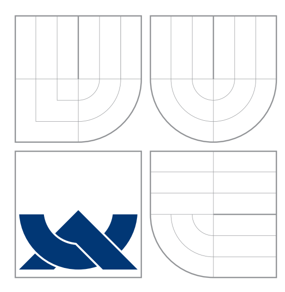
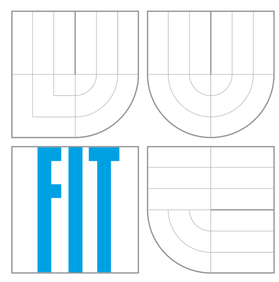


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D APLIKACE V JAVĚ

JAVA 3D APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ROMAN SLAVÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce popisuje rozdíly mezi grafickými knihovnami Java3D API a jMonkey engine. První část práce je zaměřena na vlastnosti a rozdíly v architektuře obou knihoven. Ve druhé části je pomocí několika testů srovnán jejich výkon, přičemž důraz je kladen zejména na počet vykreslených snímků za vteřinu, vytížení procesoru a pamětovou náročnost. Třetí část se pak zabývá návrhem a implementací demonstrační aplikace v prostředí jMonkey engine.

Abstract

This bachelor's thesis describes the differences between the graphic libraries of Java3D API and jMoney engine. The first part deals with the features and differences in the architecture of both libraries. In the second part, their achievements are compared by mean of several tests, especially emphasizing the number of depicted pictures a second, workload of the processor and memory-intensity. The third part concerns with a suggestion and an implementation of a demonstration application in the jMonkey engine environment.

Klíčová slova

Java, Java3D API, j3d, Java Monkey Engine, jme, swing, plugin

Keywords

Java, Java3D API, j3d, Java Monkey Engine, jme, swing, plugin

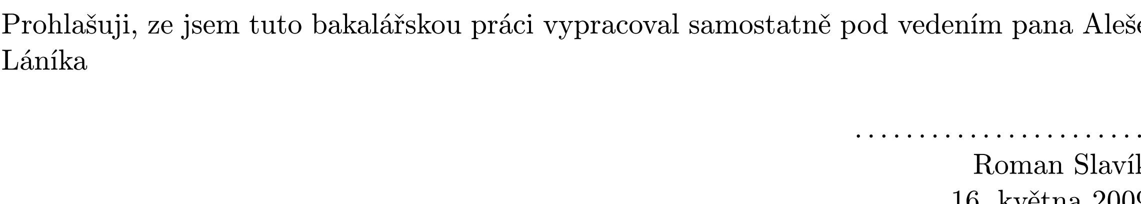
Citace

Roman Slavík: 3D aplikace v Javě, bakalářská práce, Brno, FIT VUT v Brně, 2009

3D aplikace v Javě

Prohlášení

Prohlašuji, ze jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Aleše Láníka



. . . .
Roman Slavík
16. května 2009

Poděkování

Rád bych poděkoval panu Ing. Láníkovi za odbornou pomoc při řešení této práce.

© Roman Slavík, 2009.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **4**

2 Java3D vs jME **5**
- 2.1 Historie . . . 5
- 2.2 Architektura . . . 5
- 2.3 Vazba na knihovnu komponent Swing . . . 8
- 2.4 Podpora . . . 9

3 Testování výkonu knihoven Java3D a jME **11**
- 3.1 Testovací sestavy . . . 13
- 3.2 Návrh a implementace testů . . . 13
- 3.3 Rotující krychle . . . 14
- 3.4 Model sluneční soustavy . . . 16
- 3.5 Zdvojování počtu objektů . . . 18
- 3.6 Souhrn testů . . . 25

4 Návrh aplikace **26**
- 4.1 Grafické prostředí . . . 26
- 4.2 Uživatelské rozhraní . . . 26
- 4.3 Skriptovací konzole . . . 26
- 4.4 Strom uzlů . . . 27
- 4.5 Tvorba pluginů . . . 28

5 Implementace **29**
- 5.1 Mediator . . . 30
- 5.2 Okenní aplikace . . . 31
- 5.3 3D plocha . . . 31
- 5.4 Strom uzlů . . . 33
- 5.5 Skriptovací konzole . . . 34
- 5.6 Pluginy . . . 35

6 Závěr **37**

A Manuál k ovládání aplikace **39**

B Příklad skriptu **42**

C Příklad pluginu **43**

Seznam obrázků

2.1 Přiklad grafu scény 6
2.2 Graf scény knihovny Java3D 7
2.3 Ukázka rasterizace swingových komponent knihovnou jME 9

3.1 Program Fraps 11
3.2 Propojení JConsole s JVM 12
3.3 Znázornění testů 13
3.4 Aplikace s rotující krychlí v Java3D API a jME 14
3.5 Aplikace s modelem sluneční soustavy v Java3D API a jME 16
3.6 Zdvojování objektů v Java3D API a jME při 8191 objektu 18
3.7 Zdvojování počtu objektů - FPS 800x600 Notebook 19
3.8 Zdvojování počtu objektů - FPS 1024x768 Notebook 19
3.9 Zdvojování počtu objektů - FPS 800x600 Desktop 20
3.10 Zdvojování počtu objektů - FPS 1024x768 Desktop 20
3.11 Zdvojování počtu objektů - Paměť 800x600 Notebook 21
3.12 Zdvojování počtu objektů - Paměť 1024x768 Notebook 21
3.13 Zdvojování počtu objektů - Paměť 800x600 Desktop 22
3.14 Zdvojování počtu objektů - Paměť 1024x768 Desktop 22
3.15 Zdvojování počtu objektů - Procesor 800x600 Notebook 23
3.16 Zdvojování počtu objektů - Procesor 1024x768 Notebook 23
3.17 Zdvojování počtu objektů - Procesor 800x600 Desktop 24
3.18 Zdvojování počtu objektů - Procesor 1024x768 Desktop 24

4.1 Návrh uživatelského rozhraní 27
4.2 Návrh stromu uzlů 28

5.1 Diagram tříd 29
5.2 Vzhled aplikace ChimpViewer 30
5.3 Směr pohybu ve FreeMoveMode a RotateMode 31
5.4 Zapnutá síť linek, zapnutý drátový mód a vypnuté světlo 32
5.5 Strom uzlů 33
5.6 Skriptovací konzole 34
5.7 Menu s popisem pluginu 35

Seznam tabulek

3.1 Test otáčení krychle - frekvence vykreslování 14
3.2 Test otáčení krychle - spotřeba paměti 15
3.3 Test otáčení krychle - vytížení procesoru 15
3.4 Test modelu sluneční soustavy - FPS 16
3.5 Test modelu sluneční soustavy - paměť 17
3.6 Test modelu sluneční soustavy - vytížení procesoru 17

5.1 Tabulka uzlů ve stromu 34

1 Úvod

Když v roce 1992 vydala firma ID Software hru Wolfenstein 3D, zapsala se nesmazatelným písmem do historie (nejen) počítačových her. Do této doby se na trhu vyskytovaly pouze v assembleru napsané hry využívající dvourozměrný prostor. Aplikace využívající trojrozměrný prostor existovaly již několik let, ale teprve díky obrovskému úspěchu Wolfenstaina, který dokázal, že lze napsat kvalitní aplikaci v jazyce C, vzrostl zájem o 3D grafiku i mezi obyčejnými lidmi.

Dnes je již 3D grafika běžnou součástí počítačových aplikací. Jazyk C, popř. jeho objektová varianta C++, se stal dominantním programovacím jazykem, assember se používá jen zřídka.

Mezi programátory se ovšem stává čím dál oblíbenějším jiný objektově orientovaný jazyk - Java. K jejim nesporným výhodám patří snadná přenositelnost mezi platformami a rychlost psaní aplikace. Naopak jí bývá vytýkána pomalost a až přílišná robustnost. Může tedy ohrozit pevnou pozici jazyka C v programování 3D grafiky?

Tato práce se zabývá srovnáním dvou knihoven pro práci s prostorovou grafikou v jazyce Java. Jedná se o knihovny Java3D API zaštiťovanou vývojářem javy, firmou Sun, a o opensource projekt jMonkey engine. V první kapitole budou tyto knihovny představeny z pohledu historie, architektury a schopnosti spolupracovat s jinými frameworky.

Druhá kapitola se bude zabývat porovnáním výkonů obou knihoven. V obou knihovnách budou vytvořeny testovací aplikace, ve kterých se bude měřit hlavně paměťová a procesorová nářočnost společně s rychlostí vykreslování.

Ve třetí kapitole bude popsán návrh aplikace demonstrující možnosti prostředí jMonkey engine a kapitola čtvrtá bude zaměřena na samotnou implementaci.

Poslední kapitola bude obsahovat zhodnocení práce, její přínos a možnosti rozšíření do budoucna.

2 Java3D vs jME

2.1 Historie

Java3D

Historie se datuje do roku 1996, kdy se firmy Intel, Silicon Graphics, Apple a Sun rozhodly vytvořit v Javě vlastní implementaci grafu scény [1] k řízení a organizaci 3D aplikací. V březnu 1998 byla uvolněna veřejná betaverze, ze které panovaly smíšené pocity. Knihovna měla mnoho chyb, velmi zatěžovala procesor a ke své činnosti potřebovala neúnosné množství paměti. Vývojáři to vysvětlovali tím, že chtěli komunitu navnadit a slíbili, že všechny problémy odstraní. V prosinci téhož roku byla uvolněna verze 1.0, která velkou část vytýkaných chyb odstranila. Vývoj pokračoval do poloviny roku 2003, kdy Sun práce na projektu pozastavil. V polovině roku 2004 byl vývoj nejen obnoven, ale zároveň byly zveřejněny zdrojové kódy pod licencí public source [5]. Tato licence umožňovala zájemcům z celého světa stahovat zdrojové kódy a přispívat opravami, popř. novými třídami. V únoru 2008 byla licence změněna na GPL verze 2, Sun omezil práce na vývoji a působí spíše v roli dozorce [1]. V současnosti práce na knihovně příliš nepokračují, v komunitě se o ní mluví jako o mrtvé.

jME

Základy Java Monkey Engine byly položeny v roce 2003, kdy Mark Powell studoval OpenGL. Narazil tehdy na LWJGL[2] a rozhodl se použít javu jako programovací jazyk pro své grafické nástroje. Tyto se staly základem nového engineu. Poté, co si Powell přečetl knihu Davida Aberyho 3D Game Engine Design, rozhodl se do svého vznikajícího prostředí implementovat graf scény. Zlomovým okamžikem bylo přidání projektu do seznamu softwaru na java.net. JME se stal zájmem vývojářů her a brzo bylo několik z nich přibráno do vznikající vývojového týmu. Tohle všechno se stalo v roce 2003. Projekt jME byl v polovině roku 2008 pozastaven a byly započaty práce na jME 2.0, který přidává nové možnosti. Koncem roku 2008 byla zvejřeněna jME 2.0 beta. V současnosti na projektu pracují čtyři hlavní vývojáři a desítky přispěvatelů [10].

2.2 Architektura

Prvně je vhodné uvést, co mají obě knihovny společného. Jak Java3D, tak jME používají k řízení a organizaci 3D aplikace graf scény. Tento pojem si nyní vysvětlíme.

[1] Více o grafu scény v podkapitole Architektura
[2] Více v kapitole Architektura

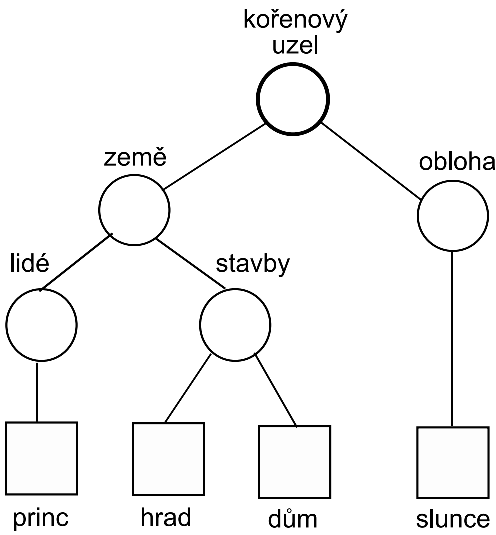
Graf scény

Jedná se o způsob organizace 3D aplikace. Základní grafická pipeline je skryta a nahrazuje ji stromová struktura. Pro vykreslení se tedy nevolají jednotlivé instrukce `OpenGL`, popř. `DirectX`, ale do grafu scény se přidá uzel reprezentující objekt [5].

Uzly se obecně dělí na dva druhy - uzel `Group` a uzel `Leaf`. Uzly mají zpravidla jeden rodičovský uzel, ale nemusí tomu tak být vždy. Výjimkou je kořenový uzel, který nemá žádného rodiče. V některých systémech je možné, aby měl uzel více než jednoho rodiče.

Uzel typu `Group` může seskupovat dohromady další uzly `Group` a `Leaf`. Pokud na uzel tohoto typu aplikujeme určitou transformaci (např. škálování, posun, rotaci), tato se projeví u všech uzlů, které tento seskupuje [7].

Do skupiny typu `Leaf` patří všechny objekty scény, které jsou pozorovatelné, světla, zvuky. Tento uzel může nést informaci o povrchu, barvě, chování. Jedná se o koncový uzel.



Obrázek 2.1: Příklad grafu scény

Java3D

Existuje zde uzel typu `Group`, který se ovšem používá výjimečně. Častěji se používají uzly z něj odvozené, konkrétně uzel `Branch` a `Transform`. Uzel `Branch` dovoluje za běhu programu přidávat další poduzly, zatímco uzel `Transform` umožňuje jejich transformaci. Není tedy možné nad Branch uzlem vykonávat transformace a za běhu programu do `Transform` přidávat potomky. Je ovšem nutné tuto schopnost v uzlu `Branch` povolit.

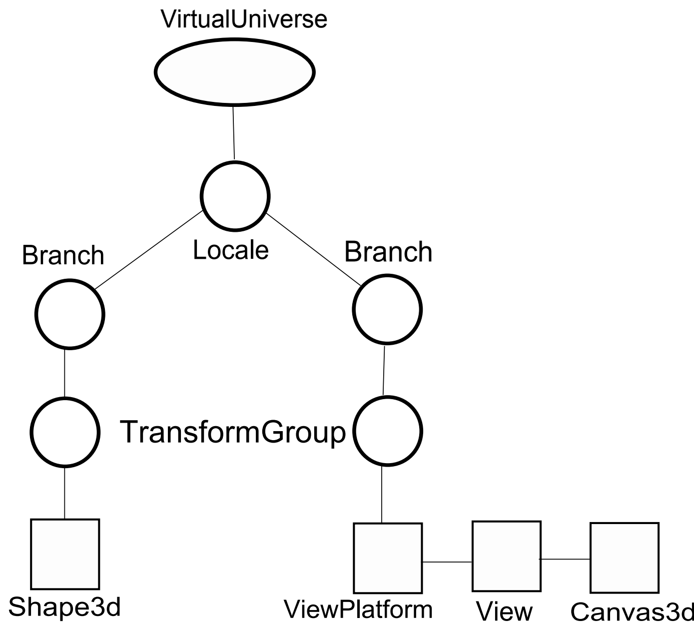
Zajímavým druhem uzlu je `Behavior`. Jedná se o Leaf uzel, který je schopen nést informaci o chování. Jednoduše řečeno je možné vytvořit tento uzel, který přidáním do Transform uzlu zajistí např. rotaci všech potomků `Transform` uzlu.

Kořenovým uzlem je `VirtualUniverse`, který vytváří virtuální 3D svět. Tento uzel je společně se svým potomkem `Locale` v aplikaci pouze jeden. `Locale` určuje absolutní pozici ve 3D světě. Všechny uzly, které tento sdružuje, mají svou svou pozici nastavenou relativně právě k tomuto uzlu.

Pohled na scénu je v Java3D tvořen jako samostatná větev `Locale` uzlu. V této větvi je umístěn uzel `ViewPlatform`, který sdružuje uzly `View` a `Canvas3D`, které se starají o zobrazení scény. Uzlu `ViewPlatform` je nadřazený uzel `TransformGroup`, takže je možno pohled transformovat, popř. mu nadefinovat požadované chování např. pomocí uzlu `Behavior`.

Java3D také narozdíl od většiny ostatních knihoven pro práci s 3D grafikou umožňuje, aby měl uzel více rodičů. V tom případě se na něj aplikují všechny transformace od všech rodičů [5].

V oblasti optimalizace LOD(Level of detail) je na tom knihovna velmi dobře. Sama si optimalizuje scénu a není tedy zapotřebí speciálních funkcí a specializovaných objektů.



Obrázek 2.2: Graf scény knihovny Java3D

jME

Nerozlišuje mezi uzly pro transformace a pro shlukování objektů. Do `Group` uzlu je možno přidávat objekty i za běhu aplikace bez nutnosti zvláštních nastavení. Je také možno s tímto uzlem provádět veškeré transformace.

Zatímco Java3D je prezentována jako knihovna pro všeobecné použití 3D grafiky, tvůrci jME svůj projekt prezentují jako engine, jehož primární cílovou skupinou jsou vývojáři her. Dá se samozřejmě použít i na jiné aplikace, ale samotná knihovna již v základní distribuci nabízí mnoho objektů využitelných právě v počítačových hrách [10].

Každý uzel může mít maximálně jednoho rodiče. I zde platí, že uzel typu `Leaf` nemůže být rodičem žádnému jinému uzlu a naopak uzel `Group` může být rodičem teoreticky neomezeného počtu uzlů.

Při vytváření scény samotná knihovna neprovádí žádné optimalizace. Ty je nutné nastavit. Mezi nejpoužívanější patří nastavení vykreslování pouze viditelných objektů. JME

totiž v defaultním režimu vykresluje i objekty, které nejsou vidět. Další užitečnou optimalizací je snížení úrovně detailů. Je ovšem nutné vytvořit nový objekt `ClodMesh` starající se o snižování LOD.

Kořenový uzel nemusí být žádného vyhrazeného typu. Pohled na scénu není přímo součástí grafu scény, s tou ho pojí `Renderer` - objekt, který se stará o vykreslování scény.

Uzly typu `Leaf` mohou stejně jako v Java3D nést informaci o materiálu, rotaci, tvaru, pozici apod.

2.3 Vazba na knihovnu komponent Swing

Jen těžko si lze představit moderní aplikaci, která obsahovala pouze 3D plochu. Taková aplikace by nebyla uživatelsky přívětivá a pravděpodobně by se nesetkala s kladnou odezvou. Nutnou součástí je přehledné a ucelené uživatelské rozhraní, které se skládá z různých tlačítek, seznamů, popisků, menu. Z tohoto důvodu je důležité se při srovnávání grafických knihoven zaměřit i na jejich schopnosti spolupráce s knihovnami standardních komponent. Nejpoužívanější knihovnou pro vytváření okenních aplikací je v jazyce Java balík `Swing`. Ten obsahuje tlačítka, okna, dialogy, listboxy a třídy ovládající jejich chování. Z tohoto důvodu se v této sekci zaměříme na schopnost integrace 3D plochy do swingové aplikace.

Java3D

Jelikož za vývojem této knihovny stojí stejná firma jako za balíkem `Swing`, je přidání 3D plochy do okenní aplikace velmi snadná a čistá práce.

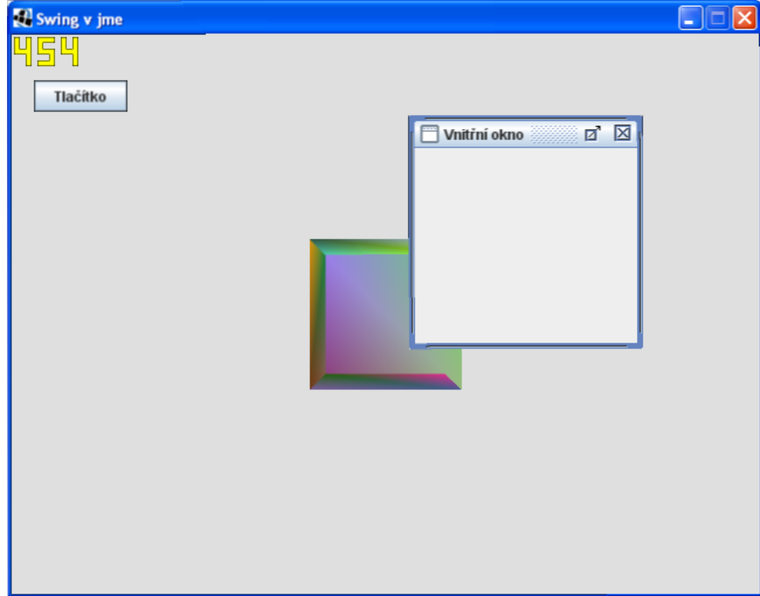
Pro přidání do kontejneru `JPanel` stačí vytvořit `Canvas3d` a ten následně přidat do kontejneru jako jakoukoliv jinou komponentu. Tímto jednoduchým postupem se dají snadno vytvořit aplikace obsahující vykreslovací plochu [5].

jME

Spojení swingových komponent s vykreslovací plochou Java Monkey Engine již není tak jednoduché a průhledné jako při použití Java3D. Je nutné do kontejneru swingové aplikace umístit `awt`[3] `Canvas`, ten pak několikrát přetypovat a nastavit mu určité vlastnosti. Výsledný `Canvas` je ovšem highweight, což znamená, že při aktivním okně musí být vždy v popředí. Jinými slovy: pokud je vedle grafické plochy umístěna nějaká komponenta, která zasahuje do plochy, bude tato vykreslena pod plochou. I toto se dá pomocí několika nastavení změnit, ale tyto úpravy značně znepřehledňují výsledný kód.

Samotní tvůrci knihovny jsou si toho vědomi a do svého projektu implementovali jiný způsob vazby na `Swing`. Tím je vkládání swingových komponent přímo do grafické plochy a jejich následná rasterizace. Tento způsob je hojně používaný ve hrách pro zobrazování různých dialogů. Samotné swingové komponenty si uchovávají všechny vlastnosti, které standardně mají. Navíc je možné nastavit, zda budou komponenty vždy zobrazeny jako dvourozměrný objekt nebo jestli mají být brány jako objekt trojrozměrný, se kterým je možno provádět transformace [10].

[3] `awt` je starší knihovna komponent, ze které vzešel swing



Obrázek 2.3: Ukázka rasterizace swingových komponent knihovnou jME

2.4 Podpora

Může se zdát, že toto kritérium je zbytečné zahrnovat do vlastností knihoven, ale opak je pravdou. Lépe se učí zacházet s knihovnou, ke které je volně dostupná kvalitní dokumentace, je o ní popsáno několik knih od různých autorů, kteří se zaměřují na jiné vlastnosti než jejich kolegové a která má širokou základnu uživatelů. Často se stává, že programátor během vytváření aplikace narazí na problém, který není schopen sám vyřešit. Ve většině případů problém prokonzultuje s kolegy (pokud vyvíjí v týmu) a pokud neobjeví řešení, začne vyhledávat v literatuře, popř. na internetu. Pokud ovšem chybí kvalitní dokumentace, podpora je mizivá a o knihovně nejsou popsány žádné knihy, pak stojí za zvážení, jestli se vůbec vyplatí psát projekt za pomocí takové knihovny.

Java3D

Knihovna je podporována nadnárodní společností a byla první knihovnou umožňují tvorbu 3D scény v jazyce Java. Je o ní tudíž napsáno několik knih, na internetu je dostupná kvalitní dokumentace a mnoho tutoriálů. Není tedy problém naučit se s ní zacházet.

Potíž ovšem nastává, pokud vývojář narazí na nějaký sofistikovaný problém. Existuje šance, že řešení nalezne na internetu, za zmínku ovšem stojí, že stránky věnující se problematice Java3D API bývají několik let staré a neaktualizované, popř. vlákna na vývojářských fórech mají poslední příspěvek starý pár let. Jestli programátor řešení nenajde a hodlá se poptat na diskuzním fóru, může být nemile překvapen dobou čekání na hodnotnou odpověď. Toto je bohužel způsobeno faktem, že ač se jedná o oficiální knihovnu pro práci s trojrozměrnou grafikou, její vývoj byl pozastaven a mnoho jejích uživatelů přešlo na jiné alternativy.

jME

Dalo by se říct, že jME je v oblasti podpory opak Java3D API. Knihovna je opensource, stojí za ní tým nadšenců, v dokumentaci chybí popis mnoha tříd, neexistuje žádná odborná literatura. Tutoriálů je poskrovnu a vysvětlují pouze základy.

Pokud se ovšem vývojář přenese přes počáteční problémy, získá k práci knihovnu, která má velkou uživatelskou základnu a která se neustále vyvíjí. Jestliže vyvstanou potíže, je velká šance, že řešení je již publikováno na oficiálním fóru knihovny. Pokud zde není a je položen dotaz, rychlost odpovědi je vysoká a často bývá nabídnuto i několik řešení.

Shrnutí

Zatímco Java3D má lepší naučnou literaturu, více tutorialů a kvalitnější dokumentaci, jME disponuje širokou uživatelskou základnou, která je schopna rychle a efektivně nabídnout řešení i na první pohled neřešitelných problémů.

3 Testování výkonu knihoven Java3D a jME

Snad nejdůležitějším požadavkem vývojářů je, aby výsledná aplikace byla rychlá. Ne ovšem, aby jen na nejnovějších strojích dosahovala ohromné rychlosti, ale aby byla plynulá i na počítačích starších. Z tohoto důvodu patří výkonové testy, někdy také nazývané performance tests, mezi nejdůležitější faktory ovlivňující konečný výběr technologie.

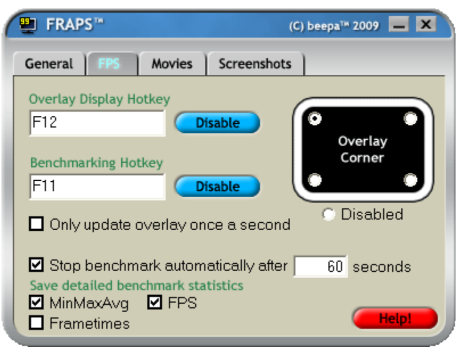
Tvůrci Java3D prezentují svůj projekt jako knihovnu použitelnou pro všechny druhy 3D aplikací. Naproti tomu tým kolem jME se zaměřuje na použitelnost při tvorbě her, tedy aby engine vykresloval dostatečný počet snímků za vteřinu pro vznik iluze plynulého pohybu.

Z výše uvedeného tedy vyplývá, že jME by měl Java3D API porážet v počtu vykreslovaných snímků za vteřinu. Toto bude jedno z kritérií, na které budou testy zaměřeny. Druhým měřeným faktorem bude množství paměti, kterou aplikace potřebuje ke svému chodu. Třetím aspektem pak bude zatěžovaní procesoru.

K měření počtu snímků za vteřinu byl použit program **Fraps**®. Pro měření alokované paměti a vytížení procesoru zase program **JConsole**.

Fraps

Jedná se o program pro měření počtu vykreslených snímků za vteřinu. Byl vyvinut (a dále vyvíjen) firmou beepa®. Mimo zachycování frekvence vykreslování umí Fraps také nahrávat video z aktuálně spuštěné aplikace. Tato funkce ovšem nebyla v práci využita, proto se jí nebudeme dále zabývat.



Obrázek 3.1: Program Fraps

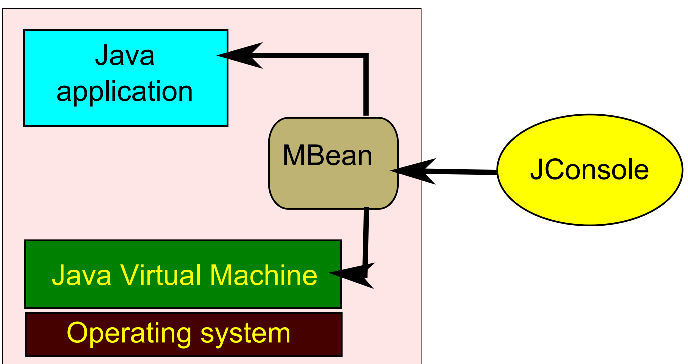
Důvodem, proč byla pro testování vybrána právě tato aplikace je ten, že program je schopen zaznamenávat hodnoty naměřených frekvencí do externího souboru. Tato funkce se velmi hodí při vytváření grafů růstu (poklesu) frekvence v závislosti na čase a změně scény.

JConsole

Tento program byl vyvinut firmou Sun a je součástí `JDK (Java Development Kit)` od verze 5.0. Protože se jedná o stejnou firmu, která zaštiťuje vývoj samotné Javy, lze předpokládat, že naměřené údaje budou odpovídat skutečnosti.

Aplikace je schopna měřit výkon více aplikací najednou. Pro započetí měření je potřeba mít spuštěnou javovou aplikaci, kterou chceme zkoumat. Každá javovská aplikace je propojena s Java Virtual Machine pomocí rozhraní `MBean`, které uchovává reference na všechny spuštěné javovské programy. K tomuto rozhraní se JConsole připojuje [3].

Tato monitorovací aplikace je schopna měřit množství využité paměti na hromadě (heap), mimo hromadu (non-heap), počet běžících vláken, načtených tříd a vytížení procesoru. V rámci této práce budou relevantní pouze údaje o použité paměti na hromadě a o vytížení procesoru. Aplikace též umí export naměřených údajů do souboru cvs, což se velmi hodí pro případnou tvorbu grafu.



Obrázek 3.2: Propojení JConsole s JVM

3.1 Testovací sestavy

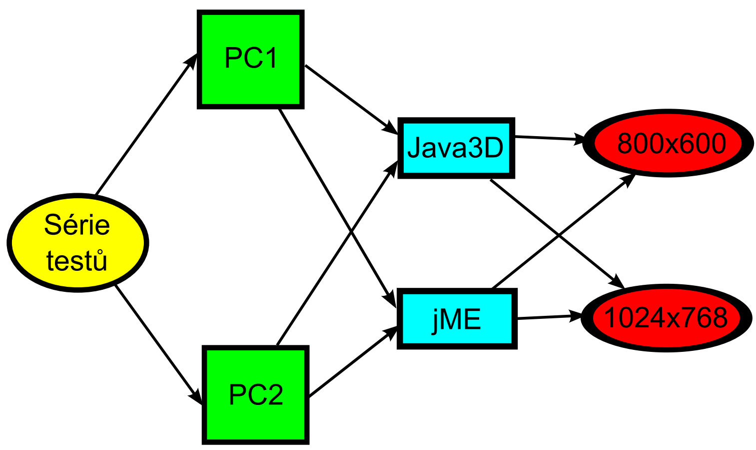
Testy budou prováděny na 2 počítačích s následující konfigurací:

Sestava Notebook
Procesor: Intel Core 2 Duo T8300 @2,40GHz
Paměť: 2 x 2048MB
Grafická karta: nVidia Quadro FX 360M 512MB
Operační systém: Windows XP 64-bit Professional SP2

Sestava Desktop
Procesor: Intel Core 2 Quad Q6600 @2,40GHz
Paměť: 4 x 2048 MB
Grafická karta: nVidia GeForce 8800GT 512MB
Operační systém: Windows XP 64-bit Professional SP2

3.2 Návrh a implementace testů

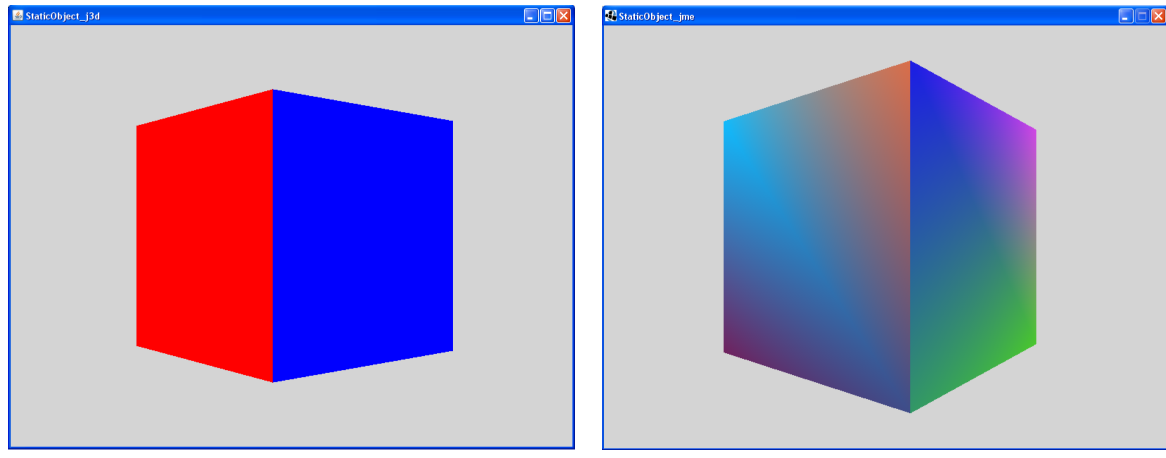
Testování se skládá ze série testů. Každá série se sestává ze dvou samostatných aplikací - jedné využívající knihovnu Java3D a druhé, která využívá jME. Obě aplikace budou s ohledem na vnitřní architekturu knihoven vytvářet totožný, popř. v rámci možností co nejvíce podobný graf scény. Každé měření bude prováděno ve dvou rozlišeních - 800x600 a 1024x768 bodů. Za běhu testovacích aplikací nebude kromě programů Fraps a JConsole spuštěna žádná další aplikace, aby nedocházelo ke zkreslování výsledků. Každá série bude provedena minimálně čtyřikrát a z naměřených hodnot bude vytvořen aritmetický průměr.



Obrázek 3.3: Znázornění testů

3.3 Rotující krychle

První ze série testů je zaměřen na schopnost knihoven vykreslovat jednoduchou scénu. Pro tyto účely byla vybrána otáčející se barevná krychle. Ta je v testu knihovny Java3D reprezentována třídou `ColorCube` - krychlí, která má každou stěnu vyvedenou v jiné barvě. Pro jME byl vybrán objekt `Box`, kterému byly nastaveny náhodné barvy povrchu metodou `randomColor()`. Celá scenérie je vykreslena za použití pouze ambientního světla. Během testování byly testovací aplikace spuštěny minimálně po dobu 1 minuty, aby naměřené hodnoty byly co nejméně ovlivněny chvilkovými výkyvy.



Obrázek 3.4: Aplikace s rotující krychlí v Java3D API a jME

FPS

Na sestavě Notebook při rozlišení 800x600 bodů dosáhla Java3D frekvence 645,567 snímků/s. Průměr jME byl 647,45 snímků/s. Rozdíl mezi oběma knihovnami je zanedbatelný. U nastavené velikosti okna 1024x768 získala Java3D 419,483 snímků/s a jME 412,433 snímků/s. I zde je rozdíl minimální, na dané konfiguraci hardwaru dosáhly knihovny velmi podobných výsledků.

U sestavy Desktop byly již rozdíly markantní. Zatímco v nižším z rozlišení dosáhla Java3D 1882,55 snímků/s, jME získala dokonce 6272,017 snímků/s. U vyššího rozlišení pak naměřená hodnota činila 1738,483 snímků/s u Java3D API a 4080,2 snímků/s u jME. Knihovna jME dosáhla znatelně lepších výsledků.

	800x600		1024x768	
	Java3D	jME	Java3D	jME
Notebook	645,567	647,450	419,483	412,433
Desktop	1882,550	6272,017	1738,483	4080,200

Tabulka 3.1: Test otáčení krychle - frekvence vykreslování

Paměť

Na slabší sestavě při nižším rozlišení zabrala knihovna Java3D pro alokaci svých objektů 3,01 MB paměti. JME naproti tomu spotřebovala už 3,8 MB paměti. Při rozlišení 1024x768 si Java3D vzala 2,91 MB a jME 3,87 MB paměti. Na slabší konfiguraci tedy Java3D pro svou činnost potřebovala méně paměti než jME.

U sestavy Desktop a rozlišení 800x600 byla spotřeba paměti 2,87 MB u Java3D API a 3,01 MB u knihovny jME. Při vyšším rozlišení činil tento údaj 2,84 MB paměti u Java3D API a celých 3,94 MB při použití knihovny jME. I na této sestavě potřebovala jME pro svou činnost více paměti.

	800x600		1024x768	
	Java3D	jME	Java3D	jME
Notebook	3,01 MB	3,8 MB	2,91 MB	3,87 MB
Desktop	2,87 MB	3,01 MB	2,84 MB	3,94 MB

Tabulka 3.2: Test otáčení krychle - spotřeba paměti

Zatížení procesoru

Na sestavě Notebook bylo při rozlišení 800x600 naměřeno knihovně Java3D průměrné vytížení procesoru 49,47%, u jME toto vytížení činilo 49,27%. Při zvýšení rozlišení na 1024x768 zatěžovala aplikace využívající Java3D API na 50,27%, při použití jME toto činilo 50,36%. Výsledky jsou velmi podobné, rozdíl mezi zatížením procesoru je minimální.

Při provádění testů na sestavě Desktop byly výsledky již zajímavější. Zatímco při rozlišení 800x600 vytěžovala Java3D procesor pouze na 28,52%, u knihovny jME to bylo již 41,44%. Na vyšším rozlišení bylo průměrné vytížení 28,38% u Java3D API a 43,35% při použití jME. Rozdíly u této sestavy byly znatelné a jako pravděpodobné se jeví, že Java3D umí lépe pracovat s vícejádrovými procesory.

	800x600		1024x768	
	Java3D	jME	Java3D	jME
Notebook	49,47%	49,27%	50,27%	50,36%
Desktop	28,52%	41,44%	28,38%	43,35%

Tabulka 3.3: Test otáčení krychle - vytížení procesoru

Zhodnocení

Zatímco na slabší sestavě Nobebook dosáhly obě knihovny porovnatelných výsledků, u silnější sestavy byly rozdíly v jednotlivých kritériích výrazné. JME se jeví jako knihovna schopná vykreslovat vyšší počet snímků za vteřinu, kdežto Java3D zřejmě vyniká v lepším využití vícejádrových procesorů. Zatímco u jME bylo využití procesoru srovnatelné, Java3D dosáhla na procesoru se 4 jádry skoro 30% úspory procesoru. V oblasti spotřeby paměti byla lepší knihovna Java3D, která ke své činnosti potřebovala alokovat méně paměti než jME.

3.4 Model sluneční soustavy

Druhý test testuje, jak jsou knihovny schopny zobrazovat animace. Scéna se bude skládat ze středové koule, která reprezentuje slunce, a několik koulí, které budou kolem středové koule obíhat. Jak Java3D, tak jME pro takovou aplikaci poskytují vhodné objekty, v obou byla pro model koule použita třída `Sphere`. Pohled byl pevně nastaven do bodu, odkud jsou viditelné všechny objekty scény. Opět nebylo použito žádné bodové nebo směrové světlo. Scénu osvětluje pouze světlo ambientní. Obě aplikace byly spuštěny po dobu 1 minuty, přičemž měřené hodnoty byly zaznamenávány až po 10s z důvodu ustálení scény.

Obrázek 3.5: Aplikace s modelem sluneční soustavy v Java3D API a jME

FPS

Zatímco v předchozím testu se rozdíly mezi aplikací využívající knihovnu Java3D a tutéž aplikací, která používá jME, projevily až na silnější konfiguraci, v tomto testu se rozdíly začaly objevovat už na sestavě Notebook. Při rozlišení 800x600 byla průměrná frekvence Java3D API 686,433 snímků/s a knihovny jME 748,1 snímků/s. Rozdíl tedy činil více než 60 snímků. Při vyšším rozlišení bylo knihovně Java3D naměřeno 443,083 snímků/s a jejímu konkurentu 497,3 snímků/s.

	800x600		1024x768	
	Java3D	jME	Java3D	jME
Notebook	686,433	748,100	443,083	497,300
Desktop	1893,033	3889,650	1796,933	2876,700

Tabulka 3.4: Test modelu sluneční soustavy - FPS

Na silnější sestavě Desktop byly rozdíly o poznání větší. Při rozlišení 800x600 sice Java3D zaznamenala slušných 1893,033 snímků/s, ovšem jME bylo naměřeno 3889,65 snímků/s. Na vyšším rozlišení činila frekvence knihovny Java3D 1796,933 snímků/s a knihovny jME

2876,7 snímků/s. Rozdíly již byly značné, na nižším rozlišení dosáhla knihovna jME více než 100% nárustu rychlosti. U rozlišení vyššího byl rozdíl více než 1000 snímků za vteřinu.

Paměť

Výsledky prvního testu dávaly tušit, že Java3D dopadne lépe i v tomto testu. Opak je ovšem pravdou. Na sestavě Notebook a rozlišení 800x600 spotřebovala Java3D průměrně 3,32 MB a jME pouze 2,96 MB. U vyššího rozlišení 1024x768 činil průměr u Java3D API 3,31 MB, jME potřebovala pouze 2,99 MB.

	800x600		1024x768	
	Java3D	jME	Java3D	jME
Notebook	3,32 MB	2,96 MB	3,31 MB	2,99 MB
Desktop	3,06 MB	2,95 MB	3,16 MB	2,96 MB

Tabulka 3.5: Test modelu sluneční soustavy - paměť

Při testování na sestavě Desktop dopadla lépe také knihovna jME, ovšem rozdíl již nebyl výrazný. V rozlišení 800x600 byla v Java3D API naměřena spotřeba 3,06 MB a v jME 2,95 MB. Pokud byly aplikace spuštěny v rozlišení vyšším, dosáhla Java3D spotřeby 3,16 MB a jME 2,96 MB.

Zatížení procesoru

Stejně jako v předchozím testu byly výsledky obou knihoven na slabší konfiguraci srovnatelné. Java3D vytěžovala u rozlišení 800x600 procesor na 51,00% a jME na 50,18%. U rozlišení vyššího byly tyto hodnoty 49,60% u Java3D API a 50,12% za použití jME. Hodnoty jsou velmi podobné, drobné odchylky mohou být způsobeny chybou měření a při vyšším počtu opakování mohou být smazány úplně.

Již dříve byla zmíněna teorie, že Java3D pracuje lépe s vícejádrovými procesory. Tato teorie byla umocněna výsledky testů na silnější sestavě, kde při rozlišení 800x600 vytěžovala Java3D procesor na 16,88%, zatímco jME na 38,72%. U rozlišení vyššího potřebovala Java3D ke své činnosti 20,08% a jME 38,65%. Java3D vytěžuje procesor znatelně méně než jME, což může být způsobeno právě lepší praci s vícejádrovými procesory.

	800x600		1024x768	
	Java3D	jME	Java3D	jME
Notebook	51,00%	50,18%	49,60%	50,12%
Desktop	16,88%	38,72%	20,08%	38,65%

Tabulka 3.6: Test modelu sluneční soustavy - vytížení procesoru

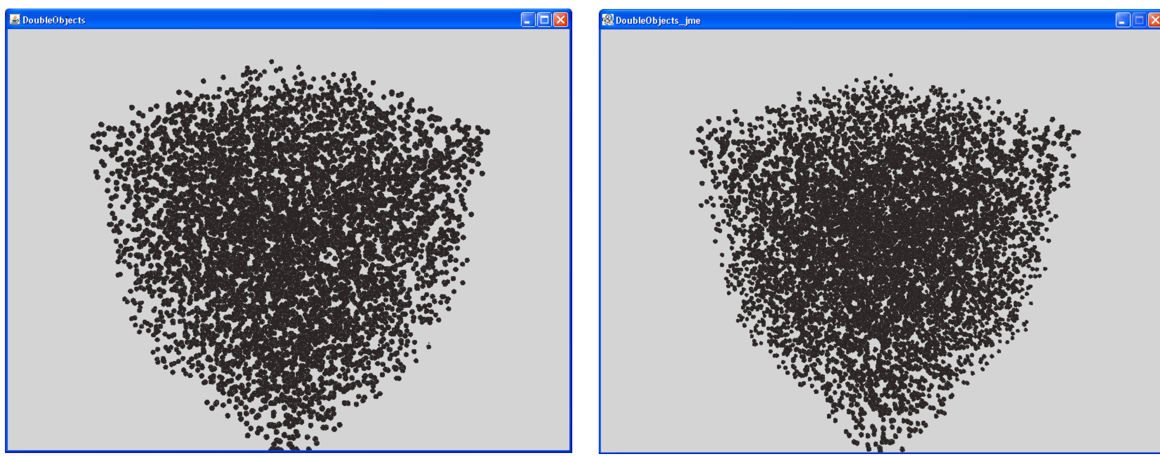
Zhodnocení

I na slabší sestavě dokázal jME vyrenderovat více snímků než Java3D. Spotřeboval také méně paměti, rozdíl činil cca 10%. Procesor obě aplikace zatěžovaly stejně. Celkově na slabší sestavě dosáhl lepších výsledků jME.

JME dosáhl vyššího počtu zobrazených snímků i na silnější sestavě. V rozlišení 800x600 dokázal jME vykreslit více než dvojnásobný počet snímků než Java3D, při vyšším rozlišení byl nárůst zhruba 50%. Rozdíl ve spotřebě paměti byl lehce ve prospěch jME, ale rozdíl byl minimální. Java3D ovšem o mnoho méně zatěžuje procesor. Zatímco Java3D v nižším rozlišení využívala pouze necelých 17% výkonu, jME potřeboval již více než 38%. Ve vyšším rozlišení se rozdíl lehce snížil, ovšem v tomto ohledu jednoznačně zvítězila Java3D.

3.5 Zdvojování počtu objektů

Dalším ze série testů je měření, jak jsou knihovny schopny se vypořádat s vysokým počtem objektů ve scéně. Pro toto testování byly vytvořeny aplikace, které postupem času zvyšují počet svých objektů scény. Program ve výchozím stavu obsahuje jen jeden objekt - kouli tvořenou třídou `Sphere`. Následně je každých 5 vteřin vyvolána akce, která do scény přidá dvojnásobný počet koulí, než který byl přidán minule. Po prvních 5 sekundách jsou tedy přidány 2 nové koule, poté 4 koule, 8 koulí, atd. Souřadnice přidávaných koulí jsou náhodně generovány a jsou ohraničeny krychlí. Z tohoto testu se dá i vypozorovat, jak se postupem času mění požadavky aplikace na paměť a procesor. Tato série se nebude zabývat průměrnými hodnotami, nýbrž jejich změnou v čase. Test měří počet objektů až do počtu 32768, při vyšším počtu již nastával problém s pamětí spotřebovanou JVM[1].



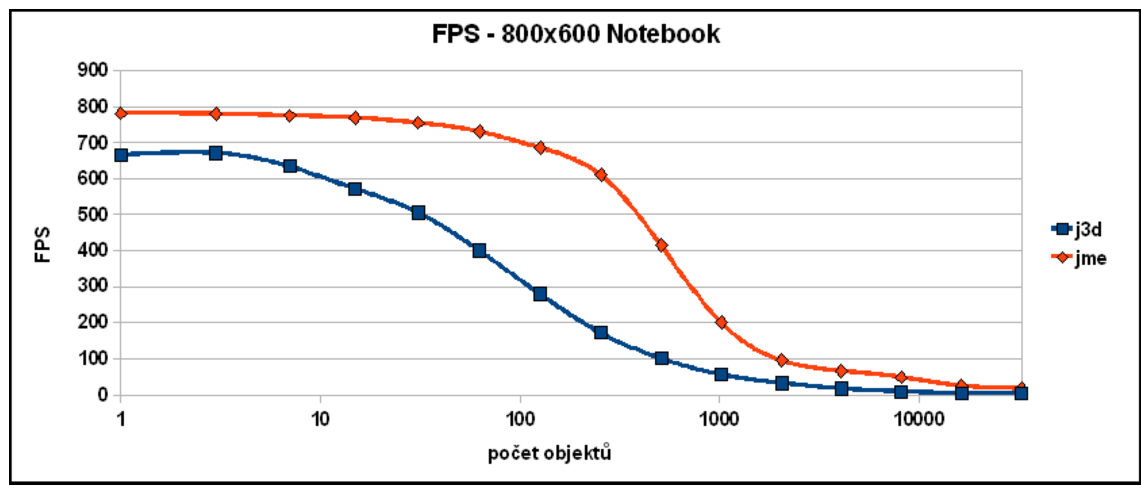
Obrázek 3.6: Zdvojování objektů v Java3D API a jME při 8191 objektu

FPS

Na sestavě Notebook dosáhla lepších výsledků knihovna jME. Zatímco při rozlišení 800x600 a 1 počátečním objektu Java3D vykreslovala scénu frekvencí kolem 760 snímků/s, jME

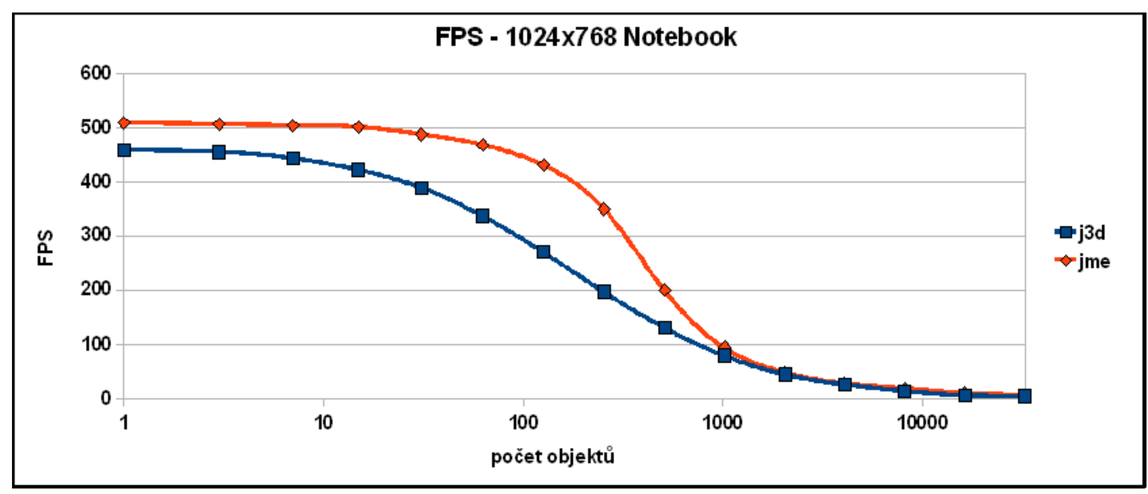
[1] Java Virtual Machine

vykreslovala rychlostí necelých 800 snímků/s. S rostoucím počtem objektů vykreslovací frekvence klesala, stále ovšem jME zvládala vykreslovat scénu rychleji než Java3D.



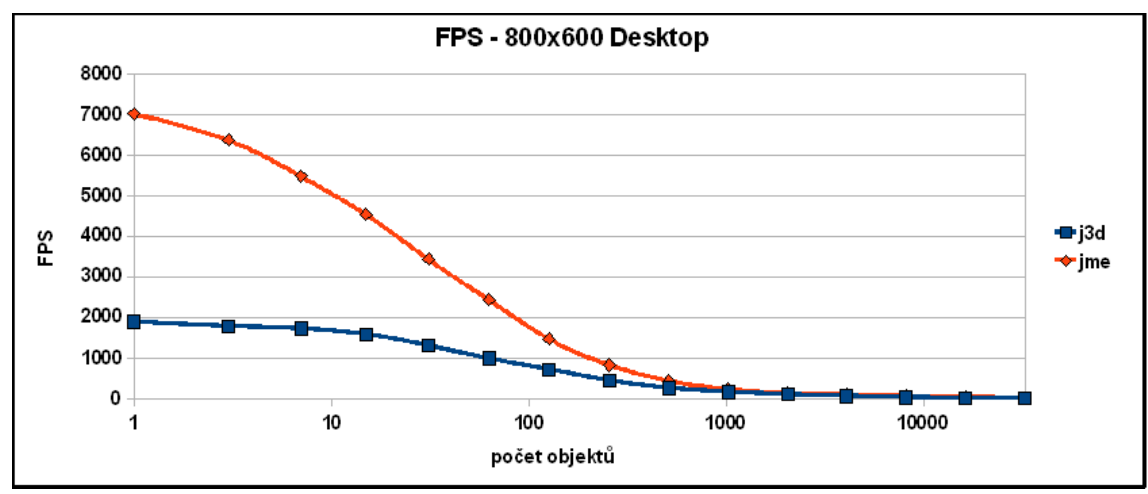
Obrázek 3.7: Zdvojování počtu objektů - FPS 800x600 Notebook

Při rozlišení 1024x768 byla počáteční frekvence u Java3D API okolo 460 snímků/s, u jME lehce přes 500 snímků/s. S postupujícím časem frekvence u obou knihoven klesaly, jME ovšem po celou dobu testu stíhala vykreslovat rychleji než konkurent.



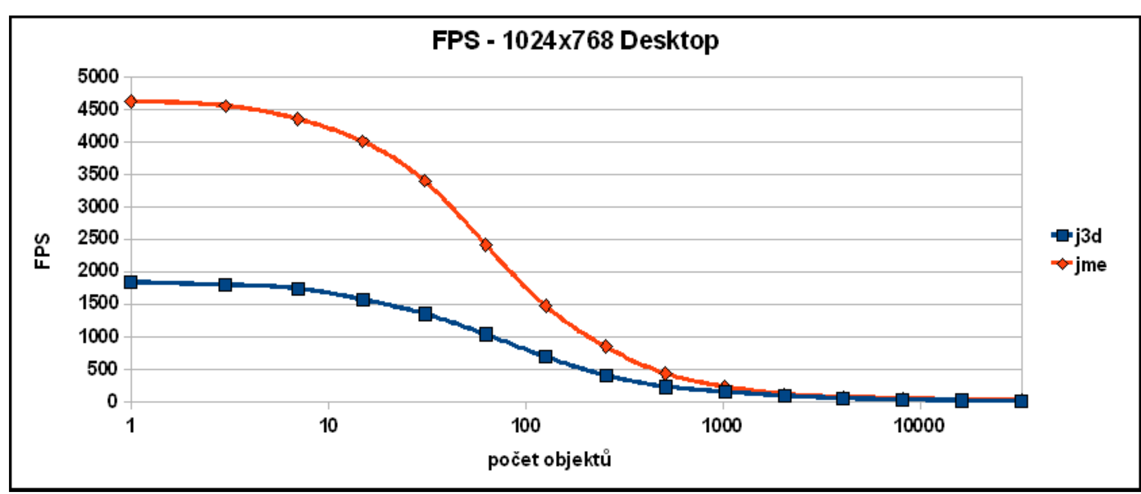
Obrázek 3.8: Zdvojování počtu objektů - FPS 1024x768 Notebook

Na silnější sestavě Desktop se rozdíl v rychlosti obou knihoven ještě prohloubil. Zatímco při rozlišení 800x600 Java3D na počátku vykreslovala rychlostí necelých 2000 snímků/s, jME stejnou scénu vykreslovala rychlostí 7000 snímků/s. Tyto hodnoty postupně klesaly, ovšem po celou dobu jME vykazovala lepší výsledky než Java3D.



Obrázek 3.9: Zdvojování počtu objektů - FPS 800x600 Desktop

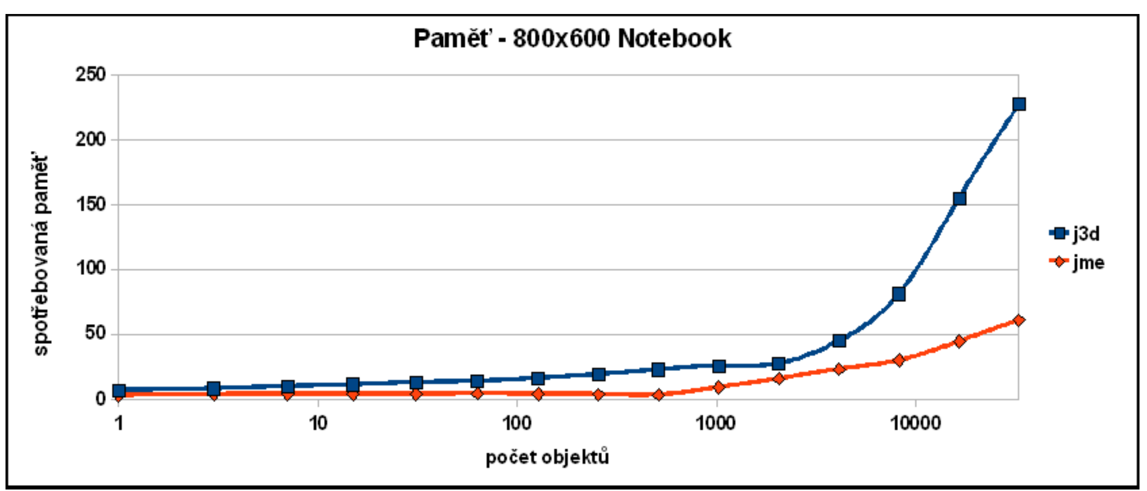
U vyššího rozlišení měla aplikace napsaná pomocí Java3D API frekvenci necelých 2000 snímků/s, jME vykreslovala rychlostí více než 4500 snímků/s. Tyto rychlosti stejně jako v předchozích testech postupem času klesaly, po celou dobu ovšem jME vykazovala lepší výsledky.



Obrázek 3.10: Zdvojování počtu objektů - FPS 1024x768 Desktop

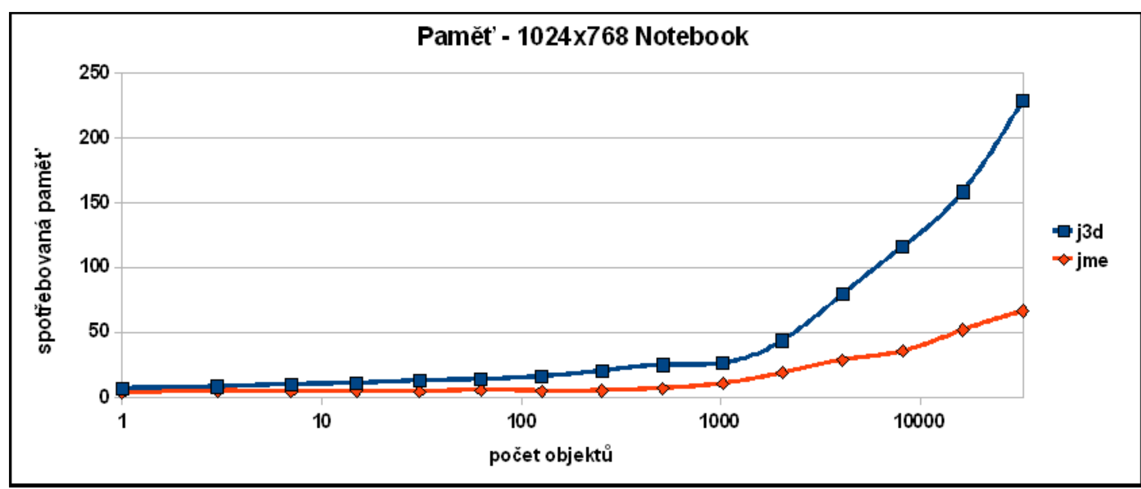
Paměť

Na konfiguraci Notebook při rozlišení 800x600 a 32767 zobrazených objektech spotřebovala pro svou alokaci knihovna Java3D více než 225MB paměti. Naproti tomu knihovně jME pro stejný počet objektů stačilo pouze 55MB. Pro toto rozlišení je tedy vhodnější jME, která spotřebovala 4x méně paměti než Java3D.



Obrázek 3.11: Zdvojování počtu objektů - Paměť 800x600 Notebook

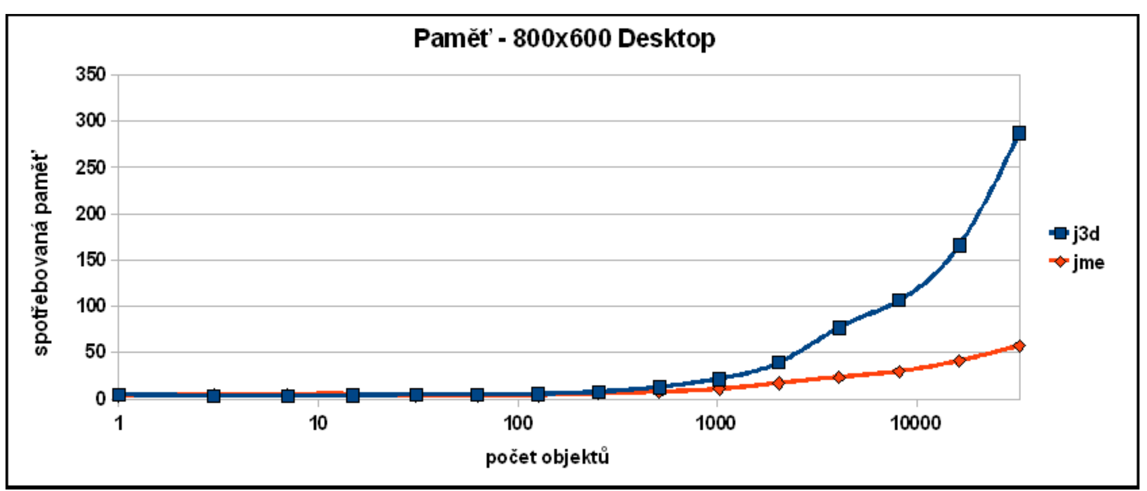
Při zvětšení rozlišení na 1024x768 spotřebovala Java3D skoro 240MB paměti a jME pouze 60MB. Z tohoto je zjevné, že pro slabší konfiguraci hardwaru dosahuje jME mnohem lepších výsledků v oblasti spotřeby paměti.



Obrázek 3.12: Zdvojování počtu objektů - Paměť 1024x768 Notebook

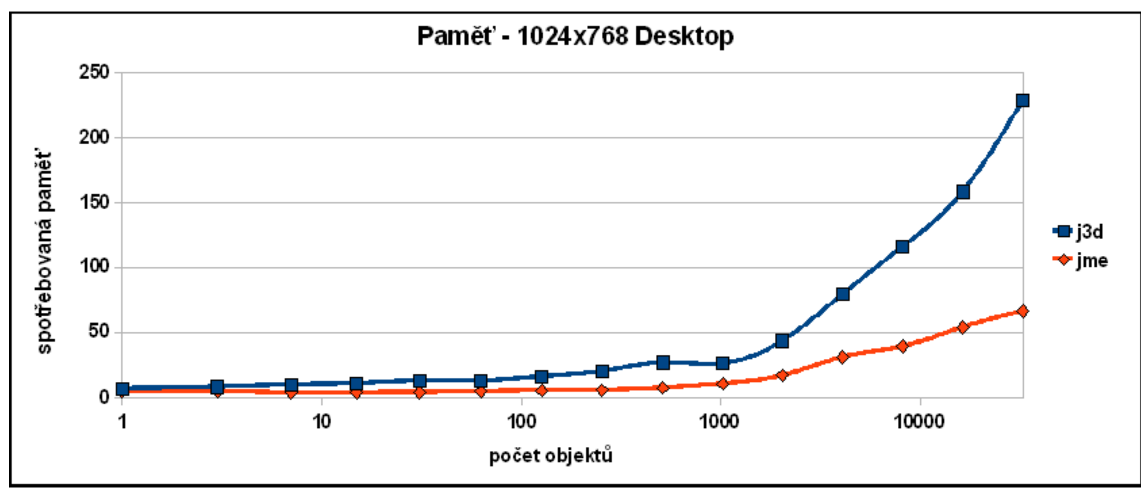
Během testování na výkonnějším počítači Desktop se ukázalo, že i na silnějších sestavách dosahuje mnohem lepších výsledků jME. Při rozlišení 800x600 spotřebovala Java3D až 238MB paměti. JME ovšem stačilo kolem 52MB pro vykreslení stejné scény.

Ani zvýšení rozlišení na 1024x768 žádnou změnu nepřineslo. I zde dosahovala knihovna jME mnohem lepších výsledků, potřebovala pro svou alokaci maximálně 63MB paměti oproti více než 240MB v případě knihovny Java3D. Z těchto výsledků lze vyvodit, že knihovna jME při své činnosti pracuje s menším množstvím objektů o menší velikosti. Z tohoto důvodu je vhodnější pro práci na slabších strojích v situacích, kdy je potřeba



Obrázek 3.13: Zdvojování počtu objektů - Paměť 800x600 Desktop

manipulovat s vysokým počtem objektů.

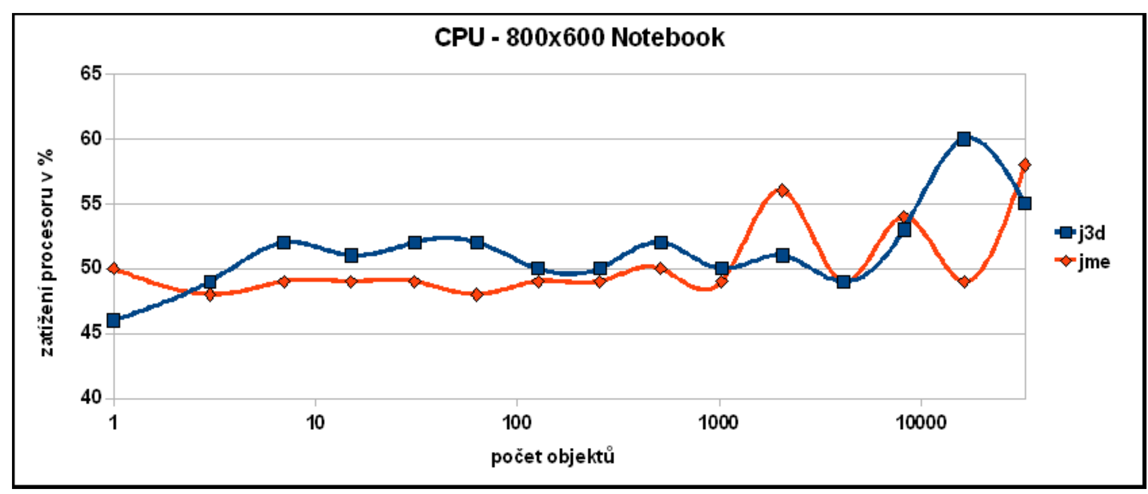


Obrázek 3.14: Zdvojování počtu objektů - Paměť 1024x768 Desktop

Zatížení procesoru

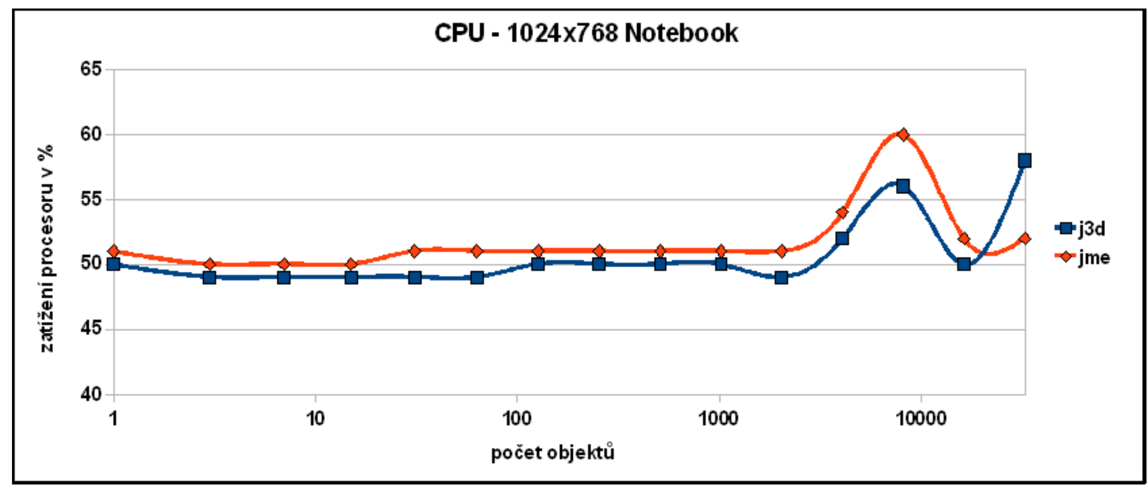
Na slabším počítači Notebook při rozlišení 800x600 produkovaly oba programy srovnatelné zatížení procesoru. Rozdíly byly v řádech procent, tedy zanedbatelné. Zajímavostí je, že s rostoucím počtem objektů vytížení procesoru nijak dramaticky nestoupalo, rozdíl mezi nejvyšší a nejnižší zátěží u Java3D API byl 15% a u jME 8%. Tyto nejvyšší a nejnižší hodnoty byly ovšem v obou případech pouze chvilkové.

Jakmile bylo rozlišení zvýšeno na 1024x768, rozdíly mezi oběma knihovnami se ještě ztenčily. Pouze při vykreslení všech 32767 koulí byl rozdíl mezi zatížením programu s Java3D



Obrázek 3.15: Zdvojování počtu objektů - Procesor 800x600 Notebook

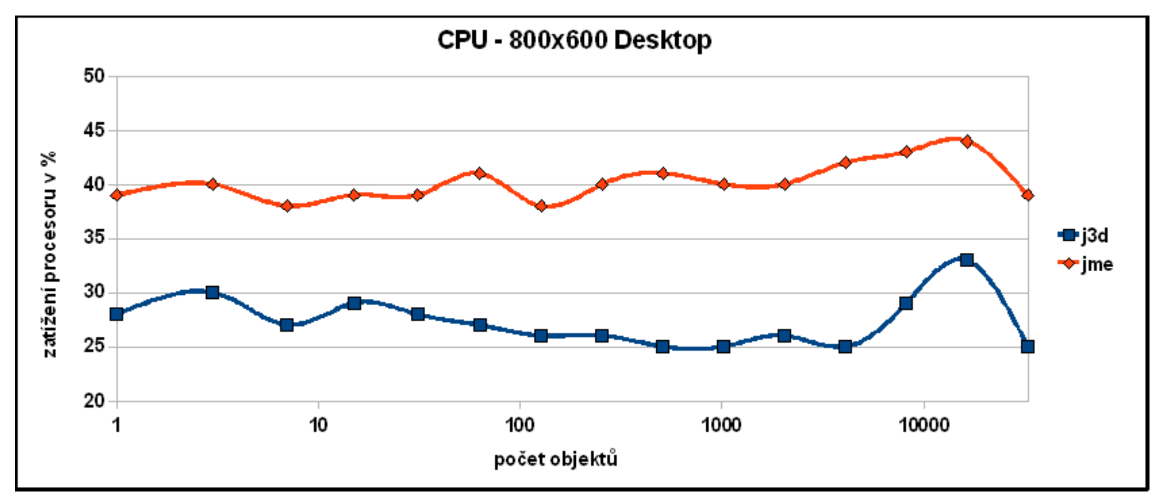
API a programu s jME větší než 5%. Naměřené hodnoty byly velmi podobné, pohybovaly se kolem 50%.



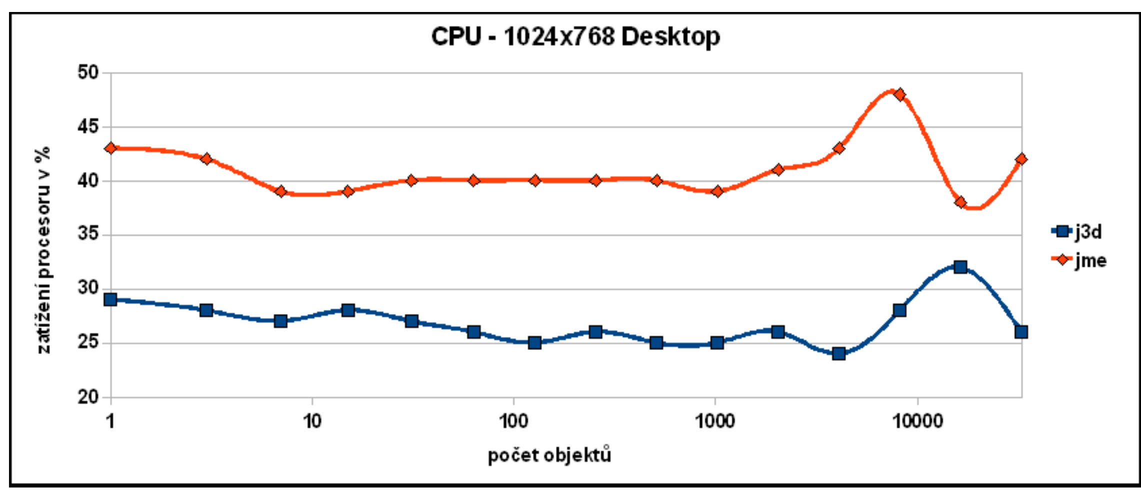
Obrázek 3.16: Zdvojování počtu objektů - Procesor 1024x768 Notebook

Při testování na sestavě Notebook se již situace změnila. Zatímco u rozlišení 800x600 aplikace s jME vytěžovala procesory průměrně na 41%, aplikace s Java3D API vykazovala zátěž pouze kolem 27%. Opět se zde ukazuje, že u vícejádrových procesorů je Java3D schopna lépe využívat potenciálu vícejádrového stroje.

I při zvýšení rozlišení na 1024x768 bodů dosahovala Java3D uspokojivějších výsledků než její konkurent, knihovna jME. Knihovna Java3D vytěžovala procesor průměrně pouze na 23%, kdežto jME na průměrně 43%. Obě knihovny se po celou dobu běhu testu držely s vytížením procesoru velmi blízko průměrné hodnotě, obě se odchýlily pouze při počtu objektů mezi 8191 a 16383. Zde vykazovaly lehce zvýšenou, popř. sníženou zátěž procesoru.



Obrázek 3.17: Zdvojování počtu objektů - Procesor 800x600 Desktop



Obrázek 3.18: Zdvojování počtu objektů - Procesor 1024x768 Desktop

Zhodnocení

Z naměřených hodnot vyplývá, že prostředí jME je schopné produkovat vyšší počet snímků za vteřinu a je také znatelně méně paměťově náročné. Naproti tomu knihovna Java3D dosahuje na vícejádrových procesorech nižšího zatížení.

V testu snímkovací frekvence docházelo u obou knihoven se zvyšujícím počtem objektů k postupnému snižování počtu vykreslovaných snímků. Během testu paměťové náročnosti naopak docházelo ke zvyšování potřeby prostoru pro alokaci vnitřních objektů. Z tohoto vybočuje měření zatížení procesoru, které se zdá být nezávislé na počtu objektů, které jsou ve scéně obsaženy.

3.6 Souhrn testů

Ve všech provedených testech dominovala knihovna jME v oblasti rychlosti vykreslování. Paměťová náročnost obou knihoven byla podobná do okamžiku, kdy scéna obsahovala více objektů. V tu chvíli přebrala iniciativu opět jME, zvlášť dobře je to patrné ve třetí sadě testů. Poslední kritérium, kterým byla zátěž procesoru, odhalilo rozdíly v závislosti na konfiguraci počítače. Na dvoujádrovém procesoru dosahovaly obě knihovny velmi podobných výsledků, na čtyřjádrovém ovšem zatěžovala Java3D procesor cca polovičně. Vezmeme-li ovšem v potaz, že zatížení procecosoru nebylo ovlivněno počtem objektů ve scéně, pak tato schopnost není tak důležitá.

4 Návrh aplikace

V této kapitole budou rozebrány požadavky na demonstrační aplikace, navrženy nejvhodnější technologie a popsány plánované součásti programu.

4.1 Grafické prostředí

Knihovna pro práci s 3D grafikou

Na základě testů provedených v předchozí kapitole byla pro práci s grafikou vybrána knihovna jME. Tato knihovna dosahuje vyšší vykreslovací frekvence a ke své práci potřebuje méně paměti, takže program bude použitelný i na slabších počítačích.

Pohyb po scéně

Je potřeba se rozhodnout, jakým způsobem bude možné se po scéně pohybovat. Jako nejlepší varianta se jeví možnost pohybování se pomocí myši a klávesnice, kdy klávesy slouží k pohybu dopředu, dozadu, nahoru, dolů a do stran, a kde pohyb myši určuje směr natočení pohledu.

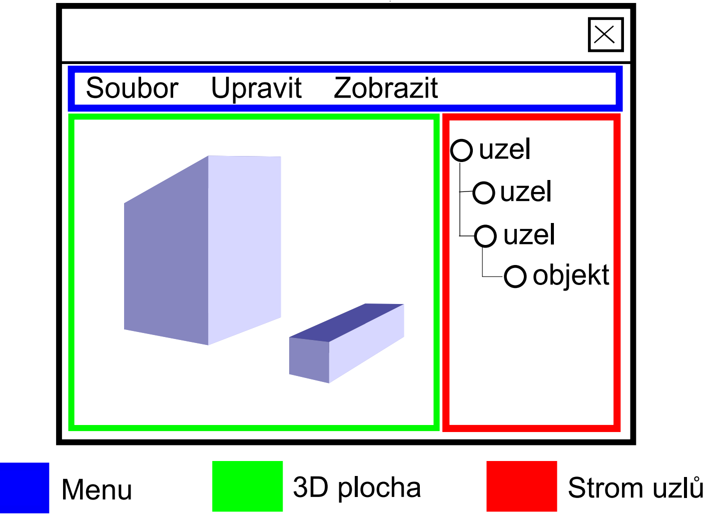
4.2 Uživatelské rozhraní

Aby byl program uživatelsky přívětivý, je vhodné ho koncipovat jako okenní aplikaci, která bude kromě samotné 3D plochy obsahovat i jiné prvky uživatelského rozhraní. Aplikace bude obsahovat menu, pomocí kterého bude možné načítat modely, zobrazovat skriptovací konzoli, popř. vyvolat různé dialogy s nastavením.

Aplikace tedy bude využívat knihovnu pro tvorbu okenních aplikací a její komponenty. Z důvodu své rozšířenosti a poskytovaných možností bude využit `Swing`, který je součástí J2SE. Existují dvě možnosti provázání komponent knihovny `Swing` s prostředím jME. Tyto možnosti byly diskutovány v kapitole zabývající se rozdíly mezi Java3D API a jME. Pro naši aplikaci je vhodnější vložení jME canvasu do okna vytvořeného pomocí `Swing`. Program tak bude vypadat uceleněji a profesionálněji.

4.3 Skriptovací konzole

Samotná skriptovací konzole může být v aplikaci zahrnuta několika způsoby. Může být zobrazena trvale na předem specifikovaném místě. V tomto případadě ovšem bude zbytečně zabírat místo, i když ji uživatel zrovna nebude potřebovat. Druhou možností je vyvolání dialogového okna, např. při stisku tlačítka, do kterého by bylo možné zadat požadovaný



Obrázek 4.1: Návrh uživatelského rozhraní

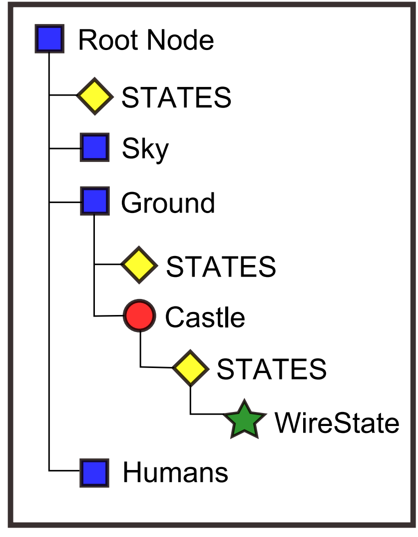
skript. Toto má ovšem nevýhodu v tom, že vyvolání takového okna znemožní práci s aplikací na pozadí do chvíle, kdy bude dialogovaé okno zrušeno/skryto. Poslední možností je skriptovací konzole jako samostatné okno, se kterým bude možno manipulovat nezávisle na aplikaci samostatné. Toto přináší nepříjemnost v podobě zobrazení konzole jako samostatného programu v panelu spuštěných aplikací, ovšem následná práce bude pro uživatele příjemnější. Jako nejlepší řešení se jeví možnost poslední, tj. nezávislé okno.

Dalším důležitým faktorem, se kterým je potřeba počítat již v návrhu aplikace, je způsob, jakým bude kód ze skriptovací konzole programem interpretován. I zde existuje několik možností - první je použití novinky v Javě 1.6, kterou je `ScriptingEngine`. Tato třída pomocí svých metod dovoluje interpretovat zadaný kód. Nevýhodou je, že používá JavaScriptovou konvenci, což může být pro mnoho uživatelů matoucí. Alternativou je použití nezávislé knihovny `BeanShell`. Ta pracuje na podobném principu jako `ScriptingEngine` ovšem s tím rozdílem, že umí vyhodnocovat programy psané se syntaxí jazyka Java [9]. Díky této vlastnosti je `BeanShell` vhodnější technologií pro použití v demonstrační aplikaci, protože je možné použít již napsané programy v Javě a ty lehce upravit.

4.4 Strom uzlů

Jelikož má mít vysledná aplikace demonstrační charakter, je vhodné uživateli umožnit vidět, jaké scéna obsahuje uzly a jak jsou na sobě závislé. Součástí programu tedy bude stromová struktura, která bude obsahovat všechny uzly a koncové objekty obsažené ve scéně. Toto jistě velmi ocení uživatelé, kteří s 3D grafikou teprve začínají a získají tak přehled o závislosti jednotlivých objektů ve scéně.

Uzly typu `Leaf` a `Branch` budou od sebe graficky odlišeny ikonou, která bude zobranena vedle názvu uzlu. Jako užitačné se jeví zobrazení určitých vlastností uzlů, např. jestli má na uzel vliv zdroj světla. Každý uzel tedy bude obsahovat poduzel, který bude zapouzdřovat nastavené vlastnosti.



Obrázek 4.2: Návrh stromu uzlů

4.5 Tvorba pluginů

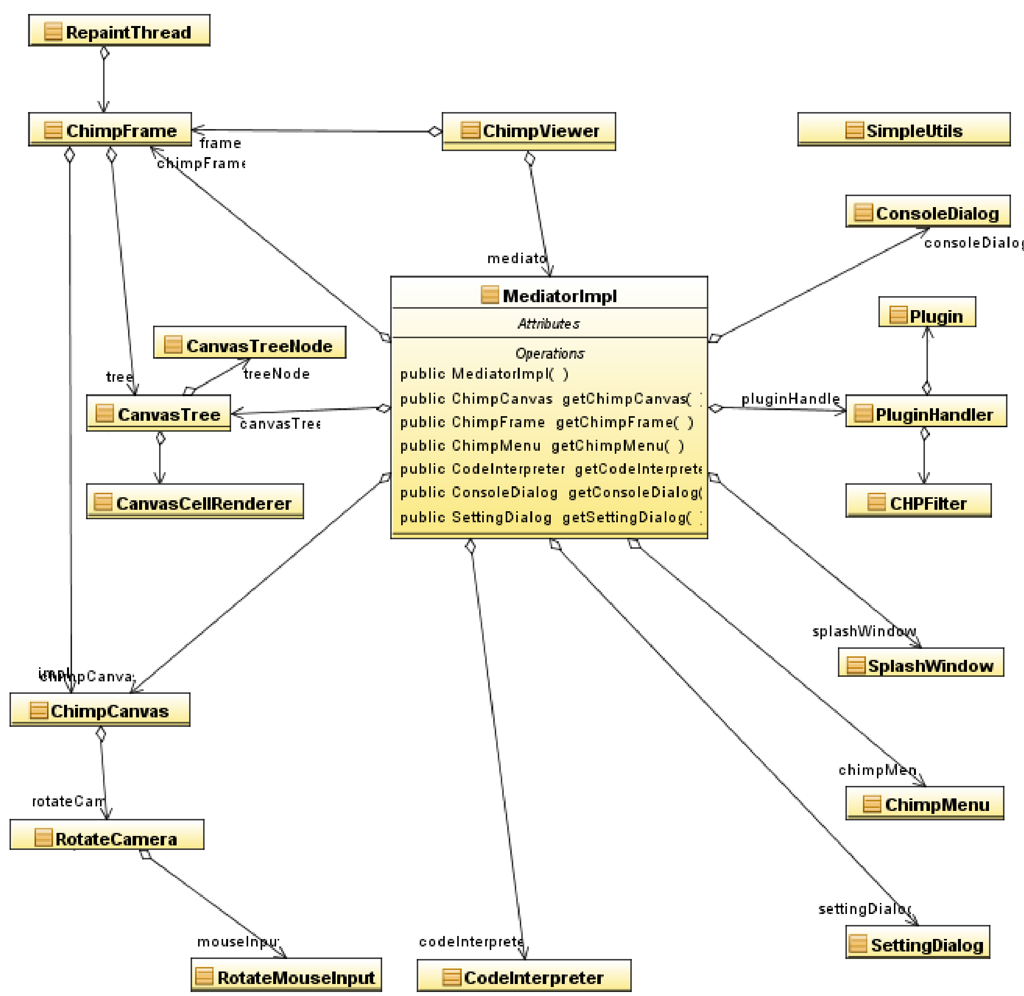
Zavedení knihovny, která dovede zpracovat zadaný zdrojový kód a ten následně interpretovat, otevírá možnost vytvoření módu pluginů. Program bude po spuštění prohledávat definovanou složku se soubory, které obsahují zdrojový kód. Ten bude načten a interpretován. Díky tomu bude možné ve spouštějící se aplikaci provést změny, které můžou ovlivnit grafické, ale i programové vlastnosti aplikace.

Zavedením tohoto rozšíření se aplikace stane pro uživatele atraktivnější, protože si budou moct sami psát rozšiřující pluginy, které do aplikace zavedou jimi požadavanou funkcionalitu.

5 Implementace

V rámci této kapitoly budou popsány implementační detaily a zajímavé funkce výsledné aplikace, stejně jako popis použitých návrhových vzorů.

Uspořádání a vazby jednotlivých tříd jsou patrné z následujícího obrázku



Obrázek 5.1: Diagram tříd

5.1 Mediator

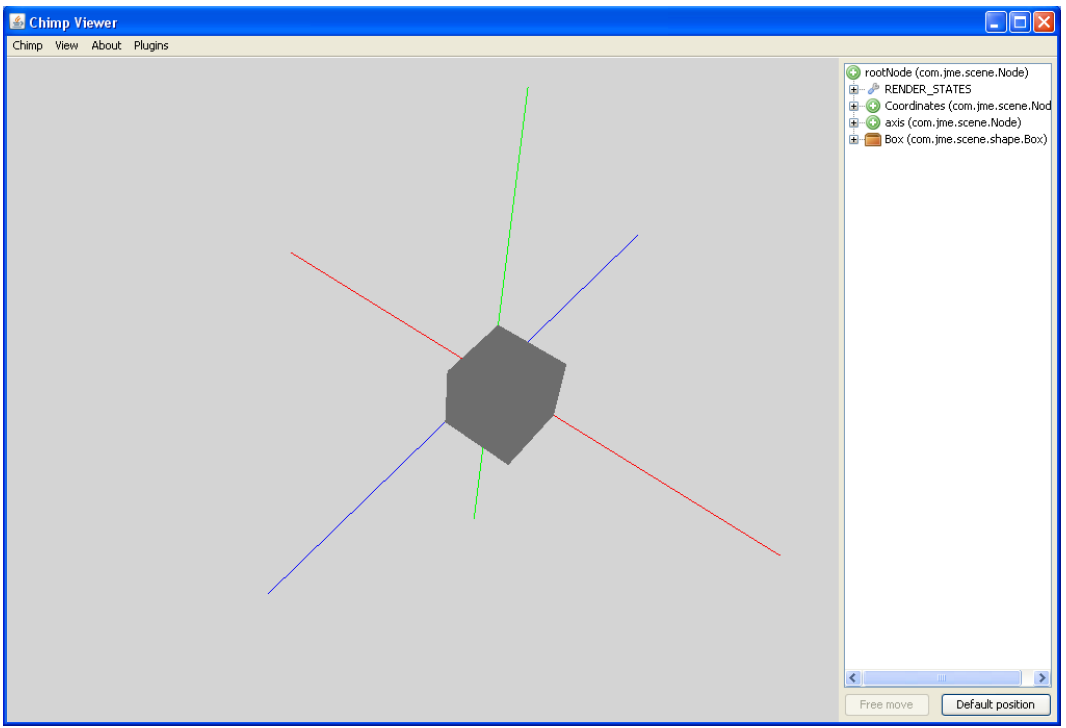
Z důvodu potřeby komunikace mnoha různých tříd mezi sebou byl ve výsledné aplikaci použit návrhový vzor **Mediator**.

Tento návrhový vzor se používá v situacích, kdy se v aplikaci vyskytuje velký počet tříd, které potřebují komunikovat s jinými třídami. Bez použití tohoto vzoru by každá třída musela uchovávat reference na všechny třídy, s kterými komunikuje, a z aplikace by se stala velmi nepřehlednou.

Mediator přichází s řešením, kdy veškerá logika komunikace mezi třídami je zapouzdřena do jedné konkrétní třídy `Mediator`. Tato třída uchovává reference na všechny třídy, které se v aplikaci účastní komunikace, naproti tomu komunikující třídy si drží pouze refenci na objekt `Mediator`. Při potřebě komunikace s okolím tato třída zavolá příslušnou metodu `Mediatoru`, která pak zajistí komunikaci mezi zbývajícími komponentami [6].

Použití návrhového vzoru značně usnadňuje přidávání nových tříd do programu a zvyšuje přehlednost programu. Tento vzor se nejčastěji používá ve variantě, kdy jednotlivé třídy neuchovávají referenci přímo na `Mediator`, ale na abstraktní třídu, popř. v javě na rozhraní `MediatorInterface`, které obsahuje metody potřebné pro fukčnost této třídy. Samotný `Mediator` toto rozhraní implementuje, čímž dává možnost vytvořit si vlastní implementaci `MediatorInteface` [2].

V aplikaci je rozhraní prezentované třídou `Mediator`, která určuje metody vyžadované komunikujícími komponentami, a její implementací `MediatorImpl`, která těmto třídám nastavuje chování.



Obrázek 5.2: Vzhled aplikace ChimpViewer

5.2 Okenní aplikace

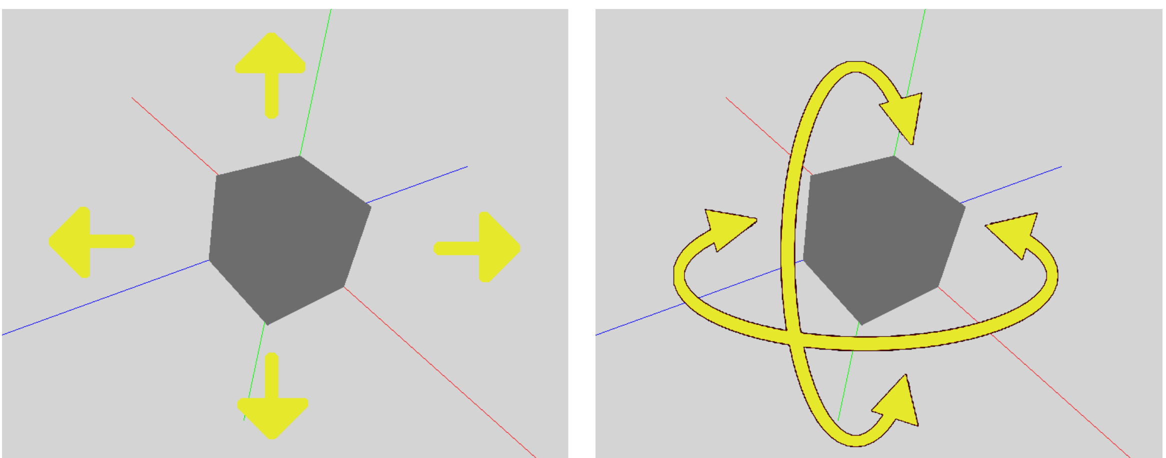
Jak již bylo zmíněno v kapitole věnující se návrhu, aplikace využívá knihovnu `Swing`, do které je vložen jME canvas. Konkrétně byla vytvořená vlastní implementace třídy `JFrame`, která obsahuje další z komponent grafického rozhraní jako je např. `CanvasTree` reprezentující strom uzlů nebo `ChimpMenu` definující menu aplikace. Třída obsahuje i samotný 3D canvas. Jelikož se jedná třídu, která potřebuje komunikovat s ostatními objekty, uchovává si referenci na `Mediator`, na kterou vznáší všechny požadavky na komunikaci.

5.3 3D plocha

3D plocha je vložena tak, že do okenní aplikace je přidán `Canvas` z balík `awt`. Ten je následně přetypován na objekt typu `JMECanvas` a je mu přiřazen pro potřeby programu vytvořený Implementor scény. Od tohoto okamžiku se původní `Canvas` tváří jako canvas knihovny jME a je s ním možno provádět veškeré úpravy. Samotný implementor scény se nachází v třídě `ChimpCanvas`, která rozšiřuje třídu `SimpleCanvasImpl`. Vytvořený canvas na rozdíl od svého rodiče poskytuje svůj kořenový uzel scény metodou `getRootNode()`. Je to z důvodu spolupráce se skriptovací konzolí, kdy je potřeba s tímto uzlem manipulovat.

Pohyb po scéně

Aplikace umožňuje 2 způsoby pohybu po scéně. První se nazývá **FreeMoveMode**, kdy je možno se po scéně pohybovat pomocí myši způsobem známým z first person akčních her. Druhou možností je tzv. **RotateMode**, kdy je pozornost přesunuta na určitý objekt ve scéně, kolem kterého začne kamera rotovat.



Obrázek 5.3: Směr pohybu ve FreeMoveMode a RotateMode

V samotném programu to vypadá tak, že uživatel se volně pohybuje po scéně pomocí FreeMoveMode a když si pak chce prohlédnout nějaký objekt, přepne se do RotateMode. Při přepnutí se pozice pozorovatele nezmění, směr pohledu je ovšem přesunut do středu objektu. Následně při stisku tlačítek klávesnici či pohybem myší dochází k rotaci kolem

objektu v požadovaných směrech. Pohled lze oddalovat a přibližovat. Kdykoliv je možné RotateMode zrušit a vrátit se do FreeMoveMode.

Je také možné vyvolat akci, která kameru přesune na výchozí pozici a nastaví výchozí směr pohledu.

Zobrazení os a sítě

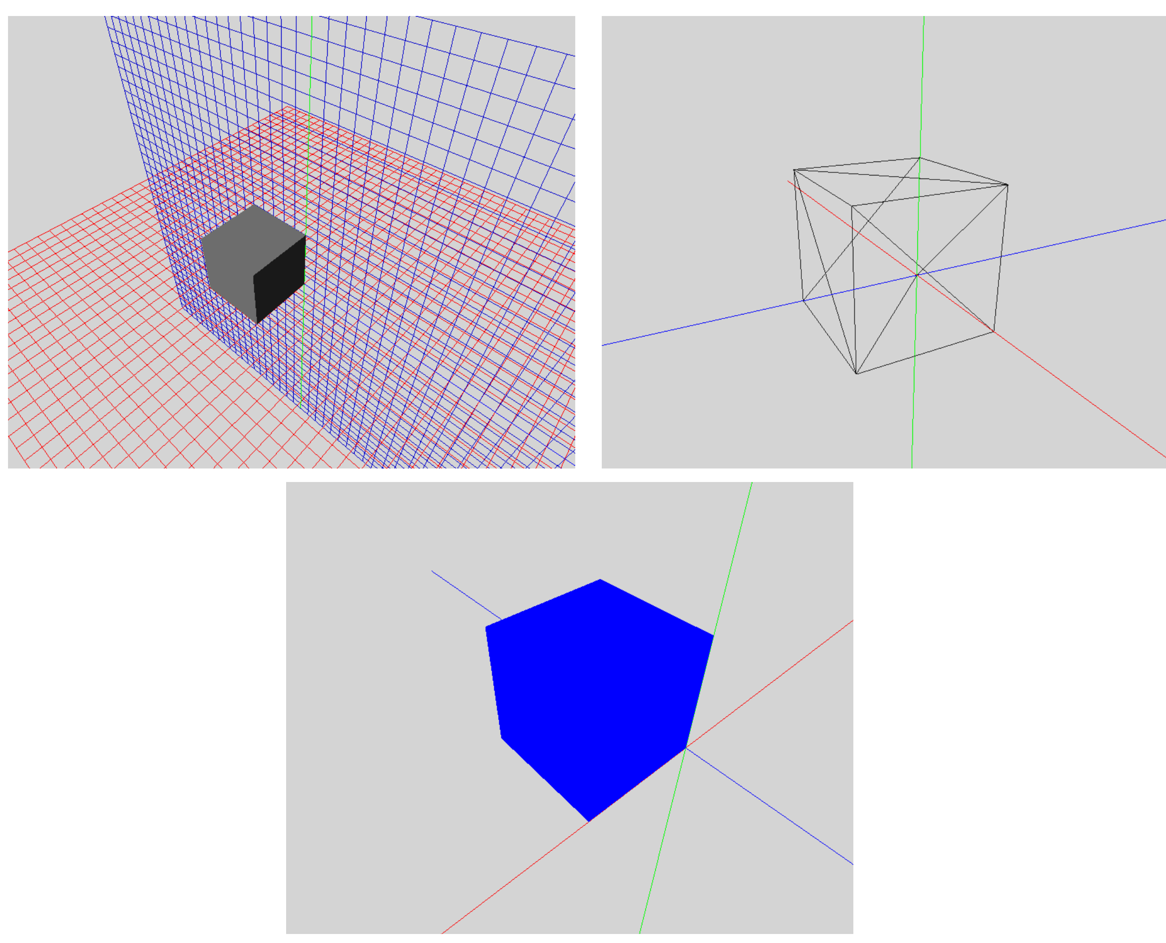
Pro snadnější orientaci v prostoru byly vytvořeny směrové osy `x, y, z`, které lze zobrazit/skrýt pomocí klávesových zkratek.

Scéna také obsahuje síť linek v rovinách `xy, xz, yz`. Ty je možno zobrazit buď z dialogu nebo pomocí klávesové zkratky.

Tyto rozšíření značně usnadňují orientaci v prostoru.

Módy zobrazení

Pro demonstraci funkcí skrytých pod povrchem grafických prostředí byly do programu přidány možnosti přepínání mezi několika módy zobrazení.



Obrázek 5.4: Zapnutá síť linek, zapnutý drátový mód a vypnuté světlo

Prvním je zobrazení drátového modelu. Při tomto zobrazení jsou odstraněny textury s plochami objektů a jsou zobrazeny pouze vrcholy a jejich spojnice. Tento mód se dá jednoduše vypnout/zapnout pomocí klávesové zkratky. Jeho užitečnost tkví ve schopnosti vidět reálný tvar těles, který může být použitím různých grafických technik opticky změněn.

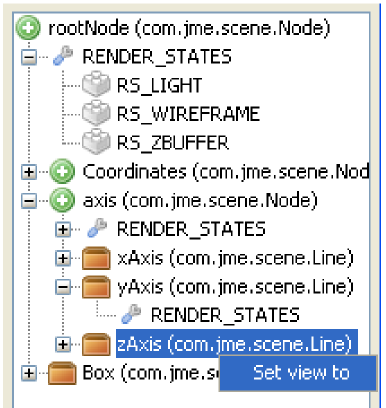
Druhou možností je vypnutí osvětlení. V tomto případě dojde k vypnutí všech zdrojů světla ve scéně a všechny objekty jsou zobrazeny pouze ve své neosvětlené podobě. Mód lze použít pro demonstaci rozdílů ve vzhledu scény se světlem a bez něj. Jeho vypnutí/zapnutí je vyvoláno pomocí klávesové zkratky.

Načítání modelů

Program umí zobrazit modely ve formátech **obj**, **3ds**, **md2**, **md3**, **ms3d** a **x3d**. Načtení modelu do scény je možné buď přes skriptovací konzoli nebo pomocí načítačího dialogu z menu. Při tomto druhu načítání není ovšem možné ovlivnit pozici, na kterou bude objekt umístěn, ta je vždy `[0,0,0]`.

5.4 Strom uzlů

Zobrazuje všechny uzly a tělesa umístěná v grafu scény. Vždy obsahuje minimálně uzel `rootNode`, který reprezentuje kořenový uzel. Ten je ve scéně obsažen vždy. Každý uzel v prostředí jME může mít nastavených až 17 vlastností typu `RenderState`, které určují např. průhlednost uzlu, použitý materiál, osvětlení, zda má aktivovaný drátový mód apod. Tyto vlastnosti jsou ve stromu uzlů také zobrazeny.



Obrázek 5.5: Strom uzlů

Nad každým uzlem stromu, který reprezentuje objekt ve scéně, je možné vyvolat kontextové menu, které nad tímto objektem aktivuje RotationMode. Strom uzlů s 3D canvasem komunikuje přes rozhraní `Mediator`, neuchovává tedy na něj žádnou referenci.

Jednotlivé uzly vyskytující se v grafu scény jsou popsány v následující tabulce

Značka	Název	Popis
	Uzel Branch	Uzel, ke kterému je možno připojit další uzly scény
	Object Leaf	Viditelný objekt scény, popř. neviditelný objekt (světlo), do kterého není možné přidávat další uzly
	Vlastnosti uzlu	Tento uzel zapouzdřuje RenderState, které má daný uzel nastaven
	RenderState enabled	Reprezentuje aktivní RenderState
	RenderState disabled	Neaktivní RenderState
	Neznámý typ uzlu	Uzel, který není definován

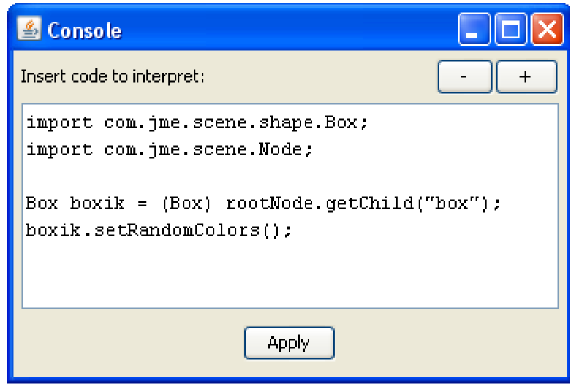
Tabulka 5.1: Tabulka uzlů ve stromu

Díky tomu, že ve stromu jsou zobrazené i `RenderState`, které jsou pro daný uzel vytvořeny, bylo umožněno tyto stavy vypínat a zapínat. Při vyvolání kontextového menu nad libovolným `RenderState` je nabídnuto povolení, popř. zakázání promítnutí tohoto stavu do scény. Povolené a zakázané stavy lze jednoduše rozeznat pomocí ikon, které jsou popsány v tabulce výše.

5.5 Skriptovací konzole

Konzole je implementovaná do samostatného okna typu `JFrame`. Výhodou tohoto řešení je možnost manipulovat s konzolí nezávisle na práci s hlavní okenní aplikací. Při ukončení hlavní aplikace dojde i ke zrušení tohoto okna.

Vlastní text skriptu se vkládá do pole typu `JTextArea`. Po potvrzení zadaného kódu je tento interpretován. Konzole obsahuje jednoduchou historii zadaných skriptů, kterou je možno procházet a dříve vložené skripty přepisovat.



Obrázek 5.6: Skriptovací konzole

Aby bylo možno upravovat aktuální scénu a případně celou aplikaci, bylo nutné definovat přípojné body, které jsou přístupné ze skriptu a jejichž změna se provede v samotném programu. Tyto přípojné body jsou 2 - prvním je proměnná jménem `rootNode`. Zde se jedná o referenci na kořenový uzel scény 3D plochy. Druhým přípojným bodem je proměnná `application`. Ta odkazuje na `Mediator`. Proč zrovna na `Mediator`? Je to z důvodu, že `Mediator` obsahuje reference na všechny komunikující třídy a není tedy problém si z něj jakoukoliv třídu vytáhnout a s tou pak manipulovat.

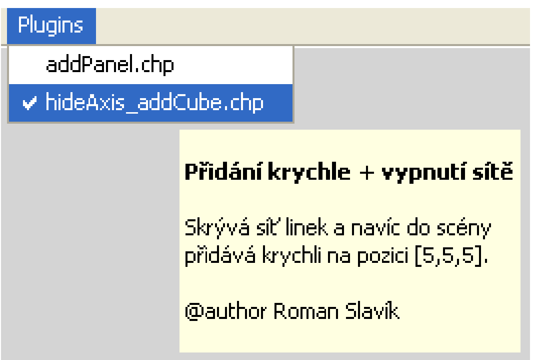
Další důležitou vlastností použití skriptů je nutnost přidat importy použitých tříd na začátek skriptu, např. `import com.jme.scene.shape.Box;`. Jinou možností je uvádět v příkazové části skriptu celou cestu k použité třídě, tento přístup je však nepraktický.

Příklady skriptů manipulujících s kořenovým uzlem a s mediatorem jsou uvedeny v příloze.

5.6 Pluginy

Systém pluginy dává užitateli možnost rozšířit program o původně neimplementované funkce, popř. změnit vzhled aplikace podle vlastního uvážení.

O logiku spojenou s pluginy se stará třída `PluginHandler`. Ta během spouštění vyhledá složku **plugins**, která se nachází ve stejném adresáři jako aplikace. Pokud je nalezena, prochází se všechny soubory typu **.chp**, jež by měly obsahovat skripty. Po načtení souboru je sekvence instrukcí předána `Mediatoru`, který zajistí vykonání zadaného kódu a jeho promítnutí do spouštěné aplikace.



Obrázek 5.7: Menu s popisem pluginu

Pro správnost funkce pluginů je potřeba dodržet zásady použití definované v části o skriptech. Tj. nutnost importů a propojení s aplikací pomocí přípojných bodů.

Pluginy je také možno vypínat. Pokud je plugin vypnut, při příštím spuštění aplikace nebude vykonán. Že je plugin zapnutý se pozná podle tak, že je vedle jeho jména v menu přítomna značka. Plugin musí začínat sekvencí

```
/* chimpEnabled=true */
```

Ta značí, je daný plugin aktivní a při spuštění aplikace se má interpretovat. Pokud dojde k jeho vypnutí, je `true` nahrazeno za `false`. Lze tedy aktivovat/vypnout plugin bez

nutnosti spouštění aplikace pouhým přepsáním souboru. Jestliže plugin řádek tento řádek neobsahuje, pak je do něj přidán s hodnotou nastavenou na `true`.

Pro snadné zacházení lze k pluginu přidat nápovědu, která se zobrazí při najetí myši na jeho jméno v menu. Vkládá se do souboru s pluginem před začátek samotného kódu mezi `/**` a `*/`. Nápověda akceptuje html tagy pro formátování textu.

Příklad komplexního pluginu je uveden v příloze.

6 Závěr

Zhodnocení

Cílem práce bylo porovnat knihovny Java3D a jME pro práci s trojrozměrnou grafikou v jazyce Java. Knihovny byly porovnávány několika kritérii, v nichž lépe dopadla knihovna jME. Ta byla následně použita jako základ demonstrační aplikace pro práci s 3D grafikou. Tato aplikace za pomoci skriptovací konzole umožňuje teoreticky jakékoliv úpravy scény pomocí vložených skriptů.

Práce na tomto zadání byla zábavná a zároveň mi dovolila proniknout hlouběji do tajů 3D grafiky. Výsledná aplikace je navíc vhodná pro vysvětlování principů počítačové grafiky novým zájemcům.

Při navrhování implementačních detailů mi velmi pomohla kniha Java 5 od Ivor Hortona [8]. Pro řešení otázek týkajících se efektivních řešení mi zase byla nápomocna Effective Java, Second edition od Joshua Blocha [4].

Rozšíření stávající práce

Z hlediska teoretického může být práce rozšířena o porovnání s dalšími knihovnami pro práci s 3D grafikou v javě, např. JPCT.

V programové části stojí za zvážení možnost nastudování detailů použití `MBean` pro monitorování výkonu a vytvoření vlastní implementace s následným přidáním do demonstrační aplikace. Tím by bylo možno získávat v reálném čase informace o výkonu.

Jiným velmi zajímavým rozšířením může být možnost přepínání mezi knihovnami Java3D a jME, kdy by si uživatel mohl zvolit, kterou z nich chce při aktuálním spuštění použít.

Velký prostor také nabízí pluginy. Případný zájemce by mohl vytvořit několik komplexních skriptů, které by přidaly do aplikace novou funkcionalitu.

Literatura

[1] Java3D. [online], Poslední modifikace: 13.března 2009, [cit. 2.května 2009], Wikipedia.
URL http://en.wikipedia.org/wiki/Java_3D

[2] Design patterns. [online], Poslední modifikace: 19. září 2008, [cit. 2.května 2009].
URL http://www.javacamp.org/designPattern/

[3] Java SE Monitoring and Management Guide. [online], Poslední modifikace: 1.sprna 2008, [cit. 2.května 2009].
URL http://java.sun.com/javase/6/docs/technotes/guides/management/

[4] Bloch, J.: *Effective Java* ^{TM}. Prentice Hall, second edition vydání, 2008, ISBN 978-0-321-35668-0.

[5] Davison, A.: *Programování dokonalých her v Javě*. Computer Press, a.s., 2006, ISBN 80-7226-944-5.

[6] Dvořák, M.: Návrhové vzory. [online], Poslední modifikace: 2. února 2008, [cit. 2.května 2009].
URL http://objekty.vse.cz/Objekty/Vzory

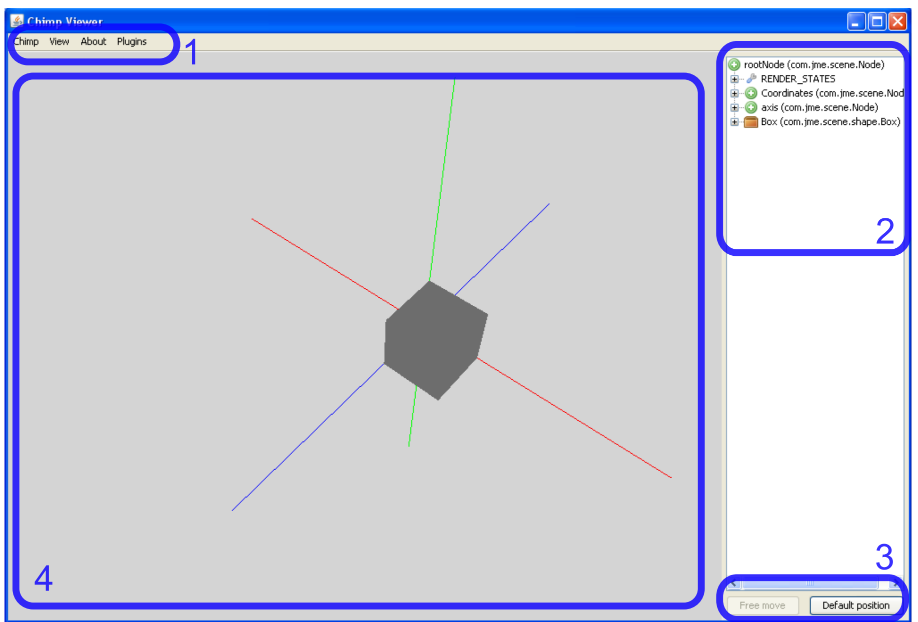
[7] Foster, G.: Understanding and implementing scene graph. [online], [cit. 2.května 2009].
URL http://www.gamedev.net/reference/programming/features/scenegraph/

[8] Horton, I.: *Java 5*. Neocortex, spol. s.r.o., 2005, ISBN 80-86330-12-5.

[9] Niemeyer, P.: Beanshell - Introduction. [online], [cit. 2.května 2009].
URL http://www.beanshell.org/intro.html

[10] Powell, M.: Java Monkey Engine. [online], Poslední modifikace: 11. února 2009, [cit. 2.května 2009].
URL http://www.jmonkeyengine.com

A Manuál k ovládání aplikace



1 - Menu

Chimp	– >	**Open Model**	CTRL + O	načtení modelu pomocí dialogu
		Exit		ukončení aplikace
View	– >	**Show Tree**	CTRL + T	zobrazí/skryje strom uzlů
		Settings	CTRL + S	nastavení aplikace
		Console	CTRL + D	otevře skriptovací konzoli
About	– >	**Application**	CTRL + B	zobrazí okno s informacemi o aplikaci
Plugins				seznam pluginů

2 - Strom uzlů

Pravý klik na objekt - vyvolání kontextového menu
Dvojitý klik na uzel - zobrazení poduzlů

Volba Set View To z kontextového menu - nastavení RotateMode nad daným objektem

Volba Enable/Disable z kontextového menu - zapnutí/vypnutí vybraného RenderState

3 - Příkazy kamery

Free Move - vypnutí RotateMode, možnost volného pohybu kamery
Default Position - přesunutí kamery do výchozí pozice

4 - 3D plocha

L - zapnutí/vypnutí světla
O - zapnutí/vypnutí drátového modelu
J - zobrazení/skrytí os
I - zobrazení/skrytí sítě linek

FreeMode

LB myši + pohyb - otočení kamery daným směrem

W - pohyb dopředu
A - pohyb doleva
S - pohyb dozadu
D - pohyb doprava
Q - pohyb vzhůru
Z - pohyb dolů

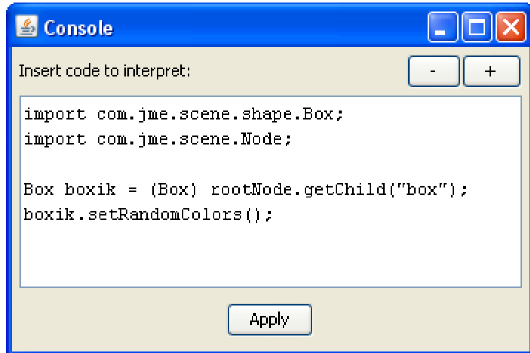
Směrové klávesy - otočení kamery daným směrem

RotateMode

LB myši + pohyb - otočení kamery kolem tělesa daným směrem
Kolečko myši - přiblížení/oddálení

W - otočení směrem nahoru kolem tělesa
A - otočení směrem doleva kolem tělesa
S - otočení směrem dolů kolem tělesa
D - otočení směrem doprava kolem tělesa

Klávesa nahoru - přiblížení k objektu
Klávesa dolů - oddálení od objektu



5 - Skriptovací konzole

Tlačítko -	ALT + UP	- předchozí skript
Tlačítko +	ALT + DOWN	- následující skript
Apply	ALT + ENTER	- vykonání skriptu

B Příklad skriptu

```
/**
 * Skript prida do sceny krychli a nastavi ji nahodne zbarveni
 *
 * @author Slavík
 * @version 1.0
 */

// nutne importy pouzitych trid
import com.jme.scene.shape.Box;
import com.jme.math.Vector3f;
import com.jme.renderer.Renderer;
import com.jme.scene.Node;
import com.jme.bounding.BoundingBox;
import com.jme.scene.state.LightState;
import com.jme.renderer.ColorRGBA;

// pridani krychle
Vector3f max = new Vector3f(5, 5, 5);
Vector3f min = new Vector3f(-5, -5, -5);
Box box = new Box("Box", min, max);
box.setModelBound(new BoundingBox());
box.updateModelBound();
box.setLocalTranslation(new Vector3f(0, 0, 0));
box.setRenderQueueMode(Renderer.QUEUE_SKIP);
box.setLightCombineMode(LightState.COMBINE_RECENT_ENABLED);
box.setRandomColors();
rootNode.attachChild(box);
```

C Příklad pluginu

```
/**
 * <b>Dialog pro zobrazení os</b>
 *
 * Tento plugin vytvoří jednoduché dialogové okno s možností volby
 * mezi zobrazením, nebo skrytím os
 *
 * @author Slavík
 * @version 1.0
 */

// je potreba dodat importy k pouzitym tridam
import chimpViewer.core.MediatorImpl;
import com.jme.renderer.ColorRGBA;
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Label;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JPanel;
import chimpViewer.jme.ChimpCanvas;
import com.jme.scene.Node;

// trida dialogoveho okna
public class BackgroundDialog extends JDialog implements ActionListener {
  private MediatorImpl application;
  public static final String YES_COMMAND = "yes";
  public static final String NO_COMMAND = "no";
```

```
public BackgroundDialog(JFrame parent, MediatorImpl mediator) {
  super(parent, "Nastavení os", true);
  this.application = mediator;
  init();
  pack();
}

// zobrazeni dialogu
public void display() {
  Dimension parentSize = application.getFrameSize();
  Point p = application.getFrameLocation();
  int width = getWidth();
  setLocation(p.x + (parentSize.width - width)/ 2,
              p.y + parentSize.height / 5);
  setVisible(true);
}

// inicializace komponent dialogu
private void init() {
  Container content = getContentPane();
  content.setLayout(new BorderLayout());

  JPanel innerContent = new JPanel();
  innerContent.setMinimumSize(new Dimension(200, 200));
  innerContent.setLayout(new BoxLayout(innerContent, BoxLayout.Y_AXIS));
  innerContent.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

  JPanel qContent = new JPanel();
  qContent.setLayout(new BoxLayout(qContent, BoxLayout.Y_AXIS));
  qContent.add(new Label("Vítejte v testovacím skriptu!"));
  qContent.add(new Label("Chcete zobrazit osy?"));

  innerContent.add(qContent);
  content.add(innerContent, BorderLayout.CENTER);

  JPanel buttonPane = new JPanel();
  buttonPane.setLayout(new BoxLayout(buttonPane, BoxLayout.X_AXIS));
  buttonPane.setBorder(BorderFactory.createEmptyBorder(30, 5, 5, 5));

  buttonPane.add(Box.createHorizontalGlue());
  JButton blackButton = new JButton("Ano");
  blackButton.addActionListener(this);
  blackButton.setActionCommand(YES_COMMAND);
  buttonPane.add(blackButton);

  buttonPane.add(Box.createHorizontalStrut(15));

  JButton blueButton = new JButton("Ne");
```

```
    blueButton.addActionListener(this);
    blueButton.setActionCommand(NO_COMMAND);
    buttonPane.add(blueButton);
    buttonPane.add(Box.createHorizontalGlue());

    content.add(buttonPane, BorderLayout.PAGE_END);

    setResizable(false);
    setModal(true);
  }

  // zachyceni akce - zobrazeni/nezobrazeni os
  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals(YES_COMMAND)) {
      Node root = application.needRootNode();
      application.getChimpCanvas().setAxisVisible(true);
    } else if (e.getActionCommand().equals(NO_COMMAND)) {
      application.getChimpCanvas().setAxisVisible(false);
    }
    setVisible(false);
  }
}

// vytvoreni okna, jeho pridani do okna aplikace, zobrazeni
BackgroundDialog bg = new BackgroundDialog(application.getChimpFrame(),
                                           application);
bg.display();
```