VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

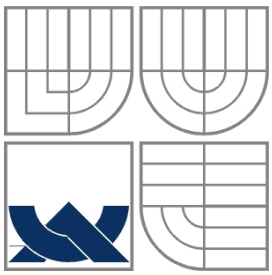
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

LETECKÝ SIMULÁTOR V JAVE

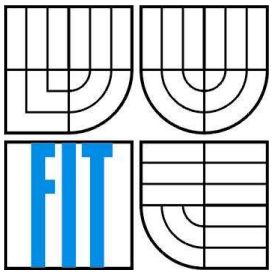
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARIÁN HACAJ
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

LETECKÝ SIMULÁTOR V JAVĚ

JAVA BASED FLIGHT SIMULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARIÁN HACAJ
AUTHOR

VEDOUCÍ PRÁCE — ING. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2009

Abstrakt

Práce popisuje (porovnává) dvě grafické knihovny Java Monkey Engine a Java 3D a rozdíly mezi nimi. Dále je v práci popsaná implementace jednoduchého leteckého simulátoru v knihovně Java Monkey Engine.

Abstract

Thesis describes (compares) two graphics libraries Java Monkey Engine and Java 3D and differences between them. It also describes the implementation of a simple airplane simulator in Java Monkey Engine library.

Klíčova slova

Java Monkey Engine, jME, Java 3D, OpenGL, DirectX, Java, 3D Grafika, Letecký simulátor.

Keywords

Java Monkey Engine, jME, Java 3D, OpenGL, DirectX, Java, 3D Graphics, Plane simulator.

Citace

Marián Hacaj: Letecký simulátor v Javě, bakalářská práce, Brno, FIT VUT v Brně, 2009

Letecký simulátor v Jave

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Aleša Láníka.
Další informace mi poskytli Martin Polák a Andrej Novosad.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Marián Hacaj
18.05.2009

Poďakovanie

Chcel by som sa poďakovať svojmu vedúcemu Ing. Alešovi Láníkovi za venovaný čas pri písaní tejto práce, ale aj za cenné rady, ktoré mi poskytol. Ďalej by som chcel poďakovať Martinovi Polákovi za poskytnutie modelu lietadla do aplikácie a Andrejovi Novosadovi za sprostredkovanie zvukov. V neposlednej rade chcem poďakovať aj svojej rodine, kamáratom, ktorí pri mne po celú dobu stáli a všetkým, ktorí aplikáciu testovali a uľahčovali mi tým prácu.

Obsah

Zoznam obrázkov

1 Úvod

Čím ďalej tým viac sa v informatickom svete rozširuje sekcia 3D grafiky. Či už v hrách, programoch, ale aj v medicíne alebo strojníctve. Táto práca poukazuje na dve možnosti, ako zmienenú 3D grafiku vytvárať i keď v pre grafiku netypickom jazyku – Jave.

Prvou voľbou je Java 3D vytváraná priamo spoločnosťou Sun Microsystems. Jej výhody sú zrejmé vďaka tomu, že je vytváraná tou istou firmou ako samotná Java (podpora, kompatibilita...), ale na druhej strane sa už Java 3D veľmi nevyvíja, čo ju robí pomalou a na vzhľad nepeknou.

Druhým riešením je viac perspektívna open source knižnica Java Monkey Engine (jME), ktorá, keďže je open source, je tvorená prevažne obyčajnými ľuďmi zo zábavy. Výhodou je, že sa neustále vyvíja, ďalej jej rýchlosť a takisto podpora na oficiálnom fóre, kde prispievajú prevažne programátori tejto knižnice.

V práci sú stručne tieto knižnice a ich časti (graf scény, transformácie, načítanie modelov...) popísané a na konci prvej časti sú vykonané testy rýchlosti a záťaže pamäte rovnakých aplikácií pre obe knižnice, aby bolo možné porovnať vzhľad a výkon jednotlivých knižníc.

V druhej časti práce je detailne popísaná implementácia jednoduchého leteckého simulátora v knižnici Java Monkey Engine, na ktorom sú ukázané všetky hlavné funkcie (transformácie, ovetlenie, terén...) 3D grafickej knižnice.

Úvodné kapitoly č. 2 až č. 4 popisujú jemne úvod do Javy, výhody a nevýhody oboch grafických knižíc a samostatnej knižnice Java, a všeobecnú definícu grafu scény. V nasledujúcej kapitole – č. 5 je detailne popísaná knižnica Java 3D podobne, ako jME v kapitole č. 6. Posledná kapitola č. 7 má za úlohu čitateľovi ukázať príklad využitia grafickej knižnice, konkrétne jME, v praktickom projekte – počítačovej hre.

2 Java

Java je objektovo orientovaný programovací jazyk. Je vyvíjaný spoločnosťou Sun Microsystems. Jeho syntax pochádza zo známych jazykov C a C++. Zdrojové programy sa neprekladajú do strojového kódu, ako je zvykom pri iných programovacích jazykoch, ale do akéhosi medzistupňa, tzv. „byte-code“, ktorý nie je závislý na konkrétnej platforme, pretože beží na virtuálnom stroji (Java Virtual Machine – JVM).

Java sa rozdeľuje do niekoľkých kategórií:

1. J2ME (Micro Edition) - pre mobilné telefóny a malé zariadenia
2. J2SE (Standard Edition) - typická inštalácia Javy pre domáce počítače
3. J2EE (Enterprise Edition) - používaná pre webové servery
4. Java Card - pre implementáciu do inteligentných čipových kariet (ako napr. SIM karta do mobilného telefónu)
5. niektoré ďalšie, používané na špecifické úlohy

V roku 1991 spoločnosť SUN Microsystems odštartovala tvz. Green project, ktorého cieľom malo byť vytvorenie programovacieho jazyka pre spotrebnú elektroniku. James Gosling ako jeden z hlavných inžinierov tak vytvoril jazyk Oak, ktorý vychádzal zo syntaxe C a C++.

Oak ako programovací jazyk spĺňal podmienku, aby bolo možné program napísať, skompilovať a spustiť na rôznych plaformách bez opätovnej rekompilácie, ktorá bola potrebná v prípade jazyka C/C++. Počas vývoja jazyka sa objavil drobný problém s názvom, kedy členovia tímu zistili, že programovací jazyk Oak už existuje, a preto sa zvolilo náhradné meno, ktoré svet pozná dodnes – Java (zaujímavosťou je, že je to podľa kávy, ktorú pili v bufete pri vymýšľaní mena, preto má Java logo kávy [10]).

I keď bola Java pôvodne vyvíjaná s cieľom použitia v spotrebnej elektronike, v ktorej Sun videl obrovský potenciál, jej cesta sa celkom nečakane obrátila na web. V dobe rozmachu internetu boli stránky viacmenej statické a dynamika bola riešená zložitým spôsobom cez CGI skripty vykonávajúce sa na strane servera.

Keďže vývojári webových stránok požadovali možnosť vyššej interaktivity, Gosling so svojím tímom jazyk upravil tak, aby mohol bežať v prostredí webového prehliadača a zabezpečoval potrebnú a požadovanú interaktivitu. Tak sa zrodili applety, ktoré dali základ ďalšiemu úspešnému ťaženiu Javy.

13. novembra 2006 Sun Microsystems uvoľnil veľkú časť zdrojového kódu Javy pod GNU General Public License (GPLv2). 8. mája 2007 uvoľnil zvyšnú časť kódov, ku ktorým mal na to práva. Reimplementácia zvyšných častí pokračuje. [8]

3 Java 3D a jME

Java 3D [1][2][4][5][6][7] a jME (Java Monkey Engine)[3] sú klientské aplikačné programové rozhrania (API) vyvíjané v SUN Microsystems (Java 3D) a nezávislým vývojárom Markom Powellom a jeho tímom (jME) pre vykresľovanie 3D grafiky používaním jazyka JAVA. Inými klientskými API sú napr. dobre známe AWT (Absract Windows Toolkit) [10] a novšie JFC/Swing (Java Foundation Class) [10], ktoré su obidve javovské knižnice pre programovanie grafických užívateľských rozhraní (GUI).

Vytváranie 3D grafiky na počítači je dlhoročný problém, reprezentovaný dlhou históriou a veľkým počtom algoritmov. SUN Microsystems nie je hlavným hráčom na trhu 3D vývoja, ale napriek tomu jeho hardware dlho podporuje vykresľovanie v 3D. Dominanté postavenie na trhu má dnes jedna z najznámejších (ak nie vôbec) knižníc – OpenGL, vyvíjaná v Silicon Graphics (SGI). OpenGL knižnica bola navrhnutá ako multi-platformová architekrúra podporovaná väčšinou operačných systémov, grafických kariet a aplikácií. OpenGL API je napísané v programovacom jazyku C a bohužiaľ kvôli tomu ho nie je možné volať priamo z Javy. Napriek tomu existujú nezávislé open-source projekty, ktoré sa snažia o volanie OpenGL funkcií z Javy a prekladajú ich do akéhosi natívneho kódu. Jedno z najznámejších a najpopulárnejších takýchto API je GL4Java.

Java 3D sa spolieha na zmienené OpenGL alebo na grafickú knižnicu DirectX od spoločnosti Microsoft pri vykreslení 3D grafiky, zatiaľ čo definícia scény, rozmiestnenie objektov a logika aplikácie ostáva v Java kóde. Keď SUN Microsystems vyvíjal API Java 3D, nechcel ním nahradiť populárne OpenGL v jazyku C. Chcel len ukázať na možnosti Javy a objektovo orientovaného programovania (OOP) oproti procedurálnemu programovaniu, kde 3D scéna pozostáva z čiar, polygónov, farieb atď., zatiaľ čo v JAVE pozostáva z objektov, čo je pre človeka určite prirodzenejšie. jME taktiež využiva OpenGL, používa však medzistupeň a to knič nicu LWJGL [3] alebo JOGL [3] [2].

3.1 Výhody Javy 3D a jME

Prvou a najdôležitejšou výhodou je to, že je celý program napísaný priamo v jazyku Java. Je preto pre programátorov veľmi atraktívne, že môžu napísať svoj program v tak prenositeľnom jazyku, ako je Java. SUN Microsystem sa drží hesla „Write-Once-Run-Anywhere“ (v preklade: Napíš raz, pusti všade). Sun samozrejme podporuje dnešné najpoužívanejšie systémy ako napr.: Microsoft Windows, Sun Solaris, Macintosh OS X a Linux.

Java 3D API a jME majú veľa čo ponúknuť vývojárovi. Dovoľujú mu opísať scénu pomocou primitívnych elementárnych operácií ako napr. jednoduché transformácie, rotácie, ďalej operácie na objekty ako používanie rôznych materiálov, svetiel atď. Toto všetko robí kód oveľa prehľadnejší

a čitateľnejší. Java 3D aj jME používajú vysokú abstrakciu opísania scény a to tzv. graf scény[9] (Scene graph), ktorý umožňuje scénu ľahko opísať, transformovať atď [2].

3.2 Nevýhody Javy 3D a jME

Pre niektorých programátorov (hlavne OpenGL – programátorov) sú niektoré možnosti OpenGL v JAVE buď veľmi ťažko implementovateľné, alebo dokonca nemožné. Nie je to avšak až také kritické, ako by sa mohlo zdať a viac-menej je to zanedbateľný problém.

Väčším problémom je Java Virtual Machine (JVM) a Grabage collector (GC). Java, Java3d a jME svojím kódom vytvárajú objekty. Tieto objekty sa potenciálne všetky môžu stať odpadom a garbage collector ich následne odstráni. Garbage collector ale funguje tak, že tento odpad zbiera, ako sa mu zachce, a tak pri vykreslení nejakej zložitej scény môže prísť k fatálnemu spomaleniu vykresľovania kvôli tomu, že GC sa rozhodol poodstraňovať nepoužívané objekty. Avšak tento problém sa stále rieši zdokonaľovaním garbage collectoru a, samozrejme, zlepšovaním hardwaru počítačov, ktorý v dnešnej dobe ide dopredu svetelnou rýchlosťou.

Programy napísané v Jave bežia priamo na JVM, takže nemajú prístup priamo na hardware, čo sa veľmi odráža na rýchlosti hlavne pri 3D aplikáciách, ktoré by mali využívať priamo príkazy grafickej karty, avšak to nie je kvôli JVM možné.

Java API môže byť celkom ťažké distribuovať medzi koncových užívateľov. Väčšina totiž používa operačný systém Microsoft Windows a ten vo svojej predvolenej inštalácii Javu neobsahuje, takže je ju potrebné stiahnuť a nainštalovať, čo nemusí byť pre bežného užívateľa ľahké. Pokiaľ programátor nepripojí knižnicu Java 3D, respektíve jME do výslednej aplikácie, musí si užívateľ stiahnuť a nainštalovať aj jednu z týchto dvoch knižníc [2].

4 Graf scény

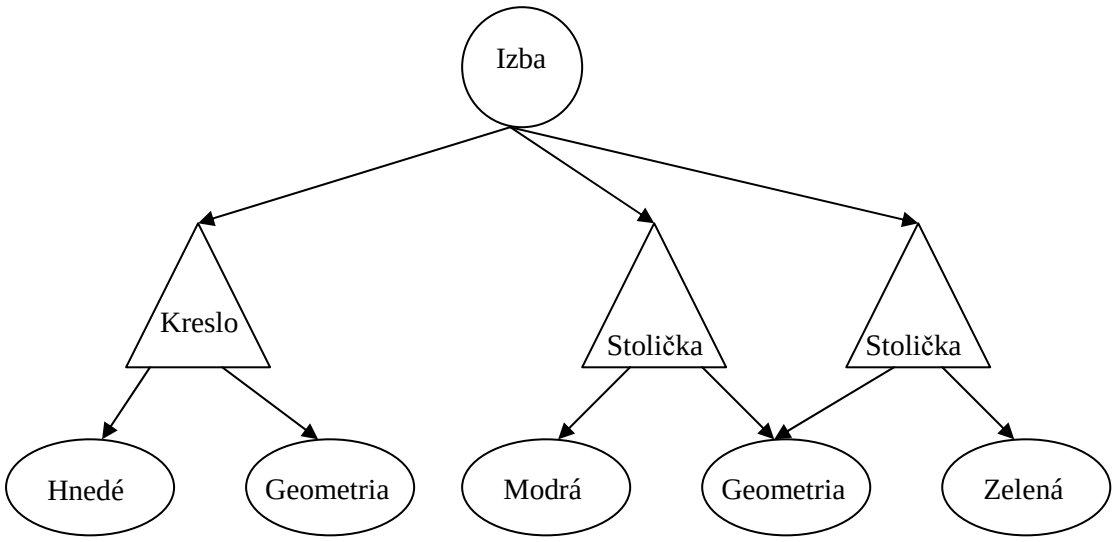
Graf scény je hlavná dátová štruktúra používaná hlavne vektorovo založenými grafickými aplikáciami a počítačovými hrami. Príkladnými programami môžu byť napr. AutoCAD, Adobe Illustrator, Acrobat 3D, OpenSceneGraph alebo CorelDRAW.

Graf scény je štruktúra, ktorá zostavuje logickú a často (ale nie nevyhnutne) priestorovú reprezentáciu grafu. Definícia grafu scény je sporná, pretože programátori, ktorí implementujú graf scény v aplikáciach a čiastočne v počitačových hrách, si berú základné prinípy a prispôsobujú ich k svojim aplikáciám. To znamená, že neexistuje jasné pravidlo, ako by mal graf scény vyzerať.

Graf scény je vlastne zoskupenie uzlov v grafe alebo v stromovej štruktúre. Uzol môže mať veľa detí, ale často len jedného rodiča, ktorý sa prejaví na všetkých jeho deťoch; operácia aplikovaná na skupinu sa automaticky prevedie na všetkych členov. V mnohých programoch združujúcich geometrické transformačné matice na každom stupni skupiny a zreťazenie matíc dokopy, je efektívna a prirodzená cesta k vykonaniu takýchto operácií. Hlavný rys je možnosť zoskupiť súvisiace objekty/tvary do zložky objektov, s ktorou sa môže hýbať, transformovať, označiť, atď., najľahšie, ako je možné - ako jeden objekt.

Tiež sa stáva v niektorých grafoch scény, že uzol môže mať vzťah k inému uzlu vrátane samého seba, alebo aspoň rozšírenie, ktoré odkazuje na iný uzol.

Graf scény prináša dve hlavné výhody: zjednodušuje programovanie 3D aplikácií a urýchľuje výsledný kód. Skrýva totiž prvky 3D grafiky na nízkej úrovni, čím programátorovi umožňuje ľahšie organizovať 3D scénu. Graf scény naviac podporuje radu komplexných grafických prkov [9].



Obrázok 1 – Príklad grafu scény [1]

5 Java 3D

5.1 Graf scény – Java 3D

Knižnica Java 3D používa graf scény k organizácii a riadeniu 3D aplikácií. Základná grafická pipeline je skrytá, nahradzuje ju stromová štruktúra pozostávajúca z uzlov, ktoré reprezentujú 3D-modely, svetlá, pozadie, kameru a radu ďalších prvkov 3D scény – graf scény.

Uzly môžu byť rôzneho typu. Delia sa do dvoch hlavných skupín: uzly typu `Group` a typu `Leaf` [1]. Uzol typu `Group` môze obsahovať ďalšie uzly(potomkov), ktoré zoskupuje dohromady, takže operácie ako posunutie, rotácia a škálovanie môžu byť aplikované hromadným spôsobom. Uzly typu `Leaf` tvoria listy grafu, ktoré často reprezentujú viditeľné časti scény (modely), ale môže ísť tiež o nehmotné modely, osvetlenie či zvuky. Uzly tohoto typu môžu naviac obsahovať uzlové komponenty určujúce farbu, odrazivosť a ďalšie vlastnosti príslušného uzla. Uzly typu `Leaf` nemôžu mať žiadneho potomka a majú len jedného rodiča. Graf scény v Jave 3D nemôže obsahovať cykly, to znamená, že graf scény v Jave 3D je acyklický graf, tak ako ho poznáme z teórie grafov [1].

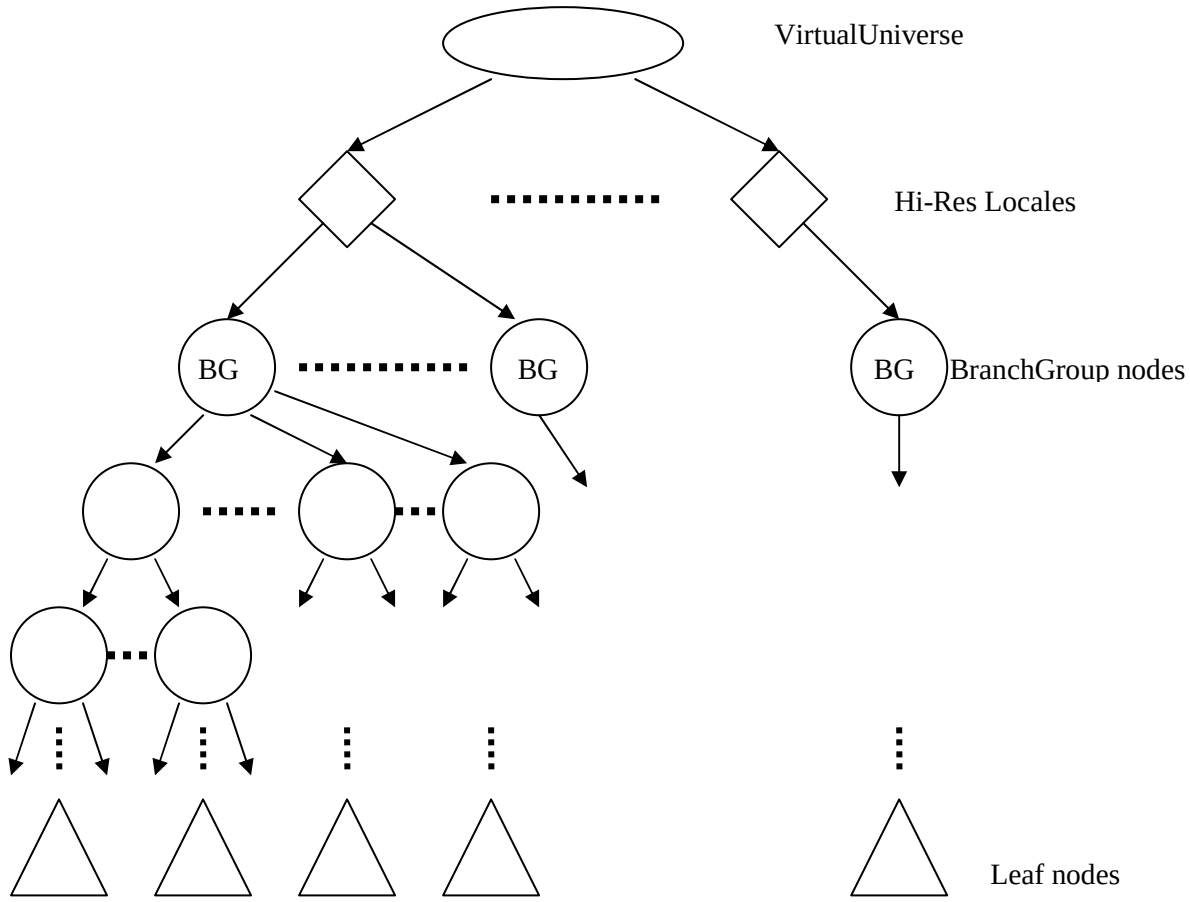
Do grafu scény môžeme ďalej pridávať správanie - to znamená uzly zachovávajúce programový kód, ktorý môže za behu programu ovplyvňovať iné uzly grafu. Typický uzol so správaním pohybuje geometrickými tvarmi, detekuje a reaguje na kolízie alebo napr. cyklicky mení osvetlenie z denného na nočné.

Miesto stromu scény sa používa graf scény, pretože uzly môžu byť zdieľané (majúce viac než jedného rodiča).

Funguje to asi tak, že hlavný, ako keby koreňový uzol, sa nazýva virtuálny vesmír (Trieda `VirtualUniverse`), na ktorý je možné pripojiť viac nezávislých podgrafov scény. Všetky podradené uzly dedia vlastnosti svojich nadradených uzlov, čiže tu funguje priama dedičnosť. Graf scény zabezpečuje organizáciu objektov scény. Vďaka tomu `renderer` vykresľuje objekty konzistentne, čím zaručuje súbežnosť vykresľovania. `Renderer` Javy 3D je schopný vykresliť určitý objekt nezávisle od druhých objektov, čím si zaručuje určitý typ nezávislosti hlavne vďaka tomu, že graf scény je možné rozdeliť na niekoľko častí.

Hierarchia grafu podporuje prirodzené priestorové zoskupenia na geometrické obejkty, ktoré sa nachádzajú na listových uzloch grafu. Vnútorné uzly sa snažia zoskupovať svojich potomkov do skupín, takže je na celú skupinu možné aplikovať jednu operáciu. Skupinové uzly taktiež vymedzujú priestorové hranice, ktoré obsahujú všetky jeho geometrie definované jeho potomkami. Priestorové zoskupovanie tak umožňuje veľmi jednoduché a rýchle vykonávanie operácií ako napr.: detekciu polohy, detekciu kolízie, nevykresľovanie polygónov, ktoré nie sú vidieť (`culling`) a pod [11].

Príklad grafu scény v Jave 3D s virtuálnym vesmírom (`VirtualUniverse`), skupinovými uzlami (`BranchGroupNodes`) a listovými uzlami (`LeafNodes`):



Obrázok 2 – Graf scény – Java 3D [11]

Takéto zobrazenie obejtov v grafe sa môže zdať mätúce, ale pritom je to celkom prirodzené. Ako príklad uvediem objekt „človek“. Ako skupinový uzol by mohol byť sám človek, ktorého listové uzly by boli konečné veci ako napr. prsty, vlasy a pod. (podľa potreby zložitosti). Tento skupinový uzol by mohol obsahovať ďalšie skupinové uzly – napr. trup, hlava a pod. Pritom keby bolo potrebné pohnúť prstom, tak pohneme len prstom, ak by bolo potrebné pohnúť s celým človekom, tak, samozrejme, pohneme so skupinovým uzlom „človek“ a nebudeme hýbať s každou časťou zvlášť – to je totiž tá hlavná výhoda tých skupinových uzlov.

Renderer Javy 3D vykresľuje jednotlivé uzly grafu postupne smerom od jeho koreňového uzla k listovým uzlom [11].

5.2 Architektúra – Java 3D

Nanešťastie dokumentácia a špecifikácia Java 3D API neobsahuje veľa informácií o vnútornej štruktúre tejto knižnice. Kvôli tomuto faktu sú veľké sťažnosti na danú politiku firmy Sun

Microsystems z komunity Javy 3D. Je totiž veľmi ťažké vedieť, ako spraviť správanie alebo ako fungujú napr. kolízie, keď človek absolútne nepozná vnútorný model. Jediné, čo je zatiaľ známe zo štruktúry Java 3D API, je len jeden list, ktorým sa čiastočne povrchovo vyjadril k architekrúre jeden z vývojárov Doug Twilleager.

Architektúra rozhrania Java 3D sa snaží byť čo najotvorenejšia rozličným metódam implementácie. Samozrejmosťou je, že si snaží zabezpečiť, aby sa nemohlo stať, že nebude možné v budúcnosti pridať vylepšenia. Špecifikácia uvádza, že jediným možným riešením ako zaručiť atomické rozšírenie grafu scény je volaním metódy `ProcessStimulus()`, ale v skutočnosti je implementácia o niečo zhovievavejšia.

Z otvoreného listu programátora Douga Twilleagera [4] vyplýva:

Sun Microsystems architektúru Javy 3D tají z dôvodu možných neustálych zmien v projekte. Stále sa ale vývojový tím snaží viac priblížiť danú špecifikáciu na viacerých miestach, ktoré sa pravdepodobne v budúcnosti nebudú, alebo len málo, meniť.

Programátorom v Sun Microsystems nestačil vláknový model klasickej Javy, tak si vytvorili svoj vlastný – bezprioritný systém vlákien, ktorý im umožňuje plnú kontrolu. Celý systém tak využíva zasielanie správ pre vykonanie zmien v štruktúrach.

Java 3D používa pre geometrické objekty štruktúry. Jednou z nich je tzv. geometrická štruktúra, ktorú obsahuje každý virtuálny vesmír. Druhou je renderovací zásobník (render bin), ktorý dané geometrické štruktúry triedi. Tento zásobník môže byť chápaný ako aktuálny snímok grafu scény.

Čo sa týka správania objektov, existuje v Jave 3D tzv. štruktúra správania (behaviour structure), ktorá priestorovo organizuje uzly správania. S toutou štruktúrou úzko súvisí vlákno - plánovač správania, ktoré aktivuje určité správanie, ak je v danej chvíli požadované.

Ako je spomenuté vyššie, Java 3D používa správy pre informovanie o zmene. Akonáhle sa čokoľvek zmení v grafe scény, je generovaná patričná správa. Tieto správy sa radia do radu a na začiatku každej iterácie plánovača správania (nekonečná slučka) sa správy spracujú a niektoré štruktúry sa aktualizujú.

Systém Javy 3D umožňuje efektívne použitie na viacerých procesoroch či obrazovkách. V druhom prípade ale čiastočne klesá výkon.

Existuje ešte monoho vlákien, ktoré tu neboli spomenuté. Je to hlavne kvôli tomu, že Sun Microsystems to viac-menej tají. Ale aspoň predstava o tom, ako to celé funguje, tu je.

5.3 OpenGL vs. DirectX – Java 3D

Ako už bolo zmienené, Java 3D umožňuje používať veľmi populárnu knižnicu OpenGL ale aj konkurenciu od spoločnosti Microsoft DirectX. Každé z nich ma pri programovaní v Jave 3D svoje výhody aj nevýhody:

Platforma:

DirectX je knižnica od Microsoftu, takže je možné ju používať len pod operačným systémom Windows.

Znaky:

Čo sa týka OpenGL, tak oproti DirectX má oveľa viac možností v programovaní. Java 3D 1.2.1 používa DirectX verziu 7.0, Java 3D 1.3 používa DirectX verziu 8.0.

Výkon:

Podľa testov je vidieť, že SUN Microsystems vkladá oveľa viac úsilia do implemetnácie OpenGL v Jave 3D. Momentálne totiž implementácia OpenGL dosahuje oveľa viac lepších výsledkov (okolo 30 až 50 %) v rýchlosti vykresľovania (FPS).

Stabilita:

Kedže OpenGL bolo naimplementované prvé v Jave 3D a má tým pádom väčšiu históriu, mohlo by sa zdať, že implementácia OpenGL bude stabilnejšia ako DirectX. Testy však prekvapivo hovoria proti OpenGL, kde viac chýb a pádov sa prejavilo práve pri OpenGL variante [6].

Detaily:

Prvky, ktoré nie sú prístupné v DirectX verzii Javy 3D oproti OpenGL:

1. Šírka čiary
2. Antialiasing čiary
3. Veľkosť bodu
4. Antialiasing bodu
5. 3D Textúra
6. Farbenie textúr

5.4 Transformácie – Java 3D

Transformácie v Jave3D sa klasicky ako pri OpenGL vyjadrujú transformačnými maticami. Ak napr. robíme transformáciu s maticou 4x4, môže to vyzerať asi takto:

$$\begin{bmatrix} m00 & m01 & m02 & m03 \\ m10 & m11 & m12 & m13 \\ m20 & m21 & m22 & m23 \\ m30 & m31 & m32 & m33 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

```
x′ = m00 . x + m01 . y + m02 . z + m03 . w
y′ = m10 . x + m11 . y + m12 . z + m13 . w
z′ = m20 . x + m21 . y + m22 . z + m23 . w
w′ = m30 . x + m31 . y + m32 . z + m33 . w
```

Ak transformujeme objekt typu `Point3f` do objektu `Point3d`, alebo objekt typu `Vector3f` do objektu typu `Vector3d`, w = 1 [1].

Pre celkový pohyb s objektami slúži v Jave 3d objekt typu `Transform3d` [1], ktorý definuje všetky typy pohybu objektu – transformácie, rotácie a škálovanie. Objekty typu `Transform3d` je možné použiť iba na špeciálne uzly grafu scény – `TransformGroup` [1] uzly. Niektoré metódy pre predstavu z objektu `Transform3d`:

`void rotX(double angle)`

Rotuje objekt okolo x-ovej osi v smere hodinových ručičiek. Hodnota je v radiánoch.

`void rotY(double angle)`

Rotuje objekt okolo y-ovej osi v smere hodinových ručičiek. Hodnota je v radiánoch.

`void rotZ(double angle)`

Rotuje objekt okolo z-ovej osi v smere hodinových ručičiek. Hodnota je v radiánoch.

`void set(Vector3f translate)`

Nastaví trasnformačnú hodnotu tejto matice.

`Transform3d` objekty tak definujú afínne homogénne transformácie, ale nie sú sučasťou grafu scény ako uzly. Tieto objekty tam musia byt prilinkované pomocou zmienených `TransformGroup` uzlov. `TransformGroup` uzly obsahujú rôzne bity, napr. pre povolenie animácie a pod.

Ak sa časť grafu scény skompiluje, renderer Javy 3D ho skonvertuje do oveľa efektívnejšej vnútornej podoby (hlavný dôvod konvertovania do internej podoby je zvýšiť výkon). Robenie transformácií v internej podobe má ale aj vedľajšie efekty. Jedným je, že je treba opraviť transformačnú hodnotu všetkých objektov v grafe [7].

5.5 Načítanie modelov – Java 3D

Zložité geometrické útvary môžeme, samozrejme, v Jave 3D vytvárať spájaním menších primitívnych objektov. Sami však musíme vyrátať 3D – súradnice jednotlivých objektov a zaistiť správne radenie vnútri geometrických objektov, čo sa dá dobre zvládnuť len pri tých najjednoduchších tvaroch, ako sú napr. kvádre alebo kužele. Omnoho rozumnejšie je vytvoriť objekt pomocou 3D – modelovacieho softwarového programu a potom ho do aplikácie načítať. Takéto objekty sa nazývajú externé modely,

pretože ich geometria (poprípade aj textúra) bola vytvorená bez pomoci Java 3D API. Keďže sa takýto model skladá vlastne z malých primitív, aj on sám osebe obsahuje svoj graf scény.

Ešte pred tým, než sa vykoná priechod grafom scény externého modelu, alebo sa na ňom spravia zmeny, musí sa tento model načítať. Knižnica Java 3D podporuje načítanie externých modelov prostredníctvom rozhrania Loader a triedy Scene. Vstupom rozhrania Loader je názov súboru a príznaky pre zapnutie a vypnutie načítania určitých prvkov modelu, ako sú napr. uzly s osvetlením, uzly so zvukom alebo uzly pohľadu.

V pomocnom balíčku knižnice Java 3D sa nachádzajú dve triedy implementujúce rozhranie Loader zamerané na konkrétne súborové formáty. Trieda Lw3dLoader vie načítať súbory so scénou z programu Lightwave 3D a trieda ObjectFile spracováva súbory typu OBJ programu Wavefront. Tretia trieda, LoaderBase, implementuje rozhranie Loader na obecnej rovine tak, aby prostredníctvom z nej odvodenej triedy bolo ľahšie realizovať načítanie ďalších formátov.

Pre knižnicu Java 3D existuje široké spektrum balíčkov pre načítanie rôznych 3D – modelov. Praktický zoznam je na adrese http://www.j3d.org/utilities/loaders.html. Jedným z najstarších a najznámejších pre Javu 3D je balíček NCSA Portfolio. Pomocou jedného rozhrania podporuje balíček radu formátov – napr. 3DS (3D Studio Max), DFX (Autocad), DEMs (Digital Elevation Maps), COB (TrueSpace) a WRL (VRML 97). Nevýhodou balíčka je, že je pomerne starý (aktuálna verzia 1.3 pochádza až z roku 1998). Ďalšou nevýhodou je, že momentálny vývoj je dokonca zastavený a podporu formátov má len na elementárnej úrovni, často sa totiž načíta len základná geometria a farby, bez textúr, správania či svetiel. Balíček ale obsahuje aj viac než rutiny na načítavanie 3D modelov, pretože obsahuje rozhranie pre niekoľko druhov vstupných zariadení a s jeho pomocou je možné ľahko zachytávať obraz na 3D – vykresľovacej ploche a prevádzať ho na videoklip [1].

5.6 Výkon – Java 3D

Java 3D bola spravená s cieľom mať čo najrýchlejší 3D grafický engine. Teraz sa pozrieme na niektoré vybavenia Javy 3D, ktoré vylepšujú jej výkon. Tieto vlastnosti sa týkajú najmä uzlov v grafe scény.

1. **Capability bits (Schopnostné bity)** [5]
   Schopnostné bity sú vlastne aplikačná cesta k opísaniu úmyslu implementácie Javy 3D. Takáto implementácia preverí tieto schopnostné bity a na ich základe zistí, ktorá z vlastností objektu sa môže zmeniť v danom čase (napr. tiene, osvetlenie...). Viacero optimalizácií sa dá dosiahnuť týmito bitmi.

2. **isFrequent bits** [5]

   Nastavenie `inFrequent` bitu znamená, že aplikácia môže často pristupovať alebo meniť tie atribúty, ktoré sú povolené príslušným schopnostným bitom. Toto sa môže použiť v Jave 3D ako pomôcka na vyhnutie sa niektorým optimalizáciám, ktoré by mohli spôsobiť, že tieto prístupy alebo modifikácie by boli nákladné. Predvolene je každý `isFrequent` bit spojený so schopnostným bitom nastavený (do 1).

3. **Compile** [5]

   V Jave 3D sú dva spôsoby metódy `compile`. Nachádzajú sa v triedach `BranchGroup` a `SharedGroup`. Keď sa zavolá táto metóda `compile()`, iba tie atribúty objektu, ktoré má povolené pomocou schopnostných bitov, sa môžu meniť. Vďaka tomuto je možné kompilovať aplikáciu v oveľa efektívnejšom vykresľovacom formáte.

4. **Bounds** [5]

   Veľa objektov v Jave 3D vyžaduje svoje hranice. Tieto objekty môžu obsahovať svetlá, správanie, hmlu, pripevnenia, pozadia, zvuky a pod. Účel týchto hraníc je limitovať priestorový obsah daného objektu. Implementácia môže ľahko ignorovať spracovanie tých objektov, ktoré sú mimo spracovávaného hlavného objektu.

5. **Neradené vykresľovanie** [5]

   Všetky objekty v Jave 3D sa vykresľujú spôsobom z koreňového uzla k listovým uzlom. To znamená, že vykresľovanie listových uzlov nemá vplyv na vykresľovanie ostatných listových uzlov. Vďaka tomu je možné radiť vykresľovanie objektov v poradí, akom sa to oplatí najviac a dosiahnuť tak oveľa lepšiu rýchlosť vo vykresľovaní.

6. **Vzhľad** [5]

   V Jave 3D existuje niečo ako Appearance `NodeComponent`. Je to objekt, ktorý slúži na vykresľovanie vzhľadu geometrie. Kedže sa vlastne skladá len z referencií, jeho implementácia je veľmi jednoduchá a rýchla. Vďaka tomu je možné obmedziť počet zmien vo vykresľovaní na lowĺevel úrovni vykresľovania.

7. **NIO buffer (New I/O)** [5]

   Toto je veľký úspech pre všetky aplikácie, ktoré používajú natívny kód jazyka C pri vykresľovaní geometrických objektov, čo sa týka úspory pamäte aj rýchlosti. Vďaka tomuto nemusia mať aplikácie dve kópie dát (jednu v C, druhú v Jave). Výkonne je to lepšie kvôli tomu, že sa nemusia kopírovať dáta z C štruktúry do Java štruktúry používaním JNI (Java Native Interface).

5.6.1 Najnovšie vylepšenia v Jave 3D 1.3

V tejto sekcii prejdem ešte ďalšie výkonnostné detaily, ktoré boli vytvorené v najnovšej verzii Javy 3D – 1.3.

1. **Hardware:** Java 3D sa spolieha na DirectX alebo na OpenGL, preto je najlepším hardwarovým zrýchlením použiť grafickú kartu, ktorá podporuje jednu z týchto knižníc.

2. **Kompilácia:** Nasledujúce kompilačné optimalizácie sú dostupné v Jave 3D verzii 1.3:

   Scene graph flattening: `TransformGroup` uzly, ktoré sa nedajú čítať alebo sa nedá do nich písať sú spojené do jedného transformačného uzla.

   Combining Shape3D nodes: Nezapisovateľné uzly typu `Shape3D`, ktoré majú rovnaké vzhľadové atribúty – neuchytiteľné, nekolidovateľné a sú pod jednou `TransformGroup` (po flatteningu) sú kombinované (interne) do jedného `Shape3D` uzla a môžu byť vykresľované s menšími výdavkami.

3. **View Frustum Culling:** Java 3D má naimplementované tzv. view frustum culling (odstraňovanie objektov, ktoré nie sú vidieť).

4. **Multithreading:** Java 3D bola implementovaná ako plne vláknový systém. V jednom čase tak môže naraz bežať niekoľko vlákien, ktoré vylepšujú výkon, ako napr. detekcia, či je objekt vidieť, samotné vykresľovanie, plánovač správania, plánovač zvuku, spracovanie vstupov, detekcia kolízií a pod. [5].

6 Java Monkey Engine

6.1 Graf scény – jME

Graf scény je skupina uzlov hierarchicky usporiadaných vzhľadom na ich priestorovú polohu. Tento strom obsahuje listové uzly, ktoré reprezentujú objekty a interné uzly, ktoré pomáhajú riadiť tieto objekty.

Existujú 4 hlavné dôvody, prečo je takéto usporiadanie pre hry výhodné:

1. Dáta v hre (objekty, grafika atď.) sú typicky veľmi veľké. Sú taktiež typicky rozmiestnené priestorovo, teda podľa ich pozícií, takže tieto dáta môžu byť ľahko zoskupené podľa polohy v 3D svete. Kedže aj reálny svet je v podstate organizovaný týmto spôsobom, je prirodzené, že pridanie položky v grafe (napr. `LightNode` [3]) ovplyvní všetky položky pod ním. Toto je zaobstarané automaticky v implementácii grafu scény.
2. Táto hierarchická štruktúra zaručuje odkazy pre samotné priestorové umiestnenie jednotlivých objektov prostredníctvom koreňového uzla vetvy grafu (koreňový uzol vetvy „obsahuje“ všetky jeho deti). Vďaka tomuto je možné ľahko odstrániť tie časti grafu, ktoré nie sú vidieť (ak nevidíme koreňový uzol vetvy, nevidíme ani jeho deti).
3. Veľa objektov vo svete je reprezentovaných práve takouto stromovou štruktúrou, preto je ľahké zobraziť túto štruktúru. Napr. postava v hre. Jej chodidlo je závislé od polohy a orientácii členka a ten je závislý od spodnej časti nohy atď. Pretože pozícia a rotácia uzlov je dedená smerom dole, rotovanie alebo posúvanie spodnej časti nohy sa prejaví aj na chodidle.
4. Je jednoduché exportovať scénu do iného formátu – napr. XML.

Graf scény v jME je tvorený dvoma typmi uzlov: Interné uzly a listové uzly. Interné uzly sa nazývajú jednoducho uzly a listové uzly sa nazývajú geometrie. Uzly môžu obsahovať deti (iné uzly alebo geometrie), zatiaľ čo geometria obsahovať deti nemôže. Každý uzol (aj geometrie) obsahujú v sebe dôležité informácie: transformácie, ohraničujúci rozsah, stavy vykresľovania a radiče.

Transformácie (Transformations) definujú orientáciu, pozíciu a škálu uzla. Vzťahujú sa na deti, takže rotovanie rodiča sa prejaví aj na jeho deti.

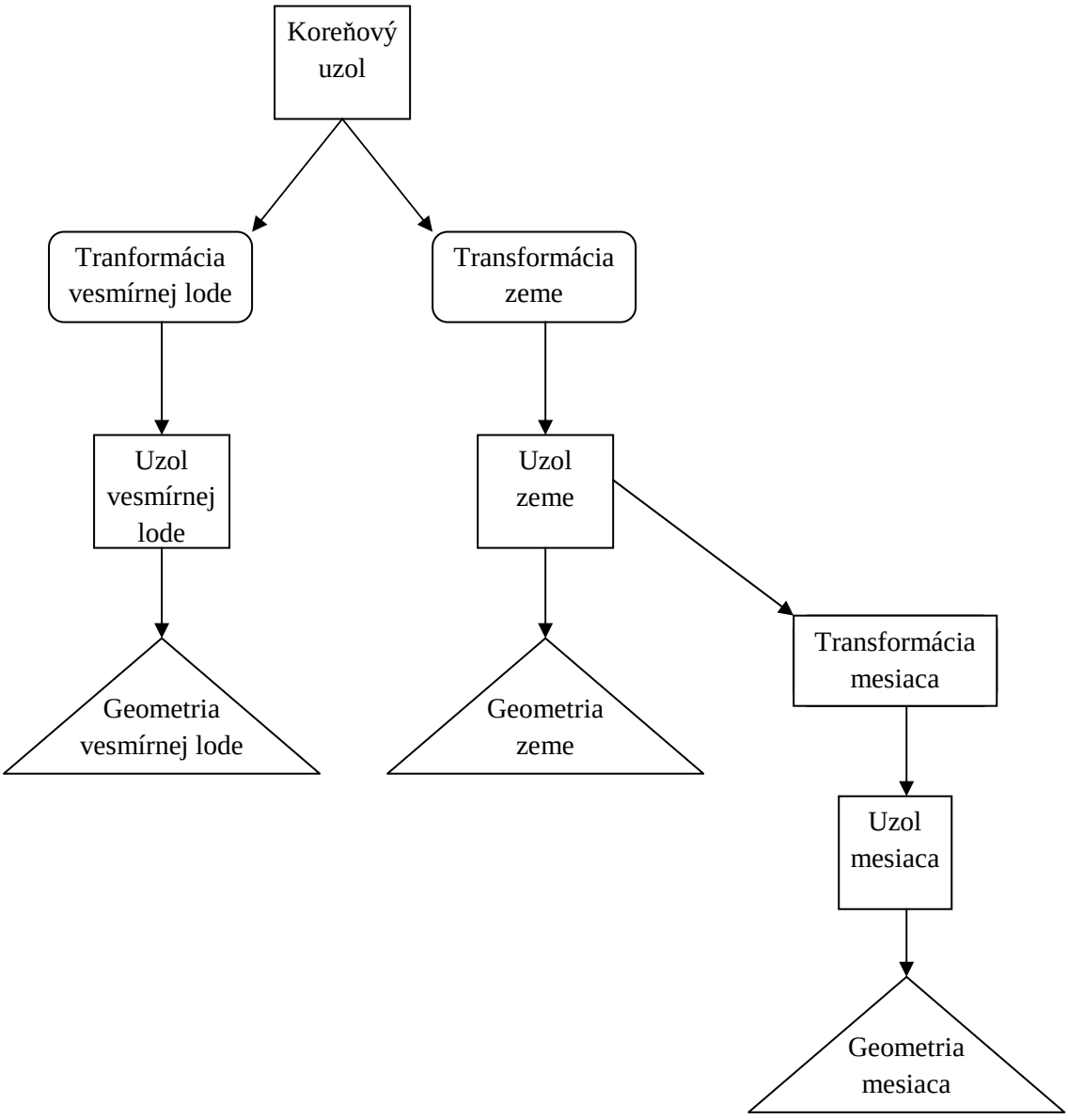
Ohraničujúci rozsah (Bounding volume) definuje minimálnu rozlohu jednotlivých uzlov. Z toho dôvodu, na geometrickej úrovni, rozsah uzla obsahuje všetky jeho vrcholy. Na úrovni uzlov rozsah obsahuje všetky rozsahy jeho potomkov.

Stav vykresľovania (Render state) určuje ako budú jednotlivé geometrie vykresľované na displej. Toto sa, samozrejme, vzťahuje aj na deti uzla. Toto zaručuje minimálne prepínanie stavu, takže napr. ak uzol obsahuje tisíce geometrií (rovnakých, napr. kríky), tento uzol môže obsahovať len jeden stav textúry, ktorá sa aplikuje na jeho tisíc detí (listov kríku).

Radiče (Controllers) sa používajú na definovanie striedavých zmien uzlov za jednotku času. Obvykle sú to rôzne typy animácií, fyzických atribútov (hádzanie lopty), atď. Radiče sa sami osebe nevzťahujú na deti, ale efekt na uzol mať môžu [3].

6.1.1 Príklad grafu scény – jME

Nasledujúci príklad popisuje zjednodušenú organizáciu vesmírnej hry. Sú tu tri hlavné geometrie: samotná vesmírna loď, zem a mesiac. Vesmírna loď môže, samozrejme, voľne lietať po vesmíre, preto je pripojená na koreňový uzol scény. Keďže mesiac obieha okolo zeme, je pripojený k uzlu zeme, aby mu bolo umožnené toto rotovanie, aj keď sa zem posunie na inú pozíciu.



Obrázok 3 – Graf scény – jME [3]

6.1.2 Tipy pre graf scény – jME

Nie vždy je výhodné navrhovať graf scény priestorovo. Niekedy je lepšie, keď sa graf scény navrhuje podľa toho, aké stavy jednotlivé uzly zdieľajú. Napr. ak máme viacero stromov, je lepšie pripojiť ich pod jeden uzol (a k nemu priradiť konkrétny vykresľovací stav), ako keby sa to malo navrhnúť podľa priestoru [3].

6.2 Architektúra – jME

jMonkey Engine (jME) bol navrhnutý ako vysokorýchlostný grafický engine. jME vypĺňa veľa potrieb za predpokladu vysokého výkonu grafu scény. jME využíva posledné schopnosti OpenGL a taktiež efektívne využíva kapacity moderných grafických kariet za predpokladu ľahkého použitia pre game-designera. jME bol navrhnutý s intuitívnym používaním. Modulárny design zaisťuje, že ako sa počítačový grafický hardware vylepšuje a OpenGL vylepšuje prispôsobilosť, jME bude rýchlo schopný využiť dané vylepšenia a prisposobiť sa podmienkam a zmenám.

V jadre jME systému je graf scény. Graf scény je dátová štruktúra, ktorá riadi dáta sveta. Vzťahy medzi dátami hry (geometria, zvuk, fyzika…) sú spravované v stromovej štruktúre s uzlami typu list (`Leaf`), reprezentujúcimi elementy jadra hry. Tieto elementy sú väčšinou tie, ktoré sú vykresľované do scény alebo hrané ako zvuky cez zvukovú kartu – sú to objekty virtuálneho sveta. Organizácia grafu scény je veľmi dôležitá a typicky závisí na aplikácii. Avšak typicky graf scény reprezentuje veľký objem dát, ktorý môže byť rozložiteľný na menšie, ľahko ovládateľné kúsky. Zvyčajne sú tieto kúsky zoskupované do nejakého typu vzťahu, obyčajne priestorovým rozmiestnením. Takéto zoskupovanie dovoľuje odstrániť veľké množstvo sekcií dát, ak nie sú potrebné byť zobrazené na konkrétnej scéne. Napr. ak mám ku scéne pripojený ako uzol objekt auto a k uzlu auto mám pripojené ostatné časti auta (ako kolesá, karosériu a pod.), môžem odpojiť len uzol auto a je zrejmé, že samotné kolesá mi v scéne nezostanú.

Pokým graf scény je jadro grafických elementov jME, aplikácia poskytuje prostriedky rýchlo vytvoriť grafický kontext a začať hlavnú slučku programu. Tvorba okna, input systému, kamerového systému a systému hry nie je viac než volanie jednej alebo dvoch metód. Toto umožňuje programátorovi nestrácať čas nad zaoberaním sa systémom, ale môže hneď začať pracovať na systéme hry. Zobrazovací systém a renderovací systém poskytujú abstrakciu nad komunikáciou s grafickou kartou. Toto ale nie vždy môže byť výhoda. Zmienená abstrakcia nad grafickou kartou sa odráža bohužiaľ aj vo výkone, čo má za dôsledok pomalé vykresľovanie.

Aktuálne renderovanie grafu scény je užívateľovi skryté. Toto je z dôvodu prepínania z a do iného rednerovacieho systému bez prerušenia aplikačného kódu. Napr. ak sa používa LWJGL od Puppy games [12], môže sa pohodlne prejsť na JOGL [3] od SUN Microsystems. Môžu tu byť síce nejaké malé rozdiely systémov, ale prechod je veľmi jednoduchý a netreba prerobiť celý projekt.

Avšak vylepšenia prevýšili manažovanie dvoch renderovacích systémov, preto je aktuálny renderovací systém len jeden – LWJGL od Puppy games.

Každá časť jME bola postavená na základe jednoduchých stavebných blokoch, takže všetky grafické prvky pochádzajú z triedy `Spatial` [1], všetky vstupy a k nim aj akcie pochádzajú z triedy `InputAction` atď [3].

6.3 LWJGL – jME

Knižnica jME využíva pre vykreslenie hraciu knižnicu LWJGL.

Lightweight Java Game Library (LWJGL) je riešením pre profesionálnych, ale aj amatérskych programátorov vyvíjať kvalitné hry v programovacom jazyku Java. LWJGL zaručuje developerom prístup k vysoko-rýchlostným medzi-platformovým knižniciam ako OpenGL (Open Graphics Library) a OpenAL (Open Audio Library). Naviac LWJGL umožňuje prístup k hráčskym ovládačom ako napr. gamepady, volanty a joysticky. A to všetko v jednoduchom silnom rozhraní Javy.

LWJGL nie je myslené tak, že sa pomocou neho dajú veľmi ľahko spraviť hry; ide hlavne o to, umožniť technológiu, ktorá povolí programátorom ľahšie spraviť veci, ktoré sú v Jave buď ťažko implementovateľné, alebo dokonca vôbec. Výrobcovia LWJGL predpokladajú, že časom a rozšírením by sa pomocou LWJGL mohli programovať kompletné hracie knižnice a enginy [12].

Rýchlosť – LWJGL

Celkový zmysel LWJGL je preniesť rýchlosť vykresľovania v Jave do 21. storočia. Preto sa v LWJGL nachádzaju napr. tieto veci:

Odstránené metódy, ktoré su efektívne pre programovanie v jazyku C, ale pre Javu nedávajú žiadny zmysel (napr. `glColor3fv`).

LWJGL vyhodí výnimku, ak na danom stroji chýba hardwarová podpora na platforme Windows. Je totiž nezmysel spúšťať hru, ak pobeží menej ako 5 fps [12].

Všadeprítomnosť – LWJGL

LWJGL je navrhnutá ako knižnica pre veľkú škálu zariadení. Od malých telefónov až po veľké multiprocesorové stroje. Kedže ešte malé zariadenia nemajú tak výkonnú JVM, neznamená to, že nikdy ju mať nebudú – raz určite. Preto je LWJGL vyvíjaná opatrne a vývojári si dávajú záležať aj týmto smerom. Preto vyvinuli OpenGL ES (embeded systems – vstavané systémy). To znamená, že:

1. Kedže je LWJGL veľmi šetrná, čo sa týka priestoru (celková veľkosť je menej než ½ MB), je to veľká výhoda pre malé zariadenia, pri ktorých nie je pamäťový priestor určite zanedbateľný.
2. LWJGL pracuje na najmenšej úrovni, a preto sa radšej sústredí na principiálne hlavné veci ako na návrh všetkých možných možností, ale vývojári zaručujú 99 % pokrytia všetkých

najpoužívanejších častí. Preto LWJGL používa len jedno okno (viac sa dá spraviť pomocou AWT či Swingu), a preto LWJGL nepoužíva viac vlákien pre vykresľovanie [12].

Jednoduchosť – LWJGL

LWJGL potrebuje byť jednoduchá, aby sa dala využiť čo najväčším počtom vývojárov. Autori chcú zaistiť, aby začínajúci programátori boli schopní rýchlo preniknúť do rozhrania tejto knižnice a aby ju profesionálni programátori mohli používať profesionálne. Výrobcovia sa museli rozhodnúť medzi dvoma paradigmami – jednou, ktorá presne sadne na OpenGL a druhou, ktorá presne sadne na mierenú sortu – od PDA až po desktopy. Preto vývojári:

1. odstránili 99 % zbytočných častí, o ktorých väčšina programátorov nevie.
2. sa rozhodli, že konzistencia je lepšia než komplexnosť. Radšej, ako používať viac typov volaní k jednej metóde a tým pádom nafukovania knižnice, si povedali: „*Žiadne polia! Aj tak sú pomalé. Poďme používať zásobníky, aj tak je to práve to, pre čo boli zásobníky stvorené*“ [12].

Šetrnosť – LWJGL

1. Malé == jednoduché – jednoduchšie je učiť sa len jeden spôsob vykonania, ako učiť sa viacero všemožných možností.
2. Malé == zdrojový kód LWJGL je menej chybový.
3. Malé == ľahko stiahnuteľné.
4. Malé == J2ME [12].

Bezpečnosť – LWJGL

Vývojári LWJGL sa rozhodli kvôli trendu zaviesť len nedávno bezpečnosť do tejto knižnice:

1. Nepoužívajú ukazatele, namiesto nich používajú zásobníky.
2. K zásobníkom sú pridané kontroly, ktoré dozerajú na pretečenie, aby sa predišlo tzv. zásobníkovým útokom [12].

Robustnosť – LWJGL

Podobne ako pri bezpečnosti si vývojári až nedávno uvedomili, že spoľahlivý systém je oveľa použiteľnejší ako rýchly systém. Keďže bola LWJGL spustená viacerými benchmarkami, rozhodli sa vývojári odstrániť assert makrá a nahradiť ich jednoduchšími podmienkami `if`(...) alebo `try-catch` blokmi. Taktiež premiestnili všetky kontroly GL chýb z natívneho kódu do kódu Javy, takže nie je potrebná oddelená debugovacia DLL knižnica [12].

6.4 Transformácie – jME

Transformácie definujú operácie, ktoré konvertujú bod z jedného súradnicového systému do druhého. Toto zahrňuje premiestňovanie, rotácie a škálovanie. V jME sa lokálne transformácie používajú pri pozicovaní objektu vzhľadom na jeho rodiča. Tzv. svetové transformácie sa vzťahujú na globálny súradnicový systém.

Typicky sa pre lineárne transformácie (mapovanie vektorov na vektory) používa matica [13]. To je: Y = MX, kde X je vektor a M je matica, ktorá aplikuje jednu alebo viac translácií (škálovanie, rotovanie alebo pozicovanie). [3]

Existujú niektoré špeciálne matice:

1. Nulová matica je matica, ktorá má na všetkých pozíciách samé nuly:

0	0	0
0	0	0
0	0	0

2. Jednotková (identická) matica – má na hlavnej diagonále jednotky, inde nuly.

1	0	0
0	1	0
0	0	1

3. Transponovanie matice $M = [m_{ij}]$ je $M^T = [m_{ji}]$. To znamená, že riadky matice M sa stanú stĺpcami a naopak.

1	1	1
2	2	2
3	3	3

↓

1	2	3
1	2	3
1	2	3

4. Matica je symetrická ak $M = M^T$.

X	A	B
A	X	C
B	C	X

jME obsahuje dva typy tried matíc: `Matrix3f` a `Matrix4f`. `Matrix3f` je matica 3x3 a je používaná najviac (je schopná škálovať a rotovať), zatiaľ čo `Matrix4f` je matica 4x4, ktorá naviac zvládne pozicovanie [3].
Násobenie vektoru s maticou umožní danému vektoru transformáciu [3][13].

Škálovanie

Ak zadefinujeme diagonálnu maticu definovanú ako D = [d_{ij}] a d_{ij} = 0 pre i != j a má všetky tieto čísla kladné, tak ide o škálovaciu maticu. Ak sú tieto koeficienty väčšie ako 1, tak sa cieľový vektor zväčší, ak menšie tak sa zmenší [3].

Rotácie

Rotačná matica potrebuje, aby jej transponovaná a inverzná matica boli rovnaké ($R^{-1} = R^T$). Následne potom rotačná matica je počítaná ako: R = I + (sin(uhol)) S + (1 - cos(uhol)S^2 kde S je [3]:

0	u_2	$-u_1$
$-u_2$	0	u_0
u_1	$-u_0$	0

Pozicovanie

Pozicovanie vyžaduje maticu 4x4, kde vektor (x,y,z) je namapovaný ako (x,y,z,1) pre násobenie. Pozičná matica je potom definovaná ako:

M	T
S^T	1

Kde M je matica 3x3 (obsahujúca rotačné/škálovacie informácie), T je pozičný vektor a S^T je symetrický vektor k T. 1 je iba konštanta [3].

6.5 Načítanie modelov – jME

Načítanie modelov do aplikácií (hier) je podstatne ľahšie ako ich tvoriť priamo pomocou základných primitívnych útvarov. Model sa jednoducho vytvorí pomocou špeciálnych programov externe – bez pomoci Javy a následne sa do aplikácie načíta.

Knižnica jME dokáže načítať veľké množstvo najznámejších formátov – 3DS (3D Studio MAX), ase, md2 (Quake 2 modely), md3 (Quake 3 modely), ms3D (MilkShape), obj (WaveFront 3D Object) a X3D. Na všetky tieto modely slúžia samostatné triedy, ktoré tieto modely konvertujú do vlastného formátu JME. Samozrejme je tento modelový formát možné načítať, pokiaľ je externý, bez použitia konverzie ako pri iných formátoch [3].

Pri testoch (viď. Príloha - Testy) som používal modely vo formáte OBJ (so záznamom o vlastnostiach modelu v súbore typu MLT, ktorý si jME takisto sama bez problémov načíta), s ktorými knižnica nemala sebemenší probém. Pri načítaní modelu vo formáte 3DS už ale problém nastal, kedy pri načítaní modelu lietadla, ktoré malo animáciu vrtule, bolo lietadlo trocha rozsypané a časti (vrtuľa, kolesá a telo) nesedeli tak, akoby mali. Modely z hry Quake 3 jME zvládla bez problémov aj s animáciami.

Ako bolo spomenuté jME si externé modely konvertuje do svojho formátu JME. K tomuto účelu využíva systém `Collada`. Modelový formát `Collada` je vytvorený skupinou Khronos. Je to open source štandart, ktorý sa neustále vyvíja.

Modelový formát tak môže byť importovaný do jME cez tzv. `ColladaImporter`, ktorý vytvorí príslušný graf scény elementov, ktoré sa nachádzajú v súbore modelu. Tento Importer súčasne podporuje geometriu a kostrovú animáciu.

7 3D Letecký simulátor – jME

Cieľom tejto práce je poukázať na výhody jednej zo zmienených dvoch knižníc. Preto som zvolil práve 3D letecký simulátor, pretože na ňom sa dajú ukázať takmer všetky základné možnosti 3D knižnice – transformácie, terén, načítanie modelov, LOD (Level Of Detail) [1] [3], voda, osvetlenie a pod. Ďalej tu nie je až tak potrebné vytvárať umelú inteligenciu, čo určite uľahčí zložitosť problému.

Z výberu knižníc Java 3D a jME som vybral knižnicu jME a to z viacerých dôvodov:

1. jME je knižnica robená s prioritou na vytváranie hier.
2. Vzhľadovo vytvára oveľa krajšie vykresľovanie ako Java 3D.
3. Podľa testov je jME oveľa rýchlejšia.
4. Po vlastných skúsenostiach v programovaní je jME viac príjemnejšia a prirodzenejšia, čo sa týka komfortu programovania.

7.1 Objekt - lietadlo

Hlavným a jediným pohybujúcim sa objektom v aplikácii je model lietadla. Je to externý model vyrobený v programe 3D Studio MAX. Skladá sa z dvoch častí a to z tela lietadla a vrtule. Sú to vlastne dva rôzne modely a to z toho dôvodu, aby bolo možné roztočiť vrtuľu nezávisle na tele modelu lietadla. Obidva tieto objekty sú vo formáte OBJ s vlastnosťami v pripojenom súbore typu mlt, pričom obidva tieto typy knižnica jME zvláda bez menších problémov. Na načítanie OBJ modelu slúži v jme trieda `com.jmex.model.converters.ObjToJme`.

Celý objekt lietadla s jeho metódami je v triede `plane_my.java`. Je to z dôvodu, že Java je vyslovene objektový jazyk a lietadlo objektom v danom prípade určite je. Táto trieda obsahuje vlastnosti lietadla ako napr. rýchlosť, akceleráciu, brzdnú silu a pod. + metódy prislúchajúce k týmto vlastnostiam a metódu `update`, ktorá je volaná s každým vykreslením snímku.

Samotný model nesie na sebe jednoduchú textúru s vojnovým maskovacím motívom. Je to vlastne obrázok vo formáte png (jME zvláda takmer všetky známe formáty), ktorý je roztiahnutý po celom povrchu lietadla. Akonáhle je ale tento model takto otextúrovaný, padá rapídne výkon, v mojom prípade až asi o 30%.

Vzhľadom na veľkosť modelu bolo nutné zmeniť mu veľkosť a preškálovať ho na desatinnú veľkosť oproti pôvodnej.

7.1.1 Ovládanie

Samotné ovládanie lietadla je zhotovené pomocou upravených tried, ktoré dedia nadradenú triedu `KeyInputAction`. Vďaka tomu je (oproti klasickej možnosti – `KeyBindings` – pri každom vykreslení snímku sa kontroluje stlačená klávesa, ktorá čaká v rade) možné stláčať viac kláves naraz, pričom lietadlo na ne reaguje. Pomocou týchto tried sú vytvorené objekty, na ktoré sú namapované klávesy príslušných akcií. Pri potrebe (stlačení tlačidla) sa volá metóda, ktorú je potrebné implementovať, ak dedíme `KeyInputAction` – `performAction`. V nej sú špecifikované úkony, ktoré sa majú vykonať.

Takýmto spôsobom je spravené celkové ovládanie a to – zmienený pohyb dopredu a dozadu, nakláňanie lietadla do strán pri zatáčaní, samotné otáčanie a nakláňanie lietadla dopredu a dozadu.

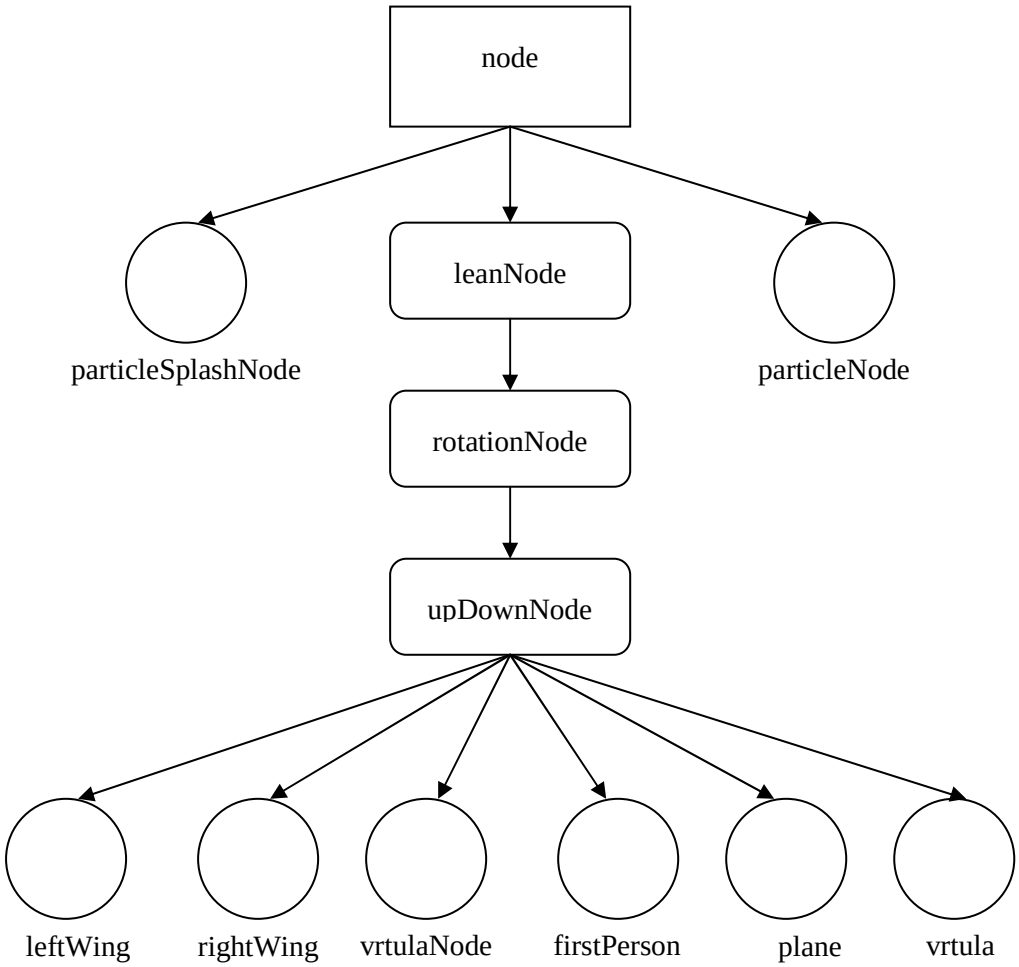
Ďalej je tu ešte posledný objekt triedy `KeyInputAction` a to tzv. „drift“ lietadla. Ten sa nevzťahuje na žiadnu klávesu, jeho metóda `performAction` sa volá prakticky s každým vykreslením snímku. Tento objekt definuje spomaľovanie lietadla, pokiaľ lietadlo nepridáva (teda nie je držaná šípka hore).

Pohyb lietadla sa ovláda pomocou šiestich kláves. Klávesy W, S, A, D slúžia na nakláňanie lietadla (W – naklonenie dopredu – lietadlo smeruje nosom dole, S – naklonenie dozadu - lietadlo smeruje nosom hore, A a D slúžia tak na zatáčanie lietadla, ako aj na nakláňanie lietadla pri zatáčaní) a šípky hore a dole slúžia ako plyn a brzda.

7.1.2 Graf scény

Graf scény lietadla je zhotovený z niekoľkých uzlov a to kvôli rotáciám a pozicovaniu. Hlavný uzol `Node` reprezentuje lietadlo ako celok. Preto je na najvyššej pozícii v grafe. Hneď pod ním sa nachádza `leanNode` – ten slúži na nakláňanie lietadla pri zatáčaní. Spomedzi všetkých transformačných uzlov je najvyššie a to preto, lebo nakláňanie lietadla sa musí vzťahovať na všetky ostatné transformácie lietadla. Naledujúci uzol je `rotationNode`. Pomocou neho sa vykonávajú rotácie, čiže zatáčanie lietadla. Zatáčanie sa musí vzťahovať zároveň na naklonenie lietadla dopredu alebo dozadu, preto sa tzv. `upDownNode`, ktorý slúži práve na zmienené nakláňanie lietadla vpred a vzad, nachádza pod ním. A toto bol posledný transformačný uzol v grafe scény lietadla. Pod ním sú už len listy, či inak geometrie. Listy `leftWing`, `rightWing` a `vrtulaNode` sú len imaginárne uzly na konci krídiel a vrtule, ktoré slúžia ako hitboxy pri kontrole zrážky lietadla. Pri zrážke sa generuje výbuch lietadla tvorený časticami (particles). Tento výbuch je reprezentovaný listom `particleNode`. Podobne ako výbuch je tvorené aj šplechnutie do vody, prezentované uzlom `particleSplashNode`. List `firstPerson` je ďalší imaginárny uzol, ktorý sa nachádza tesne pred vrtuľou. Pozícia tohoto uzla je využívaná pri prepnutí kamery do „first person“ módu. Posledné

dve geometrie `plane` a `vrtula` sú zmienené modely tela lietadla a vrtule. Všetky tieto listy sa musia vzťahovať na všetky transformácie týkajúce sa lietadla. Preto sú až na spodu výsledného grafu.



Obrázok 4 – Graf scény - lietadlo

Takýto výsledný graf je pripojený ku koreňovému uzlu celej scény pomocou uzlu `node`.

7.1.3 Transformácie

Na lietadlo sú aplikované nasledujúce transformácie:

1. Pohyb lietadla – pozicovanie
2. Rotácia lietadla (otáčanie) okolo osi Y
3. Rotácia lietadla (nakláňanie do strán) okolo osi X
4. Rotácia lietadla (nakláňanie vpred a vzad) okolo osi Z
5. Škálovanie – zmenšenie lietadla vzľadom na jeho veľkosť

7.1.3.1 Pohyb lietadla – pozicovanie

Pohyb lietadla je aplikovaný na celkové lietadlo, čiže na uzol `Node`. Lietadlo sa pohybuje v smere jeho lokálnej osi X a na základe jeho rýchlosti vynásobenej časom medzi vykreslením snímkov kvôli synchronizácii pri zmene rýchlosti vykreslenia.

Rýchlosť lietadla sa logicky mení podľa pridávania plynu a to logaritmicky (kvôli rýchlosti výpočtu je logaritmus aproximovaný priamkami). Napr. ak je rýchlosť menšia ako 20, lietadlo zrýchľuje lineárne s 20-násobkom času medzi vykreslením snímkov. Ak je rýchlosť väčšia ako 20 a menšia ako 40, zrýchľuje lietadlo lineárne 16-násobkom času medzi vykreslením snímkov atď. Maximálna rýchlosť lietadla je 150. Držaním plynu lietadlo zrýchľuje maximálne po 140, rýchlosť 150 sa dá dosiahnuť len padaním z veľkej výšky.

Ak lietadlo klesá, čiže y – súradnica aktuálneho snímku je menšia ako y – súradnica snímku predošlého, rýchlosť sa zvyšuje nasledujúcim spôsobom:

```
rýchlosť_lietadla += interpolation * abs(sin(uhol)) *
(rýchlosť_lietadla/3);
```

Kde `interpolation` je čas medzi vykreslením snímkov a `uhol` je uhol naklonenia lietadla voči rovine XZ. Z toho je zrejmé, že čím väčší uhol medzi lietadlom a rovinou XZ, tým viac bude rásť rýchlosť lietadla. Obdobne je to aj pri stúpaní lietadla do výšky (čiže y – súradnica aktuálneho snímku je väčšia ako y – súradnica predošlého snímku), len s opačným znamienkom pri pripočítaní k rýchlosti. Takto to funguje až po maximálnu rýchlosť motora lietadla a to 140. Od 140 po 150 sa k rýchlosti pripočítava len čas medzi vykreslením snímkov.

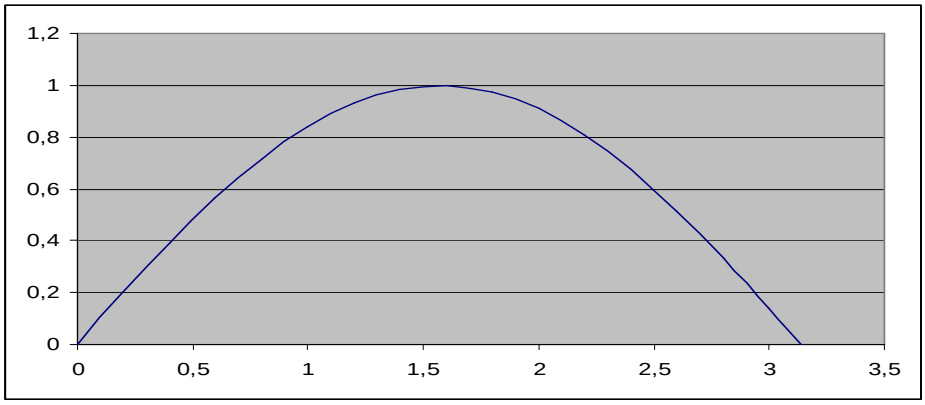
7.1.3.2 Rotácia lietadla (otáčanie) okolo osi Y

Rotácia lietadla je aplikovaná na uzol `rotationNode` a to nasledujúcim uhlom natočenia:

```
(smer_rotácie) * sin((rýchlosť_otáčania_lietadla *
rýchlosť_lietadla)/100))
```

Kde `smer_rotácie` je len číslo 1 alebo -1, ktoré reprezentuje, či sa lietadlo točí vpravo alebo vľavo.

Keďže rýchlosť lietadla sa pohybuje v rozmedzí 0 až 150 a rýchlosť otáčania lietadla je nastavená na čislo 2, je zrejmé, že v tom sínuse sa budú čísla pohybovať od 0 po 3, čo je zaokrúhlene od 0 po π, čiže klasický sínus s definičným oborom od 0 po π. Z grafu sínusu teda jasne vyplýva, že pri rýchlosti 0 sa lietadlo nedá otáčať vôbec. So zvyšujúcou sa rýchlosťou sa lietadlo otáča lepšie až po hranicu rýchlosti približne 75, kde sa lietadlo ovláda najlepšie. Ak sa rýchlosť zvyšuje ďalej, lietadlo sa ovláda horšie a horšie až po maximálnu rýchlosť 150, kde sa už lietadlo otáčať nedá vôbec.



Obrázok 5 – Graf rotácie lietadla (sínus)

7.1.3.3 Rotácia lietadla (nakáňanie do strán) okolo osi X

Táto transformácia je aplikovaná na uzol `leanNode`. Funguje to podobne ako pri rotácii lietadla okolo osi Y sposobom:

```
a = π/4*(sin((rýchlosť_otáčania_lietadla * rýchlosť_lietadla)/100));
vertikálna_zmena = true;
vertical+ = lean * čas * rýchlosť_otáčania_lietadla;
ak (vertical<(-a)){
     vertical = -a;
}
ak (vertical>(a)){
     vertical = a;
}
Nastav leanNodu otočenie o uhol vertical okolo Y – osi;
```

Kde `vertikálna_zmena` reprezentuje zmenu naklonenia lietadla, aby bolo možné lietadlo automaticky vrátiť, pokiaľ nie je stlačená klávesa. Identifikátor `lean` určuje smer natočenia (nabúda hodnoty -1 alebo 1), `vertical` je uhol natočenia. Čiže najviac sa lietadlo nakloní pri rýchlosti blízkej 75, smerom k maximu, respektíve minimu rýchlosti sa nakláňanie zmenšuje. Ak nie je držaná klávesa natočenia, lietadlo sa natočí naspať do vodorovnej polohy:

```
ak (abs(vertical)>0.1){ //Ak som naklonený
     ak(vertical > 0){ //vlavo
          //vraciame sa
          vertical -= interpolation * rýchlosť_otáčania_lietadla;
     }
     inak{//vpravo
          //vraciame sa
          vertical += interpolation * rýchlosť_otáčania_lietadla;
     }
}
inak{vertical=0;}
```

7.1.3.4 Rotácia lietadla (nakláňanie vpred a vzad) okolo osi Z

Táto transformácia je aplikovaná na uzol `upDownNode`. Funguje to znova obdobne ako pri predchádzajúcich dvoch rotáciách – podľa rýchlosti.

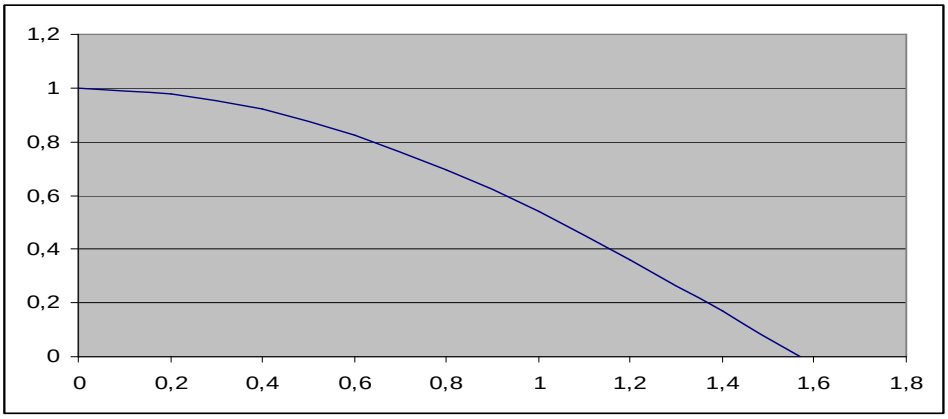
```
(-smer_rotácie) * čas * (sin((rýchlosť_otáčania_lietadla *
rýchlosť_lietadla)/100))
```

Z podobnosti kódu s predchádzajúcimi rotáciami je jasné, že najlepšie sa lietadlo nakláňa okolo rýchlosti 75 a smerom k maximálnej, respektíve minimálnej rýchlosti sa toto natáčanie zhoršuje.

Ak je rýchlosť lietadla menšia ako 100, lietadlo začína klesať, teda samovoľne sa nakláňať smerom dopredu (nosom lietadla dole). To je vyjadrené nasledujúcim kódom:

```
ak((rýchlosť_lietadla) > 0 && rýchlosť_lietadla < 100){
     ak(uhol voči rovine XZ > 0){ //padáme podľa uhla natočenia
          uhol natočenia = interpolation *
          cos(rýchlosť_otáčania_lietadla *
          (π/(rýchlosť_otáčania_lietadla * 200)) *
          (rýchlosť_lietadla);
    }
    inak{
          uhol natočenia = (-interpolation *
          cos(rýchlosť_otáčania_lietadla *
          (π/(rýchlosť_otáčania_lietadla * 200)) *
          (rýchlosť_lietadla);
    }
    Nastav uzlu upDownNode rotáciu podľa uhla „uhol natočenia“;
}
```

Keďže lietadlo klesá, iba ak je rýchlosť menšia ako 100, z výpočtu vychádza, že v kosínuse vzorca sa budú nachádzať čísla od 0 po π/2. Vychádzajúc z grafu kosínusu to znamená, že lietadlo sa bude najviac naklánať dole pri rýchlosti 0, kde kosínus dosahuje maximum, zatiaľ čo pri hodnote π/2 dosahuje kosínus minimum a to 0, čiže lietadlo prestane klesať úplne.



Obrázok 6 – Graf nakláňania lietadla (kosínus)

Ak sa lietadlo otočí dolu hlavou, začne sa lietadlo naklánať opačne, preto je vo vetve `inak` znamienko mínus pred celým vzorcom.

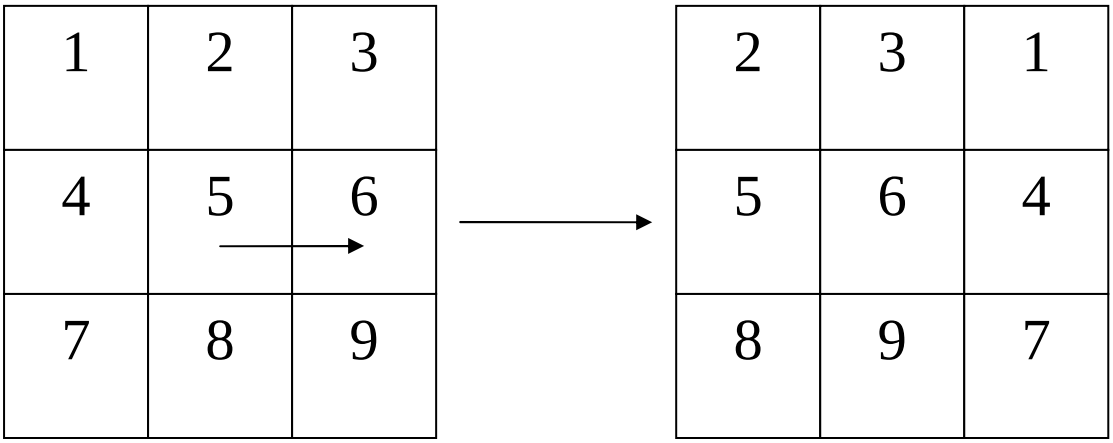
7.1.4 Kamera

Celková kamera scény sa viaže na uzol `node` objektu lietadlo. Je to z dôvodu, že táto kamera je súčasťou objektu tvoreného triedou ChaseCamera. Takáto kamera sleduje pohyb objektu v scéne podľa zadaných vlasností.

Vlastnosti objektu ChaseCamera v simulátore sú nadstavené tak, aby maximálna vzdialenosť kamery od lietadla bola 55 a minimálna 15. Kamera sa prirodzene pohybuje vzhľadom na lietadlo podľa jeho rýchlosti, čiže čím väčšia rýchlosť, tým je kamera ďalej. Samotná kamera sa dá ovládať aj myšou, čím môžeme točiť kameru okolo lietadla.

7.2 Objekt - terén

Keďže ide o letecký simulátor, mala by byť aplikácia schopná zobraziť čo najväčšiu scénu. Preto som zvolil možnosť teoreticky neobmedzenej scény. Je to riešené tak, že terén sa skladá z rovnako veľkých 9 častí – dlaždíc, ktoré na seba nadväzujú a sú v tvare štvorca. Lietadlo sa nachádza presne nad tou strednou dlaždicou a pri prechode na ďalšiu dlaždicu sa krajné dlaždice preložia tak, aby lietadlo bolo opäť v strednej.



Obrázok 7 – Ukážka preklápania terénu

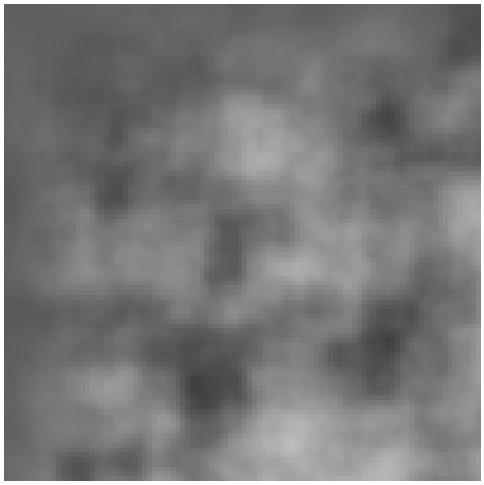
Napr. pri prechode zľava doprava, keď je v strede dlaždica číslo 5, sa presunú dlaždice číslo 1, 4 a 7 (viď. Obrázok 7 – Ukážka preklápania terénu). Lietadlo sa bude nachádzať opäť v strede štvorca, tentokrát ale nad dlaždicou číslo 6. Obdobne to funguje pri prechode do každého smeru. Týmto je možné zaručiť teoretickú nekonečnosť terénu, i keď za cenu jeho opakovania, čo však nie je až tak veľká strata vzhľadom na veľkosť dlaždíc.

7.2.1 Architektúra

Ako bolo povedané, terén je robený ako 9 na seba nadväzujúcich dlaždíc, ktoré sú usporiadané do štvorca. Takýto terén reprezentuje dvojrozmerné pole objektov triedy `TerrainBlock`. Objekty tejto triedy vytvárajú terén z pola integerov – čím vyššie číslo, tým vyšší kopec. Vďaka tomu je možné „prišiť“ hrany dlaždíc k sebe tak, aby na seba nadväzovali, priradením krajných hodnôt jednej dlaždice ku krajným hodnotám vedľajšej dlaždice.

Jednotlivé dlaždice sú vytvorené pomocou čiernobielych výškových máp podobným perlinovmu šumu. Sú to vlastne čiernobiele 16-bitové obrázky vo formáte RAW, o ktorých načítanie sa stará objekt z triedy `RawHeightMap`.

Dlaždice terénu sú priamo pripojené na koreňový uzol celej scény – uzol `root`.



Obrázok 8 – Výšková mapa

Podľa obrázku (Obrázok 8 – Výšková mapa) sa generuje výsledný terén a to tak, že čím belšia farba, tým vyšší kopec.

7.2.2 Textúry

V aplikácii sú dva druhy textúrovania terénu. Predvolene je nadstavené tzv. procedurálne textúrovanie, prepnutím je možné nastaviť tzv. splatting.

7.2.2.1 Procedurálne textúrovanie

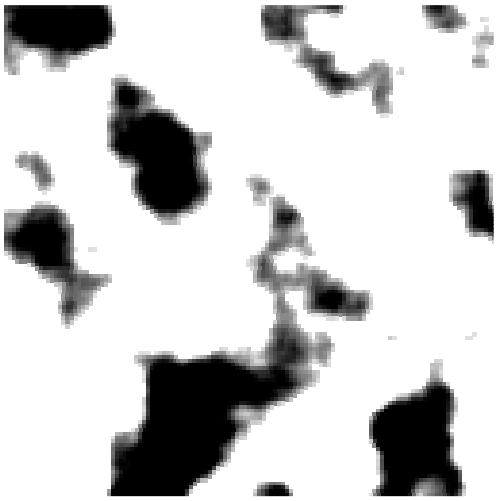
Procedurálne textúrovanie znamená, že textúry sa na terén namapujú podľa zadanej výšky. V tomto prípade je terén rozdelený na tri časti podľa výsky – najnižšia, stredná a najvyššia. Na najnižšej časti je namapovaná textúra trávy, v strednej časti je namapovaná textúra hliny a na najvyššej je textúra snehu. Tieto textúry nemajú ostré hranice prechodu. Navzájom sa čiastočne prekrývajú a sem-tam sa môže textúra zobraziť aj na inej. Takýto plynulý prechod zaobstará objekt z triedy `ProceduralTextureGenerator`.

7.2.2.2 Splatting

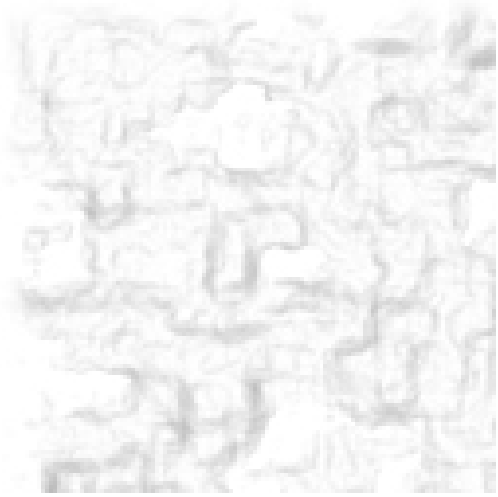
Takéto textúrovanie funguje podobne ako generovanie terénu podľa výškovej mapy. Textúra sa namapuje iba na to miesto, ktoré mu zadáme podľa obrázka v odtieni šedej. Týmto sa dá dosiahnuť oveľa prirodzenejšieho výsledného efektu, avšak za cenu menšieho výkonu. Vďaka splattingu je možné namapovať také textúry ako napr. cesta, chodník a podobné veci, ktoré s terénom a jeho výškou prakticky nesúvisia [14].

Základnou texúrou, ktorá sa namapuje na celý terén, je textúra hliny. Na nej je ako ďalšia vrstva splattingová textúra mŕtvej trávy a následne textúra zelenej trávy. Tieto textúry boli vytvorené v programe FreeWorld 3D, kde je možné splatting vygenerovať podľa výšky terénu podobne ako pri procedurálnom textúrovaní, alebo je možné priamo na terén kresliť.

Splattingom je možné taktiež pridať terénu statické tiene pomocou tzv. lightmapy. Tá zaobstará tiene na teréne, ktoré sú vygenerované pri spúšťaní aplikácie, takže to ušetrí veľa na výkone oproti dynamickým tieňom. Opäť je to len obrázok s odtieňami šedej, ktorý ukazuje, kde má byť terén tmavší. Zmienená lightmapa bola taktiež vytvorená pomocou programu FreeWorld 3D.



Obrázok 9 – Mapa pre splatting textúru



Obrázok 10 – Lightmapa

Textúrovanie terénu je možné prepínať pomocou čísiel 3 (procedurálne textúrovanie) a 4 (splatting).

7.2.3 Kolízie lietadla s terénom

Lietadlo obsahuje 3 tzv. hitboxy (na konci krídiel a na vrtuli), ktoré slúžia na detekciu kolízií s terénom. Pozícia týchto hitboxov sa kontroluje každým vykreslením snímku a ak sa nachádza pod terénom na daných x,z súradniciach, dôjde k výbuchu. To, či je hitbox pod terénom, kontroluje metóda objektu z triedy `TerrainBlock` – `getHeight(x,z)`, ktorá vracia y-súradnicu terénu v danom bode, čiže jeho výšku.

Akonáhle dôjde k výbuchu lietadla, t.j. jeden z hitboxov sa dostal pod terén, odpojí sa uzol `upDownNode`, ktorý obsahuje model lietadla, od uzla `rotationNode` v grafe lietadla a vyvolá sa výbuch častíc v uzle `particleNode` metódou `forceRespawn()`.

7.2.4 Voda

Voda síce nie je priamo pripojená k terénu, je pripojená, takisto ako terén, ku koreňovému uzlu scény. Vzhľadom ale na vlastnosti a účel vody v aplikácii, má voda veľmi podobné vlastnosti ako terén.

Vodná hladina v aplikácii je tvorená zložením objektov z tried `Quad` a `WaterRenderPass`. `Quad` reprezentuje vodnú hladinu, je to vlastne jednoduchá doska. `WaterRenderPass` vykresľuje pohybujúce (vlniace) sa textúry vody. K tomuto objektu je pripojený reflekčný uzol, v ktorom sa nachádzajú objekty, ktoré sa majú vo vode odrážať (napr. obloha). Celý objekt `Quad` je umiestnený do výšky 70.

O vlnenie textúry vody a pocitu „nekonečnej“ vody sa starajú metódy `setVertexCoords(float x, float y, float z)` – stará sa o „nekonečnú“ rozlohu `Quadu`, a `setTextureCoords(int buffer, float x, float y, float textureScale )`, ktorá zaobstará textúrovanie `Quadu` a vlnenie. Tieto metódy sa volajú každým vykreslením snímku.

Náraz lietadla do vody sa kontroluje obdobne ako pri náraze do terénu (ak je niektorý z hitboxov v menšej výške ako 70, dôjde k stretu s vodou). Pri tomto náraze sa odpojí uzol `upDownNode` od uzla `rotationNode` v grafe lietadla a vyvolá sa šplech, ktorý je vytvorený pomocou častíc (particles) v uzle `particleSplashNode`.

7.3 Prostredie

Samotné prostredie hry sa skladá z niekoľkých častí. Okrem zmieneného lietadla, terénu a vodnej hladiny, scéna obsahuje aj stromy a budovy.

7.3.1 Stromy

V aplikácii sa nachádzajú dva druhy stromov a to ihličnatý a listnatý. Pre jednoduchosť a rýchlosť hry sú stromy reprezentované iba pomocou základných geometrických útvarov – guľa, kváder a ihlan. Listnatý strom sa skladá z kvádra (kmeň stromu) a gule (koruna stromu), podobne ako ihličnatý s výnimkou koruny stromu, ktorá je v tvare ihlana.

Tieto stromy sú po teréne nasadené náhodne a to nasledujúcim spôsobom: najskôr sa úplne náhodne po všetkých dlaždiciach terénu rozmiestni 20 koreňových stromov, ktoré slúžia ako základný strom pre les. Či je to listnatý alebo ihličnatý strom je s pravdepodobnosťou ½. Následne sa pre každý koreňový strom vygeneruje náhodne 15 okolitých stromov. Opäť tu platí ½ pravdepodobnosť

listnatého, respektíve ihličnatého stromu. Týmto algoritmom je možné dosiahnuť aspoň sčasti pocit prirodzených lesíkov.

Všetky stromy sú uložené v dvojrozmernom poli, podobne ako terén. Je to kvôli tomu, aby sa pri preklápaní terénu mohli rovnako preklopiť aj stromy na daných dlaždiciach.

Koruny ihličnatých stromov majú pre urýchlenie (a demonštrovanie) vlastnosť LOD (Level Of Detail). V aplikácii je použitý konkrétne cLOD (Continous LOD), ktorý používa metódu redukcie polygónov užitím progresívnych sietí. Využíva k tomu Garland-Heckbertov algoritmus [18], ktorý vytvorí z cieleného modelu sekvenciu modelov od najvyššieho rozlíšenia po najmenšie (s jednou redukciou polygónu na jeden krok).

Pre úsporu pamäte sú všetky stromy robené ako objekty z triedy SharedNode. Je to metóda zdielaného použitia objektov.

7.3.2 Budovy

Budovy v aplikácii slúžia ako cieľ streľby, preto sú robené ako samostatný objekt. Vďaka tomu každá budova môže obsahovať svoj vlastný život (výdrž), ktorý je nastavený na hodnotu 100.

Budovy sú po scéne rozmiestné náhodne, podobne ako stromy s výnimkou tvorenia lesíkov.

Pre aspoň čiastočnú prirodzenosť sa pri výbere budovy vyberá náhodne z piatich textúr stien budovy a z dvoch textúr pre strechu budovy. Ďalej sa náhodne volia aj rozmery budovy.

Takisto ako pri stromoch, aj budovy sú uložené v dvojrozmernom poli ako terén, aby sa pri preklápaní mohli jednoducho preklopiť aj budovy.

Všetky budovy majú nastavené hranice pomocou metódy `setModelBound`, aby bolo možné detekovať zásah od strely.

7.4 Streľba

V aplikácii je možné strieľať dvoma typmi zbraní a síce guľometom a raketometom. Účelom oboch zbraní je zasiahnuť budovu. Jeden náboj (kváder) guľometu berie 5 % z predvolenej výdrže budovy, zatiaľ čo raketa berie 1/3.

7.4.1 Guľomet

Guľomet je reprezentovaný internou triedou `FireBullet`, ktorá vytvorí objekt pozostávajúci z dvoch kvádrov, ktorých pozíciu nastaví na konce krídiel lietadla. Týmto kvádrom je potrebné zvoliť hranice, aby bolo možné detekovať, či netrafili cieľ. V jME na to slúži metóda `setModelBound`.

Nakoniec sa týmto nábojom pridá kontrolér, ktorý je tvorený internou triedou `BulletMover`, ktorá pohyb náboja riadi. Táto trieda je dedená z triedy `Controller`, preto bolo potrebné naimplementovať triedu `update`, ktorá sa volá každým vykreslením snímku. Pri vyrobení objektu

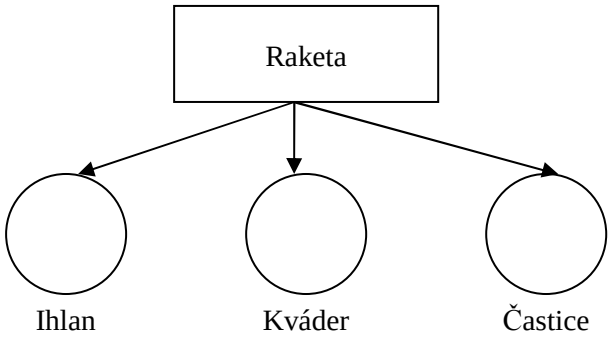
z tejto triedy sa objektu nastavia vlastnosti ako rýchlosť letu, smer letu a čas letu. Pri guľomete je rýchlosť nastavená na 700. Po piatich sekundách sa náboj odpojí zo scény.

Strielať z guľometu je možné pomocou klávesy `space`, ktorá má vlastnosť `allowRepeats` nastavenú na hodnotu `true`, aby bolo možé automaticky strieľať pri držaní klávesy `space`.

7.4.2 Rakety

Raketa je na rozdiel od guľometu robená ako samostatný objekt. Pozostáva totiž z viac častí: jednoduchý kváder ako telo rakety, ihlan ako špička rakety a častice (particles) pre plameň vychádzajúci z rakety.

Objekt vnútornej triedy `FireRocket` vyrobí takúto raketu a nastaví jej pozíciu na vrtuľu lietada. Následne objektu triedy `FireRocket` priradí kontrolér, podobne ako pri guľomete z triedy `BulletMover`. Rýchlosť rakety je 170 a takisto ako náboj z guľometu aj raketa po piatich sekundách zmizne zo scény. Samozrejmosťou sú nastavené hranice rakety, aby bolo možné detekovať náraz.



Obrázok 11 – Graf scény - Raketa

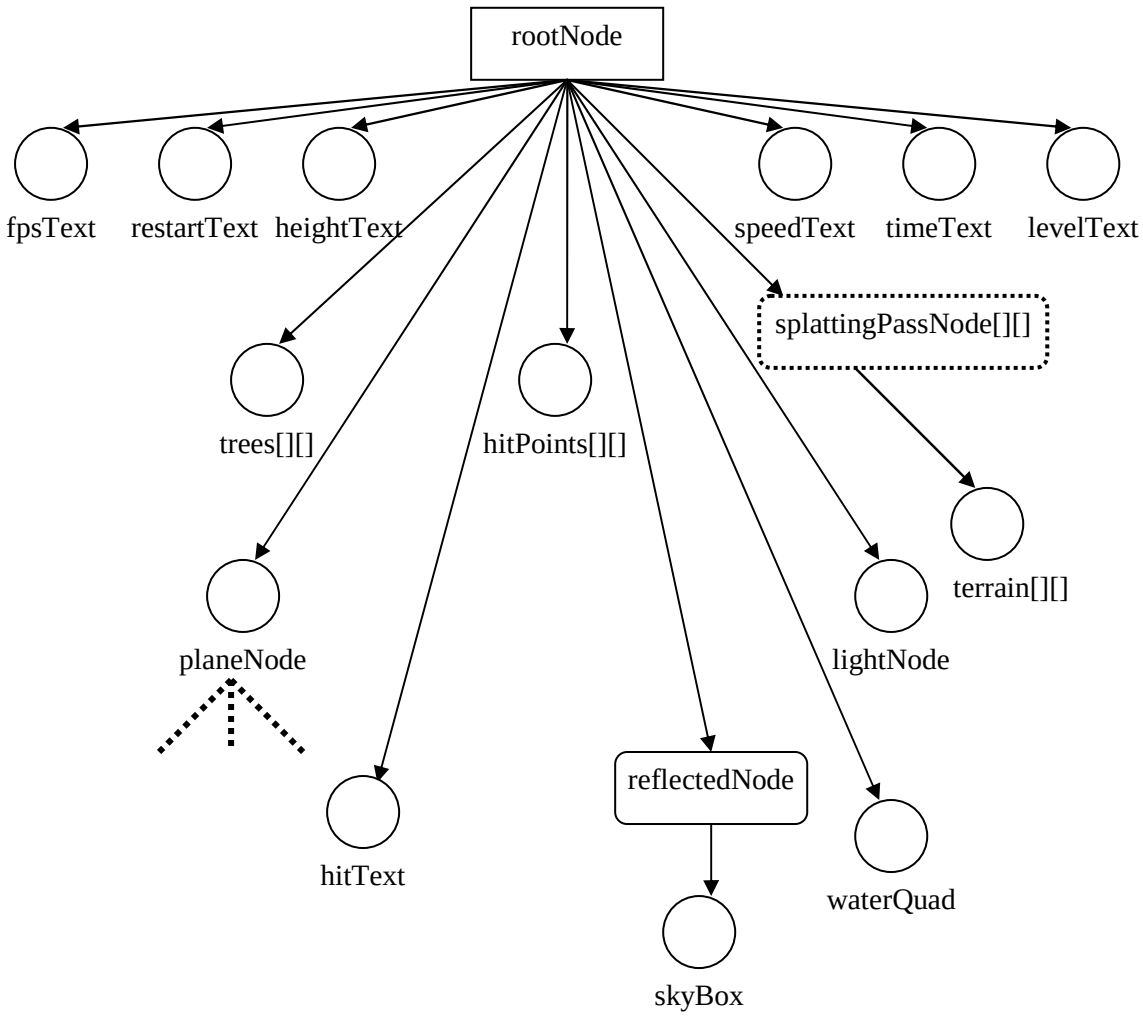
7.5 Objekt - Main

Hlavná trieda, z ktorej vzniká celá hra je trieda `Main`. Celá trieda dedí triedu `BaseGame` z knižnice jME, ktorá zaručuje volania tried `update` alebo `render` pri vykresľovaní snímku. Ďalej je potrebné naimplementovať triedy `initSystem` a `initGame`, ktoré zaobstarajú inicializáciu systému, respektíve načítanie hry (načítanie modelov, prostredia...).

V medóde `update` sa aktualizujú prvky systému – transformácie objetov, sledovanie vstupov z klávesnice, aktualizácia zvuku a pod. V metóde `render` sa daná scéna vykresľuje. Ďalšia metóda – `initSystem` slúži na inicializáciu systémového jadra hry – napr. kamera, rozlíšenie, klávesové vstupy. Rozlíšenie je prebrané z objektu triedy `ProperitiesIO`, čo je vlastne tabuľka s výberom rozlíšenia a farebnej hĺbky pred výberom hry. Metóda `initGame` len načítava, čo všetko sa má v scéne zobraziť.

Kvôli vykresľovaniu terénu v diaľke je celá scéna ponorená do jemnej hmly, tvorenej metódou `buildFog`. Farba hmly je šedá (RGB – 0.5, 0.5, 0.5) a rozprestiera sa od vzdialenosti 50 až 1000 od pozície kamery. Hustota hmly je 0.05.

7.5.1 Graf scény



Obrázok 12 – Graf scény – Main

Ako je vidieť z obrázku, väčšina objektov v scéne je pripojených priamo na hlavný koreňový uzol. Je to preto, lebo sú to také objekty, s ktorými nie je potrebné robiť žiadne transformácie, preto nepotrebujú transformačný uzol. Jediným rozdielom je objekt `skyBox`, čo je vlastne dutá škatuľa so stredom v kamere a zvnútra otextúrovaná textúrou oblohy. Objekt `skyBox` sa pohybuje vždy zároveň s kamerou, ktorá ostáva v jeho strede a je pripojený na uzol `reflectedNode` z dôvodu odrazu oblohy vo vode.

Čo sa týka terénu, pri procedurálnom textúrovaní je pripojený priamo na koreňový uzol `rootNode`, ale pri splatting textúrovaní je medzi nimi tzv. `splattingPassNode`, ktorý predáva terénu vytvorené splatting textúry.

Uzly `fpsText`, `restartText`, `timeText`, `levelText`, `timeText`, `hitText` a `heightText` sú jednoduché 2D texty z triedy `Text2D`, ktoré slúžia na prezentáciu GUI (Grafické užívateľské rozhranie).

Uzly `trees[][]` a `hitPoints[][]` reprezentujú pole stromov, respektíve budov.

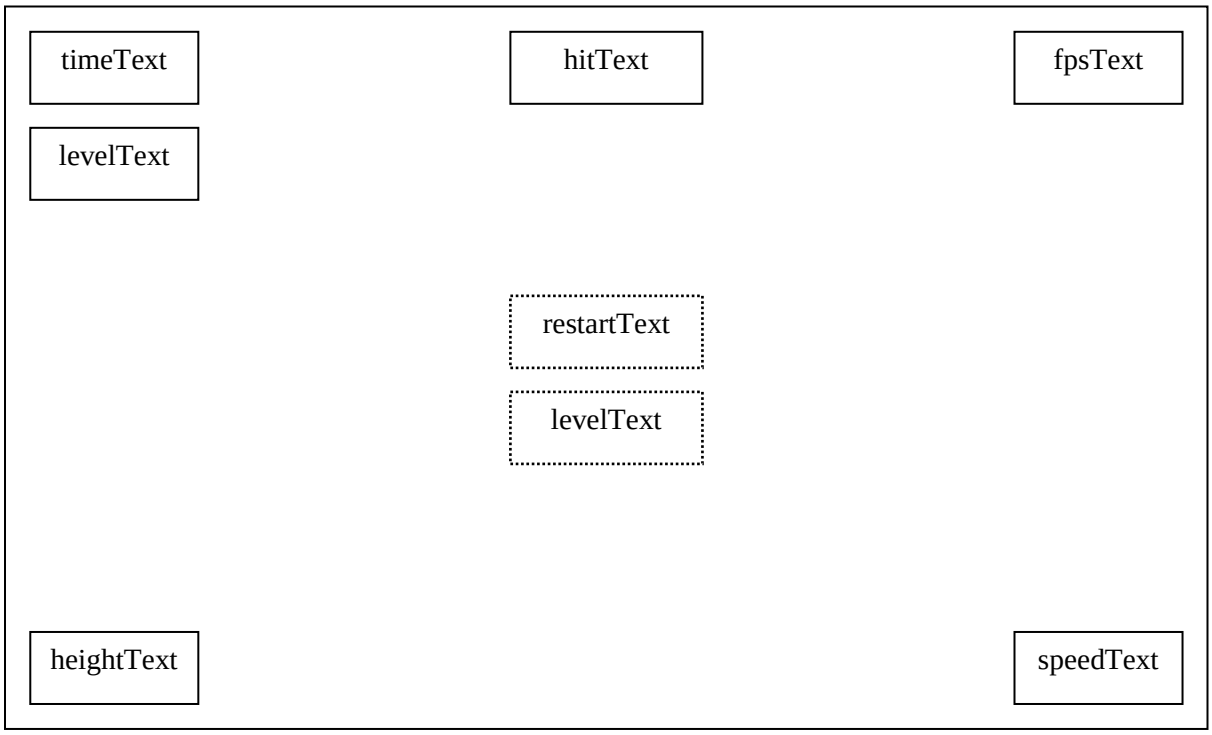
List planeNode je koreňový uzol grafu scény lietadla (viď. Obrázok 4 – Graf scény - lietadlo).

Uzol `reflectedNode` je, ako bolo zmienené vyššie, reflekčný uzol scény, do ktorého sa pridávajú objekty, ktoré by sa mali odrážať vo vode. Z dôvodu obrovských pamäťových nárokov (každý objekt odrážajúci sa vo vode je vlastne nový - taký istý objekt) je k tomuto uzlu pripojená len obloha – `skyBox`. S týmto uzlom čiastočne súvisí aj geometria `waterQuad`, ktorá má na starosti zobrazenie vodnej hladiny.

7.5.2 GUI

Grafické užívateľské rozhranie (GUI) je vytvorené len jednoduchými 2D textami zobrazujúcimi sa na obrazovke.

V pravom hornom rohu sa zobrazuje informácia o FPS, v ľavom hornom rohu je informácia o čase a číslo levelu, v strede hore sa nachádza text zobrazujúci počet zásahov, respektíve dosiahnuté skóre, v pravom dolnom rohu je číslo zobrazujúce rýchlosť lietadla a v ľavom dolnom rohu je výškomer. Pri náraze lietadla alebo pri prehratej hre sa v strede obrazovky zobrazí text s informáciou o reštarte hry. Po dosiahnutí cieľa v danom leveli sa na 3 sekundy zobrazí v strede obrazovky informácia o čísle nového levelu.



Obrázok 13 – GUI

7.5.3 Osvetlenie

Osvetlenie celej scény je riešené v metóde `buildLighting`. Na celú scénu je aplikované ambientné svetlo, čo je svetlo ktoré sa prichádza do scény zo všetkých smerov s rovnakou intenzitou. Je to základná zložka svetla. Ďalšou zložkou svetla je difúzne svetlo, ktoré zaručuje odraz svetla pochádzajúceho z jedného zdroja na objektoch rovnako do všetkých smerov [15].

Súčasťou metódy `buildLighting` je aj vytvorenie známeho javu fotoaparátov – lens flare. Knižnica jME má na tento účel triedy `LensFlare` a `LensFlareFactory`, ktoré pomocou načítaných textúr zmienený lens flare vytvoria. Lens flare je umiestnený vo vzdialenosti asi 70 od lietadla v smere zdroja difúznej zložky svetla a pohybuje sa zároveň s lietadlom.

7.5.4 Zvuky

Čo sa týka zvukov, v knižnici jME sú tvorené veľmi jednoducho. Objekt z triedy `AudioSystem` pomocou objektov z triedy `AudioTrack` načíta zvolené zvuky vo formáte `ogg` alebo `wav`. V aplikácii je to formát `ogg` kvôli úspore miesta. Po načítaní stačí v požadovanom čase zvuk pustiť pomocou metódy objektu z triedy `AudioTrack` – `play`.

Všetky zvuky v hre (výbuch budovy, výbuch lietadla, náraz lietadla do vody, zvuk výhry, zvuk prehry, zvuk strely) sú vytvorené a spúšťané tak, ako je popísané vyššie. Zvuk motora je objekt vytvorený s vlastnosťou `looping` nastavenú na `true`, kvôli opakovanému zvuku motora. Každým zavolaním metódy `update` sa zvuku motora nastaví výška tónu (pitch) podľa rýchlosti lietadla vzorcom:

```
motorPitch = 1 + (rýchlosť_lietadla / 300);
```

Keďže výška tónu môže nadobúdať len hodnoty z intervalu 1 až 2, je potrebné začať počítať od čísla 1. Maximálna rýchlosť lietadla je 150, takže zo vzorca je zrejmé, že maximálna výška tónu bude 1,5.

Keďže jME podporuje 3D zvuk, bolo potrebné zvoliť „uši“ pre príjem zvuku, ktoré sú zvolené na pozíciu kamery.

7.5.5 Anaglyph režim

Súčasťou aplikácie je aj tzv. anaglyphový režim, ktorý čiastočne umožňuje sledovanie hry v 3D móde. Anaglyph je stereo obrázok vytvorený z dvoch obrázkov (nazývajme ich pravý a ľavý), ktoré sú konvertované do navzájom odlíšených farebných rysov/obrysov (modrá a červená). Ďalším rozdielom by mal byť posun "objektu" na ľavom obrázku a pravom. Tento posun by mal byť približne taký, aký je rozdiel pohľadu na daný objekt, ak sa naň pozerá pravým alebo ľavým okom (samostatne). Pre dosiahnutie 3D efektu treba použiť špeciálne modro-červené okuliare [16].

V aplikácii je tento posun medzi „očami“ 1.75. Pre 3D efekt je potrebné, aby sa kamery, ktoré reprezentujú pravé a ľavé oko zbiehali do jedného bodu. Tento bod je nastavený vo vzdialenosti 100 od kamery.

Samotné vykresľovanie pri anaglyph režime nie je tvorené dvoma kamerami, ako by sa mohlo zdať. V jednom cykle vykreslenia sa jedna kamera nastaví na pozíciu ľavého oka, vykreslí scénu na modro a hneď nato sa nastaví na pozíciu pravého oka, kde scény vykreslí na červeno. Týmto je dosiahnutý požadovaný efekt anaglyph módu, ktorý sa v aplikácii zapína číslom 2 a vypína číslom 1.

7.5.6 Princíp hry

Princíp hry je veľmi jednoduchý. Ide o to, strieľať budovy, pričom medzi zostrelením budovy je časový limit, závislý od výšky levelu. Po zostrelení desiatich budov sa vygenerujú budovy úplne nové a zvýši sa level. Čas medzi zostrelením budov je závislý na levele spôsobom:

```
levelTime = (int)30/level;
```

Čiže pri prvom leveli je čas zaokrúhlene na integer 30 sekúnd, pri druhom 15, pri treťom 10 atď. Pri neúspešnosti (vyprší čas) sa text s informáciou o počte trafených budov zmení na výsledné skóre v tvare:

```
"Your score: Level " + level + " & " + hits + " hits"
```

čiže napr.:

```
"Your score: Level 4 & 6 hits"
```

8 Záver

Vlastný prínos

Pri tvorení tejto práce sa mi podarilo preniknúť do tajov tvorenia počítačových hier. Práca mi zároveň objasnila mnohé časti z oblasti tvorenia počítačovej grafiky a to hlavne prácu s matematickými objektami – maticami alebo quaterniónmi. Úspešne som zvládol základy veľmi zaujímavej a perspektívnej knižnice Java Monkey Engine, v ktorej by som sa chcel naďalej rozvíjať. Zadanie práce som splnil vo všetkých bodoch, najmä rozsiahlosť terénu, na ktorú bol kladený dôraz.

Java 3D vs. jME

I keď by sa mohlo zdať, že Java 3D bude mať väčšiu perspektívu, podporu a celkovo bude lepšia, nie je tomu tak. Vo všetkých ohľadoch (okrem záťaže procesoru a miestami aj pamäte) ju jME predbieha a to oveľa. Či už sa to týka rýchlosti alebo vzhľadu (viď Príloha - Testy), ale aj podpory zo strany vývojárov a komfortu programovania.

Java 3D sa už ďalej nevyvíja, zatiaľ čo jME ide neustále dopredu vďaka odhodlanosti mladých ambizióznych programátorov v okruhu Marka Powella. Keďže jME je oproti Jave 3D plne open source, môže si programátor v podstate naprogramovať, čo sám potrebuje.

Podpora Javy 3D je síce celkom dobrá (Java 3D je oveľa známejšia ako jME), jME má len jednu oficiálnu stránku, ale zato veľmi aktívne fórum, na ktoré prispievajú priamo programátori knižnice s rýchlosťou okolo 24 hodín.

jME tak poráža Javu 3D na plnej čiare a pri programovaní 3D aplikácií v Jave má oveľa väčšiu a jasnejšiu budúcnosť. Či to však tak bude, ukáže až samotný čas.

Letecký simulátor

Aplikácia bola testovaná veľkým počtom nezávislých užívateľov. Pre evidenciu chýb sa používal online bug-trackingový systém Mantis [17]. Hra ale stále obsahuje niektoré chyby, ktoré by sa mali časom odstrániť:

1. Strely prechádzajú cez terén (vzhľadom na zložitosť kontrolovania pozície strely voči dlaždiciam terénu je to zanedbateľný problém a rapídne by to spomaľovalo vykresľovanie).
2. Sú vidieť prechody medzi dlaždicami terénu (pravdepodobne by bolo potrebné prerobiť jednu z tried knižnice jME, čo ale nebolo predmetom práce).
3. V niektorých prípadoch sa vyskytli problémy s prehrávaním zvukov. Je to pravdepodobne spôsobené chybou knižnice jME a prepojením vrstvy zvuku na najnižšiu úroveň.

Návrh vylepšenia a pokračovania

V budúcnosti by sa hra mohla rozšíriť o umelú inteligenciu a taktiež o možnosť hrania sieťovej hry. Samozrejmosťou je optimalizácia, ktorá sa kvôli časovým problémom nedala stihnúť. Ďalej by bolo dobré spojiť naimplementované menu s danou aplikáciou.

Literatúra

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

[2] Selman, D.: Java 3D Programming. Mannig Publications Co., Greenwich, 2002. ISBN 1930110359

[3] Lázaro, T., et al.: jME users guide [online]. 2008. Dostupné na URL: <http://www.jmonkeyengine.com/wiki/doku.php?id=user_s_guide>

[4] Twilleager, D.: Java 3D Architecture [online]. 2008. Dostupné na URL <http://java3d.j3d.org/implementation/architecture.html>

[5] Yang, Ch.: Java 3D Performance Guide for J3D 1.3 [online]. 2006. Dostupné na URL <http://java3d.j3d.org/tutorials/quick_fix/perf_guide_1_3.html>

[6] Java 3D Implementation – DirectX vs OpenGL [online]. 2006. Dostupné na URL <http://java3d.j3d.org/implementation/java3d-OpenGLvsDirectX.html>

[7] Taylor, N.: Java 3D Transformations [online]. 2008. Dostupné na URL <http://www.macs.hw.ac.uk/~nkt/graphics/5%20Java3D%20Transformations%20Slides.pdf>

[8] Java [online]. 2008. Dostupné na URL <http://sk.wikipedia.org/wiki/Java>

[9] Scene Graph [online]. Dostupné na URL <http://en.wikipedia.org/wiki/Scene_graph>

[10] Horton, I.: Java 5, Neokortex, Praha, 2007. ISBN 8086330125

[11] Willis, A.R: Scene Graph Basics [online]. 2006. Dostupné na URL <http://www.visionlab.uncc.edu/~arwillis/online/docs/java3d-1_4_0-doc/javax/media/j3d/doc-files/SceneGraphOverview.html>

[12] Lightweight Java Game Library: About [online]. 2009. Dostupné na URL <http://www.lwjgl.org/about.php>

[13] Kršek, P. – Španěl, M.: Základy počítačové grafiky: Geometrické transformace ve 2D a 3D. Brno, 2008.

[14] Texture Splatting [online]. 2008. Dostupné na URL <http://en.wikipedia.org/wiki/Texture_splatting>

[15] Kršek, P. – Španěl, M.: Základy počítačové grafiky: Osvětlení a stínovaní 3D objektů. Brno, 2008.

[16] Anaglyph image [online]. 2008. Dostupné na URL <http://en.wikipedia.org/wiki/Anaglyph_image>

[17] Mantis bugtracking system [online]. 2009. Dostupné na URL < http://www.mantisbt.org>

[18] Continous Level of Detail, Garland-Heckbert-ov algoritmus [online]. 2008. Dostupné na URL <http://www.jmonkeyengine.com/wiki/doku.php?id=clod>

Zoznam príloh

Príloha 1. Testy

Príloha 2. CD – Zložka *JavaDoc* obsahuje vygenerovaný JavaDoc pomocou programu Eclipse
Zložka *Plagat* obsahuje prezentáciu práce prostr. plagátu
Zložka *Prelozene zdrojove subory* obsahuje preložené (class) súbory
Zložka *Spustitelny program* obsahuje spustiteľnú aplikáciu (jar)
Zložka *Zdrojove subory* obsahuje java súbory k aplikácii

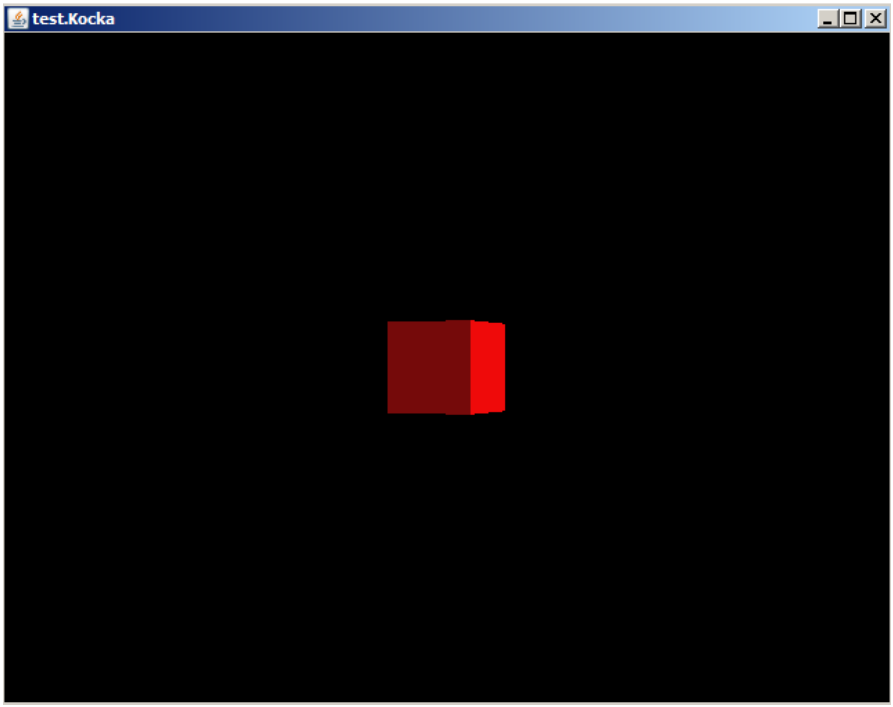
1 Príloha - Testy

V tejto prílohe budú zhrnuté poznatky a otestované obidve knižnice na jednoduchých príkladoch – tzv. „HelloWorld“ (rotujúca kocka), 500 kociek, 500 kociek textúrovaných, viac – polygónový model – textúrovaný a ukážka terénu.

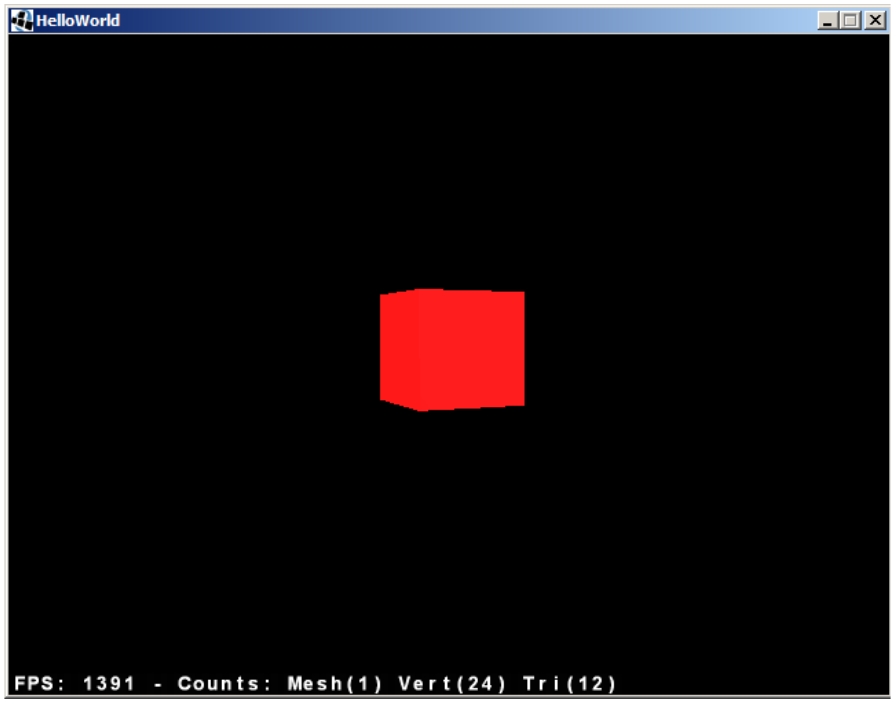
Všetky testy boli robené na počítači HP pavilion DV5-1160 s grafickou kartou Nvidia GeForce 9600 GT mobile s rozlíšením 640x480. Pre testovanie záťaže procesoru a pamäte bola použitá aplikácia od tvorcov z dielne Sun – JConsole.

1.1 HelloWorld

Aplikácia rotujúcej kocky:



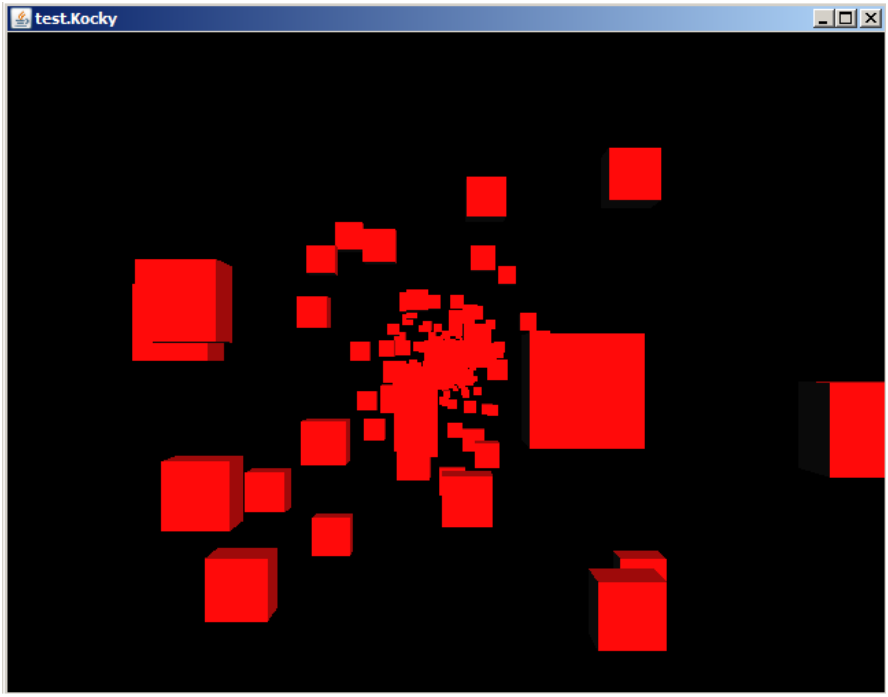
Obrázok 14 – HelloWorld – Java 3D



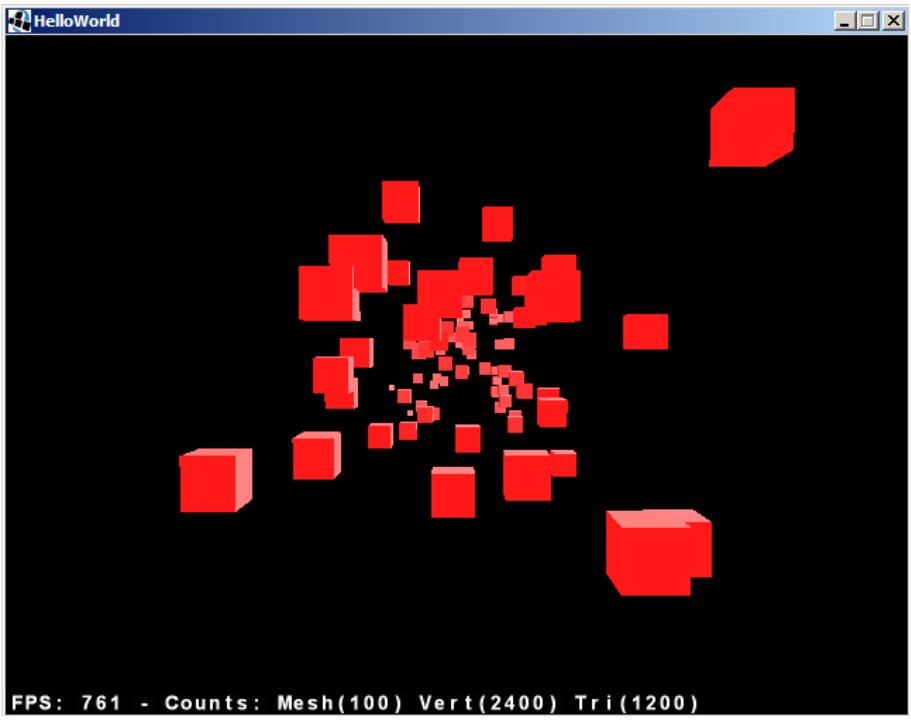
Obrázok 15 – HelloWorld - jME

1.2 500 kociek

Aplikácia zobrazujúca 500 kociek naraz:



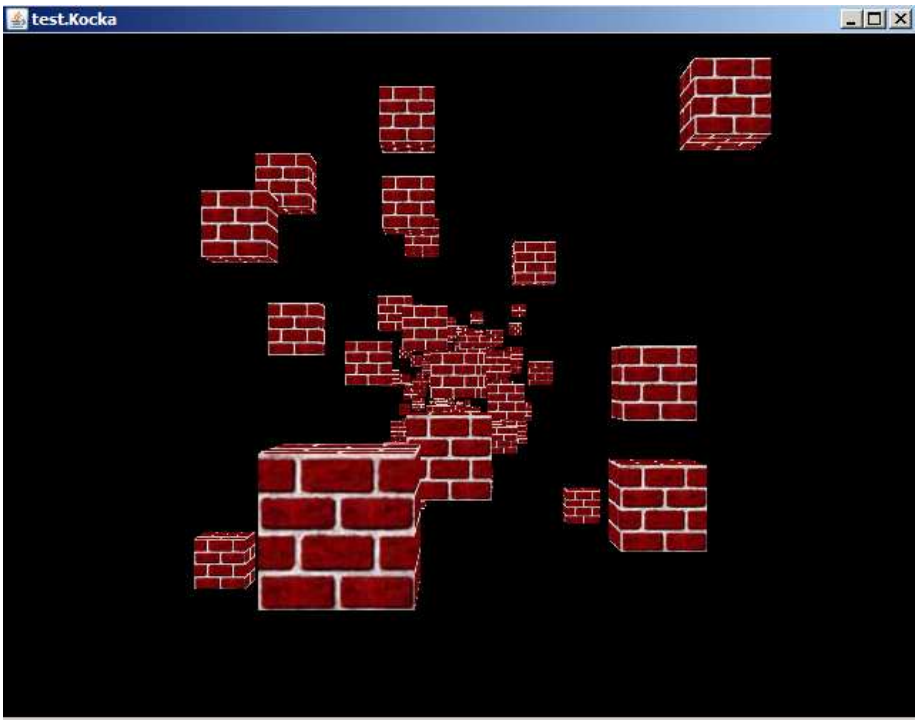
Obrázok 16 – 500 kociek – Java 3D



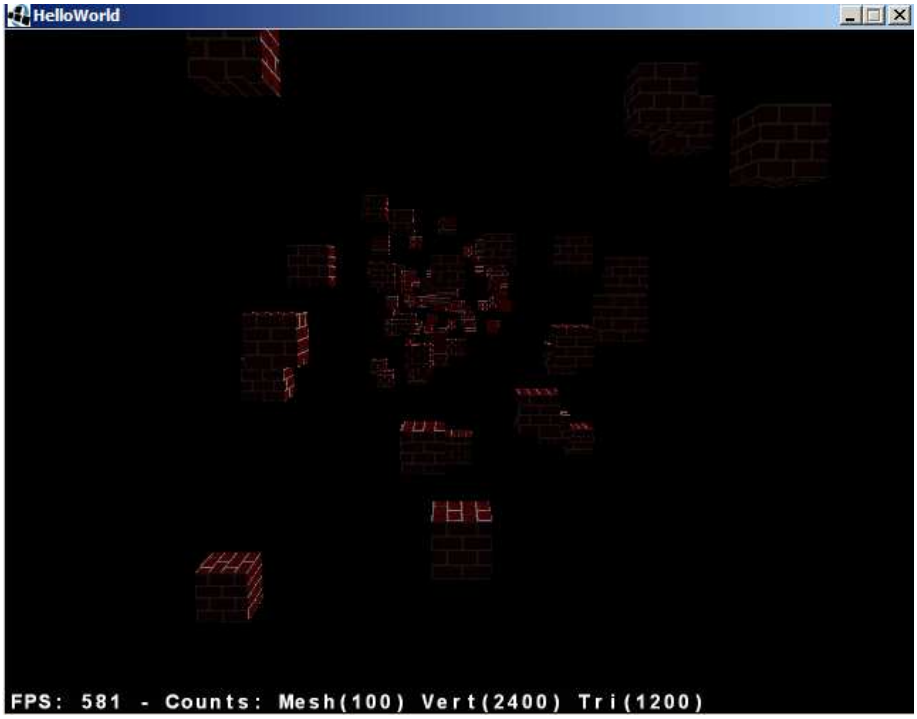
Obrázok 17 – 500 kociek - jME

1.3 500 kociek + textúry

Aplikácia zobrazujúca 500 kociek naraz s textúrami:



Obrázok 18 – 500 kociek + textúry – Java 3D



Obrázok 19 - 500 kociek + textúry – jME

1.4 Viac – polygónový model + textúry

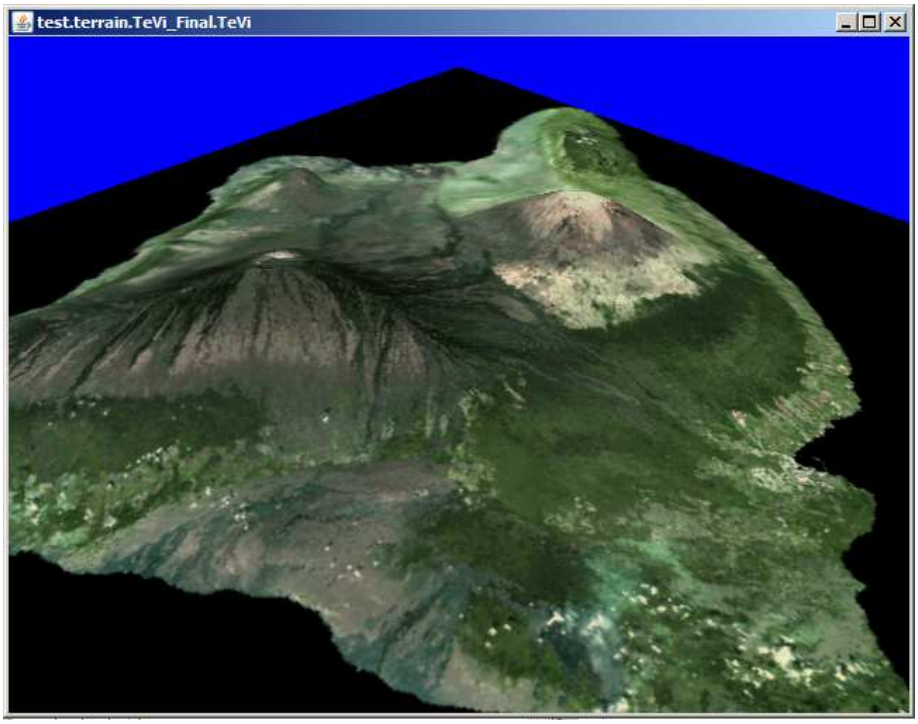
Aplikácia zobrazujúca viac-polygónový model a textúry:

Obrázok 20 – Viac-polygónový model + textúry – Java 3D

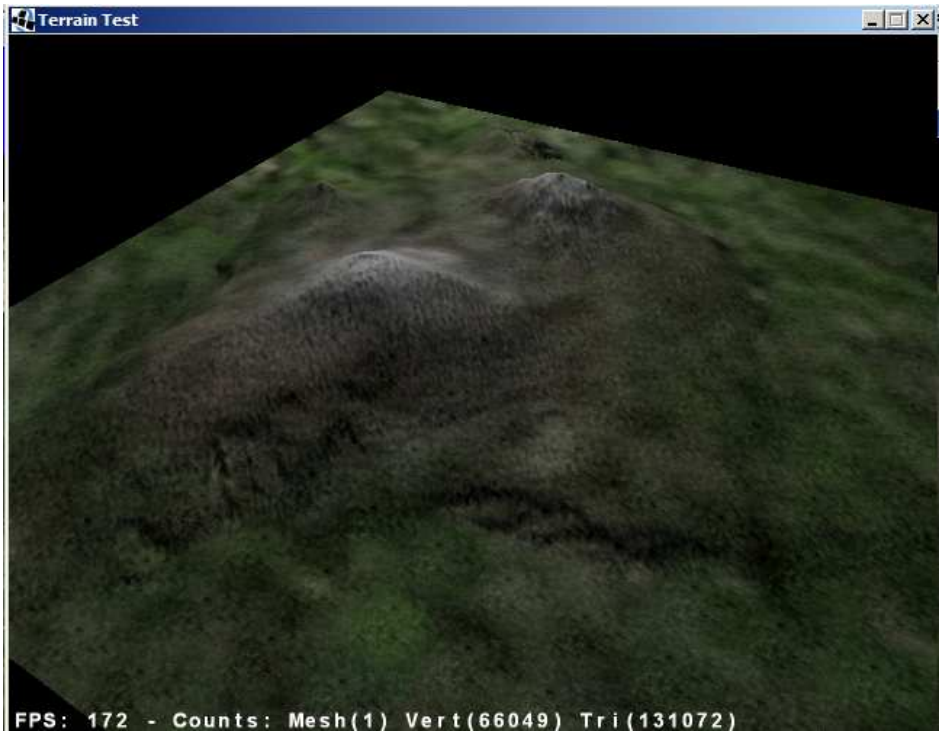
Obrázok 21 – Viac-polygónový model + textúry – jME

1.5 Ukážka terénu

Aplikácia zobrazujúca terén vygenerovaný na základe výškovej mapy:



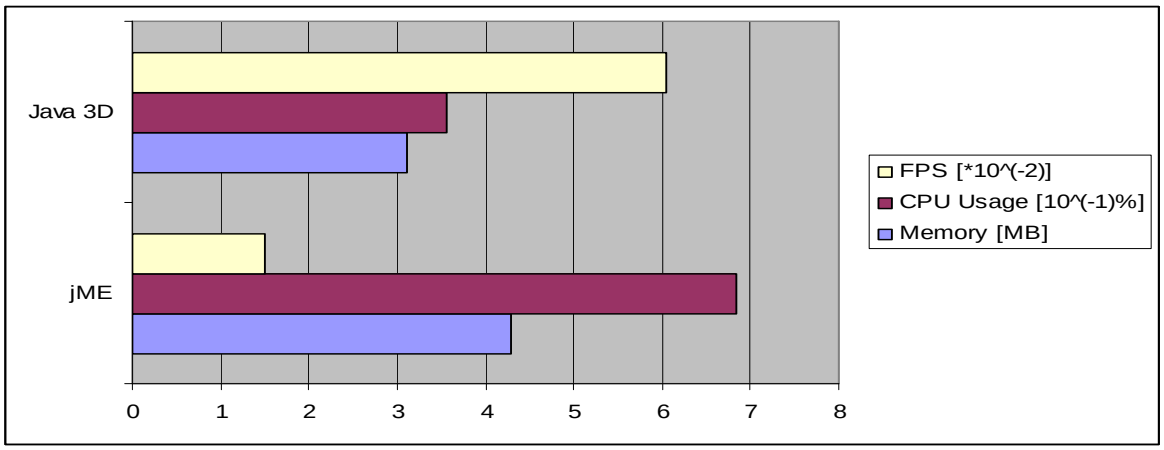
Obrázok 22 – Terén – Java 3D



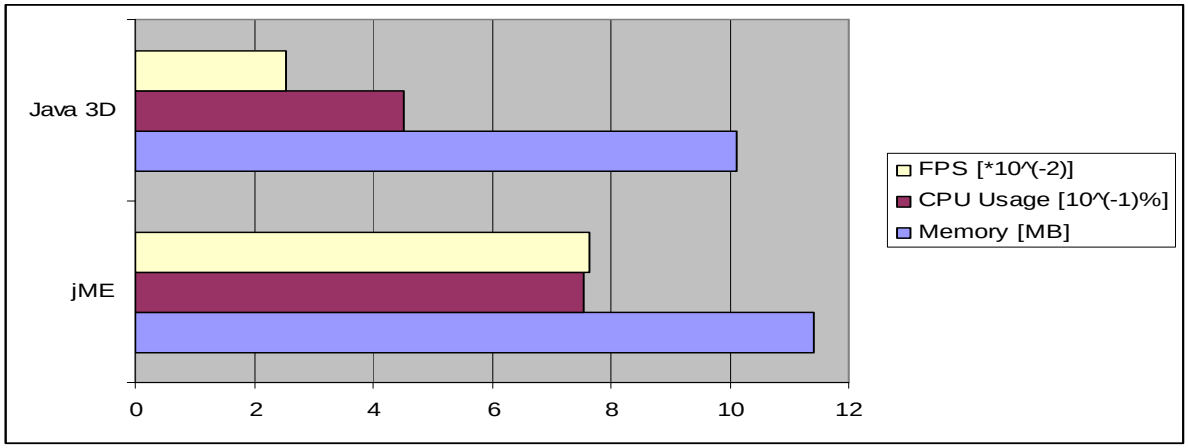
Obrázok 23 – Terén - jME

1.6 Grafy – výkon

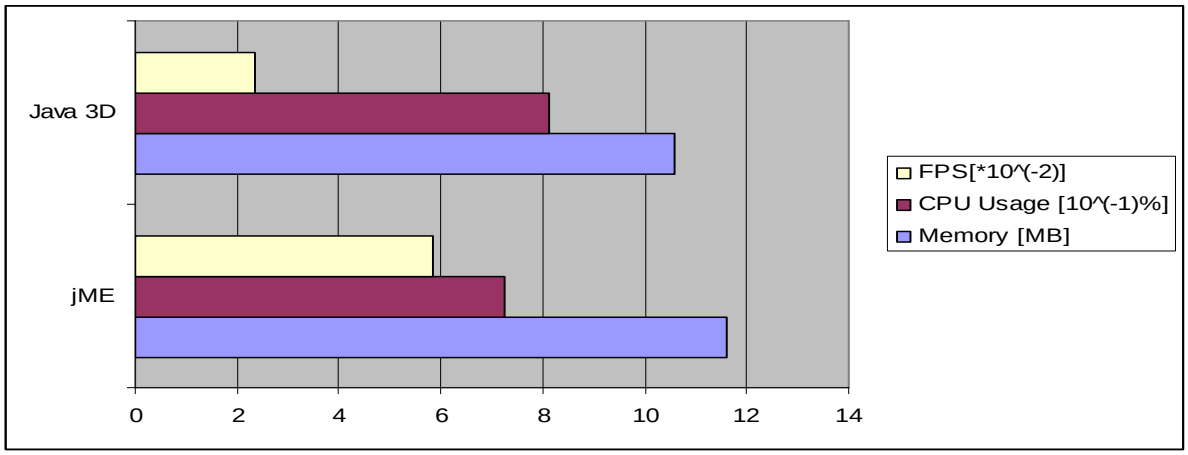
1. **HelloWorld**



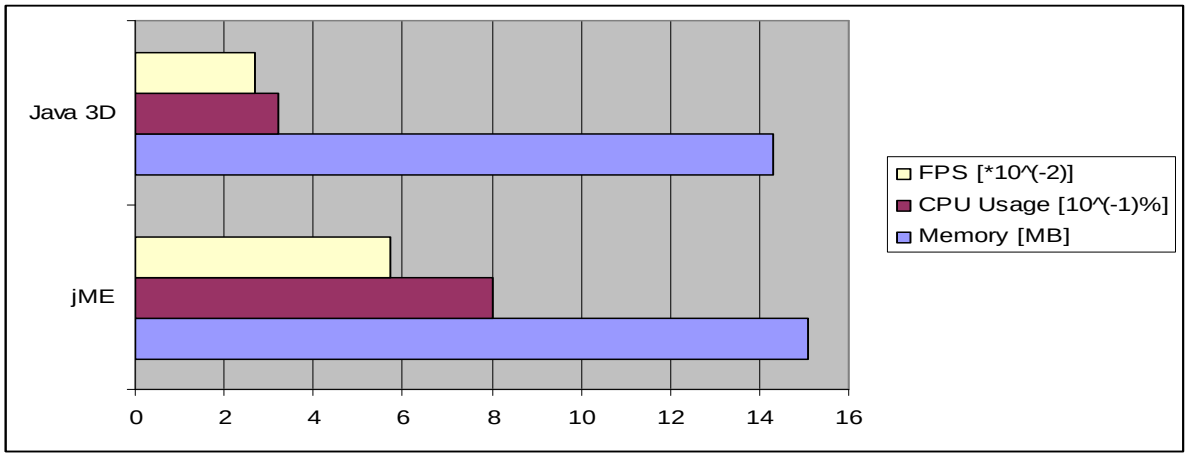
2. 500 kociek



3. 500 kociek + textúry



4. Viac-polygónový model + textúry



5. Ukážka terénu