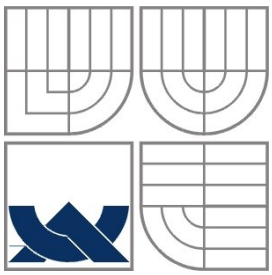
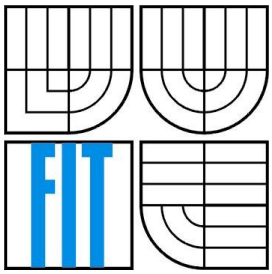


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JURAJ VANEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2008

# Zadanie

1. Seznamte se s možnostmi a vlastnostmi současných grafických akcelerátorů.
2. Prostudujte problematiku profilování programů a měření výkonnosti.
3. Prostudujte existující řešení měřící výkonnost grafických akcelerátorů.
4. Identifikujte výkonově kritické operace grafických akcelerátorů.
5. Navrhněte metodiku měření výkonnosti grafických akcelerátorů.
6. Implementujte navržené měření.
7. Interpretujte naměřené výsledky.
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

# Licenčná zmluva

Originál licenčnej zmluvy je uložený v archíve Fakulty informačních technologií Vysokého učení technického v Brně.

# Abstrakt

Námetom mojej práce je meranie výkonnosti grafického akcelerátoru pri zobrazovaní 3D scén. Zameriava sa na otestovanie výkonu akcelerátoru z každého uhla pohľadu pri výkonovo kritických operáciách v 3D grafike. Pojednáva o problematike implementácie jednotlivých meraní, ich vplyve na celkovom výsledku a vyhodnotení meraní. Konečným výstupom aplikácie je skóre, podľa ktorého je možné usúdiť, aký výkon podáva akcelerátor v porovnaní s ostatnými. Na toto porovnanie slúži on-line databáza výsledkov.

# Kľúčové slová

3D, test, OpenGL, grafická, karta, akcelerátor, porovnanie, skóre

# Abstract

My work is about performance evaluation of graphics accelerator by 3D scenes rendering. It focuses on how to test the performance of the accelerator from every point of view during the performance critical operations in 3D graphics. It deals with implementation problems of individual measures, their influence on total results and evaluation of the measurements. The final output of the application is a score, by which is possible to judge what performance is giving the accelerator in comparison with others. For this comparison is available an on-line results database.

# Keywords

3D, benchmark, OpenGL, graphics, card, accelerator, comparison, score

# Citácia

Juraj Vanek: Měření výkonnosti grafického akcelerátoru, bakalářská práce, Brno, FIT VUT v Brně, 2008

# Měření výkonnosti grafického akcelerátoru

## Vyhlásenie

Vyhlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Adama Herouta, Ph.D., pričom ďalšie informácie a poznatky som čerpal z uvedených literárnych prameňov a publikácií.

.....
Juraj Vanek
8.5.2008

## Poďakovanie

Chcel by som poďakovať Ing. Adamovi Heroutovi, Ph.D. za cenné rady a inšpirácie, ktoré mi veľmi pomohli pri tvorbe tejto práce. Ďalej by som chcel poďakovať všetkým, ktorí boli ochotní pomôcť pri zbere testovacích výsledkov z aplikácie

# Obsah

# 1 Úvod

V poslednej dobe môžeme sledovať prudký rozvoj 3D grafických aplikácií bežiacich v reálnom čase. Nie sú to len moderné 3D hry, ale aj profesionálne aplikácie, pričom kvalita výstupu sa neustále zvyšuje a dnes už skutočne nemá ďaleko od fotorealistického zobrazenia. S týmto pokrokom je úzko spätý výrazný technologický pokrok v oblasti grafických akcelerátorov. V súčasnosti majú najvýkonnejšie grafické akcelerátory teoretický výkon rádovo desaťnásobne väčší, ako klasické univerzálne procesory. Z toho nám vychádza potreba, ako medzi sebou objektívne porovnávať výkon jednotlivých grafických akcelerátorov.

Existuje pomerne veľa aplikácií, zameraných na testovanie výkonnosti grafickej karty v oblasti zobrazovania 3D scén. Medzi najpopulárnejšie patria programy typu „3D Mark“, ktoré merajú výkon v scénach podobajúcich sa na moderné počítačové hry. Súčasne slúžia aj ako demonštrácia nových technológií, ktoré sa v praxi objavujú v nasledujúcich rokoch. V profesionálnej sfére sa využívajú aplikácie, ktoré merajú výkon v reálnych CAD/CAM aplikáciách. Medzi ne patrí napríklad séria testovacích aplikácií SPEC, ktoré zisťujú výkon po rozhraním OpenGL v programoch ako 3D Studio a Maya.

Táto práca začína rozborom technického vybavenia a možností dnešných grafických akcelerátorov. Nasleduje stručný prehľad existujúcich riešení zameraných na testovanie výkonu a metodiky, ktoré používajú. Ďalej popisujem implementáciu mojej aplikácie, ktorá bola navrhnutá ako multiplatformný test s grafickým rozhraním zameraným na otestovanie grafickej karty po všetkých stránkach, interpretáciou výsledkov a on-line databázou výsledkov. Každú sériu navrhnutých testov popisujem v samostatnej kapitole. V závere je zhodnotenie výsledkov meraní, ako aj námety pre ďalšie pokračovanie projektu.

# 2 Vlastnosti grafických akcelerátorov

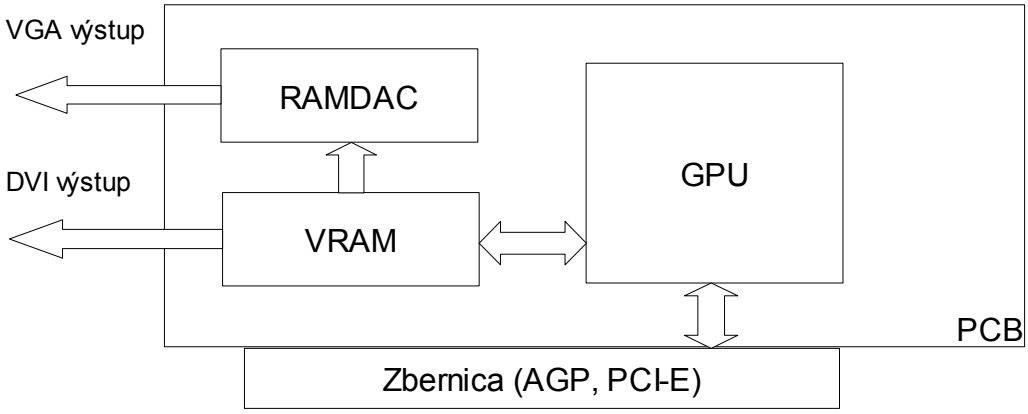
Grafický akcelerátor je samostatná jednotka v počítači, ktorá sa stará o generovanie obrazu. Názov akcelerátor je odvodený od toho, že je schopný urýchľovať (akcelerovať) náročné obrazové výpočty priamo vo svojom hardware. Do systému sa pripájajú cez zbernicu (dnes väčšinou PCI-Express, v minulosti AGP), prípadne sú integrované. V súčasnosti používané grafické karty majú všetky prítomnú 3D akceleráciu, takže v tejto kapitole sa zameriam len na moderné 3D akcelerátory. V dobe písania tejto správy (jar 2008) by sa heslovito dali vlastnosti najvýkonnejších akcelerátorov opísať nasledovne [8] [9]:

- počet tranzistorov v jadre dosahuje takmer pol miliardy
- takty jadra pohybujúce sa nad 600MHz
- masívna paralelizácia
- unifikovaná architektúra (stream processors)
- celkový teoretický výkon takmer 1 TFLOPS
- viacjadrové grafické procesory
- grafická pamäť o veľkosti viac ako 512MB
- pamäte typu GDDR4, frekvencie nad 2GHz
- podpora grafických rozhraní DirectX 10.1 a OpenGL 2.1

Súčasné grafické procesory sú teda nesmierne zložité elektronické obvody, z čoho vyplýva ich vysoká energetická náročnosť (až 150W) a potreba chladenia. V oblasti zložitosti a výkonu už dávno prekonali klasické x86 procesory.

## 2.1 Súčasti grafickej karty

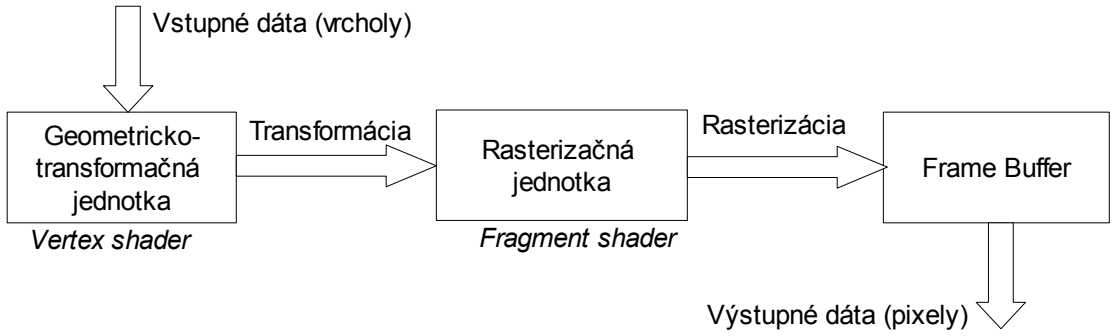
Grafické karty dneška môžeme rozdeliť na dve subkategórie: **integrované** a **samostatné**. Integrované sú umiestnené priamo na základovej doske počítača, nemajú vlastnú grafickú pamäť, ale zdieľajú systémovú. Sú najpomalšie, ale s nízkymi výrobnými nákladmi, preto sa využívajú v lacnejších PC zostavách a notebookoch, prípadne kde nie je potrebný vysoký grafický výkon (kancelárske počítače). Naproti tomu samostatné karty využívajú vlastnú dosku spojov (PCB), na ktorom je umiestnený grafický procesor (GPU) a grafická pamäť (VRAM).




Obrázok 2.1: Zjednodušená schéma samostatnej grafickej karty [8]

### 2.1.1 Grafický procesor (GPU)

GPU je hlavné výpočtové centrum grafickej karty, od ktorého najviac závisí celkový výkon. Je optimalizovaný na prácu s číslami v pohyblivej radovej čiarke a obsahuje v sebe hardwarovú implementáciu algoritmov používaných v počítačovej grafike. Súčasné grafické procesory obsahujú viacero výpočtových jednotiek (pipelines), ktoré im umožňujú vykonávať operácie paralelne. Funkcie grafického procesoru popisuje nasledujúci obrázok [8]:




Obrázok 2.2: Zjednodušený proces vytvárania obrazu v GPU

Dáta do grafického procesoru vstupujú v podobe vrcholov (vertexov). Tieto vrcholy sú pomocou **geometricko-transformačnej** jednotky osvetlené, transformované, orezané a putujú ďalej do **rasterizačnej jednotky.** Tam sa objekty rasterizujú, nanášajú textúry a počíta sa hĺbková hodnota pixelov (z-buffer). Takto vytvorené fragmenty obrazu putujú do **obrazového bufferu** (frame buffer), v ktorom sa vytvára výsledný obraz.

V dnešných GPU sú fixné geometricko-transformačné a rasterizačné jednotky (T&L, Transform & Lighting) nahradené **programovateľnými tieňovacími jednotkami** (ďalej „shadery“). Na transformáciu vrcholov sa používa tieňovacia jednotka vrcholov (ďalej „**vertex shader**“), ktorá predpripraví dáta pre tieňovaciu jednotku fragmentov (ďalej „**fragment shader**“), ktorá pracuje

s každým fragmentom scény. Výhodou tohto postupu je možnosť napísať si vlastný transformačný a svetelný model pomocou krátkeho programu (shader program) a obísť tak pevnú funkcionalitu klasických T&L jednotiek. Tak sa dajú naprogramovať efekty ako osvetľovací model pre každý pixel scény, mapovanie nerovností (bump mapping), zložitá animácia a morphing objektov, dynamické mäkké tiene atď [2].

Aktuálne sa do popredia dostáva unifikovaná architektúra, ktorá zjednocuje fragment a vertex shadery a umožňuje tak efektívnejšie spracovanie dát. Predstavme si napríklad situáciu, v ktorej má GPU spracovávať veľmi zložitú geometriu, avšak bez textúr a len s jednoduchým osvetľovacím modelom. Zaťaženie vertex shaderu je veľmi vysoké, zatiaľ čo fragment shader je prakticky nevyužitý. Unifikované jednotky tento problém riešia tak, že fungujú buď ako fragment alebo vertex shadery. V našej modelovej situácii by teda väčšina unifikovaných jednotiek fungovalo ako vertex shader a zvyšok ako fragment shader [9].

### 2.1.2 Grafická pamäť (VRAM)

Grafická pamäť slúži na ukladanie obrazových dát (frame buffer, z-buffer, vertexy, textúry), ktoré sú pomerne náročné na priestor. Samostatné karty využívajú vlastnú VRAM o veľkej kapacite (až 512MB), ktorá je veľmi rýchla (v súčasnosti využívané pamäti GDDR4 dosahujú frekvencií až 2GHz, vo výhľade sú pamäti GDDR5 s frekvenciou ešte vyššou). U lacnejších grafických kariet sa využívajú rôzne technológie, pri ktorých sa okrem grafickej pamäte menšej veľkosti využíva aj systémová pamäť (HyperMemory, TurboCache) [9].

### 2.1.3 Ostatné súčasti

K potrebným súčastiam patria obvody pre generovanie signálu pre monitor, ktoré závisia od typu rozhrania. Pri digitálnych (DVI, Display Port) je možné posielať dáta priamo, u analógových rozhraní (VGA, D-SUB) sa musia dáta previesť na analógový signál. Toto má na starosti RAMDAC prevodník, ktorý pracuje väčšinou na frekvenciách okolo 400MHz a generuje signál pre monitor. Súčasné grafické karty sú schopné vykresliť obraz pri rozlíšeniach až 2560x1600 pixelov s obnovovacou frekvenciou 75Hz.

Ďalšími voliteľnými súčasťami grafickej karty môžu byť rôzne podporné obvody a čipy, spomeniem napríklad čipy pre dekódovanie videa, vytváranie signálu pre televízny prijímač (TV OUT) , pre prijímanie signálu z analógových zariadení ako videokamera (TV IN, spoločne nazývané aj VIVO) alebo integrovaný TV tuner na príjem televízneho signálu. Podrobnejšie sa o ďalších súčastiach píše na [9].

# 3 Prehľad existujúcich testovacích aplikácií (benchmarkov)

V predchádzajúcej kapitole sme si spomenuli, aké sú vlastnosti a možnosti dnešných grafických akcelerátorov. Je zrejmé, že dobrá testová aplikácia by mala obsahovať testy, ktoré preveria každú súčasť grafickej karty a bude čo najviac nezávislá na zvyšku systému (hlavne procesore). Ešte pred samotným predstavením vlastného riešenia aplikácie si urobíme stručný prehľad existujúcich aplikácií na testovanie výkonu grafického akcelerátoru.

## 3.1 Herné benchmarky

Tieto testy sú zamerané primárne na skupinu hráčov počítačových hier, kde zisťujú výkon grafickej karty v scénach, ktoré pripomínajú populárne 3D hry (syntetické testy), alebo priamo v reálnych hrách, pomocou interných či externých testovacích nástrojov.

Medzi najznámejšie patria aplikácie typu **3D Mark**. Patrí sem rodina syntetických testov, ktoré vyvíja fínska spoločnosť Futuremark. Získali si obrovskú popularitu, pretože ako prvé prinášali možnosť jednoduchého testovania výkonu pri technológiách, ktoré vtedy neboli ešte veľmi rozšírené:

- T&L a DirectX 7 : 3D Mark 2000
- Pixel/Vertex shadery a DirectX 8: 3D Mark 2001
- DirectX 9: 3D Mark 03 – 06
- DirectX 10: 3D Mark Vantage

Metodika merania je taká, že sú pripravené (najčastejšie 4) veľmi komplexné scény, ktoré simulujú moderné náročné 3D hry. Každá scéna trvá niekoľko minút a meria sa pri nich výkon v snímkoch za sekundu (FPS). Z týchto scén sa potom počíta celkové skóre, pričom vyššiu dôležitosť majú testy, ktoré využívajú viac nových technológií. Ďalej nasledujú testy zamerané na teoretický výkon karty (rýchlosť vyplňovania, vykresľovania polygónov) doplnené o testy zamerané na nové technológie. K porovnávaniu výsledkov slúži on-line databáza výsledkov (ORB – On-line Result Browser), v ktorej sú v súčasnosti výsledky tisícov užívateľov [11]. Nevýhodou týchto testov je, že výkon v skutočných hrách môže byť odlišný, keďže ide o syntetické testy (čo sa týka hlavne novších verzií). Ďalšou nevýhodou je ich viazanosť na rozhranie DirectX, čo umožňuje beh len v prostredí OS Microsoft Windows.

Z ostatných syntetických aplikácií, ktoré merajú komplexný výkon grafického akcelerátoru, môžem spomenúť ešte **GLExcess**, ktorý vznikol z pôvodne diplomovej práce a stal sa dosť uznávaným benchmarkom pre rozhranie OpenGL, avšak v súčasnosti mu chýba podpora najnovších technológií a nie je už ďalej vyvíjaný. Ostatné aplikácie sú viac-menej zamerané len na jednu výkonovú oblasť (napríklad aplikácie zamerané čisto na výkon shader jednotiek).

Medzi ďalšiu skupinu testov patria testy **priamo v 3D hrách**, buď pomocou vstavaného testového módu, alebo cez programy schopné merať výkon zobrazenia v snímkoch za sekundu (napr. FRAPS). Najčastejšie sa meria v hrách, ktoré boli svojim spôsobom prelomové svojou grafikou, alebo patria medzi najrozšírenejšie (spomeniem napríklad Crysis, Quake4, Unreal Tournament a pod.). Porovnávanie je už oveľa presnejšie, pretože ide o výkon v skutočnej 3D aplikácii. Nevýhodou je nutnosť zaobstarať si hru a nižšie pohodlie pri meraní (väčšinou ide o meranie len v príkazovom riadku, prípadne využitie skriptov).

## 3.2 Profesionálne benchmarky

Patria medzi pomerne dôležitú skupinu aplikácií, pretože merajú výkon v profesionálnych 3D aplikáciách ako sú rôzne CAD/CAM systémy a 3D modelovacie nástroje ako 3D Studio, Cinema4D, Maya. V neposlednom rade dnes pomaly vznikajú aplikácie, ktoré dokážu využiť výkon GPU na všeobecné výpočty, napríklad fyzikálne simulácie, takže vznikajú benchmarky zamerané na výpočtovú silu GPU.

Medzi najznámejšie a najviac používané patria:

- **SPECviewperf -** ide o sériu testov od neziskovej organizácie SPEC, 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 (ktoré na rozdiel od herných aplikácií má v profesionálnych jasnú prevahu) a 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 [12].
- **CineBench –** 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 [13].

Ďalšou možnosťou (podobne ako u herných benchmarkov) je možnosť merať výkon priamo v zvolenej aplikácii s použitím príslušného externého nástroja na meranie výkonu.

# 4 Testová aplikácia

Moja testová aplikácia (***„gluxMark“***) funguje na podobnom princípe ako ostatné (predchádzajúca kapitola). Implementuje niekoľko sérií testov, z ktorých každá sa zameriava na určitú výkonovo kritickú operáciu 3D grafiky pod rozhraním OpenGL. Výstupom aplikácie je súbor s výsledkami, ako aj celkové skóre, pomocou ktorého je možné porovnať výkon medzi rôznymi kartami. Ide o čisto syntetický benchmark, jeho výsledky teda nemusia zodpovedať skutočným výsledkom grafických kariet v reálnych aplikáciách (ako sú hry a 3D editory), viacej sa zaoberá teoretickým výkonom karty. Aplikácia bola vytvorená tak, aby bola pokiaľ možno čo najviac nezávislá na výkonu procesoru a ostatných súčastí systému, ako aj na použitých ovládačoch a operačnom systéme.

Návrh aplikácie bol vytvorený podľa metód objektovo orientovaného programovania, ktoré poskytuje neporovnateľne vyšší komfort pri programovaní než klasické procedurálne programovanie. Implementácia prebiehala v jazyku **C++.** Program sa skladá z modulov, čiže umožňuje postupné rozširovanie o nové funkcie. Celkový prehľad jednotlivých modulov, tried a funkcií nájdete v prílohe 2.

Veľký dôraz bol kladený na **prenositeľnosť** aplikácie pod rôznymi operačnými systémami, preto bol použitý platformovo nezávislý toolkit OpenGL GLUT [4] na grafické zobrazenie a vizuálny toolkit wxWidgets [5] na implementáciu užívateľského rozhrania. Celá aplikácia pozostáva z dvoch programov. Jedným je grafické užívateľské rozhranie, ktoré umožňuje vizuálne nastavovať vlastnosti testov, zobrazuje základné informácie o systéme a spúšťa hlavný testovací program. Druhým je samostatná testová aplikácia, ktorá beží v celoobrazovkovom (fullscreen) režime a vykonáva samotné merania. Po ukončení všetkých testov uloží výsledky do XML súboru, z ktorého potom grafické rozhranie užívateľa informuje o celkovom skóre a výsledkoch.

## 4.1 Výstupy aplikácie

Aplikácia pracuje s XML súbormi pre ukladanie a čítanie údajov. Formát XML som zvolil, pretože umožňuje jednoduché ukladanie údajov a ich vyhľadávanie v strome dokumentu. Umožňuje efektívne spracovanie údajov ako na strane klienta, tak aj na strane serveru (kapitola 6.2). Čiastkové výsledky každého testu sa ukladajú do XML súboru ***results.xml***.

Súbor obsahuje niekoľko hlavných sekcií:

```
<root>          //koreňový uzol XML dokumentu
  <sysinfo>
    <system>    //informácie o systéme
        ...
    <gfx>       //informácie o grafickej karte
        ...
  </sysinfo>
    <settings>  //nastavenia benchmarku
    ...
//nasledujú výsledky testov, formát je spoločný:
<seria_testov>
  <testX>
    <vysledok> 0 </vysledok>
  </testX>
<seria_testov>
//posledná sekcia dokumentu, výsledné skóre
<total_score>0</total_score>
<testX_score>0</testX_score>
```

Tento súbor vzniká po úspešnom zakončení všetkých testov, pričom je uložený do hlavného adresára s programom (prípadne inde ak program beží na súborovom systéme len na čítanie). Súbor sa potom použije na vygenerovanie HTML stránky s výsledkami, prípadne ho užívateľ môže nahrať do databázy výsledkov (kapitola 6.2).

### 4.1.1 Získavanie informácií o systéme

Po spustení grafického rozhrania toto spustí hlavnú aplikáciu s parametrom `-info`. Pri tomto nastavení sa inicializuje OpenGL okno, avšak len za účelom získania informácií hlavne o grafickej karte pomocou rozhrania OpenGL. Výsledky sa uložia do súboru ***sysinfo.xml***, ktorý má takú istú štruktúru, ako vyššie opísaný súbor ***results.xml***.

Získavanie informácií prebieha viacerými spôsobmi. Najjednoduchší je pomocou OpenGL funkcií `glGetString` a `glGetIntegerv` [1]. Pomocou nich je možné zistiť (na základe rôznych parametrov posielaných funkciám) názov grafickej karty, výrobcu, podporovanú verziu OpenGL a rôzne špecifické vlastnosti ako napríklad maximálnu veľkosť textúry, počet premenných shader programu atď.

Informácie o zvyšku systému (procesor, operačná pamäť) sa zisťujú podľa toho, v akom operačnom systéme sa nachádzame:

- **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)` [6].
- **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`.

## 4.2 Načítavanie externých entít do aplikácie

Externými entitami rozumieme dáta (objekty, textúry), ktoré sú do programu načítané z externých súborov. Keďže nie sme priveľmi obmedzení výslednou veľkosťou aplikácie, rozhodol som sa objekty a textúry pripraviť mimo aplikácie a do nej ich potom importovať (nie je potrebné vytvárať objekty a textúry priamo v programe napríklad pomocou procedurálnych generátorov).

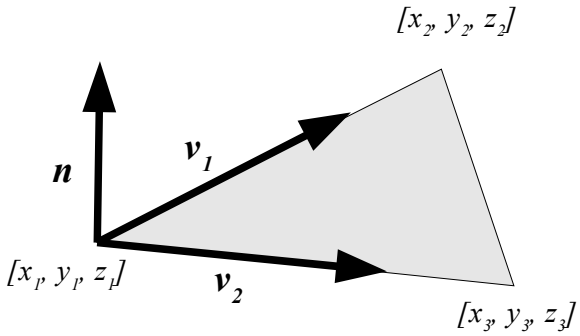
### 4.2.1 3D objekty

3D objekty sú importované zo súboru vo formáte **3DS.** Tento formát je veľmi rozšírený medzi vývojármi, pretože je pomerne jednoduchý, dobre zdokumentovaný a má vysokú podporu medzi 3D modelovacími aplikáciami. Keďže ide o binárny formát, je veľmi efektívny pri ukladaní dát [10].

Základom štruktúry 3DS súboru sú zhluky dát. Každý zhluk je označený postupnosťou bytov, ktoré identifikujú veľkosť a typ dát, ktoré zhluk označuje. Medzi najdôležitejšie zhluky patria **názov súboru, zoznam vrcholov, zoznam textúrových koordinátov** a **zoznam plôch**. Objekt sa rekonštruuje tak, že zoznam plôch tvoria indexy do poľa vrcholov. Z týchto vrcholov sa potom vytvorí jeden polygón objektu. Keď teda prejdeme celý zoznam plôch, máme zrekonštruovaný celý objekt. Ku každému vrcholu je ďalej možné pridať koordinát textúry (index je opäť v zozname plôch).

Čo však súbor neobsahuje, je informácia o povrchovej normále vrcholu. Normály definujú, ktorým smerom je vrchol privrátený k zdroju svetla a s akou intenzitou ho odráža. Z geometrického hľadiska je normála vektor kolmý na dva vektory, ktoré ležia v jednej rovine. Keďže 3DS objekt je tvorený trojuholníkmi, dá sa vypočítať normála povrchu ako vektorový súčin dvoch vektorov

vytvorených z vrcholov trojuholníka, kde $x_n$, $y_n$, $z_n$ sú súradnice vrcholov a $n$ je výsledný normálový vektor:




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

Obrázok 4.1: Výpočet normály ako kolmého vektoru na rovinu

Navyše je potrebné poznať orientáciu polygónu. OpenGL totiž predvolene akceptuje vrcholy zoradené proti smeru hodinových ručičiek, takže je potrebné vygenerovať dáta vrcholov v takomto poradí. Takto vygenerovaná normála sa potom vloží do zoznamu normál na rovnaký index ako mal vrchol (pri rekonštrukcii sa tak ku každému vrcholu priradí zodpovedajúca normála).

Objekt sa vykresľuje vo forme OpenGL display listu, čo výrazne urýchľuje vykresľovanie (objekt sa rekonštruuje z vrcholových dát len jediný raz). Jednotlivé plochy sa vykresľujú ako trojuholníky (`GL_TRIANGLES`), vrcholy sa definujú ako `glVertex3f()`, normály ako `glNormal3f()` a textúrové koordináty pomocou `glTexCoord2f()`. Táto jednoduchá knižnica neobsahuje všetky funkcie na načítanie kompletného 3DS súboru, akými sú definícia viacero objektov, materiálov, zoznam použitých textúr atď. Pre naše účely však postačuje iba rekonštrukcia 3D siete objektu. Podrobný popis formátu sa dá nájsť na [10].

Väčšina 3D objektov použitých v programe, bola vytvorená autorom v 3D editore Cinema4D CE6, ktorý ako starší program je voľne šíriteľný s určitými obmedzeniami. Ostatné objekty sú voľne dostupné „free“ objekty, ktoré je možné pre nekomerčné účely využívať zdarma, prípadne objekty priamo vytvoriteľné pomocou nadstavbových knižníc OpenGL GLUT a GLU (napr. „známa“ čajová kanvica sa dá vykresliť príkazom `glutSolidTeapot()` ) [4].

Pre úsporu miesta sú všetky súbory s 3D dátami navyše skomprimované do archívu ZIP a počas behu sa dynamicky dekomprimujú. Pri tomto procese je využitý objekt `wxZipInputStream` z toolkitu wxWidgets, ktorý dekomprimuje archív a potom je možné súbor ďalej načítať pomocou štandardných C/C++ funkcií pre prácu so súbormi.

### 4.2.2 Textúry

Textúra je typicky 2D obrázok, ktorý sa nanáša na 3D objekt za účelom imitácie povrchovej štruktúry. OpenGL akceptuje textúry vo formáte 24-bit RGB (prípadne 32-bit RGBA), takže je potrebné previesť externé dáta do tohto formátu. Existuje síce formát BMP (Windows bitmap), ktorý obsahuje dáta priamo v tomto formáte, no keďže je nekomprimovaný, je prakticky nepoužiteľný (veľké nároky na priestor). Jeho binárnou formou je formát TGA (Targa), avšak ten v nekomprimovanom stave je tiež pomerne veľký. Rozhodol som sa teda použiť formát JPG, ktorý používa stratovú kompresiu a odvádza dobrú prácu pri kompresii fotografií a textúr. Je však potrebné ho dekomprimovať a previesť dáta na interný OpenGL formát. Na dekompresiu a rozkódovanie formátu používam objekt `wxImage` prítomný v toolkite wxWidgets. Textúra sa z obrazových dát vytvorí pomocou volaní OpenGL funkcií `glGenTextures()`, `glBindTexture()` a `gluBuild2DMipmaps()`. Posledný príkaz dáva tušiť, že z každej textúry sú generované mip-mapy s nižším rozlíšením. Toto riešenie zvyšuje kvalitu obrazu pri vzdialených textúrach, ako aj výkon. Ako najvyššiu možnú kvalitu je možné použiť anizotropné filtrovanie textúr prístupné cez OpenGL rozšírenie `GL_EXT_texture_filter_anisotropic` [1].

Textúry použité v programe sú väčšinou voľne šíriteľné textúry a fotografie, ktoré je možné nájsť na internete. Niektoré obrázky boli dodatočne upravené grafickým editorom GIMP.

# 5 Testy

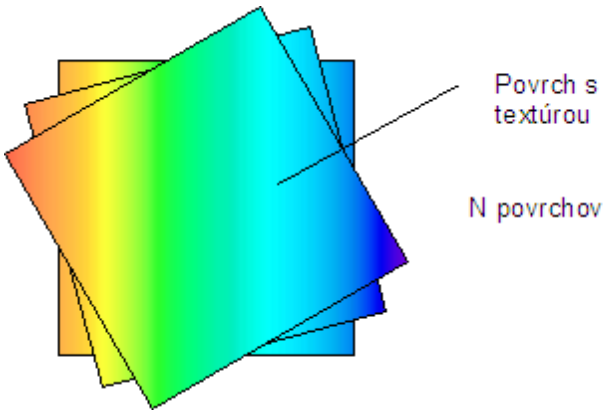
Aplikácia obsahuje niekoľko sérií testov, v ktorých sú podobné testy. Keďže používam synteticky vytvorený jednoduchý grafický engine, nemožno očakávať, že podobné výkony bude podávať akcelerátor v reálnych aplikáciách – ide čisto o teoretický výkon pod rozhraním OpenGL. Všetky testy bežia v režime celej obrazovky, v rozlíšení a farebnej hĺbke, aká bola nastavená na začiatku testu. Spoločnou časťou všetkých testov je informačná obrazovka pred začiatkom testu, kde je názov aktuálneho testu. Počas priebehu sa zobrazuje počet snímkov za sekundu (FPS), pričom na pozadí sa počíta celkový počet snímkov od začiatku behu testu.

Po skončení všetkých testov sa vypočíta celkové skóre. To je súčet snímkov za sekundu všetkých testov krát váha každého testu (kapitola 6.1). Ako referenčná grafická karta slúži *ATI Radeon 9800PRO*, na ktorej prebehol aj vývoj aplikácie. Nasleduje podrobný popis implementovaných testov, ako aj ich názvy uvádzané v programe.

## 5.1 Testy vypĺňovania (fillrate)

Pri týchto testoch sa vyhodnocuje, koľko pixelov dokáže grafická karta vykresliť (rasterizovať) za určitý čas (sekunda). Test prebieha tak, že na jednoduchý povrch sa aplikuje farebný prechod alebo textúra(y) a potom na základe počtu snímkov za sekundu a rozlíšenia sa vypočíta rýchlosť vypĺňovania v miliónoch pixelov za sekundu (Mpixels/s).

Testy majú spoločné rozhranie: na 32 štvorcových povrchov, ktoré ležia za sebou v smere osi z, sa aplikuje buď farebný prechod alebo textúra. Dôvod použitia viacerých povrchov je zvýšenie presnosti merania, pretože pri viac povrchoch sa vykreslí viacej pixelov a znižuje sa závislosť ostatných častí systému, najmä procesoru.



Obrázok 5.1: Umiestenie povrchov v testoch

- **Test1: Pixel Fillrate (Z-buffer off)** – na povrchy sa aplikuje farebný prechod. Keďže z-buffer je v tomto teste vypnutý, zdá sa, akoby povrchy ležali "cez seba". Z hľadiska výkonu ide o to, že karta musí vykresliť každý povrch zvlášť.

- **Test 2: Pixel Fillrate (Z-buffer on) -** vykresľovanie prebieha presne ako u prvého testu, rozdielom je zapnutý z-buffer. Karta teda nemusí vypĺňať všetky časti povrchu, stačí len tie, ktoré nie sú prekryté ostatnými (Obrázok 5.1) To vedie k celkovo vyššiemu výkonu oproti prvému testu.
- **Test 3: Texel Fillrate (Single texturing)** – namiesto farebného prechodu sa nanáša na každý povrch jedna textúra. Všetky povrchy sú potom zmiešané dohromady pomocou blendingu (`glBlendFunc(GL_SRC_ALPHA,GL_ONE, 0.07)`), čo donúti kartu prepočítať každý pixel každého povrchu. Výsledky sú podobné prvému testu, nie však zhodné, pretože aplikácia textúr je náročnejšia operácia ako aplikácia jednoduchej farby. Je využitá len jedna textúrovacia jednotka.
- **Test 4: Texel Fillrate(Multi texturing)** – oproti minulému testu je podstatný rozdiel v tom, že sa využijú všetky dostupné textúrovacie jednotky grafickej karty. Na multitexturing sa využíva OpenGL rozšírenie `GL_ARB_multitexture`, takže karty bez podpory tohto rozšírenia nebudú schopné spustiť tento test. Povrchy sú modifikované pridaním multi textúrových koordinátov, aby bolo možné aplikovať na povrch toľko textúr, ako je len možné. To umožní vykresliť viacej povrchov v jednom priechode, takže čím viac má karta textúrovacích jednotiek, tým vyšší by mal byť výkon oproti predchádzajúcemu testu.

Po skončení každého testu sa vypočíta rýchlosť vyplňovania (fillrate) v megapixeloch za sekundu, pričom *FPS* je počet zobrazených snímkov za sekundu a $resoluton_{x,y}$ je rozlíšenie obrazovky v pixeloch:

$$fillrate_{testX} = \frac{(resolution_x . resolution_y . FPS)}{1000000} MPixels/s$$

Výsledné skóre z tejto série testov sa určí podľa vzorca:

$$skóre = \frac{(test1_{fillrate} + 0{,}5 . test2_{fillrate} + 1{,}5 . test3_{fillrate} + 2{,}5 . test4_{fillrate})}{10}$$

Najvyššiu váhu má test zameraný na multitexturing, pretože ten sa v súčastnosti najviac využíva pri vykresľovaní realistických povrchov (ktoré sú realizované dvomi a viacerými textúrami).

**Zhrnutie série testov:**

- **Účel:** zistiť výkon akcelerátoru v oblasti rasterizácie objektov.
- **Závisí od:** výkonu rasterizačných a textúrovacích jednotiek v GPU.
- **Váha v rámci celej aplikácie: 25%.** Textúry sú používané prakticky v každej 3D aplikácii, preto je veľmi dôležité, aký výkon bude karta podávať pri ich vykresľovaní.
- **Skóre na referenčnej karte:** 1500 bodov.

## 5.2 Polygónové testy

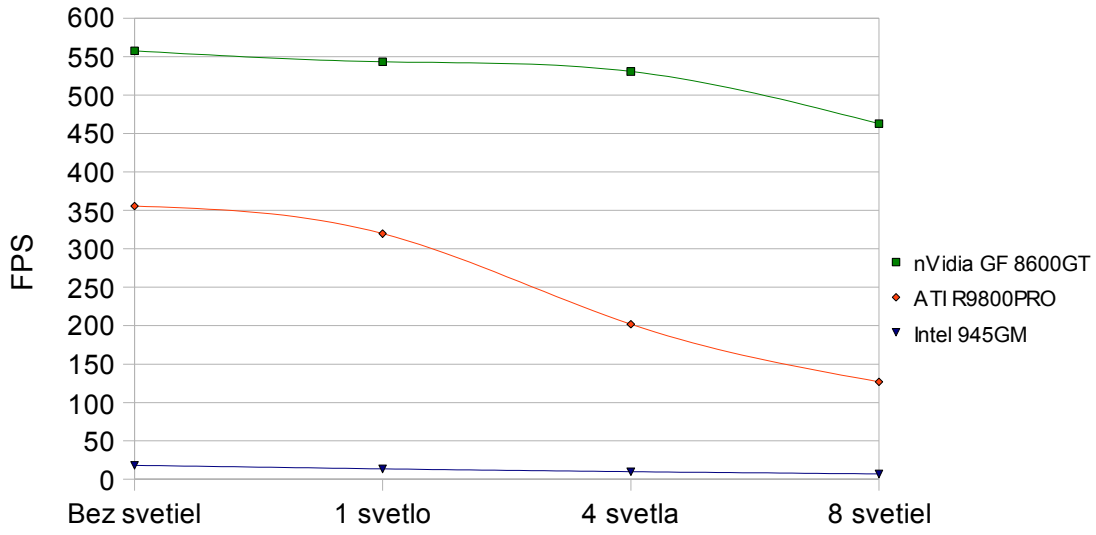
Základom týchto testov sú polygónové modely (zložené v tomto prípade z trojuholníkov), ktoré sa načítajú z externého 3DS súboru (kapitola 4.2.1) pomocou triedy `Object_3DS` a jej metód. Každý test pozostáva zo štyroch menších častí. V prvej je vykreslený objekt bez osvetlenia, v ďalších sa postupne zvyšuje počet svetiel od 1 cez 4 až po maximálny počet 8:

Obrázok 5.2: Polygónový model použitý v 1.teste pri rôznom osvetlení (od 0 svetiel až po 8 svetiel)

To zvyšuje záťaž geometrickej jednotky, ktorá musí pre každé svetlo počítať osvetlenie celej scény zvlášť. Aby testy neboli ovplyvnené rýchlosťou vyplňovania, sú použité pomerne kompaktné modely s vysokým počtom trojuholníkov a scény sú buď bez textúr, alebo sú použité malé textúry bez pokročilého filtrovania. Rozoberme si teraz jednotlivé testy:

- **Test1: Low Polygon Count (15 000 triangles/frame) –** objekt hlavy (Obrázok 5.2) je zložený z 15 000 trojuholníkov, patrí teda k najmenej náročným testom. Takisto nie sú použité žiadne zložité transformácie, len jednoduchá rotácia celého objektu
- **Test2: Medium Polygon Count (50 000 triangles/frame)** – na scéne sa nachádza 5 základných objektov (stolička, stôl, šálka, kanvica a lyžička), ktoré sú potom niekoľkokrát skopírované a rozmiestnené. Efekt odrazu v podlahe sa dosahuje vykreslením celej scény so zrkadlovito otočenou osou x (`glScalef(1.0,-1.0,1.0)`) a prekrytím odrazu priesvitnou plochou. Opäť sa teda nejedná o pokročilý efekt a na podlahu je aplikovaná len jedna textúra.
- **Test3: High Polygon Count (150 000 triangles/frame)** – jeden 3D model automobilu je zložený z viacerých objektov (karoséria, interiér...). Pre efekt odlesku na karosérii je použitá sféricky generovaná textúra s nízkym rozlíšením (environment mapping, generovanie príkazom `glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)`). Pre zvýšenie náročnosti testu sa automobil vykreslí dvakrát.

- **Test4: Very High Polygon Count (500 000 triangles/frame)** – najnáročnejší z tejto série testov, obsahuje takmer pol milióna polygónov. Takého počtu sa dosahuje hlavne rozmiestnením 100 objektov trávy a troch objektov stromov. Model auta je použitý z predchádzajúceho testu. Po technologickej stránke je práve na tomto teste vidieť pokles výkonu pri použití viacerých svetiel.




Graf 5.1: Pokles výkonu zobrazenia pri použití viacerých svetiel pri troch rôznych graf. kartách

Z grafu je vidno, že výkon závisí predovšetkým od hrubej sily GPU, takže majú výhodu modernejšie grafické procesory s vyšším taktovaním (tu konkrétne *GeForce 8600GT*).
V každom teste sa vypočíta, koľko polygónov je pri aktuálnom nastavení svetiel schopný akcelerátor vykresliť v miliónoch trojuholníkov za sekundu:

$$polygonRate = \frac{počet_{trojuholníkov} . FPS}{1000000} MTriangle/s$$

Celkové skóre zo všetkých testov sa určí podľa vzorca:

$$skóre = 3 . test1_{polygonRate} + 4 . test2_{polygonRate} + 5 . test3_{polygonRate} + 6 . test4_{polygonRate}$$

Dôležitosti testov narastajú so zvyšujúcim sa počtom polygónov, pretože sa tým zvyšuje presnosť merania. Pri nízkom počte polygónov je totiž pomerne výrazný vplyv fillrate na celkovú rýchlosť (akcelerátor strávi viac času rasterizáciou ako transformáciou polygónov).

**Zhrnutie série testov:**

- **Účel:** zistiť výkon akcelerátoru v oblasti transformácie a osvetľovania polygónových objektov.
- **Závisí od:** výkonu geometricko-transformačných jednotiek v GPU.
- **Váha v rámci celej aplikácie: 33%.** Keďže prakticky všetky 3D aplikácie pracujú s polygónovými modelmi, rýchlosť ich vykresľovania je dôležité kritérium pri posudzovaní celkového výkonu akcelerátoru.
- **Skóre na referenčnej karte:** 2000 bodov.

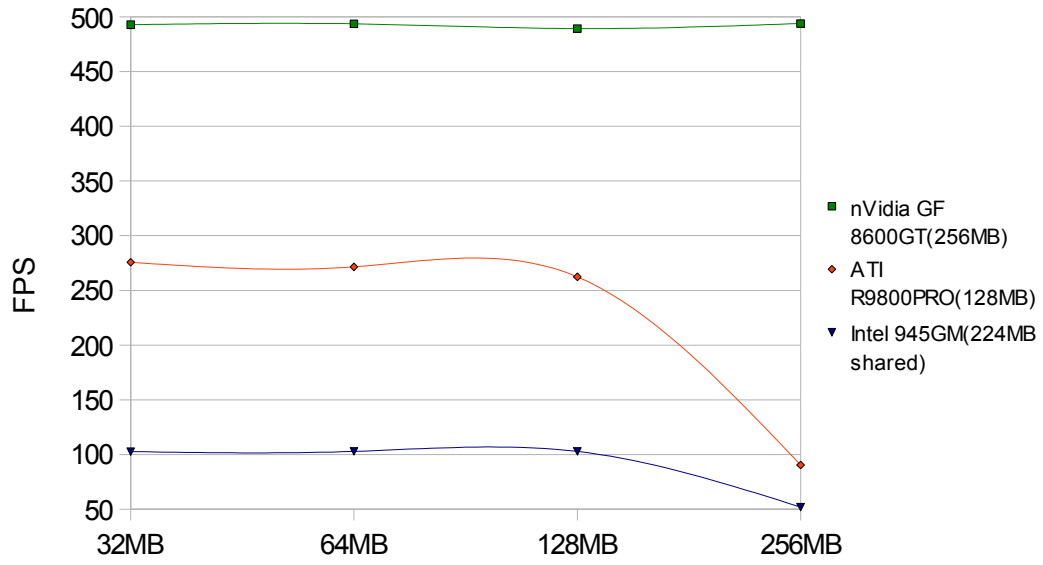
## 5.3 Testy grafickej pamäte

Pri tomto teste sa zisťuje, aký maximálny objem textúr je grafická karta schopná vykresliť bez výraznej straty výkonu. Základom každej scény je nízko-polygónový terén (ktorý sa vytvára z externého súboru), na ktorý sa aplikujú vygenerované textúry. Základná textúra o rozmere 512x512 pixelov sa načíta z JPG súboru (trieda `Texture`, kapitola 4.2.2) . Z tejto textúry sa potom generuje *N* textúr, pričom *N* sa volí tak, aby sa dosiahol zvolený celkový objem textúr. Tento počet vygenerovaných textúr sa dá vypočítať nasledovne, kde $textura_x$ a $textura_y$ sú rozmery textúry v pixeloch, *bpp* je farebná hĺbka textúry a *objem* je zvolená spotreba pamäte v MB, ktorú chceme dosiahnuť:

$$N = \frac{objem_{MB} . 2^{20}}{textura_x . textura_y . bpp}$$

Nové textúry sa generujú z pôvodnej textúry pomocou metódy `texture.MakeTexture()`. Tieto textúry sú posielané do video pamäte a v každom vykresľovacom cykle sa použije práve jedna textúra (`glBindTexture(GL_TEXTURE_2D, tsurface[N].texID)`). Takto sú využité všetky vygenerované textúry a priestor, ktorý majú obsadený vo video pamäti. Počet generovaných textúr sa postupne zvyšuje, čím rastie aj spotreba pamäte (od 32MB v prvom teste až po 256MB v poslednom teste). Pre zvýšenie náročnosti testu je na textúry použité anizotropné filtrovanie (OpenGL rozšírenie `GL_EXT_texture_filter_anisotropic`). Úroveň filtra je maximálna možná, akú podporuje grafická karta (typicky to býva 16x).

Účel týchto testov je taký, že ak je objem textúr väčší ako veľkosť pamäte na ich uloženie, musia sa textúry premiestniť do systémovej pamäte, do ktorej prístup je znateľne pomalší ako do pamäte karty.




Graf 5.2: Pokles výkonu pri použití rozsiahlych textúr

Zvýhodnené sú teda grafické karty s viac ako 256MB VRAM (ktoré nezaznamenávajú fakticky žiadny pokles výkonu), svoju úlohu hrá aj rýchlosť VRAM (pri vyššej frekvencii môže akcelerátor rýchlejšie pristupovať k textúram v nej uložených). Celkové výsledky sa opäť uvádzajú v snímkoch za sekundu (FPS). Váha jednotlivých testov sa zvyšuje s rastom objemu textúr, takže celkové skóre zo všetkých testov sa vypočíta nasledovne:

$$skóre = 0{,}5 \,.\, FPS_{32\mathrm{MB}} + FPS_{64\mathrm{MB}} + 1{,}5 \,.\, FPS_{128\mathrm{MB}} + 2 \,.\, FPS_{256\mathrm{MB}}$$

**Zhrnutie série testov:**

- **Účel:** zistiť vplyv veľkosti a rýchlosti grafickej pamäte na výkon pri vykresľovaní rozsiahlych textúr.
- **Závisí od:** veľkosti a rýchlosti grafickej pamäte.
- **Váha v rámci celej aplikácie: 16%.** Tento test ukazuje dôležitosť veľkosti grafickej pamäte, čo však nepatrí k najvýznamnejším aspektom výkonu.
- **Skóre na referenčnej karte:** 1000 bodov.

## 5.4 Testy realistických metód zobrazenia

Najnáročnejšia séria testov, ktorá otestuje grafický akcelerátor po všetkých stránkach. Zameriava sa na metódy, ktoré zvyšujú realitu scény. Medzi tieto metódy patria (okrem iných) dynamické odrazy pomocou stencil bufferu a dynamické tiene vytvárané pomocou tieňových máp. Grafický procesor je zaťažený transformáciami veľkého počtu detailných objektov a počítaním odrazov, grafická pamäť je využitá na ukladanie dynamickej tieňovej mapy vo vysokom rozlíšení (až 2048x2048 pixelov). Vo všetkých troch testoch sa vykresľuje rovnaká scéna, len sú použité odlišné efekty:

- **1.test** – odrazy pomocou stencil bufferu
- **2.test** – dynamické tiene pomocou tieňovej mapy
- **3.test** – obe metódy použité dohromady

### 5.4.1 Vytváranie odrazov za využitia stencil bufferu

Stencil buffer je súčasť obrazového bufferu , ktorá umožňuje vykresľovanie do zvolenej časti obrazu. Dá sa použiť na vytvorenie množstva efektov – okrem odrazov to môžu byť dynamické ostré tiene (shadow volumes) a konštrukcia telies pomocou CSG (Constructive Solid Geometry). My sa zameriame na dynamické, rovinné odrazy. Algoritmus funguje nasledovne [1]:

- vynulujeme stencil buffer a pripravíme ho na zápis masky, do ktorej sa bude vykresľovať (`glStencilFunc(GL_ALWAYS, 1, 1)`)
- vykreslíme zrkadliacu plochu do stencil bufferu. Zasiahnuté pixely bufferu sa nastavia na logickú „1“ (`glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE)`)
- nastavíme buffer do režimu čítania a do vzniknutej masky vykreslíme zrkadlový odraz scény invertovaním osi y pomocou volania `glScalef()` (Obrázok 5.3).

Obrázok 5.3: Zrkadlovo obrátená scéna vykreslená do masky stencil bufferu

- zapneme orezávaciu rovinu, aby sa odraz neobjavil nad zrkadliacou plochou (`glClipPlane(GL_CLIP_PLANE0, eqr)`)
- vykreslíme normálnu scénu

Pre vylepšenie efektu sa ešte vykreslí pôvodné zrkadlo s textúrou, ktorému sa nastaví prehľadnosť. Tým vznikne ilúzia vyleštěného kamenného stola. Tento efekt celkovo vyžaduje dvojpriechodové

vykreslenie (vykreslenie tej istej scény najskôr do stencil bufferu a potom z pohľadu kamery do color bufferu).

### 5.4.2 Dynamické tieňové mapy

Ide o jednu z metód na zobrazenie tieňov v scéne. Keďže je obrazovo založená, umožňuje vytvárať tiene aj pri komplexných scénach, pričom nepotrebujeme poznať polygónovú reprezentáciu objektov (na rozdiel od druhej rozšírenej metódy, vytváranie tieňových telies pomocou stencil bufferu). Zvolil som jednoduchšiu variantu tohto algoritmu, no menej efektívnu, keďže scéna sa vykreslí až v troch priechodoch [7]:

1. **priechod (pass):** celá scéna sa vykreslí z pohľadu svetla do textúry. Keďže nás zaujíma len z-hodnota každého pixelu, môžeme vypnúť vykresľovanie do farebného bufferu. Render do textúry je možné vytvoriť dvomi spôsobmi:
   - priamy render do textúry s pomocou framebuffer objektu (FBO) s pripojenou textúrou (cez OpenGL rozšírenie `GL_EXT_framebuffer_object`). Ak grafická karta podporuje toto rozšírenie, ide o predvolený spôsob vykreslenia.
   - menej efektívne kopírovanie pixelov z obrazovky do textúry (funkcia `glCopyTexSubImage2D()` ). V tomto prípade závisí maximálna veľkosť tieňovej mapy od rozlíšenia obrazovky.
2. **priechod:** vykreslíme scénu pri normálnom osvetlení.
3. **priechod:** urobíme projekciu tieňovej mapy do scény z pohľadu kamery. Pri tomto využívame automatické generovanie textúr (`glTexGenfv(GL_S, GL_EYE_PLANE, eq)`). Pred samotnou projekciou je ale treba vypočítať súradnice textúry, ktoré dostaneme súčinom (`glMultMatrixf()`) matíc pozície svetla, kamery a maticou nábehu, ktorá znormalizuje koordináty textúry tieňovej mapy. Po projekcii porovnávame hĺbkové hodnoty aktuálnej scény s hĺbkovou hodnotou tieňovej mapy. Ak má pixel scény menšiu hodnotu ako texel tieňovej mapy, nachádza sa v tieni a vykreslí sa s menším osvetlením. Toto porovnanie sa dá celé urobiť v hardware grafickej karty, pomocou OpenGL rozšírenia `GL_ARB_shadow`.

Nevýhodou tieňových máp je okrem náročnosti na video pamäť aj ich obmedzená veľkosť, čo pri veľkom priblížení scény vedie k aliasingu („zubatým hranám“). Existuje niekoľko metód na ich odstránenie (podrobnejší popis v [7]), avšak keďže toto nie je námetom aplikácie, pre jednoduchosť som ich ponechal bez zmeny (Obrázok 5.4).

Obrázok 5.4: „Zubaté“ okraje tieňu pri použití mapy s nízkym rozlíšením

**Tretí test** (shadows&reflections), ktorý v sebe zahŕňa obe metódy, potrebuje na vykreslenie scény až 4 priechody, čím sa stáva najnáročnejším testom celej aplikácie.

Celkové skóre testov sa počíta takto (najvyššiu prioritu má posledný test):

$$skóre = 2 . FPS_{test1} + 3 . FPS_{test2} + 4 . FPS_{test3}$$

**Zhrnutie série testov:**

- **Účel:** zistiť výkon akcelerátoru pri vykonávaní náročných algoritmov, ktoré zvyšujú realitu 3D scény.
- **Závisí od:** výkonu celého grafického akcelerátoru, predovšetkým však od GPU.
- **Váha v rámci celej aplikácie: 8%.** Testy sa zameriavajú na všetky súčasti grafického. akcelerátoru, ktoré však už boli otestované v predchádzajúcich sériách. Preto nižšia váha pri celkovom hodnotení.
- **Skóre na referenčnej karte:** 500 bodov.

## 5.5 Testy programovateľných tieňovacích jednotiek (shaderov)

Táto séria je zameraná na test programovateľných tieňovacích jednotiek, ktoré nahradzujú fixné transformačné a osvetľovacie jednotky používané v minulosti (kapitola 2.1.1.). Tie sú používané masovo v najnovších hrách, a umožňujú takmer fotorealistický vzhľad renderovanej scény. Umožňujú implementovať efekty ako osvetlenie na každý pixel scény, mapovanie nerovností (bump-mapping), odrazy a refrakciu, post-process filtre (ako sú rozmazanie, žiara...), animáciu atď. Tieto testy obsahujú jednoduché efekty, ktoré zaťažia ako fragment, tak aj vertex shader. Testy vyžadujú podporu OpenGL

Shading Language (GLSL), ktorá funguje na kartách ATI Radeon 9500 a lepší a nVidia GeForce FX a lepšie. Podrobný popis GLSL ako aj príklady na rôzne shadery obsahuje publikácia [2].

### 5.5.1 Test 1: Phongov svetelný model

Klasický osvetľovací model použitý vo fixných geometrických jednotkách počíta osvetlenie len na každý vertex objektu (per-vertex, Gouraud shading, popísané v [1]). Avšak keď má objekt malý počet polygónov, je vidieť nekvalita použitého osvetlenia najmä na odleskoch objektu (Obrázok 5.5).

Naproti tomu shadery nám umožňujú počítať osvetlenie pre každý pixel scény podľa Phongovho osvetľovacieho modelu (Phong shading). Je to výkonovo náročnejšie, ale scéna vyzerá neporovnateľne lepšie. Shadery sú implementované v jazyku GLSL, ktorý sa veľmi podobá jazyku C. Tým je vývoj shaderov výrazne uľahčený, keďže staršie verzie shaderov sa ešte museli písať v assembleri.

Obrázok 5.5: Porovnanie: Phong shading (vľavo) a Gouraud shading(vpravo)

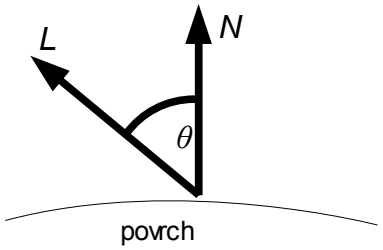
**Vertex shader** má podstate za úlohu len preposlať údaje do fragment shaderu a vykonať transformáciu vertexu. Zistíme si údaje o normále vertexu a o pozíciách svetiel vzhľadom k nemu. Potom je potrebné ešte zistiť aj pohľadový vektor, čo je vzdialenosť kamery od vertexu:

```
normal = gl_NormalMatrix * gl_Normal;  //normala povrchu
lightDir = vec3(gl_LightSource[0].position.xyz); //smer svetla
eyeVec = -vec3(gl_ModelViewMatrix * gl_Vertex);  //pozicia pohladu
gl_Position = ftransform();    //transformacia vertexu
```

Všetky náročné per-pixel výpočty vykonáva **fragment shader**, test je teda veľmi závislý na jeho výkone. Pre každý pixel sa vypočítajú tri zložky osvetlenia – okolité osvetlenie (ambient), difúzne osvetlenie (diffuse) a odlesk (specular). Okolité osvetlenie sa vypočíta prostým vynásobením okolitej zložky svetla a materiálu:

```
vec4 col_ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
```

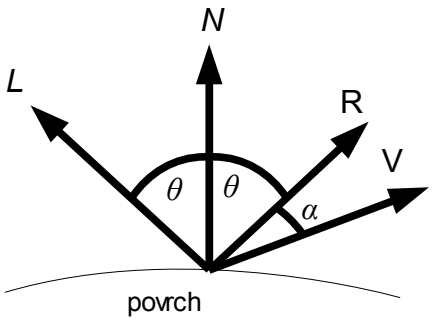
Difúzne osvetlenie vypočítame pomocou Lambertovho osvetľovacieho modelu ako skalárny súčin vektoru svetla a normály povrchu krát difúzna zložka svetla a materiálu, kde $I_L$ je difúzna zložka svetla, $r_D$ je difúzna zložka materiálnu, $N$ je vektor normály a $L$ je vektor svetla:




$$I_D = I_L . r_D . \cos\Phi$$
$$I_D = I_L . r_D . (\vec{N} . \vec{L}) \quad [3]$$

Obrázok 5.6: Lambertov osvetľovací model

Zostáva určiť poslednú zložku, odlesk. Ten sa vypočíta pomocou Phongovho osvetľovacieho modelu, kde $I_L$ je zložka odlesku svetla, $r_S$ zložka odlesku materiálu, $V$ je pohľadový vektor, $L$ je vektor svetla, R je opačný vektor svetla a $n_S$ je lesklosť materiálu (od 0 po 128):




$$I_S = I_L . r_S . \cos\alpha$$
$$I_S = I_L . r_S . (\vec{V} . \vec{L})^{n_S} \quad [3]$$

Obrázok 5.7: Phongov osvetľovací model

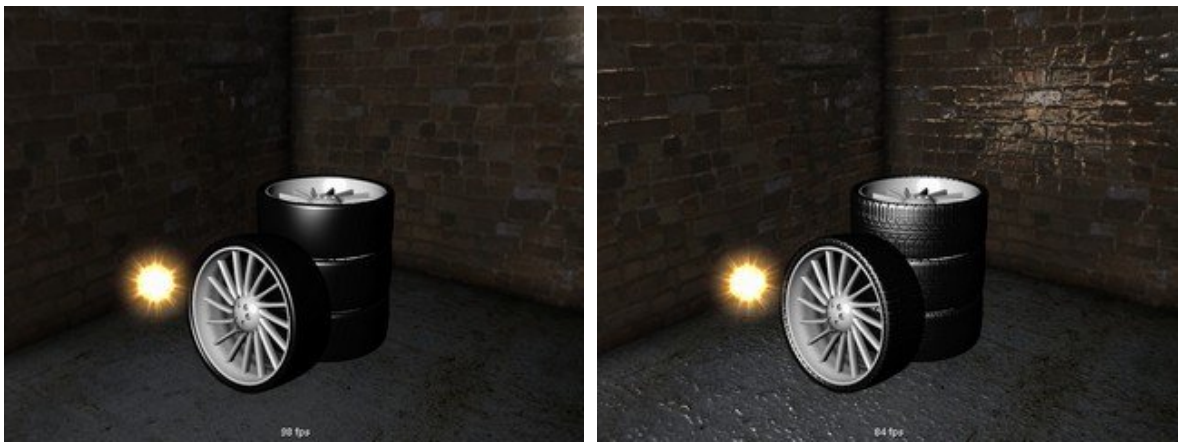
Celková farba fragmentu sa dosiahne sčítaním všetkých vypočítaných zložiek:

```
gl_FragColor = col_ambient + col_diffuse + col_specular
```

V samotnom programe je potreba načítať shader zo súboru, skompilovať ho a pripojiť k programu, potom len stačí nahradiť fixné jednotky shaderom. Načítanie shaderu má na starosti funkcia `LoadShader()`. Z takto získaného reťazca sa vytvorí a skompiluje shader (`glShaderSource()`, `glCompileShader()`), ktorý sa nakoniec zlinkuje s hlavnou aplikáciou (`glLinkProgram()`, `glUseProgramObject()`). Toto treba urobiť pre fragment aj vertex shader, čím je shader pripravený na použitie. Scéna pozostáva z modelu kanvice (`glutSolidTeapot()`), ktorá je osvetlená tromi bodovými svetlami. Pre každý pixel sa tak musí počítať osvetlenie zo všetkých troch svetiel.

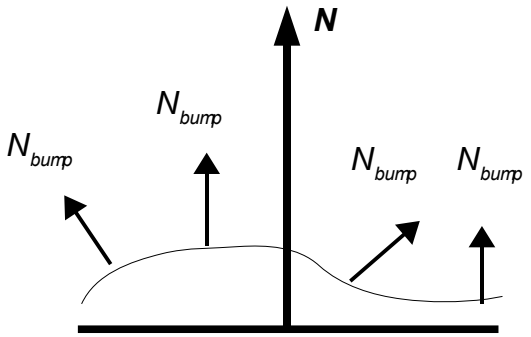
### 5.5.2 Test 2: Mapovanie nerovností (bump mapping)

Bump-mapping je technológia, ktorá umožňuje získať povrchu dojem nerovnosti, pričom jeho geometria sa nemení. Existuje viacero druhov implementácie, my sa zameriame na vizuálne najkorektnejšiu variantu modifikácie povrchovej normály pomocou fragment shaderu [2].



Obrázok 5.8: Rozdiel medzi vypnutým (vľavo) a zapnutým (vpravo) bump mappingom

Používa sa modifikovaný shader z minulého testu. Vertex shader je zhodný, vo fragment shaderi je pridaný multitexturing, pretože na každý objekt sú aplikované dve textúry: farebná mapa a bump mapa. K textúre pristupujeme pomocou premennej typu `sampler2d` a funkcie `texture2D(tex,gl_TexCoord[0].st)`. Hodnoty intenzity z bump mapy (alebo tiež normal mapy) sa používajú na modifikáciu povrchovej normály: jas texelu z bump mapy sa vynásobí s interpolovanou normálou, čím sa dosiahne efekt hrboľatého povrchu (upravená normála ovplyvňuje inenzitu svetla). V obrázku *N* označuje pôvodnú normálu, $N_{bump}$ sú modifikované normály [2]:




```
vec3 N = normal * vec3(bumpmap)
```

Obrázok 5.9: Modifikácia povrchovej normály pri bump mappingu

Osvetlenie sa vypočíta takisto ako v predchádzajúcom teste, ibaže nakoniec sa celková farba fragmentu vynásobí s farebnou zložkou textúry. Používajú sa pomerne veľké bump-mapy, takže tento test je náročný na výkon fragment shaderu.

### 5.5.3 Test 3: Animovaný vertex shader

Opäť je použitý upravený shader z minulého testu. Fragment shader je prakticky identický, vo vertex shaderi však pribudla modifikácia Y-súradnice každého vertexu. Tá spočíva vo využití funkcie sinus na zmenu hodnoty, čo vo výsledku produkuje efekt vlnenia morskej hladiny:

```
vec4 v = gl_Vertex;
wave = sin(5.0*v.x + time)*cos(5.0*v.z)*0.2;     //efekt vlny
```

Zmena času (ako rýchlosť vlnenia) sa do shaderu posiela z hlavnej aplikácie cez uniformnú premennú `time`. Tento test zaťažuje fragment shader (per-pixel osvetlenie a bump mapping), avšak menej ako v predchádzajúcej scéne, pretože je použitá len jedna textúra ako bump mapa, ktorá je navyše v nižšom rozlíšení. Podstatne viac je zaťažený vertex shader, ktorý musí navyše modifikovať pozíciu každého vertexu.

Keďže najnáročnejší je druhý test (bump mapa vo vysokom rozlíšení a per-pixel osvetlenie), vo výpočte celkového skóre má najvyššiu váhu:

$$skóre = 2 . FPS_{test1} + 4 . FPS_{test2} + 2 . FPS_{test3}$$

**Zhrnutie série testov:**

- **Účel:** zistiť výkon akcelerátoru pri vykonávaní naprogramovaných inštrukcií s jednotlivými vrcholmi a fragmentami.
- **Závisí od:** výkonu programovateľných tieňovacích jednotiek v GPU.
- **Váha v rámci celej aplikácie: 16%.** Shadery majú veľké využitie moderných aplikáciách, preto výkon pri ich zobrazovaní je dôležitý ukazovateľ výkonnosti celého grafického procesoru.
- **Skóre na referenčnej karte:** 1000 bodov.

# 6 Interpretácia výsledkov

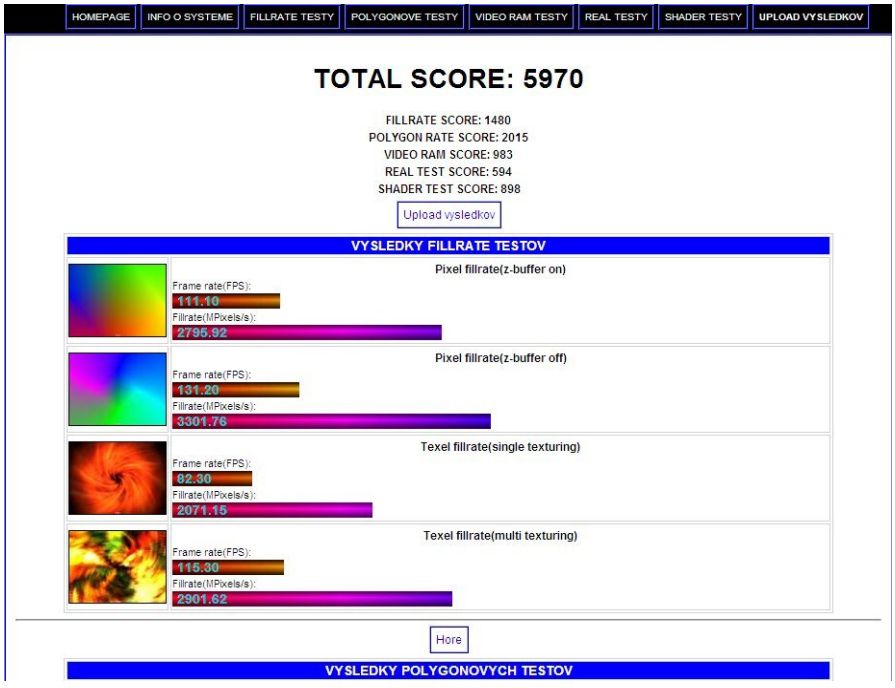
Aby sa výsledky dali porovnávať medzi rôznymi graf. kartami a systémami, treba dodržať určité konvencie. **Implicitné nastavenie testu** je nasledovné: rozlíšenie 1024x768, 32-bit farby, bez anti-aliasingu, predvolené nastavenie ovládačov grafickej karty. Pred samotným testovaním je potrebné **vypnúť všetky aplikácie, ktoré ovplyvňujú celkový výkon počítača**. Pre maximálny výkon je vhodné pred spustením testu reštartovať počítač.

## 6.1 Celkové skóre

Celkové skóre sa vypočíta ako súčet čiastkových výsledkov z jednotlivých sérií testov (viď kapitola 5). Ak teda akcelerátor nie je schopný spustiť niektorú sériu testov (najčastejšie shader testy pri starších integrovaných kartách), celkové skóre je nižšie:

*total_score = fillrate_score + polygon_score + videoram_score + real_score + shader_score*

Celkové skóre na referenčnej karte (*ATI 9800PRO*) je **6000 bodov,** čo približne zodpovedá výsledkom benchmarku 3D Mark 03. Nie však presne, pretože treba brať do úvahy, že 3D Mark 03 funguje pod rozhraním DirectX a je viacej herne zameraný. A v neposlednom rade moja aplikácia nie je optimalizovaná pre všetky grafické karty, vzhľadom k nedostatku testovacích systémov (počas vývoja aplikácie boli neustále k dispozícii len dve grafické karty). Výsledky sú užívateľovi podané v grafickej podobe vo forme HTML stránky:

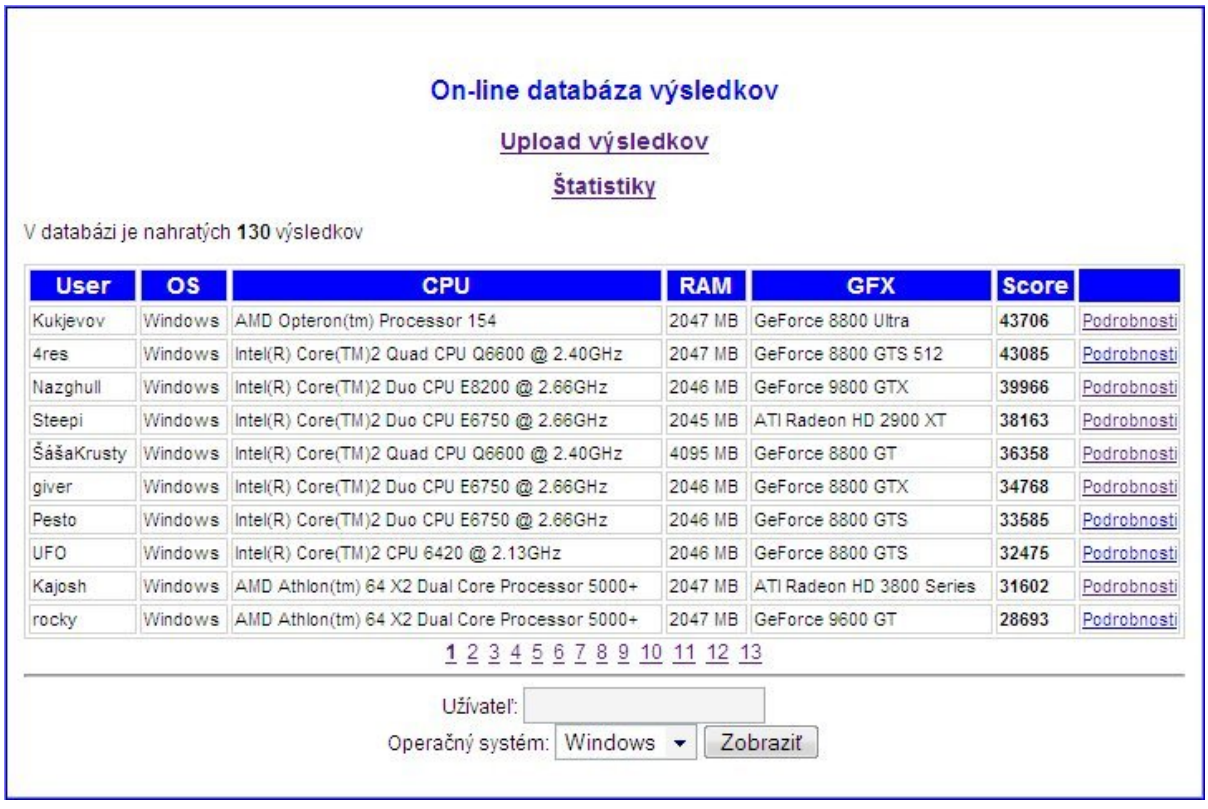



Obrázok 6.1: Vygenerovaná HTML stránka s výsledkami

Táto stránka sa dynamicky generuje z výsledkového XML súboru (kapitola 4.1). Na začiatku stránky sú údaje o celkovom skóre a čiastkových výsledkoch jednotlivých testov. Každý test má pri sebe svoj obrázok a zobrazené výsledky v grafickej forme (stĺpce s farebným prechodom). Maximálny rozsah stĺpcových grafov je 0-600 snímkov/sekunda (tento rozsah je opäť odvodený od výsledkov referenčnej karty). Poslednou súčasťou HTML stránky sú detailné informácie o systéme (kapitola 4.1.1.) a odkaz na nahratie výsledkov do databázy.

## 6.2 Databáza výsledkov

Keďže samotné skóre vytvorené aplikáciou by bolo samo o sebe bezcenné, je treba mať možnosť vytvorené skóre porovnávať s inými užívateľmi. To je dôvod, prečo som vytvoril on-line databázu výsledkov, ktorá zhromažďuje a umožňuje porovnávať výsledky rôznych užívateľov. Táto databáza je prístupná na URL http://www.stud.fit.vutbr.cz/~xvanek29/bench/.




Obrázok 6.2: Databáza výsledkov

Základom je databáza, v ktorej sú uložení užívatelia, najdôležitejšie informácie o systéme a celkové skóre. Pri požiadavku na podrobnejšie preskúmanie výsledku zvoleného užívateľa sa generuje stránka s podrobnosťami. Generovanie prebieha z uloženého XML súboru pomocou parseru napísaného v jazyku PHP. Je možné vyhľadávať užívateľa podľa mena a podľa operačného systému.

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 (nasledujúca kapitola).

## 6.3 Výsledky testovania

K dátumu dokončenia práce (8.5.2008) bolo do databázy nahratých **130 výsledkov**, z toho **106 unikátnych** (jeden užívateľ môže nahrať viacero výsledkov). Rozpätie výsledkov bolo veľmi široké, od 98 bodov karty *S3 Twister* až po 43706 bodov u karty *GeForce 8800Ultra*. Dominantné zastúpenie mali predovšetkým mobilné a integrované grafické karty, ktoré však dosahovali nižších výsledkov v porovnaní s ich desktopovými alternatívami.

Priemerné skóre dosiahlo úrovne **7227.04 bodov**, medián z výsledkov je **2865 bodov**. Tento pomerne veľký rozdiel je spôsobený veľkým počtom mobilných a integrovaných grafických kariet so skóre okolo 2000 bodov, a na druhej strane prítomnosť veľmi výkonných kariet v prvej desiatke so skóre viac ako 10 000 bodov, čo zdvihlo celkový priemer.

Z hľadiska rozšírenosti dominovali integrované karty od spoločnosti Intel, konkrétne čipy *i945GM* a *i915GM*. Najrozšírenejšou samostatnou mobilnou kartou sa stala *GeForce Go7300*, desktopovou kartou *GeForce 8600GT*. Celkovo boli najrozšírenejšie grafické karty od spoločnosti nVidia , nasledované ATI. Zvyšok tvorili karty Intel:

1. *NVIDIA* **43-krát (40.57%)**
2. *ATI* **31-krát (29.25%)**
3. *Intel* **27-krát (25.47%)**
4. *S3* **1-krát (0.94%)**

Keďže aplikácia je multiplatformná, ponúka sa možnosť porovnať kvalitu ovládačov grafickej karty pod OS Windows a Linux. Niekoľko užívateľov poskytlo výsledky aj pod oboma OS. Najvyrovnanejšie výsledky boli u kariet značky *nVidia*, čo sa dá vysvetliť kvalitnými ovládačmi pre oba OS. Rozoberme si teraz ovládače od *ATI* na príklade karty *ATI Mobility Radeon X1400*. Pod OS Windows dosiahla skóre **3571**, pod OS Linux to bolo **3244**. Tento rozdiel je veľmi malý, takže môžeme konštatovať, že kvalita open-source ovládačov *ATI* sa citeľne zvýšila (oproti minulosti). U ovládačov pre karty *Intel* je situácia trochu iná. S čipom *i915* a procesorom o takte 1400MHz v prostredí Windows dosiahla skóre **615** bodov, v Linuxe to bolo len **369**.

Po výkonovej stránke dominovali karty z rodiny *GeForce 8800* (výsledky nad 30-tisíc bodov). Zaujímavý je vyšší výsledok tejto generácie oproti novšej rade *9800*, ale nie prekvapivý, pretože po hardwarovej stránke sa karty radu *9800* príliš nelíšia od starších. Pri jednotlivých sériách testov takisto dominovali karty radu *8800*, okrem polygónových testov. Tu dosahovali najlepšia výsledky karty *ATI Radeon HD2900XT* a HD*3800*, čo sa dá vysvetliť vyššími taktami grafického jadra týchto kariet.

Karty s najvyšším skóre:

1. *GeForce 8800 Ultra* **43706 bodov**
2. *GeForce 8800 GTS 512* **43085 bodov**
3. *GeForce 9800 GTX* **39966 bodov**
4. *ATI Radeon HD 2900 XT* **38163 bodov**
5. *GeForce 8800 GTX* **34768 bodov**

Seriózne závery sa však na základe týchto výsledkov nedajú príliš robiť, pretože veľkosť databázy (130 výsledkov) je pomerne malá, navyše väčšina výsledkov pochádza od mobilných a integrovaných grafických kariet. Výsledky môžu byť aj skreslené nedostatočným počtom zhodných systémových konfigurácií. Aj z tejto malej vzorky akcelerátorov je však vidieť veľké rozdiely medzi samostatnými a integrovanými kartami, kde integrované nemajú často dostatočný výkon pre náročné 3D aplikácie. Čo sa týka poradia výkonnosti kariet, vcelku zodpovedá výsledkom, aké pri porovnaní dávajú komerčné benchmarky. V súčasnosti sa javia ako výkonnejšie karty od firmy *nVidia*. Zaujímavé je zistenie, že karty staršej generácie (napr. *ATI Radeon 9800PRO*) stále výkonovo stačia na rozšírené mobilné grafické karty novšej generácie (napr. *GeForce 8600M GS*).

# 7 Záver

Aplikáciou bolo otestované široké spektrum grafických akcelerátorov na rôznych systémoch a pod rôznymi operačnými systémami (Windows XP, Vista, rôzne distribúcie Linuxu). To považujem za slušný úspech, keďže vo väčšine prípadov sa nevyskytli vážnejšie problémy so stabilitou aplikácie a nekorektným zobrazovaním. Toto potvrdzuje správnosť multiplatformného návrhu aplikácie a použitie rozhrania OpenGL. V rámci porovnania výsledkov medzi rôznymi OS môžeme konštatovať, že v poslednej dobe sa ovládače určené pre OS Linux zlepšujú a vo veľa prípadoch podávali rovnaké, ak nie lepšie výsledky. Bohužiaľ sa vyskytlo pár príkladov falšovania výsledkov, čo je námet na ďalšie vylepšenie aplikácie (zabezpečenie súboru s výsledkami).

Pre zaujímavosť uvediem zopár porovnaní celkového skóre z mojej aplikácie (***gluxMark***) v porovnaní s komerčným benchmarkom 3D Mark 2003. Uvediem len rozsahy, pretože sa nedá úplne vylúčiť vplyv procesoru a celého systému, ako v mojej aplikácii, tak aj v 3D Mark 2003. Údaje o výkone akcelerátorov v 3D Marku som získal z [11], pričom som vyberal systémy s podobným výkonom, ako nahraté výsledky v mojej databáze:

| Grafická karta | Skóre: 3D Mark 2003 | Skóre: gluxMark |
|---|---|---|
| *nVidia GeForce 8800 GT* | 30 000 – 40 000 | 30 000 – 35 000 |
| *nVidia GeForce 8600 GT* | 13 000 – 17000 | 12 000 – 14 000 |
| *ATI Radeon 9800PRO* | 5500 – 7000 | 5500 – 6000 |
| *nVidia GeForce Go7300* | 3500 – 4500 | 2300 – 3000 |

Celkové výsledky sa preukázali ako vcelku podobné, ale samozrejme nie sú úplne zhodné. Treba si uvedomiť, že 3D Mark 2003 funguje pod rozhraním DirectX, kde môžu mať grafické karty odlišný výkon ako v OpenGL, ako aj približné určenie rozsahov skóre (veľmi rozsiahla databáza 3D Marku).

Vývoj aplikácie bola veľmi zaujímavá činnosť, ktorá ma nesmierne obohatila o nové poznatky z oblasti počítačovej grafiky a grafických akcelerátorov. V budúcnosti je možné aplikáciu ďalej rozvíjať, pridávať nové testy a zlepšovať užívateľské rozhranie a databázu výsledkov. Po nevyhnutných modifikáciách a optimalizáciách by mohla tvoriť základ omnoho komplexnejšej, platformovo nezávislej a slobodnej aplikácie, zameranej na nezávislé testovanie výkonu grafických kariet pod rozhraním OpenGL, s prístupom a ovládaním podobným ako komerčné 3D Marky. Existuje totiž veľmi málo takýchto aplikácií pod rozhraním OpenGL, prípadne existujúce aplikácie sú už technologicky a morálne zastaralé. Očakávané vypustenie nového rozhrania OpenGL 3.0 určite prinesie nové možnosti, ako zvýšiť vizuálnu kvalitu 3D aplikácií a opäť sa vyskytne potreba porovnávania výkonu aj pod týmto rozhraním.

# Literatúra


[1] Shreiner, D., Woo, M., Neider, J., aj.: *The OpenGL Programming Guide,* Fourth Edition. 2003, 928 s, ISBN 0321481003

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

[3] Kršek, P.: *Základy počítačové grafiky*, studijní opora, FIT VUT v Brně, 2006

[4] Kilgard, M. J.: *The OpenGL Utility Toolkit (GLUT) Programming Interface* [online]. 1996, URL: http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf (máj 2008)

[5] wxWidgets: *A cross-platform GUI library* [online]. 2007
URL: http://docs.wxwidgets.org/stable/ (máj 2008)

[6] Microsoft Corporation: *Microsoft Developer's Network* [online]. 2008
URL: http://msdn.microsoft.com/en-us/downloads/ms348103.aspx (máj 2008)

[7] Everitt, C., NVIDIA Corporation: *Shadow Mapping* [online]. 2001,
URL: http://developer.nvidia.com/object/shadow_mapping.html (máj 2008)

[8] Guinot, J. *Introduction to Graphics Controllers* [online]. 2006,
URL: http://www.ozone3d.net/tutorials/graphics_controllers.php (máj 2008)

[9] Wikipedia: *Graphics card* [online]. 2008,
URL: http://en.wikipedia.org/wiki/Graphics_card (máj 2008)

[10] Autodesk Ltd.: *3D Studio File Format (3ds)*. [online]. 1994
URL: http://www.whisqu.se/per/docs/graphics56.htm (máj 2008)


**Odkazy na WWW stránky popisovaných testovacích aplikácií:**


[11] Futuremark Corporation: On-line Results Browser
URL: http://service.futuremark.com/search/form.action

[12] The Standard Performance Evaluation Corporation
URL: http://www.spec.org/

[13] MAXON Computer GmbH: Cinebench R10
URL: http://www.maxon.net/pages/download/cinebench_e.html

# Zoznam príloh

Príloha 1. Spustenie a ovládanie aplikácie

Príloha 2. Stručný prehľad tried a funkcií v zdrojových kódoch

Príloha 3. CD so zdrojovými kódmi aplikácie, spustiteľnou aplikáciou, manuálom a touto správou

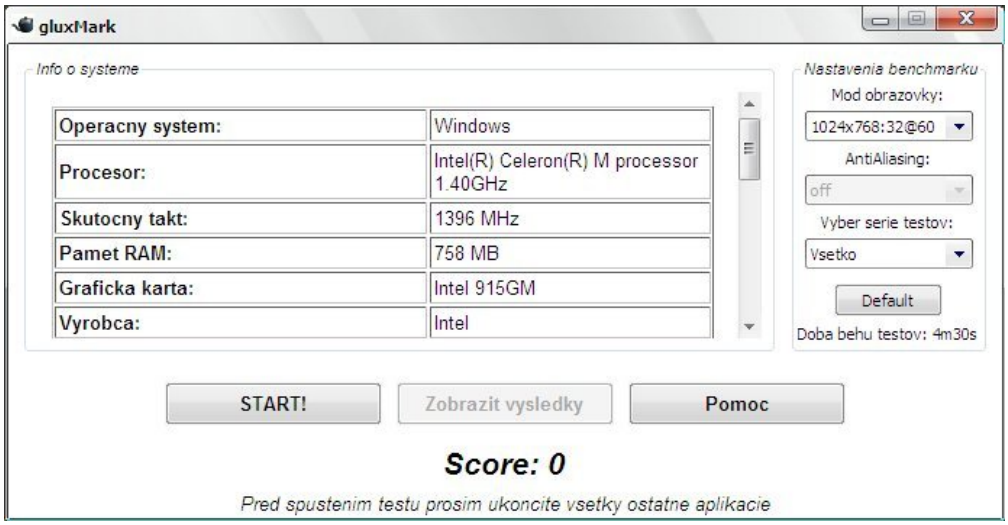
# Príloha č. 1 – Spustenie a ovládanie aplikácie

Aplikácia sa dá spustiť buď z predpripravených binárnych súborov, alebo preložením zdrojových kódov pomocou príkazu `make`. Na preloženie je potrebné mať nainštalované vývojové knižnice a hlavičkové súbory grafického rozhrania OpenGL GLUT a vizuálneho toolkitu wxWidgets vo verzii aspoň 2.8.

**Požiadavky na systém:**

- Procesor: s frekvenciou aspoň 1GHz
- Pamäť: stačí 256MB, pre lepší beh niektorých testov je lepšie 512MB
- Miesto na disku: 10MB
- Grafická karta: prakticky každá s podporou OpenGL 1.4 ako *nVidia GeForce 3*, *ATi Radeon 8500* apod. Doporučené karty pre hladký priebeh všetkých testov sú *GeForce FX* a lepšie, prípadne *ATI Radeon 9500* a lepšie
- Operačný systém: Windows (XP, Vista), Linux
- Ovládače grafickej karty s podporou OpenGL vo verzii aspoň 1.4 (lepšie ale OpenGL 2.0)

Na spustenie testov slúži jednoduché grafické rozhranie, kde sa dajú nastaviť základné parametre programu a testov.




Obrázok 7.1: Užívateľské rozhranie aplikácie

- **Rozlíšenie:** je možnosť vybrať si z každého podporovaného rozlíšenia, farebnej hĺbky a obnovovacej frekvencie monitora. Napríklad výber 1024x768:32@60 značí rozlíšenie 1024x768 pixelov, farebnú hĺbku 32-bit na pixel a obnovovaciu frekvenciu 60Hz.

- **Séria testov:** dá sa spustiť buď celý benchmark, prípadne si vybrať sériu testov:
  1. **Vsetky** - všetky testy (implicitná voľba)
  2. **Fillrate** - testy vyplňovania
  3. **Polygony** - polygónové testy
  4. **VideoRAM** - testy grafickej pamäte
  5. **Realne metody** - metódy realistického zobrazenia, tiene a odrazy
  6. **Shadery** - testy fragment a vertex shaderov

Tlačidlo **Default** nastaví implicitné parametre testu (rozlíšenie 1024x768 pri 32-bitoch, všetky testy). **START!** spustí samotné testovanie, **Zobrazit vysledky** zobrazí (po skončení testov) súbor s výsledkami a tlačidlo **Pomoc** odkáže užívateľa na web stránku s návodom. Úplne dole je vypísané dosiahnuté skóre po skončení testov. Počas priebehu je možné program ovládať cez klávesy. Test je možné kedykoľvek ukončiť stlačením "ESC". Jednotlivé scény sa dajú preskakovať klávesou "medzera" a pre zaujímavosť je tu klávesa "W", ktorá zapne drôtené zobrazenie(wireframe) všetkých objektov.

V prípade nefunkčnosti grafického rozhrania, alebo kvôli možnosti dávkového spúšťania jednotlivých testov sa hlavná aplikácia dá ovládať pomocou parametrov v príkazovom riadku. Formát parametrov:

```
bin/bench.exe séria_testov režim_zobrazenia antialiasing
```

Séria testov:

- `all` – všetky testy
- `fillrate` – testy zamerané na rýchlosť rasterizácie
- `polygon_count` – polygónové testy
- `video_RAM` – testy grafickej pamäti
- `real_methods` – testy metód realistických zobrazení
- `shaders` – testy zamerané na programovateľné jednotky

Režim zobrazenia: nastavuje sa ním mód zobrazenia v celoobrazovkovom režime. Je to reťazec vo formáte $res_x$`x`$res_y$`:bpp@freq`, kde $res_x$ a $res_y$ sú rozmery rozlíšenia obrazovky, `bpp` je bitová hĺbka farieb a `freq` je obnovovacia frekvencia monitoru. Príklad: `1024x768:32@60`.

Antialiasing: slúži na nastavenie úrovne vyhladzovania hrán pri použití multi samplingu. Predvolená úroveň je `off` (vyhladzovanie vypnuté), ďalšie úrovne sú `2x`, `4x` a najvyššia je `6x`.

Ďalšie informácie k aplikácii je možné nájsť na URL: http://www.stud.fit.vutbr.cz/~xvanek29/bench

# Príloha č.2 – Stručný prehľad tried a metód v zdrojových kódoch

Testovací program sa skladá z nasledovných modulov:

- ***main.cpp:*** vstupný bod aplikácie, hlavný modul. Implementuje funkciu `main`, ktorá inicializuje rozhranie GLUT pre zobrazenie na celej obrazovke. Nastavenie tohto režimu sa prijíma cez parametre. Ďalej inicializuje OpenGL, časovač a prvý test. Podľa zvoleného rozsahu testov po skončení aktuálneho testu buď prejde na ďalší test, alebo ukončí program.
- ***globals.h:*** hlavičkový súbor obsahujúci všetky potrebné OpenGL aj systémové knižnice, ako aj externé premenné a konštanty používané v celom programe.
- ***utils.h:*** obsahuje najpoužívanejšie funkcie OpenGL, akými sú vykreslenie obdĺžnika, nastavenie svetiel a materiálov, zapnutie/vypnutie priehľadnosti, ako aj nastavenie kamery a základné farby.
- ***mat.cpp/mat.h:*** skladá sa z dvoch tried. Trieda `Vector` implementuje základné operácie s vektormi ako zistenie dĺžky, normalizáciu a skalárny/vektorový súčin. Trieda `Matrix4x4` obsahuje len zopár základných funkcií na prácu s maticami o rozmeru 4x4 (OpenGL matica), ako sú násobenie a vrátenie určitého riadku.
- ***object.cpp/object.h:*** má triedu `Object_3DS`. Zabezpečuje všetko potrebné pre import objektu vo formáte 3DS ako je načítanie zo súboru, výpočet normál a vytvorenie display listu pre vykreslenie v OpenGL.
- ***texture.cpp/texture.h:*** trieda `Texture`, ktorá slúži na import textúry vo formáte JPG. Obsahuje funkciu na načítanie a vytvorenie textúry s požadovanými parametrami, akými sú filtrovanie a mip-mapping.
- ***font.cpp:*** obsahuje funkciu, ktorá vypisuje na obrazovku text pomocou bitmapových znakov z knižnice GLUT (`glutBitmapCharacter()` ).
- ***test.cpp/test.h:*** základná trieda `Test`, ktorá v sebe obsahuje vlastnosti a metódy spoločné pre všetky testy. Je tu počítanie celkových snímkov testu, nastavenie doby trvania testu a výpočet priemerného počtu snímkov za sekundu. Takisto tu sú virtuálne konštruktory a deštruktory na inicializáciu, deinicializáciu a spustenie testu. Poslednou dôležitou súčasťou je funkcia na získanie informácií o systéme (kapitola 4.1.1.).
- ***test_*.cpp/test_*.h:*** samotné implementácie jednotlivých sérií testov (kapitola 5). Každý súbor obsahuje triedu `Test_*` odvodenú od triedy `Test` a pridáva si do nej vlastnosti potrebné pre tento typ testu. Inicializácia väčšinou znamená načítanie externých dát, nastavenie potrebných OpenGL vlastností a spustenie testov. Deinicializácia zase uvoľňuje použité dynamické dátové typy. Vždy je implementovaná funkcia `Run()`, ktorá vykonáva

samotný priebeh testu, ako aj funkcia `SaveResults()`, ktorá uloží výsledky testu. Ostatné doplnkové metódy sú vytvárané podľa zamerania a rozsiahlosti testu (napr. metóda `DrawScene()` na vykreslenie celej scény).

- ***Glee.c/GLee.h:*** *OpenGL Easy Extension library* - voľne šíriteľná knižnica na uľahčenie práce s OpenGL rozšíreniami od autora Bena Woodhausa (URL: http://elf-stone.com/glee.php).

Grafické rozhranie sa skladá z nasledovných modulov:

- ***gui.cpp/gui.h:*** implementuje grafické rozhranie wxWidgets aplikácie. Aplikácia obsahuje štandardný cyklus spracovania správ, na základe ktorých reaguje na príkazy užívateľa. Hlavnou triedou okna je `glFrame`, ktorá v sebe obsahuje grafické prvky (tlačítka, menu) na ovládanie aplikácie. Na základe nastavení potom spúšťa hlavnú aplikáciu a čaká na ukončenie testov. Potom vypočíta celkové skóre pomocou metódy `CalculateScore()` a zobrazí v hlavnom okne.
- ***parser.cpp:*** obsahuje dve funkcie: `ParseSystemInfo()` zisťuje zo súboru *sysinfo.xml* (vytvoreného hlavnou aplikáciou) informácie o systéme metódou parsovania XML súboru. Pri tom sa používa wxWidgets objekt `wxXmlDocument`, ktorý uľahčuje získavanie dát z XML súboru. Druhá funkcia, `ParseResults()`, získava dáta zo súboru *results.xml*. Z týchto dát vytvorí HTML stránku a uloží ju do adresára s programom (*results.html*). Stránka sa potom otvorí v predvolenom prehliadači.