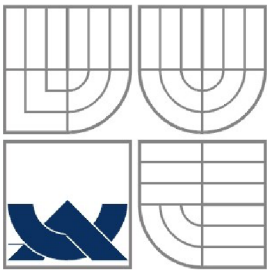
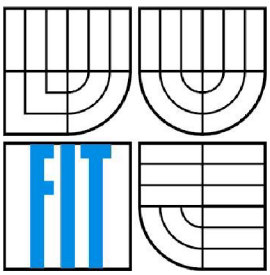


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MĚŘENÍ VÝKONNOSTI GRAFICKÉHO AKCELERÁTORU

PERFORMANCE EVALUATION OF GRAPHICS ACCELERATOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JURAJ VANEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2010

Abstrakt

Tato práce pojednává o možnostech a funkcích moderních grafických akcelerátorů a měření jejich výkonu pod rozhraním OpenGL. Využívají se při tom rozšířené algoritmy na zobrazování scény v reálném čase. Zaměřuje se na otestování každé části grafického řetězce akcelerátoru podobně, jako na měření rychlosti zobrazení pokročilých efektů a teoretickou rychlost obecných výpočtů pomocí grafického procesoru. Toto testování je realizováno pomocí více sérií testů a jejich následném vyhodnocení. Výsledná aplikace umožňuje nastavovat parametry testů a jejím výstupem je skóre, podle kterého je možné srovnávat výkon akcelerátoru s ostatními.

Abstract

This paper deals with possibilities and functions of modern graphic accelerators and with measuring performance under OpenGL interface. Widespread algorithms to render scene in real-time are used. It focuses on how to test every part of accelerator's graphic pipeline as well as measure performance in rendering of advanced effects and theoretical speed at general purpose calculations through graphic processor. This testing is realized by implementing multiple test series and their further evaluation. Final application enables setting of test parameters and outputs a score, by which is possible to judge accelerator's performance in comparison among themselves.

Klíčové slova

test, výsledky, výkon, počítačová, grafická, karta, vykreslovací řetězec, akcelerátor, skóre, měření, srovnání, OpenGL, GLSL, shader, scéna

Keywords

test, results, performance, computer, graphics, card, rendering pipeline, accelerator, score, measuring comparison, OpenGL, GLSL, shader, scene

Citace

Juraj Vanek: Měření výkonnosti grafického akcelerátoru, diplomová práce, Brno, FIT VUT v Brně, 2010

Měření výkonnosti grafického akcelerátoru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostaně pod vedením Ing. Adama Herouta, Ph.D., přičem další informace a poznatky jsem čerpal z uvedených literárních pramenů a publikací.

……
Juraj Vanek
26.5.2010

Poděkování

Chtěl bych poděkovat Ing. Adamu Heroutovi, Ph.D. za cenné rady a inspirace, které mi velice pomohly při tvorbě této práce. Dále bych chtěl poděkovat všem, kteří mi poskytli různé testovací platformy na vývoj a následné testování aplikace.

Obsah

1 Úvod

Za poslednú dekádu sme svedkami prudkého rozvoja grafických akcelerátorov, ktoré už prestávajú byť zariadením používaným výhradne na hry, ale začínajú sa používať aj vo všeobecných výpočtoch. Moderný grafický akcelerátor dnes ponúka výkon superpočítača spred desiatich rokov za zlomkovú cenu. Spolu s tým vzrastá realita grafického výstupu aplikácií pracujúcich v reálnom čase, a to najmä vďaka tlaku herného priemyslu. Súčasné 3D hry a aplikácie ponúkajú efekty, ktoré sa čím viacej približujú reálnemu obrazu (aj keď stále ide len o aproximáciu fyzikálnych zákonov, napríklad pri šírení svetla).

Keďže na trhu existuje veľké množstvo grafických akcelerátorov s rôznou hardwarovou výbavou a funkciami, je potrebné mať nástroje na efektívne porovnávanie ich výkonnosti. Ide nielen o celkový teoretický výkon, ale aj o výkon pri zobrazovaní určitej škály efektov – napríklad jeden akcelerátor môže byť rýchly pri zobrazovaní scény s mnohými trojuholníkmi, ale môže byť pomalý pri aplikácií textúr, prípadne per-pixel efektov. Preto je pomerne dôležité vedieť (napr. z hľadiska nákupu novej grafickej karty), ktorá karta sa najlepšie hodí na požiadavky kladené zvolenou aplikáciou.

Na to sa dajú využiť už spomínané hry alebo profesionálne aplikácie, ktoré však neponúkajú nástroje na priame meranie výkonu, prípadne je obsluha zložitá (skripty v príkazovom riadku). Na efektívne, komfortné meranie boli vytvorené špeciálne aplikácie, ktoré umožňujú nastavovať parametre testu v grafickom rozhraní a ich výstupom býva skóre, podľa ktorého je možné priamo porovnávať výkon. Nevýhodou je, že nemusia presne odzrkadľovať výkon v reálnych aplikáciách. K takýmto benchmarkom (testovým aplikáciám) patria napríklad známe benchmarky 3D Mark od fínskej spoločnosti Futuremark, prípadne množstvo ďalších menších programov, ktoré sú však zamerané len na jednu oblasť výkonu. K profesionálnym benchmarkom je zase možné zaradiť sériu aplikácií SPEC, ktorá meria výkon v profesionálnych 3D aplikáciách.

Táto práca začína popisom možností a vlastností moderných grafických akcelerátorov, pričom sa podrobne zameriame na rozbor grafického reťazca (pipeline) a potenciálne úzke miesta, od ktorých závisí výkon celého zobrazenia. Pri ďalšom popise existujúcich riešení aj mojej aplikácie sa bude vychádzať práve z grafického reťazca – aké operácie a testy zaťažujú jeho konkrétnu časť. Aplikácia a jej testy je teda navrhnutá tak, aby komplexne zaťažila každý prvok grafického reťazca, ako aj celý reťazec pri zobrazovaní náročných efektov používaných v realistickom zobrazovaní.

Ďalej nasleduje popis existujúcich riešení na meranie výkonu so zameraním sa na rozsiahle benchmarky s mnoho testami (3D Mark, SPEC). Spomenuté sú aj graficky pokročilé herné enginy používané v najnovších hrách.

V ďalšej kapitole nasleduje podrobný popis implementovaných sérií testov a algoritmov, ktoré používajú. Algoritmy sú ale popísané len zbežne (s odkazmi na publikácie), viacej sa budeme zameriavať hlavne na to, aký účel má daná testová séria/efekt a akú má váhu vo výslednom hodnotení.

Popisom výslednej aplikácie sa zaoberá piata kapitola. Popíšeme si aplikáciu a použitý grafický engine (aké využíva technológie a princíp funkcie). Dôležitou časťou je popis ovládania aplikácie, získavanie informácií o systéme a popis formátu výstupných súborov. Nakoniec si zhrnieme váhy jednotlivých testov, výpočet celkového skóre a jeho uplatnenie v on-line databázi výsledkov.

Posledná kapitola sa zaoberá systémom zberu výsledkov a ich analýzou. Výkonnosť zobrazenia v jednotlivých testoch bude podrobne rozobraná na príklade vzorových akcelerátorov s podobnými parametrami. Asi najdôležitejšia časť práce je analýza výsledkov z on-line databázy, výsledky a porovnania akcelerátorov na základe rýchlosti vykresľovania špecifických scén.

Táto diplomová práca plynulo naväzuje na semestrálny projekt, z ktorého preberá (s istými rozšíreniami a aktualizáciami) prvé 4 kapitoly a úvodné časti z kapitoly 5. V práci sa vyskytuje veľa pôvodne anglických výrazov, pre ktoré buď neexistuje český/slovenský ekvivalent, alebo by jeho používanie nebolo príliš vhodné. Preto budem uvádzať takéto výrazy najprv s približným prekladom a potom už len samotný výraz. Názvy všetkých grafických kariet a čipov budú uvedené kurzívou.

2 Grafické akcelerátory

Grafické akcelerátory od svojho vzniku (za prvý sa dá považovať *nVidia GeForce 256* z roku 1999, ktorý ako prvý začal akcelerovať 3D transformácie a osvetlenie) prebehli pomerne rýchlym vývojom. V súčasnosti ide o veľmi výkonné jednotky, ktoré sa starajú o vykresľovanie obrazu. V dobe písania tejto správy (máj 2010) sa dajú spoločné vlastnosti akcelerátorov zhrnúť nasledovne [6]:

- počet tranzistorov v jadre dosahuje viac ako dve miliardy
- takty jadra pohybujúce sa nad 800MHz
- masívna paralelizácia, unifikovaná architektúra (stream procesory)
- celkový teoretický výkon takmer 2 TFLOPS
- viacjadrové grafické procesory (multi-GPU)
- grafická pamäť o veľkosti viac ako 2GB
- pamäte typu GDDR5, frekvencie nad 4GHz
- podpora grafických rozhraní DirectX 11 a OpenGL 4.0
- rozhrania pre všeobecné výpočty cez GPU: OpenCL a Direct Compute

Spoločnou vlastnosťou moderných akcelerátorov je plne programovateľný grafický reťazec na grafickom procesore, takže v ďalšom texte sa budeme zameriavať primárne na výkonné grafické karty s podporou shader modelu 4.0 a viac.

2.1 Grafický procesor (GPU)

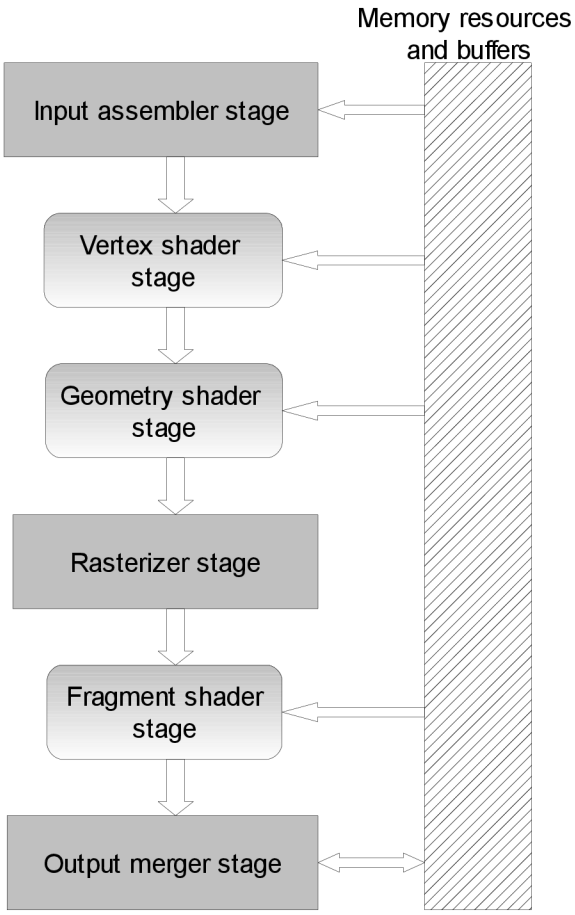
Grafický procesor je hlavné výpočtové centrum grafickej karty, od ktorého najviac závisí celkový výkon. Je optimalizovaný na prácu s desatinnými číslami a obsahuje hardwarovú implementáciu mnohých algoritmov používaných v počítačovej grafike. Aby sme mohli efektívne merať výkon GPU, potrebujeme vedieť, akým spôsobom sa vytvára výsledný obraz z grafických dát. Tento proces je známy ako **grafický reťazec (graphics pipeline**, Obrázok 2.1). Na moderných grafických akcelerátoroch je tento reťazec z veľkej časti programovateľný, čím umožňuje implementáciu rôznych efektov (napr. osvetľovacie modely).

Programovateľná grafická pipeline sa objavila s príchodom tieňovacích jednotiek (**shaders**) a s GPU *GeForce 3*. Postupným vývojom [6] sa shader jednotky vyvinuli z jednoduchých, assemblerom riadených inštrukčných sád s obmedzenou dĺžkou (shader model 1.0) cez vyššie shader jazyky (shader model 2.0, jazyky GLSL, HLSL a Cg), podmienené príkazy a cykly (shader model 3.0) až po plne programovateľné jednotky, schopné vykonávať všeobecné výpočty (stream procesory, shader model 4.0 a 5.0).

Súčasti (stages) grafickej pipeline spoločné pre DirectX 10 a OpenGL 3.2 [5] :

1. **Input assembler** – číta dáta(body, polygóny atď) z bufferov naplnených v 3D aplikácii (napr. vertex buffer objekty pre vrcholové dáta).
2. **Vertex shader** – pracuje s jednotlivými vrcholmi (vertexami), vykonáva ich transformáciu, morfing, prípadne počíta osvetľovací model. Na vstupe aj výstupe má vždy len jeden vertex, nemôže teda vytvárať nové vrcholy. Každý vertex nesie viacero dát (vertexových atribútov) , napr. pozícia, farba, ID atď.

3. **Geometry shader** – pracuje s celými primitívami (body, polygóny), čiže ak má na vstupe napr. trojuholník, pracuje s jeho tromi bodmi. Môže emitovať nové vrcholy, prípadne zahodiť vykreslenie celého primitíva. Novo vytvorené vrcholy sú pripojené k reťazcu a putujú do rasterizačnej fázy. Táto časť grafickej pipeline je nepovinná, môže sa vynechať.
4. **Rasterizer** - orezáva primitíva pohľadovým objemom, vykonáva perspektívnu korekciu, interpoluje hodnoty z vrcholov a pripravuje dáta pre fragment/pixel shader.
5. **Fragment/pixel shader**[1] – pracuje s jednotlivými fragmentami, využíva interpolované hodnoty z vertex alebo geometry shaderu. Pre každý fragment môže počítať napr. osvetľovací model, aplikovať textúru, prípadne množstvo iných efektov. Výstupom je farba fragmentu, prípadne je možné fragment vylúčiť z ďalšieho spracovania (discard).
6. **Output Merger** – zodpovedá za finálne spracovanie dát, ktoré vznikli v priebehu pipeline (farba z fragment shaderu, hĺbkové informácie, prípadne stencil buffer) a zmiešava ich (blending) pre konečný výsledok celej pipeline.



Obrázok 2.1: Súčasti grafickej pipeline

Shader model 5.0 kompatibilná pipeline ešte navyše obsahuje **teselačnú jednotku.** Teselácia je proces, pri ktorej sa polygónová sieť vstupného objektu rozdeľuje na menšie trojuholníky, čím v konečnom výsledku umožňuje zjemniť povrch objektu, zaobliť hrany, prípadne pridať nové detaily pomocou výškovej textúry. Teseláciu ovládajú dva shadery – v DirectX 11 nazvané ako Hull shader a Domain shader, v OpenGL 4.0 je ich ekvivalentom Tessellation Control Shader a Tessellation Evaluation Shader [7].

1 V DirectX sa používa pojem pixel shader, v OpenGL zase fragment shader

2.1.1 Operácie zaťažujúce jednotlivé časti grafickej pipeline

Pre vývoj testovacej aplikácie sa ďalej budeme hlavne zaoberať programovateľnými časťami pipeline, teda vertex, geometry a fragment shadermi. Pre použiteľné výsledky merania je totiž potrebné získať index výkonu každej časti pipeline, ako potom aj všetkých častí. Z toho dôvodu je potrebné poznať operácie, ktoré zaťažujú len určitú časť pipeline a tie potom implementovať ako sériu testov.

1. **Vertex shader (VS)** – keďže spracúva vrcholy, najvyššiu záťaž bude predstavovať spracovávanie objektov s veľmi vysokým počtom polygónov a počítaním per-vertex efektov akými sú osvetľovací model, textúry používané ako výškové mapy, prípadne procedurálne generovanie terénu.
2. **Geometry shader (GS)** – ponúkajú sa testy, pri ktorých sa budú generovať nové vrcholy/primitíva. Nové vrcholy môžu vznikať pri teselácii povrchu, prípadne pri využití GS na vytváranie polygónov z jednotlivých bodov ako časticový systém. GS je možné využiť aj na inštancovanie objektov, keď je vykresľovaný len jeden objekt a GS generuje jeho kópie (inštancie). Ďalšou vlastnosťou GS je aj previazanosť s viac-vrstvovým framebuffer objektom, ktorá umožňuje vykresliť naraz obrazy primitíva do viacerých textúr. Toto sa dá využiť na dynamické odrazy počítané v jedinom vykresľovacom priechode (single pass).
3. **Fragment/pixel shader (FS,PS)** – zaťažujú ho operácie ako per-pixel osvetlenie, efekty na zvýšenie detailov povrchu (napr. normal mapping alebo parallax mapping). Ďalšou aplikáciou môže byť aplikácia kubickej textúry a pomocou nej počítané odrazy alebo refrakcia. Čistú výpočtovú silu FS ukazuje generovanie procedurálnych textúr s využitím šumových algoritmov.

Treba si ale uvedomiť, že vzhľadom k unifikovanej architektúre dnešných GPU, kde stream procesor vykonáva vertex, geometry aj fragment kód, takto vlastne meriame vždy výkonnosť všetkých stream procesorov pri vykonávaní rôzneho kódu (na rozdiel od predchádzajúcich generácií akcelerátorov, ktoré mali dedikované vertex a fragment shader jednotky).

Na zaťaženie celej pipeline sú vhodné komplexné algoritmy na realistické zobrazenie scény akými sú predovšetkým osvetlenie s vysokým dynamickým rozsahom (high dynamic range, HDR), dynamické tiene alebo efekt okolitého prostredia (ambient occlusion, AO). Tieto testy sú vhodné aj pre testovanie rýchlosti grafickej pamäte, pretože využívajú vykresľovanie (render) do textúry pomocou framebuffer objektov, čo predstavuje pomerne veľký dátový tok (extrémnym prípadom je tzv. supersampling, kde sa scéna prepočíta vo vyššom rozlíšení a zobrazí v nižšom, čo umožňuje solídne vyhladenie hrán aj u transparentných objektov a textúr).

2.1.2 Využitie GPU na všeobecné výpočty

Samostatnou kapitolou sú aplikácie, ktoré využívajú výpočtovú silu GPU na všeobecné výpočty (tzv. GPGPU výpočty[2]), pomocou rozhraní OpenCL a Direct Compute. Ako je spomínané v úvode sekcie 2.1, shader model 4.0 a vyšší je možné využiť na všeobecné výpočty. Tento obor začína byť veľmi perspektívny, pretože pomocou GPU so stovkami stream procesorov je možné niektoré aplikácie urýchliť rádovo až 100x. Využitie je najmä u prirodzene paralelných algoritmov, ktoré sa vyskytujú predovšetkým v spracovaní obrazu a videa. Veľký potenciál sa skrýva aj v akcelerácií vedeckých

2 www.gpgpu.org

simulácií, kde často spolu interagujú milióny častíc (ide o algoritmy s kvadratickou časovou zložitosťou $O(n^2)$), kde na klasických CPU by výpočet trval veľmi dlho. No a v neposlednom rade je tu využitie v zábavnom priemysle, kde podobné technológie môžu priniesť do hier kvalitnejší fyzikálny model a umelú inteligenciu.

Existuje teda dobrý dôvod pridať do testovacej aplikácie všeobecne výpočtové testy (napr, pod rozhraním OpenCL) a merať výkon, v tomto prípade teoretický (typicky sa určuje v miliardách floating point operácií za sekundu, GFLOPS/s).

2.2 Grafická pamäť a ostatné súčasti

Okrem samotnej grafickej pipeline, ktorá zahŕňa grafické jadro, treba zahrnúť aj vplyv rýchlosti a veľkosti grafickej pamäti (VRAM), do ktorej sa ukladajú všetky objekty, textúry, buffery a ostatné dáta potrebné pre vykresľovanie. Vplyv VRAM na výsledky testov je teda veľmi výrazný a treba s ním počítať. Úzkym hrdlom býva najčastejšie rýchlosť VRAM a šírka zbernice, ktorou komunikuje s GPU. V súčasnosti používané výkonné grafické karty používajú GDDR5 pamäti a najmenej 256-bit širokú zbernicu. Veľkosť pamäte je tiež dôležitá, hlavne pri vykresľovaní scény vo vyššom rozlíšení s použitím vyhladzovania hrán. Je však problematické vytvoriť testy čisto pre meranie výkonu grafickej VRAM, pretože sú zákonite zviazané s GPU. Vplyv VRAM ale bude najlepšie vidieť na zložitých scénach v množstvom efektov vo vysokom rozlíšení. Typickým reprezentantom efektu náročného na VRAM je osvetlenie s vysokým dynamickým rozsahom, kde sa využíva render do textúry v desatinnom formáte (floating point) a následný tzv. tonemapping na nižší rozsah (viacej v sekcii 4.5.1).

K ďalším dôležitým súčastiam grafickej karty patrí doska spojov (PCB), ktoré prepája grafický procesor s pamäťou a so systémovou zbernicou. Táto zbernica je dnes výhradne typu PCI-Express (rozhranie AGP sa už prestalo používať). Signál sa posiela z grafickej karty najčastejšie pomocou digitálnych rozhraní DVI alebo Display port, pričom niektoré karty umožňujú výstup aj na viacero monitorov (napr. technológia ATI Eyefinity umožňuje pripojiť až 3 monitory na jednu kartu[3]). Výhoda digitálneho prepojenia s monitorom je v tom, že odpadá potreba prevodníka na analógový signál, ako to bývalo u starších rozhraní VGA a D-SUB.

3 http://sites.amd.com/us/underground/products/eyefinity/Pages/eyefinity.aspx

3 Existujúce testové aplikácie

V súčasnosti existuje viacero aplikácií (benchmarkov), ktoré sa zameriavajú na meranie výkonu buď v herných, alebo profesionálnych aplikáciach. Rozsiahle aplikácie obsahujú viacero sérií testov, ktoré sú zamerané na určenie výkonu z každého hľadiska a ich výstupom býva čiastkové a výsledné skóre. Naproti tomu malé aplikácie sú zamerané na otestovanie jednej súčasti grafického akcelerátoru a nemajú tak rozsiahle výstupy. Často bývajú benchmarky implementované v hrách, prípadne ako technologické ukážky nových efektov. Ďalej nás budú zaujímať modernými benchmarky a hry, ktoré využívajú funkcie grafickej pipeline popísané v sekcii 2.1.

3.1 Rozsiahle testovacie aplikácie

Tieto aplikácie majú v sebe implementované množstvo testov a skúmajú výkon z viacerých hľadísk. Môžeme ich rozdeliť podľa viacerých kritérií, ale hlavné delenie bude na herné a profesionálne benchmarky. Ďalej je možné ich deliť podľa technológií, ktoré využívajú (napr. DirectX, OpenGL).

Najznámejšie herné/syntetické benchmarky (väčšinou od spoločnosti Futuremark, fungujúce pod rozhraním DirectX) sú:

- **3D Mark 2001** – ako prvý implementoval do meraní podporu pixel a vertex shaderov (shader model 1.0, rozhranie DirectX 8)
- **3D Mark 03, 05** – oba využívajú rozhranie DirectX 9 a shader model 2.0 na zobrazovanie pokročilých efektov. 3D Mark 05 navyše pridal podporu vyššieho jazyka HLSL
- **3D Mark 06** – predstavil použitie shader modelu 3.0 a DirectX 9.0c na pokročilých efektoch ako je HDR osvetlenie
- **3D Mark Vantage** – prvý shader model 4.0/DirectX 10 benchmark
- **GL Excess** – nezávislý OpenGL benchmark zameraný viac na teoretické meranie výkonnosti na menej rozsiahlych scénach.

K profesionálnym aplikáciám (fungujúcim väčšinou pod OpenGL) môžeme zaradiť:

- **SPEC Viewperf** – zisťuje výkonnosť pri zobrazovaní reálnych scén z CAD a iných 3D modelovacích programov
- **Cinebench 9,10,11** – zamerané opäť na výkon akcelerátoru pri zobrazovaní 3D scén v modelovacom programe Cinema 4D

Podrobne si rozoberieme dve aplikácie: 3D Mark Vantage ako herný DirectX benchmark a SPEC Viewperf ako profesionálny OpenGL benchmark.

3.1.1 3D Mark Vantage

Je dielom fínskej spoločnosti Futuremark a na trh bol uvedený v roku 2008. Primárne je zameraný na hráčskú komunitu, keďže testy pripomínajú scény z populárnych 3D hier. Využíva DirectX 10 rozhranie, pričom izoluje výsledky CPU a GPU do osobitných testov. Na výpočet fyzikálneho modelu používa GPU simulácie.

Z použitých technológií môžeme spomenúť HDR osvetlenie, pokročilé materiálové shadery v jazyku HLSL, dynamické tieňové mapy, post processing efekty ako žiara (bloom), odlesky šošovky (lens flare), hĺbka pohľadu, rozmazanie pohybu a generátory častíc na GPU (podrobnejší popis aplikácie a testov je možné nájsť v [8]).

Testy sú rozdelené do viacerých sekcií:

1. **Herné testy** – zaťažujú celú grafickú pipeline vykresľovaním a simuláciou zložitých herných scén. Preto majú aj najväčšiu váhu pri hodnotení.
2. **CPU testy** – merajú výkon CPU pri pokročilých výpočtoch fyzikálneho modelu a umelej inteligencie.
3. **Teoretické testy** – merajú výkonnostný dopad pri použití efektov zaťažujúcich určité časti pipeline. Sem patria fill rate testy, parallax occlusion mapping a Perlinov šum pre zaťaženie pixel shaderu. Fyzikálna simulácia látky a časticové systémy zase slúžia na zistenie výkonu vertex a geometry shaderu.

Testy je možné spúšťať s rôznymi nastaveniami a detailnosťou efektov. Výsledné skóre sa počíta z herných testov a je oddelené pre procesor a GPU. Výsledky je možné porovnávať s ostatnými užívateľmi v on-line databázi výsledkov.

3.1.2 SPEC Viewperf

Ide o sériu testov od neziskovej organizácie SPEC [9], ktorá si kladie za cieľ nezávislé porovnávanie výkonu v profesionálnych aplikáciách. Na vyhodnotenie výkonu používajú programové jadrá z aplikácií 3D Studio, Maya, Pro/Engineer, Solidworks a Catia. Testy používajú rozhranie OpenGL a dáta definované pomocou vertex buffer objektov. Základom sú testy s vysokým počtom detailných modelov, aplikované vyhladzovanie hrán a následné geometrické transformácie. Skóre sa určuje z rýchlosti zobrazovania v snímkoch za sekundu.

Testy sú orientované predovšetkým na profesionálne grafické karty, ktoré majú výrazne vyšší výkon v OpenGL ako herné karty (fyzicky síce ide o tie isté jadrá, ale profesionálne majú lepšiu optimalizáciu ovládačov pre OpenGL).

3.2 Ostatné testové aplikácie, herné demá

Tieto aplikácie sú väčšinou zamerané len na jednu oblasť výkonu, prípadne ako ukážka nových technológií. Uvedieme si niekoľko príkladov, spolu s odkazom na stránky produktu:

- **ShaderMark**[4] – zameriava sa na testovanie výkonu pixel shaderov pod rozhraním DirectX 9 s použitím jazyka HLSL. Umožňuje aplikovať rôzne shader efekty (normal mapping, environment mapping, anizotropné osvetlenie, HDR) a porovnávať kvalitu zobrazenia a výkon týchto efektov.

4 http://www.shadermark.com/start.html

- **FurMark**[5] - jednoduchá aplikácia zameraná primárne na maximálne zaťaženie grafického jadra a odhaľovanie artefaktov vznikajúcich pri nedostatočnom chladení. Intenzívne zaťažuje najmä fragment shader a grafickú pamäť.
- **CineBench 10/11**[6]**-** ide o benchmark postavený na profesionálnom modelovacom a renderovacom programe Cinema4D. Používa jadro programu, avšak len jeho zobrazovaciu časť. Podobne ako SPECviewperf pracuje pod rozhraním OpenGL. Umožňuje nastaviť viacero módov, od čisto softwarového vykresľovania až po plné OpenGL hardwarové vykresľovanie. Skóre sa počíta na body, ktoré vychádzajú zo zobrazených snímkov za sekundu

Herné demá a testovacie nástroje sú založené na reálnych hrách a často vychádzajú ako demonštrácia nových technológií. Inou možnosťou je použitie externých meracích programov a merať výkon priamo v hre. Každá hra využíva svoj tzv. **engine**, čo je vlastne jadro funkcií na vykresľovanie scény, efektov ale aj napr. fyzikálnych simulácií. Medzi príklady pokročilých enginov môžeme zahrnúť:

- **Unigine**[7] – pokročilý multiplatformný engine s podporou DirectX 11 a OpenGL. Implementuje väčšinu efektov používaných pri realistickom zobrazení (spomeňme hlavne HDR, dynamické tiene, ambient occlusion a od DirectX 11 aj teseláciu). Ako prvý DirectX 11 benchmark bolo v roku 2008 vypustené práve demo Unigine Heaven.
- **CryEngine 2 [10]** – vydaný prvýkrát v roku 2007 ako súčasť známej hry Crysis. Ako prvý implementoval pokročilé efekty ako ambient occlusion, dynamické mäkké tiene, rozšírenú fyziku a animáciu postáv, a to všetko v rozhraní DirectX 9/10. Nebola síce vydaná žiadna testovacia aplikácia postavená na tomto engine, ale je možné testovať priamo v hre pomocou skriptov.
- **id Tech 5**[8] – ešte vo vývoji, má tendenciu stať sa najpokročilejším OpenGL engine (jeho predchodca je id Tech 4, viac známy ako Doom 3 engine). Základom sú opäť pokročilé shader efekty ako HDR, tiene, motion blur a pod., rozšírené o technológiu MegaTexture, ktorá umožňuje používať textúry s veľmi veľkým rozlíšením a detailami.

Medzi ďalšie pokročilé enginy môžeme zahrnúť ešte napr. Unreal Engine 3, X-ray, Dunia a mnohé iné[9].

5 http://www.ozone3d.net/benchmarks/fur/
6 http://www.maxon.net/index.php?id=162&L=0
7 http://unigine.com/
8 http://en.wikipedia.org/wiki/Id_Tech_5
9 Celý zoznam na http://en.wikipedia.org/wiki/List_of_game_engines

4 Návrh a implementácia testov

Na úvod tejto kapitoly je vhodné si položiť otázku, aký je účel ďalšej testovej aplikácie, keďže existuje veľa podobných, komerčných aj nekomerčných testovacích aplikácií (prehľad v predchádzajúcej kapitole). Odpoveď je nasledovná, oproti ostatným prináša moja aplikácia tieto výhody:

- pracuje pod rozhraním OpenGL pre HW akceleráciu a SDL pre interakciu s operačným systémom. Obe tieto rozhranie sú **multiplatformné**, aplikácia je tak schopná behu na OS Windows aj Linux (na rozdiel od iných aplikácií, ktoré väčšinou pracujú pod rozhraním DirectX a teda fungujú len na OS Windows)
- implementuje pokročilé techniky pomocou najnovších funkcií z rozhrania OpenGL a zisťuje ich výkonnosť pod týmto rozhraním (aktuálne neexistuje ucelený OpenGL benchmark, ktorý by sa zameriaval na teoretické aj praktické funkcie najnovších akcelerátorov)
- k dispozícii celý zdrojový kód aplikácie
- on-line databáza výsledkov

Testy boli navrhnuté a implementované na základe poznatkov o grafickej pipeline. Zdrojom inšpirácie boli aj existujúce testové aplikácie (predovšetkým séria 3D Mark). Nebudeme sa ale zameriavať len na výkonnosť jednotlivých častí grafickej pipeline, ale aj aký je výkonový dosah pri aktivácii rozšírených algoritmov, ktoré sa dnes využívajú v moderných grafických aplikáciách. Preto v testoch, ktoré demonštrujú nejaký algoritmus, je pre ukážku rozdielu vo výkone ukázaná aj scéna bez tohto efektu. Táto časť testu sa ale nezapočítava do celkového hodnotenia, je tam čisto len na porovnanie výkonnostného dopadu.

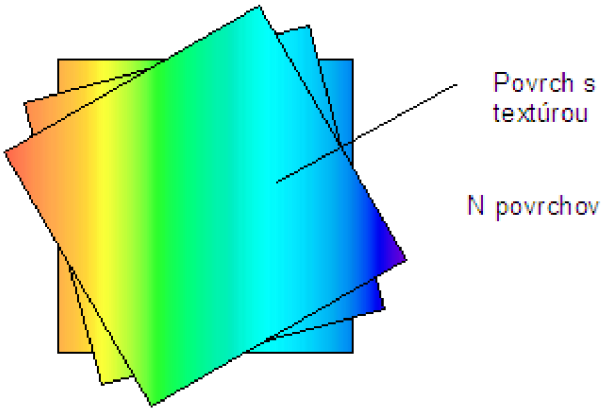
Keďže engine (podrobný popis v sekcii 5.1) je synteticky vytvorený (tzn. nie je v ňom implementovaná žiadna hra alebo iná praktická aplikácia), nemožno očakávať, že výsledky budú presne korešpondovať s inými testovými, alebo hernými aplikáciami. Bola ale snaha implementovať všetky algoritmy tak, ako sú používané v ostatných 3D aplikáciách.

Je implementovaných 6 sérií testov: teoretické **fillrate testy**, testy zamerané na **fragment, geometry a vertex shader**, ďalej **komplexné testy** s pokročilými efektami a nakoniec **výpočtový test** na určenie teoretického GPGPU výkonu pod rozhraním OpenCL. Testovaciu séria a rozlíšenie je možné nastaviť ako parameter aplikácie a po skončení testov sa výsledky uložia do XML súboru. Celkových počet testov je **21**. Tento počet je vyšší preto, lebo som chcel ukázať, aké sú výkonové rozdiely v implementácii najrozšírenejších postupov a efektov, ktoré sa využívajú v moderných 3D aplikáciach. Všetky testy budú podrobne popísané v ďalších kapitolách, názvy testov sú také, aké sa objavujú v aplikácii (v angličtine).

Na konci každej kapitoly o testovacej sérii prinesieme prehľadné zhrnutie série testov: účel série testov, akú časť grafickej karty najviac vyťažuje, percentuálnu váhu v rámci celkového skóre a čiastkové skóre na **referenčnej karte *nVidia GeForce GTX 285***, ktorá dosahuje celkové skóre 10 000 bodov v nastavení Mainstream (o výkonnostných nastaveniach a referenčnej karte bude pojednávané v sekcii 5.2.2).

4.1 Fill rate testy

Táto séria testov patrí medzi teoretické testy, ktoré zisťujú rýchlosť rasterizačnej jednotky grafického procesoru pri vyplňovaní obrazovky pixelmi (anglicky: to fill). Fill rate sa teoreticky dá vypočítať ako súčin taktovacej frekvencie grafického jadra s počtom textúrovacích jednotiek, v praxi sa však takých hodnôt nedosahuje (treba brať do úvahu aj čas transformácie vertexov, prípadne riešenie viditeľnosti pri finálnej kompozícii fragmentu).



Obrázok 4.1: Znázornenie povrchov využitých pri fillrate testoch

Testy prebiehajú tak, že na viacero štvorcových povrchov (konkrétne 64) je aplikovaná jedna alebo viac textúr. Textúry majú pomerne malé rozlíšenie (128x128 pixelov), aby sa minimalizoval vplyv rýchlosti VRAM (väčšia textúra vyžaduje viacej prístupov do pamäti). Povrchy sa prekrývajú a je použitý blending, čiže akcelerátor musí prepočítať každý pixel každého povrchu. Táto metóda má rovnaký základ ako v benchmarkoch 3D Mark 2001-06. Takéto nastavenie spresňuje meranie, pretože sa znižuje vplyv transformačnej jednotky. Hlavná záťaž sa takto dostáva na rasterizačnú jednotku a fragment shader.

Po skončení testov sa vypočíta rýchlosť vyplňovania v miliónoch texeloch za sekundu (MTexels/s) podľa nasledovného vzorca:

$$fillrate = \frac{(resolution_x . resolution_y . FPS)}{1000000} MTexels/s$$

Nasleduje popis jednotlivých testov, spolu s ich percentuálnymi váhami v rámci série.

4.1.1 Test 1 – single texturing

V tomto teste sa používa jedna textúra na jeden povrch. Účelom je zistiť výkon pri použití jedinej textúrovacej jednotky. Len jedna textúra na povrch sa v praxi príliš nevyužíva, väčšinou býva aplikovaných viac textúr (farebná textúra, normal textúra, svetelná mapa atď). Preto váha tohto testu v záverečnom hodnotení je najnižšia.

Váha testu: 15%

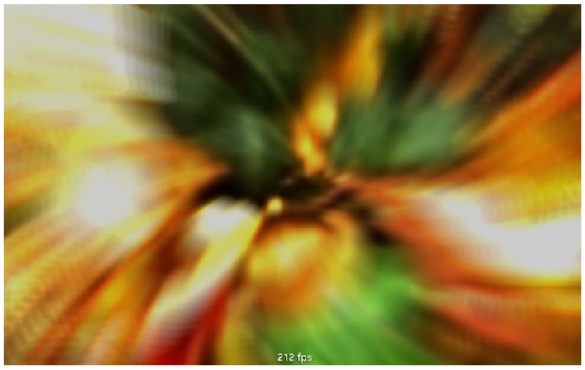
4.1.2 Test 2 – multi texturing

Oproti prvému testu sa nepoužíva jedna, ale všetky dostupné textúrovacie jednotky. U väčšiny OpenGL implementácií je tento počet rovný 8 (zisťuje sa príkazom `glGetIntegerv(GL_MAX_TEXTURE_UNITS, &tex_units)`. Výkon oproti prvému testu je potom násobne vyšší, lebo akcelerátor dokáže v jednom priechode naraz aplikovať viac textúr a nie je potrebné tak vykresľovať všetky povrchy (čo musí vykonať akcelerátor, ktorý ma menej textúrovacích jednotiek). Všetky textúrovacie jednotky sú aktivované vo fragment shaderi, ktorý aj vykonáva konečnú kompozíciu všetkých textúr (textúry sa zmiešavajú v aditívnom móde). Toto nastavenie odzrkadľuje dnešné 3D aplikácie, ktoré využívajú viacej textúr aplikovaných na jeden povrch. Váha tohto testu je preto vyššia ako u prvého.
Váha testu: 30%

4.1.3 Test 3 – 16-bit floating point textures

Nastavenie testu je podobné ako u predchádzajúceho testu. Podstatným rozdielom ale je, že v tomto prípade sa výsledok renderuje do 16-bit textúry s pohyblivou dátovou čiarkou (dátový formát GL_RGBA16F). Spolu tak máme až 64-bitový farebný rozsah na pixel (oproti 32-bit u klasického modelu RGBA). Pri renderi do textúry je použitý **framebuffer objekt (FBO**, OpenGL rozšírenie `GL_EXT_framebuffer_object`) s pripojenou render textúrou. Keďže renderom do floating point textúry získame hodnoty, ktoré sú mimo rozsah klasického 32-bit RGBA modelu, musí nasledovať **tonemapping,** aby sa výsledok dal zobraziť v rozsahu 0.0 – 1.0 (resp. 0 – 255 na jednu farebnú zložku pri RGB modeli) na monitore. Ukážka jednoduchého tonemappingu ako je implementovaný v shaderi ([14], sekcia 1.3):

```
float YD = exposure * (exposure/brightMax + 1.0) / (exposure + 1.0);
color *= YD;
```



Obrázok 4.2: Scéna z 2.testu používajúca 32-bit textúry s normálnym dynamickým rozsahom

Obrázok 4.3: Scéna z 3.testu používajúca render do 64-bit float textúry s vyšším dynamickým rozsahom

Na počiatku je v premennej `color` hodnota vo vysokom dynamickom rozsahu. Expozíciou je možné upraviť celkovú svietivosť výslednej kompozície. Premenná Y označuje jasovú hodnotu transformovaného pixelu, YD násobiaci faktor. Výsledkom je hodnota premennej `color` v rozsahu (0.0 – 1.0, alebo pri použití klasického typu unsigned byte pre RGBA je to 0 – 255). Výsledok sa aplikuje ako textúra na štvorec, ktorý pokrýva celú obrazovku.

Takto sa vlastne meria rýchlosť kopírovania do floating point textúry, čo je kritická výkonová operácia pri aplikáciách využívajúce HDR osvetlenie, ktoré dnes využíva prakticky každý pokročilý engine (viacej o HDR osvetlení v sekcii 4.5.1). Test závisí aj od rýchlosti VRAM, pretože sú prenášané pomerne veľké objemy dát (render do pripojenej textúry k framebufferu v každom snímku). Váha testu preto zodpovedá jeho dôležitosti.
Váha testu: 40%

4.1.4 Test 4 – 32-bit floating point textures

Podobne ako u predchádzajúceho testu, aj tu sa využíva floating point textúra a následný render scény do nej pomocou FBO. Rozdielom je použitie 32-bit textúry (formát GL_RGBA32F), čo síce prináša vyšší dynamický rozsah (až 128-bitov na pixel), ale významne znižuje výkonnosť. Rozdiel vo výslednom vzhľade scény totiž nie je markantný, a teda nestojí za znížený výkon zobrazenia. Z tohto dôvodu sú dnešné grafické procesory optimalizované predovšetkým na výpočty s 16-bit textúrami – ušetrí sa tým predovšetkým grafická pamäť (podrobnejšie o limitáciách v publikácii [14], sekcia 2.3.3).

Váha tohto testu z tohto dôvodu je nižšia ako u predchádzajúceho testu, avšak nie je vylúčené, že v budúcnosti sa bude tento formát používať (umožňuje vyššiu presnosť výpočtov po celej pipeline).
Váha testu: 15%

4.1.5 Zhrnutie série testov

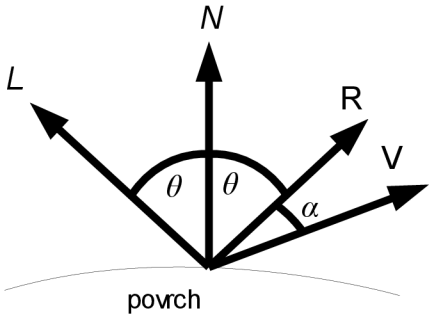
- **Účel:** zistiť teoretický výkon akcelerátoru pri aplikácii celočíselných aj floating-point textúr
- **Závisí od:** výkonu rasterizačných a textúrovacích jednotiek v GPU, rýchlosti VRAM
- **Váha v rámci celej aplikácie: 12%.** Aj keď je dôležitá vysoká rýchlosť aplikácie textúr, ide o čisto teoretické testy, ktoré nemajú príliš obdobu v reálnych aplikáciách. Preto nižší význam pri celkovom hodnotení
- **Skóre na referenčnej karte:** 1200 bodov.

4.2 Fragment shader testy

Fragment shader programy sa spúšťajú pre každý jeden pixel rasterizovaného primitíva, takže zo všetkých testových sérií sú najviac závislé na použitom rozlíšení obrazovky a úrovni vyhladzovanie hrán. Ako popísané v sekcii 2.1.1, fragment shadery najviac zaťažujú per-pixel operácie ako napr. Phongov svetelný model. Na rozdiel od predchádzajúcej série, kde mali testy spoločné rozhranie, tu je každý test úplne odlišný. Spolu bolo implementovaných 5 testov, ktoré merajú výkon v často používaných per-pixel operáciách akými sú osvetlenie, normal mapping, parallax mapping, generovanie procedurálnych textúr a odrazy pomocou kubických textúr.

4.2.1 Test 1 – per-pixel point lights

Test využíva **Phongov osvetľovací model** ([1], kap. 9.2) pre každé svetlo, pričom počet svetiel v scéne sa postupne zvyšuje z jedného až na osem. Z hľadiska zaťaženia fragment shaderu ide o to, že svetelný model sa počíta v cykle pre každé svetlo. V teste je využitý shader model 3.0 s dynamickým vetvením, kde premenná môže byť podmienkou cyklu. Nemusíme tak dopredu poznať počet svetiel a v priebehu vykresľovania ich môžeme dynamicky zapínať. Takto nám stačí jeden shader na počítanie svetelného modelu pre ľubovoľný počet svetiel. Nevýhodou je, že tento postup je nepoužiteľný pre shader model 2.0 (GPU založené na tomto shader modeli sú ale už zastaralé a takmer sa nevyužívajú) a aj na akcelerátoroch podporujúcich tento shader model je vykresľovanie pomalšie, než keby sa použil dopredu známy počet opakovaní (svetiel). Svetelný model sa počíta postupne, najprv sa určí difúzna zložka svetla pomocou Lambertovho modelu:



$$I_D = I_L . r_D . N . L$$

Obrázok 4.4: Phongov osvetľovací model

kde I_L je difúzna zložka svetla, r_D difúzna zložka materiálu, N vektor normály a L svetelný vektor. Odlesková zložka (specular) sa počíta nasledovne:

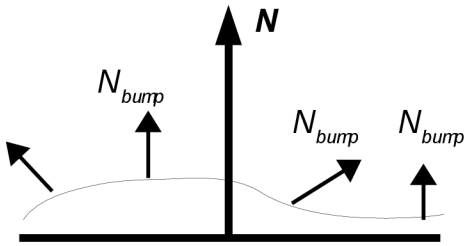
$$I_S = I_L . r_S . V . L^{n_S}$$

kde I_L je zložka odlesku svetla, r_S je zložka odlesku materiálu a n_S je sila odlesku (od 0 – 128). K takto získanému osvetleniu je ešte pridaný efekt zoslabovania intenzity svetla s rastúcou vzdialenosťou (attenuation). Tento efekt používa virtuálnu guľu okolo svetla, ktorá reprezentuje dosah svetla. Tento dosah sa počíta pomocou rovnice gule a s využitím vzdialenosti vrcholu od svetla [1]:

```
lightDir = (gl_LightSource[i].position.xyz + eyeVec + camPos)/
           (lightRadius);
att = max(0.0, 1.0 - dot(lightDir, lightDir));
```

kde `lightRadius` je dosah svetla, `camPos` aktuálna pozícia kamery a `eyeVec` pohľadový vektor svetla. Toto pridá na náročnosti výsledného testu, výsledkom je však pomerne realistické osvetlenie.

Pre zvýraznenie efektu per-pixel osvetlenia a ďalšie zvýšenie náročnosti testu je aplikovaný efekt tzv. **normal mapping** (tiež nazývaný aj bump mapping). Tento efekt modifikuje povrchovú normálu na základe načítanej textúry vo farebnom priestore normál a výsledkom je potom efekt nerovného povrchu. V jazyku GLSL je implementácia nasledovná:



Obrázok 4.5: Modifikácia povrchovej normály pri bump mappingu

```
bump_normal = normal +
texture2D(bumpTex,coord).xyz;
```

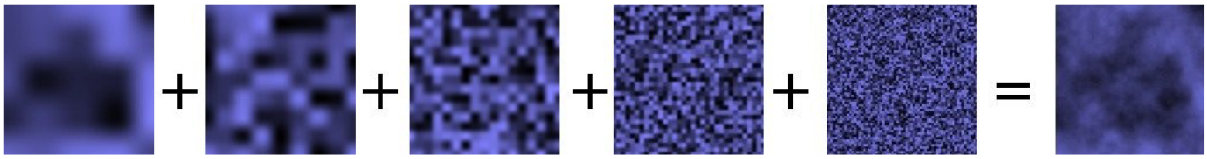
kde `normal` je pôvodná normála a `bumpTex` textúra obsahujúca normal/bump mapu.

Per-pixel osvetlenie je dnes dominantne používané v porovnaní s klasickým per-vertex osvetlením, pretože okrem vyššej kvality osvetlenia umožňuje pridávanie aj ďalších per-pixel efektov ako napr. normal mapping. Váha testu je preto dosť vysoká pri vytváraní skóre.
Váha testu: 30%

4.2.2 Test 2 – Perlin noise

Jednou z veľkých výhod fragment shaderu je možnosť vytvárať procedurálne textúry priamo za behu aplikácie. Výhoda je jasná – nie je treba žiadne externé úložisko pre textúru a neexistuje problém aliasu textúry, pretože sa dajú pixely dopočítať procedurálne (namiesto filtrovania u klasických textúr).

Väčšina generátorov procedurálnych textúr je založená na **Perlinovej šumovej funkcii** [15]. Základ funkcie spočíva v generovaní náhodných hodnôt a ich vzájomnou interpoláciou dostávame základný šum. Tento šum je možné násobiť s inými šumami o rôznej amplitúde, čím dostávame oktávy šumu. Výsledný šum je potom detailnejší:



Obrázok 4.6: Výsledok kombinácie šumov s rôznymi amplitúdami do jedného [15]

Najviac problematické sa ukázalo generovanie náhodných čísel v shaderi. Nakoniec je použitý tento pseudo-náhodný generátor, ktorý generuje 4 náhodné vektory:

```
vec4 rand(vec2 A,vec2 B,vec2 C,vec2 D){
    vec2 s = vec2 (12.9898,78.233);
    vec4 tmp = vec4( dot(A,s),dot(B,s),dot(C,s),dot(D,s));
    return fract(sin(tmp)  * 43758.5453);
}
```

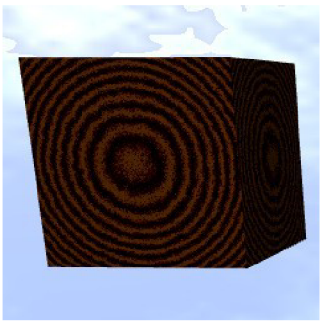
kde konštanty sú založené na prvočíslach. Takto vzniknuté náhodné hodnoty šumu je však treba pred použitím lineárne filtrovať, inak by bol výsledok príliš hrubý. Lineárny filter berie priemer z okolitých vzorkov, čím zvyšuje kvalitu, ale aj nároky na výkon. Nakoniec sa jednotlivé šumy kombinujú do jedného (Obrázok 4.6) podľa nastaveného počtu oktáv. Viacej oktáv = vyššie detaily, ale aj vyššie nároky na výkon.

Takto vytvorený generátor šumu sa používa na vytvorenie mrakov na oblohe. Vytvárajú sa dva druhy mrakov, každý s iným nastavením šumu, ktoré sa nakoniec zložia vo finálnej kompozícii. Testovacia aplikácia postupne prejde z 2 oktáv až na 16, čím výrazne vzrastajú nároky na fragment shader (u 16 oktáv treba pre každý fragment počítať až 64 vzorkov šumu). Šumové generátory sa používaju v mnohých aplikáciách kde je potrebný vysoký detail zobrazenia, prípadne na pridanie určitej náhodnosti povrchu, preto je váha testu pomerne významná.

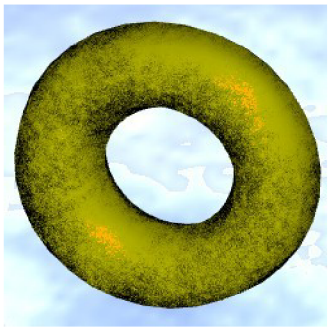
Váha testu: 20%

4.2.3 Test 3 – Procedural shaders

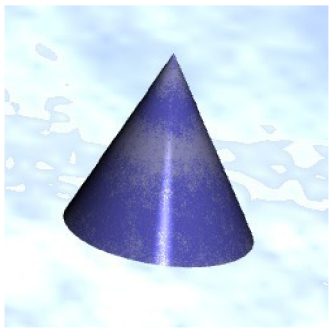
Účel tohto testu je zaťažiť fragment shader vytváraním procedurálnych materiálov. Pre ich vytváranie je vo väčšine prípadov použitá šumová funkcia popísaná v predchádzajúcej kapitole. Modifikovaním šumu matematickými funkciami ([1], kap. 11) je možné dosiahnuť zaujímavých materiálov:



Drevo – letokruhy sú dosiahnuté použitím funkcie sínus na výsledky šumu



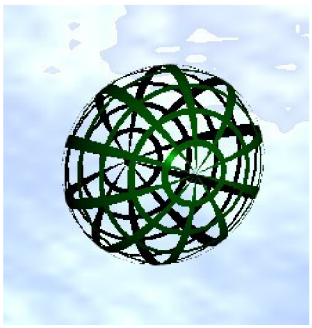
Animovaná bump-mapa – šum je animovaný pomocou externej uniformnej premennej a aplikovaný ako bump mapa



Mramor – šum v tomto prípade tvorí žilky minerálu. Šum je v tomto prípade trojrozmerný (upravená funkcia z minulej kapitoly)

Ďalšie shadery nevyužívajú šum, ale rozličné shader funkcie na dosiahnutie opakovaného vzorku:

Tehly – využívajú sa funkcie `fract()` a `step()` na periodické vykresľovanie čiar



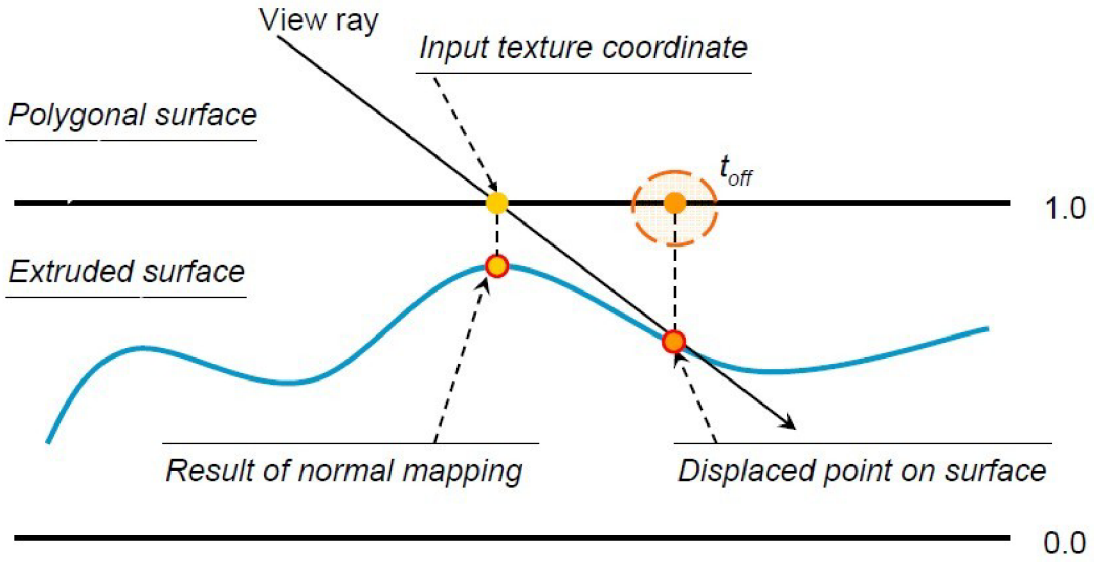
Mriežka – podobne ako u predchádzajúceho sa využíva funkcia `fract()`. V tomto shaderi je navyše podmienka, ktorá zahodí fragment ak sa nachádza uprostred mriežky

Výsledky tohto testu sú podobné ako u predchádzajúceho, pretože sa používa rovnaká šumová funkcia. Tá je ale u materiálu mramoru rozšírená do 3D, čim vzrastajú jej výkonové nároky. Z dôvodu použitia rovnakej šumovej funkcie bude váha testu o niečo nižšia ako u predchádzajúceho testu.

Váha testu: 10%

4.2.4 Test 4 – Parallax mapping

Parallax mapping je technológia, ktorá zlepšuje hĺbkový vnem scény s použitím výškovej textúry. Oproti klasickému normal mappingu totiž nielen modifikuje povrchovú normálu, ale podľa pohľadového vektoru aj posúva (offsetuje) textúrové koordináty. Takto vzniká výraznejší efekt nerovného povrchu. Existuje viacero prístupov k implementácii tohto efektu. Vizuálne najkorektnejší a výpočtovo najviac náročný je parallax occlusion mapping (POM) [16], ktorý využíva ray-casting na určenie korektného offsetu textúry z pohľadového vektoru a výškovej mapy. Algoritmus začína na pôvodnom textúrovom koordináte. Potom sa postupuje po výškovej mape a hľadá sa priesečník s pohľadovým vektorom a hĺbkovou mapou. Vzdialenosť tohto priesečníka od pôvodného koordinátu je hľadaný offset textúry:



Obrázok 4.7: Princíp parallax mappingu ([16], strana 16)

V praxi je možné tento algoritmus zjednodušiť a zrýchliť tak, že uvažujeme výškovú mapu ako hladký povrch, kde sa dva susedné texely od seba príliš nelíšia (majú rovnakú hĺbku). Potom môžeme offset počítať ako aproximáciu [16]:

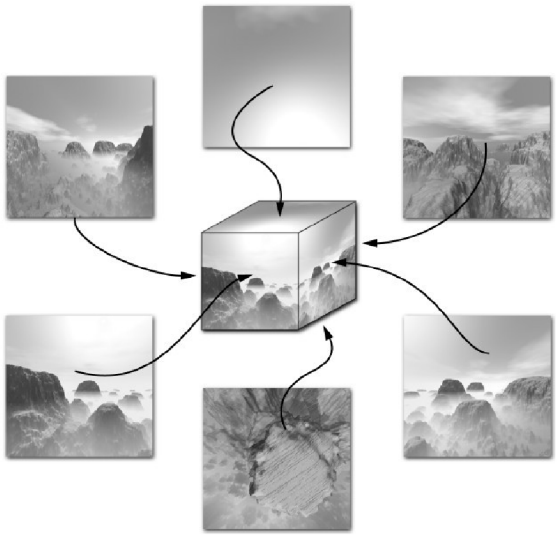
```
float height = texture2D(Bump, texCoord).a;
float offset = 0.035 * (2.0 * height - 1.0);
vec2 parallaxTexCoord = texCoord + offset * viewVec.xy;
```

Konštanta 0,035 reprezentuje krok, s ktorým sa vzorkuje výšková mapa a dobre vyzerá v aktuálnej scéne (pre iné scény a textúry je potrebné ju zmeniť). Tento postup je použitý aj v tomto teste, avšak nepodáva také dobré výsledky ako POM, ktorý umožňuje zahrnúť aj samozatieňovanie jednotlivých fragmentov.

Test začína s jednoduchým per-pixel osvetlením (ako popísané v kapitole 4.2.1), pričom postupne sa zapína normal a parallax mapping. Je teda možné pozorovať poklesy výkonu pri použití týchto technológií, ako aj zvyšovanie kvality povrchu. Tieto efekty výrazne vylepšujú vzhľad renderovaných povrchov, preto v celkovom hodnotení majú zodpovedajúcu dôležitosť.
Váha testu: 20%

4.2.5 Test 5 – Static reflections and refractions

Odrazy a refrakcia napomáhajú k zvyšovaniu reality scény, a pretože sa počítajú pre každý pixel mapovaného objektu, sú veľmi dobrým testom pre fragment shader. Pri mapovaní odrazov a refrakcie na objekt sa využívajú kubické textúry, ktoré v sebe obsahujú obraz okolia objektu. V tomto teste sa budeme zaoberať len statickými kubickými textúrami (tj. predpočítanými), avšak ako ukážeme v ďalších testoch (sekcia 4.3.2), dajú sa odrazy vytvárať aj dynamicky renderom okolia objektu do textúry.



Obrázok 4.8: Príklad aplikácie kubickej textúry [14]

Kubická textúra obsahuje 6 textúr, ktoré sa mapujú na steny kocky. Označujú sa potom podľa tej steny, na ktorú sú namapované: *positive X, negative X, positive Y, negative Y, positive Z, negative Z.* Okrem odrazov sa často používajú na reprezentáciu oblohy alebo iného prostredia v scéne, pretože na rozdiel od klasických sférických textúr netrpia distorziou a umožňujú vyššie detaily (za cenu vyššej spotreby grafickej pamäte). Veľkosť jednej časti kubickej textúry v tomto teste je 512x512 pixelov.

Pre výpočet odrazov potrebujeme poznať smer normály povrchu a pohľadový vektor vzhľadom k pozícii kamery. Potom vo fragment shaderi sa dá využiť zabudovaná funkcia `reflect()`, ktorá vypočíta vektor odrazený od pohľadového. Tento vektor sa potom použije ako 3D textúrový koordinát do kubickej textúry (tá na rozdiel od 2D textúr má aj priestorový koordinát, ktorý obsahuje stenu kocky). V jazyku GLSL to vyzerá nasledovne [14]:

```
vec3 reflVec = reflect(eyeVec, normal);
vec4 reflection = textureCube(tex, reflVec);
```

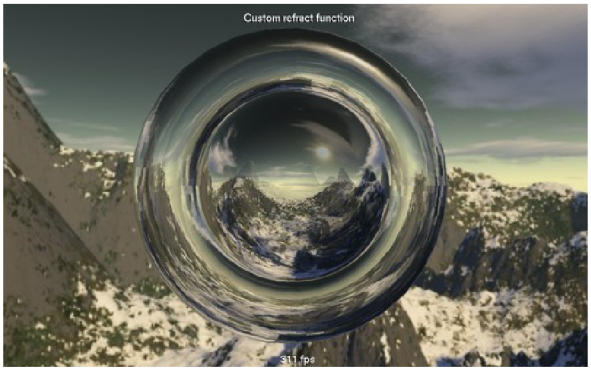
kde `eyeVec` je normalizovaný pohľadový vektor, `normal` je normála povrchu a `tex` je sampler pre kubickú textúru.

Refrakcia využíva podobne ako odraz pohľadový vektor a normálu, rozdiel je v počítaní vektoru pre kubickú textúru. Refrakcia láme príchodzí pohľadový vektor a ten používa pre kubickú textúru. Na to existuje zabudovaná funkcia `refract()`, avšak pre porovnanie je implementovaná aj funkcia, ktorá počíta refrakciu presne podľa Snellovho zákona lomu:

$$n_1 \,.\, sin(\Theta_I) = n_2 \,.\, sin\,(\Theta_T)$$

kde n_1,n_2 sú indexy lomu materiálov, Θ_I uhol dopadu a Θ_T je uhol lomu. V teste sa potom porovnáva výkon takto počítanej refrakčnej funkcie so zabudovanou funkciou z GLSL (zabudovaná by mala byť optimalizovaná a teda podávať lepší výkon).

Posledný efekt využitý v tomto teste je tzv. chromatická disperzia [14]. V praxi je totiž svetlo zložené z viacerých farebných zložiek a každá zložka sa láme inak ako ostatné. To môžeme pozorovať napr. na vodnej hladine, kde sa vytvárajú dúhové obrazce na rozhraní dvoch prostredí. Tento efekt môžeme aproximovať vypočítaním indexu lomu a refrakcie pre každú zložku farby zvlášť – v našom prípade pre R, G a B. Výsledok je potom mix farieb vzniknutých pri refrakcii jednotlivých zložiek:



Obrázok 4.10: Snímok z testu refrakcie

Obrázok 4.9: Efekt chromatickej disperzie

Testy prebiehajú postupne tak, že na začiatku sa zobrazuje odraz a refrakcia s použitím vlastnej refrakčnej funkcie. Potom sa použije zabudovaná a zisťuje sa rozdiel vo výkone. Neskôr sa aktivuje efekt chromatickej disperzie a opäť sa porovnáva výkon pri použití vlastnej a zabudovanej refrakčnej funkcie. Váha testu je porovnateľná s predchádzajúcim (tieto efekty sú v moderných 3D aplikáciach takisto pomerne dosť využívané).

Váha testu: 20%

4.2.6 Zhrnutie série testov

- **Účel:** zistiť výkon akcelerátoru pri aplikácii často používaných per-pixel efektov
- **Závisí od:** výkonu rasterizačnej jednotky a stream procesorov vykonávajúcich fragment shader kód
- **Váha v rámci celej aplikácie: 22%.** Fragment shader efekty majú významné postavenie a využitie pri realistickom zobrazovaní scény, takže výkon v per-pixel operáciách je dôležitý ukazovateľ aj pri celkovom výkone akcelerátoru.
- **Skóre na referenčnej karte:** 2200 bodov.

4.3 Geometry shader testy

Geometry shader (GS) sa v grafickej pipeline (kap. 2.1) nachádza za vertex shaderom a striktne ho vyžaduje. Na rozdiel od vertex shaderu dokáže generovať nové vrcholy priamo (počet vygenerovaných vrcholov na jedno primitívum je daný implementáciou, minimum podľa štandardu je 1024 [3]). Vstupom/výstupom shaderu sú základné primitíva ako bod, úsečka, trojuholník a pás trojuholníkov. Je možné tak napr. vygenerovať trojuholník z bodu.

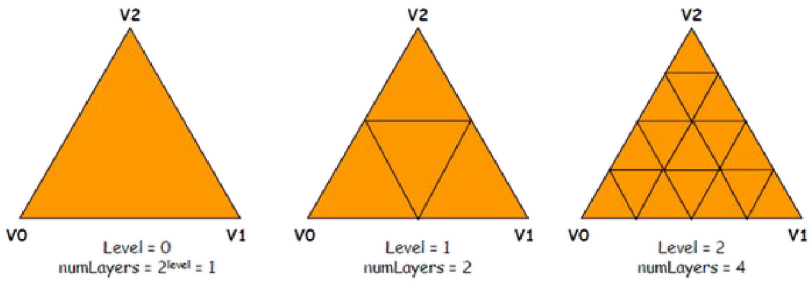
GS sa objavil s príchodom shader modelu 4.0 a DirectX 10. V OpenGL bol prístupný od verzie 2.1 ako rozšírenie, verzia 3.2 už ho obsahuje priamo vo svojej špecifikácii [3]. Podpora v ovládačoch prišla najprv pre akcelerátory značky nVidia (2006), neskôr AMD/ATI (až leto 2009). Je podporovaný od verzie 1.20 jazyka GLSL (v zdrojovom súbore preto treba túto verziu špecifikovať pre kompilátor). Syntax je rovnaká ako pri písaní iného shaderu, rozdiel je ale vo vstupných premenných, ktoré sú vždy v poli (pracujeme s primitívami, napr. trojuholník má 3 vrcholy, bude sa teda čítať z 3-prvkového poľa) [17].

Keďže podpora GS je v OpenGL pomerne nová, používa ich veľmi málo aplikácií. Preto táto testovacia séria patrí prakticky k jedinej možnosti, ako otestovať výkonnosť GS v rozhraní OpenGL (k začiatku roku 2010 nie je známy zatiaľ žiadny benchmark zameraný čisto na GS). Z tejto malej rozšírenosti aplikácií plynie aj problematická optimalizácia GS v ovládačoch, kde sa môžu vyskytovať chyby v zobrazení. Preto celkovej tejto sérii testov bude daná nižšia dôležitosť ako je to u predchádzajúcich (málo aplikácií, čo ich využívajú, podobná situácia je aj v rozhraní DirectX).

4.3.1 Test 1 – Tesselation with depth map

Teselácia je proces, pri ktorej sa objekt zložený z trojuholníkov rozdeľuje na nové trojuholníky, ktoré sú generované dynamicky z pôvodných. Umožňuje tak zvýšiť detailnosť objektov pri zachovaní pamäťových nárokov na uloženie 3D modelu. Často sa kombinuje s výškovou textúrou alebo normal mapou, kde sa pri teselácii zároveň mení pozícia nových vrcholov v závislosti na hodnote textúry. Významné použitie má teselácia aj pri zobrazovaní rozsiahleho terénu, kedy je možné dynamicky meniť úroveň detailov (LOD) v závislosti na vzdialenosti od kamery (blízko kamery bude vyšší stupeň teselácie ako ďalej od nej).

Základom tohto konkrétneho testu je rovina, zložená z 3000 trojuholníkov. Na ňu je aplikovaný geometry shader, ktorý na základe výškovej textúry mení pozíciu vertexov na ose Y (displacement mapping). Tento shader zároveň postupne vykonáva teseláciu povrchu, čím sa zvyšujú detaily generovaného terénu.



Obrázok 4.11: Postupná teselácia trojuholníku v parametrickom priestore [17]

Základná idea teselácie je v tomto prípade rozdeliť vstupný trojuholník na 4 menšie pomocou delenia v parametrickom priestore trojuholníku – súradnice vyjadrujeme vzhľadom k začiatku trojuholníku [17]. Pri teselačnej úrovni 4 tak dostávame z pôvodného trojuholníku 256 nových, čím postupne vzrastajú nároky na GS a tým aj detailnosť povrchu. Pre porovnanie je v teste zobrazený aj model terénu v drôtenom režime. Počas priebehu testu sa postupne zvyšuje úroveň teselácie až na úroveň 4. Pre čo najväčšiu záťaž sa v GS počíta aj svetelný model.

Počas priebehu testu sa na niektorých grafických kartách objavovali chyby v zobrazení, prípadne veľké výkonnostné straty. Tieto problémy (u kariet nVidia) čiastočne vyriešili aktualizované verzie ovládačov, napriek tomu má tento test nižšiu váhu v celkovom hodnotení. Treba aj spomenúť, že GS v shader modeli 4.0 nebol priamo určený na teseláciu povrchov, preto sa v DirectX 11 pipeline vyskytuje špecializovaná teselačná jednotka s výrazne vyšším výkonom ako GS.

Váha testu: 30%

4.3.2 Test 2 – Single pass dynamic cubemapping

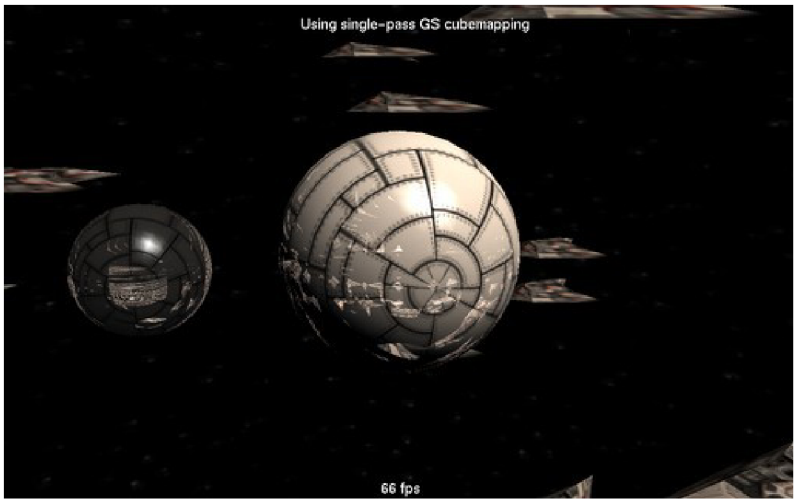
Geometry shader má zaujímavú vlastnosť, ktorou je možnosť renderovať primitívum naraz do šiestich vrstiev framebuffer objektu s aplikovanými textúrami. Dá sa tak vykresliť naraz 6 strán kubickej textúry (Obrázok 4.8) pre reflexiu v jednom priechode (oproti klasickej metóde ktorá spočíva v 6 - priechodovom renderi do textúry pre každú stranu kocky).

Vrstvený framebuffer je novinka v OpenGL 3.2, ktorá umožňuje naviazať kubickú textúru naraz pomocou funkcie `glFramebufferTexture()` (predtým sa museli pripájať farebné textúry po jednom) [3] . Stačí teda vytvoriť jeden FBO s jediným kubickým textúrovým objektom a v GS 6-krát vykresliť transformované primitívum do každej vrstvy (premenná `gl_Layer` určuje aktuálnu textúrovú vrstvu) :

```
for (layer = 0; layer < 6; layer++){
  gl_Layer = layer;                //nastavíme sa na aktuálnu vrstvu
  for (i = 0; i < 3; i++){
    //transformujeme a vykreslime vertex
    //(pre jednoduchosť transformácie neuvádzam)
    gl_Position = gl_ModelViewProjectionMatrix * gl_PositionIn[i];
    EmitVertex();
  }
  EndPrimitive();
}
```

Oproti klasickému postupu, ktorý využíva 6 textúr kubickej textúry naviazaný na framebuffer, tak odpadá nastavovanie transformačných matíc – scéna a všetky objekty v nej sa musia vykresliť 6-krát pri všetkých smeroch pohľadu, čo kladie veľké nároky na transformačnú jednotku a grafickú pamäť.

Nevýhodou vrstveného prístupu ale je absencia orezávania objektov, ktoré nie sú viditeľné, takže táto metóda nemusí vždy znamenať vyššiu výkonnosť zobrazenia. Na porovnanie sú v teste implementované obe metódy, pričom veľkosť reflexie je 512x512 pixelov pre každú stranu kubickej textúry.



Obrázok 4.12: Chyby v zobrazení pri použití GS a vrstveného framebufferu (karta *ATI Radeon HD 4650*)

Žiaľ, podpora tejto metódy v ovládačoch je veľmi zlá - na kartách *GeForce* nefungovala vôbec (problémy s vrstveným FBO), u kariet *Radeon* bol pozorovaný 30%-ný nárast výkonu pri použití GS, avšak za cenu chýb v zobrazení (Obrázok 4.12). Keďže s novšími verziami ovládačov sa tieto problémy nevyriešili (u kariet ATI dokonca zhoršili), tento test je vypustený z celkového hodnotenia.
Váha testu: 0%

4.3.3 Test 3 – GPU particles

Keďže geometry shader dokáže vytvárať polygóny z jednotlivých bodov, ponúka sa možnosť ich využiť ako generátor častíc implementovaný výhradne na GPU. V aplikácii sa len nastaví pozícia emitoru častíc ako bod, pričom shader generuje nové častice a stará sa o ich vykresľovanie.

Implementovaný je veľmi jednoduchý časticový systém, ktorý generuje častice padajúce smerom nadol. Keď častica dosiahne zem, zaniká a vytvorí sa nová na pozícii emitoru. Pozícia aj rýchlosť každej častice sa náhodne generuje. Do scény preto stačí pridať pár emitorov a je možné dosiahnuť bohatého časticového efektu (v tomto prípade efekt dažďa). Keďže primitíva sú generované, dosahuje sa úspora pamäte oproti iným metódam, navyše je možné vytvárať plnohodnotné 3D objekty namiesto 2D spritov.

Počas priebehu testu sa zvyšuje počet častíc (a tým aj počet vrcholov, ktoré musí GS generovať) postupne z 6400 až na 22 500 častíc. Z tohto počtu častíc a snímkovej frekvencie sa potom počíta a zobrazuje počet vygenerovaných častíc za sekundu.

Tento test ukazuje najčastejšie využitie geometry shaderu ako generátora častíc, preto má z celej série najvyššiu váhu. Ďalším dôvodom najvyššej váhy je skutočnosť, že na rozdiel od predchádzajúcich techník je podpora tohto efektu v ovládačoch vcelku bezproblémová a boli zaznamenané len malé prepady výkonu a chyby v zobrazení.
Váha testu: 50%

4.3.4 Test 4 – Geometry shader instancing

Geometry instancing je technológia, pri ktorej sa priamo vykresľuje iba základný objekt, z ktorého sa potom generujú kópie (inštancie), ktoré nie sú uložené v pamäti. To znamená, že sa generujú dynamicky a výkon zobrazenia by mal byť tým pádom vyšší. Ďalšou prednosťou je aj úspora grafickej pamäte. Treba ale podotknúť, že GS nie je priamo určený na generovanie duplikátov objektu (na to sú prispôsobené špeciálne funkcie API, ktoré sú popísané v sekcii 4.4.1), preto tento test využíva nízkopolygónový model (pri vyššom počte polygónov sa výkonový prínos stráca). Pre porovnanie tento test ukazuje aj klasické vykresľovanie na bázi vertex buffer objektov. U geometry shader instancingu sa kreslia len 4 objekty, pričom zvyšok sa generuje (duplikuje) v shaderi a náhodne rozmiestňuje v priestore.

GS použitý v tomto teste je pomerne jednoduchý, keďže len duplikuje príchodzie primitívum a nastavuje mu novú (náhodnú) pozíciu. Náhodný generátor funguje podobne ako popísané v kapitole 4.2.2. Pracuje teda v cykle s pevným počtom opakovaní a v každom cykle pošle na výstup kópiu vstupného primitíva. Takýto instancing v praxi nie je príliš použiteľný, pretože OpenGL má vlastné funkcie, ktoré umožňujú rýchle generovať instancie bez použitia geometry shaderu (príkaz `glDrawElementsInstanced()` nad vertex buffer objektami [3]). Na testovací účel ako záťažový test pre geometry shader je však možné túto techniku použiť. Opäť sa ale vyskytovali určité problémy s ovládačmi, keď niekedy dochádzalo k drastickým prepadom výkonu pri použití GS instancingu. Ďalším dôvodom zníženej váhy testu je aj skutočnosť, že na takéto účely nie je veľmi HW implementácia GS navrhnutá (podobne ako u testu teselácie v sekcii 4.3.1).
Váha testu: 20 %

4.3.5 Zhrnutie série testov

- **Účel:** zistiť výkon akcelerátoru pri generovaní nových vrcholov a operácií s takýmito vrcholmi
- **Závisí od:** výkonu stream procesorov vykonávajúcich geometry shader kód a od výkonu transformačnej jednotky
- **Váha v rámci celej aplikácie: 10%.** Váha je takto nízka z dvoch dôvodov: prvým je, že GS efekty nie sú v praxi veľmi používané (ani v DirectX aplikáciách), druhým je problematická podpora zložitejších efektov ako napr. teselácia či vrstvený framebuffer v ovládačoch, kde sa vyskytujú chyby v zobrazení a prepady výkonu.
- **Skóre na referenčnej karte:** 1000 bodov.

4.4 Vertex shader testy

Vertex shader, ako už napovedá názov, sa používa na prácu s jednotlivými vrcholmi primitíva. Často používanými efektami je per-vertex osvetlenie, ktoré (ak má model málo vrcholov) kladie na GPU menšie nároky ako výpočet per-pixel osvetlenia. Ďalej je možné ho využiť na efekty ako napr. procedurálne generovanie terénu alebo aplikácia výškových textúr za účelom zvýšenia detailnosti povrchu. Zároveň je možné tak určiť teoretický výkon vo vykresľovaní polygónových objektov, preto sú súčasťou tejto testovej série aj merania rýchlosti vykresľovania v miliónoch vrcholov za sekundu (MVertices/s), ktoré sa počítajú nasledovne:

$$vertexRate = \frac{počet\ vrcholov\ scény\ .\ snímkov\ za\ sekundu}{1000000}\ \text{MVertices/s}$$

Sú implementované 3 série testov, zamerané na typické per-vertex operácie: osvetľovací model, procedurálne generovanie povrchu a aplikáciu výškovej textúry modifikujúcej povrch.

4.4.1 Test 1 – Per-vertex lighting with geometry instancing

Tento test používa Phongov osvetľovací model (tak ako bol popísaný v kapitole 4.2.1). Na rozdiel od fragment shaderu, model sa počíta len pre každý vrchol primitíva – u nízkopolygónových objektov má preto takéto osvetlenie vyšší výkon (za cenu nižšej kvality osvetlenia, ktoré je interpolované po celom polygóne):

Obrázok 4.13: Rozdiel medzi per-pixel (vľavo) a per-vertex osvetlením

Základom scény je model auta, ktorý obsahuje 56-tisíc vrcholov. Okolo neho sú rozmiestnené objekty, tvoriace mriežku. Pri týchto kópiách mriežky je využitá technológia **geometry instancing (GI)**. Na rozdiel od GI v sekcii 4.3.4, kde inštancie generoval geometry shader a spôsobovalo to problémy s výkonom u zložitejších objektov, tu ide o HW akcelerovanú funkciu z OpenGL 3.2 API. GI funguje tak, že z referenčného objektu znovu používa maximum informácií – polohy jednotlivých vertexov, ich vertexové atribúty a čiastkové výpočty u napr. svetelného modelu. Výhoda je aj v tom, že na zobrazenie všetkých objektov sa použije jediný shader a jediná transformačná matica, čiže odpadajú operácie akými sú prepínanie shader programov a 3D transformácie. Tým sa znižuje závislosť výkonu od CPU (ktorý nemusí vykonávať výpočet novej matice pre každý objekt) a je možné vykresľovať mnoho objektov jediným príkazom `glDrawElementsInstanced()`. Rozmiestnenie v priestore sa deje v shaderi pomocou nasledujúceho kódu:

```
vec4 vertex = gl_Vertex;
//zmena pozície pomocou ID inštancie
vertex.x += 5*(gl_InstanceID%INSTANCES);
vertex.z += 5*(gl_InstanceID/INSTANCES);
```

kde `gl_InstanceID` je premenná posielaná z API pre každé primitívum a označuje jeho ID. Premenná `INSTANCES` je počet generovaných inštancií. Treba podotknúť, že staršie karty túto funkciu nepodporujú a v tom prípade sú objekty vykresľované menej efektívnou metódou (rozmiestnením v priestore pomocou transformačných matíc). Bez ohľadu na použitú metódu, vykresľovaných je viac ako 1,2 milióna trojuholníkov v každom snímku, takže už len samotné vykreslenie bez osvetlenia kladie vysoké nároky na transformačnú jednotku akcelerátoru a vertex shader.

V priebehu testovania je postupne zapínaných viacero svetiel (podobne ako v 4.2.1, aj tu je využitý shader model 3.0 a dynamické vetvenie) a meria sa teoretický výkon v MVertices/s. Keďže náročné scény v 3D aplikáciách častokrát obsahujú veľmi detailné modely, je dôležitý výkon akcelerátoru pri vykresľovaní takýchto modelov a celých scén. Aj technológia GI je pomerne široko využívaná pri opakujúcich sa podobných objektoch (napr. tráva, stromy, skaly...). Preto má tento test najvyššiu váhu z celej série vertex shader testov.

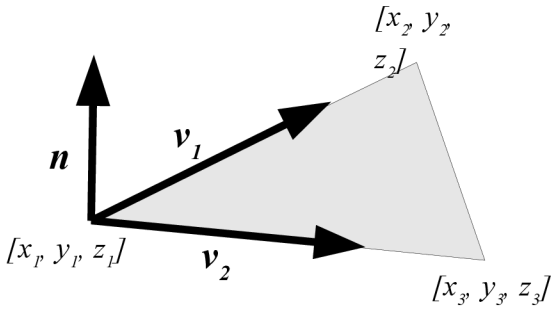
Váha testu: 45%

4.4.2 Test 2 – GPU generated waves

V tomto teste sa využíva vertex shader ako procedurálny generátor povrchu, tu konkrétne morskej hladiny. Efekt spočíva v dynamickej modifikácii Y-súradnice každého vertexu pomocou periodických funkcií sínus a kosínus. Základom je nosná vlna s veľkou amplitúdou a konštantným smerom, ktorá sa postupne spočítava s menšími vlnami s rozličným smerom a inou rýchlosťou. V shaderi to vyzerá nasledovne:

```
vertex.y = 3.0*sin(vertex.x/16.0 + time)
         + sin(vertex.x/8.0 + 2.0*time) + sin(vertex.z/10.0 + time)
         - 3.5*perlinNoise2D(vertex.xz/512.0,2.0,4.0);
```

Funkcia `perlinNoise2D` vnáša do generátoru náhodnosť, ktorá vyplýva z Perlinovho šumu (ako popísané v sekcii 4.2.2). Takto sa modifikujú vrcholy, no treba navyše modifikovať aj normálu pre správne osvetlenie povrchu. To sa robí výpočtom vlnovej funkcie pre ďalšie dva susedné vrcholy, z ktorých sa určia smerové vektory k pôvodnému vrcholu. Vektor kolmý na tieto dva vektory je potom hľadaná normála. Kolmý vektor sa dá jednoducho vypočítať pomocou vektorového súčinu (v GLSL je na to zabudovaná funkcia `cross()`).



$$\vec{v}_1 \in [x_2 - x_1, y_2 - y_1, z_2 - z_1]$$
$$\vec{v}_2 \in [x_3 - x_1, y_3 - y_1, z_3 - z_1]$$
$$\vec{n} = \vec{v}_1 \times \vec{v}_2$$

Obrázok 4.14: Výpočet normály z aktuálneho vrcholu pomocou dvoch susedných vrcholov

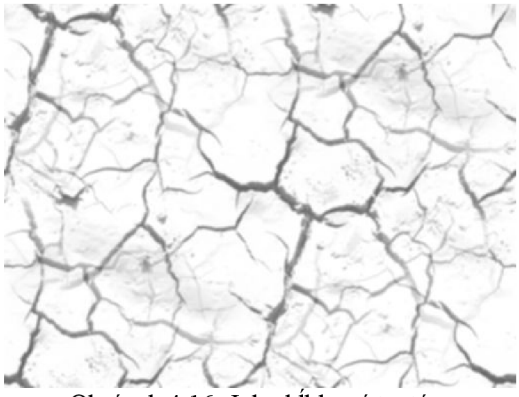
Takto je zaručené korektné osvetlenie modifikovaného povrchu pomocou Phongovho modelu. Pre zvýšenie náročnosti testu sú tu ešte aplikované efekty ako odrazy pomocou kubickej textúry (podobne ako v teste 4.2.5) a efekt hmly. Vertex shader je potom aplikovaný na rovinnú polygónovú sieť tvorenú 250-tisíc trojuholníkmi.

Tento test ukazuje možnosti procedurálneho generátora povrchu, čo sa často využíva aj v moderných 3D aplikáciách (hlavne na predvedenom efekte vlnenia vodnej hladiny). Oproti minulému testu ale bude váha nižšia, pretože per-vertex osvetlenie sa využíva vo väčšej miere ako procedurálne povrchy.
Váha testu: 25%

4.4.3 Test 3 – Vertex displacement mapping

Vo vertex shaderi, podobne ako vo fragment aj geometry shaderi, je možné použiť 2D aj 3D textúry z externých zdrojov. Kým ale vo fragment shaderi sú používané hlavne pre ich farebné informácie, vo vertex shaderi je dôležitá ich hĺbková informácia. Vo všeobecnosti stačí mať textúru v škále sivej, kde potom čím viac je bod tmavší, tým má väčšiu hĺbku. Takto sa dajú vytvárať pomerne zložité povrchy (najčastejšie terény) s použitím nejakého základného primitíva (napr. roviny, gule) a hĺbkovej textúry. Tento proces je známy ako **displacement mapping**, pretože mení pozíciu (displacing) aktuálneho vrcholu podľa hodnoty texelu v hĺbkovej textúre. Korektná orientácia normály je zaručená použitím prídavnej normal mapy (na rozdiel od procedurálneho terénu, kde bolo nutné normálu dopočítať). Na nasledujúcich obrázkoch vidíme aplikáciu hĺbkovej textúry na polygónovú sieť:

Obrázok 4.15: Terén s aplikovaným displace efektom



Obrázok 4.16: Jeho hĺbková textúra

Test patrí k náročnejším, pretože na celej scéne sa nachádza vyše 650-tisíc trojuholníkov s aplikovanou displace textúrou a per-vertex osvetlením. Navyše sa demonštruje rozdiel medzi zapnutým a vypnutým displace efektom, čo umožňuje zistiť prípadné výkonové straty pri použití tohto efektu. Čo sa týka váhy, tento efekt nie je až tak masívne používaný, často sa však používa v spolupráci s procedurálnym generátorom na vytváranie realistických vodných hladín, prípadne celých terénov.
Váha testu: 30%

4.4.4 Zhrnutie série testov

- **Účel:** zistiť výkon akcelerátoru pri operáciách s vrcholmi
- **Závisí od:** výkonu stream procesorov vykonávajúcich vertex shader kód a od výkonu transformačnej jednotky
- **Váha v rámci celej aplikácie: 17%.** Výkonnosť vertex shaderu je veľmi dôležitá, pretože rozsiahle scény často pozostávajú z rádovo miliónov trojuholníkov, ktoré treba transformovať a aplikovať rôzne per-vertex efekty ako napr. osvetlenie. Keďže ale nemajú až také masívne využitie ako per-pixel efekty, ich váha bude trochu nižšia.
- **Skóre na referenčnej karte:** 1700 bodov.

4.5 Komplexné testy

Ak predchádzajúce série testov boli zamerané na jednotlivé časti programovateľnej grafickej pipeline, táto séria sa zameriava komplexne na celú grafickú pipeline a zaťažuje celé GPU spolu s VRAM. Toto je dosiahnuté použitím efektov, ktoré sa používajú v realistickom zobrazení v reálnom čase. Sú to najmä HDR osvetlenie, efekt okolitého prostredia a dynamické tiene. Testy sú postavené na podstatne zložitejších scénach ako predchádzajúce testy s mnoho trojuholníkmi (záťaž na vertex shader a transformačnú jednotku) s pokročilými per-pixel efektami (záťaž pre fragment shader) a renderom častí scény do textúry (napr. pri tieňoch, záťaž pre VRAM). Táto testová séria má vďaka zameraniu sa na celú grafickú kartu najvyššiu váhu v celkovom skóre.

4.5.1 Test 1 – HDR lighting

HDR (High Dynamic Range) osvetlenie je v súčasnosti veľmi používaný efekt, ktorý umožňuje lepšie vnímanie reality skrz vyšší dynamický rozsah, než aký je používaný u non-HDR aplikácií. Na rozdiel od klasického osvetľovacieho modelu, ktorý pracuje s RGB farebným modelom a 256 možnými úrovňami (typicky dátový typ unsigned byte, čiže 24 bitov na pixel), HDR rendering pracuje s floating-point hodnotami na pixel, čo umožňuje dosiahnuť oveľa vyšší dynamický rozsah (až 1:65 536 pri použití 16-bit float hodnôt). Nastáva problém, ako tento rozsah previesť do formátu zobraziteľnom na bežných zariadeniach ako sú monitory – na to existuje tonemapping, ktorý prevádza vysoký dynamický rozsah na nízky [14].

Obrázok 4.17: Scéna s vypnutým HDR

Obrázok 4.18: Scéna so zapnutým HDR

Pre real-time zobrazenie HDR osvetlenia v OpenGL je použitý nasledujúci postup [14]:

1) render scény do floating-point textúry (typicky stačí 16-bit float, vyššia presnosť je výkonovo veľmi náročná). Rozlíšenie render textúry je rovnaké, ako rozlíšenie obrazovky.
2) na takto získanú textúru aplikujeme shader, ktorý extrahuje oblasti s vyšším jasom (nastaviteľný prah).
3) ďalší shader vykoná rozmazanie textúry získanej v postupu 2). Použije sa Gaussov filter v dvoch krokoch – horizontálne a vertikálne. Pre urýchlenie a zlepšenie kvality rozmazania sa textúra podvzorkuje na polovičné rozlíšenie, rozmaže sa a potom sa roztiahne späť do plnej veľkosti.
4) finálna kompozícia textúr získaných v krokoch 1) a 3), pričom na prevod do nižšieho rozsahu je v shaderi implementovaný tone-mapping.

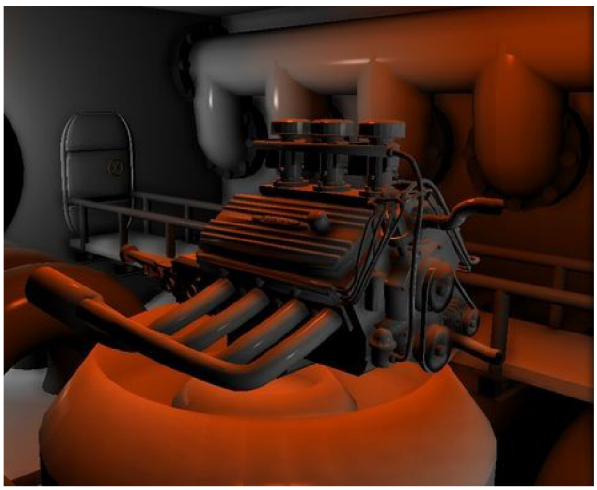
Ak je zapnuté vyhladzovanie hrán, namiesto klasického framebuffer objektu sa použije framebuffer s podporou vyhladzovania hrán (multisampled FBO, prístupný cez OpenGL rozšírenie `GL_EXT_framebuffer_multisample`). Až z neho sú potom dáta kopírované do obyčajného FBO (pomocou funkcie `glBlitFramebuffer()`), čím sa tento proces stáva veľmi náročným na výkon grafickej pamäte.

Test začína s vypnutým HDR osvetlením, ktoré sa v polovici priebehu testu zapne, aby bol vidieť výkonnostný prepad pri použití tohto efektu. Prvá časť (HDR vypnuté) sa do celkového hodnotenia nezarátava. Keďže HDR osvetlenie je veľmi využívaná technika, má tento test veľkú váhu v celkovom hodnotení.

Váha testu: 30%

4.5.2 Test 2 – Screen space ambient occlusion

Tento efekt je aproximáciou globálneho osvetlenia využívaného pri radiačných metódach. Idea je taká, že objekty nachádzajúce sa blízko seba sa navzájom zatieňujú (occluders) – typický príklad je roh v miestnosti, ktorý zatieňujú ostatné steny, z čoho vyplýva nižšia intenzita osvetlenia. Algoritmus, používaný pri offline aplikáciach ako napr. raytracing, spočíva vo vysielaní lúčov z aktuálneho bodu a určovanie priesečníkov s ostatnými objektami. Čím viac a čím bližšie sa tieto priesečníky nachádzajú, tým bude znížená intenzita osvetlenia. Avšak tento postup je nepoužiteľný v aplikáciach, ktoré využívajú rasterizáciu – nemáme možnosť skúmať interakcie medzi objektami. Preto bola vyvinutá metóda nazvaná Screen Space Ambient Occlusion (SSAO, prvýkrát využitá v CryEngine 2 [10]), ktorá pracuje v obrazovom priestore a na rekonštrukciu vzájomných interakcií medzi objektami využíva Z-buffer a normálový buffer. Nie je taká presná ako úplné AO, ale pre real-time aplikácie je dostatočná.



Obrázok 4.19: Scéna s ambient occlusion efektom

Obrázok 4.20: Samotný výstum SSAO algoritmu

Postup SSAO algoritmu je nasledovný [18]:

1) využijeme render naraz (s použitím multiple render targets, MRT) do dvoch textúr – jedna obsahuje v RGB kanáli farebný obraz scény, v alfa kanáli hĺbkové hodnoty z-bufferu, druhá obsahuje veľkosť a orientáciu normál ako farebný obrázok
2) z použitých dvoch textúr máme dosť informácií na výpočet SSAO, kde pre každý pixel:
    1. vysielame náhodné vzorky do okolitých pixelov v závislosti na nastavenom dosahu efektu. Na náhodné vektory je použitá textúra s náhodnými hodnotami

2. zisťujeme rozdiel medzi vzorkami – čím je nižší, tým viac je aktuálny pixel zatienený vybraným náhodným vzorkom. Normály sú použité na odstránenie aliasingu, ktorý by vznikal pri plynulých prechodoch v hĺbke (napr. rovná stena otočená voči kamere), kde by vznikali čierne pruhy.
3. výpočet plynulého prechodu hodnoty od aktuálneho pixelu k vybranému (napr. pomocou $1/x^2$ úbytku)

3) takto získanú textúru s ambient occlusion hodnotami opäť rozmažeme (pre urýchlenie je všetko počítané v polovičnom rozlíšení)
4) finálna kompozícia vyrenderovanej scény v 1) s AO termom získanom v 3) (Obrázok 4.20)

Táto metóda je pomerne náročná na grafickú pamäť, lebo vyžaduje veľa prístupov k textúre (napr. pri AO počítané v 8 vzorkoch s 3x3 rozmazaním ide až o 32 prístupov na každý pixel), čiže okrem rýchleho GPU vyžaduje aj rýchlu grafickú pamäť (ako však uvidíme v sekcii 6.1, tieto teoretické predpoklady nemusia vždy platiť). Existuje ale viacero metód na vylepšenie tohto efektu ([2], kap. 12), ktoré dávajú vyššiu kvalitu výstupu a menšie nároky na výpočtový výkon. Keďže efekt okolitého prostredia sa v moderných 3D aplikáciách používa pomerne výrazne, jeho váha je podobná ako u HDR osvetlenia.
Váha testu: 30%

4.5.3 Test 3 – Dynamic soft shadows

Test sa zaoberá rýchlosťou zobrazovania mäkkých, dynamických tieňov za využitia tieňových textúr (máp). V súčasnosti ide o najpoužívanejšiu metódu zobrazenia dynamických tieňov. Výhodou je nezávislosť na komplexnosti geometrie scény, avšak za cenu aliasingu a vyššej spotrebe pamäte. Metóda spočíva v renderi scény do hĺbkovej textúry z pohľadu svetla a následnej projekcii tejto textúry do pohľadového priestoru a porovnávaním hĺbkovej hodnoty v textúre s hĺbkovou hodnotou aktuálne vykresľovaného pixelu. Tiene je možné vylepšiť pomocou fragment shaderu pridaním rozmazania okrajov, čím sa potlači aliasing.

Popis algoritmu [19]:

1) render tieňovej mapy ako textúry, ktorá obsahuje hĺbkové hodnoty a renderuje sa z pohľadu svetla (Obrázok 4.22). Veľkosť tohto pohľadu (a teda aj textúry) je 4096x4096 pixelov.
2) transformácia a projekcia tejto textúry na všetky objekty v scéne.
3) pre každý pixel prebieha porovnanie – ak má pixel scény menšiu hĺbkovú hodnotu ako texel tieňovej mapy, nachádza sa v tieni a vykreslí sa s menším osvetlením (alebo sa prekryje tmavou textúrou ako v našom prípade). Toto porovnanie prebieha kompletne v HW grafickej karty (rozšírenie `GL_ARB_shadow`).
4) pri tomto porovnaní zároveň prebieha v shaderi rozmazanie okrajov tieňa s použitím jednoduchého priemerovacieho filtra (PCF – Percentage closer filtering).

Obrázok 4.21: Scéna s aplikovanou tieňovou mapou

Obrázok 4.22: Vizualizácia použitej tieňovej mapy (render scény z pohľadu svetla)

Výsledkom sú pomerne realistické tiene, ktoré je možné ešte vylepšiť rôznymi technikami (napr. perspektívne mapovanie tieňových máp, podrobný popis v [2], kap. 8 a 10). Pri použití veľkej tieňovej mapy (viac ako 2048x2048 pixelov) vzrastajú nároky na výpočtový výkon GPU a najmä VRAM (používa sa render do textúry). Keďže táto technológia sa používa takmer v každej aplikácii zaoberajúcej sa realistickým 3D zobrazením, výsledky tohto testu majú vysokú dôležitosť v celkovom hodnotení. Opäť test ukazuje rozdiel medzi zapnutými a vypnutými tieňmi, čo umožňuje určiť výkonovú stratu pri zapnutí tohto efektu.

Váha testu: 30%.

4.5.4 Test 4 – Render with supersampling

Tento test využíva technológiu zvanú supersampling, ktorá sa používala (a v menšej miere stále používa) na vyhladzovanie hrán objektov a textúr, avšak v ďalšom vývoji sa od nej upustilo kvôli jej vysokým výkonovým nárokom, pretože vyžaduje výkonnú grafickú kartu s rýchlou VRAM. To z nej robí ideálny test na výkon predovšetkým VRAM.

Kľúčom tejto metódy je render scény do textúry vo vysokom rozlíšení, väčšinou dvojnásobnom ako je rozlíšenie obrazovky. Čiže napr. pri rozlíšení 1024x768 sa scéne interne prepočíta v rozlíšení 2048x1536. Textúra s vyrenderovanou scénou vo vysokom rozlíšení sa potom prevzorkuje na rozlíšenie obrazovky. Pretože sa pri vzorkovaní použije lineárny filter, výsledkom je dokonalé vyhladenie hrán nielen 3D objektov, ale aj transparentných textúr (napr. plot, listy apod.), ako vidieť na obrázkoch 4.23 a 4.24.

Je zrejmé, že na VRAM sa kladú v tomto prípade vysoké nároky (vyžaduje sa 4x väčší pamäťový priestor). Napríklad pri rozlíšení obrazovky 1920x1080 má framebuffer veľkosť 8MB, ale pri supersamplingu je použitý framebuffer s textúrou o rozlíšení 3840x2160, čo dáva veľkosť 32MB. Ak chceme dosiahnuť s touto metódou interaktívne zobrazenie v aspoň 60 snímkov za sekundu, je potrebný dátový tok takmer 2GB/s na každú sekundu (a to ešte nie je započítaný čas a priestor zmeny veľkosti textúry).

Obrázok 4.24: Pôvodná scéna, všimnite si aliasu hlavne pri transparentných textúrach (tráva, listy)

Obrázok 4.23: Scéna so supersamplingom, výsledkom je dokonalé vyhladenie hrán aj transparentných textúr

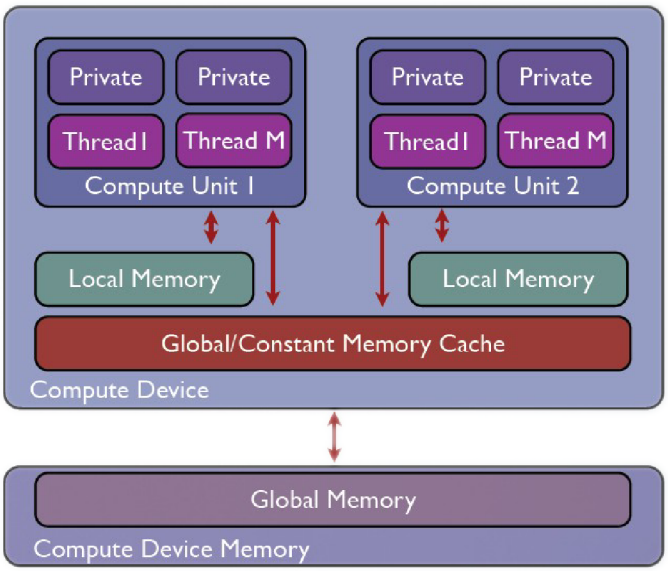
Keďže dnes sa technológia supersamplingu pomerne málo používa (len u naozaj výkonných kariet s výkonovou rezervou na takéto vylepšenie obrazu), je zaradená len ako náročný pamäťový test a jeho váha je výrazne nižšia ako u predchádzajúcich testov (ktorých technológie sa reálne používajú).
Váha testu: 10%

4.5.5 Zhrnutie série testov

- **Účel:** zistiť celkový výkon akcelerátoru pri zobrazovaní náročných komplexných efektov
- **Závisí od:** výkonu celého GPU a VRAM
- **Váha v rámci celej aplikácie: 27%.** Tieto testy by mali najlepšie odzrkadľovať nároky moderných 3D aplikácií, kde sú tieto efekty hojne využívané. Preto najvyššia percentuálna váha zo všetkých sérií testov.
- **Skóre na referenčnej karte:** 2700 bodov.

4.6 Výpočtové testy

S príchodom DirectX 10 generácie grafických kariet, ktoré už nemali dedikované fragment a vertex shader jednotky ale všeobecné stream procesory, je už možné takéto GPU využiť na všeobecné výpočty. Výhodou GPU architektúry oproti klasickým CPU je masívna paralelnosť: moderné GPU obsahujú stovky stream procesorov, ktoré sa dajú považovať za výpočtové jadrá. Je logické, že táto výhoda sa ukáže v paralelných výpočtoch – u sériových je takáto architektúra veľmi pomalá a neefektívna.



Obrázok 4.25: Schéma výpočtových jednotiek v GPU
([20], Episode 2)

Ako vidíme na obrázku, programovací model vychádza z architektúry GPU [20], kde stream procesor spracúva jedno výpočtové vlákno (thread), procesory sú organizované do blokov (compute unit), ktoré používajú malú (menej ako 32kB), ale veľmi rýchlu zdieľanú pamäť (local memory) a nakoniec, tieto bloky spolu zdieľajú pomalšiu, no rozsiahlu (až 1GB) globálnu pamäť (global memory).

Treba podotknúť, že nie všetky GPU plno podporujú takúto architektúru, napríklad akcelerátory *ATI Radeon* série *4000* nemajú podporu lokálnej pamäti (je emulovaná cez globálnu), čo v konečnom dôsledku vedie k zníženému výkonu oproti akcelerátorom, ktoré plne podporujú túto výpočtovú architektúru.

Existuje viacero rozhraní na výpočty pomocou GPU, medzi najpoužívanejšie patria:

- **CUDA:** štandard vytvorený firmou nVidia (funguje teda len na jej akcelerátoroch)
- **OpenCL:** otvorený štandard udržiavaný konzorciom Khronos group (stoja aj za štandardom OpenGL)
- **Direct Compute:** súčasť rozhrania DirectX 11 (umožňuje ale výpočty aj na DirectX 10 HW), spravované firmou Microsoft

Rozhodol som sa v teste využiť rozhranie OpenCL, ktoré ako otvorený štandard funguje na všetkých akcelerátoroch (na rozdiel od CUDA) a operačných systémoch (na rozdiel od Direct Compute).

4.6.1 Test 1 – N-Body simulation

N-Body simulácia sa často používa pri demonštráciách paralelného výpočtového výkonu. Ide o zaujímavý algoritmus s časovou náročnosťou $O(N^2)$, ktorý beží pomaly na sériovom HW ako napr. jednojadrové CPU, ale (keďže sa dá ľahko paralelizovať) veľmi rýchlo na paralelnom HW, akými sú grafické procesory. Z tohto dôvodu pramení jeho časté užitie pri demonštrácii vysokého výpočtového výkonu GPU nad CPU (rádovo 100x vyšší).

Základom sú tisíce častíc (v našom teste 4096), kde každá častica je v interakcii s ostatnými cez vzájomné gravitačné pôsobenie. Toto pôsobenie medzi dvomi časticami s indexmi i,j je možné vyjadriť z rovnice gravitačného potenciálu:

$$f_{ij} = G\frac{m_i m_j}{\|r_{ij}\|^2}\cdot\frac{r_{ij}}{\|r_{ij}\|}$$

kde m_i, m_j sú hmotnosti častíc, r_{ij} je vektor vzdialenosti medzi časticami a G je gravitačná konštanta. Z tejto rovnice môžeme vyjadriť celkovú silu, ktorou na časticu i pôsobia všetky ostatné častice:

$$F_i = \sum_{1\le j\le N} f_{ij} = Gm_i \cdot \sum_{1\le j\le N} \frac{m_j r_{ij}}{\|r_{ij}\|^3},\ j \neq i$$

Výpočet môžeme paralelizovať tak, že každá výpočtová jednotka bude mať na starosti práve jednu časticu. Na urýchlenie výpočtu je možné využiť lokálnu pamäť – rozdelíme všetky častice do blokov, kde v každom bloku budeme zdieľať pozície častíc z bloku. Tým klesne počet prístupov do globálnej pamäte a zvýši sa výkon. Podrobnejší popis tohto algoritmu a jeho optimalizácií je možné nájsť v publikácii [2], kap. 31.

Ako spomínané na začiatku sekcie 4.6, je použité rozhranie OpenCL na výpočet. Oproti ostatným testom je teda navyše potrebné:

1) získať podporovanú platformu (GPU) a inicializovať OpenCL zariadenie (device)
2) pripojiť skompilovaný zdrojový kód kernelu (výpočtovej jednotky)
3) alokovať buffery na zápis a čítanie dát na zariadení
4) nastaviť veľkosť dát (4096 častíc) a veľkosť výpočtového bloku (v OpenCL nazvaný ako work-group), typická veľkosť je 256 vlákien (work-items)
5) pred každým vykreslením scény vypočítať novú polohu častíc na základe ich vzájomnej interakcie.

Doba tohto výpočtu sa zaznamenáva (pomocou OpenCL profilovania), aby skóre nebolo ovplyvnené rýchlosťou renderu, pretože každá častica sa renderuje ako 3D objekt, čo vedie k nezanedbateľnému spomaleniu.

Výstupom testu je okrem priemerného počtu snímkov (tie nás teraz príliš nezaujímajú) priemerná doba výpočtu. To spolu so znalosťou, koľko operácií s pohyblivou desatinnou čiarkou (FLOPS) zaberie jedno spustenie kernelu, nám umožňuje určiť hrubý výpočtový výkon v miliardách operácií s pohyblivou desatinnou čiarkou (GFLOPS/s). Konkrétne, u N-Body simulácie si jedna interakcia vyžiada 21 FLOPS. Z tohto výkonu sa potom počíta aj celkové skóre testu. Keďže v tejto sérii je len jeden test, má plnú váhu 100%.

Zdrojový kód kernelu v tomto teste (`data/kernels/nbody.cl`) pochádza z príkladov rozhrania ATI Stream (ATI Stream SDK[1]).

4.6.2 Zhrnutie série testov

- **Účel:** zistiť teoretický výkon GPU pri všeobecných paralelných výpočtoch
- **Závisí od:** výkonu celého GPU aj VRAM
- **Váha v rámci celej aplikácie: 12%.** Keďže ide o teoretický test, váha je zodpovedajúco nižšia. Druhým dôvodom je nie úplne bezproblémová implementácia OpenCL rozhrania v HW grafických kariet, kde niektoré typy (ATI) sú znevýhodnené pomalou lokálnou pamäťou
- **Skóre na referenčnej karte:** 1200 bodov

1 http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx

5 Testová aplikácia

Všetky testové série popísané v predchádzajúcej kapitole sú implementované v testovej aplikácii. Tá sa skladá z dvoch častí:

- **hlavná aplikácia (bin/bench.exe)**, ktorá beží v príkazovom riadku a vykonáva všetky testy. Má grafický výstup pomocou rozhrania SDL, môže bežať v režime celej obrazovky (predvolené) ako aj v okne. Ak nie je v systéme nainštalované rozhranie OpenCL, použije sa verzia aplikácie bez výpočtových testov (bin/bench_noCL.exe).
- **grafické rozhranie (gluxMark2.exe)**, ktoré umožňuje nastavovať parametry testu, spúšťa testovací program, ukazuje informácie o systéme a prezentuje výsledky v grafickej podobe. Na implementáciu grafického rozhrania je využitá knižnica **wxWidgets**[10].

Na renderovanie scén a efektov aplikácia používa vlastný engine, ktorý si stručne popíšeme.

5.1 Použitý engine a technológie

Grafický engine („**gluxEngine**“) je napísaný v jazyku **C++** s využitím objektového programovania a modulov. Podrobný popis týchto modulov je možné nájsť na CD s aplikáciou (vygenerovaný pomocou programu **Doxygen**). Engine využíva rozhrania **OpenGL** [3] a **SDL**[11], z čoho vyplýva multiplatformnosť riešenia, a funkčnosť aplikácie na širokom spektre operačných systémov. Na pohodlnú prácu s OpenGL rozšíreniami je použitá knižnica OpenGL Extension Wrangler (**GLEW**)[12].

Engine je založený na rozhraní OpenGL 3.2 (kompatibilný mód, aby bol schopný behu aj na starších akcelerátoroch bez plnej podpory OpenGL 3.2), ktoré podporuje mnoho pokročilých funkcií už priamo v svojej špecifikácii (jedná sa hlavne o geometry shadery). Jadrom celého systému je generátor materiálov (funkcia `BakeMaterial()` v triede `TMaterial`), ktorý dynamicky generuje shadery v jazyku **GLSL** (verzia 150) [4] podľa nastavených vlastností. Je tak možné nastaviť vlastnosti povrchu akými sú farba, odlesk, základné textúry, bump a parallax mapping, environment a cube mapping, ako aj aplikácia tieňových máp. Ak sa vyžaduje pokročilejší materiál, je možné načítať a použiť vlastný súbor so zdrojovým kódom shaderu.

Celá scéna (trieda `TScene`) obsahuje zoznamy objektov, materiálov a svetiel. Každý objekt (`TObject`) má definované základné vlastnosti ako pozícia, veľkosť, vrhanie tieňa a materiál. Materiál sa opäť skladá zo základných vlastností ako farba, priehľadnosť a aplikované textúry. Po definícii materiálu sa dynamicky generuje shader, ktorý ho reprezentuje. Svetlá (`TLight`) použité v scéne majú okrem základných vlastností ako pozícia a farby možnosť tiež vrhať tieň (každému svetlu, ktoré vrhá tieň, sa pripraví tieňová textúra a framebuffer objekt na render tieňa). Engine ako celok umožňuje aj pokročilé efekty akými sú dynamické tiene, HDR osvetlenie, ambient occlusion a dynamické odrazy pomocou cube maps. Súčasťou je aj napojenie enginu na rozhranie **OpenCL** (`TCompute`), pomocou ktorého je možné vykonávať všeobecné výpočty a simulácie pomocou GPU.

10 http://www.wxwidgets.org/
11 http://www.libsdl.org/
12 http://glew.sourceforge.net/

Podrobnejší popis použitých technológií je v kapitole o jednotlivých testoch (4). Engine je stále vo vývoji, takže postupne bude obohacovaný o nové funkcie – chýba mu napr. graf scény a pokročilá animácia kamery a objektov.

5.1.1 Práca s objektami a textúrami

Keďže chceme testy vzhľadom priblížiť čo najviac k reálnym scénam, nevyhneme sa aplikácií externých objektov a textúr. Takisto sú potrebné výpisy textu na obrazovku, čo vyžaduje podporu metódy vykresľovania znakov na obrazovku.

- **Objekty:** je možné využiť základné primitíva ako rovina, kocka, guľa, kužeľ a pod, ktoré sú generované priamo v kóde, prípadne načítavať externé súbory vo formáte 3DS. Tento formát je pomerne rozšírený a keďže je binárny, úsporne ukladá 3D dáta. Na čítanie objektov aj celých scén v tomto formáte je použitá open-source knižnica **lib3ds** [11], ktorá uloží informácie zo súboru do prehľadných dátových štruktúr. S týmito štruktúrami potom pracuje funkcia `TObject::Create()`, pre ktorú sú podstatné tieto štruktúry: zoznam všetkých objektov (a k nim zodpovedajúcich materiálov), zoznam materiálov a textúr, ako aj nastavenia scény (kamera, svetlá). 3D objekty sú reprezentované zoznamom vrcholov, textúrových koordinátov a plôch. Tieto dáta sa dajú priamo použiť do vertex buffer objektov (VBO) pre rýchle zobrazovanie. Objekt sa potom registruje do zoznamu objektov v scéne a aplikuje sa naň vopred vytvorený materiál. Nie je implementovaný žiadny graf scény ani iná hierarchická štruktúra objektov.

  Väčšina objektov bola vytvorených autorom pomocou 3D modelovacieho programu Cinema4D CE 6 (ktorý je dostupný zdarma), ostatné sú voľne šíriteľné modely z on-line knižnice TurboSquid[13] (ide o modely v testoch popísaných v sekciách 4.4.3 a 4.5)
- **Textúry:** do aplikácie sa načítavajú z externých TGA súborov [12]. TGA ako binárny formát s priamo uloženými dátami v 24-bit RGB formáte sa vyznačuje vysokou rýchlosťou načítania, avšak za cenu vyššej priestorovej náročnosti. Z textúr sa potom automaticky generujú MIP-mapy (pomocou funkcie z knižnice GLU, `gluBuild2DMipmaps()`) a je možné zvoliť lineárne alebo anizotropné filtrovanie (cez OpenGL rozšírenie `GL_EXT_texture_filter_anisotropic`). Všetky tieto metódy implementuje trieda `Texture`.

  Zdroje textúr sú rôzne, ale vždy sa jedná o obrázky, ktoré sú voľne prístupné a bez autorských práv. Niektoré kubické textúry sú použité z [14].
- **Fonty:** v aplikácii sú použité fonty ako textúra, ktorá sa aplikuje na polygóny. Jeden polygón sa potom použije na vykreslenie jedného znaku. Výhodou tohto postupu je absolútna nezávislosť na systémových fontoch a vyššia rýchlosť vykresľovania ako u bitmapových fontov. Použité sú display listy, kde každý znak má priradený polygón a časť textúry, ktorá obsahuje daný znak (potrebujeme teda 256 display listov na každý znak z tabuľky ASCII. Postup, vytvorenie a kreslenie fontu je podrobnejšie popísaný v [13] a v aplikácii ho zabezpečujú funkcie `TScene::BuildFont()` a `TScene::DrawScreenText()`.

13 http://www.turbosquid.com

14 http://www.codemonsters.de/home/content.php?show=cubemaps

5.2 Rozhranie aplikácie, vstupy a výstupy

Na prácu s aplikáciou a nastavovanie parametrov testu je pripravené grafické rozhranie, pomocou ktorého je možné nastaviť a spustiť testovanie. Aplikácia ďalej poskytuje základné aj rozšírené informácie o systéme, zamerané najmä na vlastnosti a schopnosti grafickej karty v rozhraniach OpenGL a OpenCL. Po prebehnutí testov poskytuje informácie o čiastkových aj celkovom skóre ako aj možnosť vygenerovať HTML stránku s detailnými výsledkami.

S aplikáciou je možné pracovať aj v príkazovom riadku, pričom je parametrami možné nastaviť rozlíšenie a úroveň vyhladzovania hrán. Podrobný popis parametrov a ovládania aplikácie je v Prílohe 1.

5.2.1 Získavanie informácií o systéme

Aby sme vedeli porovnávať výsledky medzi rôznymi systémami, je nevyhnutná základná detekcia HW konfigurácie počítača. Na toto slúži testová aplikácia, ktorá pri spustení s parametrom `info` zavolá funkciu `Test::SystemInfo()`, ktorá zistí informácie o systéme a uloží ich do súboru *sysinfo.xml*. Tieto informácie o grafickej karte a podporovaných OpenGL/OpenCL funkciách sa získavajú priamo, pomocou parametrov posielaných funkciám `glGetString a glGetIntegerv` (hodnoty parametrov je možné nájsť v [3], strana 378) a sú spoločné pre všetky operačné systémy. Informácie o procesore a operačnej pamäti sa získavajú podľa toho, v akom OS bola aplikácia spustená:

- **Windows:** informácie sa zisťujú z registrov. Názov a taktovaciu frekvenciu procesoru nájdeme vo vetve `HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0`. Na prečítanie hodnoty z registra používame štandardné WinAPI funkcie `RegOpenKeyEx` a `RegQueryValueEx`. Informácie o operačnej pamäti je možné zistiť z funkcie `GlobalMemoryStatus(&mem)`.
- **Linux:** všetky informácie o systéme sa nachádzajú v adresári `/proc`. Pri detekcii procesoru stačí otvoriť `/proc/cpuinfo` a vyhľadať reťazce, ktoré nás zaujímajú – v našom prípade názov procesoru a jeho skutočný takt. Detekcia operačnej pamäte prebieha podobne, údaje čítame z `/proc/meminfo`.

5.2.2 Výstupy

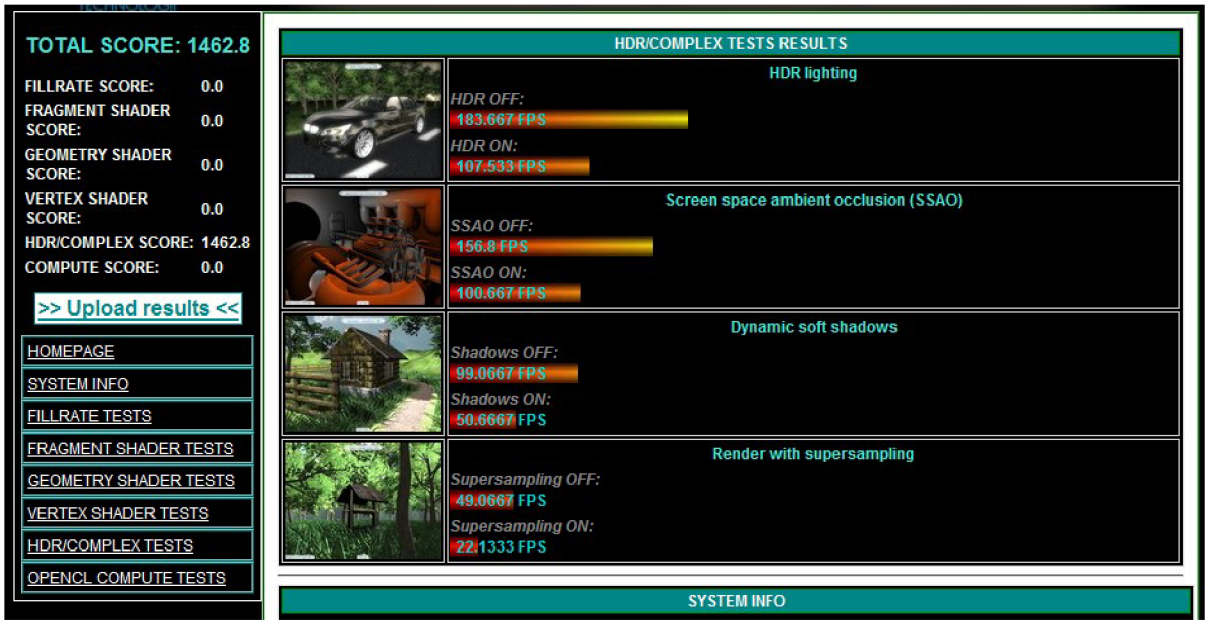
Výstupom aplikácie je XML súbor *results.xml* s informáciami o systéme a po prebehnutí všetkých testov aj informácie o výkone v jednotlivých testoch. Generuje sa postupne popri priebehu testov (každá séria volá funkciu `Test::SaveResults()`), do súboru sa zapíše po ukončení testov. Tento súbor obsahuje niekoľko hlavných sekcií:

```
<root> //koreňový uzol XML dokumentu
      <sysinfo>
            <system> //informácie o systéme </system>
            <gfx> //informácie o grafickej karte </gfx>
      </sysinfo>
```

```
<settings> //nastavenia benchmarku
</settings>
//nasledujú výsledky testov, formát je spoločný:
<test_series>
      <testX>
            <fps></fps>
            <score></score>
      </testX>
</test_series>
//posledná sekcia dokumentu, výsledné skóre a kontrolný súčet
<total_score>0</total_score>
<check>0</check>
</root>
```

Ako vidíme, XML formát predstavuje jednoduchú a prehľadnú formu na uloženie všetkých výsledkov. Každá séria testov obsahuje svoje skóre, z ktorého sa nakoniec počíta čiastkové a celkové skóre. Hodnota celkového skóre sa odvíja od **referenčnej grafickej karty**. Ako takúto kartu som použil ***nVidia GeForce GTX 285*** (výkonný reprezentant DirectX 10 a OpenGL 3.2 generácie akcelerátorov), ktorá získa **10 000 bodov** v nastavení „**Mainstream**“ (podrobnosti o nastavení v ďalšej sekcii). Aby sa zamedzilo podvádzaniu pri nahrávaní výsledkov, skóre je zabezpečené kontrolnou sumou. Táto suma sa potom počíta a porovnáva pri nahrávaní na server. Upravené XML súbory sú odmietnuté.

V grafickom rozhraní je implementovaná funkcia `TForm::ParseResults()`, ktorá z tohto XML súboru dynamicky generuje HTML stránku. Tá predkladá výsledky užívateľovi v prijateľnej grafickej forme s tabuľkami a grafmi. Obsahuje takisto odkazy do on-line databázy a na nahratie výsledkov:

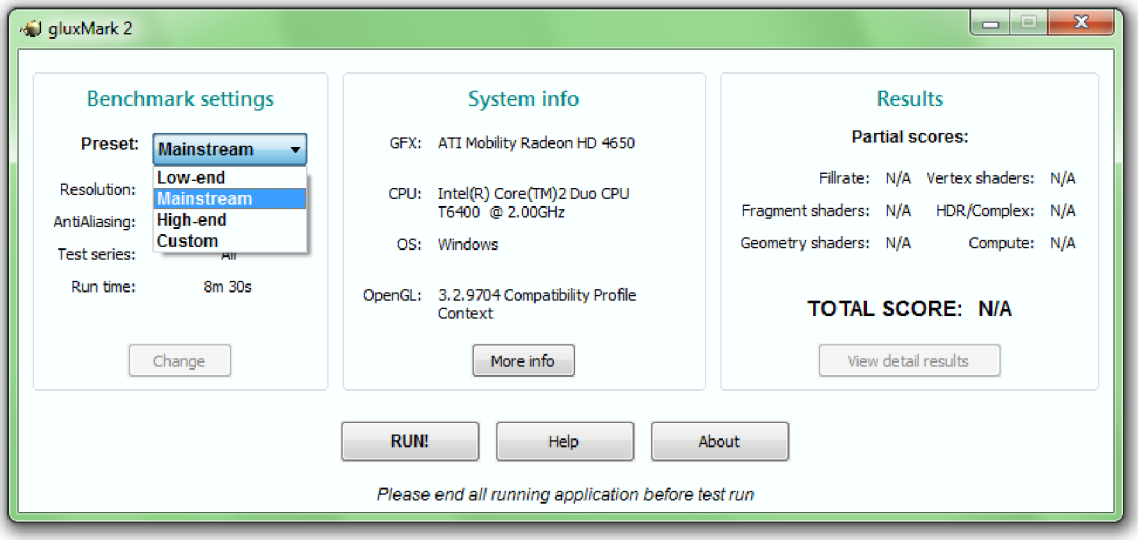


Obrázok 5.1: Vygenerovaná HTML stránka s výsledkami

5.2.3 Nastavenie testovania

Samotné testovanie znamená spustenie aplikácie a prebehnutie všetkých sérií testov. Aby sme mohli porovnávať rôzne akcelerátory, treba dodržať jednotné nastavenie parametrov testovania. Hoci je aplikácia navrhnutá ako multiplatformná, pre rôzne systémy by nemuselo vyhovovať jednotné grafické nastavenie aplikácie. Z tohto dôvodu sú pripravené 3 výkonnostné nastavenia (**presets**) tak, aby vyhovovali čo najväčšiemu počtu rôzne výkonných systémov s rôznymi rozlíšeniami monitora. Presety sa líšia v nastavení rozlíšenia a vyhladzovania hrán (antialiasing, AA):

- **High-End preset:** pre najvýkonnejšie stroje, nastavenie je 1920x1080, 8xAA. Toto rozlíšenie je charakteristické pre FullHD displeje, veľký počet pixelov (2 Mpix) a vzorkov vyhladzovania kladie vysoké nároky na grafickú kartu. Toto nastavenie je najmenej závislé na ostatných súčastiach systému (hlavne CPU).
- **Mainstream preset:** pre stredne výkonné stroje, 1280x720, 4xAA. Rozlíšenie označované aj ako HD Compatible, vhodné aj pre notebooky. Nároky tu sú nižšie, ale väčšmi sa prejaví CPU vplyv (snaha o potlačenie zavedením 4x antialisingu)
- **Low-End preset:** pre najslabšie stroje a notebooky, nastavenie 1024x768, AA off. Toto nastavenie je najviac závislé aj na CPU, takže by sa naozaj malo používať len na slabých strojoch neschopných spustiť vyššie nastavenia.



Obrázok 5.2: Grafické rozhranie s výberom nastavení testovania

V grafickom rozhraní je ďalej možné si prispôsobiť nastavenia (**Custom preset**), no takého výsledky nie je možné globálne porovnávať medzi sebou a ani nie sú zahrnuté do štatistík v databáze. Okrem nastavení rozhranie obsahuje aj informácie o systéme, odkazy na domovskú stránku projektu ako aj výzvu na ukončenie všetkých aplikácií pred začatím testovania. Spustené aplikácie totiž môžu negatívne ovplyvňovať priebeh testu a takto získaný výsledok je potom bezcenný.

5.2.4 Výpočet skóre

Skóre sa počíta z váh jednotlivých testov (vysvetlenie váh a dôležitosti testov je v kapitole 4), tieto váhy si teraz zhrnieme v tabuľke. Skóre sa uvažuje ako výsledok na referenčnej karte v nastavení Mainstream:

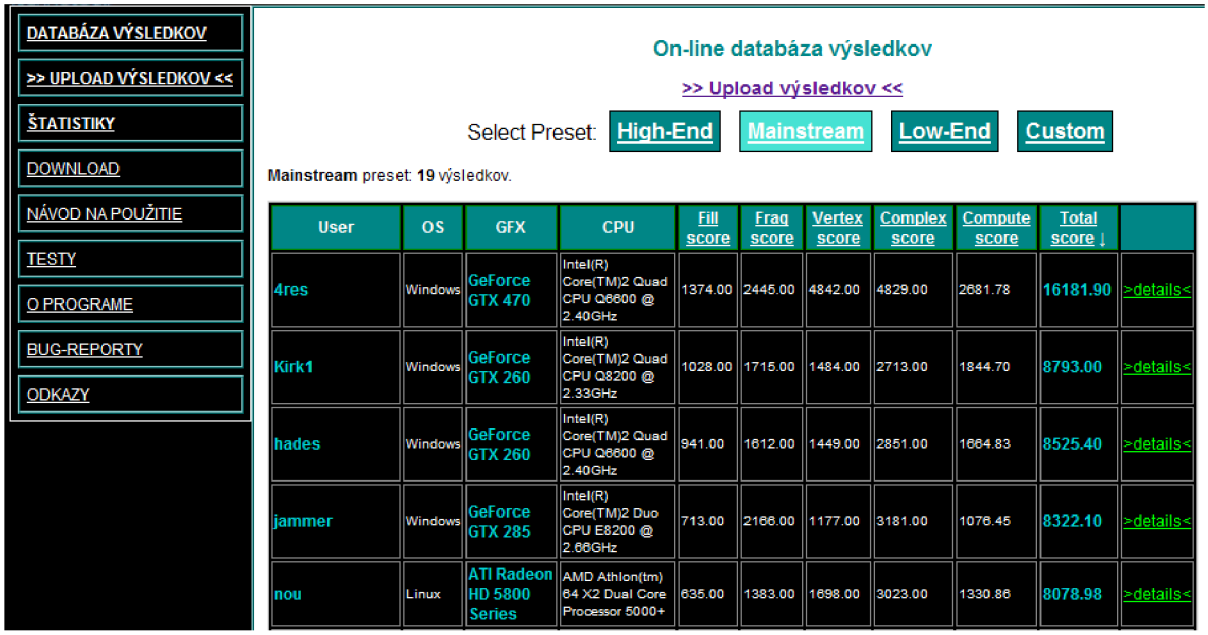
Poradie	Názov testu	Váha v rámci série	Váha celkom	Skóre
Fillrate testy				
1.	Fillrate - Single texturing	15,00%	1,80%	180
2.	Fillrate - Multi texturing	30,00%	3,60%	360
3.	Fillrate - 16-bit floating point texures	40,00%	4,80%	480
4.	Fillrate - 32-bit floating point texures	15,00%	1,80%	180
Fragment shader testy				
5.	Per-pixel point lights	30,00%	6,60%	660
6.	Perlin noise	20,00%	4,40%	440
7.	Procedural shaders	10,00%	2,20%	220
8.	Parallax mapping	20,00%	4,40%	440
9.	Static reflections and refractions	20,00%	4,40%	440
Geometry shader testy				
10.	Tesselation with depth map	30,00%	3,00%	300
11.	Single pass dynamic cubemapping	0,00%	0,00%	0
12.	GPU particles	50,00%	5,00%	500
13.	Geometry shader instancing	20,00%	2,00%	200
Vertex shader testy				
14.	Per-vertex lighting with geometry instancing	45,00%	7,65%	765
15.	GPU generated waves	25,00%	4,25%	425
16.	Vertex displacement mapping	30,00%	5,10%	510
Komplexné testy				
17.	HDR lighting	30,00%	8,10%	810
18.	Screen space ambient occlusion	30,00%	8,10%	810
19.	Dynamic soft shadows	30,00%	8,10%	810
20.	Render with supersampling	10,00%	2,70%	270
Výpočtové testy				
21.	N-Body simulation	100,00%	12,00%	1200
	CELKOM:		**100,00%**	**10000**

Celkové skóre je prostým súčtom skóre jednotlivých testov. Toto výsledné skóre je asi najdôležitejším výstupom aplikácie, pretože umožňuje priamo porovnávať medzi sebou rôzne grafické karty.

5.2.5 On-line databáza

Aplikácia síce ako finálny výstup ponúka celkové skóre v porovnaní s referenčnou kartou, avšak samé o sebe by bolo nepoužiteľné, pretože potrebujeme hromadne porovnávať rôzne grafické karty, nielen testovanú s referenčnou. Preto som vytvoril on-line databázu výsledkov, ktorá umožňuje nahrávať výsledky testovania a porovnávať ich medzi rôznymi grafickými kartami.

Databáza sa nachádza na stránkach http://www.stud.fit.vutbr.cz/~xvanek29/gluxmark2/, stránky sú naprogramované s využitím jazyka PHP a MySQL databázy na uloženie výsledkov. Stránky okrem samotnej databázy obsahujú aj odkazy na stiahnutie aplikácie, pokyny k testovaniu, návod na použitie a HW nároky aplikácie. Výsledky sú ukladané a zobrazované pre každý výkonnostný preset zvlášť. Je možné si ich zoradiť podľa celkového, ako aj podľa čiastkových skóre, pretože v niektorých sériách sú rýchle grafické karty, ktoré môžu mať celkové skóre nižšie ako karty pred nimi.



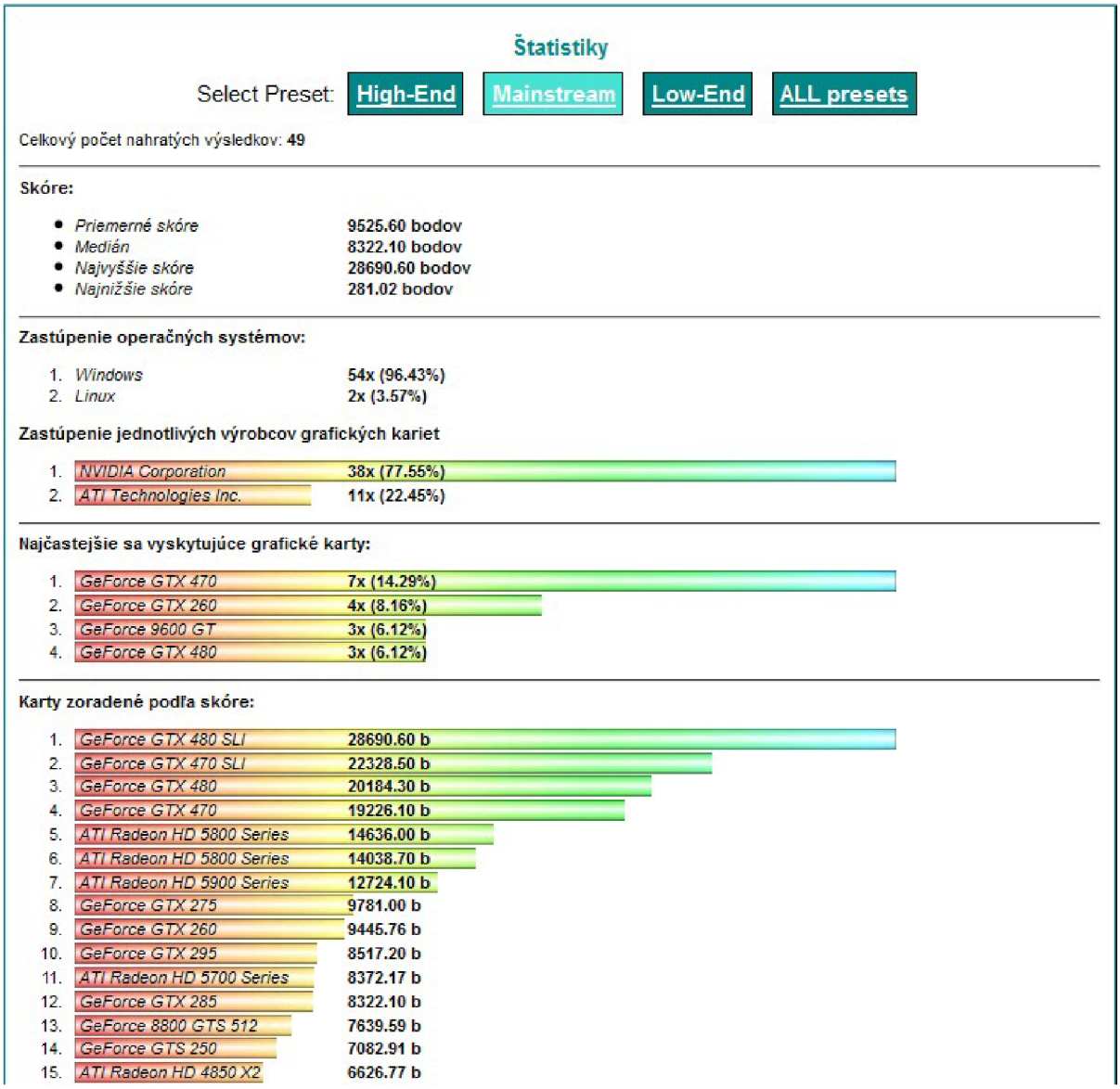
User	OS	GFX	CPU	Fill score	Frag score	Vertex score	Complex score	Compute score	Total score ↓	
4res	Windows	GeForce GTX 470	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	1374.00	2445.00	4842.00	4829.00	2681.78	16181.90	>details<
Kirk1	Windows	GeForce GTX 260	Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz	1028.00	1715.00	1484.00	2713.00	1844.70	8793.00	>details<
hades	Windows	GeForce GTX 260	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	941.00	1612.00	1449.00	2851.00	1664.83	8525.40	>details<
jammer	Windows	GeForce GTX 285	Intel(R) Core(TM)2 Duo CPU E8200 @ 2.66GHz	713.00	2166.00	1177.00	3181.00	1076.45	8322.10	>details<
nou	Linux	ATI Radeon HD 5800 Series	AMD Athlon(tm) 64 X2 Dual Core Processor 5000+	635.00	1383.00	1698.00	3023.00	1330.86	8078.98	>details<

Obrázok 5.3: Stránka aplikácie s on-line databázou výsledkov

U každého nahratého výsledku je možné zobraziť aj podrobnosti o jednotlivých testoch, ako aj informácie o systéme a nastavení testu. Tieto informácie sú uložené na serveri v podobe XML súborov (výsledkové súbory, ako je popísané v sekcii 5.2.2). Bol preto napísaný parser v jazyku PHP, ktorý vygeneruje výsledkovú stránku z údajov v súbore - podobne ako sa generuje táto stránka v programe. Keďže zdieľajú rovnaký CSS štýl, aj vyzerajú rovnako (Obrázok 5.1).

Upload výsledku spočíva v nahratí vytvoreného súboru results.xml (popis v sekcii 5.2.2) pomocou formuláru na stránke. Pri tomto nahrávaní sa kontroluje kontrolná suma u skóre, čím umožní detekovať falošné a zámerne upravené výsledky.

Zo všetkých výsledkov sa potom vytvára štatistika, ktorá obsahuje údaje o počte testovaných systémov, priemerné skóre a medián. Potom ukazuje jednotlivé zastúpenie výrobcov grafických kariet a najrozšírenejšie karty medzi užívateľmi. Pre určenie najvýkonnejších grafických kariet tu sú výkonové rebríčky, ktoré v každej sérii testov ukazujú najvýkonnejšie karty. Podobne ako u databázy, aj štatistiky sú rozdelené podľa troch výkonnostných presetov.



Obrázok 5.4: Stránka so štatistikami z nameraných výsledkov

Okrem databázy a štatistík sa na stránkach nachádzajú aj odkazy na stiahnutie aplikácie, základné informácie o programe a pokynoch, ako sa má testovať. Súčasne je tam prehľad chýb v programe, ktoré boli odhalené počas testovania. Pre zahraničných užívateľov sú stránky preložené aj do anglického jazyka.

6 Výsledky testovania

K dobe odovzdania práce bolo aplikáciou otestovaných **120** systémov s rôznymi grafickými kartami. Táto vzorka je síce príliš malá na seriózne závery o výkonnosti rôznych grafických akcelerátorov, ale dá sa z nej určiť, ktoré akcelerátory podávajú najvyšší výkon v špecifických oblastiach grafických výpočtov.

Pre predstavu o závislosti HW výbavy grafickej karty (najmä technologickej úrovne GPU a rýchlosti VRAM) na výsledkoch testu si podrobne rozoberieme výkonnosť vzorových akcelerátorov na každom teste. Ďalej vysvetlíme príčiny rozdielov vo výkone a na záver porovnáme s výsledkami v komerčných aplikáciách.

Bohužiaľ, vzhľadom k problémom s výkonnosťou, pádmi aplikácie a chybami v obraze, boli z testovacej aplikácie **vynechané geometry shader testy**. Ich výsledná váha v celkovom skóre tak je 0% a nie sú vôbec započítané do celkového hodnotenia (je ich ale možné spustiť pri vlastnom nastavení testovacej aplikácie). Tento krok pramení zo snahy zaručiť bezproblémový priebeh testov na čo najširšej škále rôznych systémov.

6.1 Analýza výkonnosti vzorových akcelerátorov

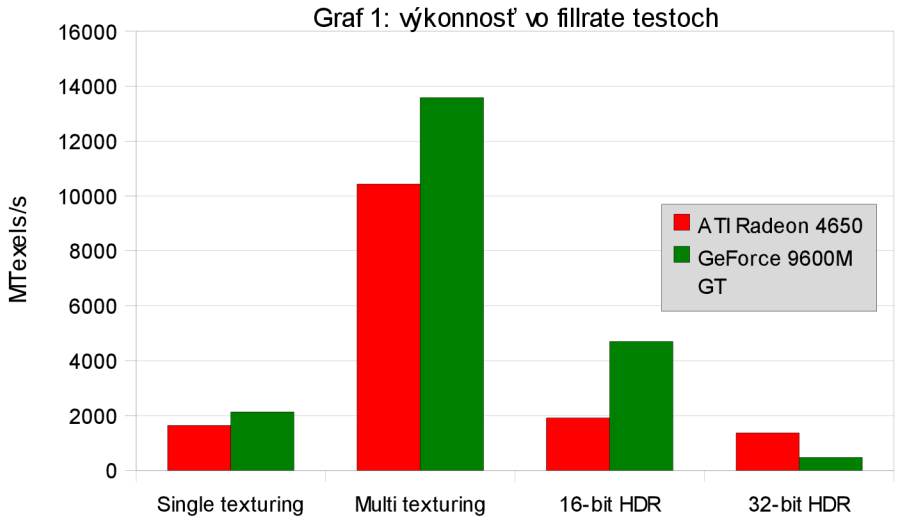
Ako vzorové karty zvolíme ***nVidia GeForce 9600M GT*** a ***ATI Mobility Radeon HD 4650***. Dôvody voľby práve týchto kariet sú také, že sú od odlišných výrobcov, obe podporujú všetky funkcie z OpenGL 3.2 a sú pomerne rozšírené (v notebookoch). Rozdiely v HW sú také, že *GeForce* má jadro s menej výpočtovými jednotkami, ale rýchle pamäte (testovaná verzia, vyrábajú sa aj s DDR2 pamäťami), *Radeon* má jadro s viacej stream procesormi, ale pomalšie pamäte. Bude zaujímavé sledovať, ako sa tieto rozdiely prejavia v testoch, pretože v reálnych aplikáciách ukazuju veľmi podobný výkon. *Radeon* je mierne výkonnejší, to však ale platí len pre verziu s DDR2 pamäťami. Verzia s DDR3 je znateľne rýchlejšia (až o 20%). Predstavme si teraz technické parametre:

	nVidia GeForce 9600M GT	***ATI Mobility Radeon HD 4650***
Počet stream procesorov:	32	320 (64 5D jadier[15])
Rýchlosť jadra:	500 MHz	550 MHz
Rýchlosť a typ pamätí:	800 MHz GDDR3	600 MHz DDR2
Šírka zbernice:	128-bit	128-bit
Veľkosť pamäte:	1024 MB	1024 MB
Výrobná technológia:	65nm	55nm
Počet tranzistorov:	314 miliónov	514 miliónov

Teraz si rozoberieme výkonnosť akcelerátorov v každej testovej sérii (ako popísané v kapitole 4). Oba akcelerátory boli testované na podobných systémoch (notebookoch) s procesorom Intel Core 2 Duo 2 GHz a 4 GB RAM. Ako operačný systém bol použitý Windows 7 64-bit. U každej série je graf, ktorý ukazuje výkon v každom teste. Ak je niektorý test zložený z viacerých sub-testov (napr. u per-pixel osvetlenia sa postupne zapínajú svetlá od 1 až po 8), je uvedené skóre z najnáročnejšieho nastavenia.

15 ATI používa VLIW superskalárne stream procesory v skupinách po 5 (5D), nVidia používa 1D skalárne procesory, ale taktované na dvojnásobok frekvencie jadra [6]

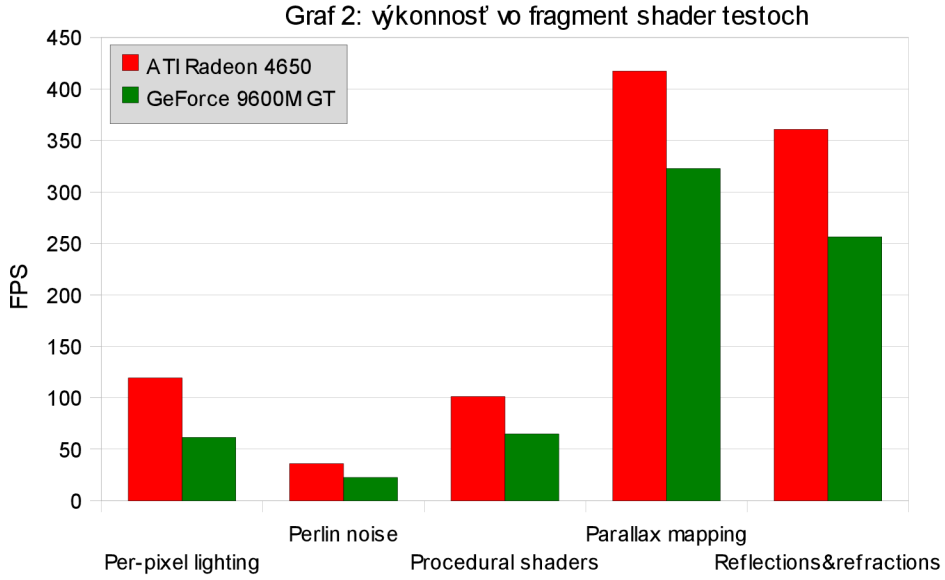
6.1.1 Fillrate testy



Skóre: *Radeon* **214b**, *GeForce* **221b**. *GeForce* je rýchlejšia o **3%.**

Podľa grafov má v tejto sérii navrch karta *GeForce*. To sa dá vysvetliť rýchlou GDDR3 pamäťou na frekvencii 800MHz, ktorá umožňuje rýchle čítanie textúry z VRAM. Rozdiel je najväčší v teste na render do 16-bit floating-point textúry, kde je navyše ešte potrebné prenášať dáta z framebufferu do textúry. Avšak skóre sa dramaticky mení u 32-bit HDR textúr. Ako spomínané v sekcii 4.1.4, súčasné akcelerátory sú optimalizované primárne na 16-bit textúry pre malé rozdiely v kvalite a menšie priestorové nároky ako u 32-bit formátu. *Radeon* v tomto prípade ukazuje lepšiu optimalizáciu, čo v konečnom dôsledku ovplyvňuje celkové skóre z tejto série testov a znižuje výkonnostný náskok *GeForce*.

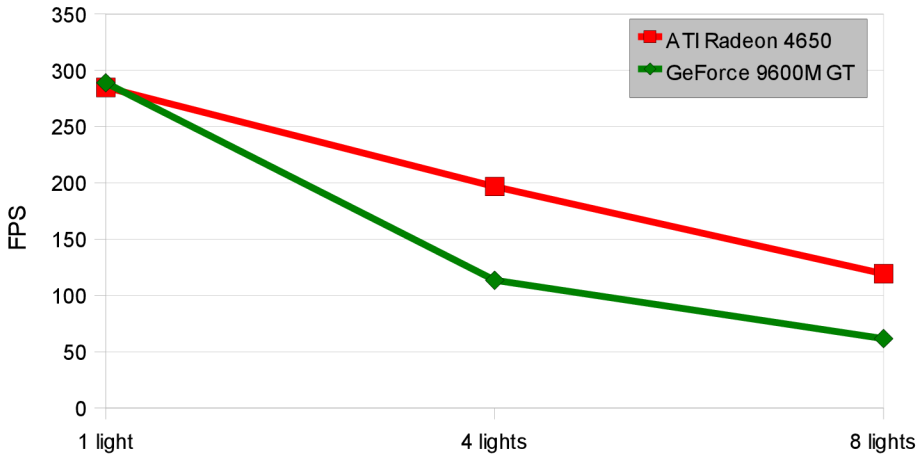
6.1.2 Fragment shader testy



Skóre: *Radeon* **479b**, *GeForce* **347b**. *Radeon* je rýchlejší o **38%.**

V tejto sérii naopak *Radeon* ukazuje, že pri náročných per-pixel efektoch je viac dôležitý počet stream procesorov spracúvajúci fragment shader kód ako rýchla VRAM na aplikáciu textúr. Dvojnásobok počtu stream procesorov znamená výkonový nárast takmer o 40%. K vyššiemu nárastu by určite pomohla aj rýchlejšia VRAM. Výkonový náskok *Radeonu* je prakticky rovnaký vo všetkých testoch. Pre zaujímavosť si ešte uvedieme graf testu per-pixel osvetlenia s postupným zapínaním svetiel:

Graf 3: výkon pri per-pixel osvetlení

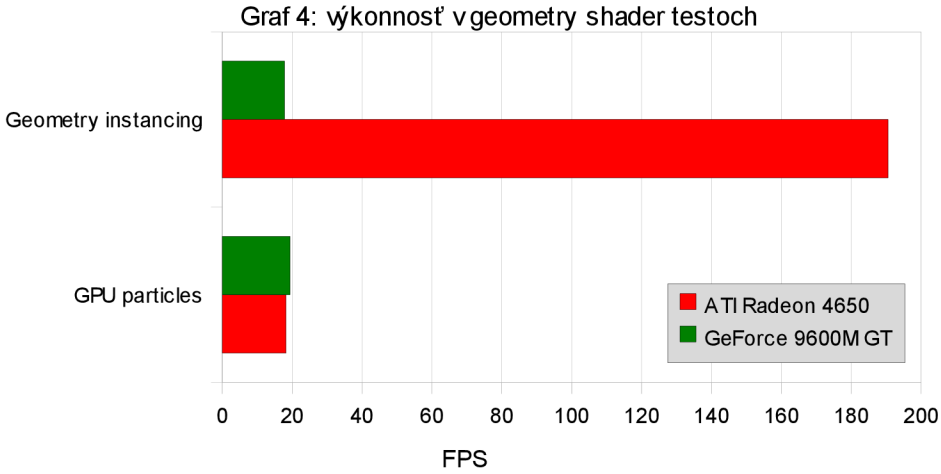


Aj keď pri nastavení jedného svetla je výkon takmer totožný, pri zapnutí viacerých svetiel je vidieť, že prepad výkonu u *GeForce* je omnoho výraznejší (u 8 svetiel je takmer dvojnásobný).

6.1.3 Geometry shader testy

U tejto série vynechávame prvé dva testy z dôvodu nefunkčnosti – na *Radeonu* nefungovala teselácia, na *GeForce* zase dynamické odrazy.

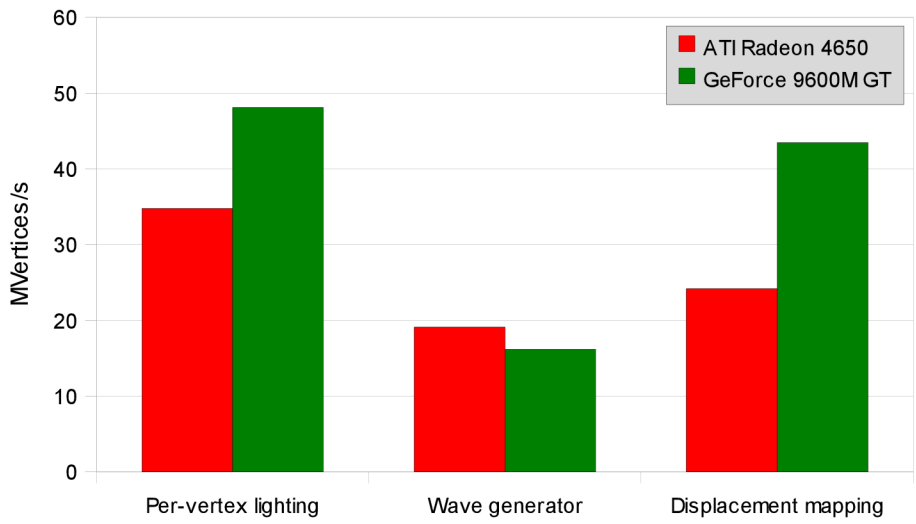
Graf 4: výkonnosť v geometry shader testoch



Vypovedaciu hodnotu má len časticový test – tu vidíme podobný výkon geometry shaderu u oboch kariet. Naopak, u instancingu je obrovský prepad výkonu u karty *GeForce*. Tento prepad nie je spôsobený rozdielmi v HW akcelerátorov, ale takmer isto sa jedná o problém s ovládačmi. Z toho dôvodu je aj ťažké určiť, prečo je výkon v časticovom teste podobný (keďže *Radeon* má viacej stream procesorov). Preto u tejto série testov nebudeme robiť výkonnostné porovnania.

6.1.4 Vertex shader testy

Graf 5: výkonnosť vo vertex shader testoch

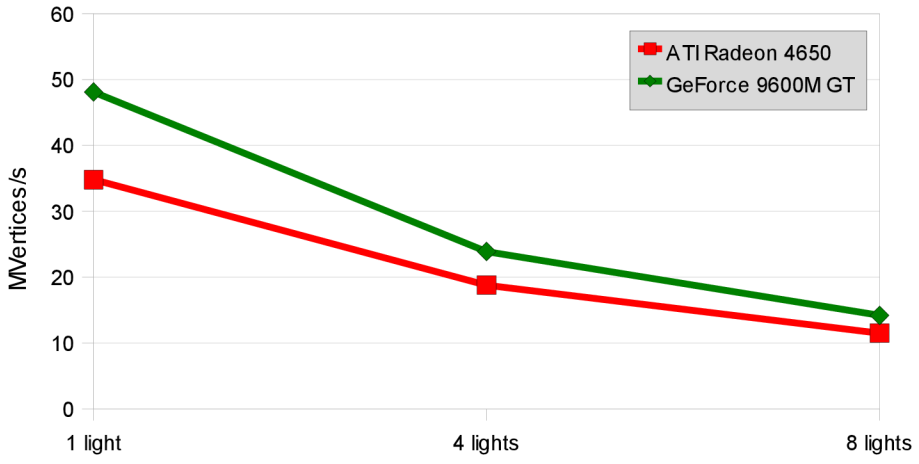


Skóre: *Radeon* **352b**, *GeForce* **451b**. *GeForce* je rýchlejšia o **28%**.

V tejto sérii má opäť viacej výkonnostne navrch karta *GeForce*. Vysvetlenie opäť spočíva v rýchlej VRAM, keďže jadro môže rýchlejšie pristupovať k vertexovým dátam uloženými v podobe vertex bufferov v pamäti. Ešte väčší nárast zaznamenáva v teste na displacement mapping, kde navyše karta ešte musí pristupovať k textúre deformujúcej povrch.

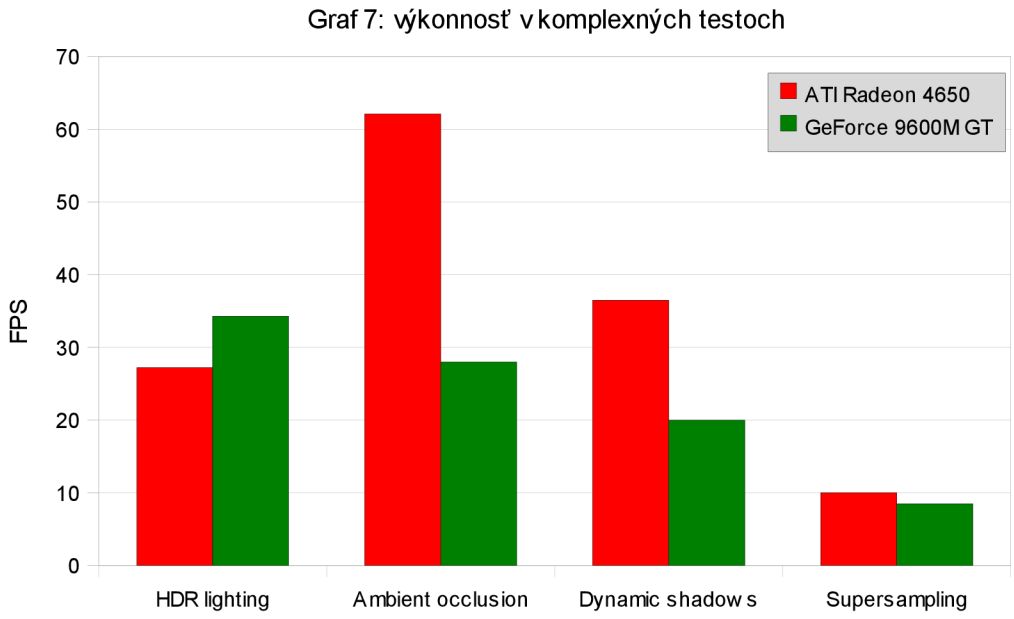
Situácia sa mení (v prospech *Radeonu*) v teste na generátor vĺn. Keďže sa tu používa málo externých dát (len textúra oblohy) a všetko ostatné sa generuje procedurálne, výhodu má karta s viacej výpočtovými jednotkami v jadre. Rozdiel ale nie je až taký razantný (18%). Podobne ako u per-pixel osvetlenia, ukážeme si aj graf per-vertex osvetlenia (opäť z jedného svetla až po 8):

Graf 6: výkon pri per-vertex osvetlení



Výkonové poklesy sú zodpovedajúce a u oboch kariet podobné (s vyšším výkonom na karte *GeForce*).

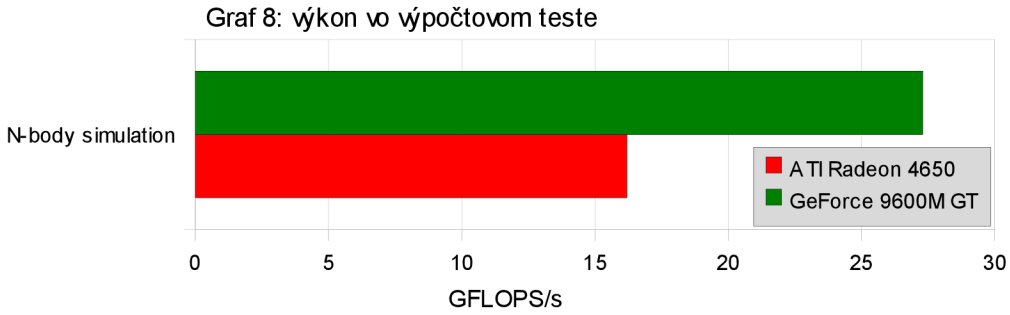
6.1.5 Komplexné testy



Skóre: *Radeon* **785b**, *GeForce* **492b**. *Radeon* je rýchlejší o **60%**.

Komplexné testy dávaju pomerne zaujímavé výsledky. Dalo by sa očakávať, že testy budú viac závislé na rýchlosti VRAM vzhľadom k použitému algoritmu renderu do textúry u týchto efektov. Je to ale vidieť len u prvého testu na HDR osvetlenie. Vysvetlenie: všetky testy okrem HDR osvetlenia sú pomerne náročné na výkon fragment shaderu, kdežto u HDR fragment shader len extrahuje jasné časti z obrazu a vykonáva rozmazanie. Navyše je v tomto prípade použité vyhladzovanie hrán, čo vyžaduje vyšší dátový tok z dôvodu použitia multisampled framebuffer objektu a následnej kópii dát do obyčajného FBO. Naproti tomu ambient occlusion efekt používa pomerne zložitý shader kód (detekcia tmavých oblastí, počítanie poklesu), preto je vidieť najvyšší nárast výkonu *Radeonu* oproti *GeForce*. Podobne u dynamických tieňov, kde sa používa pomerne náročné porovnávanie hodnoty tieňovej textúry so z-bufferom, projekcia tieňa a rozmazanie jeho okrajov. U supersamplingu nie sú rozdiely veľké, tu sa vplyv GPU a VRAM vyrovnáva.

6.1.6 Výpočtové testy



Skóre: *Radeon* **242b**, *GeForce* **408b**. *GeForce* je rýchlejšia o **69%**.

V tomto teste má jasnú prevahu karta *GeForce*. Pre vysvetlenie treba poznať architektúry oboch kariet. Ako spomínané v sekcii 4.6, karty generácie *Radeon HD 4000* nemajú plnú podporu OpenCL v tom, že nepodporujú lokálnu pamäť (je emulovaná cez globálnu). V tom dôsledku napriek vyššiemu počtu stream procesorov (64 oproti 32) ukazuje karta *GeForce* vyšší výkon. Ten je navyše aj umocnený rýchlejšou VRAM. Ak by karta *Radeon* ale podporovala lokálnu pamäť, výsledky by boli otočené (*Radeon HD 4650* má teoretický výpočtový výkon takmer 2x vyšší ako *GeForce 9600M GT*).

6.1.7 Zhrnutie

Na záver porovnáme získané výsledky s výsledkami z komerčných benchmarkov (3DMark 2006,05,03 a 3DMark Vantage). Opäť, benchmarky prebehli na oboch testovaných grafických kartách s rovnakými nastaveniami na notebookoch s CPU Intel Core 2 Duo 2GHz, 4GB RAM a Win7 64-bit:

	Mobility Radeon HD 4650	***GeForce 9600M GT***	**%-ný náskok Radeonu**
gluxMark2	**2028b**	**1927b**	**5,2%** (**21,1%** bez OpenCL)
3D Mark Vantage (P-preset)	1995b	1459b	**36,7%**
3D Mark 2006	4985b	5683b	**-14%**
3D Mark 05	9949b	9592b	**3,7%**
3D Mark 03	15 407b	14 889b	**3,4%**

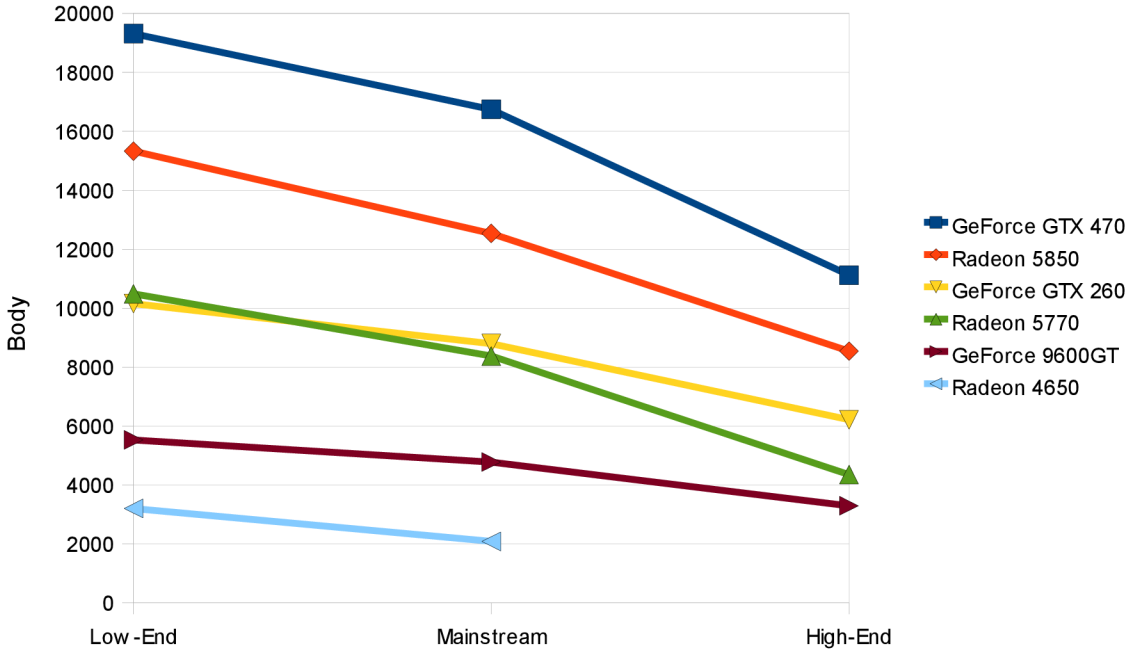
Z mojej aplikácie vyšlo, že *Radeon* je rýchlejší o **5,2%**. Na začiatku tejto sekcie sme začínali s tým, že v reálnych aplikáciách a benchmarkoch je karta *Radeon* (testovaná verzia) mierne rýchlejšia, čiže tento výkonový rozdiel by zodpovedal. Podobné výsledky dosahujú aj 3DMark 2005 a 2003, avšak v 3D Mark 2006 je *GeForce* rýchlejšia. To by sa dalo vysvetliť tým, že táto verzia 3D Marku vo veľkej miere používa HDR testy a ako sme si ukázali v sekcii 6.1.5, *GeForce* má v takýtcho testoch výkonnostný náskok.

Zaujímavá situácia ale nastane, ak si zoberieme výsledky bez OpenCL testu, kde má *Radeon* významný výkonnostný handicap. Ak odpočítame tento test z celkového hodnotenia, dostaneme skóre pre *Radeon* **1840b** a pre *GeForce* **1519b**. Potom je rozdiel medzi kartami **21%**, čo už síce nezodpovedá priamo výsledkom z reálnych aplikácií, ale blíži sa to výsledku benchmarku 3D Mark Vantage (**36%**). Dôvod takéhoto rozdielu môže byť nasledovný: 3DMark Vantage využíva funkcie z DirectX 10, ktoré je ekvivalentom OpenGL 3.2 u mojej aplikácie. Reálne aplikácie nevyužívajú často všetky možnosti rozhrania (napr. u hier je stále najrozšírenejšie rozhranie DirectX 9) a preto sa výkon v novších rozhraniach môže líšiť – nemusí záležať až tak na hrubom výkone akcelerátoru, ale aj jeho technologickej úrovni, prípadne na úrovni ovládačov.

6.2 Výsledky z on-line databázy

K dňu dokončenia práce (22.5.2010) sa podarilo nazbierať **117** výsledkov, z toho **34** pri nastavení High-End, **56** pri nastavení Mainstream a **16** pri nastavení Low-End. Ostatné výsledky **(11)** boli nahraté pri vlastnom nastavení benchmarku. Niekoľko užívateľov nahralo svoje výsledky pod rôznymi presetmi, môžeme si tak ukázať výkonové poklesy pri zvyšovaní rozlíšenia a úrovne vyhladzovania hrán.

Graf 9: Výkonnosť v rôznych nastaveniach (presetoch)



Výkonnostné prepady sú najvýraznejšie u slabších kariet (*Radeon HD 4650*), a potom hlavne medzi nastaveniami Mainstream a High-End. Silné karty ako *Radeon HD 5850* majú poklesy skoro lineárne, čo svedčí o sile týchto čipov aj vo vysokých rozlíšeniach s vysokým stupňom vyhladzovania hrán.

V nasledujúcom texte si rozoberieme výsledky z jednotlivých nastavení (presetov), pričom všetky výsledky pochádzajú z on-line databázy a jej štatistík.V texte sa budú vyskytovať pojmy ako „karta s podorou DirectX 11“, čo znamená, že ide o kartu s podporou shader modelu 5.0 a OpenGL 4.0 (podobne ako „karta s podporou DirectX 10“ znamená kartu s podporou shader modelu 4.0 a OpenGL 3.2). Tieto pojmy sú uvádzané na skrátenie textu a vychádzajú z bežnej praxe pri označovaní jednotlivých generácií akcelerátorov. Ďalej sa pri vysvetľovaní výkonnostných rozdielov budeme často odkazovať na popis architektúr moderných akcelerátorov, ktorý je možné nájsť v [6].

6.2.1 High-End preset

Toto nastavenie (rozlíšenie 1920x1080, 8x vyhladzovanie) je určené hlavne pre výkonné grafické akcelerátory, čo sa preukázalo aj pri nahratých výsledkoch (majú vyššie zastúpenie). Výsledky si zhrnieme v tabuľke:

Počet nahratých výsledkov:	**34**
Počet unikátnych výsledkov:	**30**
Najvyššie skóre:	**19 674,3b** (*GeForce GTX 480 SLI*)
Najnižšie skóre:	**1367,76b** (*GeForce GT 240M*)
Priemerné skóre:	**8590,62b**
Medián:	**9009,82b**
Najviac rozšírený OS:	Windows **(100%)**
Najviac rozšírený výrobca graf. kariet:	nVidia **(70,00%)**
Najviac rozšírené karty:	*GeForce GTX 470* **(23,33%)** *Radeon 5850/5870*[16] **(16,67%)**

Pri tomto nastavení sa najčastejšie vyskytovali najvýkonnejšie externé karty od oboch hlavných výrobcov (AMD/ATI a nVidia). Najčastejšie zastúpené boli zástupcovia DirectX 11 kariet od nVidie, teda čipy *GeForce GTX 470* a *480*. Druhý menovaný čip dosiahol výkonnostný rekord so ziskom takmer 20-tisíc bodov. Treba však spomenúť, že išlo o dve grafické karty v SLI konfigurácii. Tieto karty (spolu s DirectX 11 kartami od ATI, *Radeon HD 5850* a *5870*) nemali prakticky žiadne výkonové problémy pri behu testov, snímková frekvencia sa držala vysoko nad 100 FPS. Je zrejmé, že ak užívateľ chce maximálny výkon, tak treba siahnuť po takýchto kartách.

Čo sa týka DirectX 10 generácie kariet (najčastejšie zastúpené čipmi *GeForce GTX 260,285,295* a *Radeon HD 4850/70*) tiež ukazovali veľmi dobrý výkon aj v tomto extrémnom nastavení. Výrazne zaostávali ale mobilné verzie týchto čipov (napr. *Mobility Radeon HD 5850*, *GeForce GT 240M*).

6.2.2 Mainstream preset

Najviac výsledkov bolo nahratých pri tomto nastavení (rozlíšenie 1280x720, 4x vyhladzovanie). Je tu aj najväčšia škála rôznych akcelerátorov, od integrovaných (napr. *ATI Radeon HD 3200*) až po tie najvýkonnejšie externé (napr. *GeForce GTX 480*).

Počet nahratých výsledkov:	**56**
Počet unikátnych výsledkov:	**49**
Najvyššie skóre:	**28 690,6** (*GeForce GTX 480 SLI*)
Najnižšie skóre:	**281,02b** (*Radeon HD 3200*)
Priemerné skóre:	**9525,60b**
Medián:	**8322,10b**
Najviac rozšírený OS:	Windows **(96,43%)**
Najviac rozšírený výrobca graf. kariet:	nVidia **(77,55%)**
Najviac rozšírené karty:	*GeForce GTX 470* **(14,29%)** *GeForce GTX 260* **(8,16%)**

16 informácie o karte z OpenGL nazývajú tieto karty ako „ATI Radeon HD 5800 Series“, takže sa nedá presne rozlíšiť, či ide o verziu 5850 alebo 5870

Výkonovo opäť dominovali DirectX 11 karty od nVidie, podobne ako v nastavení High-End. To aj preto, že mnoho užívateľov nahralo výsledky pod obomi nastaveniami. Toto nastavenie bez problémov zvládali aj DirectX 10 čipy, ako aj ich výkonnejšie mobilné verzie. Výraznejšie prepady nastávali až u slabších čipov ako napr. *Radeon HD 4650* alebo *GeForce 9500*. Vyskytla sa tu aj jedna karta z DirectX 9 generácie – *GeForce 7300GT*, avšak ako vyplýva z nízkeho skóre (503b), takéto karty nie sú už vhodné pre moderné 3D aplikácie.

6.2.3 Low-End preset

Podľa očakávania, do tohto nastavenia (rozlíšenie 1024x768, bez vyhladzovania) bolo nahratých najviac výsledkov od slabších grafických kariet (integrovaných, prípadne slabšie verzie externých kariet v notebookoch).

Počet nahratých výsledkov:	**16**
Počet unikátnych výsledkov:	**16**
Najvyššie skóre:	**19 314,5b** (*GeForce GTX 470*)
Najnižšie skóre:	**630,68b** (*GeForce G105M*)
Priemerné skóre:	**8666,50b**
Medián:	**8806,22b**
Najviac rozšírený OS:	Windows (**100%**)
Najviac rozšírený výrobca graf. kariet:	nVidia (**62,50%**)
Najviac rozšírené karty:	*GeForce GTX 470* (**25,00%**)

Pre beh v tomto nastavení postačuje akákoľvek grafická karta s podporou DirectX 10, s výnimkou naozaj slabých a integrovaných čipov (*GeForce G105, Radeon HD 3200*). Bohužiaľ, databáza je ochudobnená o výsledky od kariet Intel, ktoré síce hardwarovo podporujú shader model 4.0 (a teda teoreticky aj OpenGL 3.2), avšak podpora týchto funkcií v ovládačoch chýba. Je to škoda, pretože tieto integrované grafické jadrá majú dominantné zastúpenie (a to nielen v notebookoch) a bolo by zaujímavé porovnať ich výkon v OpenGL s integrovanými čipmi od ATI a nVidie.

6.2.4 Celkom

Teraz si zhrnieme štatistiky zo všetkých výkonnostných nastavení. Keďže nemá zmysel uvádzať navzájom nekompatibilné skóre, zobrazíme si len zastúpenia grafických kariet a operačných systémov:

Počet nahratých výsledkov:	**120**
Počet unikátnych výsledkov:	**75**[17]
Najviac rozšírený OS:	Windows (**98,29%**)
Najviac rozšírený výrobca graf. kariet:	nVidia (**66,67%**)
Najviac rozšírené karty:	*Radeon HD 5850/5870* ***(14,67%)*** *GeForce GTX 470* (**12,00%**) *GeForce GTX 480* (**10,67%**)

17 unikátne výsledky sa počítajú výskyt jedného užívateľa s jednou grafickou kartou (je samozrejme možné, že jeden užívateľ nahral viacero výsledkov)

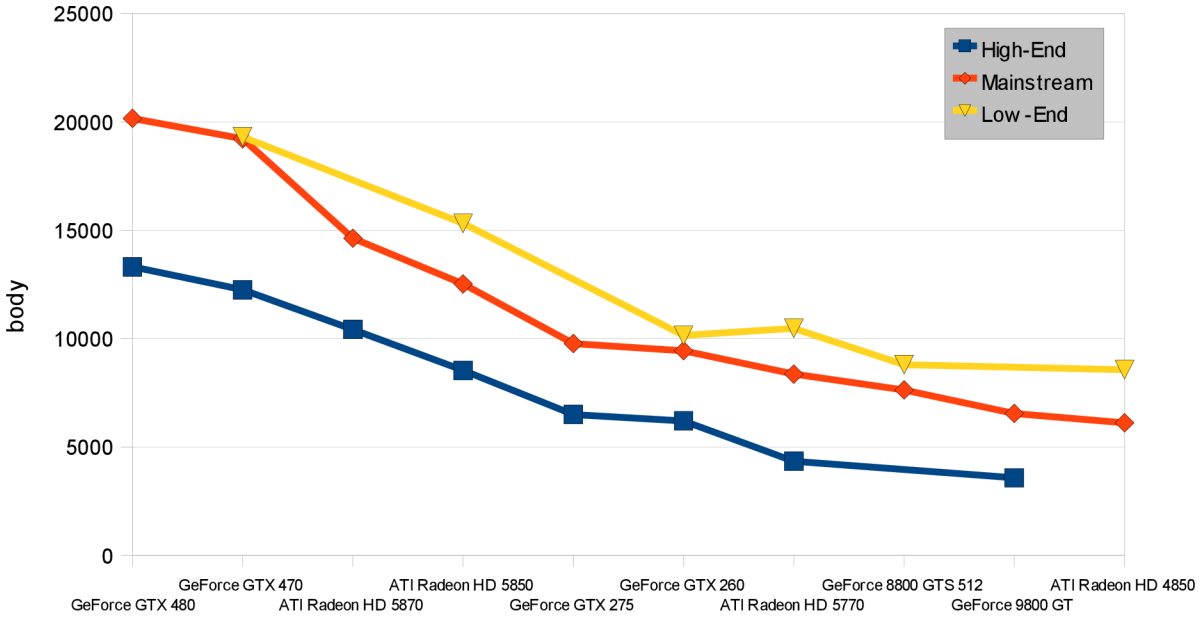
Medzi grafickými kartami tak majú najväčšie zastúpenie karty spoločnosti nVidia, a to viac ako dvojtretinové. Čo sa týka jednotlivých kariet, najviac zastúpení mali (aj napriek pomerne krátkej dostupnosti na trhu) karty *GeForce GTX 470/480* s kartami zo série *ATI Radeon HD 5800*. Pomerne rozšírené boli aj DirectX 10 karty *GeForce 9800/9600GT* a *GeForce GTX 260*.

Treba si uvedomiť, že táto malá vzorka výsledkov neodzrkadľuje reálne rozloženie jednotlivých grafických kariet, pretože väčšina výsledkov bola vytvorená technicky zdatnými užívateľmi s výkonnými zostavami a notebookmi. K štatisticky hodnotným záverom by sa dalo dôjsť až s rádovo tisícom nahratých výsledkov od rôznych užívateľov.

6.3 Zhodnotenie výsledkov

V tejto poslednej kapitole si zanalyzujeme všetky nazbierané výsledky z on-line databázy. Zameriame sa nielen na porovnávanie samotných grafických kariet, ale aj jednotlivých generácií (hlavne rozdiely medzi DirectX 11 a DirectX 10 generáciou kariet). Budeme sa venovať aj rozdielami medzi single- a multi-GPU konfiguráciami. Na úvod si ukážeme graf výsledkov desiatich najvýkonnejších jednočipových kariet vo všetkých nastaveniach:

Graf 10: 10 najvýkonnejších grafických kariet vo všetkých nastaveniach



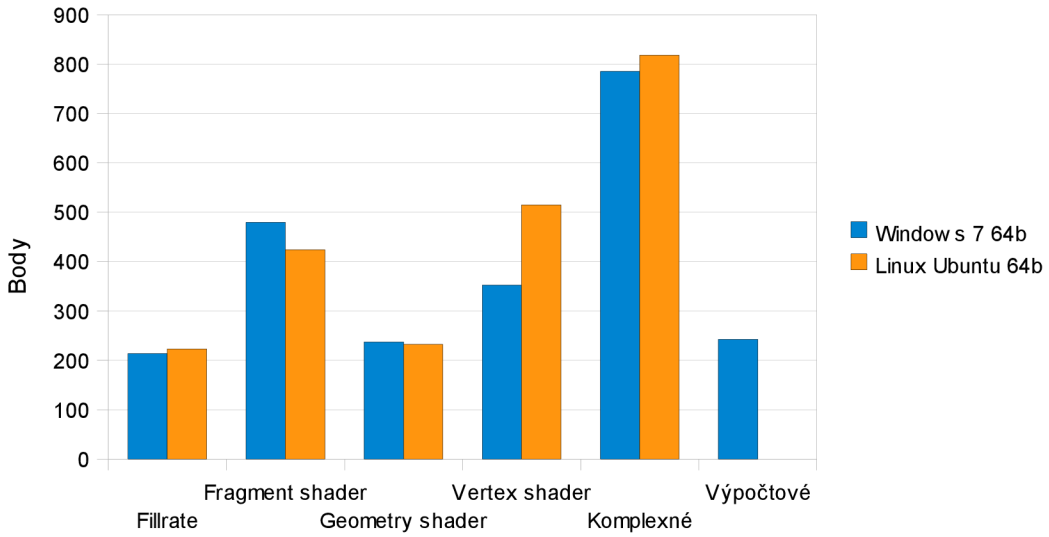
Ako vidíme z grafu, po výkonnostnej stránke dominovala moderná DirectX 11 generácia grafických kariet (*GeForce GTX 470,480* a *ATI Radeon HD* série *5700,5800,5900*), s pomerne výrazným odstupom od staršej, DirectX 10 generácie (*Radeony HD 2xxx,3xxx,4xxx* a *GeForce 8,9,2xx*). DirectX 9 generácia (zastúpená ale len jednou kartou) ukázala dnes už veľmi slabý výkon.

Najmenej výkonné boli podľa očakávania integrované čipy, ako aj externé karty, ktoré síce používajú jadro s podporou moderných technológií, no je veľmi zredukované (znížené počty stream procesorov, užšia pamäťová zbernica). Výhodou je však nízka spotreba a akcelerácia HD videa spolu s kompozitnými správcami okien Windows Aero alebo Compiz-Fusion.

Medzi operačnými systémami je podľa očakávania najsilnejšie zastúpenie systémov Windows (v štatistike sa nerozlišuje medzi jednotlivými verziami) s viac ako 98% podielom. Nízke rozšírenie

Linuxu ale môže byť spôsobené aj problémami pri behu aplikácie pod týmto OS. Pre zaujímavosť si môžeme uviesť porovnanie výkonnosti karty *ATI Radeon HD 4650* pod OS Windows 7 64-bit a Linux Ubuntu 8.10 64-bit, oba systémy používali ovládače ATI Catalyst vo verzii 10.3. Testy prebehli s nastavením Mainstream.

Graf 9: Porovnanie výkonnosti medzi Windows a Linux



Vidíme, že vo väčšine prípadov je výkon v oboch OS takmer identický. Väčšie rozdiely sú len vo vertex shader testoch v prospech Linuxu. Výpočtové testy pod Linuxom cez OpenCL bohužiaľ nefungovali na tomto akcelerátore. Celkovo ale môžeme vyhlásiť, že z hľadiska výkonnosti sú na tom ovládače pre Windows aj Linux rovnocenné (u ATI). Nemôžeme to ale vyhlásiť jednoznačne, pretože chýba viac výsledkov pod týmto OS.

Absolútne najvyššieho výkonu dosiahli dve karty *GeForce GTX 480* v SLI konfigurácii. Aplikácia síce nemá natívnu podporu multi-GPU, avšak niektorým užívateľom sa podarilo manuálnym nastavením dosiahnuť významné výkonové zisky pri použití druhej grafickej karty. Iné multi-GPU konfigurácie ale neboli také úspešné, ako ukazuje tabuľka. **Pozor:** okrem prvého výsledku (s kartami *GTX 480*) nie sú single-GPU výsledky dosiahnuté na tom istom systéme toho istého užívateľa, ale je vybraný výsledok na podobnom systéme od iného. To môže prinášať skreslenie, ale na zistenie či došlo k nárastu výkonu to postačuje.

Karta:	Skóre	Nárast oproti single-GPU
GeForce GTX 480 SLI	17 803,4 (High-End)	**42,80%**
GeForce GTX 480	12 467,3 (High-End)	-
GeForce GTX 470 SLI	22 328,5 (Mainstream)	**33,4%**
GeForce GTX 470	16 743, 6 (Mainstream)	
Radeon HD 5970 (2 GPU)	12 724,1b (Mainstream)	**1,5%**
Radeon HD 5850[18]	12 531,3b (Mainstream)	-
GeForce GTX 295 (2 GPU)	6504,8 (High-End)	**4,7%**
GeForce GTX 260[19]	6210,7 (High-End)	-

18 GPU Radeon HD 5850 približne odpovedá jednému GPU z duálnej karty Radeon HD 5970

19 GPU GeForce GTX 260 približne odpovedá jednému GPU z duálnej karty GeForce GTX 295

Významné škálovanie multi-GPU je dosiahnuté iba u kariet *GeForce GTX 480/470*,ktoré zaznamenali nárast vo výkone o viac ako jednu tretinu. Naopak, u ostatných konfiguráciách je zisk prakticky zanedbateľný a je vidieť, že z duálnej karty sa využíva len jedno GPU. Situáciu by mohlo zlepšiť manuálne nastavenie multi-GPU profilu v ovládačoch a optimalizácia aplikácie pre takéto konfigurácie.

Teraz si uvedieme najvýkonnejšie grafické karty v jednotlivých testových sériách, spolu aj s hodnotou najlepšieho výsledku. Bude treba ale rozlišovať medzi single- a multi-GPU konfiguráciami, ktré dosahujú vyšší výkon. Ak nie je uvedené inak, uvažuje sa Mainstream preset, ktorý obsahuje najviac nahratých výsledkov zo širokej škály akcelerátorov. Uvedieme len slovnú analýzu výsledkov, podrobné štatistiky je možné nájsť na webových stránkach aplikácie.

6.3.1 Výsledky vo fillrate testoch

Najvyšší fillrate medzi jednočipovými kartami dosahovali *ATI Radeon HD 5870*. Tieto karty majú vysokú taktovaciu frekvenciu GPU (viac ako 800MHz), čo spolu s 256-bit zbernicou a GDDR5 pamätami prináša veľmi vysoký teoretický výkon pri aplikácii textúr. Konkurenčné karty *GeForce GTX 480/470* síce tiež využívajú GDDR5 pamäti, no majú nižší takt jadra (len okolo 700MHz). Iná situácia je pri duálnych kartách, ktoré by mali mať teoreticky dvojnásobný fillrate (čo sa aj potvrdilo). Uvedieme si výsledky z multi-texturingu.

Najlepší výsledok (single-GPU): 62,4 GTexels/s, *ATI Radeon HD 5870*. Jedno-čipová *GeForce GTX 480* má síce vyšší fillrate pri multi-texturingu (65,9 GTexels/s), avšak náskok stráca pri floating-point textúrach.
Najlepší výsledok (multi-GPU): 110,6 GTexels/s, *GeForce GTX 480 SLI*

6.3.2 Výsledky vo fragment shader testoch

Výkon fragment shaderov má najvyšší karta *Radeon HD 5870*, avšak tesne nasledované kartami *GeForce 470/480*. Je to dané vysokým počtom stream procesorov v jadrách týchto kariet (napr. *Radeon HD 5870* ich má až 1600). Priebeh testov je na všetkých výkonných kartách podobný, so snímkovou frekvenciou presahujúcou 1000 FPS v niektorých testoch. Ako najviac náročný sa ukázal test na Perlinov šum so 16-timi oktávami, kde sa snímková frekvencia pohybovala najviac na hranici 200 FPS (v High-End nastavení).

Najlepší výsledok (single-GPU): 3272b, *Radeon HD 5870*
Najlepší výsledok (multi-GPU): 4988b, *GeForce GTX 480 SLI*

6.3.3 Výsledky vo vertex shader testoch

Vo vertex shaderoch majú veľmi vysoký náskok DirectX 11 karty od nVidie, ktoré dosahujú viac ako dvojnásobne lepšie výsledky oproti konkurenčným kartám ATI. To sa dá vysvetliť pomocou architektúry čipov *GTX 470/480* [6], kde bola oproti minulej generácii posilnená geometricko-transformačná jednotka vzhľadom na požiadavky DirectX 11/OpenGL 4.0 (HW teselácia [7]). ATI síce tiež podporuje DirectX 11 a tiež bola posilnená geometrická jednotka, avšak výkon v teselácii je nižší aj v iných testoch (napr. Unigine Heaven predstavený v sekcii 3.2). Nie je však vylúčená ani

chyba v ovládačoch ATI, pretože teoreticky by mala mať nová generácia dvojnásobný výkon oproti predchádzajúcej generácii, čo sa neprejavilo.

Najvýkonnejšie karty dokážu za jednu sekundu vykresliť obrovský počet, viac ako 500 miliónov vrcholov. Oproti DirectX generácii to je nárast priemerne takmer päťnásobný, pretože tie dokázali vykresliť max. 100 miliónov vrcholov (napr. *GeForce GTX 260*).
Najlepší výsledok (single-GPU): 646,9 MVertices/s, *GeForce GTX 480*
Najlepší výsledok (multi-GPU): 1225,9 MVertices/s *GeForce GTX 480 SLI*

6.3.4 Výsledky v komplexných testoch

Výsledky v týchto testoch by mal najviac zodpovedať výsledkom z reálnych aplikácií, čo sa čiastočne preukázalo. Vedú síce opäť karty *GeForce GTX 470/480*, avšak náskok pred *ATI Radeon* série *HD 5800* nie je až taký veľký (v High-End nastavení, pri Mainstream je rozdiel pomerne veľký). V reálnych aplikáciách síce bývajú ATI karty niekedy aj rýchlejšie, no treba brať do úvahy, že v rámci OpenGL nVidia už dlhý čas ukazujú lepšie výsledky vzhľadom k lepšej podpore OpenGL funkcií v ovládačoch.

Ako najnáročnejší test sa ukázal test na supersampling, ktorý na plynulý beh vyžaduje naozaj veľmi výkonnú kartu. Ostatné testy bežali aj na stredne výkonných kartách s výkonom viac ako 50 FPS.

Najlepší výsledok (single-GPU): 6150b, *GeForce GTX 480*
Najlepší výsledok (multi-GPU): 7189b, *GeForce GTX 480 SLI*

6.3.5 Výsledky vo výpočtovom teste

Najviac rozporuplné výsledky sa ukázali v tomto teste. Podľa očakávania síce prvé miesta zaujali DirectX 11 karty od nVidie, no zarážajúci je ale hlboký prepad nových ATI akcelerátorov až pod úroveň DirectX 10 akcelerátorov od nVidie. Pritom DirectX 11 ATI karty by mali byť omnoho lepšie vybavené aj optimalizované na výpočty pomocou OpenCL. Aj počet výpočtových jednotiek (20 v *Radeonu HD 5870* oproti 15 v *GeForce GTX 480*) , naznačuje, že tieto karty mali zaujať prvé miesta v tomto teste. Príčinou môže byť nevhodne zvolený výpočtový proces, prítomnosť len jedného výpočtového testu, prípadne chyby v OpenCL ovládači od ATI.

Aj napriek kontroverzným výslekom je jasne vidieť, že aj obyčajný akcelerátor strednej triedy (napr. *GeForce 9600GT*) dokáže vo výpočtovom výkone prekonať aj najvýkonnejšie procesory. Pre porovnanie, jeden z najvýkonnejších štvorjadrových procesorov, Intel Core i7, dosahuje **teoretického** výkonu okolo 60 GFLOPS/s. Čip GeForce 9600 GT dosahuje **praktického** výkonu takmer 80 GFLOPS/s, a to za výrazne nižšiu cenu. Najvýkonnejšie karty dosiahli v tomto teste výkonu viac ako 200 GFLOPS/s. Teoretický výkon dvoch GPU by mal byť ešte vyšší, no v tomto teste sa neprejavilo multi-GPU nastavenie.

Najlepší výsledok (single-GPU): 217,3 GFLOPS/s, *GeForce GTX 480*
Najlepší výsledok (multi-GPU): 224,1 GFLOPS/s, *GeForce GTX 480 SLI*

7 Záver

Účelom tejto práce bolo vytvoriť aplikáciu, s ktorou by bolo možné otestovať výkonnosť moderných grafických akcelerátorov pomocou sérií testov zameraných na jednotlivé časti vykresľovacieho procesu. Okrem tohto mala demonštrovať výkonnosť rozličných metód používaných v realistickom zobrazovaní a možnosť porovnávať výsledky medzi rôznymi systémami. A v neposlednom rade išlo aj o snahu ukázať možnosti nových funkcií z rozhrania OpenGL 3.2, ktoré nie sú masovo používané.

Tento účel sa podarilo splniť, pretože k dňu odovzdania práce bolo aplikáciou otestovaných viac ako stovka rôznych systémových konfigurácií s rôznymi grafickými akcelerátormi. To sa dá považovať za slušný úspech, pretože vývoj aplikácie nebol vôbec bezproblémový a ustavične bolo potrebné riešiť problémy so stabilitou aplikácie a chybami zobrazenia na rôznych testovacích platformách. Tieto problémy nakoniec viedli k úplnému vyradeniu geometry shader testov z hodnotenia – kompromis vykonaný zo snahy zaručiť bezproblémový beh aplikácie na čo najviac systémoch.

On-line databáza výsledkov ukázala výkonnostnú prevahu moderných akcelerátorov s podporou rozhraní DirectX 11 a OpenGL 4.0, kde oproti minulej generácii bolo často vidno viac ako dvojnásobný výkon v niektorých testoch. Niekoľko užívateľov nahralo svoje výsledky aj z multi-GPU konfigurácií, kde sa dalo u zopár výsledkov pozorovať dramatický nárast výkonu oproti jedinej karte. Čo sa týka rozšírenosti jednotlivých kariet, v najväčšej obľube boli karty novej generácie od firmy nVidia. To ale vyplýva z faktu, že veľa užívateľov programu boli technicky zdatní entuziasti, ktorí investujú nemalé prostriedky do vylepšovania svojho PC. Čo je však sklamaním, je malý počet výsledkov nahratých pod operačným systémom Linux, takže sme boli pripravení o možnosť podrobnejšie porovnať výkon pod touto platformou a platformou Windows.

Porovnaním výsledkov z mojej aplikácie s komerčnými benchmarkami vidno, že zostavený rebríček výkonnosti grafických kariet približne zodpovedá výsledkom v komerčných aplikáciách. Nie je však rovnaký, pretože iné benchmarky sú často postavené na rozhraní DirectX a orientované na moderné 3D hry, na ktoré výrobcovia grafických kariet kladú väčší dôraz pri optimalizácii svojich ovládačov. A pretože takmer všetky namerané výsledky pochádzajú od „herných“ grafických kariet, tento rozdiel sa ešte viac zvýrazní. Navyše do celkového skóre zaraďujem aj výsledky z OpenCL výpočtového testu, ktorý podával kontroverzné výsledky, kde niektoré grafické procesory výkonovo zaostali za očakávaním.

Aj keď z tejto vzorky výsledkov sa nedajú príliš robiť seriózne závery, dá sa určiť, ktoré grafické akcelerátory sa najviac hodia na špecifické operácie z oblasti počítačovej grafiky. To by som považoval za najväčší prínos aplikácie, pretože na základe nameraných výsledkov nám umožňuje vybrať správny akcelerátor na realizáciu zvolenej úlohy. Tou môže byť napríklad 3D modelovací nástroj s vytvorenou rozsiahlou architektonickou scénou, nástroj na konverziu videa pomocou grafického procesoru alebo jednoducho len najnovšia počítačová hra s pokročilou grafikou a fyzikou.

Na aplikácii by sa dalo mnohé vylepšiť, určite by pomohlo lepšie testovanie a zmena dôležitosti niektorých testov, ktoré negatívne ovplyvňovali výsledky. Takisto užívateľská základňa bola dosť jednoliata, takže do budúcna by bola potrebná väčšia propagácia aplikácie. A keďže vývoj v počítačovej grafike je veľmi rýchly, bude potrebné do grafického enginu stále integrovať najnovšie technológie (aktuálne z rozhrania OpenGL 4.0) a porovnávať výkon pri zobrazovaní náročných, realistických scén využívajúce všetky dostupné technológie používané v real-time zobrazení.

Literatúra

[1] Rost, R.J.: *OpenGL Shading Language*, Second Edition. 2006, 800 s, ISBN 0321334892

[2] Nguyen, H.: *GPU Gems 3*. 2007, 1008 s, ISBN-10: 0321515269

[3] Segal, M., Akeley K.: *The OpenGL® Graphics System: A Specification (Version 3.2 (Compatibility Profile))* [online]. 24.7.2009. Dostupný z WWW: http://www.opengl.org/registry/doc/glspec32.compatibility.20090803.pdf (máj 2010)

[4] Kessenich, J.: *The OpenGL® Shading Language: Language Version: 1.50* [online]. 24.7.2009. Dostupný z WWW: http://www.opengl.org/registry/doc/GLSLangSpec.1.50.09.pdf (máj 2010)

[5] Microsoft Developer Network: Pipeline Stages (Direct3D 10) [online]. 2009. Dostupný z WWW: http://msdn.microsoft.com/en-us/library/bb205123(v=VS.85).aspx (máj 2010)

[6] Huspeka M.: Přehled desktopových grafických čipů [online]. 2.12.2009. Dostupný z WWW: http://www.svethardware.cz/art_doc-D84BE30774AB8A06C12574240036B8F0.html (máj 2010)

[7] Rákos, D.: A brief preview of the new features introduced by OpenGL 3.3 and 4.0 [online]. 15.3.2010. Dostupný z WWW: http://rastergrid.com/blog/2010/03/a-brief-preview-of-the-new-features-introduced-by-opengl-3-3-and-4-0/ (máj 2010)

[8] Futuremark Inc.: *3D Mark Vantage Whitepaper* [online]. 5.9.2008. Dostupný z WWW: http://www.futuremark.com/pressroom/companypdfs/3DMark_Vantage_Whitepaper_v100_Rev2.pdf (máj 2010)

[9] Standard Performance Evaluation Corporation.: SPECviewperf 10 Viewset Descriptions [online]. 19.6.2007. Dostupný z WWW: http://www.spec.org/gwpg/gpc.static/vp10info.html (máj 2010)

[10] Mittring, M. : *Finding Next Gen – CryEngine2,* SIGGRAPH 2007 [online]. 2007. Dostupný z WWW: http://ati.amd.com/developer/SIGGRAPH07/Chapter8-Mittring-Finding_NextGen_CryEngine2.pdf (máj 2010)

[11] Kyprianidis, J.E. : lib3ds [online]. 2010 Dostupný z WWW: http://code.google.com/p/lib3ds/ (máj 2010)

[12] Pipho, E., Turek, M.: Nahrávání komprimovaných i nekomprimovaných obrázků TGA [online]. 2008. Dostupný z WWW: http://nehe.ceske-hry.cz/tut_33.php (máj 2010)

[13] D'Agata G., Turek, M.: 2D fonty z textur [online]. 2008. Dostupný z WWW: http://nehe.ceske-hry.cz/tut_17.php (máj 2010)

[14] Houlmann, F., Metz, S. : *High Dynamic Range Rendering in OpenGL* [online] 11.9.2006 Dostupný z WWW: http://transporter-game.googlecode.com/files/HDRRenderingInOpenGL.pdf (máj 2010)

[15] Elias, H. : Perlin Noise, [online] 22.11.2003. Dostupný z WWW: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm (máj 2010)

[16] Tatarchuk, N. : *Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows*, Symposium on Interactive 3D Graphic and Games [online] 2006. Dostupný z WWW: http://ati.amd.com/developer/i3d2006/I3D2006-Tatarchuk-POM.pdf (máj 2010)

[17] Bailey, M.: *GLSL geometry shaders*, Oregon State University, [online] 10.5.2009. Dostupný z WWW: http://web.engr.oregonstate.edu/~mjb/cs519/Handouts/geometry_shader.pdf (máj 2010)

[18] Game Rendering: SSAO implementation, [online] 2009.
Dostupný z WWW: http://www.gamerendering.com/2009/01/14/ssao/ (máj 2010)
[19] Paul's Projects: OpenGL Shadow Mapping tutorial, [online] 2007.
Dostupný z WWW: http://www.paulsprojects.net/tutorials/smt/smt.html (máj 2010)
[20] Gohara, D. : OpenCL Video Tutorials, [online] 26.8.2009.
Dostupný z WWW: http://www.macresearch.org/opencl (máj 2010)

Prílohy

Príloha 1. Návod na spustenie a ovládanie aplikácie
Príloha 2. CD so zdrojovými kódmi aplikácie, súborom README, Doxygen dokumentáciou a touto správou

Príloha 1 – návod na spustenie a ovládanie aplikácie

Nie je potrebná žiadna inštalácia programu, stačí prekopírovať súbory z CD na disk. **Pozor:** aplikáciu je možné spustiť aj priamo z CD, ale potom nebude možné ukladanie výsledkov vzhľadom k nemožnosti zapisovať na CD. Ďalší postup závisí na operačnom systéme:

- **Windows:** aplikácia sa spustí súborom *gluxMark2.exe* (v hlavnom adresári). Ak by spustenie zlyhalo z dôvodu chýbajúcich .dll, treba doinštalovať Visual Studio Redistributable Package (nachádza sa na CD, zložka `/install-libs`).
- **Linux:** spustí sa príkazom *./gluxMark2*. Pre beh sú potrebné knižnice uvedené v ďalšej sekcii.

Preklad zo zdrojových kódov

Okrem predpripravených binárnych súborov je k dispozícii aj zdrojový kód. Keďže je aplikácia multiplatformná, je možné ju skompilovať pod OS Windows aj Linux.

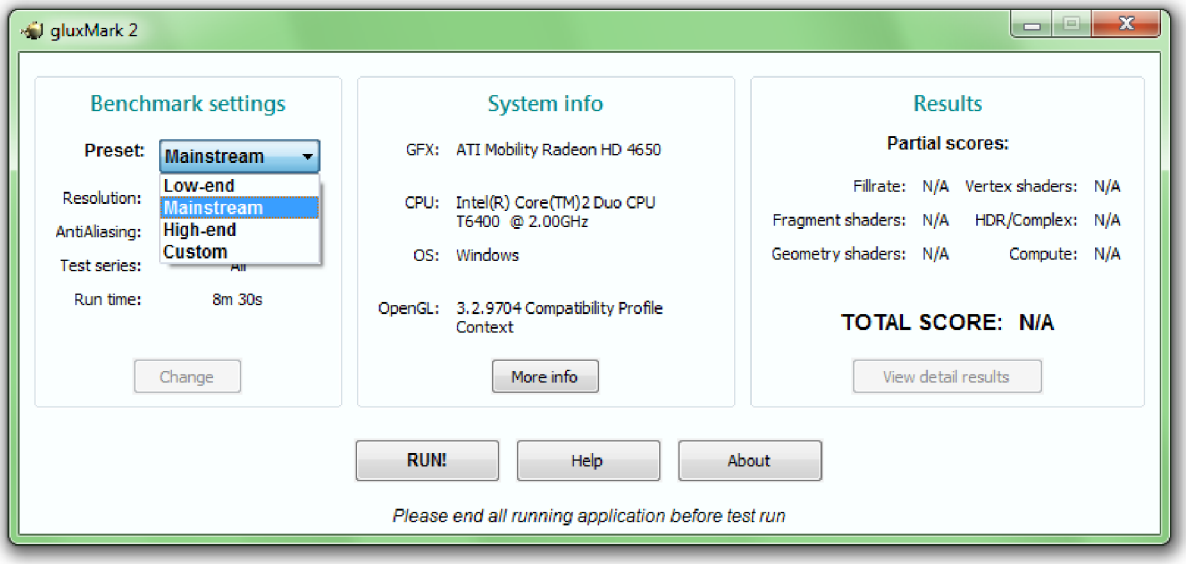
- **Windows:** v archíve sú projektové súbory pre Visual Studio 2008, spolu aj s použitými knižnicami (skompilované ako 32-bitové).
- **Linux:** pripravený Makefile pre preklad, je ale potrebné doinštalovať tieto knižnice:
  - vývojové knižnice pre OpenGL a GLU (nie sú zahrnuté)
  - vývojové knižnice pre wxWidgets 2.8 (nie sú zahrnuté)
  - SDL v1.3 (na CD, zložka `/install-libs`)
  - lib3DS v2.0 (na CD, `/install-libs`)
  - GLEW v1.5.3 (na CD, `/install-libs`)

Požiadavky na systém

- **Procesor:** aspoň 1.6 GHz (úplné minimum je Intel Atom)
- **Pamäť:** najmenej 512MB (záleží od systému, pre Win Vista/7 treba viac ako 1GB)
- **Miesto na disku:** 50MB
- **Grafická karta:**
  - Minimum: karta s podporou OpenGL 2.1 a shader modelu 3.0 (inak povedané, s HW podporou DirectX 9.0c), čiže *GeForce 7* alebo *Radeon X1000* s aspoň 256MB video RAM
  - Doporučené: karta s podporu OpenGL 3.2 a shader modelom 4.0 (DirectX 10), čiže *GeForce 8,9,2xx,4xx* a *Radeon HD2xxx, HD3xxx, HD4xxx, HD5xxx* s aspoň 512MB video RAM
- **Operačný systém:** Windows(XP,Vista,7), Linux (testované na Ubuntu , malo by fungovať všade kde je hardwarová akcelerácia OpenGL a prítomné potrebné knižnice)
- **Najnovšie ovládače grafickej karty s podporou OpenGL vo verzii 3.2** (POZOR: ak chcete spustiť OpenCL testy aj na ATI akcelerátoroch, treba ešte nainštalovať rozhranie ATI Stream)

Návod na použitie - grafické rozhranie

Na spustenie testu slúži grafické rozhranie, kde sa dajú nastaviť základné parametre programu:



Základom je zvolenie výkonnostného nastavenia podľa výkonnosti počítača. Sú pripravené tri nastavenia (Presets):

- **Low-End**: 1024x768, 0xAA - vhodné pre menej výkonné stroje a notebooky
- **Mainstream**: 1280x720, 4xAA - stredne výkonné stroje
- **High-End**: 1920x1080, 8xAA - pre výkonné stroje s FullHD monitorom

Okrem predpripravených nastavení je možné zvoliť aj vlastné (***Preset->Custom*** a potom ***Change***), avšak takéto výsledky nie sú akceptované v on-line databázi. Je možné meniť rozlíšenie, beh v okne/fullscreen, úroveň vyhladzovania hrán a výber testovej série.

Pred samotným testovaním je potrebné vypnúť všetky programy, ktoré ovplyvňujú celkový výkon počítača (browsery, rôzne IM programy, hry alebo náročné grafické aplikácie). Pre maximálny výkon je vhodné pred spustením testu reštartovať počítač.

Po dodržaní týchto pokynov potom stačí kliknúť na **RUN**. Ak aplikácia beží, je možné ju kedykoľvek ukončiť pomocou **Escape**. Po úspešnom skončení testov je možné si zobraziť výsledky tlačidlom **View detail results** v podobe HTML stránky, na ktorej je potom odkaz na nahratie výsledkov do on-line databázy.

Aplikácia zobrazuje aj informácie o systéme (hlavne o grafickej karte), detailnejšie informácie zobrazíte tlačidlom **More info**. Posledné dve tlačidlá, **Help** a **About** Vás nasmerujú na domovskú stránku aplikácie, resp. zobrazia základné info o aplikácii.

Návod na použitie - príkazový riadok

Aplikáciu je možné ovládať aj v príkazovom riadku (treba sa nastaviť na aktuálny adresár s programom).

Použitie: `bin/bench.exe [-w|-f resX resY][-aa value]`

Parametry:

- `-w, -f`: mód obrazovky (beh v okne/fullscreen)
- `resX, resY`: rozlíšenie obrazovky v pixeloch
- `-aa`: sila antialiasingu (0 až 8, u niektorých kariet (napr. GeForce GTX 480) až po 32)
- `-t`: testova séria ("all", "fillrate", "fragment","geometry", "vertex", "complex", "compute", "info"), prípadne číslo jednotlivého testu (od 0-20, číslované v poradí v akom testy nasledujú)
- `- dev`: developer mode. V tomto nastavení je možné sa pohybovať po každej scéne myšou:
  - ľavé tlačítko: rotácia okolo počiatku scény
  - stredné tlačítko: horizontálny a vertikálny pohyb kamery
  - pravé tlačítko: zoom

  Ďalej je v tomto móde možné preskakovať jednotlivé testy klávesou medzera ako aj prepnúť na drôtené zobrazenie klávesou W. V tomto móde sa z pochopiteľných príčin nepočíta skóre.

Napríklad príkaz `bin/bench.exe -t fragment -w 1024 768 -aa 4`
spustí fragment shader testy v okne, v rozlíšení 1024x768 a 4x antialisingom. Pri tomto behu sa potom do príkazového riadku vypisuje aj log (čo všetko sa inicializuje, nahráva a prípadne chyby), podobne ako aj čiastkové skore.

Všetky tieto informácie a mnoho ďalších je možné nájsť aj na stránkach aplikácie:
http://www.stud.fit.vutbr.cz/~xvanek29/gluxmark2