných objektů v kombinaci s osvětlováním.

3.3.1 Appearance

Třída *Appereance* nastavuje atributy renderování pro Mesh nebo Sprite, zapouzdřuje *Material, PolygonMode*, *CompositingMode*, *Fog* a texturové mapy. Dohromady rozhodují o finálním vzhledu povrchu tělesa na displeji. Třída Appereance také definuje renderovací vrstvu, která znovu rozhoduje o pořadí renderování povrchů. Standardně

nemá nový Mesh deklarované povrchové vlastnosti a nebudou na něj aplikovány textury, mlha, světla, viditelnost.

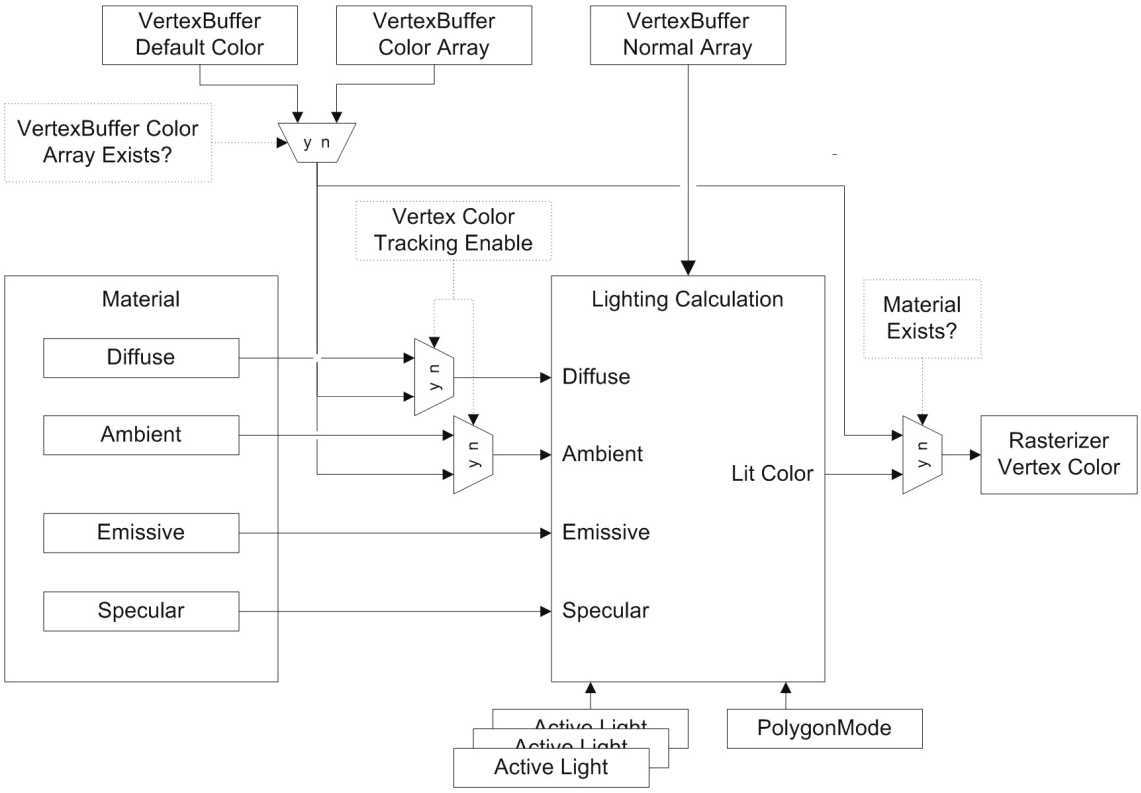
Pořadí renderování se vnitřně třídí podle implementace tak, aby vrstvy s nižším číslem byly vykresleny dříve, než vrstvy s vyšším číslem. Toto se aplikuje na renderování World, Group, (sub)Mesh. Neprůhledné vrstvy jsou vždy umístěny v renderovacím řetězci před vrstvy transparentní.

Záporná čísla vrstev se dají použít na speciální efekty (například na odrazy, oblohu, auru, apod.), ale v běžných případech se negativních hodnot nevyužívá.

3.3.2 Material

Javax.microedition.m3g.Material vymezuje, jak bude povrch reagovat na světlo. Proces obarvování vrcholů tělesa začíná kontrolou, zda existuje materiál povrchu. Pokud není materiál určen, použije se pro proces barvení pole vertex color, nebo (jestliže pole neexistuje) defaultní barva. Když je materiál určen, světla ovlivní výsledný vzhled povrchu, ale je zapotřebí aby definice Meshe obsahovala pole vertex normals.

- *Diffuse* nastavuje základní barvu povrchu. Tento parametr není dostupný pro směrová světla a poziční.
- *Ambient* definuje „neosvícenou“ barvu materiálu. Hodnota je násobena konstantou ambient color použitého světla. Parametr se ignoruje pro směrová a poziční světla.
- Parametr *Emissive* určuje základní barvu materiálu, která je vidět i tehdy, když má světlo nulovou intenzitu.
- *Specular* vymezuje „osvícenou“ barvu jasných povrchů. Tento parametr se ignoruje pro ambient a směrová světla (hodnota je nastavena na nulu).



Obrázek 10: renderovací řetězec povrchu modelu

3.3.3 PolygonMode

PolygonMode charakterizuje shading mode, culling mode, winding mode, oboustrannost (včetně osvětlování oboustranných polygonů), perspektivní korekce textur.

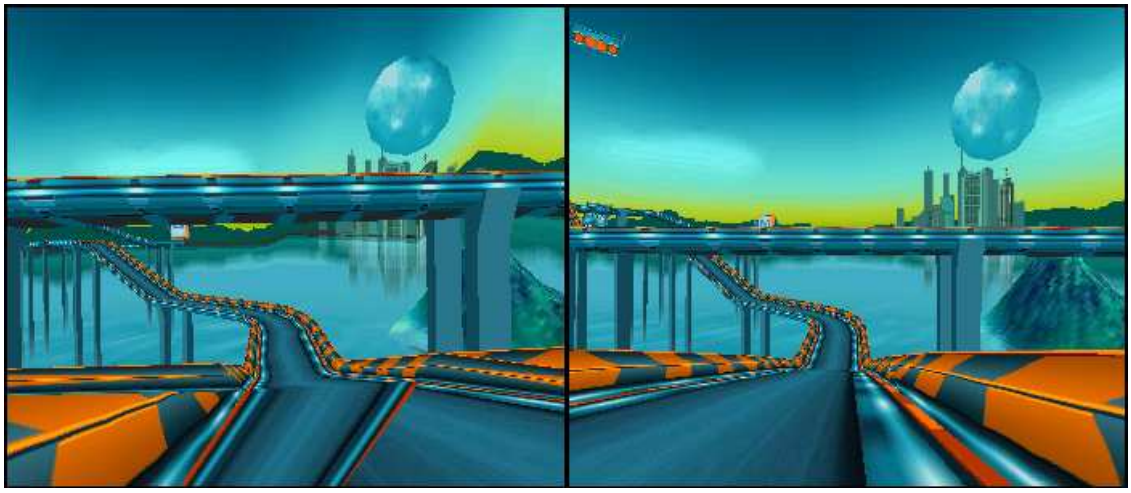
Typ stínování (*shading* mode) může být nastaven buď na *flat* nebo *smooth* (Gouraudovo stínování). V režimu *flat* je celá barva polygonu stanovena jedinou normálou. Při *smooth* stínování jsou barvy vrcholů polygonu počítány zvlášť a výsledná barva každého pixelu se získá interpolací.

Culling mode nastavuje zda jsou polygony tříděny jako viditelné z „přední“, „zadní“ strany nebo vůbec. *Winding* mode stanovuje, která strana je přední a která zadní. Pokud jsou vrcholy polygonu určeny ve směru hodinových ručiček a winding mode je nastaven na clockwise (ve směru hodin), je přední stranou polygonu ta, která se vykresluje na obrazovku ve směru hodin. Winding mode může být nastaven i na counterclockwise (proti směru hodinových ručiček) a v tomto případě je pořadí vykreslování obrácené.

Perspektivní korekce vymezuje, kdy mají texturované polygony použít algoritmus perspektivní korekce pro zamezení nežádoucího efektu protahování u velkých po-

lygonů, speciálně u těch, které jsou blízko oblasti kamerového řezu scény. Zapnutá korekce zpomaluje renderovací algoritmus o 10-20 % v závislosti na implementaci.

Obrázek vlevo má perspektivní korekci vypnutou. Díky značné perspektivě a malému počtu polygonů se textura na některých polygonech blízko cesty a oblohy viditelně deformuje. Perspektivní korekce se téměř neprojevuje na objektech malých a vzdálených od kamery. Je-li použita technika Level of detail, může se korekce často kompletně vypnout pro malé a vzdálené modely.



Obrázek 11: rozdíl mezi zapnutou a vypnutou korekcí perspektivy (forum.nokia.com)

3.3.4 Fog

Třída *javax.microedition.m3g.Fog* implementuje efekt mlhy s nastavitelnou vzdáleností. Když se modely vzdalují od kamery, dochází k tónování barvy polygonů k předem nastavené RGB barvě mlhy. Lze nastavit interval vzdálenosti. Parametr *near* určuje kde bude mlha začínat a *far* kde bude intenzita 100%.

Jsou k dispozici dvě různé metody výpočtu vzdálenosti mlhy: *linear* (lineární) a *exponential* (exponenciální). Lineární metoda je rychlejší, ale exponenciální realističtější. Exponenciální metoda dovoluje nastavit faktor hustoty.

3.3.5 CompositingMode

Polygony se dají překrývat několika způsoby, a to:

- *Alpha* – vykresluje se vážený průměr mezi zdrojovým pixelem a podkladovým pixelem. Míchání je počítáno pro každý pixel interpolací z barev vrcholů.

- *Alpha_add* – počítá se vážený průměr mezi zdrojovým pixelem a podkladovým pixelem a výsledek se přičte k barvě podkladu.
- *Modulate* – intenzity zdrojového a podkladového pixelu se mezi sebou násobí.
- *Modulate_x2* – výpočet je stejný jako u Modulate, navíc je výsledná intenzita násobena 2×.
- *Replace* – výchozí metoda (také nejrychlejší). Podkladový pixel je přímo nahrazený zdrojovým pixelem.

Obrázek 12: různé režimy překrývání (forum.nokia.com)

Polygon se považuje za transparentní, je-li nastaven jiný způsob překrývání než Replace. Průhlednost ovlivňuje pořadí vykreslování i renderovací rychlost.

3.3.6 Image2D

Image2D je třída reprezentující 2-rozměrný obraz. Může se vytvořit v Java MIDP, AWT image nebo jako raw image data.

Dostupné formáty:

- *Alpha* – specifikována je pouze informace o průhlednosti. Pokud jsou data ve tvaru bajtového pole, musí každý pixel mít pouze jeden bajt.

- *Luminance* – určena je pouze informace o jasu. Tento typ se dá použít pro simulaci světla nebo tam, kde je potřeba měnit jas. Vyžadován je jeden bajt na pixel.
- *Luminance_alpha* – obsahuje data o jasu i průhlednosti. Jsou vyžadovány dva bajty na pixel.
- *RGB* – specifikované jsou základní složky barev. Zapotřebí jsou tři bajty na pixel.
- *RGBA* – stejně jako RGB, navíc je zde informace o průhlednosti. Jsou požadovány čtyři bajty na pixel.

3.3.7 Texture2D

Třída *Texture2D* (javax.microedition.m3g.texture2D) kombinuje dvojrozměrnou texturu s informací, jak se má aplikovat na podsíť modelu. Tato informace zahrnuje obrazová data, texel filtering (filtrování pixelů textury), souřadnicovou transformaci textury a composition mode (skládání).

Implementace podporuje pouze rozměry textury dělitelné dvěma (výška a šířka nemusí být stejné). Maximální velikost textury závisí na platformě a lze ji zjistit metodou *getProperties()* v Graphics3D.

Pozice texelu (pixelu textury) na displeji je počítána aplikací transformace (stejně jako u vrcholů polygonu) na souřadnice vrcholů textury, které musí být definovány ve VertexBuffer objektu Mesh. Matice transformace textury obsahuje stejné komponenty jako transformace vrcholu (posun, rotace, měřítko, uživatelská matice).

Po transformaci je interpolací spočítána pozice všech pixelů na celém polygonu. Výsledek je zobrazen na displej buď s nebo bez perspektivní korekce, což je nastaveno třídou PolygonMode.

Když je dána přesná hodnota projekce transformace posunu nebo měřítka mezi dvěma texely (pixely textury), je nutné aproximovat barvu výsledného bodu. Dostupné jsou dvě odlišné metody filtrování. Jenom první typ musí být podporován implementací. Texel filtering:

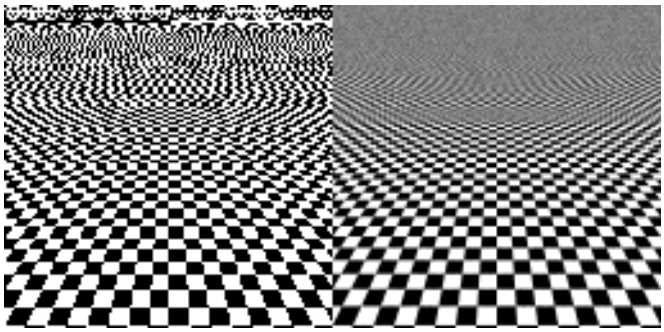
- *Nearest neighbor* – přesná pozice je zaokrouhlena na nejbližší texel, nepřesnost je zde zcela ignorována. Je to nejrychlejší (defaultní) filtrovací režim,

ale může docházet k případům kdy jsou texely větší než pixely a výsledkem je hranatá struktura.

- *Linear filtering* – pozice se počítá váženým průměrem mezi čtyřmi texely. Polygon vpravo na obrázku využívá lineární filtrování (někdy nazýváno bilinear filtering).

Obrázek 13: rozdíl mezi filtrovacími režimy (forum.nokia.com)

Když jeden pixel překrývá více texelů je třeba použít zmenšenou verzi textury aby se snížil její aliasing způsobený nedostatečnou aproximací. Mipmapping je metoda, která přefiltruje textury tak, aby nedocházelo k předešlé chybě, ale za cenu větších paměťových nároků. Obrázek vlevo nemá použitý mipmapping. Ukázka vpravo využívá mipmapping a je zde vidět mnohem lepší aproximace blízko horizontu.



Obrázek 14: rozdíl mezi zapnutým a vypnutým mipmappingem (forum.nokia.com)

Mipmap a texel filtering se dají nastavit nezávisle metodou *setFiltering()*. Parametr *levelFilter* volí typ mipmapping filtrování následujícím způsobem:

- *Base level* – kompletně vypíná mipmapping filtrování, což zvyšuje rychlost renderování a snižuje paměťové nároky.

- *Nearest neighbor* – používá jeden mipmap pro každý pixel.
- *Linear filtering* – interpoluje se mezi dvěmi mipmap úrovněmi vzhledem ke vzdálenosti pixelu. Tento typ v kombinaci s linear texel filtering se také nazývá termínem trilinear filtering.

Pokud je nastavený režim nearest neighbor nebo linear filtering, jsou mipmapy generovány automaticky, většinou opakovaným zmenšováním měřítka textury na poloviční velikost pomocí 2×2 box filtru. Každá nová mipmapa zvyšuje paměťové nároky na textury ¼ původní velikosti originálu.

3.3.8 Texture Blending and Multitexturing

Filtrovaný pixel textury může být kombinován s barvou podkladové vrstvy několika odlišnými způsoby. První vrstva v řadě je vždy interpolovaná barva vertexu. Na ní může být přidáno několik dalších vrstev. Finální výsledek skládání textur závisí na typu textury a režimu míchání (blending mode).

3.4 Speciální techniky a efekty

3.4.1 Sprites

Javax.microedition.m3g.Sprite3D formuluje speciální typ uzlu, který lze použít na velmi rychlé renderování 2D obrazů. Sprite jsou v podstatě 2D obrazy, které mohou využít jakoukoliv část Texture2D.

Sprites jsou užitečné pro urychlení renderování, když není vyžadována 3D geometrie (třeba u vzdálených objektů). Jsou výhodné také pro speciální efekty jako odrazy, oblaka, kouř nebo vodní bubliny. Mají konstantní hodnotu z-buffer a pouze vlastnosti Fog a CompositingMode pro Appearance. Část efektu exploze na obrázku používá transparentní 2D sprites.

Obrázek 15: efekt exploze pomocí 2D sprite (forum.nokia.com)

Na rozdíl od texturových map mohou mít libovolný rozměr a pomocí ořezávacího obdélníku lze použít jen část zdrojového obrazu. Maximální velikost závisí na implementaci.

3.4.2 Morphing

Třída *javax.microedition.m3g.MorphingMesh* je podobná třídě Mesh, navíc definuje několik pozic vrcholů pro jednu skupinu polygonů. Výsledná pozice vrcholu se vypočítá interpolací mezi původní a aktuální polohou vertexů bez nutnosti velkého počtu pomocných vrcholů.

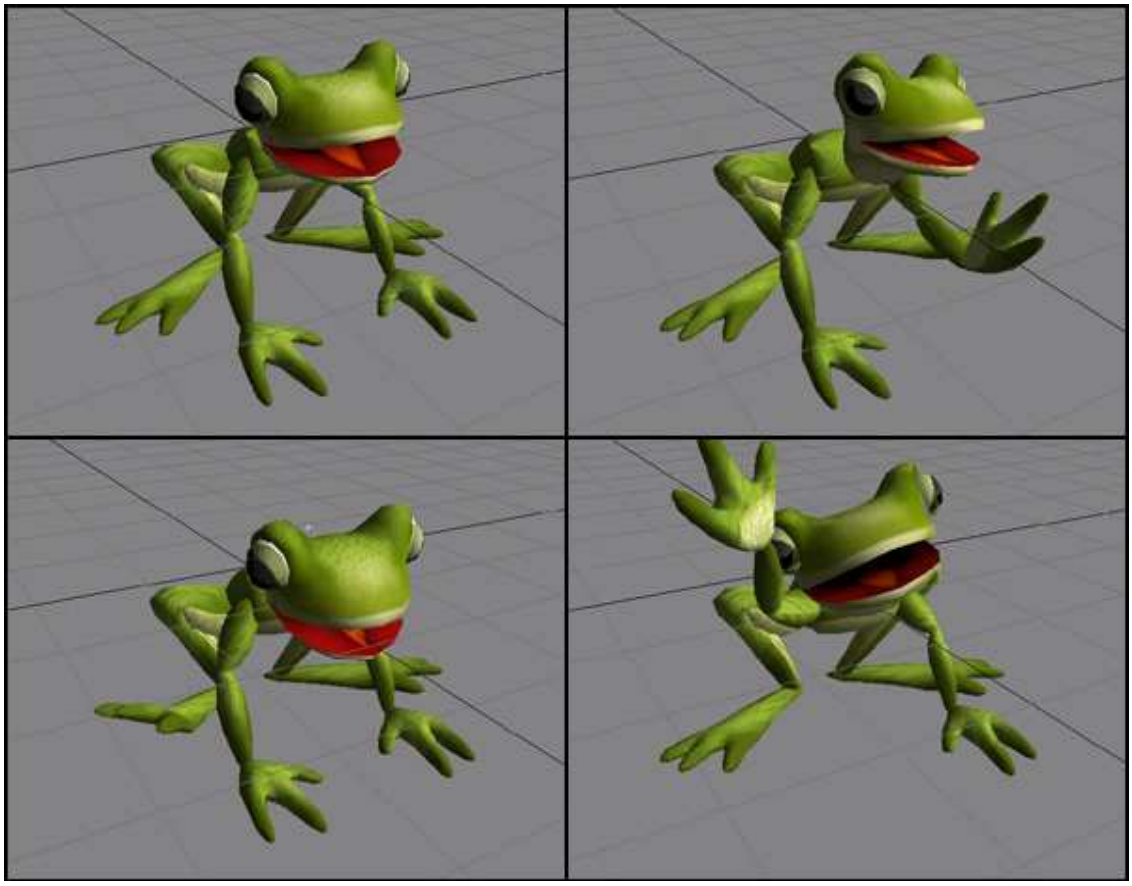
Obrázek 16: morphingem vytvořené výrazy obličeje (forum.nokia.com)

3.4.3 Skinning

Skinning je termín často používaný v současných tutoriálech programování her pro počítače i konzole. Poskytuje způsob jak aproximovat chování kůže a oblečení při pohybu kostí. Simuluje jemné ohýbání v kloubech modelu (lokty a kolena). To bylo

problémem starších implementací bez podpory skinningu, kde vznikaly ostré, hranaté a nereálné úhly.

Na obrázku jsou vidět končetiny žáby ohýbané bez viditelných ostrých hran v několika krocích. Nejlépe je to vidět v levém dolním snímku na kotníku pravé dolní končetiny.

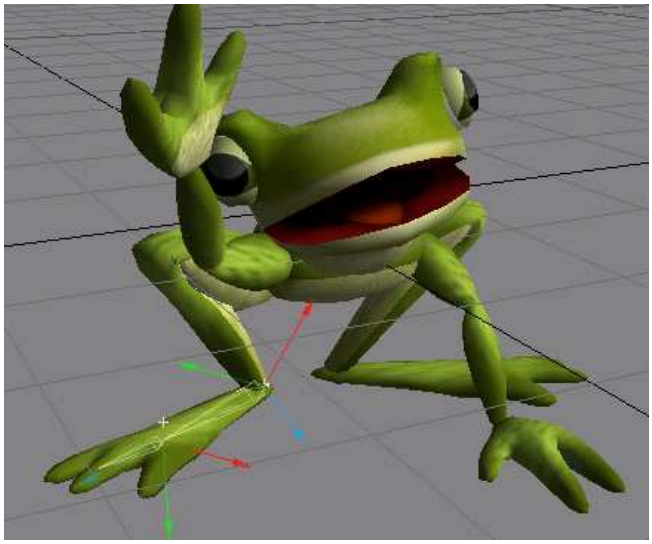


Obrázek 17: model využívající skinning pro vytvoření přirozených ohybů v kloubech (forum.nokia.com)

Skinning přiřazuje různou váhu vrcholům objektu (někdy se používá výraz weight map) a kombinuje matici rotace s těmito parametry. To znamená, že konečná poloha vrcholu je odvozena od transformace vrcholu vzhledem k několika kostem a výpočtu poměrné vzdálenosti.

Z důvodu velkého množství dat potřebných i u jednoduché kostry se nedoporučuje skinning implementovat a animovat manuálně pomocí kódu. Daleko snazší je využít profesionální modelovací software.

Model žáby má desítky definovaných kostí, ale pro jednoduchost jsou ukázány pouze kosti pravé zadní končetiny. První kost hýbe celým chodidlem a druhá přirozeně jenom patou.



Obrázek 18: kosti pravé zadní končetiny (forum.nokia.com)

Javax.microedition.m3g.SkinnedMesh definuje normální Mesh doplněný o kostru, která obsahuje informace o transformacích kostí pro model. Každá kost má předdefinovanou klidovou pozici. Pokud je kost v klidové pozici, je výsledný model přesnou kopií originálu.

3.4.4 RayIntersection

Třída javax.microedition.m3g.RayIntersection se dá použít na nalezení průsečíku ve scéně. Paprsek lze vystřelit z libovolného bodu jakýmkoliv směrem a pokud protne Mesh nebo Sprite, je možné zjistit normálu vrcholu průsečíku, vzdálenost nebo souřadnice textury. RayIntersection má mnoho použití, jako je například selekce objektů nebo detekce kolize.

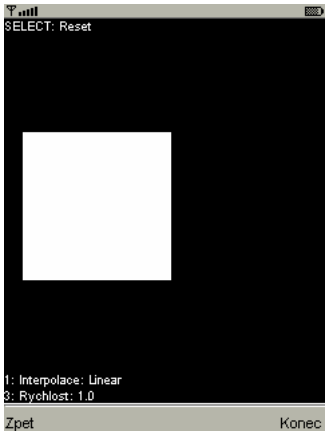
4 Aplikace

Projekt demonstruje v několika příkladech způsob použití aplikačního rozhraní JSR-184. Vývoj probíhal v prostředí NetBeans 5.5 s doinstalovaným SonyEricsson J2ME wireless toolkit WTK2, který umožňuje testování kódu na reálném mobilním telefonu (SonyEricsson K750i). Screenshoty byly pořízeny z emulátoru mobilního telefonu – DefaultColorPhone, který je součástí balíčku J2ME Wireless toolkit 2.2.

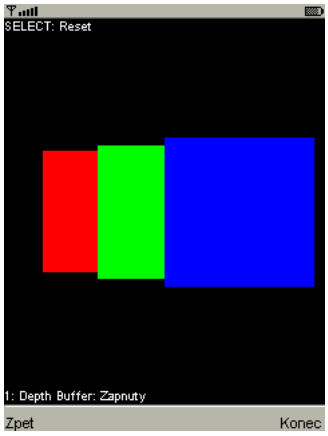
4.1 Hlavní menu

Main.java – zobrazuje nabídku ukázek. Je možné zvolit jeden z příkladů Java mobile 3D graphics nebo aplikaci ukončit. Po spuštění některé z ukázek se načte trojrozměrná scéna, kterou lze ukončit a vrátit se do hlavní nabídky.

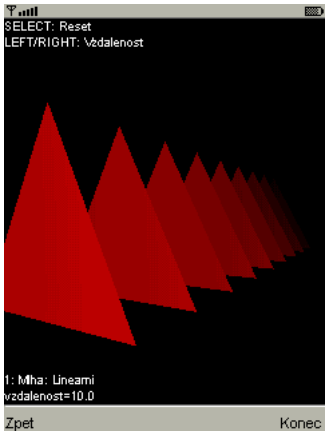
4.2 Přehled ukázek



Animation.java (Animace) – zobrazuje interpolační metody animace čtverce, který se pohybuje mezi čtyřmi předem definovanými body. Klávesou „7“ lze měnit algoritmus interpolační metody pro výpočet aktuální polohy (Step, Linear, Spline) a klávesou „9” rychlost pohybu.

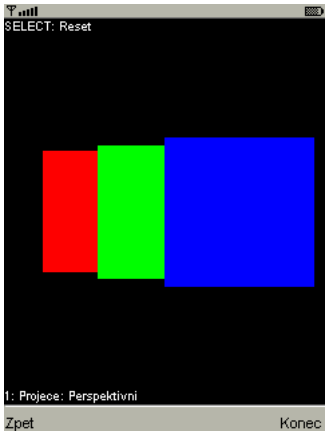


DepthBuffer.java (DepthBuffer) – testuje vliv funkce DepthBuffer na zobrazení třech čtverců. Každý z nich má jinou souřadnici polohy na ose *z* (také na ose *x*, ale pouze z důvodu viditelnosti). Modrý čtverec je v popředí a červený v pozadí. Klávesou „7“ se zapíná a vypíná DepthBuffer.

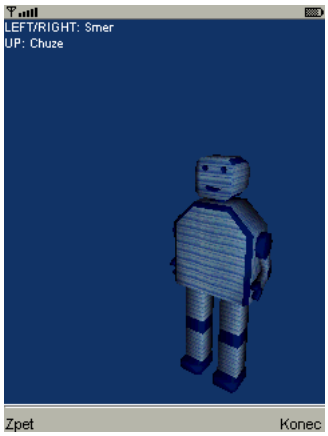


Fogg.java (Mlha) – názorně zobrazuje metody výpočtu mlhy aplikované na trojúhelníky. Klávesou „7“ lze mlhu vypnout nebo nastavit lineární, exponenciální způsob výpočtu vzdálenosti. Šipkami vpravo a vlevo se nastavuje intenzita mlhy.

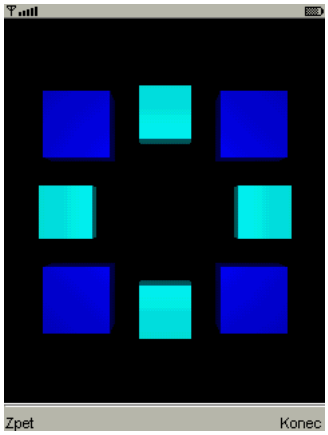
Lighting.java (Svetla) – simuluje vliv typu světla na osvětlení koule. Klávesou „7“ je možné měnit typ světla a to Directional, Omni, Spot a Ambient. Šipkami vlevo a vpravo se rotuje poloha zdroje světla podle osy *y*.



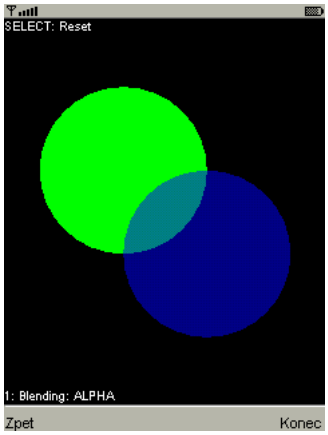
Projection.java (Projekce) – testuje vliv perspektivní korekce na zobrazení scény tří čtverců. Čtverce jsou umístěny stejně jako v ukázce DepthBuffer. Klávesou „7“ se zapíná a vypíná perspektivní korekce scény.



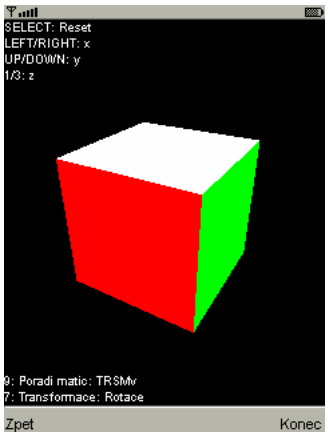
Robot.java (Robot) – zobrazuje trojrozměrný model robota importovaný z *.m3g* souboru. Robot byl vymodelován a animován v prostředí 3D Studio Max. Součástí *.m3g* souboru je i animace chůze. Šipkami vlevo a vpravo se ovládá směr a šipkou nahoru chůze vpřed.



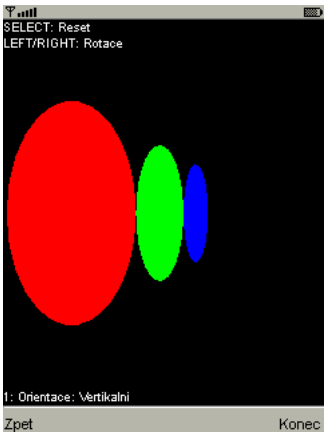
SceneGraph.java (SceneGraph) – prezentuje přístup k objektům pomocí stromové struktury scény (SceneGraph). Jednoduché krychle rotují na kružnici kolem středu obrazovky a současně mění polohu na ose *z*, světle a tmavě modré čtverce navzájem proti sobě. Využívá se zde transformace skupiny prvků (Group).



Compositing.java (Skladani) – zobrazuje metody skládání (blending) objektů. Klávesou „7“ lze nastavit režimy: Alpha, Alpha_add, Modulate, Modulate_x2, Replace.



ModelTransformations.java (Transformace modelu) – prezentuje druhy transformací krychle implementované JSR-184. Klávesou „*“ se mění typ transformace (Posun, Rotace, Meritko, Lomeni), „#“ pořadí součinu matic TRSM nebo RTSM (T – matice posunu, R – matice rotace, S – matice měřítka, M – všeobecná homogenní matice). Transformace se ovládá směrovými šipkami (osa *x*, *y*) a klávesami „7“ a „9“ (osa *z*).



ViewTransformations.java (Transformace pohledu) – zobrazuje transformaci kamery pro vertikální a horizontální pozici. Šipkami vlevo a vpravo se ovládá rotace kamery. V ukázce je zapnutá perspektivní korekce scény.

Závěr

Cílem této bakalářské práce bylo prostudovat principy zobrazování trojrozměrných dat, naučit se základy platformy J2ME a vyvinout ukázkovou aplikaci pro mobilní telefon s využitím podpory 3D JSR-184. API JSR-184 dosáhlo kombinace jednoduchého použití s možností podrobného přístupu (v případě nutnosti), což je velmi kladně hodnocená vlastnost z hlediska vývoje. Ve fázi modelování je doporučené využít profesionální autorský nástroj, jako například 3D Studio Max, pro maximální urychlení vývoje aplikace. Díky rozsahu API JSR-184 práce neobsahuje všechny detaily, ale spíše se zaměřuje na popis a vysvětlení hlavní přístupů při vytváření 3D scény v mobilním telefonu.

Stěžejní částí práce jsou příklady, ze kterých je snadné pochopit způsob přístupu k různým grafickým problémům trojrozměrné scény. Vývoj proběhl v prostředí Netbeans 5.5 s doinstalovaným SonyEricsson J2ME wireless toolkit WTK2, který umožňuje přímé testování aplikace na mobilním telefonu pomocí bezdrátové technologie bluetooth. Ukázky obsahují různé typy transformací kamery a modelu krychle, skládání objektů (compositing mode), práci se scénou (scene graph), příklady různých typů světel, mlhy, animací, ale nejdůležitější částí je import scény z .m3g souboru a její ovládání na klávesnici telefonu.

Možné rozšíření práce vidíme především v detailnějším pohledu na metody a třídy JSR-184 spolu s vývojem kompletního zobrazovacího enginu například pro navigační software GPS.

Seznam použité literatury

[1] J. Žára, B. Beneš, P. Felkel: *Moderní počítačová grafika*. Computer press, 1998

[2] Tomas Akenine-Möller, Eric Haines: *Real-time rendering*. A K Peters, 2002

[3] Alan Watt, Mark Watt: *Advanced Animation and Rendering Techniques – Theory and Practice*. Addison-Wesley, 1992

[4] Internetový magazín – vývoj aplikací J2ME. [online]. [2007-4-17]
URL: interval.cz/vyvoj-aplikaci/j2me

[5] Qusay H. Mahmoud, *JAVA2 – Micro Edition*. Grada, 2002

[6] Internetové stránky vývojářů SonyEricsson. [online]. [2007-4-25]
URL: developer.sonyericsson.com

[7] Internetové fórum Nokia. [online]. [2007-5-15]
URL: www.forum.nokia.com

[8] Specifikace JSR-184: Mobile 3D Graphics API. [online]. [2007-5-23]
URL: www.jcp.org/en/jsr/detail?id=184

[9] Internetové stránky Autodesk 3ds max. [online]. [2007-5-23]
URL: www.autodesk.cz/3dsmax

[10] Plíva, Z., Drábková, J.: Metodika zpracování diplomových, bakalářských a vědeckých prací na FM TUL. [online]. [2006-27-4]
URL: www.fm.tul.cz/files/jak_psat_DP.pdf

Přílohy

Na přiloženém CD je elektronická verze bakalářské práce ve formátu PDF, zdrojové kódy ukázkové aplikace včetně datové struktury projektu pro IDE Netbeans 5.5.