VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

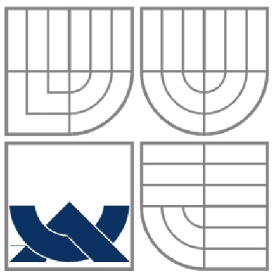
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JEDNODUCHÁ HRA VYUŽÍVAJÍCÍ ROZHRANÍ JAVA3D

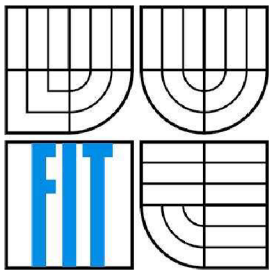
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Jakub Obr
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JEDNODUCHÁ HRA VYUŽÍVAJÍCÍ ROZHRANÍ JAVA3D

SIMPLE GAME USING JAVA3D API

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — Jakub Obr
AUTHOR

VEDOUCÍ PRÁCE — Ing. Vítězslav Beran
SUPERVISOR

BRNO 2007

Abstrakt

Tato práce je zaměřena na aplikační rozhraní Java 3D a jeho použití pro tvorbu herních aplikací. Obsahuje popis tohoto rozhraní a stručný popis jeho alternativ. Dále uvádí návrh testů pro toto rozhraní a výsledky těchto testů. V závěru popisuje návrh jednoduché hry využívající toto rozhraní a hlavní implementační body tohoto návrhu.

Klíčová slova

Java 3D, programování her, vývoj her, testování výkonu, OpenGL, graf scény, Xith3D, Java Monkey Engine

Abstract

This thesis concentrates on an application interface Java 3D and its usability for game programming. It presents description of this interface and a short presentment of its alternates. It also presents testing scheme for this interface and the tests results. In the end it shows simple game concept using this interface and main implementation features of this concept.

Keywords

Java 3D, game programming, game development, performance testing, OpenGL, scene graph, Xith3D, Java Monkey Engine

Citace

Obr Jakub: Jednoduchá hra využívající rozhraní Java3D. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Jednoduchá hra využívající rozhraní Java3D

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vítězslava Berana. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jméno Příjmení
Datum

Poděkování

Chtěl bych poděkovat svému vedoucímu, Ing. Vítězslavu Beranovi, za odbornou pomoc, vstřícnost a ochotu pří práci na tomto projektu.

Obsah

Úvod

Ve světe vývoje počítačových her dnes stále dominuje programovací jazyk C++, avšak v této oblasti se začíná prosazovat i jeho nemalý konkurent v doméně objektově orientovaného programování, jazyk Java. Ten byl ještě nedávno velice zavrhován pro svou nízkou rychlost a přestože dnes již dosahuje obdobných výpočetních výkonů jako jeho kompilovaný protějšek C++[7], je na něj stále nahlíženo s jistými předsudky. Nespornou výhodou jazyka Java je však jeho platformová nezávislost a rychlost vývoje aplikací v něm. Dnes má Java k dispozici již několik knihoven pro práci s grafikou, jednou z nichž je i knihovna pro obecné použití, Java 3D, poskytující vysokoúrovňové rozhraní pro práci s 3D grafikou. Od své první specifikace v roce 1997 se dnes dostala do verze 1.5 a účelem této práce je otestovat její použitelnost pro tvorbu her.

Mimo popisu knihovny Java 3D v druhé kapitole tato práce nejprve obsahuje stručné seznámení s alternativami k ní z oblasti 3D her. Dále je ve třetí kapitole navržena sada testů pro otestování rozhraní Java 3D, stručně popsána jejich implementace a nakonec jsou výsledky provedených testů zhodnoceny. Do práce je přidáno grafické znázornění některých výsledků testů. Kompletní výpis výsledných grafů je možné nalézt v příloze.

Cílem práce je také vytvoření hry s využitím rozhraní Java 3D. Ve čtvrté kapitole práce je zahrnut návrh této hry a její hlavní implementační rysy. Zdrojové texty testovací aplikace a vytvořené hry je možné nalézt na přiloženém CD.

V závěru práce jsou shrnuty její výsledky a je vyhodnocena využitelnost Javy 3D pro tvorbu herních aplikací a je diskutováno možné další pokračování v rámci diplomové práce.

1 Dostupné prostředky pro 3D grafiku v Javě

V současnosti existuje několik alternativ ke knihovně Java 3D. Lze je rozdělit na dvě hlavní skupiny: nízko-úrovňové rozhraní vázající jazyk Java na OpenGL a rozhraní s vlastním grafem scény.

1.1 Java vázaná na OpenGL

Hlavními zastupiteli nízko-úrovňových rozhraní vázající jazyk Java na OpenGL jsou LWJGL a JOGL.

1.1.1 LWJGL

Lightweight Java Game Library je rozhraní, pracující s OpenGL 1.5, určené pro profesionální i amatérské programátory. Protože pracuje s nejnovější verzí Javy, podporuje použití NIO[1] a celoobrazovkového režimu (oboje představeno v Javě 1.4). Nepodporuje však AWT, ani Swing a pro zobrazování si vytváří vlastní okno. Vazbu na OpenGL zajišťuje podobně jako JOGL(viz níže) mapováním OpenGL funkcí z jazyka C do Javy. Stabilní verze 1.0 z ledna roku 2007 byla vytvořena s důrazem na výkon, jednoduchost a přenositelnost.

1.1.2 JOGL

Nejnovější vazbou Javy na OpenGL je rozhraní Java OpenGL. Stejně jako LWJGL podporuje nejnovější verzi Javy a OpenGL, je však integrován do knihoven AWT a Swing a je znatelně rozsáhlejší. Pro přístup k rozhraní OpenGL a jeho rozšíření JOGL využívá volání JNI[2]. Díky tomu je sice struktura metod souvisejících s JOGL odlišná od ostatních metod v aplikaci, ovšem pro programátory seznámenými s OpenGL z jazyka C je mnohem jednodušší pro použití. Přístup využití volání JNI umožňuje například pomocí textur generovat nejrůznější typy stínů.

1.2 Rozhraní s grafem scény

Dalšími zastupiteli programování ve 3D v Javě jsou aplikační rozhraní pro práci s grafem s podporou nízko-úrovňové vazby na OpenGL.

---

[1] New I/O je vylepšená síťová služba v Javě

[2] Java Native Interface je framework umožňující volat v Javě aplikace a knihovny napsané v jiném jazyce [15]

1.2.1 Xith3D

Tato knihovna používá pro graf scény stejný základ jako knihovna Java 3D. Na rozdíl od Javy 3D však umí volat operace rozhraní OpenGL přímo. Protože je API Xith3D velmi podobné Javě 3D, je přenos programů napsaných pomocí knihovny Java 3D do prostředí knihovny Xith3D poměrně jednoduché. Systém Xith3D dokáže pracovat například nad rozhraním JOGL, JGL a JSR 231[3].

1.2.2 Grafický systém jME

Java Monkey Engine je systém pro práci s grafem scény inspirovaný grafem z knihy 3D Game Engine Design od Davida H. Eberlye. JME je postaveno na LWJGL a plánována je i podpora pro rozhraní JOGL.

[3] Specifikační proces společnost Sun, jehož součástí je vazba Javy na OpenGL. Vychází z rozhraní JOGL. JSR 231 se stane oficiální vazbou Javy na OpenGL. Více v [13]

2 Popis rozhraní Java 3D

Knihovna Java 3D tvoří aplikační rozhraní pro práci s grafem scény a umožňuje nám jeho tvorbu, vykreslení a manipulaci s ním na vysoké úrovni. Je k dispozici ve dvou variantách pracujících buď nad rozhraním OpenGL nebo nad DirectX Graphics. Ve své distribuci je šířena s oběma verzemi a podle dostupnosti rozhraní v systému je daná varianta použita. Pokud jsou na systému k dispozici obě, je upřednostňováno rozhraní OpenGL, které je hlavní podporované rozhraní a knihovna by s ním měla pracovat rychleji [8]. Od verze 1.5.0 Javy 3D je možné použít i renderovací pipeline JOGLu.

Vyšší výkon je v Javě 3D dosahován optimalizací grafu scény. Knihovna zároveň dovoluje obejít graf scény v tzv. Immediate-módu (okamžitý mód), čímž programátorovi nabízí větší kontrolu nad vykreslováním a řízením scény. Dále Java 3D nabízí mimo Immediate módu ještě renderování v tzv. Retained módu, který pro svou činnost vyžaduje konstrukci grafu scény a informace o objektech, které se při renderování mohou měnit, a kompilovaném retained módu, který může některé části grafu zkompilovat do svého vnitřního formátu a tím zvýšit výkon aplikace.

Jedním z rysů knihovny Java 3D je také oddělení virtuálního (uzly ViewPlatform a View) a fyzického světa v grafu scény. To umožňuje pouhou změnou parametrů využít řadu výstupních zařízeních, ať jde například o monitor, stereo brýle či systém CAVE[4].

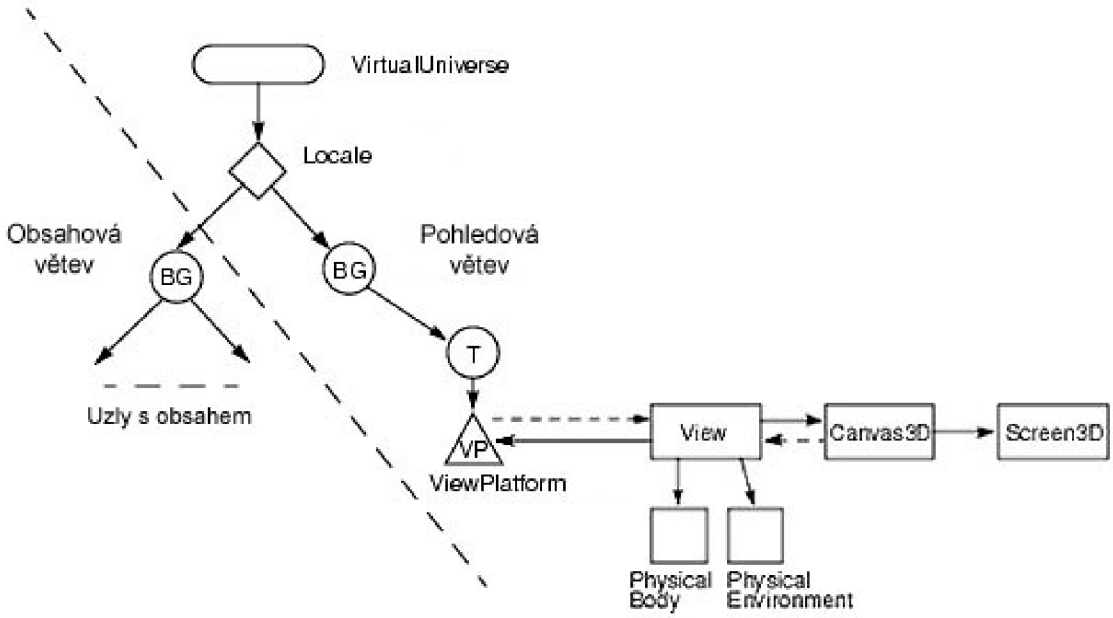
2.1 Graf scény Javy3D

Základní grafická pipeline je skryta za grafem scény Javy 3D, což programátorovi usnadňuje 3D programování a může se zaměřit spíše na návrh scény než na vlastní vykreslování. Java 3D nepovoluje v grafu scény smyčky, jedná se tedy o orientovaný acyklický graf. Pomocí grafu scény knihovna Java 3D na implementační úrovni obstarává například vynechání objektů nacházejících se mimo zorné pole pozorovatele (view culling), nebo schovaných za jinými objekty (occlusion culling), LOD[5] a mnoho dalšího. Stromová struktura grafu scény sestává z uzlů Node. Tyto uzly mohou reprezentovat 3D modely, světla, zvuky, pozadí, kameru a další prvky scény. Existují dva hlavní typy uzlů. Uzel typu Group může obsahovat další uzly a tím je seskupit dohromady a uzel typu Leaf reprezentuje listy grafu, což jsou jednak viditelné prvky (například geometrické tvary), ale i nehmotné objekty, světla či zvuky. Uzly typu Leaf zároveň mohou obsahovat uzlové komponenty, které určují geometrii, barvu, odrazivost a další vlastnosti pro tento uzel. Uzly Leaf pak mohou mezi sebou sdílet uzlové komponenty a objekty tak lze využít například se stejnou geometrií, ale odlišným materiálem.

---

[4] CAVE Automatic Virtual Environment je systém virtuální reality založený na projekci obrazu na čtyřech obrazovkách obklopujících pozorovatele

[5] Level Of Detail – úroveň detailu



Obrázek 2.1: Graf scény

2.1.1 Struktura grafu scény

Základní struktura grafu scény je většinou pro všechny aplikace shodná. Vrcholovou entitu grafu tvoří Virtuální vesmír(Virtual Universe), který nám poskytuje oddělený prostor pro definování objektů a jejich vztahů ve scéně. Virtuální vesmír zároveň slouží jako vstupní bod pro render Javy 3D. Pod něj se zařazuje objekt typu Locale definující pevnou pozici ve vesmíru s využitím souřadnic o vysokém rozlišení (HiResCoord), z nichž vycházejí všechny ostatní objekty pod ním. Objekt Locale také slouží hlavně jako kontejner pro všechny pod-grafy scény, které jsou uvozeny uzlem typu BranchGroup. Ve většině případů jsou pod uzlem Locale pouze dvě tyto větve, jedna obsahová a druhá pohledová. Do obsahové větve by se měly vkládat uzly související s obsahem, jako jsou například trojrozměrné tvary, světla či chování. Obsahová větev je pro každou aplikaci různá. Na rozdíl od toho pohledová větev, obsahující objekty související se zobrazením, je pro většinu aplikací stejná.

Do grafu scény se pohledový model zapojuje přes listový uzel ViewPlatform. Rodiče tohoto uzlu pak mohou ovlivnit pozici a orientaci pohledu do scény. Objekt View(Pohled) v sobě sdružuje nastavení renderovacího procesu a také odkazy na renderovací plátna Canvas3D. Javě 3D lze tak najednou vytvořit více aktivních pohledů, každý se svou skupinou pláten. Jedním z příkladů možného nastavení, které v sobě objekt pohledu sdružuje, je BackClipCistance určující hranici viditelnosti pro vzdálené objekty a MinimumFrameDuration, přes který lze zvolit minimální délku trvání snímku a tím tedy maximální snímkovou frekvenci vykreslování. Mezi další objekty pohledové větve patří již zmíněná kreslící plocha Canvas3D, což je v podstatě 3D verze AWT komponenty Canvas, a objekt Screen3D obsahující informace o zobrazovacím zařízení. Mimo to jsou v pohledové větvi ještě

objekty s informacemi o fyzickém světě, jako uživateli a prostředí, ve kterém se uživatel nachází. Jedná se o objekty PhysicalBody a PhysicalEnvrinoment. Tento pohledový model, zcela odlišný od modelu založeném na kameře známém z nízko-úrovňových rozhraní, byl vytvořen, aby zaujímal široké pole použitelnosti, například i v aplikacích pro interakci uživatele s virtuálním světem.

Základním listovým objektem obsahové větve je objekt Shape3D pro specifikaci geometrických tvarů. Mezi dalšími objekty jsou pak například hranice, neboli Bounds, pro nastavení oblasti působnosti některých operací nebo pro nastavení zástupce tvaru složitějších objektů pro detekci kolizí. Poslední objekt, který zmíním, je objekt Behaviour, jenž nám umožňuje vložit do scény chování aktivované za určených podmínek. Na této třídě jsou pak postaveny Interpolátory, které automaticky mění s časem určité parametry objektů (například RotationInterpolator nám umožňuje automatizovat rotaci objektů).

Abychom nemuseli pokaždé sestavovat virtuální vesmír a jeho pohledovou větev, nabízí nám Java 3D objekt SimpleUniverse, který za nás vytvoří virtuální vesmír, objekt Locale a pohledovou větev scény a to v té nejběžněji používané konfiguraci, se kterou si ve většině aplikací vystačíme.

Všechny objekty, které jsou připojeny do větve aktivního virtuálního vesmíru jsou tzv. živé (live). S živými větvemi grafu lze pak provádět pouze činnosti, které jsme pro daný uzel povolili před připojením do virtuálního vesmíru. Jako příklad takové činnosti je povolení schopnosti nastavit transformaci transformačnímu uzlu grafu za běhu. Pokud by uzel tuto schopnost neměl, pak by program při pokusu nastavit transformaci pro daný uzel vyvolal výjimku.

Dodatečně mohou být uzly grafu také kompilované, kdy Java 3D provádí určité optimalizace pro zvýšení výkonu. S kompilovaným grafem scény se musí zacházet stejně jako s živým, tedy je na něm možné provádět pouze ty operace, které jsme před kompilací povolili.

Struktura obsahové větve grafu scény by měla odrážet vazbu mezi objekty ve virtuálním světě. Seskupení do společných uzlů dává programátorovi možnost manipulovat se skupinou objektů jako s jednotlivou entitou. Skupina uzlů má zároveň své prostorové ohraničení určené ohraničením geometrií pod ním a prostorové seskupování umožňuje Javě 3D efektivní správu operací jako detekce těsné blízkosti, detekce kolizí a ořezávání objektů mimo zorné pole pozorovatele.

2.2 Výkon

Proti knihovně je často namítáno, že je na hry příliš pomalá, jediný seriózní test byl však proveden v roce 2002 Jacobem Marnerem, který porovnával výkon OpenGL(použil jazyk C++ a v Javě vazbu GL4Java pro OpenGL) a Javy 3D. Jako nejrychlejší se ukázala verze s C++ a o něco pomalejší implementace GL4Java. Java 3D vyšla asi 2,5krát pomalejší, ovšem test byl prováděn na verzi knihovny 1.3.0, která obsahovala chybu ovlivňující výkon aplikace. Od té doby nebyly testy opakovány a dnes je již knihovna ve verzi 1.5.1.

3 Testování rozhraní Java 3D

Před samotnou implementací hry bylo nutné otestovat schopnosti a možnosti rozhraní, protože nebyly k dispozici žádné aktuální záznamy o provedených testech nad tímto rozhraním.

3.1 Testovací konfigurace

WinXP:

Operační systém: Windows XP Professional SP2

Procesor: Intel Pentium 4 3GHz

Grafická karta: GeForce 7600 GT

Paměť: 768 MB

Ubuntu:

Operační systém: Ubuntu 7.10

Procesor: Intel Centriono Duo 1,6GHz

Grafická karta: Intel Graphics Media Accelerator 950

Paměť: 1 GB

(Notebook HP 530)

3.2 Návrh testů

Účelem testování nebylo vytvořit benchmark pro otestování výkonu grafické karty, ale zaměřit se na implementaci rozhraní Javy 3D, konkrétně na oblasti související s hrami pro následnou implementaci hry v tomto rozhraní. Protože základem každé aplikace v Javě 3D je graf scény, rozhodl jsem se otestovat závislost rozložení objektů v grafu na rychlosti vykreslování. Java 3D zároveň implementuje i detekci kolizí, která je pro hry rovněž klíčová a proto je na ni zaměřen druhý test rozhraní. Poslední test byl vytvořen až po implementaci hry samotné. Jelikož při hře docházelo ke značnému zpomalení, tedy trhání, při připojování rozsáhlých větví do grafu scény za chodu, byl sestaven ještě poslední test zaměřený na zpomalení při připojování částí grafu za běhu aplikace.

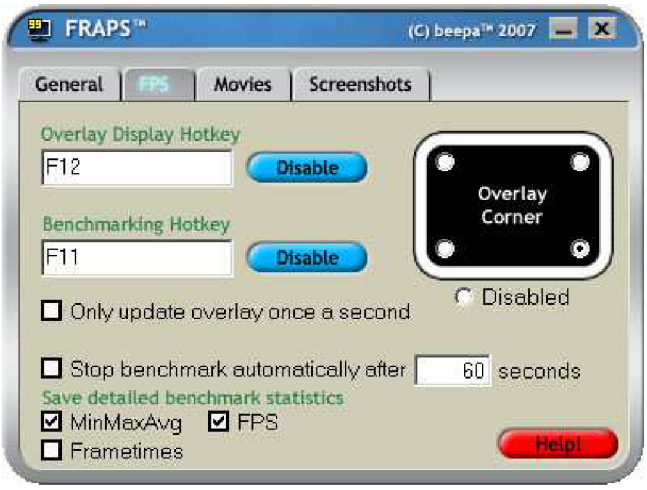
3.3 Implementace testů

Pro testování byla vytvořena samostatná aplikace spustitelná s různými parametry pro různé nastavení testů. Do scény byla pro testování vkládána množina objektů. Těm byla předem vygenerována jejich velikost a souřadnice do externího souboru společného pro všechny testy, aby byly data pro všechny testy stejná. Mimo těchto objektů scéna obsahuje nadefinovaná chování Javy 3D typu

RotationInterpolator či PositionInterpolator pro automatickou změnu polohy kamery či rotaci testovaných objektů. V testech jsou jako testované objekty použity předdefinované objekty typu ColorCube, což jsou krychle s každou stěnou o jiné barvě. Objekt ColorCube musí být vložen do transformačního uzlu typu TransformGroup, aby bylo možné určit jeho polohu na scéně. Připojovaný testovací objekt je tedy uzel TransformGroup a primitivum ColorCube. Mimo ColorCube se v testech vyskytuje ještě objektu typu Sphere, tedy geometrický útvar koule. Rychlost zobrazování je měřena ve snímcích za vteřinu pomocí vytvořeného chování, které po uplynutí přednastaveného počtu snímků(/milisekund) vypočítá z uběhnutého času(/počtu snímků) snímkovací frekvenci. Při počítání snímkovací frekvence pro přednastavený čas běhu však vzniká problém, protože Java 3D nevrací počet zobrazených snímků ale počet vyrenderovaných snímků, tedy snímků, při kterých docházel na scéně k nějaké změně, proto pokud vytvoříme statickou scénu a necháme ji spuštěnou na dobu 10 vteřin, Java 3D vrátí pouze 2 snímky, což je způsobenou změnou z prázdné scény na scénu s obsahem. Z tohoto důvodu, pokud potřebujeme test spuštěný po určitou časovou dobu, je vhodné pro měření použít externí nástroj. Jako tento externí nástroj byla použita aplikace Fraps (Více v [3]).

3.3.1 Fraps

Fraps je aplikace od společnosti beepa®, která dokáže mimo jiné měřit a zaznamenávat snímkovací frekvenci aplikací využívajících rozhraní DirectX nebo OpenGL. Mimo minimální, průměrné a maximální hodnoty fps[6] zároveň nabízí ukládání průběžných hodnot fps během testu. Protože zahájení ukládání snímkovací frekvence vyžaduje stisk klávesy (defaultně F11), je třeba pro automatizaci testů spustit testovací aplikací se specifickým nastavením, aby po spuštění testu stisknutí klávesy vyvolala programově. V době tvorby práce byla aplikace k dispozici ve verzi 2.9.4., která byla použita při testování Javy 3D.



Obrázek 3.1: Uživatelské rozhraní aplikace Fraps

[6] Frames Per Second – Snímky za vteřinu

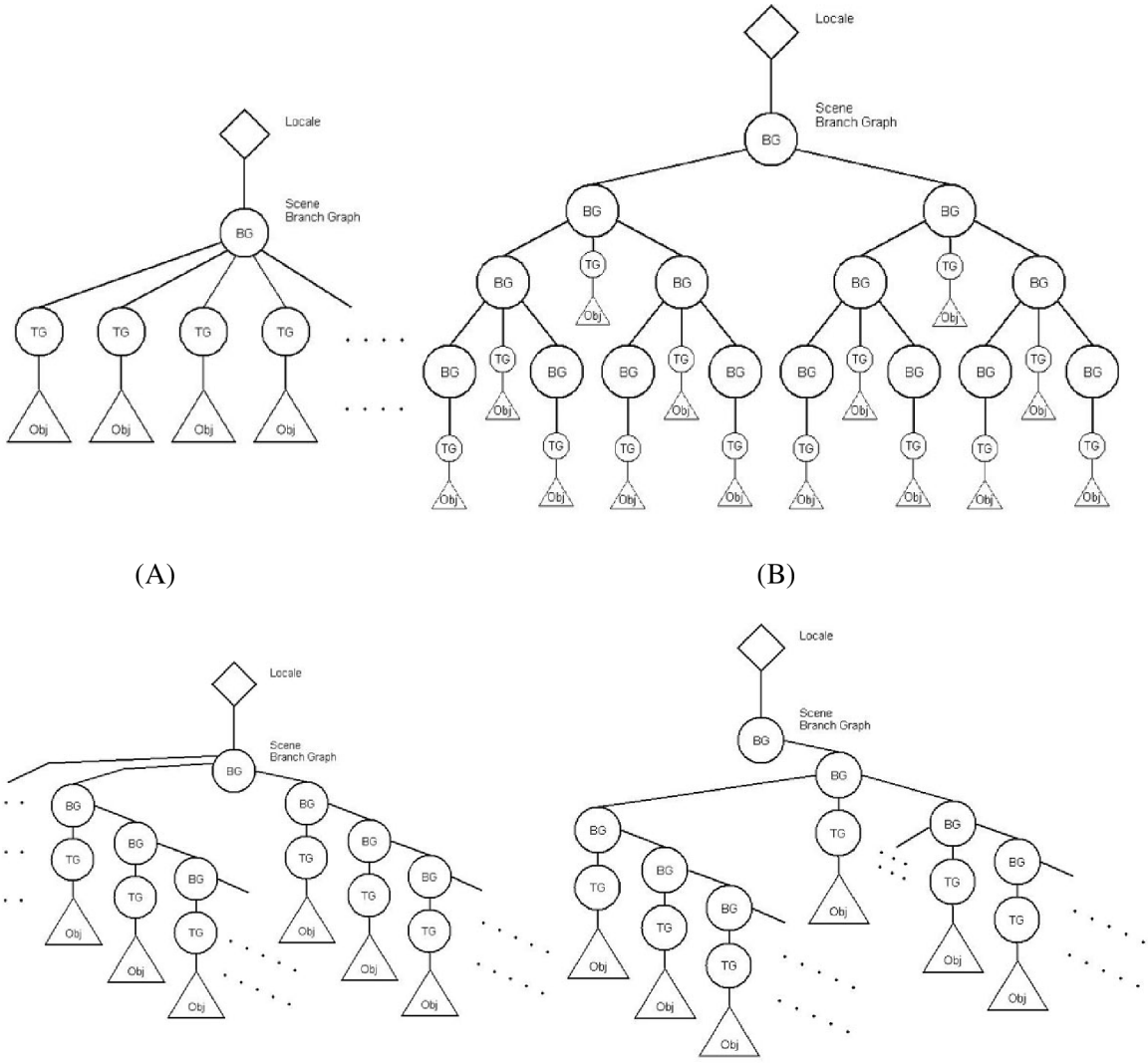
3.3.2 Testy

Testy je možné plně automatizovat vytvořením jednoduchého skriptu například v Command Shellu v operačním systému Windows XP. Více informací o nastavení testovací aplikace je k dispozici v uživatelské příručce.

3.3.2.1 Test závislosti rozložení objektů v grafu scény

Tento test slouží ke zjištění závislosti mezi rozložením objektů ve scéně do stromové struktury uzlů a snímkovací frekvencí. Jsou vytvořeny čtyři druhy grafu stromu pro testovaní. První zařazuje objekty přímo pod hlavní uzel grafu scény (Scene Branch Graph) a slouží jako referenční pro ostatní druhy stromů. Jeho rozložení je vykresleno na obrázku 3.1 (A). Zbylé tři druhy pracují s grafem scény jako s binárním stromem. První z nich vytváří vyvážený strom ukázaný na obrázku 3.1 (B). Při vytváření druhého, nevyváženého, stromu, bylo zjištěno omezení Javy 3D zpracovat maximálně 1433 pod sebe zařazených uzlů. Při překročení této limitní hodnoty je vytvářena výjimka „StackOverflowError“ při kompilaci uzlů a nebo při jejich nastavování do živého stavu po připojení do živé větve grafu. Aby bylo možné do nevyváženého grafu připojit až 10000 objektů, jsou vytvořeny dva přístupy pro připojování části grafu překračující tento limit. První, ilustrovaný na obrázky 3.1 (C), připojuje přesahující větev do kořenového grafu a druhý, na obrázku 3.1 (D), připojuje přesahující část vždy k první volné levé větvi grafu. V záštitu tohoto testu je zároveň test vlivu před-kompilace grafu scény na rychlosti vykreslování.

Test probíhá pro počet testovacích objektů v hodnotách z rozsahu od 1 do 10000, jednou při rotaci kamery kolem těchto objektů a jednou při rotaci objektů kolem středu scény a statické kamery. Každý test je spuštěn vícekrát (5krát nebo 3krát) a výsledná hodnota je brána jako medián těchto hodnot pro potlačení chyb v měření. Každý test je spuštěn po dobu 5000 snímků a výsledná snímkovací frekvence je vypočítána z času trvání testu.



Obrázek 3.2: Rozložení grafu stromu pro testování: (A) referenční, (B) vyvážený, (C) nevyvážený 1, (D) nevyvážený 2

3.3.2.2 Test detekce kolizí

Pro test závislosti snímkové frekvence na aktivované detekci kolizí na objekty ve scéně je vytvářena scéna s testovacími objekty stejně jako v předchozím testu, avšak rozložení objektů ve scéně je testováno již jen jedno, tedy to uvedené jako referenční pro předchozí test, a kamera i scéna je během testu statická, tedy bez rotací. Do scény je vloženo 8 „detekčních“ koulí obíhajících po kruhu v prostoru vygenerovaných testovacích objektů a zároveň klesajících přes ně od shora dolů.

Test probíhá pro počet testovacích objektů v hodnotách z rozsahu od 500 do 10000 s nastavenou hladkostí detekčních koulí (počtu jejich segmentů) na výchozí hodnotu 15, dále pak 40 a 80. Java 3D nabízí dva způsoby detekce kolizí. Jeden, při kterém se pro test kolize používají nadefinované hranice (Bounds), jako například koule (BoundingSphere) pro rychlejší výpočty a druhý způsob, jenž pro detekci používá geometrii objektu. Z toho důvodu je test spuštěn celkem

třikrát, poprvé bez aktivovaného chování pro detekci kolizí na detekčních koulích pro získání referenčních hodnot, po druhé s chováním pro detekci kolizí s použití hranic objektu a potřetí s použitím geometrie objektu pro test kolize. Chování pro detekci kolizí nemá v sobě žádný výpočetní kód spouštěný při jeho aktivaci, kromě smyčky pro projití výčtového typu s kolizemi (většinou pouze jedna) a přičtení počítadla aktivovaných chování.

Snímkovací frekvence je pro tento test měřena externím nástrojem, protože je vhodné, aby test probíhal pouze po dobu pohybu detekčních koulí přes objekty, tedy s nastavením časového omezení testu. Jak bylo zmíněno výše, při tomto testu dochází k nesprávnému vyhodnocování snímkovací frekvence.

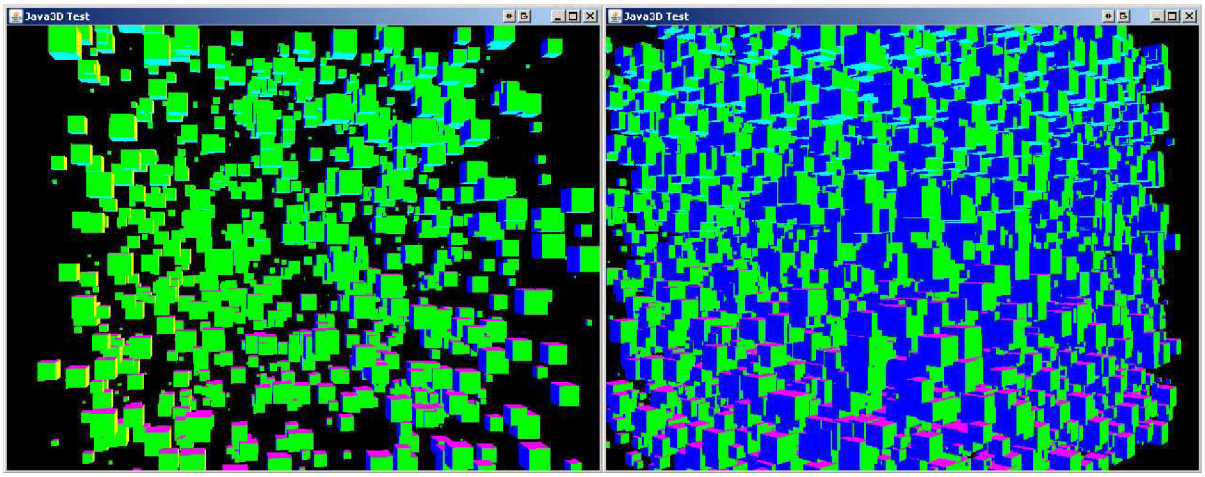
U testovací konfigurace Ubuntu není možné externí nástroj použít, protože není k dispozici jeho verze pro operační systém Linux, proto je u ní použito vnitřní měření fps.

3.3.2.3 Test připojování větví do grafu scény za chodu

Pro test připojování větví do grafu scény za chodu je pohled kamery mimo prostor testovacích objektů a pohybuje se zleva směrem doprava do prostoru s objekty. Větev grafu s testovacími objekty je do grafu scény připojována ještě v době před jejím vynořením v zobrazované části scény, aby test grafu s větví připojenou před spuštěním a s větví připojenou za chodu zobrazoval stejné množství grafických informace. Test tedy probíhá vždy dvakrát, jednou s větví grafu vloženou do scény ještě před započetím renderování a podruhé během něho. Test je zároveň spouštěn s předkompilovanou a nepředkompilovanou připojovanou větví. Množství připojovaných objektů je testováno v hodnotách z rozsahu od 1 do 10000. Zároveň byla do testovací scény přidána řada koulí po zobrazované dráze, aby při renderování nevynikla hluchá místa, kdy je zobrazena prázdná plocha (což je stejné jako při nehybné scéně) a nejsou tak renderem započítávány renderované snímky. Snímkovací frekvence tohoto testu je rovněž měřena externím nástrojem.

3.4 Provádění testů a výsledky testů

Při testování byly na testovaných počítačích ukončeny všechny nepotřebné programy a nebyla na nich prováděna žádná jiná činnost. U testované konfigurace WinXP bylo zpozorováno ořezávání grafů na snímkovací frekvence kolem 60 fps. Toto bylo způsobeno obnovovací frekvencí primárního monitoru (na testovací konfiguraci WinXP - LCD monitoru s vertikální obnovovací frekvencí 60Hz). Aby byla ztráta dat při ořezání co nejmenší, byl pro test nastaven jako primární monitor s vertikální obnovovací frekvencí 75Hz. Testy ořezané na hodnotě 60 fps byly z výsledné sady vypuštěny. Takovéto omezení a ořezání se na testovací konfiguraci Ubuntu s operačním systémem typu Unix neprojevilo.

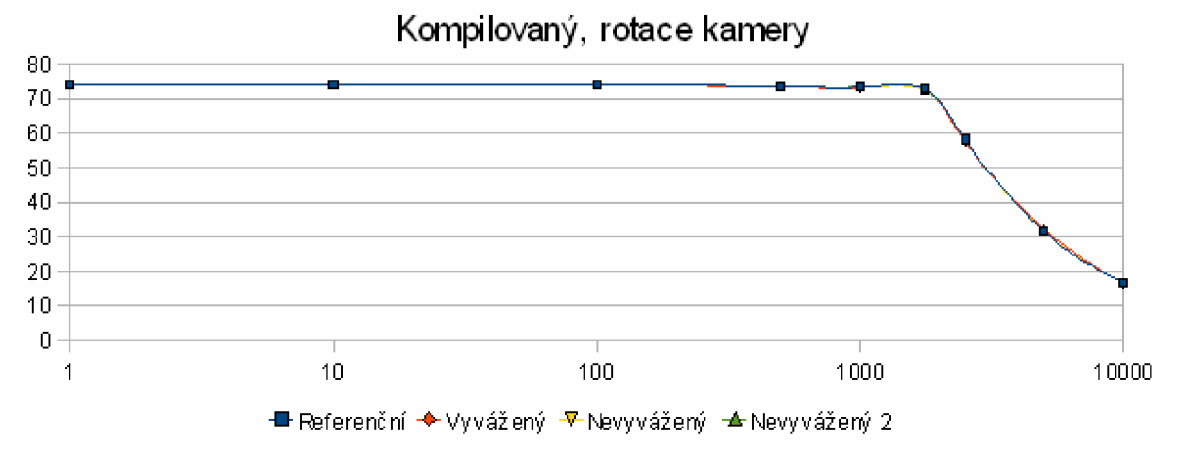


Obrázek 3.3: Snímky z testování závislosti rozložení, vlevo 1000, vpravo 10000 objektů

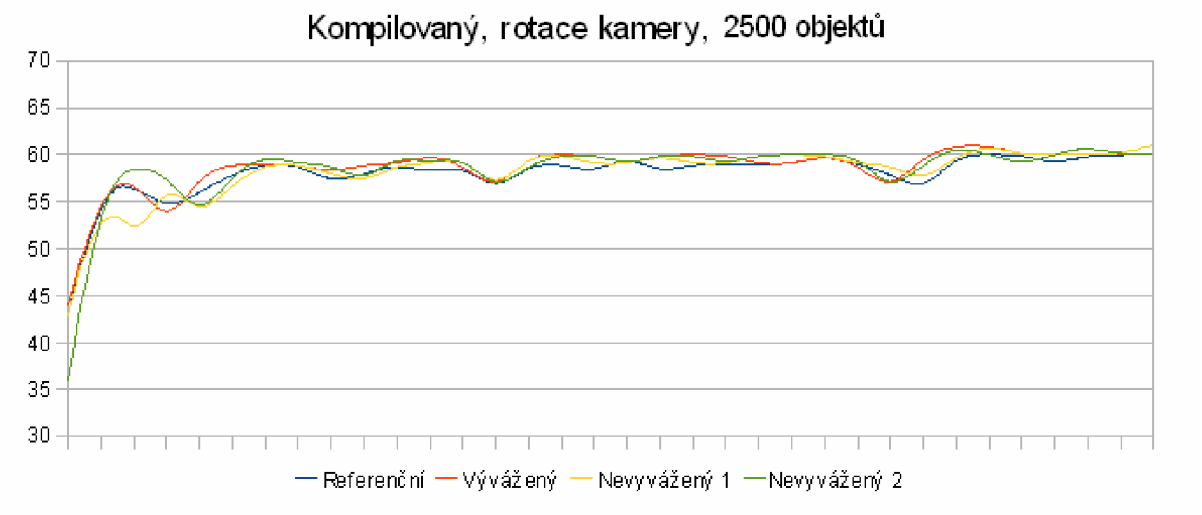
3.4.1 Test závislosti rozložení objektů v grafu scény

Test byl spuštěn na testovací konfiguraci WinXP (použité výchozí rozhraní OpenGL) a Ubuntu (rozhraní OpenGL). Výsledky testů a z nich vyvozené grafy ukazují, že rychlost vykreslování není ovlivněna procházením grafu renderem Javy 3D, tedy uspořádáním objektů v grafu scény. Pro ověření dosažených výsledků testů byly testy spuštěny ještě s externím programem pro měření snímkovací frekvence, aby testy nemohly být ovlivněny způsobem získávání počtu snímků při vykreslování přes vnitřní funkci Javy 3D. Tento test dosažené výsledky pouze potvrdil a protože je jejich informační hodnota bezvýznamná, nebyly tyto grafy zahrnuty do přílohy. Aplikace nám však zároveň nabídla pohled na průběh fps během testu. Graf 3.2 tento průběh ukazuje pro 5000 testovacích objektů a zároveň také potvrzuje vyvozenou nezávislost rozložení objektů v grafu scény na snímkovací frekvenci vykreslování. Jediný rozdíl ve výsledcích testů byl zaznamenán pouze, podle očekávání, mezi testy s rotací kamery a testy s rotací celé scény, kde byl pokles snímkovací frekvence až dvojnásobný pro test rotace scény, jde však pouze výsledky ovlivněny výkonem grafického adaptéru a ne rozhraní Java 3D.

Přestože grafy z tohoto testu neprokázaly očekávanou závislost, poslouží ještě pro návrh hry, pro zvolení vhodného omezení snímkovací frekvence a s tím související maximální množství zobrazených objektů na scéně.



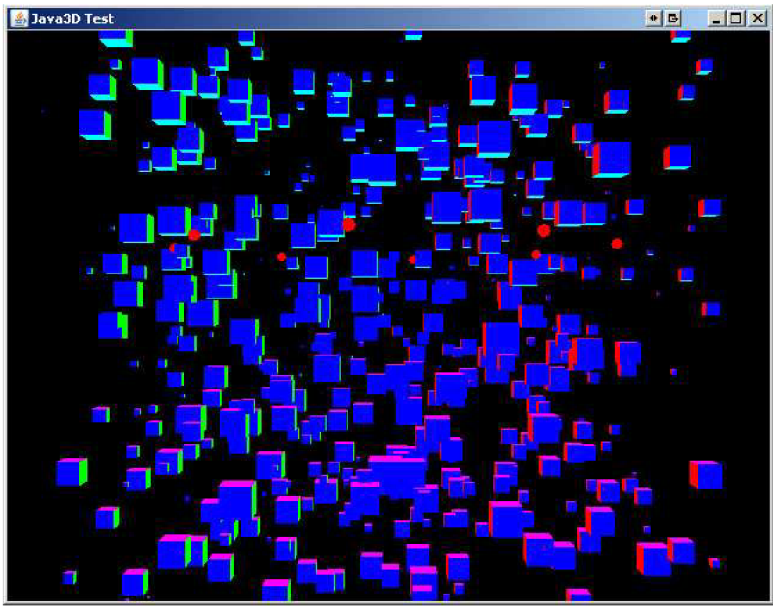
Graf 3.1: Ukázka výsledného grafu testování závislosti rozložení objektů v grafu scény při rotaci kamery a předkompilované větvi grafu na konfiguraci WinXP



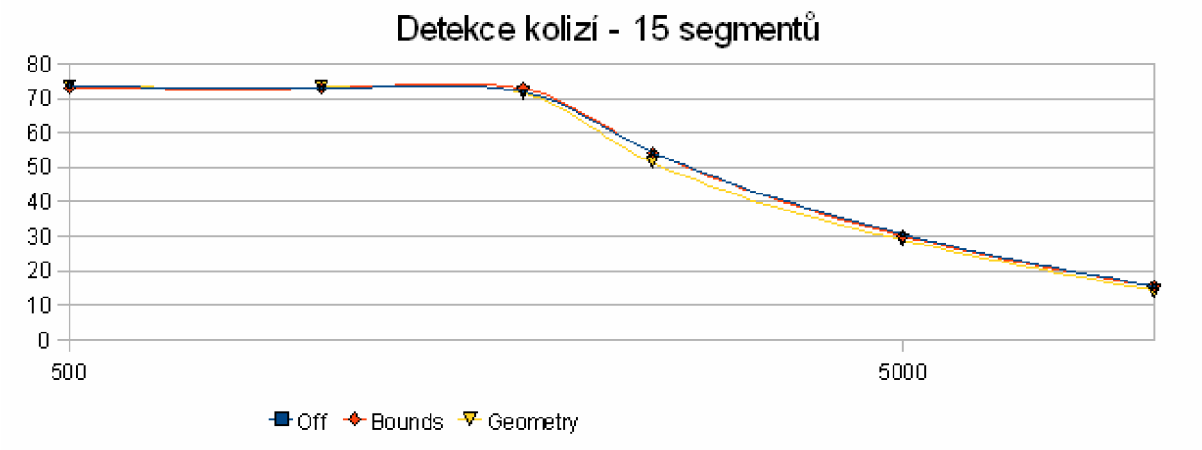
Graf 3.2: Ukázka výsledného grafu průběhu snímkovací frekvence testování závislosti rozložení objektů v grafu scény při rotaci kamery, předkompilované větvi grafu a 2500 objektů na scéně na konfiguraci WinXP

3.4.2 Test detekce kolizí

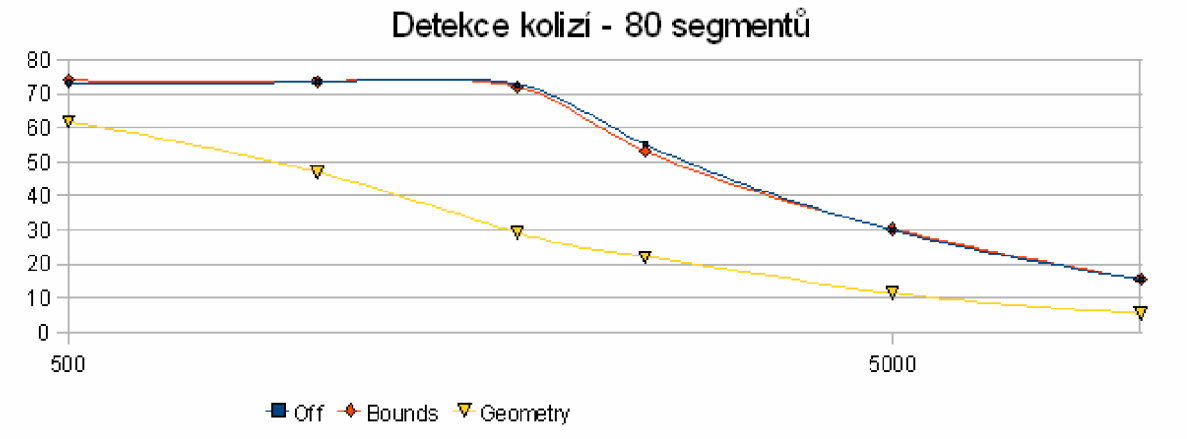
Test byl proveden na testovací konfiguraci WinXP s použitím externího nástroje Fraps a na konfigurace Ubuntu s použitím vnitřního, časově založeného, měření snímkovací frekvence a vyšlo najevo, že na této testovací konfiguraci je tento způsob měření mnohem stabilnější, než na konfiguraci WinXP a přestože naměřené snímkovací frekvence neodpovídaly skutečným hodnotám, je na nich patrný rozdíl výsledků testů podobný konfiguraci WinXP. Výsledky testu ukázali, že při použití hranic objektu k testování kolize neovlivní snímkovací frekvenci. Použití geometrie však již přináší značnou výpočetní režii a je vidět pokles snímkovací frekvence, který je tím větší, čím je kolizní objekt složitější. Při použití detekčních koulí s výchozí segmentací byl tedy pokles téměř zanedbatelný (cca 2 fps), při segmentaci na hodnotě 40 byl však pokles již až 20fps a při 80 až 30 fps.



Obrázek 3.4: Snímek z testování detekce kolizí pro 500 objektů



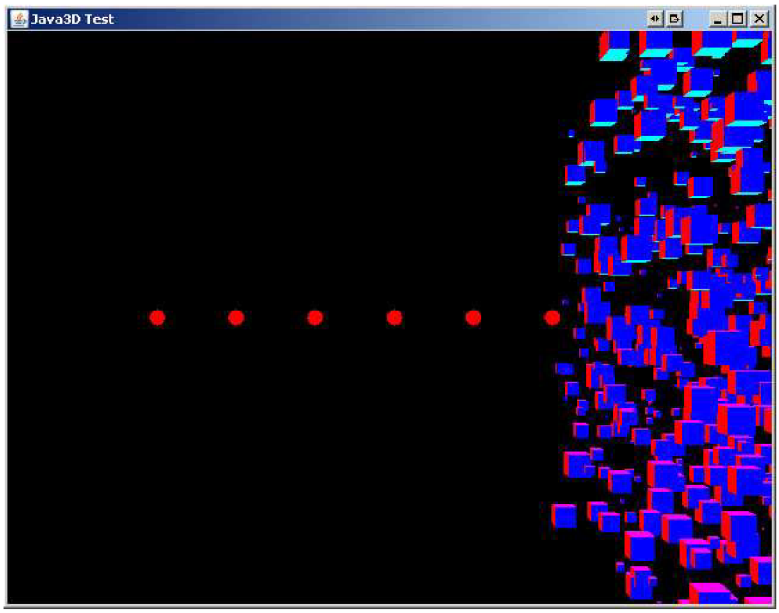
Graf 3.3: Ukázka výsledného grafu testování detekce kolizí při 15 segmentových detekčních objektů na konfiguraci WinXP



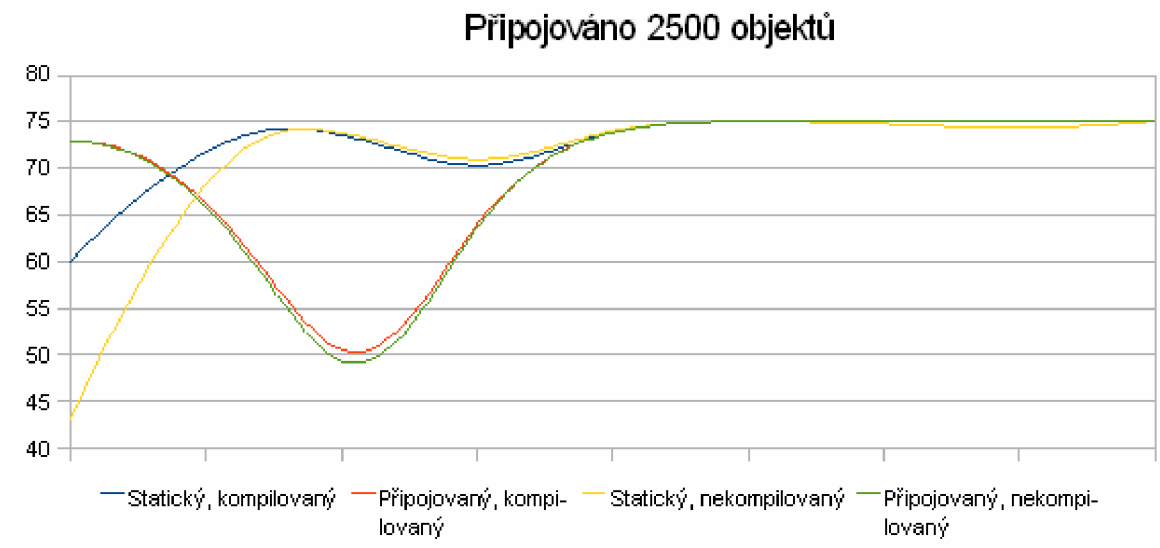
Graf 3.4: Ukázka výsledného grafu testování detekce kolizí při 80 segmentových detekčních objektů na konfiguraci WinXP

3.4.3 Test připojování větví do grafu scény za chodu

Tento test byl opět spouštěn pouze na konfiguraci WinXP. Výsledný graf sestavený z průměrných hodnot snímkovací frekvence naměřený externím programem neukázal žádný vliv připojování větví do grafu scény na snímkovací frekvenci. Pokles snímkovací frekvence je viditelný až při pohledu na průběh fps během testu. Zde je při pohledu na připojování větve s 1000 objekty znatelný pokles fps v době připojení o 10 snímků. Velikost tohoto pokles je vyšší s tím, čím je připojováno větší množství objektů a u 10000 objektů klesne snímkovací frekvence až na hodnotu 2 fps.



Obrázek 3.5: Snímek z testování připojování větví



Graf 3.4: Ukázka výsledného grafu testování připojování větví do grafu scény při 2500 objektech na konfiguraci WinXP

3.5 Celkové zhodnocení výsledků testů

Zjištěná nezávislost rozložení objektů v grafu scény na rychlosti vykreslování nám dává možnost vytvářet libovolně složité grafy a soustředit se tedy doopravdy na tvorbu hry samotné než na optimalizace vyváženosti grafu pro rychlé procházení grafem. Samozřejmě je stále platné pravidlo o prostorovém rozdělení scény. Použití geometrie pro detekci kolizí značně snižuje rychlost vykreslování, proto je výhodné používat pouze detekci založenou na definovaných hranicích objektů, která je pro většinu případů dostačující. Největší problém nastává při připojování rozsáhlejších větví do grafu scény za běhu. Aby nedocházelo ke zpomalení při připojování, je nezbytné rozložit připojení větví s větším množstvím objektů do několika menších částí, kdy nebude zpomalení znatelné.

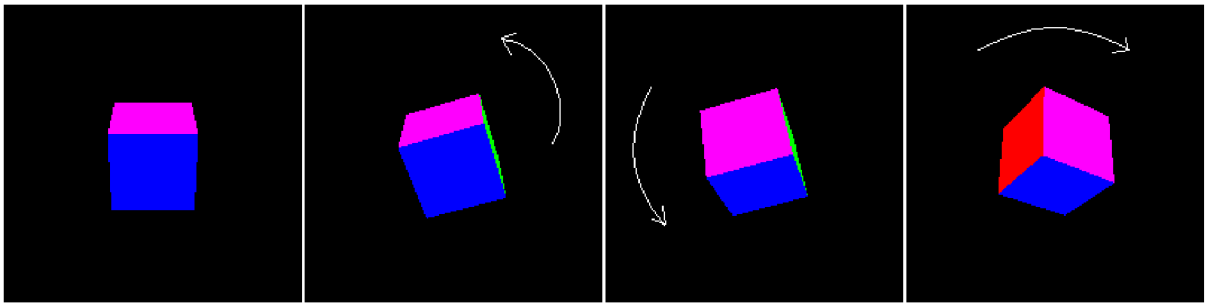
Ve výsledcích jednotlivých testů nebyly diskutovány testy kompilovaných a nekompilovaných větví, kde před-kompilace větví měla zanést určité urychlení vykreslování. Při testování však tento jev nebyl zpozorován a proto tyto testy nebyly ve vyhodnocení dílčích testů diskutovány. Jsou však všechny zaneseny do grafů a je možné je kompletní nalézt v příloze.

4 Tvorba hry

Cílem této části je vytvoření jednoduché hry s využitím testovaného rozhraní Java 3D. Jako herní žánr jsem zvolil simulátor, konkrétně vesmírný letecký simulátor, protože lépe otestuje dané rozhraní svým akčním podáním více než například logická hra jako šachy.

4.1 Návrh hry

Hra ve stylu vesmírného leteckého simulátoru by měla být z pohledu třetí osoby, kdy je mimo scény vidět i část vesmírné lodi, která označuje hráče. Mimo scény by na herní obrazovce měl být zobrazen HUD[7] se základními informacemi o aktuálním stavu hry jako například skóre a množství poškození lodi. Pohyb ve vesmíru by neměl být omezen a měl by být řízen natáčením plavidla do stran kolem osy směru letu a dále pak nakláněním nahoru a dolů. Ovládání plavidla by mělo být možné jak myší, tak klávesnicí. Scéna, tedy vesmír, ve které s bude plavidlo pohybovat by měla být generována náhodně a měla by být zdánlivě nekonečná. Cílem hry by mělo být sbírání určitých předmětů ve vesmíru a získávat tím za ně body. Průletem vesmírem by zároveň mělo hráči klesat palivo, které může doplnit jeho sebráním objektu paliva ve vesmíru. Srážky s některými objekty na scéně by měly způsobovat zvýšení poškození. Hra končí dosažením stoprocentního množství poškození nebo nulové hladiny paliva. Mělo by být možné nastavit také úroveň obtížnosti.



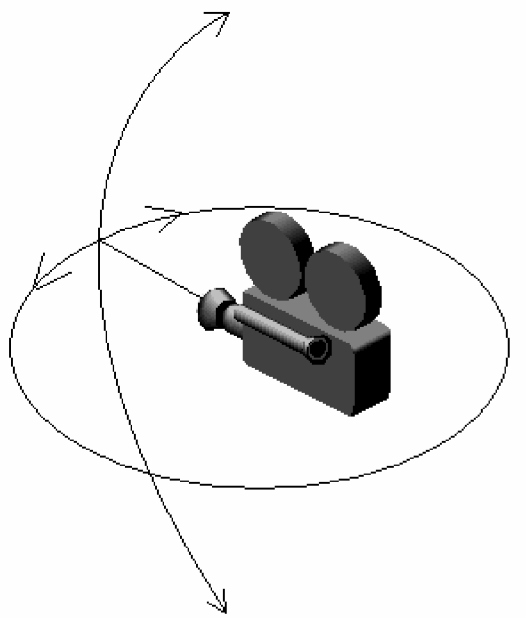
Obrázek 4.1: Ukázka skládání rotace objektu pro neomezený pohyb ve vesmíru – objekt je rotován kolem jeho lokálních os, ne kolem os virtuálního světa (1. snímek – bez rotace, 2. snímek – rotace kolem osy z, 3. snímek – přidána rotace kolem osy x, 4. snímek – přidána další rotace kolem osy z)

4.2 Změny v návrhu

Při implementaci hry podle návrhu jsem narazil na problém s rotací, který se mi nepodařilo vyřešit. Protože rotace objektů podél různých os, tedy násobení transformačních matic, není komutativní, bylo by nutné si pro libovolné natočení pamatovat všechny předchozí úhly a postupně je aplikovat, což je

[7] Head-Up Display je z způsob vizuálního podání informací souvisejících se hrou hráčovi [4]

výkonově neúnosné, nebo natočení přepočítávat za použití základní goniometrických funkcí, čehož se mi nepodařilo přes značnou snahu dosáhnout, nebo použít quaterniony, ty však přesahují mé matematické znalosti. Z tohoto důvodu byl pohled změněn z třetí osoby na pohled z první osoby a byl vytvořen stejný způsob pohybu kamery jako u her žánru FPS[8]. U těchto her je omezením kamery ve vertikálním směru do maximálního natočení k zemi nebo ke stropu.



Obrázek 4.1: Ukázka omezení rotace kamery u her žánru FPS

4.3 Implementace hry

Okno hry je tvořeno pouze komponentou Canvas3D, do které Java3D vykresluje vyrenderovaný obsah. Tato komponenta zachytává všechny události a předává je Javě 3D, přes kterou lze tyto události zpracovávat pomocí chování vložených do grafu scény. Takto je ve hře přes třídu odvozenou od třídy Behaviour vytvořeno ovládání pomocí klávesnice a myši. Při ovládání klávesnicí se při příchodu události pohled natáčí vždy o stejný krok. Při ovládání myší závisí úhel natočení na vzdálenosti kurzoru od jeho poslední pozice. Aby nebylo nutné hlídat pozici kurzoru v okně, je po každém zpracování události kursoru nastavena jeho pozice do středu zobrazovací plochy.

Natáčení, tedy rotace pohledu je implementována podle změn v návrhu jako u her žánru FPS, kdy je kamera natočena nejdříve kolem horizontální ose a poté kolem vertikální osy souřadného systému.

Pohyb hráče vpřed je řízen aplikací a hráč nemá možnost rychlost pohybu ovlivnit. Posun je uskutečněn každých 20ms, což odpovídá snímkovací frekvenci 50fps, a vyvolává jej vytvořené

[8] First Person Shooter je označení žánru počítačových her charakteristický simulací vlastního pohledu herní postavy[5]

chování TimeBehaviour s nastaveným kritériem na probuzení WakeupOnElapsedTime, které zařídí vyvolání po uplynutí zvoleného časového intervalu.

Jednou za každých 5 kroků pohybu je zároveň volána funkce pro obnovení informací pro HUD. Vykreslování dvourozměrného obrazu na plochu s 3D scénou je docíleno rozšířením třídy Canvas 3D a přetížením metody postRender, která je Javou 3D volána po dokončení renderování pro aktuální snímek.

Aby mohl být vesmír nekonečný, je celá scéna rozdělena do matice 3x3x3, kde každá buňka obsahuje „pod vesmír" třídy SpaceBox s aktuálními objekty na zobrazovanými na scéně. Hráč je po celou dobu hry udržován ve středové buňce (1, 1, 1). a pokaždé když tuto buňku opustí, stane se aktuální buňka novou středovou buňkou a do scény jsou dogenerovány chybějící okolní buňky (stěna matice o velikosti 3x3 buněk).

Třída SpaceBox má svůj prostor opět rozdělený do matice, tentokrát však ne vždy o stejných rozměrech, do které jsou již generovány jednotlivé objekty vesmír, objekty odvozené ze třídy SpaceObject. Generování objektů do matice je prováděno z toho důvodu, aby se náhodně generované objekty nepřekrývaly a nebyl k nim tak hráčovi znemožněn přístup.

Každý objekt odvozený ze třídy SpaceBox musí implementovat metodu interact, která je volána při střetu hráče s objektem a dává tak objektu možnost ubrat hráčovi životy(tedy zvýšit jeho poškození), či například zvýšit skóre a po té se odstranit, aby nebylo možné jeho vícenásobné použití.

Pro objekty načítající svůj tvar z externího souboru ve formátu 3DS[9] je použit balíček NCSA Portfolio z knihy Programování dokonalých her v Javě [1] spojující loadery různých formátů souborů s 3d daty do jednoho.

Informace o tom, jaké množství kterých objektů se má vygenerovat, v sobě uchovává třída LevelSettings a je závislá na aktuálním herní úrovni. Mimo informací o objektech je v této třídě také záznam o rychlosti a velikosti stěny matice pro danou úroveň.

Pro generování objektů SpaceBox je vytvořeno samostatné vlákno, aby jeho výpočetní náročnost neovlivňovala hraní. Z toho důvodu zároveň generátor vytváří do své cache paměti určitý počet objektů SpaceBox dopředu, aby vlákno s herní částí nemusel čekat na jejich vygenerování.

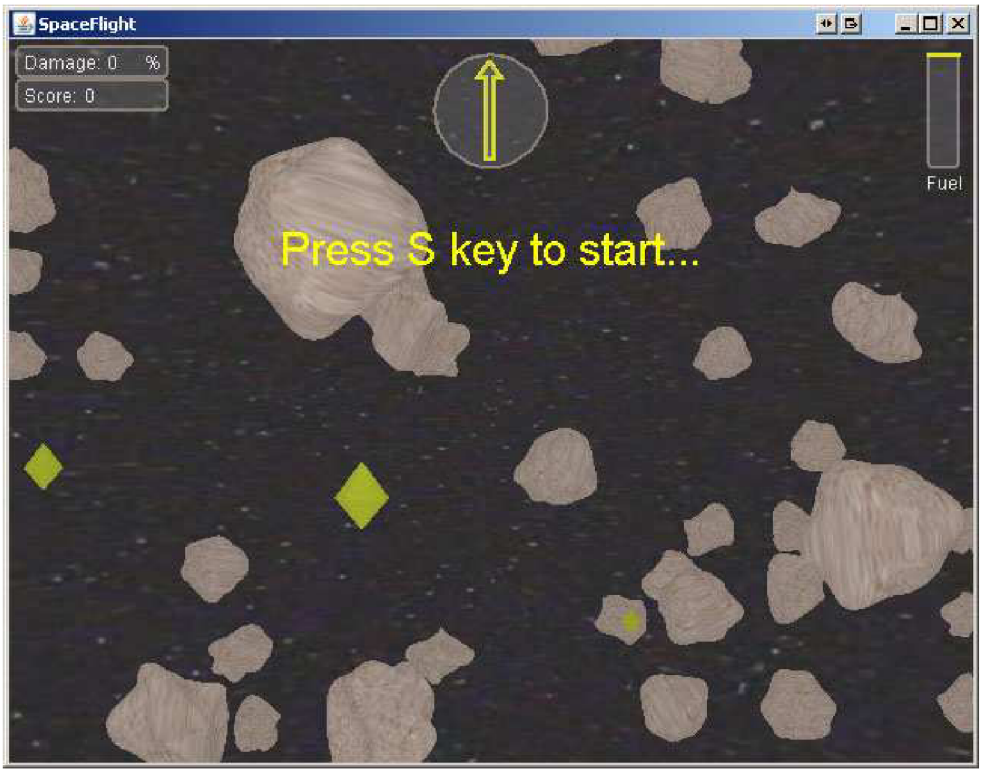
Pro detekci kolizí nebylo možné použít systém detekce kolizí zabudovaný v Javě 3D protože neumožňuje aktivovat detekci na samostatný hraniční objekt (Bounds) potřebný pro reprezentaci kamery. Proto se na detekci kolize testují na všechny hranice všech objektů v aktivní buňce vesmíru na průnik s bodem na aktuální pozici hráče.

Mimo generovaných objektů je ve scéně pozadí s hvězdami. To je vytvářeno jako geometrické pozadí, kdy je textura pozadí natažena zevnitř určitého tělesa, v tomto případě koule, aby nebylo připevněné k pohledu kamery, ale měnilo se v závislosti na jejím natočení ve scéně.. Pro vytvoření pozadí byla použita třída Javy 3D, Background.

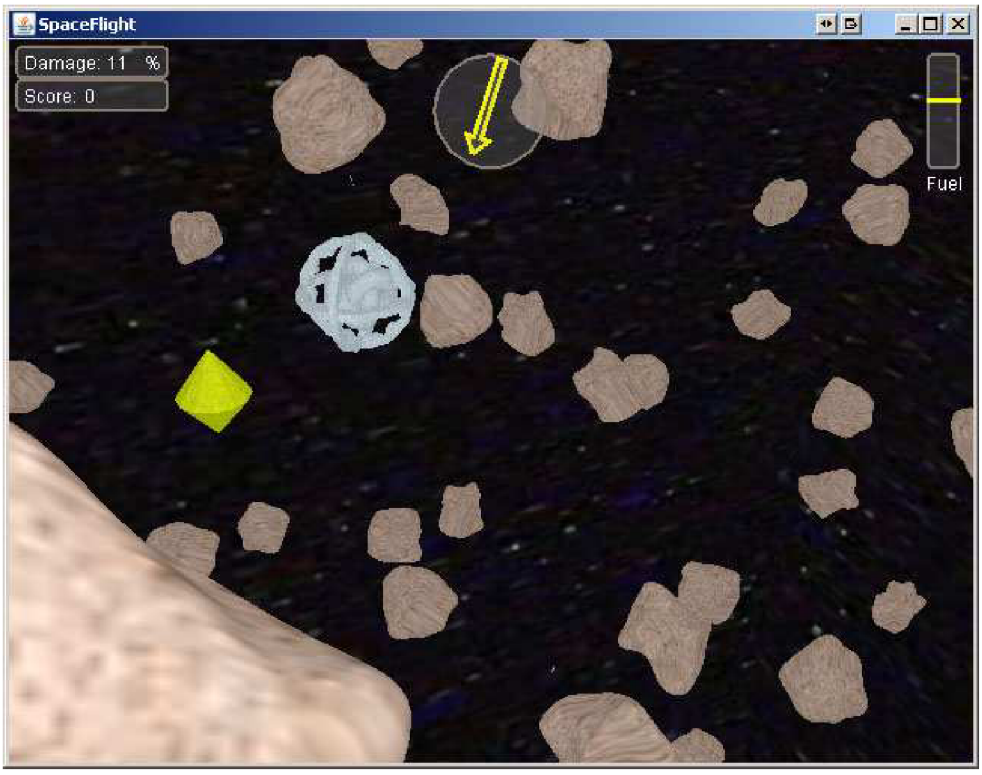
---

[9] 3DS - 3D Studio File Format je souborový formát pro popis trojrozměrných scén. Více v [6]

Scéna je dále osvětlována dvěma světelnými zdroji. První je základní ambientní osvětlení třídy AmbientLight nahrazující všudypřítomné světlo. Druhým světelným zdrojem je bodové světlo osvětlující objekty před hráčem a tudíž měnící polohu a směr podle polohy hráče/kamery.



Obrázek 3.3: Snímek ze hry po jejím spuštění



Obrázek 3.3: Snímek z průběhu hry

4.4 Návrh úprav a vylepšení implementace hry

Velkým nedostatkem v implementaci je v generování nekonečného vesmíru, kde při připojování dogenerovaných částí hlavní matice vesmíru dochází na chvíli k pozastavení vykreslování během něhož není vyhodnocován pohyb myši a po opětovném spuštění dojde k vyhodnocení polohy kursoru a pokud hráč při tomto „záseku“ pohnul myší od jeho poslední pozice, bude náhle změněn směr pohledu, což z této vyplývající neplynulosti vykreslování činí hru téměř nehratelnou. Tento problém nelze vyřešit jednoduše například zvětšením objektu SpaceBox, kdy by sice připojování probíhalo méně častěji, avšak s mnohonásobně vyšším počtem připojovaných objektů by mnohonásobně stoupla i doba připojování větve do grafu scény. Řešením problému by bylo připojování větve po menších částech, což by ovšem znamenalo zásadní změnu v implementaci a proto nebyl tento nedostatek ve hře opraven.

Dalším nedostatkem je již zmíněné natáčení pohledu, které neumožňuje absolutně volný pohyb ve vesmíru, jaký by byl u hry tohoto typu očekáván. Nejvhodnější implementací by bylo použití quaternionů, jejichž použití je pro řešení takovéhoto problému všeobecně doporučováno. Zároveň by pro tento typ hry byl vhodnější původně navrhovaný pohled ze třetí osoby.

Poměrně jednoduchého vylepšení by se dalo docílit v oblasti rozšiřitelnosti, pro kterou je hra díky svému schématu generování objektů již připravena. Pokud by se vytvořilo načítání nastavení úrovní ze souboru bylo by nejen možné úrovně libovolně modifikovat, ale zároveň přidávat do hry nově vytvořené objekty.

5 Závěr - Využitelnost API Java 3D pro tvorbu herních aplikací

Java 3D nám poskytuje velmi dobrou základnu pro snadnou tvorbu 3D aplikací. Pro tvorbu her nám zároveň dává k dispozici prostředky využitelné v této sféře, jako je LOD, detekce kolizí a picking[10]. Dle provedených testů však vyplývá, že Java 3D není přizpůsobena pro připojování části větví za chodu, ale pravděpodobně pro sestavení celé scény před jejím použitím. Tento fakt se však vzájemně vylučuje s mohutným objektovým návrhem v Javě 3D, který nebyl primárně vytvářen pro hry, ale pro všeobecné použití, a proto všechny testy končily na maximálním počtu testovaných objektů na hranici 10000. Nad touto hranicí byly paměťové nároky již pro testovanou konfiguraci neúnosné. Z tohoto problému a z toho, že nebyla Java3D vytvořena přednostně s ohledem pro tvorbu her vyplývá vymezení využitelnosti pro nenáročné herní aplikace s předem vytyčeným herním světem.

Java 3D tedy využitelná pro hry je, ne však pro hry akčního žánru, kde je rychlost vykreslování kritická. Vhodnější použití je například pro adventury či hry logické. Zároveň se díky pohledovému modelu Javy 3D nabízejí zajímavé možnosti využitelnosti pro tvorbu interaktivních her například s podporou HMD[11] a polohových senzorů. Pro tuto oblast by měla být Java 3D plně přizpůsobena, jak vyplývá z nalezeného zdroje [14]. Zda však skutečně poskytuje dostatečně rychlou odezvu pro interaktivní aplikace je z výsledků této práce velice diskutabilní a určení využitelnost Javy 3D pro tuto oblast mnohonásobně přesahuje rámec této práce.

V době vzniku Javy 3D, kdy nebyly dostupné žádné jiné podobné knihovny pro práci s 3D grafikou v Javě, bylo její použití pro hry výlučné, dnes jsou však k dispozici knihovny vyvýjeny primárně pro tvorbu her, jako například již zmíněný Xith3D, který je podle zdroje[10] dvojnásobně rychlejší než Java 3D. Použití Javy 3D ve hrách dnes proto dává smysl jen pro případ využití zdrojů osob již seznámených s tímto rozhraním. Avšak i toto nemusí být opodstatněným důvodem, protože rozhraní Xith3D z Javy 3D vychází a náklady na přechod mezi nimi by byl minimální.

I přes značné zápory pro využitelnost Javy 3D pro hry, vzniklo ve době jejího úsvitu několik kvalitních herních projektů, z nichž zmíním alespoň projekt leteckého simulátoru FlyingGuns[11], který je jako jeden z mála dostupný dodnes.

Zajímavým pokračováním na téma této práce v rámci práce diplomové by mohlo být porovnání Javy 3D, Xith3D a jME na implementační úrovni a případný návrh vlastního enginu pro tvorbu 3D her v Javě. Zároveň by bylo zajímavé profilovat kód Javy 3D a zjistit, co způsobuje zpomalení při

---

[10] Vybírání objektů na scéně pomocí paprsku, bodu či jiných prostředků.

[11] Head-mounted display je způsob zobrazování informací uživateli přes displeje připevněné k hlavě [9]

připojování větví do scény a případně navrhnout řešení tohoto problému. Za prostudování by rovněž stála oblast využití Javy 3D pro zmíněné interaktivní aplikace.

Literatura

[1] Davison, A. *Programování dokonalých her v Javě*. Brno, SNTL 2006.

[2] *Propojení Javy a OpenGL*. Dokument dostupný na URL
http://jogl.sislik.net/propojeni-javy.php

[3] *Fraps*. Dokument dostupný na URL
www.fraps.com

[4] Wikimedia Foundation, *HUD (Computer gaming)*. Dokument dostupný na URL
http://en.wikipedia.org/wiki/HUD_%28computer_gaming%29

[5] Wikimedia Foundation, *FPS (žánr počítačových her)*. Dokument dostupný na URL
http://cs.wikipedia.org/wiki/First-person_shooter

[6] *Vektorové grafické formáty a metaformáty*. Dokument dostupný na URL
http://www.root.cz/clanky/vektorove-graficke-formaty-a-metaformaty/

[7] *Performance tests show Java as fast as C++*. Dokument dostupný na URL
http://www.javaworld.com/jw-02-1998/jw-02-jperf.html?page=1

[8] *Java 3D Implementation - OpenGL vs DirectX*. Dokument dostupný na URL
http://java3d.j3d.org/implementation/java3d-OpenGLvsDirectX.html

[9] Wikimedia Foundation, *Head-mounted display*. Dokument dostupný na URL
http://en.wikipedia.org/wiki/Head-mounted_display

[10] *[java3d vs xith3d] Performances*. Dokument dostupný na URL
http://javagaming.org/forums/index.php?topic=3549.0

[11] *FlyingGuns*. Dokument dostupný na URL
http://www.flyingguns.com/

[12] Sun Microsystems, *The Java 3D API Specifications*. Version 1.3, červen 2002

[13] Sun Microsystems, *JSR 231: Java Binding for the OpenGL API*. Version 1.3, červen 2002
http://jcp.org/en/jsr/detail?id=231

[14] Sowizral, Henry A., Deering, Michael F. *The Java 3D API and Virtual Reality*.
květen/červen 1999
http://csdl.computer.org/dl/mags/cg/1999/03/g3012.pdf

[15] Wikimedia Foundation, *Java Native Interface*. Dokument dostupný na URL
http://en.wikipedia.org/wiki/Java_Native_Interface

Seznam příloh

Příloha 1.

Manuál k používání hry SpaceFlight

Popis aplikace

Tato aplikace slouží jako ukázková hra využívající rozhraní Java 3D. Hra je napsána v jazyce Java a pro její přeložení ze zdrojového kódu je potřeba k dispozici JDK 1.6.

Překlad aplikace

Pro překlad hry ze zdrojového kódu byl vytvořen build soubor a je možné ho provést pomocí nástroje Apache Ant s parametrem compile. Aplikace se přeloží do adresáře run.

Spuštění aplikace

Pro běh aplikace je nezbytná instalace Javy 3D. Na CD je spolu se zdrojovými texty k dispozici instalátor Javy 3D verze 1.5.1 pro operační systém Windows XP a Linux. Binární soubory Javy 3D lze rovněž stáhnout z https://java3d.dev.java.net/binary-builds.html .

Po překladu je v adresáři run vytvořen pro operační systém Windows XP batch soubor **SpaceFlight.bat**, přes který lze hru spustit.

Aplikaci lze spustit v libovolném operačním systému pomocí následujícího příkazu:

```
java -Xms256m -Xmx512m -cp "spaceflight.jar;ncsa/portfolio.jar" SpaceFlight
```

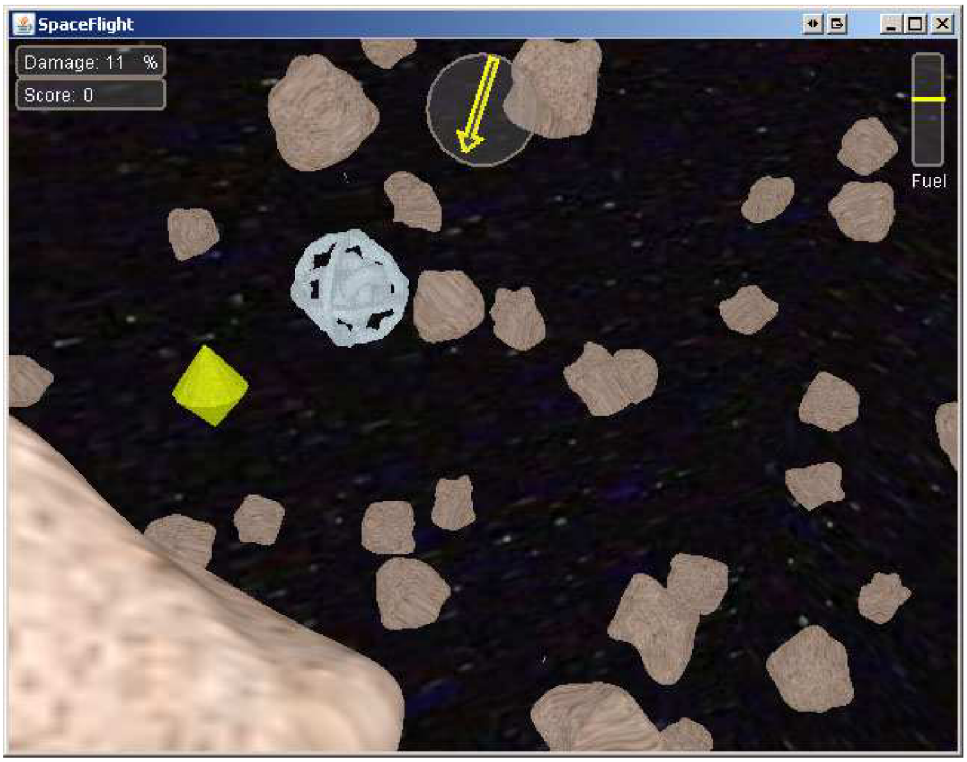
Hru lze spustit s parametrem --level *cislo* nebo -l *cislo* a určit tak počáteční úroveň hry.

Uživatelské rozhraní aplikace

Po spuštění hry se otevře herní okno s textem „Press S key to start…“ vyzívající uživatele ke stisku klávesy S k započetí hraní. Již v této fázi je v okně vidět uživatelské rozhraní hry. V levém horním rohu se nachází pod sebou množství poškození lodi (Damage) a počet získaných bodů (Score), v pravém horním rohu je indikátor množství paliva (Fuel) a uprostřed horní části herního okna je ukazatel, kde se nachází objekt pro doplnění paliva. Během hry lze stiskem klávesy Escape hru pozastavit a jejím opětovným stiskem ve hře opět pokračovat.

Průletem vesmírem hráči postupně klesá hladina paliva, kterou může obnovit sebráním objektu s palivem. Při nárazu na meteorit, tedy jeho průchodem, je v každém okamžiku, kdy se uživatel s meteoritem střetává, zvýšeno množství poškození. Hra končí, pokud poškození dosáhne hranice 100% nebo hladina paliva klesne na minimum. Při dosažení určitého množství bodů dojde ke zvýšení úrovně hry a stoupne rychlost pohybu lodi a změní se počet objektů ve hře. Cílem hry je mít na jejím konci co nejvyšší skóre.

Obrázek 1: Okno hry těsně po spuštění



Obrázek 2: Okno během hraní. Sebrání žlutého předmětu zvýší hráči skóre, sebrání modrého předmětu se hráči doplní hladina paliva.

Obrázek 3: Pozastavení hry po stisku klávesu escape

Obrázek 4: Konec hry – množství poškození dosáhlo 100%

Příloha 2.

Manuál k používání testovací aplikace

Popis aplikace

Tato testovací aplikace slouží ke spuštění testovací úlohy podle zvolených parametrů pro testování rozhraní Java 3D. Aplikace je napsána v jazyce Java a pro její přeložení ze zdrojového kódu je potřeba k dispozici JDK 1.6.

Překlad aplikace

Pro překlad aplikace byl vytvořen build soubor a je možné ho provést pomocí nástroje Apache Ant s parametrem compile. Aplikace se přeloží do adresáře run. Spolu se zdrojovými kódy aplikace je ve zdrojovém adresáři k dispozici soubor s testovacími daty TestingObjects.txt. Tento soubor je po přeložení zkopírován do adresáře run.

Spuštění aplikace

Pro běh aplikace je nezbytná instalace Javy 3D. Na CD je spolu se zdrojovými texty k dispozici instalátor Javy 3D verze 1.5.1 pro operační systém Windows XP a Linux. Binární soubory Javy 3D lze rovněž stáhnout z https://java3d.dev.java.net/binary-builds.html .

Po překladu je v adresáři run vytvořen pro operační systém Windows XP batch soubor **tester.bat**, přes který lze po dodání parametrů testování spustit.

Aplikaci lze spustit v libovolném operačním systému pomocí následujícího příkazu:

```
java -Xms256m -Xmx512m -cp "tester.jar" Tester parametry
```

Nastavení testů lze měnit spuštěním aplikace s určitými parametry. Seznam těchto parametrů je v následující tabulce:

Parametr	**Hodnota paramteru**	**Popis paramtru**	**Defaultní hodnota**
--objects -o	unsignedint	Nastaví počet objektů v testovací scéně	100
--colidgeolevel -cgl	unsignedint	Nastaví počet segmentů detekčních koulí pro test kolize. Pokud je nastavena hodnota 0, použije se defualtní hodnota pro počet segmentů Javy 3D, tedy 15.	0 (defaultní hodnota Javy3D)
--tree -t	0/1/2/3	Nastaví způsob vložení testovacích objektů do grafu scény 0 – Referenční (vložení do kořenového uzlu) 1 – Vyvážený strom 2 – Nevyvážený strom 1 3 – Nevyvýžený strom 2	0
--rotate -r	intbool	Zapne/vypne rotaci kamery kolem scény	1 (zapnuto)

--scenerotate -sr	intbool	Zapne/vypne rotaci scény místo kamery, pokud je zapnuta rotace kamery.	1 (zapnuto) 0 (vypnuto)
--frames -f	unsignedint	Nastaví počet snímků, po který aplikace poběží.	5000
--miliseconds -ms	unsignedint	Nastaví počet milisekund, po který aplikace poběží.	12000
--camfly -cf	intbool	Zapne/vypne průlet kamery ve směru pohledu	0 (vypnuto)
--compile -c	intbool	Zapne/vypne kompilaci větve s testovacími objekty	1 (zapnuto)
--collisiontest -ct	intbool	Zapne/vypne test na detekci kolizí s detekčními koulemi.	0 (vypnuto)
--collisiondetect -cd	intbool	Zapne/vypne detekční chování pro detekční koule.	0 (vypnuto)
--collisiongeometry -cg	intbool	Zapne/vypne testování detekce kolizí podle geometrie namísto testování podle hranic objektu.	0 (vypnuto)
--attachtest -ad	intbool	Zapne/vypne test na připojování větví	0 (vypnuto)
--attach -at	intbool	Zapne/vypne připojování větve s testovacími objekty za chodu	0 (vypnuto)
--fraps	intbool	Zapne/vypne programové vyvolání stistku klávesy F11 před spuštěním testu. Tato funkčnost je vytvořena speciálně pro aplikaci fraps.	0 (vypnuto)
--flypos -fp	float	Nastaví konečnou pozici kamery při průletu kamery ve směru pohledu	250.0
--objectsfile -of	string	Nastaví soubor, ze kterého budou načítány pozice a velikosti objektů. Co řádek v souboru, to definice jendoho objektu. Struktura řádku je následující: pozice_x pozice_y pozice_z velikost	TestingObjects.txt
--direct3d -d3d	intbool	Zapne/vypne použití pro renderování Direct3D místo OpenGL	0 (vypnuto)

Poznámky:
float značí hodnotu typu float se znaménkem
intbool značí hodnotu 0 pro false(vypnuto) a hodnotu 1 pro true(zapnuto), ostatní hodnoty nemají na nastavení vliv
unsignedint značí hodnotu typu integer bez znaménka
string značí řetězcovou hodnotu

Uživatelské rozhraní aplikace

Veškeré nastavení aplikace se provádí pomocí parametrů spuštění aplikace. Po spuštění aplikace je otevřeno okno, ve kterém je spuštěn test a po jeho úspěšném provedení je okno zavřeno a výsledek testu je vypsán na standardní výstup.

Výsledné hodnoty jsou na standardní výstup zapsány na jednom řádku, odděleny středníkem a jsou v následujícím pořadí:

- Datum a čas spuštění testu, pokud je program spuštěn s parametrem fraps

- Způsob vložení objektů do grafu scény (NONE/BALANCED/UNBALANCED_V1/ UNBALANCED_V2)
- Počet testovacích objektů
- Počet snímků za vteřinu
- Počet snímků vyrenderovaných během testu
- Počet kolidovaných objektů, pokud je zapnuto detekční chování u detekce kolizí

Příklad spuštění aplikace

tester.bat -t 0 -o 1000 -f 2500 -of TestingObjects.txt -r 1 -c 1

nebo

java -Xms256m -Xmx512m -cp "tester.jar" Tester -t 0 -o 1000 -f 2500 -of TestingObjects.txt -r 1 -c 1

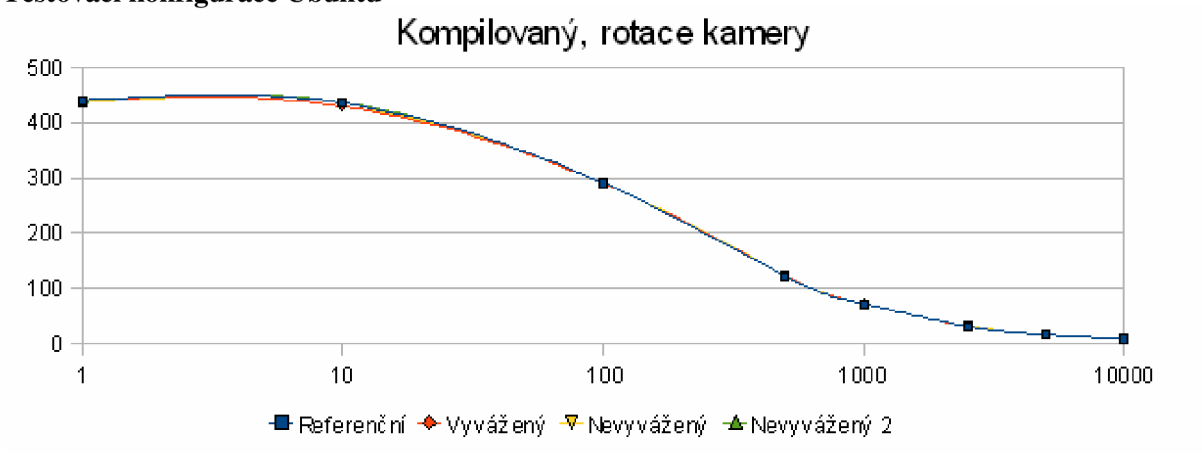
Příloha 3.

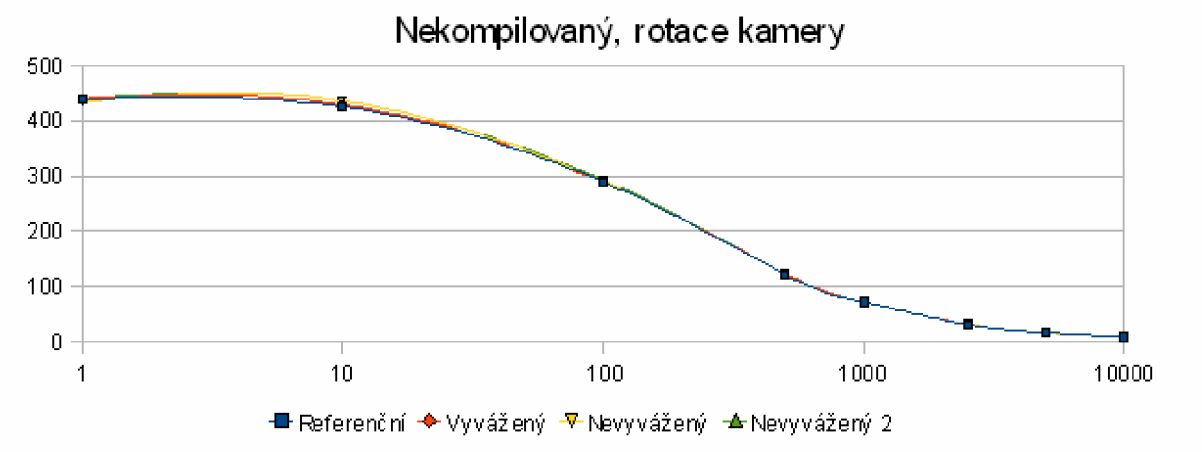
Výsledné grafy provedených testů

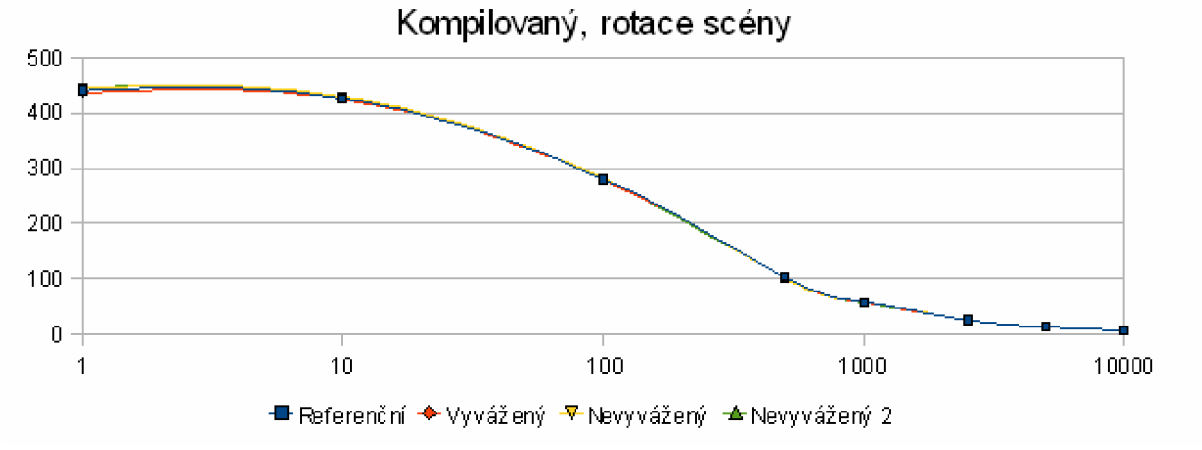
1. Test závislosti rozložení objektů v grafu scény

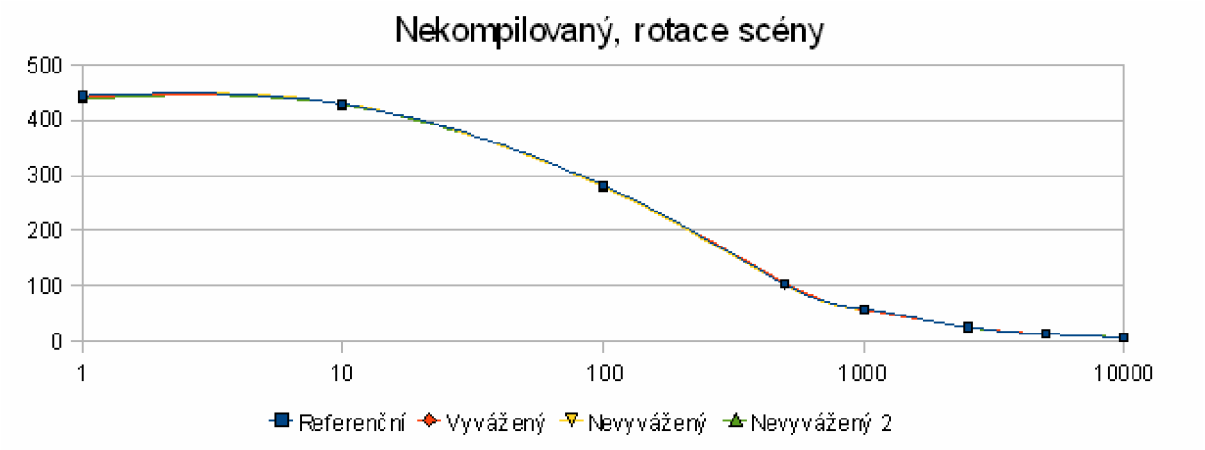
Poznámka: Na horizontální ose následujících grafů je vynesen počet testovacích objektů, na vertikální ose počet snímků za vteřinu

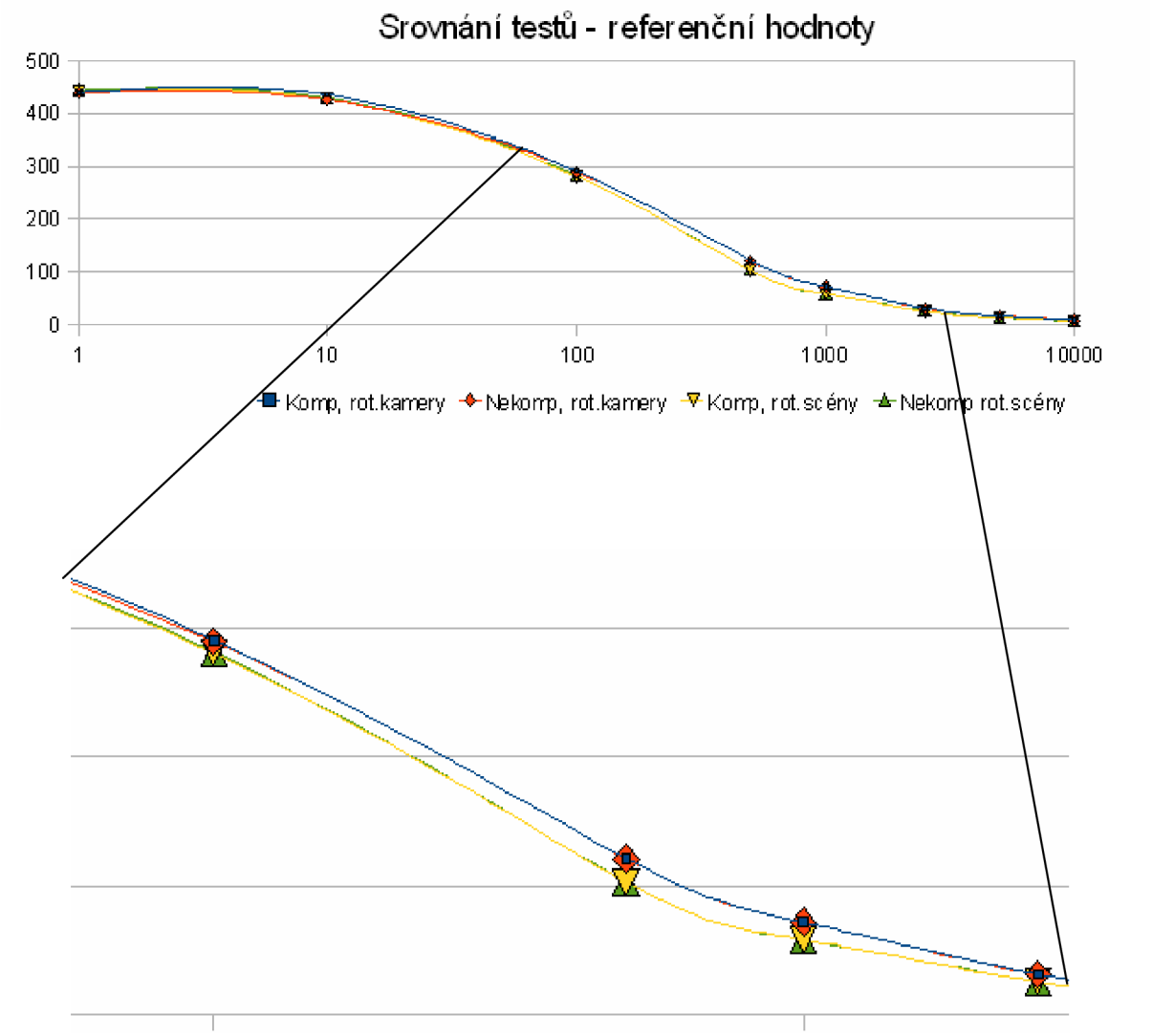
Testovací konfigurace Ubuntu



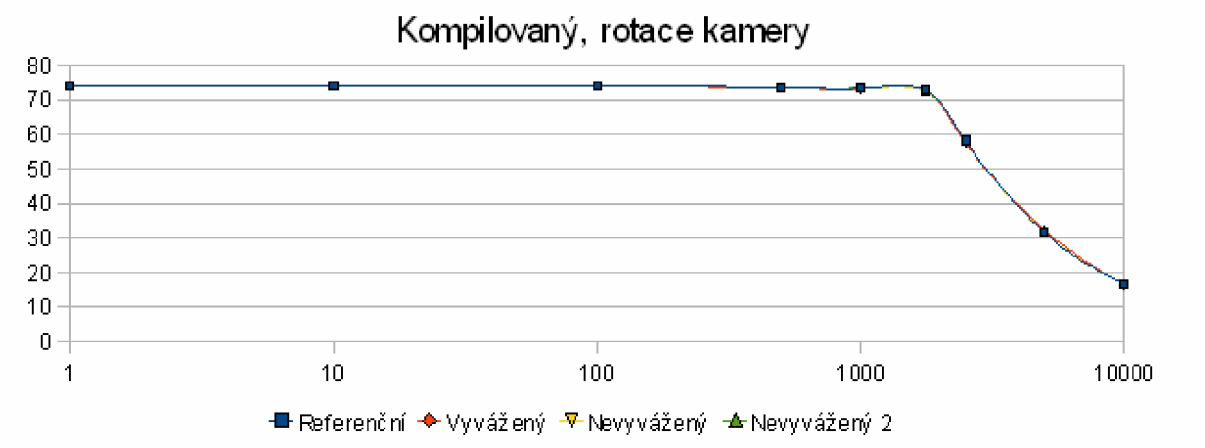


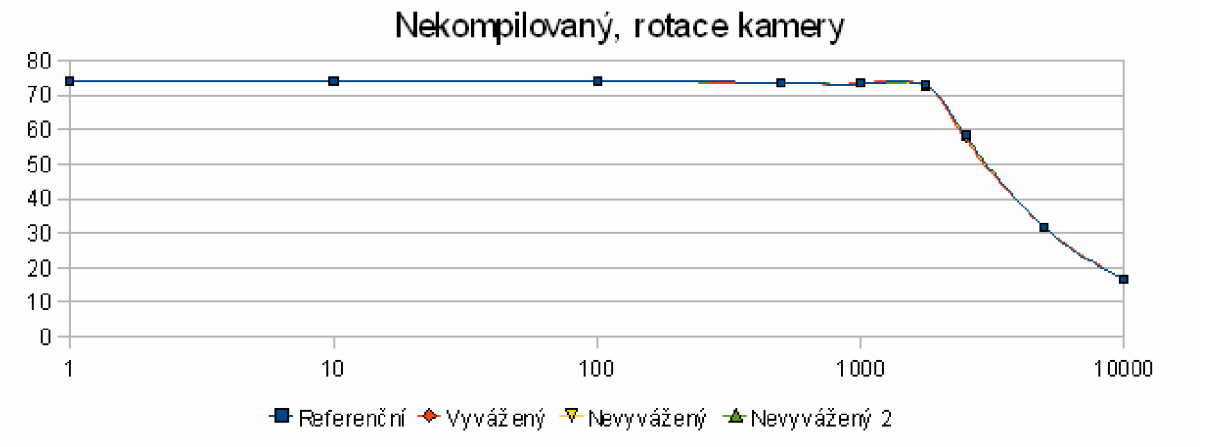


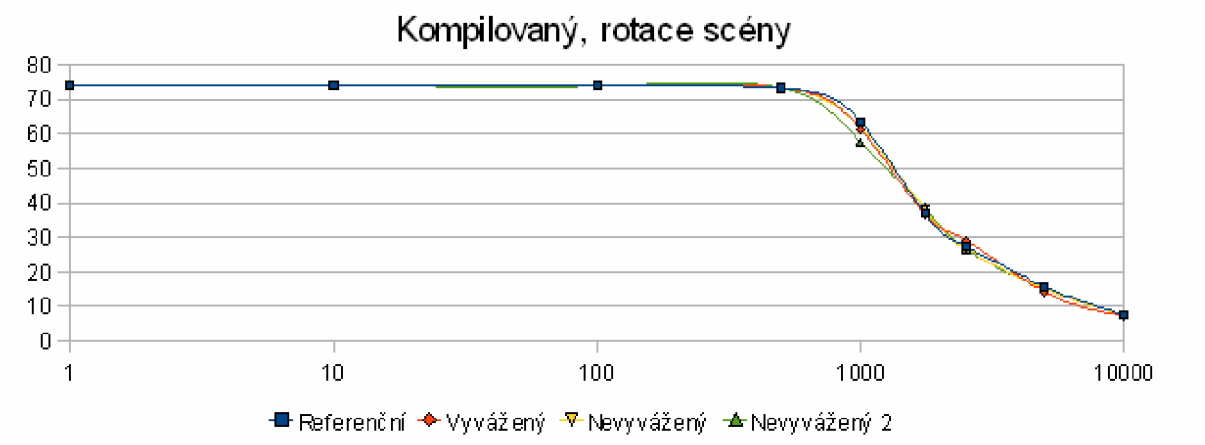

Nekompilovaný, rotace scény
500
400
300
200
100
0
1
10
100
1000
10000
Referenční
Vyvážený
Nevyvážený
Nevyvážený 2

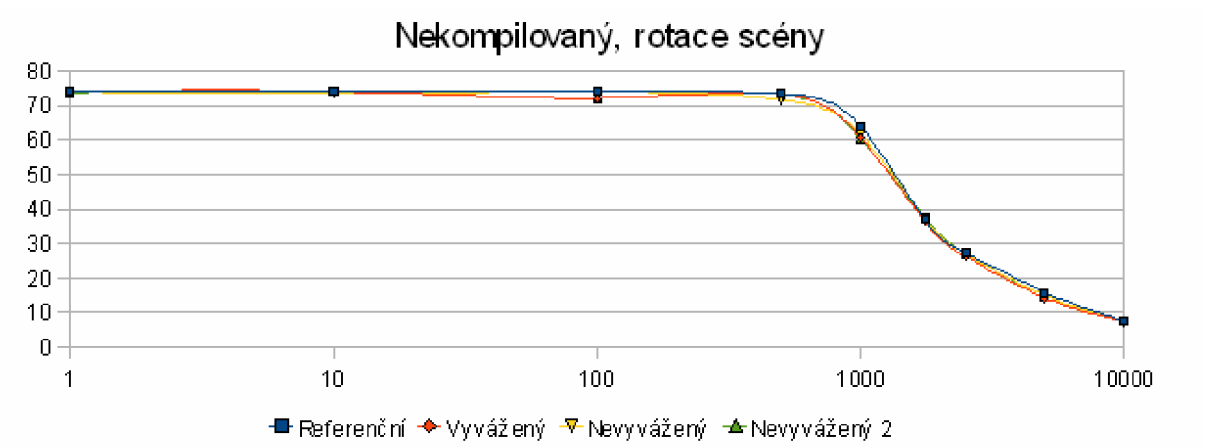

Srovnání testů - referenční hodnoty
500
400
300
200
100
0
1
10
100
1000
10000
Komp, rot.kamery
Nekomp, rot.kamery
Komp, rot.scény
Nekomp rot.scény

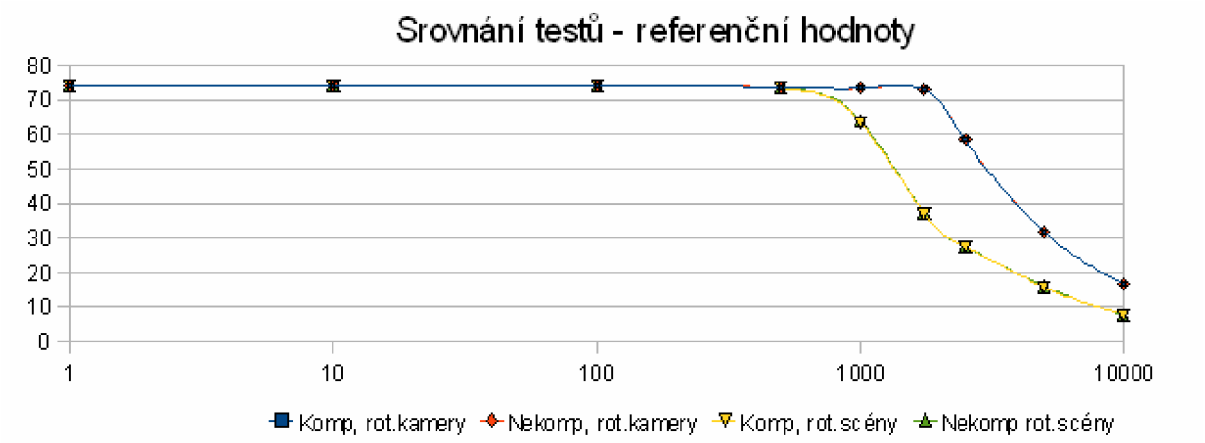
Testovací konfigurace WinXP





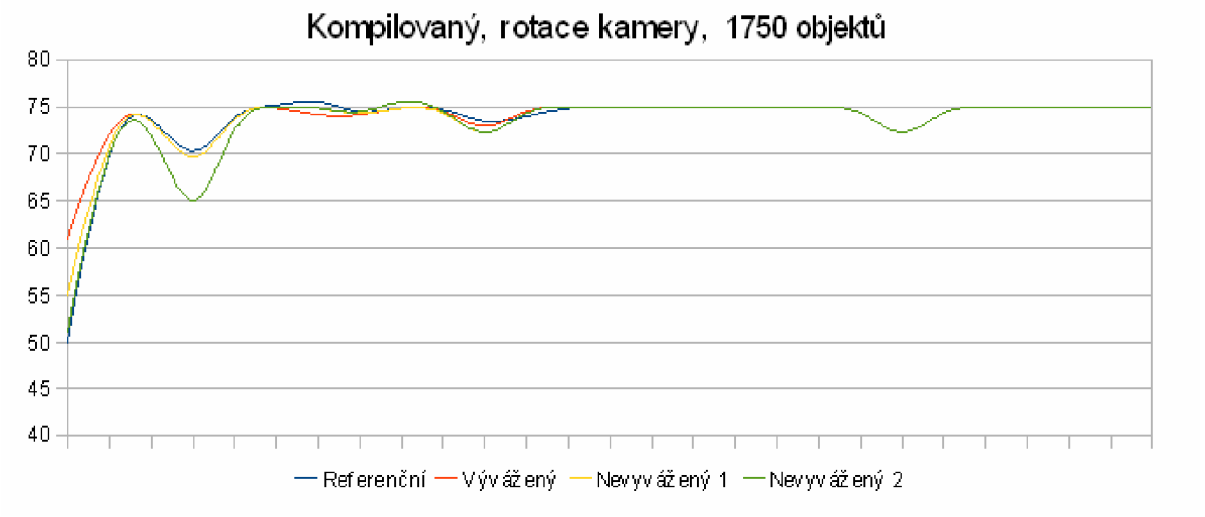


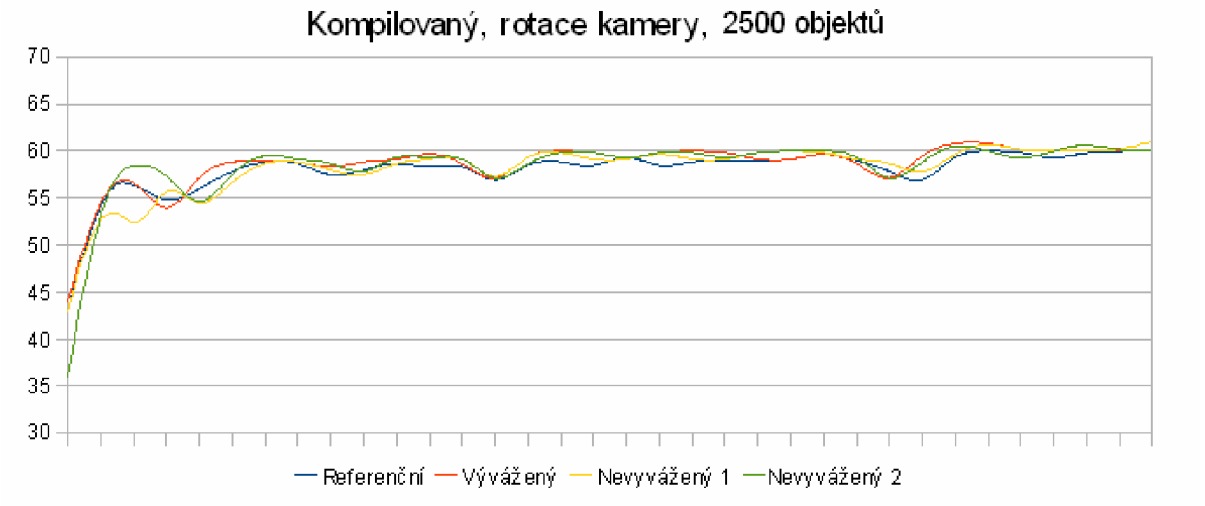


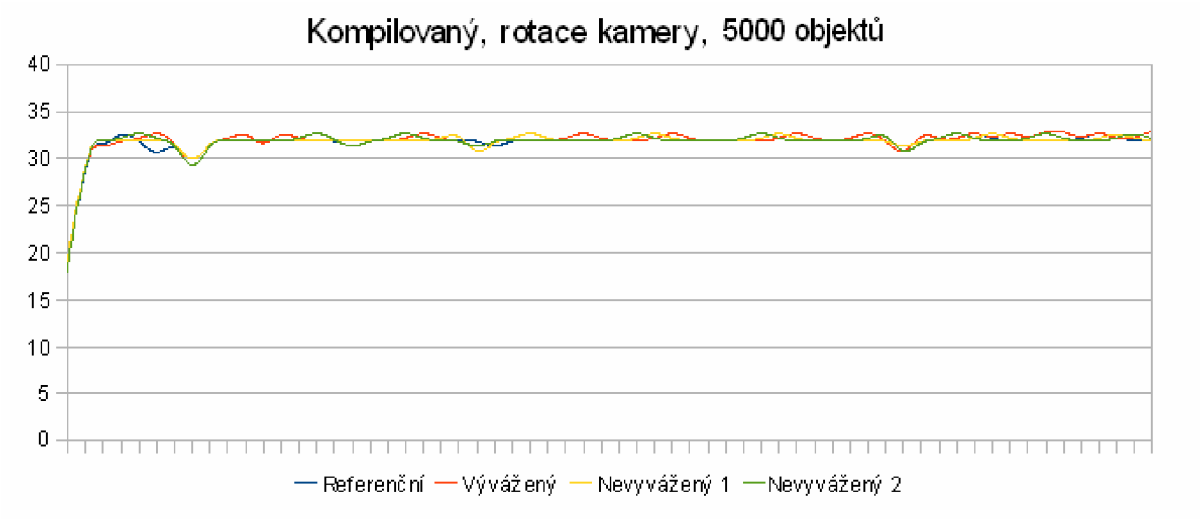


Testovací konfigurace WinXP – Grafy průběhu snímkovací frekvence během testu

Poznámka: Na vertikální ose následujících grafů je vynesen počet snímků za vteřinu, na horizontální ose jednotlivé časové úseky vzorků



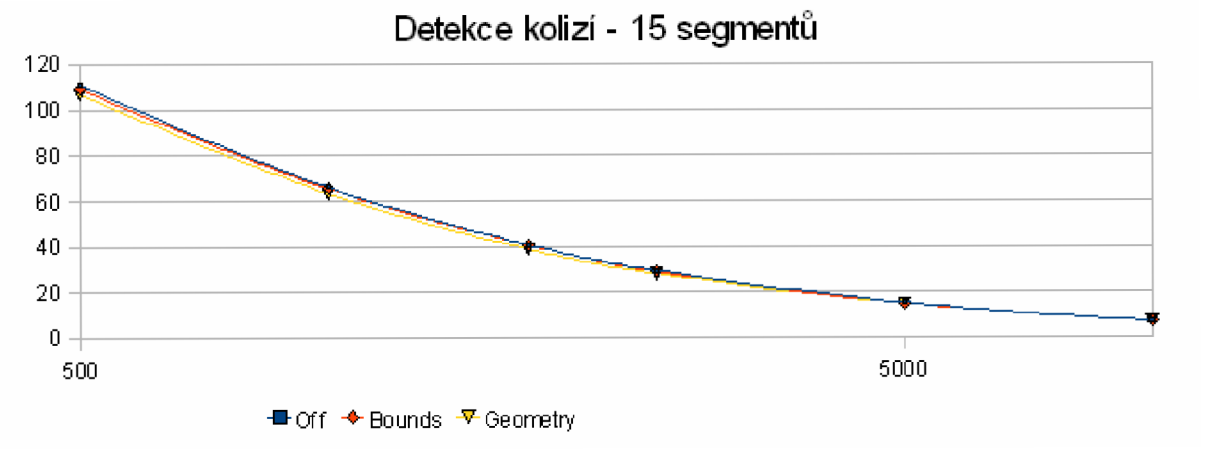


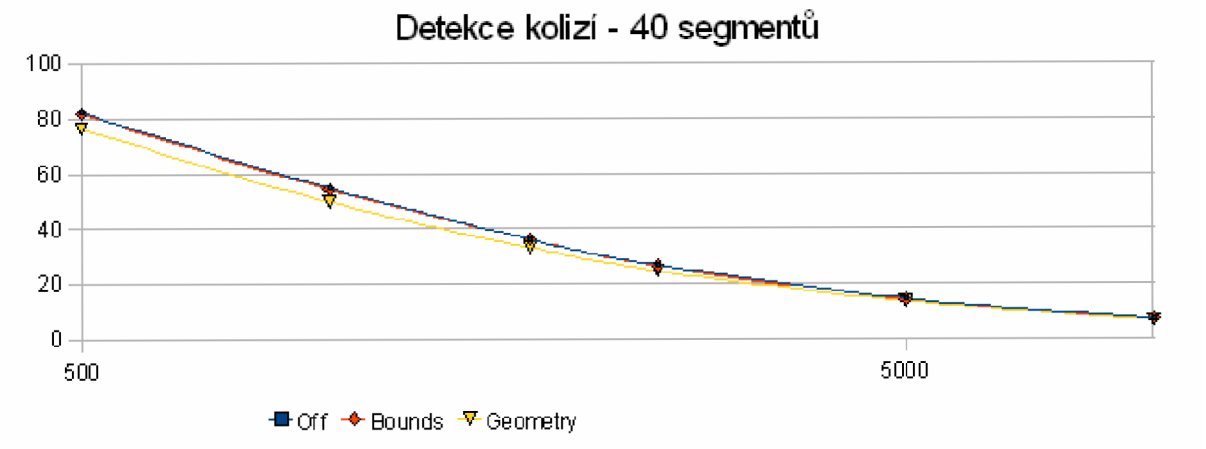

Kompilovaný, rotace kamery, 5000 objektů
40
35
30
25
20
15
10
5
0
Referenční
Vyvážený
Nevyvážený 1
Nevyvážený 2

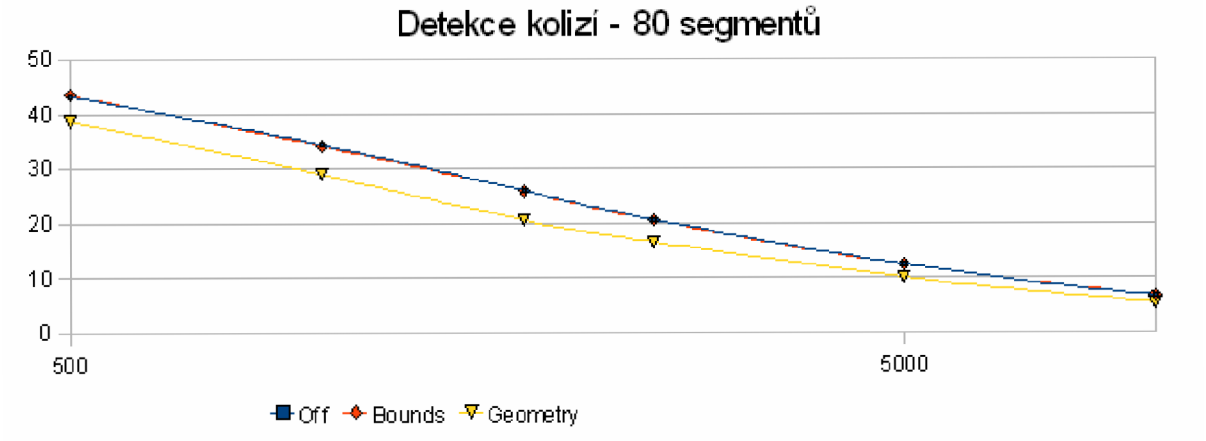
2. Test detekce kolizí

Poznámka: Na horizontální ose následujících grafů je vynesen počet testovacích objektů, na vertikální ose počet snímků za vteřinu

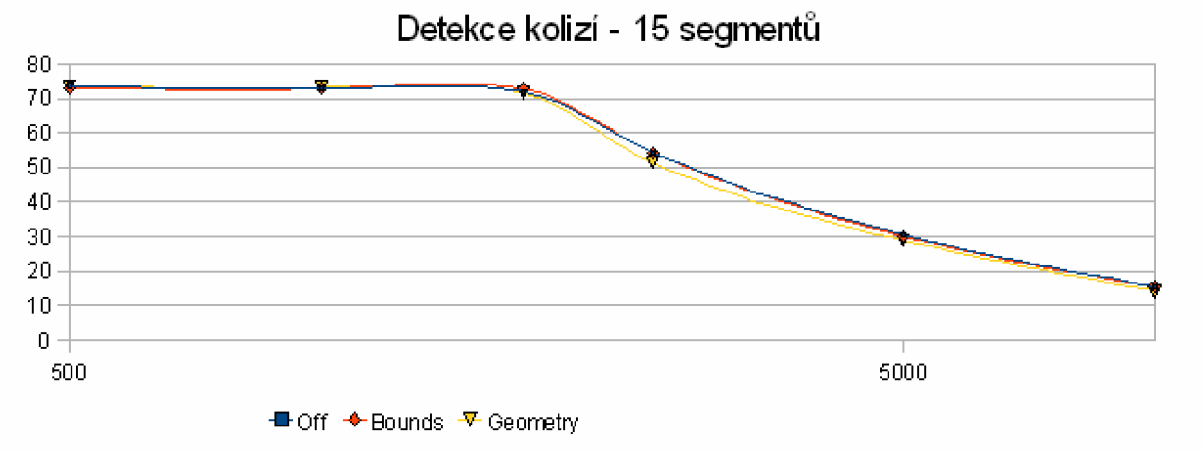
Testovací konfigurace Ubuntu

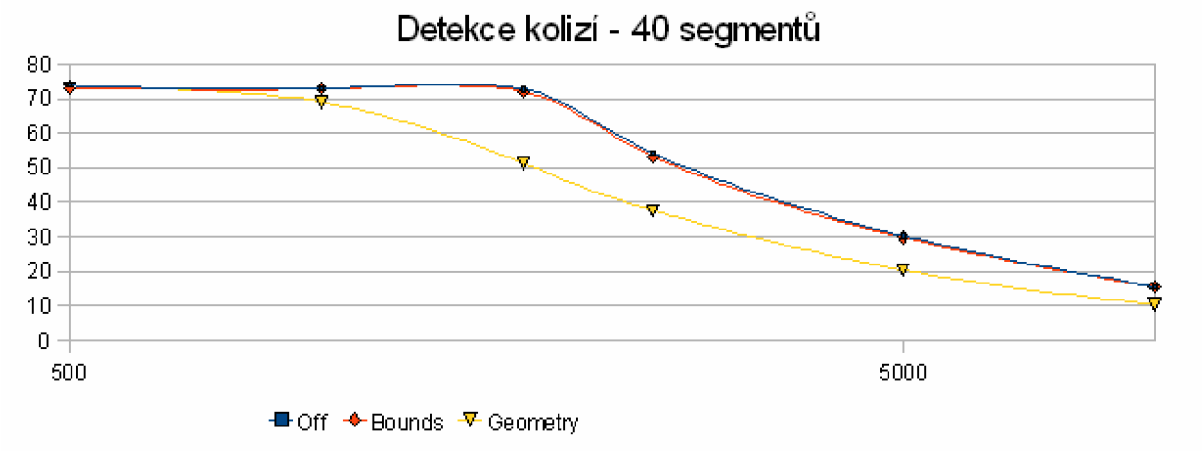


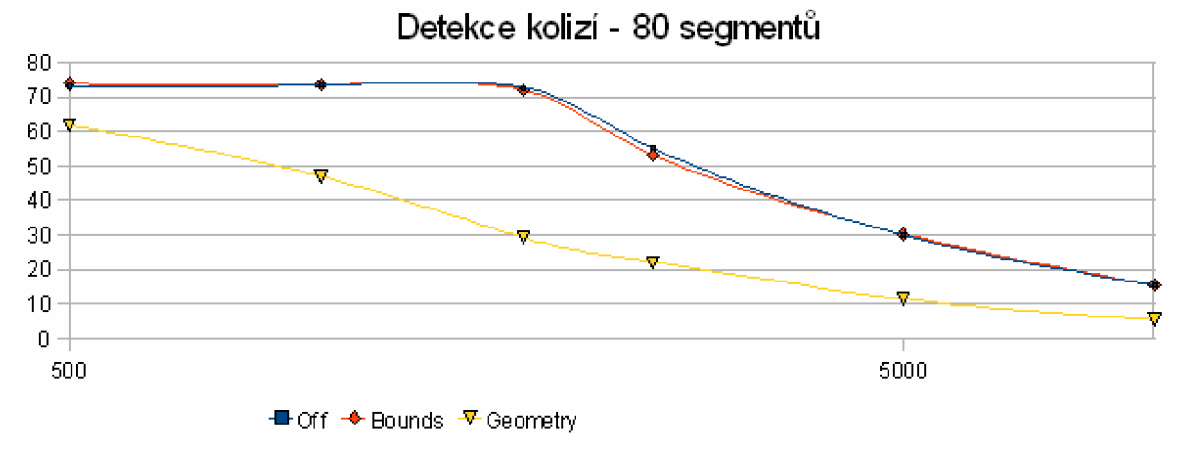




Testovací konfigurace WinXP



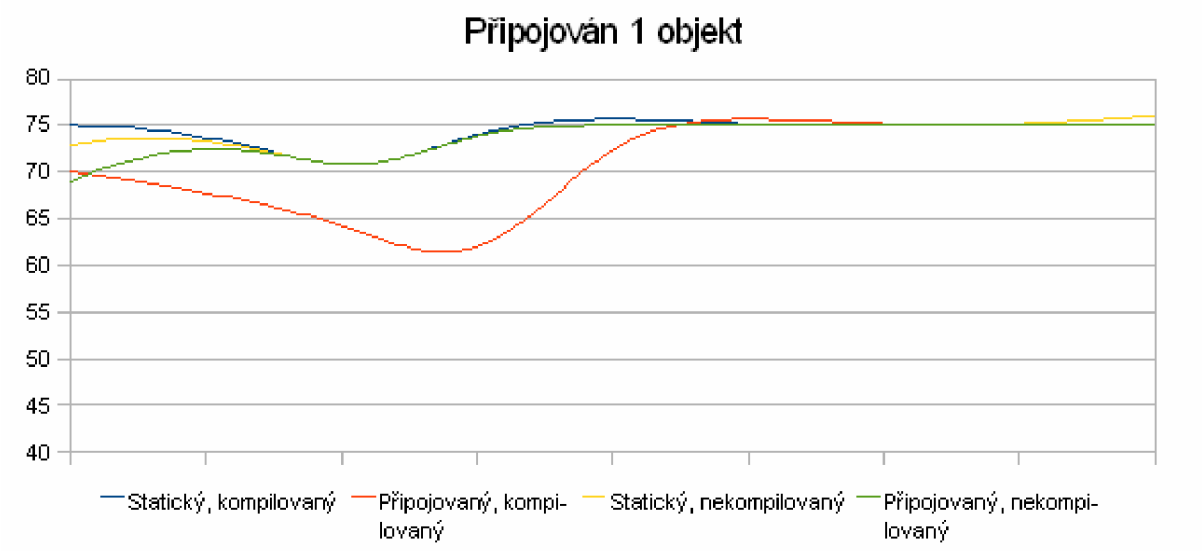


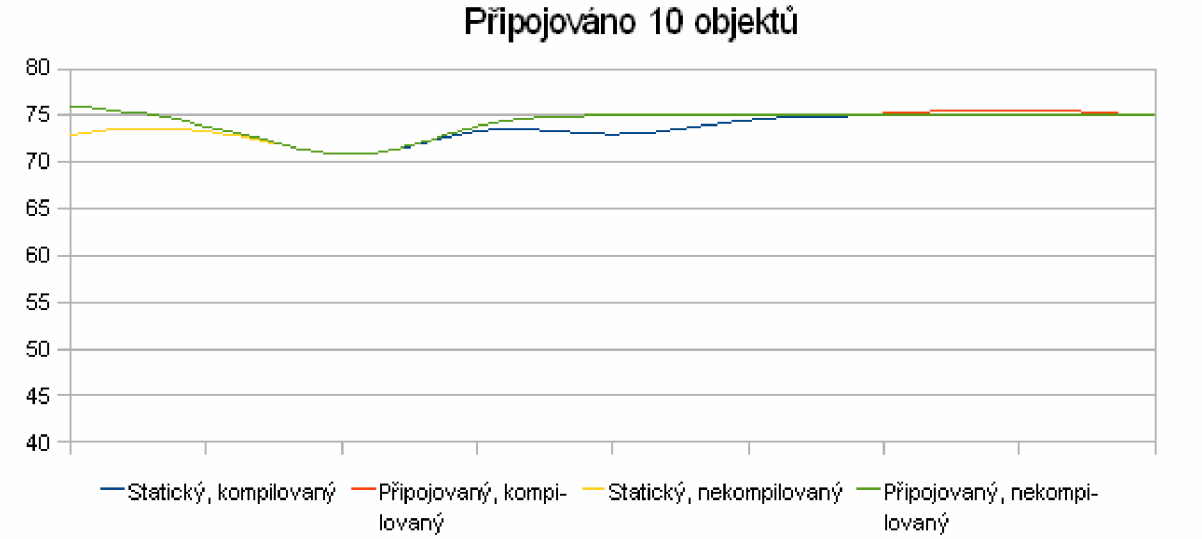


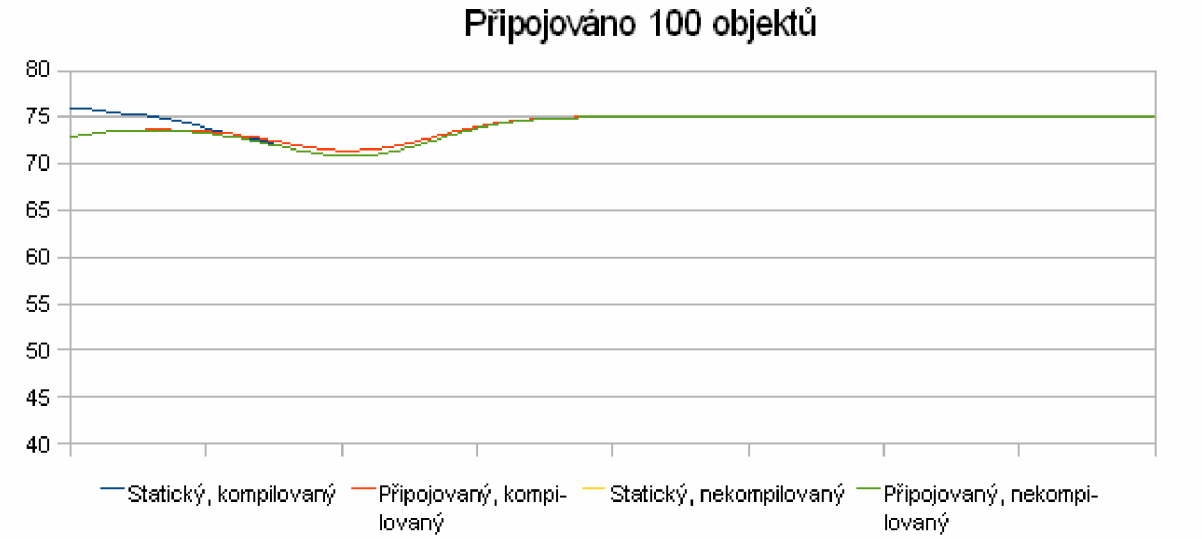
3. Test připojování větví do grafu scény za chodu

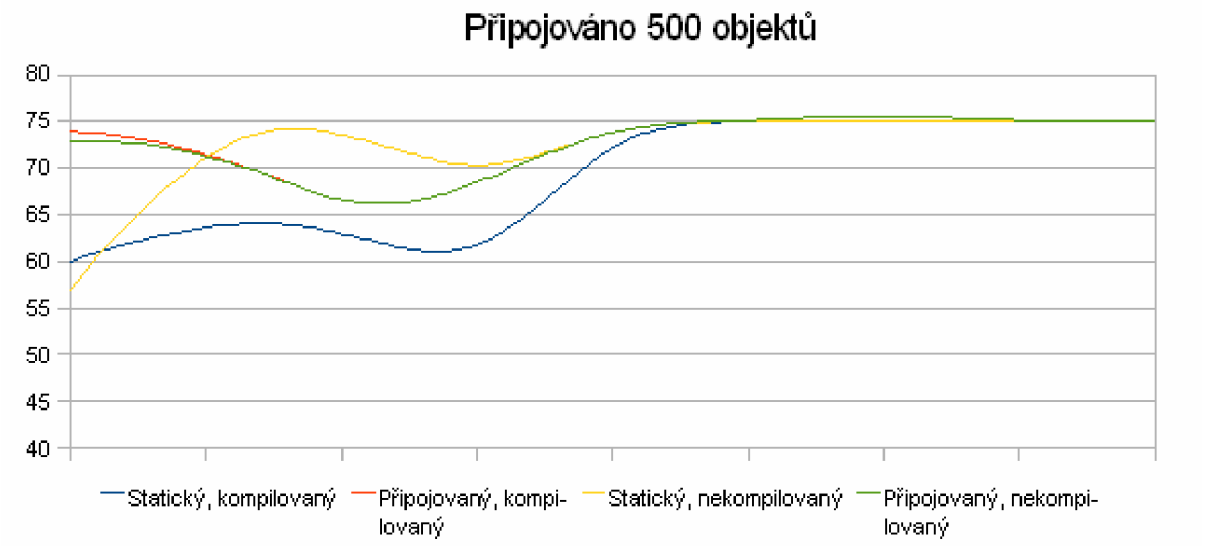
Testovací konfigurace WinXP – Grafy průběhu snímkovací frekvence během testu

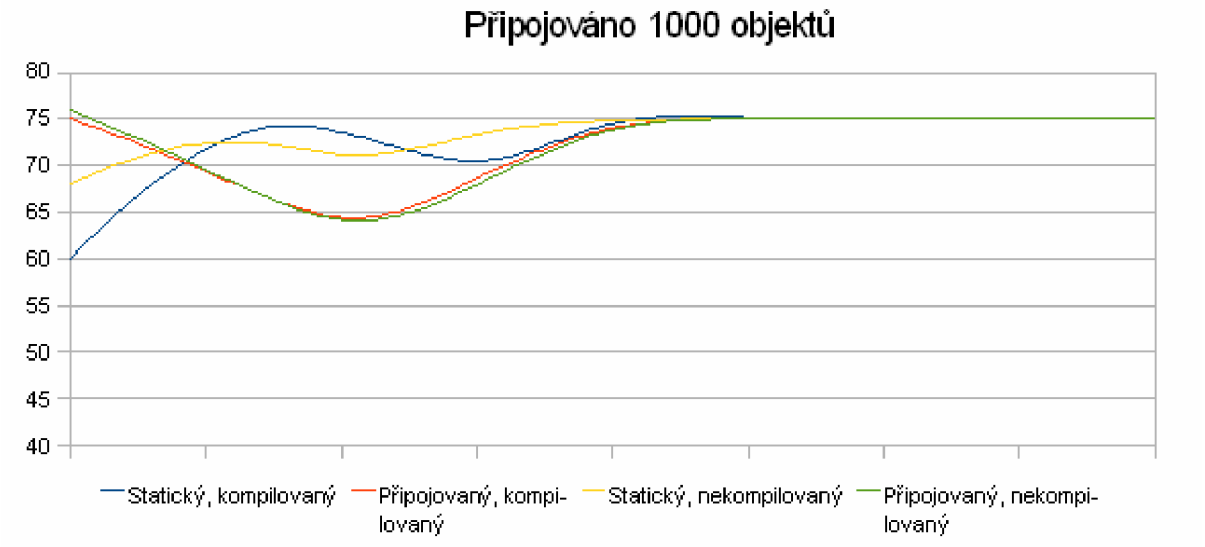
Poznámka: Na vertikální ose následujících grafů je vynesen počet snímků za vteřinu, na horizontální ose jednotlivé časové úseky vzorků

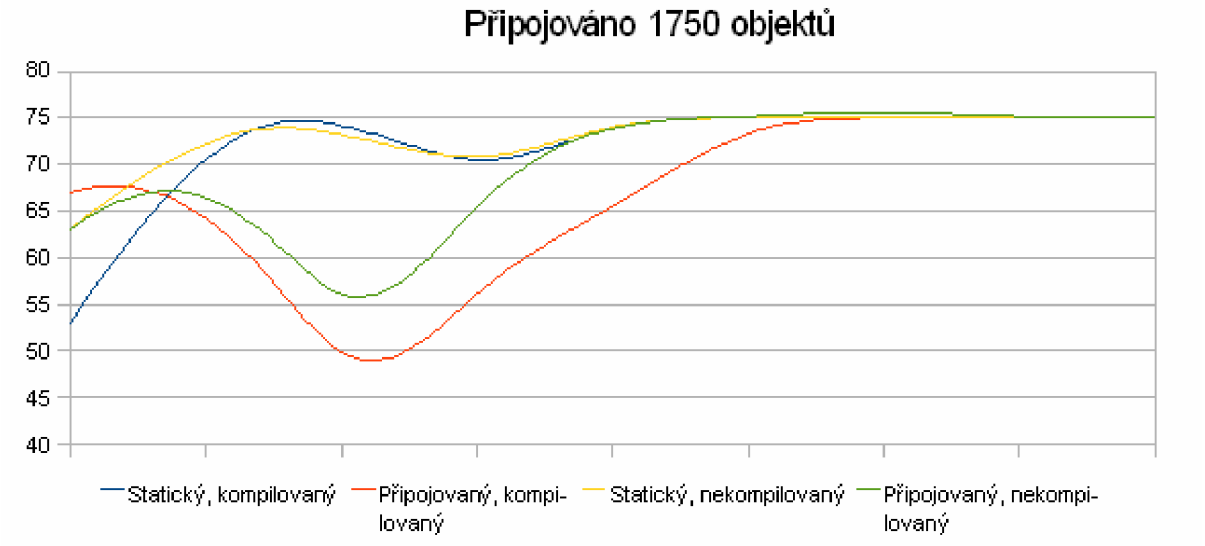


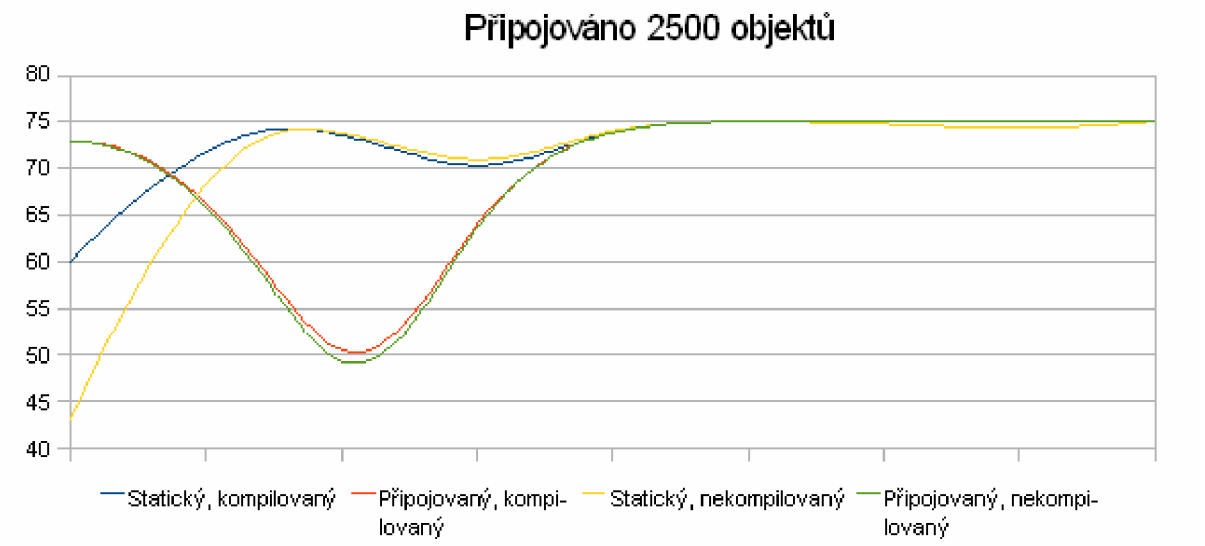


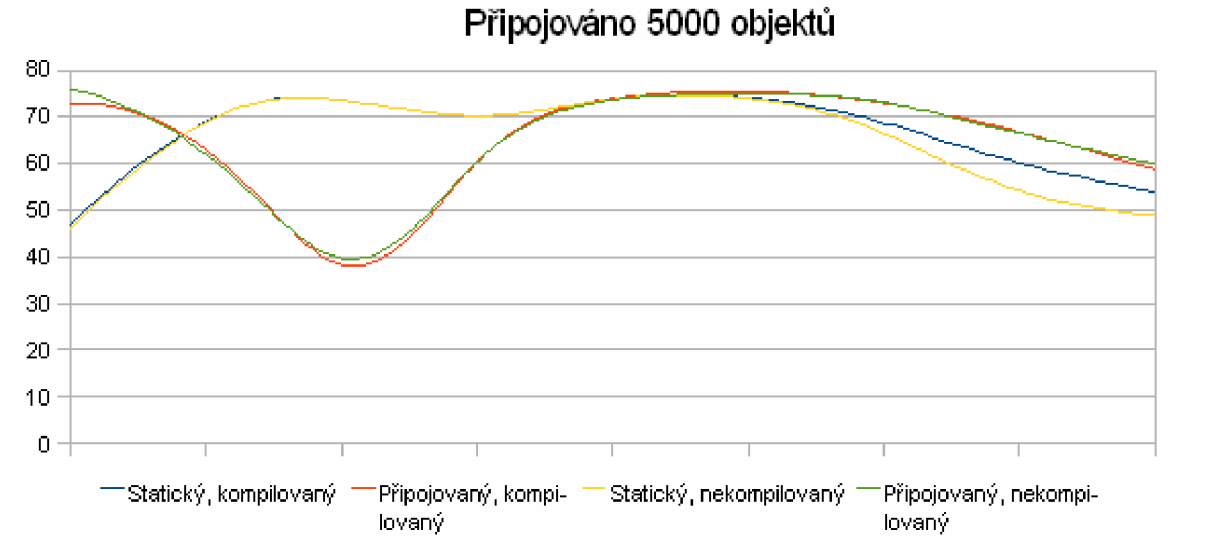


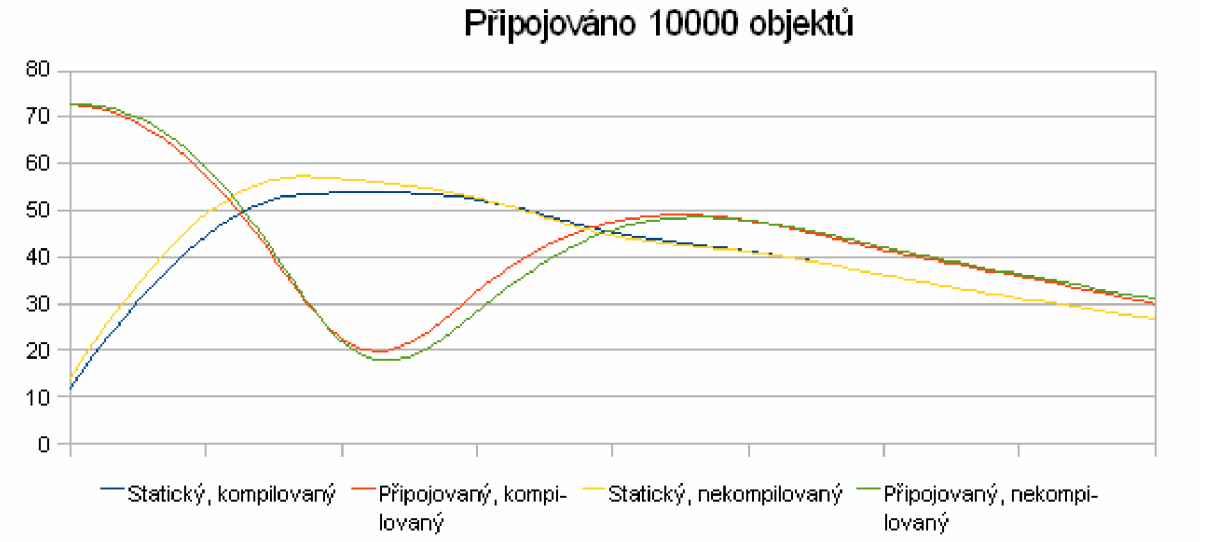

Připojováno 500 objektů
80
75
70
65
60
55
50
45
40
Statický, kompilovaný
Připojovaný, kompi-lovaný
Statický, nekompilovaný
Připojovaný, nekompi-lovaný


Připojováno 1000 objektů
80
75
70
65
60
55
50
45
40
Statický, kompilovaný
Připojovaný, kompi-lovaný
Statický, nekompilovaný
Připojovaný, nekompi-lovaný


Připojováno 1750 objektů
80
75
70
65
60
55
50
45
40
Statický, kompilovaný
Připojovaný, kompi-lovaný
Statický, nekompilovaný
Připojovaný, nekompi-lovaný


Připojováno 2500 objektů
80
75
70
65
60
55
50
45
40
Statický, kompilovaný
Připojovaný, kompi-lovaný
Statický, nekompilovaný
Připojovaný, nekompi-lovaný


Připojováno 5000 objektů
80
70
60
50
40
30
20
10
0
Statický, kompilovaný
Připojovaný, kompi-lovaný
Statický, nekompilovaný
Připojovaný, nekompi-lovaný


Připojováno 10000 objektů
80
70
60
50
40
30
20
10
0
Statický, kompilovaný
Připojovaný, kompi-lovaný
Statický, nekompilovaný
Připojovaný, nekompi-lovaný

Testovací konfigurace WinXP – průměrné hodnoty snímků za vteřinu

Poznámka: Na horizontální ose grafu je vynesen počet testovacích objektů, na vertikální ose počet snímků za vteřinu