VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Semestrální projekt

Měřič výkonnostních charakteristik OpenGL

Dalibor Pernica 2006

Měřič výkonnostních charakteristik OpenGL

Vedoucí:

Pečiva Jan, Ing., UPGM FIT VUT

Konzultant:

Pečiva Jan, Ing., UPGM FIT VUT

Přihlášen:

Pernica Dalibor

Zadání:

1. Seznamte se s OpenGL API a grafickou knihovnou GLUT.
2. Navrhněte a naimplementujte aplikaci pro měření výkonnosti OpenGL.
3. Zpracujte dokumentaci k aplikaci pro budoucí rozšiřování.
4. Zpracujte sadu měřících testů a proveďte měření na různých hardwarových konfiguracích.
5. Naměřené výsledky zpracujte a publikujte na internetu.

Kategorie:

Počítačová grafika

Implementační jazyk:

C++

Operační systém:

Linux, Windows

Literatura:

www.opengl.org

Prohlášení:

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Jana Pečivy.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Poděkování:

Tímto chci poděkovat panu Ing. Janu Pečivovi za odborné vedení a čas strávený nad semestrálním projektem.

Abstrakt:

Cílem této práce je stručný popis návrhu a implementace programu pro měření výkonnostních charakteristik OpenGL s výhledem na jeho další vývoj. Popsaný program je novou vylepšenou verzí svého předchůdce. Při návrhu byl kladen důraz na jednoduchost a snadnou rozšiřitelnost, což si vynutilo použití objektového přístupu. V druhé části jsou uvedeny výkonnostní charakteristiky pro různé grafické karty naměřené tímto programem během testování. Sleduje se propustnost vrcholů pro různé vykreslovací operace (klasické trojúhelníky, pole vrcholů, VBO). V závěru práce jsou uvedeny náměty na další vývoj.

Klíčová slova:

OpenGL, GLUT, výkonové ukazatele, rasterizace, grafický akcelerátor, vertex, pole vrcholů, vertex buffery objektů, VBO

Abstract:

The work focuses on description and implementation of a OpenGL performance analyzer that uses OpenGL Utility Toolkit (GLUT) library. The goal of the project is to develop simple and extendable program. The second part of the work presents experimental benchmarks while testing this analyzator on different graphic cards. The benchmarks focuses on vertex throughput of each card for various rendering operations (classic triangles, vertex arrays, VBOs). The final part of the project deals with possible improvements of the developed analyzator.

Key words:

OpenGL, GLUT, benchmark, rasterization, rendering, graphic accelerator, vertex, vertex arrays, vertex buffer objects, VBO

Obsah

1. Úvod ... 7
2. OpenGL ... 8
2.1 Historická poznámka ... 8
2.2 Princip OpenGL ... 8
2.3 Zpracování dat v OpenGL ... 9
2.4 Úzká místa ovlivňující výkonnost OpenGL ... 10
2.5 Nadstavby OpenGL ... 10
2.6 GLUT ... 10
3. Měření výkonnosti OpenGL ... 11
3.1 Motivace ... 11
3.2 Základní myšlenka ... 11
3.3 Teoretická realizace ... 11
3.4 Praktická realizace ... 12
4. Popis programu ... 13
4.1 Funkce programu ... 13
4.2 Princip měření výkonu ... 13
4.3 Scény ... 14
4.4 Displaylisty, textury, světla ... 16
4.5 Minimalizace chyb ... 16
4.6 Statistiky – význam hodnot ... 17
4.7 Podpora pro gnuplot ... 18
5. Implementace ... 19
5.1 Funkce programu ... 19
5.2 Diagram tříd ... 20
5.3 Činnost programu ... 21
5.4 Popis jednotlivých tříd ... 22
5.5 Další poznámky ... 28
5.6 Zdrojové soubory ... 29
5.7 Pomocné skripty ... 31
6. Výkonnostní charakteristiky ... 32
6.1 Měření ... 32

6.2 Vyhodnocení..... 32
7. Další vývoj.....33
8. Závěr.....34
9. Literatura..... 35
10. Přílohy..... 36
10.1 Ovládání programu.....36
10.2 Výkonnostní charakteristiky..... 41
10.3 Porovnání výkonnostních charakteristik..... 94
10.4 Výkonnost Linux vs Windows pro GFFX5200..... 102

1. Úvod

Posledních patnáct let bylo poznamenáno nebývalým rozvojem pokročilých technik v oblasti grafického hardware – z grafických karet se stávají grafické akcelerátory. Grafické akcelerátory se definitivně oddělily od svých předchůdcům a nechaly je daleko za sebou. Dříve složité a prakticky nerealizovatelné projekty, se přesouvají do popředí zájmu. Teprve nové technologie umožnily snížit ceny produktů a otevřít tak trh široké veřejnosti.

V celé řadě odvětví se začínají uplatňovat náročné grafické aplikace (medicína, strojírenství, modelování, simulace, filmová produkce, herní průmysl atd.). Výrobci přicházejí s výkonnějšími akcelerátory a začíná se objevovat požadavek přesného měření výkonu umožňující objektivně zhodnotit kvality grafických akcelerátorů.

Námětem této práce je měření výkonu grafických akcelerátorů při použití knihovny OpenGL. Knihovna OpenGL je nejrozšířenější programové rozhraní k akcelerovaným grafickým kartám. Ve své podstatě se jedná vysoce optimalizovanou modelovací knihovnu orientovanou na 3D grafiku. Oblibu získala svou jednoduchostí, rychlostí a přenositelností.

2. OpenGL

2.1 Historická poznámka

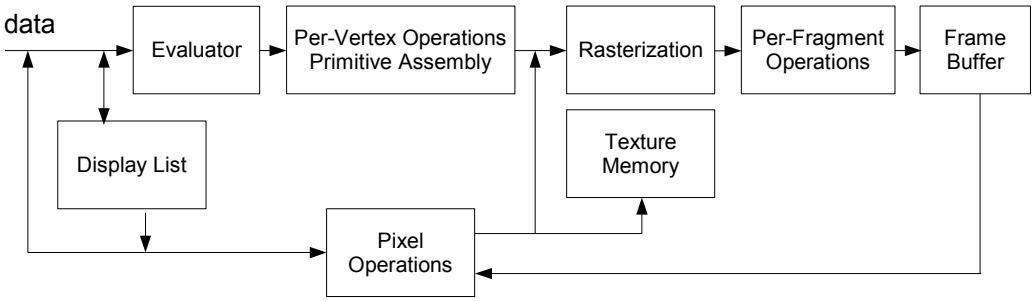
Knihovna OpenGL byla navržena firmou Silicon Graphics Inc. (SGI) jako programové rozhraní k akcelerovaným grafickým kartám. Předchůdcem byla programová knihovna IRIS GL od téže firmy, určená pro počítače se specializovaným hardwarem pro grafické operace. Když se ji SGI pokoušela rozšířit i na jiný hardware, vyvstaly problémy. Knihovna OpenGL je výsledkem snahy SGI o zvýšení přenositelnosti knihovny IRIS GL na různé typy grafických akcelerátorů.

2.2 Princip OpenGL

OpenGL je procedurální. Sestavení scény se děje prostřednictvím volání API funkcí (přibližně 120), které zahrnují vykreslovaní grafických primitiv (body, úsečky, polygony, bitmapy, pixmapy) v dvourozměrném a trojrozměrném prostoru. Kromě toho OpenGL podporuje osvětlování, stínování, texturování, animaci atd. OpenGL neposkytuje žádné funkce pro platformově závislé operace jakými jsou vykreslování oken, interakce s uživatelem, obsluha souborového systému. Za obsluhu těchto operací je zodpovědný systém na kterém program běží.

2.3 Zpracování dat v OpenGL

Níže uvedený diagram blokově zachycuje způsob jakým OpenGL zpracovává data. Data (*data*) vstupují z levé strany a procházejí výkonným řetězcem. Data jednak určují jaké geometrické objekty se mají vykreslovat a také jakým způsobem se s objekty v jednotlivých blocích pracuje.



Obr. 2.1: Schéma OpenGL

Data mohou vstupovat přímo do řetězce nebo mohou být dočasně uloženy v display listech (*Display List*) pro pozdější zpracování.

Evaluátor (*Evaluator*) má na starosti aproximaci křivek a povrchů ze vstupních parametrických dat. Na výstupu tohoto bloku se objevují informace o jednotlivých vrcholech.

Jednotka pro operaci s vrcholy a sestavování primitiv (P*er-Vertex Operations*, *Primitive Assembly*) zpracovává geometrická primitiva – body, úsečky, segmenty a polygony. Vrcholy jsou transformovány, je vypočteno osvětlení a vzniklá primitiva jsou ořezána ořezávací rovinou.

Rasterizační jednotka (*Rasterization*) generuje fragmenty, které dané primitivum pokrývá. Takto vytvořené fragmenty vstupují do posledního bloku pro zpracování fragmentů (*Per-Fragment Operations*). Blok provádí nad fragmenty finální operace (úpravy podmíněné předchozí hodnotou ze Z-bufferu, blending, maskování atd.) než vstoupí v podobě pixelu do framebufferu.

Do řetězce mohou kromě vrcholů vstupovat i rastrová data – bitmapy, pixmapy, textury. Tyto data vynechávají první fázi zpracování popsaného výše a jsou zpracovány v bloku pro zpracování pixelů (*Pixel Operations*). Výstup je uložen do paměti textur (na později) nebo vstupuje do rasterizační jednotky. Výsledný fragment je pak uložen do framebufferu jako by se jednalo geometrická data.

2.4 Úzká místa ovlivňující výkonnost OpenGL

Celková výkonnost jakéhokoliv zařízení je ve skutečnosti určena pouze úzkými místy (*bottleneck*). V případě OpenGL jsou úzkými místy:

1) aplikace – nedodává dostatečně rychle data do OpenGL

2) zpracování geometrických dat – OpenGL nestačí zpracovávat vrcholy

3) zpracování rastrových dat – OpenGL není schopna rychle rasterizovat grafická primitiva

Úzká místa hrají klíčovou roli při návrhu programů (her) orientovaných na 3D grafiku, kdy se hledá kompromis mezi požadavky a možnostmi grafického akcelerátoru. ([4])

2.5 Nadstavby OpenGL

Z hlediska datové reprezentace vykreslované scény poskytují funkce OpenGL pouze základní rozhraní pro přístup ke grafickým akcelerátorům. Existují však rozšiřující knihovny, které funkcionalitu dále zvyšují. Jednou ze základních knihoven používaných společně s OpenGL je knihovna GLU (OpenGL Utilities), která umožňuje využívat tesselátory (rozložení nekonvexních polygonů na trojúhelníky), evaluátory (výpočet souřadnic bodů ležících na parametrických plochách) a vykreslovat kvadriky (koule, válce, kužely a disky). Další nadstavbovou knihovnou je Open Inventor, pomocí kterého lze konstruovat celé scény složené z hierarchicky navázaných objektů. ([1])

2.6 GLUT

GLUT (OpenGL Utility Toolkit) je další nadstavbovou knihovnou OpenGL. GLUT definuje a implementuje aplikační rozhraní pro tvorbu oken a jednoduchého grafického uživatelského rozhraní, přičemž je systémově nezávislá tj. pro práci s okny se na všech systémech používají vždy stejné funkce, které mají stejné parametry. Do knihovny jsou také přidány funkce pro vykreslování bitmapového a vektorového písma v několika základních řezech. ([1])

3. Měření výkonnosti OpenGL

3.1 Motivace

Obecné měření výkonu je jediný způsob, jak kvantitativně popsat vlastnosti zkoumaného zařízení. Existuje více způsobů podle toho, čím se měření zabývá. Nejčastěji se zkoumá, které zařízení je z dané množiny zařízení celkově nejlepší nebo se měří výkon na jednom zařízení za účelem zjištění jeho úzkých míst (*bottleneck*).

3.2 Základní myšlenka

Hovoříme-li o měření výkonu grafických operací, pak máme namysli měření doby za jakou se určitá grafická operace provede resp. kolikrát se provede. Poměr obou hodnot pak udává výkon s jakým byla operace provedena.

Při měření se zpravidla sleduje počet vrcholů vykreslených za sekundu, počet pixelů vykreslených za sekundu nebo počet snímků vykreslených za sekundu v závislosti na velikosti vykreslovaného objektu.

3.3 Teoretická realizace

Následující úsek kódu naznačuje, jak může být měření výkonu realizováno v programu:

```
Time start = getTime();
drawScene();
glFinish();
Time elapsed = getTime() - start;
```

Obr. 3.1: Měření času

Na začátku kódu je zaznamenán aktuální čas, je provedena operace (např. v našem případě vykreslení scény, ale může to být jakákoliv jiná operace), následuje vynucené dokončení všech rozpracovaných úkonů (`glFinish()`) a na závěr je vypočítána celková doba provádění jako rozdíl aktuálního a dříve zaznamenaného času.

Analýza řešení

Je třeba si uvědomit, že takto naměřený čas není skutečný, ale zdánlivý. Ve výpočtu celkové doby se neuvažuje doba potřebná na zavolání funkce `getTime()`, neuvažuje se ani doba nutná

na zavolání kreslící funkce `drawScene()`. Má-li být měření výkonu přesné, musí být tyto doby do výpočtu zahrnuty.

3.4 Praktická realizace

Následující kód ukazuje praktickou realizaci měření času:

```
Time a = getTime();
Time delta = getTime() - a;
[…]
Time start = getTime();
drawComplexScene();
glFinish();
Time elapsed = getTime() – start - delta;
```

Obr 3.2: Zpřesnění měření času

Při inicializaci je nejprve zjištěna doba potřebná k vykonání funkce `getTime()` a uložena do proměnné `delta`. Je-li doba příliš malá, je vhodné provést měření pro více opakování a určit průměrnou hodnotu. Ve výkonném kódu je zaznamenán aktuální čas, je provedeno několik zkoumaných operací ve složitější scéně, následuje vynucené dokončení všech rozpracovaných úkonů (`glFinish()`) a na konec je vypočítána celková doba provádění jako rozdíl aktuálního a dříve zaznamenaného času zpřesněná dobou vykonávání funkce `getTime()`.

Analýza řešení

Uvedený kód nyní zpřesňuje měření dvěma způsoby:

1) odečítá dobu potřebnou pro vykonání funkce `getTime()` od celkového času

2) kreslí složitější scénu v které několikrát opakuje zkoumanou operaci

Předpokladem úspěšného zpřesnění je, aby doba kreslení složitější scény byla mnohonásobně vyšší než doba volání funkce `drawComplexScene()`. Za tohoto předpokladu bude doba volání zanedbatelná a neměla by se ve výsledku projevit. Problémem ovšem zůstává vhodná „složitá scéna“. Scéna vyžaduje další přídavný kód (proměnné, cykly...), který sice vytváří potřebnou složitost, ale opět vnáší do měření další nepřesnosti[1].

1 I provedení cyklu něco stojí.

4. Popis programu

Následující kapitolu je třeba chápat jako nenásilné seznámení s možnostmi programu. Program je neustále ve vývoji, proto popis nezabíhá do detailů a věnuje se činnosti programu obecně.

4.1 Funkce programu

Program umožňuje měřit a prověřovat výkonnost OpenGL několika testy/scénami. Každá scéna se zaměřuje na jinou oblast zobrazování např. vykreslování trojúhelníků, přepínání textur, přepínání materiálů atd. Kromě tohoto hrubého dělení je dále možné každou scénu doplnit osvětlením, nastavit blending, volit texturování, nechat vykreslit scénu pomocí displaylistu atd. V neposlední řadě je možné měnit počty a velikosti zobrazovaných objektů a dobu měření.

Testy lze spouštět samostatně nebo dávkově (též rychlotest). Dávkové spouštění se vyplatí při velkém počtu testů, kdy se program spouští pouze jednou oproti opakovanému spouštění v případě provádění testů samostatně. Dávkové spouštění spolu s pomocnými skripty[2] též umožňuje automatizované zpracovávání výsledných hodnot a jejich zobrazení pomocí volně šířeného nástroje `gnuplot` (www.gnuplot.info).

Program disponuje prostředky pro zpřesnění hodnot. Zpřesnění je založeno na mediánovém filtru, který vybírá z opakovaných měření (seřazených podle výkonu) prostřední hodnoty (další popis je v kapitole 4.5 na straně 16).

Všechny uvedené funkce a řada dalších jsou ovlivnitelné parametry z příkazové řádky při spuštění programu (přehled parametrů viz strana 36).

4.2 Princip měření výkonu

Měření výkonu spočívá v měření doby za jakou se zkoumaná operace provede popř. kolikrát se operace provede. Program sleduje

1) počet vrcholů vykreslených za sekundu,
2) počet pixelů vykreslených za sekundu a
3) počet snímků vykreslených za sekundu

2 Program byl původně vytvářen pro Linux.

v závislosti na velikosti vykreslovaného objektu. Co se týká posledního parametru, tak ten je na periferii zájmu, klíčové jsou první dva. Všechny naměřené hodnoty jsou uváděny v přehledné tabulce vypisované buď v průběhu nebo až na konci měření (viz kapitola 4.6 na straně 17).

4.3 Scény

Skladba scén[3] vznikla historickým a experimentálním vývojem. Atraktivní měření přetrvaly a méně zajímavější byly nahrazeny. V nejbližší době se dá se očekávat jejich další rozšiřování.

U všech scén je zásadně využívána ortogonální projekce. Ortogonální projekce má velkou přednost v tom, že nedeformuje vykreslované objekty vlivem jejich různé vzdálenosti od pozorovatele a tudíž lze bez problémů vypočítat počet vykreslovaných bodů pouze na základě rozměrů objektu. Počet bodů spolu s dobou vykreslování vstupuje do výpočtů.

4.3.1 Trojúhelníky

Scéna je určena pro měření doby vykreslování trojúhelníků a zjišťování propustnosti vertexů za sekundu.

Základním objektem je čtverec sestávající ze dvou nepřekrývajících se trojúhelníků. Kreslení začíná v levém horním rohu a postupně se klade čtverec vedle čtverce do zaplnění celé plochy. Po zaplnění plochy je změněna z-ová souřadnice (změna je nastavitelná) a kreslení opět začíná v levém horním rohu dokud není nakreslen požadovaný počet objektů.

Program podporuje 5 variant kreslení: klasické trojúhelníky (`GL_TRIANGLES`), pás trojúhelníků (`GL_TRIANGLE_STRIP`), trs trojúhelníků (`GL_TRIANGLE_FAN`), kreslení trojúhelníků pomocí pole vrcholů (`GL_VERTEX_ARRAY`) a vertex bufferu objektů (`GL_VERTEX_BUFFER_ARB`).

4.3.2 Osmistěny

Scéna je určena pro měření propustnosti vertexů za sekundu při zvolené světelné konfiguraci.

Základním objektem je pravidelný osmistěn sestávající ze dvou trojúhelníkových trsů z nichž jeden tvoří horní část a druhý spodní část. Každý trs tvoří 4 trojúhelníky. Způsob vykreslování okna je stejný jako u trojúhelníků navíc se osmistěny otáčí kolem své osy. Osmistěny z jednotlivých vrstev se protínají.

3 Nebo chcete-li „scénová základna“ popř. „scénový park“.

4.3.3 Přepínání textur

Scéna je určena pro měření doby přepnutí textury. Sama o sobě neměří dobu přepnutí, měří pouze dobu za jakou je scéna vykreslena jako celek v níž má každý objekt jinou texturu. Doba musí být vypočítána dodatečně vně programu.

Způsob vykreslování je shodný s vykreslováním klasických trojúhelníků s tím rozdílem, že každý objekt (čtverec) je opatřen jinou texturou.

Vyhodnocení doby přepnutí

Spolu se statistikami pro přepínání textur je nutné změřit dobu vykreslování téhož počtu objektů při stejném nastavení textur bez přepínání[4] (tj. všechny objekty mají stejnou texturu). Doba nutná k přepnutí je pak:

$$\Delta t = \frac{t_s}{f_s} - \frac{t_0}{f_0} = \frac{t_s f_0 - t_0 f_s}{f_s f_0} ,$$

kde t_s je doba za kterou se vykreslí f_s snímků při přepínání textur a t_0 je doba za kterou se nakreslí f_0 snímků bez přepínání textur. Pro velmi malé časy t_s a t_0 je vhodné zvolit delší měřící interval.

4.3.4 Přepínání materiálů a barev

Scéna je určena pro měření doby změny materiálu nebo barvy. Měnit barvu je možné s každým vrcholem, trojúhelníkem, objektem (čtvercem) nebo pouze jednou pro celou scénu[5].

Vykreslování je totožné s vykreslováním klasických trojúhelníků, mění se pouze barva a materiál jednotlivých vrcholů podle požadavků uživatele.

Vyhodnocení doby přepnutí

Způsob vyhodnocení je podobný s přepínáním textur. Jednou hodnotou je doba za kterou vykreslí scéna při přepínání materiálů/barev a druhou hodnotou je doba vykreslování scény bez přepínání materiálů/barev.

4 Pro měření bez přepínání textur se musí použít jiná scéna.

5 V současné verzi se mění barvy a materiály pouze pomocí `glColor()` a `glMaterial()`

4.4 Displaylisty, textury, světla

Displaylisty

Program umožňuje volitelně měřit výkonnost kreslení displaylistu. Celá scéna (trojúhelníky, osmistěny, ...) je nejprve „vykreslena“ do displaylistu a poté se měří doba vykreslování displaylistu. Samozřejmostí je inteligentní vytváření displaylistu tj. pouze když se mění scéna nebo rozměry objektů.

Textury

V současné době jsou k dispozici 4 druhy texturování. Objekty je možné nechat vykreslit bez textur, s jednou texturou, dvojicí multitextur a čtveřicí multitextur. Nastavitelné je i filtrování textur – parametry vychází z GLUT (popis parametrů viz strana 36).

Při zapnutých světlech je mísící funkce textury přepnuta z `GL_REPLACE` na `GL_MODULATE`. Další textury v případě multitextur mají mísící funkci `GL_BLEND`.

Světla

Program podporuje 8 plně uživatelsky nastavitelných světel. Lze nastavovat polohu, typ světla (bodové, směrové, reflektorové), útlum (konstantní, lineární, kvadratický) a barevné složky (ambientní, difúzní, odlesk).

Volitelně je možné dát světla do pohybu. Pohyb je docílen inkrementováním a dekrementováním x-ové a y-ové polohy světla. Překročí-li poloha okraj okna je inkrementování změněno na dekrementování a naopak[6]. S pohybem reflektorových světel se musí pracovat obezřetně, aby v okrajových oblastech nesvítil kužel mimo okno[7].

4.5 Minimalizace chyb

Během měření je program ovlivňován celou řadou zpravidla krátkých rušivých dějů (překreslí se konzolové okno, disk začne pracovat[8], pohne se myší atd.). Všechny tyto děje nepříznivě působí na přesnost naměřené doby.

K potlačení vlivu zmíněných dějů je možné s výhodou využít mediánový filtr. Při zapnutém mediánovém filtru se každé měření provede několikrát a všechny tyto dílčí výsledky putují

6 Světlo se „odráží“ od okrajů.

7 Pravděpodobně by došlo ke zkreslení výsledků.

8 Tradiční „zachrochtání“ disku po spuštění programu.

do bufferu. Po jeho naplnění jsou výsledky seřazeny a je vypsána prostřední hodnota (medián). Velikost bufferu je nastavitelná.

Mediánový filtr poskytuje dva druhy vyhodnocení v závislosti na způsobu opakování. Je možné

1) nechat opakovat jednotlivá měření (tj. jeden řádek ve statistice) nebo

2) zopakovat celý test.

První způsob je výhodný při měření výkonu displaylistu, kdy se displaylist kompiluje pouze v prvním měření při změně rozměrů a/nebo počtu objektů a pro další opakování se již nekompiluje. U tohoto způsobu hrozí nebezpečí zkreslení výsledků grafickou kartou, která teoreticky může provádět optimalizace[9]. Nevýhodou je, že jednotlivá měření („řádky“ statistiky) jsou vypisovány v průběhu měření a dochází k překreslování konzolového okna[10].

Druhý způsob by měl dostatečně eliminovat výše zmíněné „podobnostní“ optimalizace, ale pro změnu vyžaduje opakované kompilování displaylistu, které rovněž může ovlivnit měření. Na rozdíl od prvního způsobu vypisuje finální statistiky až na závěr.

4.6 Statistiky – význam hodnot

Program v průběhu nebo po skončení vypisuje naměřené statistiky. Statistiky mají dvě části. V první části jsou uvedeny informativní údaje pro uživatele o hardwaru (cpu, paměť, grafická karta, ovladač) a nastavení aktuálního testu (textury, blending, Z-buffer atd.). V druhé části jsou naměřené hodnoty. Následující řádky budou věnovány druhé (hodnotové) části a jejich významu.

Příklad výpisu pro test trojúhelníků:

```
#count wdth hght frame  t[ms]    fps   vps[mil/s]  pps[mil/s]
1000      2    2   615     98   6264.5   37.587     25.058
 950      6    6   595    126   4738.7   27.010    162.063
 900     10   10   543    201   2702.5   14.593    243.221
```

Komentář k výpisu

První řádek uvádí, že se vykreslilo 1000 objektů (1 objekt sestává ze 2 trojúhelníků – viz kapitola 4.3.1na straně 14) o velikost 2x2 pixely. Během měření se podařilo vykreslit 615 snímku

9 Přece jen kreslí se tatáž scéna se stejným počtem a rozměry objektů.

10 Překreslování konzolového okna lze předejít přesměrováním výstupu do souboru popř. do vhodného filtru.

za celkovou dobu 98 ms. Snímků, vrcholů a bodů vykreslených za sekundu je:

$$fps = frames / time = 6264.5$$
$$vps = frames \cdot count \cdot vertcount / time = 615 \cdot 1000 \cdot 6 / 0.098 = 37.587\text{mil}$$
$$pps = frames \cdot count \cdot width \cdot height / time = 615 \cdot 1000 \cdot 2 \cdot 2 / 0.098 = 25.058\text{mil}$$

4.7 Podpora pro gnuplot

Výstupní statistiky se dají bez dalších úprav použít jako vstup pro `gnuplot`. Informativní část je záměrně zakomentována a hodnotová část respektuje požadovaný vstupní formát pro `gnuplot`. Každé měření (*dataset*) je odděleno dvěma prázdnými řádky na začátku.

Při dávkovém měření je možné navíc obohatit parametry testů o příkazy pro `gnuplot`. Takto vzniklé finální statistiky načítá skript `chart.sh` a spolu s příslušným příkazem předává `gnuplot`. Popis formátu dávkového souboru je uveden na straně 39.

5. Implementace

Účelem této kapitoly je vyplnit „prázdné místo“, které vzniká mezi diagramem tříd a samotným zdrojovým kódem. Měla by přiblížit činnost programu a vysvětlit vztahy mezi jednotlivými třídami z pohledu programátora.

5.1 Funkce programu

Funkce programu byly obecně zmiňovány výše (kapitola 4.1 na straně 13), zde uvedu pouze ty, které jsou z pohledu implementace výjimečné. Mezi ně patří

1) samostatné a dávkové zpracování testů a

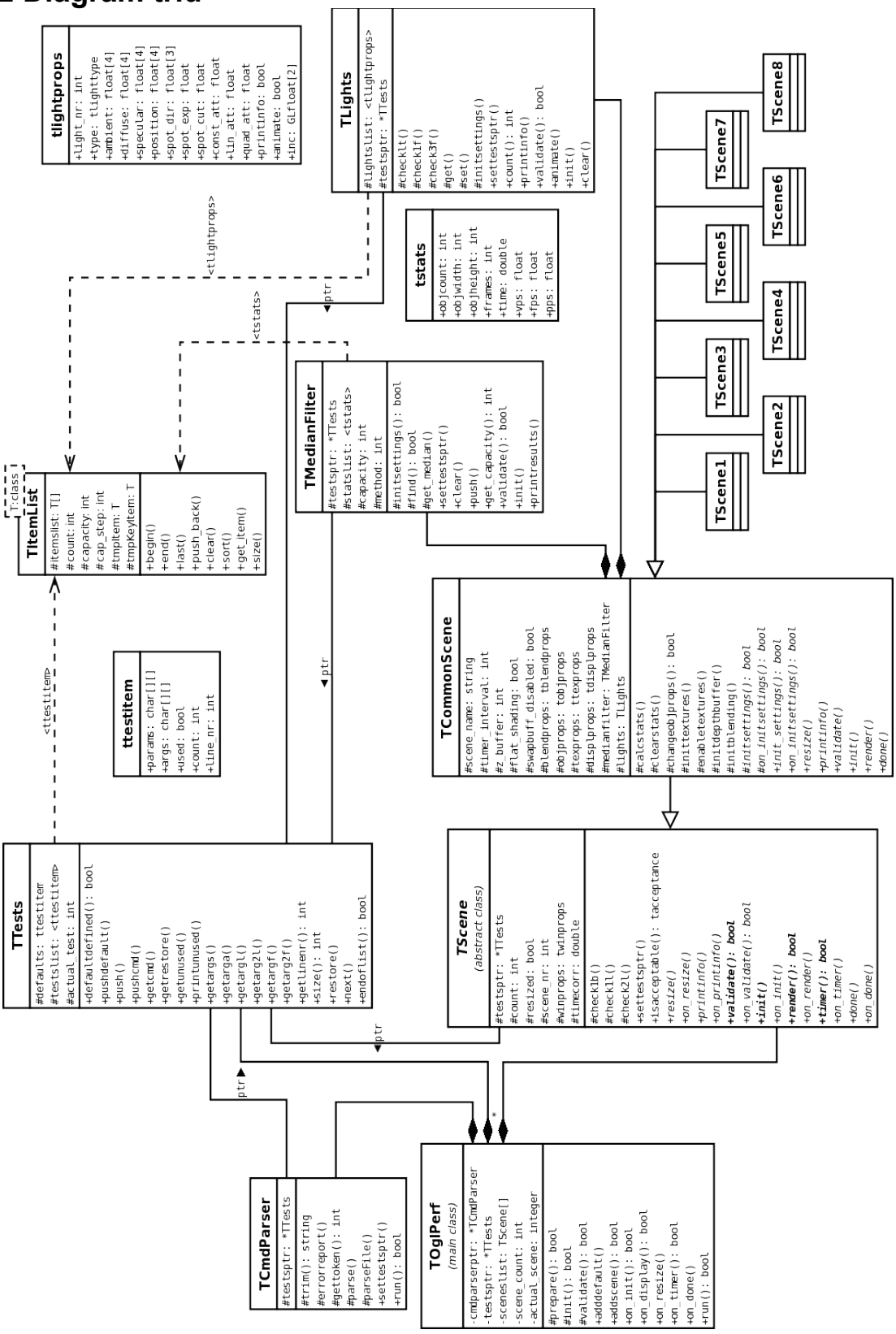
2) časový a snímkový interval měření.

Samostatné a dávkové zpracování klade jiné nároky na parsování (`TCmdParser`) a v konečném důsledku ovlivňuje i vzhled finálních výsledků[11] (`TCommonScene`).

Doba měření určená počtem snímků vyžaduje zvláštní zacházení při ukončování měřícího intervalu – ukončení nelze řešit čekáním na událost časovače (`TCommonScene`).

11 Při dávkovém zpracování jsou výsledky zpravidla doprovázeny příkazy pro `gnuplot`.

5.2 Diagram tříd



5.3 Činnost programu

Veškeré činnosti spojené s měřením zastřešuje objekt[12] `TOglPerf`. `TOglPerf` v sobě udržuje seznam (přesněji dynamické pole) všech dostupných scén. Každá scéna se vždy přidává v hlavním programu (soubor `main.cpp`) prostřednictvím metody `TOglPerf.addscene()`. Všechny scény dostávají přiděleno pořadové číslo. Do `TOglPerf` se též z hlavního programu přidávají implicitní hodnoty všech parametrů metodou `TOglPerf.adddefault()`. Tyto hodnoty se pak použijí v případě, že uživatel žádný parametr nezadá, nebo když parametr chybí.

V `TOglPerf` je vytvářen `TCmdParser` a `TTests`. `TCmdParser` parsuje příkazovou řádku případně načítá soubor (rychlotest) a parsuje jednotlivé řádky souboru. Řádku vždy rozdělí na dvojici parametr (jeden string) a argumenty (druhý string) a tuto dvojici ukládá do `TTests`, přičemž všechny parametry a argumenty z jednoho řádku příslušejí k jednomu testu – v `TTest` se tak vytvoří jeden záznam (pokud je v souboru více řádků, pak je i v `TTest` více záznamů). Dojde-li během parsování k chybě, je na to ihned upozorněno a parsování končí s chybou.

V hlavním programu je provedena obvyklá počáteční inicializace (`glutInit()`, `glutInitDisplayMode()`, `glutCreateWindow()`), čímž se GLUTovský automat uvede do startovního stavu a pozdější validace scén vychází z platných hodnot. Po inicializaci je zavolána `TOglPerf.run()`, čímž se spustí `TCmdParser` a provede se výše zmíněné parsování příkazové řádky. Proběhne-li vše v pořádku je zavolána `TOglPerf.validate()`, kde se provede kontrola parametrů všech požadovaných testů.

Kontrola parametrů probíhá tak, že ze seznamu všech testů v `TTests` se bere jeden test po druhém a hledá se k němu (z dříve vytvořeného seznamu scén) scéna, která je schopna aktuální test přijmout. Jinými slovy pořadové číslo scény, tak jak byly vytvářeny a přidávány do `TOglPerf`, se musí shodovat s číslem uvedeným na příkazové řádce. Rozhodnutí, zda scéna test přijme nebo odmítne, provádí metoda `TScene.isacceptable()`. Metoda současně vypisuje chybovou hlášku, je-li číslo scény mimo povolený rozsah.

Program nekončí po prvním neúspěchu, ale pokračuje dál v ověřování dalších testů (je to de facto zotavení z chyby běžné u kompilátorů, umožňující uživateli opravit více chyb najednou). Ze scény, schopné přijmou aktuální test, je volána `TScene.validate()`, která provede kontrolu všech parametrů. Objeví-li se v parametrech chyba, scéna vypíše chybovou hlášku a vrátit neúspěch.

12 Pojmy objektu a třídy v textu volně zaměňuji, proto tímto prosím ortodoxní programátory o shovívavost.

Tímto způsobem se projdou všechny testy a scény. Dopadne-li vše úspěšně, jsme zpět v hlavním programu (`main.cpp`), kde se dokončí inicializace GLUTu a zaregistrují se obslužné funkce. Těsně před nekonečnou smyčkou je volána `onInit()`, která připraví scénu pro první test. V `onInit()` se volá `TOglPerf.on_init()` a ta se již postará o vyhledáni správné scény k aktuálnímu testu. Od nalezené scény se spustí `TScene.init()`, kde se nastaví vše potřebné (počty objektů, blending, Z-buffer...). Jistým zjednodušením je, že `TScene.init()` se nemusí opětovně zkoumat správnost parametrů testu, protože už byly zkontrolovány ve `TScene.validate()` a může z parametru přímo načítat hodnoty.

Požadavky/události GLUTu (vykreslení okna, timer, změna velikosti okna...) jsou přijímány v hlavním programu a je vždy zavolána obslužná funkce z `TOglPerf`. `TOglPerf` pak zavolá příslušnou metodu z aktuální scény a ta se postará o zbytek. Scéna rovněž sama určuje, kdy se má spustit další test – to se děje v `on_timer()`.

5.4 Popis jednotlivých tříd

Popis se soustředí na funkci tříd a na jejich spolupráci. U každé třídy jsou zmíněny nejdůležitější metody v kontextu jejich činnosti. Popis parametrů je pak uveden ve zdrojových kódech.

5.4.1 TOglPerf

`TOglPerf` je nejdůležitější a jedinou třídou přes kterou je možné přistupovat k programu. Disponuje metodami pro přidávání nových scén (`addscene()`), pro definici implicitních hodnot všech[13] parametrů (`adddefault()`). Dále poskytuje metody pro obsluhu událostí jakými jsou inicializace (`on_init()`), překreslení okna (`on_render()`), změna rozměrů okna (`on_resize()`), událost od časovače (`on_timer()`) a konečně „uklízecí“ metodu pro korektní uvolnění paměti (`on_done()`). Všechny tyto události/metody volají odpovídající metody v aktuálně probíhající scéně.

Po přidání potřebných scén (potomků `TScene`) a definici implicitních hodnot parametrů jsou veškeré přípravné práce (parsování a kontrola parametrů) zahájeny metodou `run()`. Končí-li přípravy úspěšně je z hlavního programu (`main.cpp`) volána `onInit()` a tím je spuštěno měření.

V rámci této třídy jsou vytvářeny objekty `TCmdParser` (parsování parametrů), `TTests` (seznam parametrů všech testů). Třída spravuje seznam všech scén.

13 Některé speciální parametry jsou inicializovány na jiném místě (např. pro třídu `TLights`).

Zdrojový kód třídy je v `oglperf.h` a `oglperf.cpp`.

5.4.2 TTests

`TTests` je další velmi důležitou třídou sloužící jako úložiště implicitních hodnot parametrů, parametrů jednotlivých testů a také příkazů pro automatizované zpracování naměřených hodnot. Data jsou do úložiště dodávány dvěma způsoby. Implicitní hodnoty jsou dodávány z hlavního programu (`main.cpp`) prostřednictvím `TOglPerf.adddefault()`. Parametry testů a příkazy výhradně přidává `TCmdParser`.

Třída poskytuje celkem 3 metody pracující se 3 různými seznamy pro ukládání parametrů. Implicitní hodnoty parametrů jsou ukládány metodou `pushdefault()` do seznamu `defaults` ke kterému se přistupuje pouze v případě, že není možné nalézt hodnotu parametru v parametrech testu (tj. když uživatel hodnotu nezadá). Parametry testů jsou směřovány metodou `push()` do seznamu `testslist`, z něj čte hodnoty aktuální scéna před zahájením činnosti. Příkazy pro dávkové zpracování jsou ukládány metodou `pushcmd()` do seznamu `cmdlist()`. Obsah je posílán na výstup spolu s výsledky, přičemž se tisknou pouze příkazy (jsou-li nějaké) pro prováděný test (popis dávkového souboru viz strana 39).

Třída poskytuje 6 metod pro čtení hodnot parametrů ve scénách. Podle druhu parametru je možné číst string (`getargs()`), pole stringů (`getarga()`), jednu nebo dvě celočíselné hodnoty (`getargl()`, `getarg2l()`) a jednu nebo dvě reálné hodnoty (`getargf()`, `getarg2f()`). Nejuniverzálnějším způsobem, ale také nejnáročnějším pro další zpracování, je načítání pole stringů[14].

Objekt `TTests` vytváří `TOglPerf` a ukazatel na něj šíří do všech objektů (viz metoda `settestsptr()` u jednotlivých tříd). Každý objekt tak má přístup ke svým parametrům.

Zdrojový kód třídy je v `test.h` a `tests.cpp`.

5.4.3 TCmdParser

`TCmdParser` plní funkci parseru parametrů příkazové řádky popř. parametrů testů uvedených v souboru pro dávkové zpracování (formát souboru viz strana 39).

Klíčovou metodou je metoda `run()`, které jsou předány parametry příkazové řádky. Podle parametrů rozhodne, jestli se bude načítat ze souboru. Zjištěné parametry předává objektu

14 Využívá se u `TLights` a `TMedianFilter`.

TTests.

Zdrojový kód třídy je v cmdparser.h a cmdparser.cpp.

5.4.4 TScene

TScene je abstraktní třída poskytující nejzákladnější společné funkce pro ovládání scén. Řada atributů je statických (globálních), díky čemuž ovlivňuje všechny scény současně.

Třída má přehled o celkovém počtu vytvořených instancí (statický atribut count) a na základě něj přiděluje každé scéně unikátní číslo (atribut scene_nr). Implementuje metodu isacceptable(), která rozhoduje podle unikátního čísla scény a hodnoty parametru požadované scény, zda scéna test přijímá. Metodu využívá TOglPerf.validate().

TScene poskytuje svým potomkům metody check1b(), check1l(), check2l() pro snadnější načítání hodnot parametrů z TTests včetně kontroly rozsahu s výpisem případných chybových hlášek.

Dále deklaruje řadu metod pro obsluhu událostí, které jsou určeny k předefinování v potomcích. Za zmínku stojí metoda resize(), která obnovuje ve statickém atributu winprops aktuální velikost okna a nastavuje příznak resized. Je ve vlastním zájmu potomků, aby reagovaly na tuto skutečnost a nulovaly příznak (řešeno v TCommonScene).

V konstruktoru třídy se zjišťuje průměrná doba trvání volání funkce getTime().Změřená hodnota je uložena do atributu timecorr. Sama třída hodnotu nikde nepoužívá, je připravena pro potomky (využívá ji TCommonScene::on_render()).

Zdrojový kód třídy je v scene.h a scene.cpp.

5.4.5 TCommonScene

TCommonScene je potomkem abstraktní třídy TScene. Vytváří pro své potřeby objekty TLights a TMedianFilter. Předefinovává řadu metod předka a doplňuje tak veškerý kód nezbytný pro měření, obsluhu událostí, nastavování vykreslování a kontrolu správnosti parametrů. Všechny dosavadní scény (celkem 8) jsou odvozené právě z této třídy a ve většině případů pouze předefinovávají inicializační metodu on_init() a vykreslovací metodu on_render().

Třída je zodpovědná za kontrolu parametrů metodou validate() a podle nich nastavuje

vykreslování metodou `init()` spolu s metodami `inittextures()`, `initdepthbuffer()`, `initblending()`. Kontrola parametrů a nastavování týkající se světel jsou přenechány třídě `TLights`.

`TCommonScene` se dále se stará o měření doby vykreslování v metodě `render()`. Tatáž metoda určuje konec měřícího intervalu v případě, že je délka měřícího intervalu stanoven maximálním počtem snímků.

Metoda `timer()` je volána při ukončení měřícího intervalu. K ukončení intervalu může dojít buď po uplynutí požadovaného časového intervalu (událost od časovače v GLUTu) nebo na žádost metody `render()`, je-li doba měření určena počtem snímků. V obou případech se o zavolání metody stará `TOglPerf`. Metoda `timer()` předává naměřené hodnoty objektu `TMedianFilter`, jehož činností je sběr a výpis výsledků. Podle zaplněnosti mediánového filtru rozhoduje o ukončení měření a výpisu hodnot, opakování měření se stejnými parametry nebo o změně parametrů (parametry jsou myšleny počty a velikosti objektů). Dojde-li během měření ke změně velikosti okna (`resized()`), jsou při nejbližším volání `timer()` naměřené hodnoty zahozeny a měření se pro stejné nastavení opakuje.

Před zahájením měření třída tiskne informace o procesoru, paměti, grafické kartě a ovladači spolu s aktuálním nastavení testu (metoda `printinfo()` volaná z `ToglPerf::on_init()`). Aktuální nastavení se čte přímo z GLUTu, takže odráží skutečně nastavené parametry[15].

Zdrojový kód třídy je v `scenecom.h` a `scenecom.cpp`.

5.4.6 TLights

Úkolem `TLights` je správa světel počínaje načítáním požadovaných parametrů z `TTests` až po jejich povolování, nastavování poloh a parametrů vykreslování.

Třída podobně jako `TCommonScene` je zodpovědná za kontrolu parametrů metodou `validate()` a podle nich nastavuje vykreslování metodou `init()`. Tiskne aktuální nastavení světel (`printinfo()`).

15 Je to jakási zpětná kontrola, že je nastaveno to, co má být nastaveno.

Třída u zvolených světel umožňuje automaticky měnit polohu světel metodou `animate()`. Metoda je volána z `TCommonScene::render()` a u všech světel, které se mají pohybovat mění polohu inkrementováním/dekrementováním jedničky k/od x-ové resp. y-ové souřadnice světla. Překročí-li poloha okraj okna je inkrementování změněno na dekrementování a naopak.

Zdrojový kód třídy je v `lights.h` a `lights.cpp`.

5.4.7 TMedianFilter

`TMedianFilter` slouží jako prostředek ke zpřesnění naměřených hodnot a minimalizaci rušivých dějů. Provádí sběr a tisk naměřených hodnot z opakovaných měření (viz kapitola 4.5 na straně 16). Třída podporuje dva způsoby opakování a to buď opakování jednoho měření nebo opakování celého testu (druhy měření jsou podrobněji popsány je v kapitole 4.5 na straně 16).

Třída je podobně jako `TCommonScene` zodpovědná za kontrolu parametrů metodou `validate()` podle nichž nastavuje parametry sběrného bufferu `init()`.

Data jsou do mediánového filtru dodávány z metody `timer()` objektu `TCommonScene`. Třída na základě počtu a rozměrů objektů určuje zda bude měření zopakováno nebo zda mohou být počet a/nebo rozměry objekty změněny a v neposlední řadě určuje konec měření (`push()`).

Zdrojový kód třídy je v `median.h` a `median.cpp`.

Činnost objektu

`TMedianFilter` je zodpovědný za korektní opakování[16] měření nebo celého testu. Na základě již provedených měření a zvolené metody opakování musí určit další postup tj. vyžádat opakování stávajícího měření, vyžádat změnu počtu/velikostí objektu nebo ukončit test a nechat vypsat finální výsledky.

Je-li požadováno opakování pouze jednoho měření a ne celého testu, je situace poměrně snadná. Mediánový filtr požaduje opakování dokud není buffer zaplněn. V okamžiku zaplnění vydá povel k tisku hotových výsledků volající metodě (tou je `TCommonScene::timer()`) a je zahájen nový test.

Složitější situace nastává v případě, že má být opakován celý test. Mediánový filtr dopředu nezná počet měření prováděného testu a tudíž ani nemůže během prvního testu vydat povel k opakování.

16 Samozřejmě pokud je opakování požadováno tj. velikost mediánového filtru je alespoň 2.

Filtr k rozhodnutí potřebuje, aby byl celý test proveden minimálně 2 krát[17], přičemž druhé spuštění musí být zařízeno ve spolupráci s volající metodou (`TCommonScene::timer()`). Problém byl vyřešen doplněním nové položky `stat_nr` do struktury `tstats` (struktura v sobě nese právě naměřené hodnoty – časy, snímky, vps, pps, fps) určující pořadové číslo sekvence. Výsledný efekt je ten, že celý test (tedy několik měření) má stejné číslo. Další test má číslo o jedničku vyšší atd. Ukončení opakování se pak určuje podle tohoto čísla. Číslo používají a spravují metody `find()` a `push()`. Princip vyplyne z následujícího popisu.

Popis vybraných metod

`bool find(tstats item, bool * first_in_seq, bool * last_in_seq)`
Metoda hledá v dosud vykonaných měřeních (v bufferu) položku jejíž počet objektů a rozměry se shodují s položkou `item`. (pozn. `item` je nová hodnota, která má být do mediánového bufferu zanesena) Pokud takovou nalezne, tak se dále zkoumá číslo sekvence `stat_nr` předchozí a následující položky. Na základě ní se určí, zda je dotazovaná položka `item` na začátku (`first_in_seq`) nebo na konci měřené sekvence (`last_in_seq`). Nepodaří-li se žádnou vhodnou položku najít, je oznámeno neúspěšné hledání a položka není považována ani za počáteční a ani za koncovou. Metoda je využívána v `push()`.

`tmedianrequest push(tstats item)`
Metoda ukládá nově příchozí měření `item` do bufferu. Nejprve pomocí `find()` určí o jaké měření (o jakou položku) vzhledem k seznamu se jedná (měření začínající novou sekvenci nebo měření končící předchozí sekvenci) a podle výsledku buď ponechá nebo zvýší číslo sekvence `stat_nr`. Měření je pak uloženo do bufferu.

Metoda, po uložení nového měření, určuje podle zaplněnosti bufferu, čísla sekvence a typu posledního měření další operaci. Může si vynutit

1) opakování aktuálního měření se stejnými vlastnostmi objektu (`mrRepeatProps`),
2) změnu vlastností objektu s případným ukončením testu, pokud velikost nebo počet překročí stanovenou mez (`mrChangePropsAndExit`),
3) opakování celého testu z počátečních hodnot objektu (`mrRestartProps`),
4) tisk výsledků, protože měření nebo test jsou u konce (`mrPrintResults`) nebo

17 Toto je důvod, proč druhá „opakovací“ metoda vyžaduje minimální velikost bufferu 2.

5) změnu vlastností objektu s případným opakování testu, pokud velikost nebo počet překročí stanovenou mez (`mrChangePropsAndRestart`).

Správné nastavování velikosti a počtu objektů stejně tak jako tisk výsledků řídí volající metoda `TOglPerf::timer()`.

`printresults()`
Metoda určí správné pořadí řazení jednotlivých sloupců a seřadí hodnoty jednotlivých měření nejprve podle počtu a velikosti objektu a poté seřadí vzniklé souvislé bloky podle hodnoty vrcholů za sekundu (vps). Ze souvislých bloků vybírá a tiskne prostřední hodnotu (medián) jakožto finální výsledek.

5.4.8 TItemList

`TItemList` je šablonovou třídu (*template*) pro vytváření seznamu prvků. Poskytuje základní metody pro přidávání, odebírání a výběr prvků. Jedná se o výrazně zjednodušenou variantu třídy známé z STL (*Standard Template Library*).

Zdrojový kód třídy je v `list.h` a `list.cpp`.

Popis vybraných metod

`void sort(bool (*comp)(T*a, T*b),long int begind,long int endind)`
Metoda řadí (quicksort) celý seznam nebo jeho část určenou dvěma indexy. Poněvadž seznam může obsahovat prvky libovolného typu (i složitější struktury), metoda vyžaduje funkci, která provede porovnání dvojice takovýchto prvků. Řazení části seznamu využívá `TMedianFilter`, který řadí dvouprůchodově (viz kapitola 5.4.7 na straně 26).

5.5 Další poznámky

Žádná ze tříd (kromě `TItemList`) není vybavena přiřazovacím a kopírovacím konstruktorem, protože se jejich použití nepředpokládá. Aby i přesto nedošlo omylem k jejich uplatnění, protože C++ je vytváří implicitně, jsou deklarovány jako prázdné funkce v části `private` objektu. Pokud se někde nedopatřením kopírovací konstruktor vyskytne (např. při předávání objektu hodnotou), tak díky tomuto triku bude program nezkompilovatelný a chyba bude ihned odhalena. ([5])

5.6 Zdrojové soubory

Celý program sestává z 31 souborů[18] z nichž 3 jsou původními hlavičkovými soubory OpenGL. Zbývající soubory (nebo dvojice `.h` a `.cpp`) vždy implementují jeden logický celek.

`defs.h`
V souboru jsou uvedeny konstanty vyskytující se v programu – názvy parametrů, povolené rozsahy hodnot, implicitní hodnoty parametrů a chybové hlášky.

`glext.h, glxext.h, wglext.h`
Původní hlavičkové soubory OpenGL dovážející rozhraní rozšíření verze 1.1 a vyšší pro lepší přenositelnost na Windows. Hlavičky jsou použity v `draw.h/cpp`.

`draw.h/cpp, octahedr.inc, texswap.inc, triangle.inc, trifan.inc, trimatcol.inc, tristrip.inc`
Zde jsou umístěny veškeré kreslící a pomocné inicializační funkce. Kreslící funkce se zpravidla vyskytují s drobnou obměnou ve více variantách (např. bez textury, s texturou, s multitexturou), proto je společná výkonná část extrahována do souboru a měněná část doplněna podmíněným překladem pomocí maker. Soubor je pak direktivou `include` vložen do příslušné funkce.

`median.h/cpp, oglperf.h/cpp, lights.h/cpp, scene.h/cpp, cmdparser.h/cpp, scenecom.h/cpp, tests.h/cpp, list.h`
Zdrojové kódy všech tříd. Bližší popis činnosti viz kapitola 5.4 na straně 22.

`main.cpp`
Toto je hlavní program. Vytváří scény (potomky `TScene`), provádí inicializaci GLUTu a registruje zpětná volání, z nichž jsou volány obslužné metody hlavní třídy `TOglPerf`.

`common.h/cpp`
Soubor implementuje pomocné funkce jakými jsou: přesný čas, práce s dynamickým polem, ...

`help.h/cpp`
Zde je implementována jediná funkce starající se o výpis nápovědy. Funkce přijímá parametry příkazové řádky a je-li požadována nápověda, provede její výpis. Všechny názvy parametrů a implicitních hodnot (a na pár výjimek) v nápovědě jsou vypisovány z konstant definovaných v `defs.h`. Případná změna názvu nebo hodnoty parametru se tak ihned projeví v nápovědě, aniž

18 Soubor `.h` a k němu odpovídající `.cpp` jsou počítány jako jeden.

by bylo nutné do nápovědy zasahovat. Funkce umožňuje výpis nápovědy jako běžný text nebo jako manuálovou stránku.

`scene1.h/cpp, scene3.h/cpp, scene4.h/cpp`
Scény kreslí klasické trojúhelníky (`GL_TRIANGLES`), pásy (`GL_TRIANGLE_STRIP`) a trsy trojúhelníků (`GL_TRIANGLE_FAN`). Třídy jsou odvozeny od `TCommonScene` a předefinovávají metody `on_init()` a `on_render()`. V `on_init()` se nastavují parametry vykresleného objektu (`obj_props`) a určuje se, která funkce z `draw.h/cpp` bude provádět vykreslování (bez textury, s texturou, …). Metoda `on_render()` provádí vykreslování a současně měří dobu vykreslování.

`scene2.h/cpp`
Scéna kreslí otáčející se osmistěny pro test osvětlení. Třída je odvozena od `TCommonScene` a předefinovává metody `on_init()` a `on_render()` a přidává pomocnou metodu `rotate()`. V `on_init()` se nastavují parametry vykresleného objektu (`obj_props`) a určuje se, která funkce z `draw.h/cpp` bude provádět vykreslování (bez textury, s texturou, …). Metoda `on_render()` pootáčí osmistěnem, vykresluje scénu a měří dobu. Vrcholy osmistěnu jsou uloženy v poli a před každým vykreslením jsou metodou `rotate()` přepočítány do nové pootočené polohy.

`scene5.h/cpp`, `scene8.h/cpp`
Scény kreslí klasické trojúhelníky (`GL_TRIANGLES`) pomocí pole vrcholů (*vertex array*) a vertex bufferů (*vertex buffers*). Třídy jsou odvozeny od `TCommonScene` a předefinovávají metody `on_init()`, `on_render()`, `on_timer()`, `on_resize()` a `on_done()` a přidávají pomocné metody `initarrays()` a `loadarrays()`. Starají se o veškerou inicializaci a vykreslování – nevolá se žádná funkce z `draw.h/cpp`.

Třída vytváří 3 pole pro vrcholy, barvy a textury. Inicializaci všech polí provádí `initarrays()` a `loadarrays()` načítá tyto pole do grafické karty. Metoda `initarrays()` bere v úvahu aktuální velikost objektu a velikost okna a podle nich počítá vrcholy. Dojde-li ke změně velikosti okna (`on_resize()`) nebo objektu (`on_timer()`) jsou pole přepočítána a opětovně načtena do grafické karty.Metoda `on_render()` provádí vykreslování a současně měří dobu vykreslování.

`scene6.h/cpp`

Scéna kreslí klasické otexturované trojúhelníky (`GL_TRIANGLES`), přičemž každá dvojice trojúhelníků (objekt) je pokryt jinou texturou. Třída je odvozena od `TCommonScene` a předefinovává metody `on_init()` a `on_render()`. V `on_init()` se nastavují parametry vykresleného objektu (`obj_props`) a určuje se, která funkce z `draw.h/cpp` bude provádět vykreslování (bez textury, s texturou, …). Metoda `on_render()` provádí vykreslování a současně měří dobu vykreslování.

`scene7.h/cpp`

Scéna kreslí klasické trojúhelníky (`GL_TRIANGLES`) s nastavitelnou změnou barvy nebo materiálu pro každý objekt, trojúhelník či vrchol. Třída je odvozena od `TCommonScene` a předefinovává metody `on_validate()`, `on_init()` a `on_render()`. Metoda `on_validate()` dodatečně načítá a kontroluje specifické parametry pro tuto scénu.
V `on_init()` se nastavují parametry vykresleného objektu (`obj_props`) a určuje se, která funkce z `draw.h/cpp` bude provádět vykreslování (změna objektu, trojúhelníku, ...). Metoda `on_render()` provádí vykreslování a současně měří dobu vykreslování.

5.7 Pomocné skripty

Součástí programu je skript `chart.sh` usnadňující prohlížení a exportování naměřených grafů do souboru při dávkovém zpracování. Za tímto účelem je proto nutné v dávce spolu s parametry testů uvádět i sekvenci příkazů pro `gnuplot`. Příkazy jsou označeny `!` (vykřičníkem) na začátku řádku za nímž následuje konkrétní příkaz. Ve výsledných statistikách jsou příkazy odlišeny `##` (dvě mříže) na začátku řádku od ostatních údajů. Skript vybírá označené řádky (příkazy) a ukládá je do dočasného souboru, který je pak předán `gnuplot`u.

Podporovanými výstupy do souboru jsou formáty png (*Portable Network Graphics*) a ps (*PostScript*). Výstup se přesměruje automatickým přidáním příkazu `set terminal` na začátku dočasného souboru. Uživatel se tak nemusí o nic starat.

Skript poskytuje několik maker, kterými lze nahradit v době vytváření dávky neznámé hodnoty. Ve většině případů se jedná o nahrazování makra konstantou (název souboru, číslo sloupce ve statistice). Jediným makrem vyžadujícím zvláštní zacházení je `__TESTNR__`, které zastupuje číslo aktuálního grafu (*dataset*). Skript při průchodu výslednými statistikami počítá dvojice prázdných řádků oddělujících jednotlivé grafy (pozn. první dva prázdné řádky se nezapočítávají).

6. Výkonnostní charakteristiky

Kapitola zmiňuje hlavní oblasti, na které se testy zaměřují, a způsob vyhodnocení výsledků, naměřené charakteristiky jsou pak uvedeny v příloze na straně 41.

6.1 Měření

Program disponuje poměrně širokým nastavením vykreslování, které vede k velmi velkému množství různých měření. Není v lidských silách jednoho člověka prověřit a zanalyzovat všechny kombinace[19], proto byly na různých kartách prověřovány „pouze“ tyto oblasti:

1) displaylisty a Z-buffer
2) vertex arrays, vertex buffer objects
3) pásy trojúhelníků (*strip*)
4) typická scéna (displaylist, 1 textura)
5) přepínání textur
6) světla
7) přepínání barev a materiálů
8) porovnání Windows vs Linux

6.2 Vyhodnocení

U všech uvedených testů byl v rámci minimalizace rušivých vlivů používán mediánový filtr s kapacitou 20 měření a všechny výstupy byly směrovány do souboru. Měřící interval byl stanoven na 10 snímků.

Testy byly prováděny dávkově (rychlotestem) a jednotlivá měření doplněny sekvencí příkazů pro `gnuplot`. Výsledné charakteristiky byly vyexportovány skriptem `chart.sh` do souboru formátu png.

19 Ne všechny kombinace mají nějaký praktický smysl.

7. Další vývoj

Nutnost stanovení vhodné metriky pro objektivní porovnávání charakteristik. Porovnávat maximální (tj. první) hodnotu v měřeném testu je nedostatečná a výrazně zkreslující (viz GF2 a GF3). Metrika musí uvažovat tvar charakteristiky vzhledem k velikosti strany objektu.

Potřeba lepšího zpřesňování naměřených časových intervalů uvažující časy spotřebované na volání veškerých pomocných funkcí a opakování cyklů – nyní se uvažuje pouze doba volání měřící funkce `getTime()`.

Měření výkonnosti pruhů (*strip*) a trsů (*fan*) trojúhelníků tak jak je vyrobeno není příliš ideální. Nyní se kreslí pouze čtverec sestávající ze dvou trojúhelníků, což se podle změřených charakteristik nevyplatí (viz měření GF4TI4200 na straně 52). Aby se mohla uplatnit nějaká úspora, muselo by se jich v jeden okamžik kreslit více – nejlépe nastavitelný počet.

Kreslení za pomocí vertex arrays a vertex buffer objects vyžaduje další vývoj a testy nejen pro klasické trojúhelníky, protože dosažené výsledky neodpovídají očekávání. S jistotou nelze vyloučit ani chybu v implementaci.

Současné testy měří vždy úzkou oblast z množiny všech možných operací (ať už jsou to trojúhelníky, vertex arrays, ...) a vše na ortogonální projekci. Zajímavá z pohledu měření a atraktivní z pohledu uživatele by byla scéna s „reálnou“ krajinou v perspektivní projekci (dům, strom, obloha, …), kde by se otestovala řada operací (trojúhelníky, displaylisty, přepínání textur, vertex arrays, …) v jeden okamžik. Statistiky by tak odrážely reálný výkon za běžných podmínek.

Dalším vylepšením by mohlo být uživatelské rozhraní pro zakládání nových testů nebo sestavování několika testů z již vytvořených. Otázkou je, jestli by si to nevyžádalo vytvořit jiný program. Problematická by byla údržba – při každé změně měřiče by musel být upraven i návrhář testů.

Vylepšit lze i skript pro automatické vytváření grafů. Skript by kromě dosavadních maker, kdy se pouze nahrazuje makro konstantou, mohl poskytovat makra pro údaje z informativní části statistik (videokarta, ovladač, procesor, název scény, měřící interval atd.).

Ideálním řešením by byl informační systém s výsledky měření pro různé testy většího množství grafických karet a uživatel by si dotazem volil kombinace testů a zkoumal různá hlediska. Systém by musel umět základní operace pro práci s těmito daty (signály) jako jsou sčítání, odčítání apod.

8. Závěr

V rámci semestrálního projektu byl vytvořen a odladěn program pro měření výkonnostních charakteristik grafických karet využívající GLUT. Program nevznikl „na zelené louce“, ale prošel jistým vývojem u něhož jsem měl tu čest být od samého začátku. Dovolím si tedy věnovat pár slov jeho historii.

První verze programu byla vyvinuta z neobjektového prototypu, který měl sloužit pouze k otestování základních kreslících funkcí a ověření správnosti statistik. Prototyp byl, bohužel, dokončen jako finální program a postupným přidáváním dalších funkcí se z něj stal nepřehledný monolit. Není tudíž žádným překvapením, že se záhy objevil požadavek na přehlednost a snadnou rozšiřitelnost, který si vynutil objektový přístup. A tak vznikla druhá verze – tento program.

V první řadě bylo snahou navrhnout snadno pochopitelný a do budoucna snadno rozšiřovatelný program. To se, doufám, podařilo vyřešit dostatečně a přidávání nových funkcí či scén by nemělo činit potíže. Dalším požadavkem bylo odstranění otrocké práce, jak se zpracování grafů, tak i se samotnou detekcí hardwaru. Jak dalece se povedlo splnit druhý požadavek, nechť posoudí uživatel.

Na naměřené výkonnostní charakteristiky uvedené v příloze je třeba nahlížet jako na experimentální. Charakteristikám sice byla věnována zvýšená pozornost, ale zdaleka nemusejí odpovídat skutečnosti (v programu mohou být chyby). Ověřování správnosti naměřených hodnot je problematické, protože žádné zaručeně správné měření jsem neměl k dispozici. Nedá se ani stoprocentně spoléhat na výsledky uváděné výrobcem. Uváděné výsledky mají mnohdy příchuť reklamního sci-fi.

O přesnosti měření lze říct totéž, co u správnosti naměřených hodnot. I když se snažím výsledky zpřesnit mediánovým filtrem, měření se stále nedostává z vlivu klíčových faktorů jakými jsou ovladač a operační systém. Z charakteristik je alespoň spolehlivě vypozorovatelný trend, že propustnost vertexů s rostoucí stranou objektu klesá (do čehož dále vstupují textury, světla atd.). Nutno podotknout, že naměřené charakteristiky jsou nejlepší možné – program nedělal nic jiného než vykreslování. V reálném programu (ve hře), kde se provádí řada dalších operací s vykreslováním nesouvisejících, se nemusí takového výkonu dosáhnout.

Co říci na závěr? Chtěl bych touto cestou poděkovat všem, kteří se na předchozí verzi programu podíleli. Původní kód sloužil jako inspirace a poučení.

9. Literatura

[1] P. Tišnovský. *Grafická knihovna OpenGL*, www.root.cz, 2003.

[2] Silicon Graphics. *OpenGL Reference Manual*, Silicon Graphics, Inc., 1994.

[3] R. S. Wright. *OpenGL SuperBible*, Waite Group Press, 1999.

[4] D. Shreiner. *Performance OpenGL*, www.performanceopengl.com, 2003.

[5] S. Prata. *Mistrovství v C++*, Computer Press Brno, 2001.

10. Přílohy

10.1 Ovládání programu

Veškeré ovládání programu se odehrává na příkazové řádce. Testy lze provádět samostatně nebo dávkově (rychlotest). Spouští se jednou z těchto syntaxí:

`oglperf` [parametry]

`oglperf` soubor

Při samostatném provádění testů jsou všechna požadovaná nastavení pro jeden test provedena prostřednictvím parametrů z příkazové řádky. Program podporuje krátkou a dlouhou verzi parametrů, jejich popis je uveden níže. Je-li program spuštěn bez parametrů, jsou použity implicitní hodnoty.

Dávkové spouštění umožňuje provést více měření na jedno spuštění programu, vyžaduje však textový soubor s popisem jednotlivých testů (formát je uveden dále).

10.1.1 Parametry programu

`-h`, **`--help`**

Vytiskne nápovědu k programu jako přehled všech parametrů se stručným popisem jejich funkce.

`-hm`, **`--helpman`**

Vygeneruje manuálovou stránku s nápovědou.

`-t`, **`--timer-interval`** `n`

Nastavuje délku měřícího intervalu na n. Kladná hodnota představuje časový interval v milisekundách a záporná hodnota určuje délku měření v počtech snímků. Po uplynutí doby n milisekund nebo po vykreslení n snímků je zahájeno další měření.

`-r`, **`--window-size w h`**

Nastavuje šířku w a výšku h okna programu. Do hodnoty se nezapočítává okraj okna.

`-g`, **`--median`** `[n] [method] [`**`d`**`|`**`debug`**`]`

Určuje velikost mediánového filtru `n` a způsob opakování měření `method` (`a` – měření je opakováno `n` krát, `b` – celý test je opakován `n` krát). Výstup lze doplnit i ladícím výpisem `d` nebo `debug`, který obsahuje všechny naměřené hodnoty. Pořadí parametrů není rozhodující.

`-a`, **`--init-count`** `n`

Nastavuje počáteční počet vykreslovaných objektů. Měření začíná s tímto počtem objektů.

-b, **--final-count** n

Nastavuje koncový počet vykreslovaných objektů. Při překročení meze je zahájeno další měření.

-c, **--count-increment** n

Změna počtu objektů v každém kroku (při uplynutí měřícího intervalu). Je povolena i záporná hodnota v případě, že koncový počet objektů je menší než počáteční.

-d, **--init-size** w h

Výchozí velikost vykreslovaných objektů (šířka a výška).

-e, **--final-size** w h

Koncová velikost vykreslovaných objektů (šířka a výška).

-f, **--size-increment** w h

Změna velikosti objektu v každém kroku (při uplynutí měřícího intervalu). Je povolena i záporná hodnota.

-m, **--textures** n

Nastavení druhu texturování (0 bez textur, 1 jedna textura, 2 dvě multitextury, 3 čtyři multitextury).

-x, **--texture-filtering** min mag

Způsob filtrování textur. Vychází z parametrů funkce glTexParameter().

-q, **--texture-scale** n

Určuje, kolikrát má být originální textura o rozměrech 256x256 pixelů zmenšena.

-**i**, **--z-increment** n

Změna z-ové souřadnice v případě, že je celé plocha zaplněna. Je povolena i záporná hodnota.

-l, **--lights** options

Nastavení světel. Detailní popis parametrů viz níže.

-s, **--scene** n

Volba scény.

-z, **--z-buffer** n

Nastavení Z-bufferu. Vychází z parametrů funkce glDepthFunc(). Je-li n = 0, je Z-buffer vypnut.

-bl, **--blending** src dst

Nastavení blendingu. Vyhází z parametrů funkce glBlendFunc().

-p, **--with-displaylist**

Zapíná použití displaylistu. Scéna je nejprve nakreslena do displaylistu a poté se používá displaylist.

-w, **--switching-mode** n

Způsob přepínání barev a materiálů (0 změněno pouze jednou, 1 změněno s každým vrcholem, 2 změněno s každým trojúhelníkem, 3 změněno s každým objektem). Má smysl pouze u příslušné scény.

-o, **--material-settings** n

Povoluje přepínání materiálů (implicitně je povoleno přepínání barev) a určuje, které parametry jsou u materiálu nastavovány (1 ambient, 2 diffuse, 4 specular, 8 emission). Vychází z možností funkce `glMaterial()`.

-fs, **--flat-shading**

Zapíná konstantní stínování. Implicitně je nastaveno Gouraudovo stínování.

-ds, **--disable-swap**

Zakazuje přepínání bufferů pomocí `glutSwapBuffers()`.

10.1.2 Nastavování světel

Argumenty v nastavení jsou volitelné (nemusí být použity) a mohou být uvedeny v jakémkoliv pořadí s výjimkou čísla světla. Pokud je číslo světla uvedeno tak musí být uvedeno na prvním místě. Je-li použito více parametrů pracujících se světly, je vyhodnocování všech parametrů prováděno zleva doprava (dá se s výhodou využít pro globální a individuální nastavování).

n

Číslo světla, kterému budou měněny parametry. Pokud není číslo uvedeno, provádí se požadované nastavení pro všechny dosud zapnutá světla.

pnt, **point**

spt, **spot**

drc, **direct**

Nastavuje typ světla – bodové, reflektorové, směrové.

pos, **position** x y z

Mění pozici světla. Povoleny jsou celá i reálná čísla.

sdi, **spot-direct** x y z

cut, **spot-cutoff** n

exp, **spot-exp** n

Nastavení je určeno pro reflektorové světlo. Určuje směr hlavního světelného paprsku, úhel kužele a míru koncentrovanosti světla.

cat, **const-att** n

lat, **linear-att** n

qat, **quad-att** n

Nastavení útlumu – konstantní, lineární, kvadratické

amb, **ambient** r g b

dif, **diffuse** r g b

spc, **specular** r g b

Nastavení ambientní, difúzní a spekulární složky světla. Musí být použity reálná čísla.

info, **print-info**

Zapíná vypisování detailních informací o nastavených světlech.

ani, **animate**

Povoluje „automatické pohybování“ světlem. Pohyb je docílen změnou x-ové a y-ové souřadnice . Mezními hodnotami jsou hranice okna. S umístěním a směrem reflektorového světla se musí pracovat obezřetně, aby v okrajových částech kužel nemířil mimo okno.

10.1.3 Formát souboru s rychlotestem

Soubor s rychlotestem sestává z následujících částí:

1) komentáře – jsou uvozeny znakem # (mříže) a od něj pokračují do konce řádku, mohou být uvedeny kdekoliv

2) příkazy gnuplotu – jsou uvozeny znakem ! (vykřičník) na začátku řádku, případné úvodní mezery jsou ignorovány (je možné používat makra – viz dále)

3) parametry testu – stejné jako by byly zapisovány na příkazovou řádku

Finální statistiky vytvořené rychlotestem zpracovává skript `chart.sh`, který poskytuje několik základních maker usnadňující zadávání proměnných údajů jakými jsou jméno souboru s daty (bez něj by nešlo soubor lehce přejmenovat) nebo číslo testu (bez něj by nebylo možné snadno slučovat více výsledků).

Přehled podporovaných maker:

__FILE__	jméno zpracovávaného souboru s výsledky
__TESTNR__	číslo aktuálního datasetu (začíná se od 0)
__COUNT__	číslo sloupce s počtem objektů

__WIDTH__	číslo sloupce se šířkou objektu
__HEIGHT__	číslo sloupce s výškou objektu
__FRAME__	číslo sloupce s celkovým počtem vykreslených snímků
__TIME__	číslo sloupce s celkovou dobou vykreslování
__FPS__	číslo sloupce s hodnotami FPS
__VPS__	číslo sloupce s hodnotami VPS
__PPS__	číslo sloupce s hodnotami PPS
__PAUSE__	je nahrazeno '__DELAY__ "gnuplot: Press RETURN..."'
__DELAY__	je nahrazeno '-1' nebo '0' podle požadovaného výstupu (graf, soubor)

10.2 Výkonnostní charakteristiky

Byly měřeny výkonnostní charakteristiky následujících grafických akcelerátorů:

Akcelerátor	Verze ovladače	CPU	Paměť
GF2	1.5.2	Intel Pentium 4 2.26GHz	523800 KB
GF3	1.5.2	???	261596 KB
GF4TI4200 (1)	1.5.3	Intel Pentium 4 2.53GHz	523744 KB
GF4TI4200 (2)	1.4.1	Intel Pentium 4 2.53GHz	523744 KB
GF6600	1.5.3	AMD Athlon64 3500+	1048048 KB
GF6800	1.5.2	Intel Pentium 4 2.53GHz	1048032 KB
GFFX5200	1.5.2	Intel Pentium 4 2.60GHz	522992 KB
GFFX5900XT	1.5.2	Intel Pentium 4 2.80GHz	522992 KB
Radeon9200	1.3.3604	AMD Athlon XP 2000+	523808 KB
RadeonX550	2.0.5523	Intel Pentium 4 3.00GHz	522668 KB

U všech akcelerátorů byly postupně prověřovány: displaylisty a Z-buffer, vertex arrays a vertex buffer objects, pásy trojúhelníků (*strip*), typická scéna (displaylist, 1 textura), přepínání textur, světla, přepínání barev a materiálů.

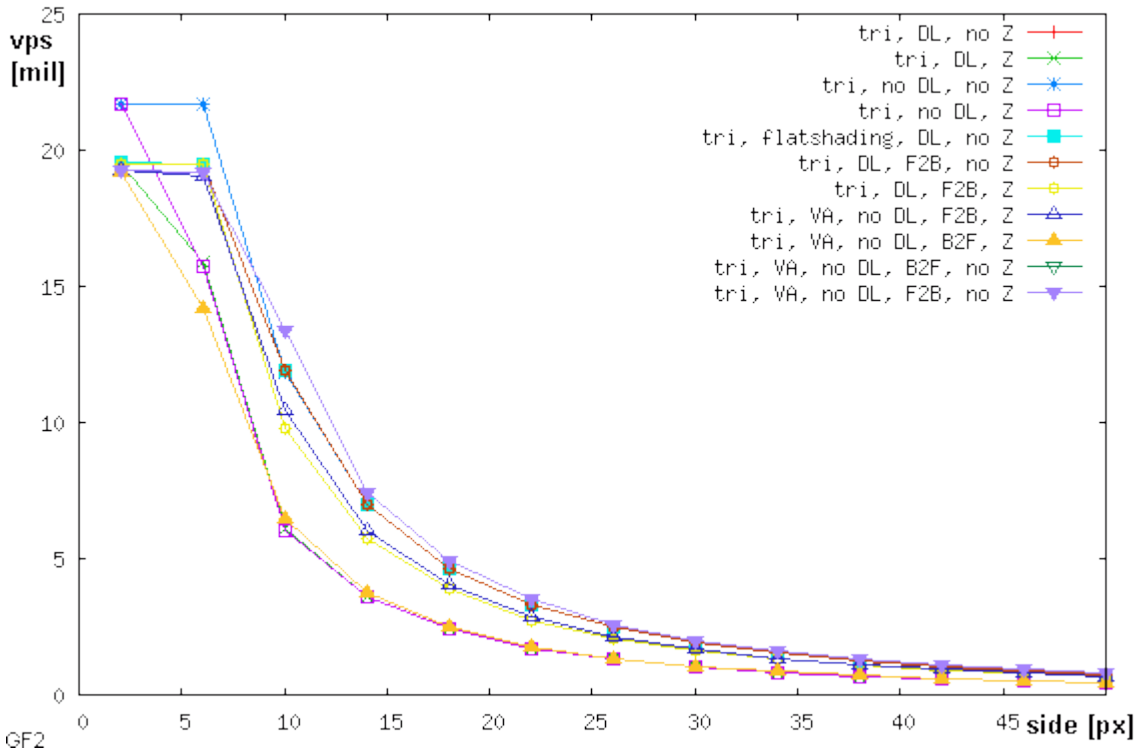
Každý test používal mediánový filtr s kapacitou 20 měření a všechny výstupy byly přesměrovány do souboru. Měřící interval byl stanoven na 10 snímků.

Význam zkratek v popisech charakteristik

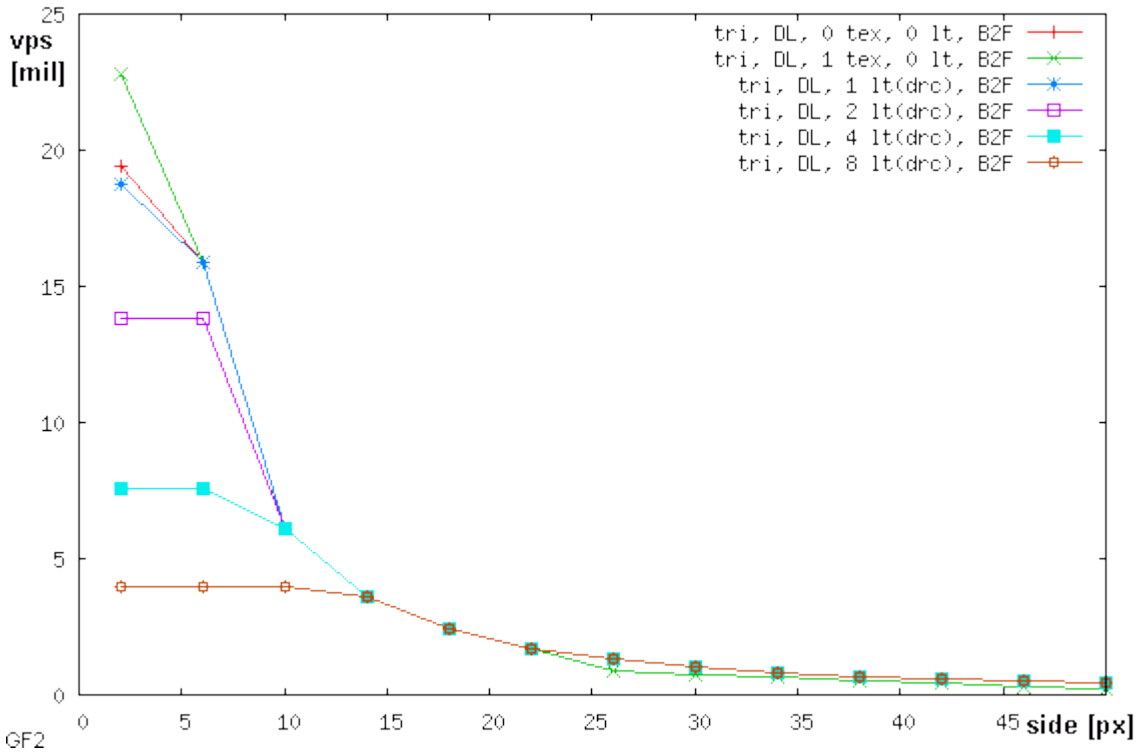
tri	trojúhelník	1 lt(drc)	zapnuto 1 směrové světlo
textri	otexturovaný trojúhelník	1 lt(pnt)	zapnuto 1 bodové světlo
tex	textura	1 lt(spt)	zapnuto 1 reflektorové světlo
col	barva	per tri	změna s každým trojúhelníkem
mat	materiál	per vert	změna s každým vrcholem
DL	displaylist	overall	bez provádění změn
Z	Z-buffer	diff	nastavena difúzní složka
VA	vertex arrays	diff+amb	nastavena difúzní, ambientní složka
VBO	vertex buffer object	1 still	použita 1 textura pro celou scénu
F2B	kreslení zepředu dozadu	1 switch	textura měněna s každým objektem
B2F	kreslení odzadu dopředu		

10.2.1 NVIDIA GeForce2 MX/AGP/SSE2

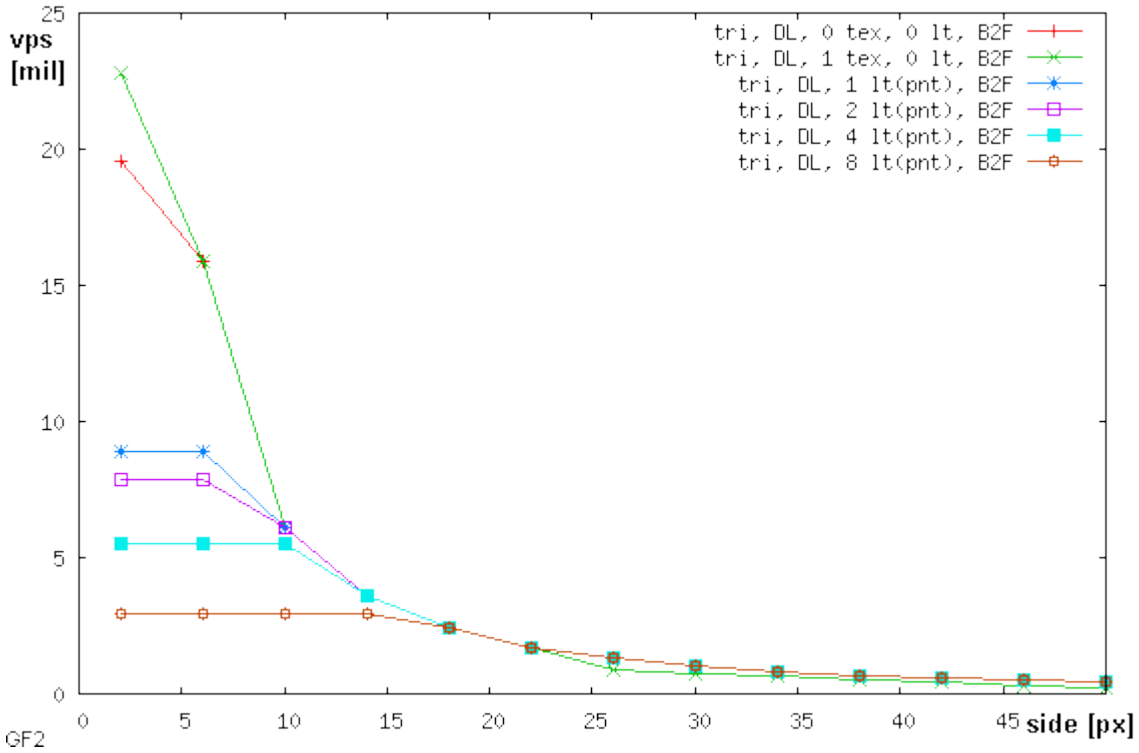
Test display listu a Z-bufferu (bez textur)



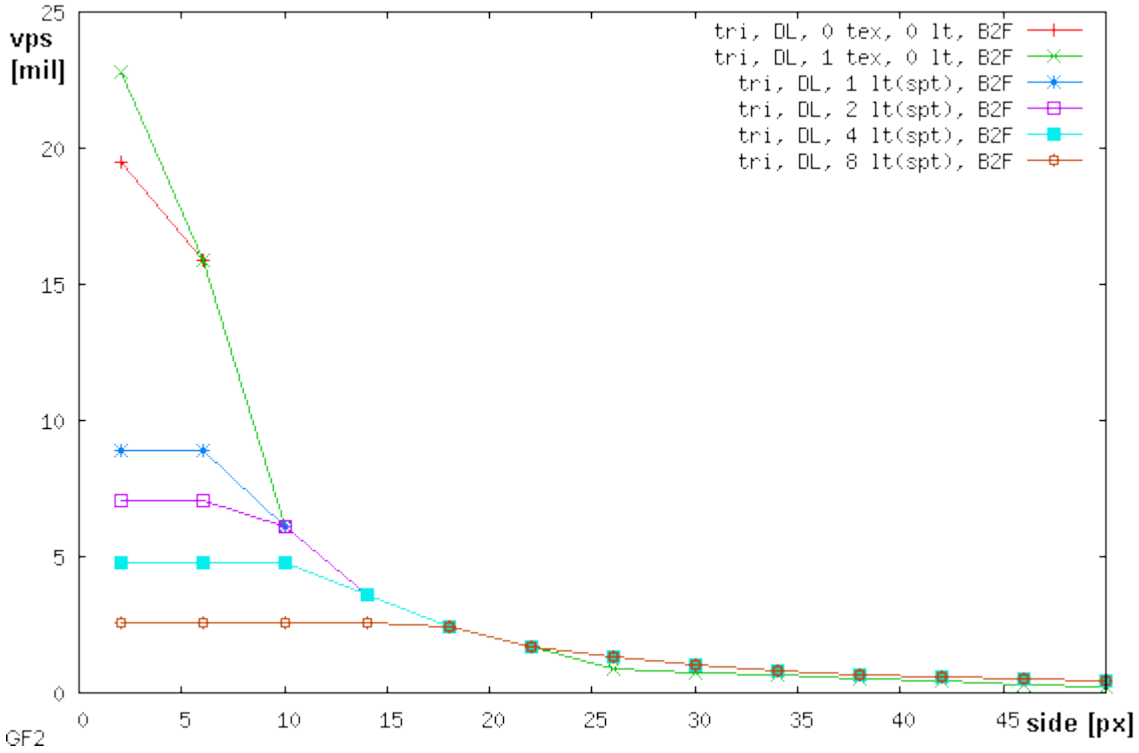
Test směrových světel (bez textur)



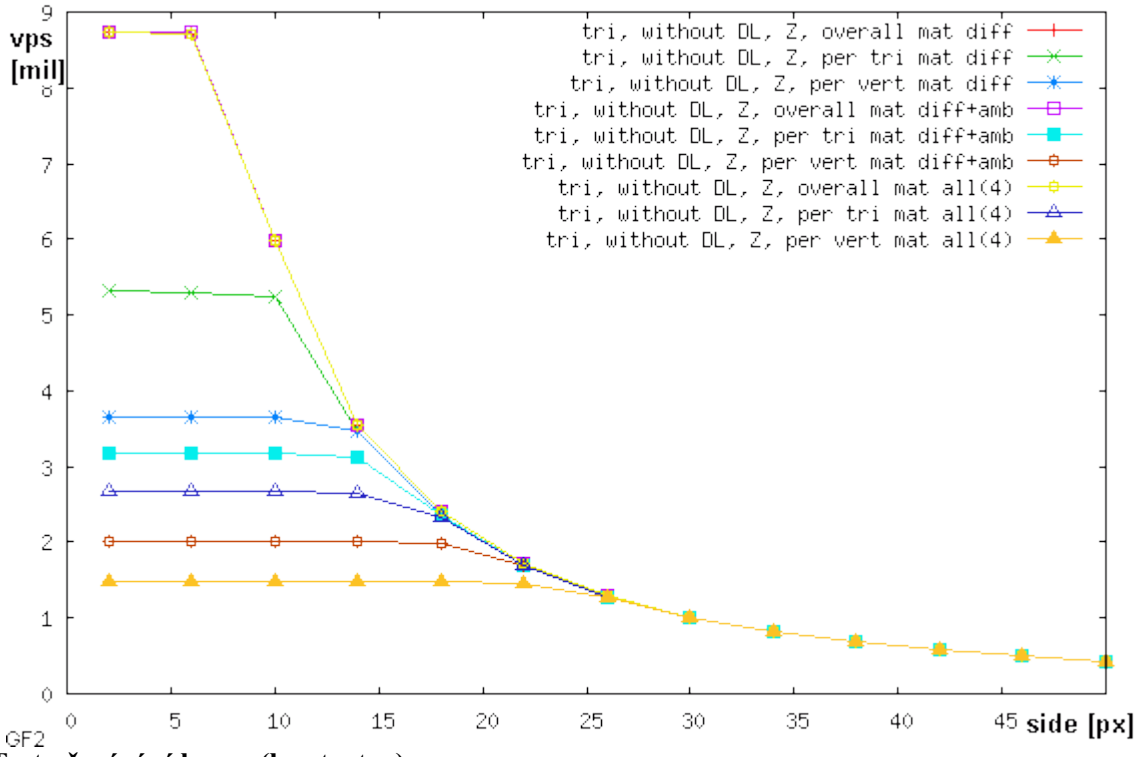
Test bodových světel (bez textur)



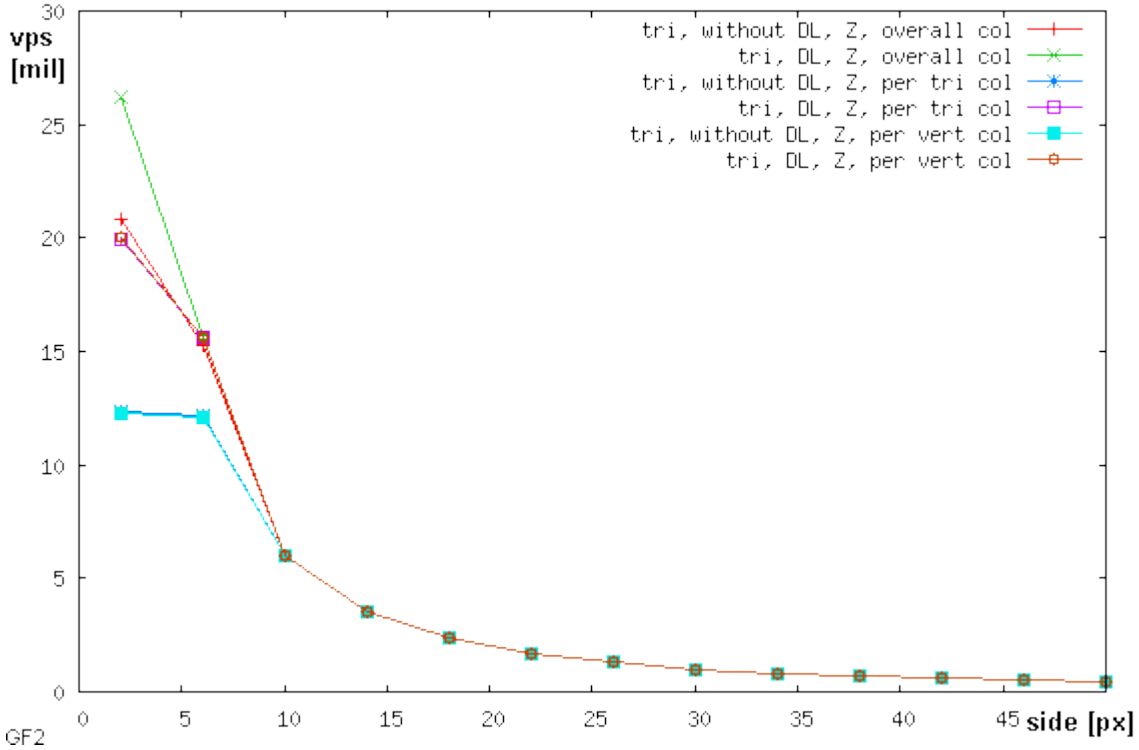
Test reflektorových světel (bez textur)



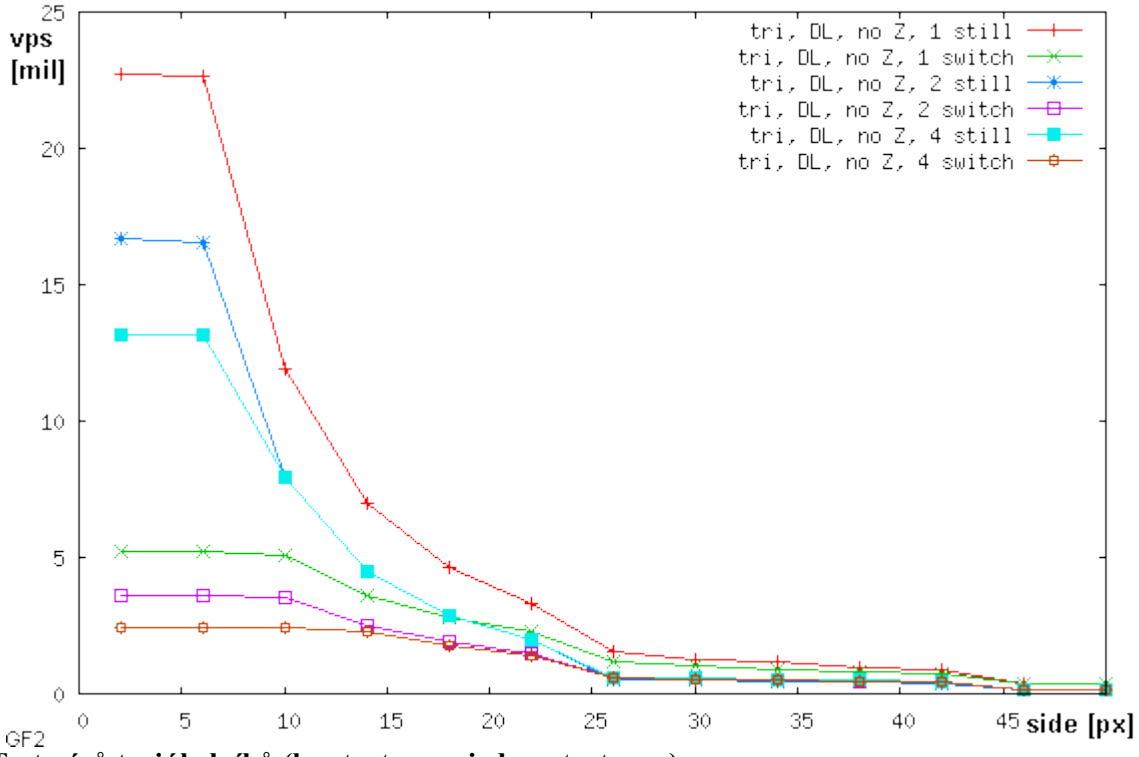
Test přepínání materiálů (bez textur)



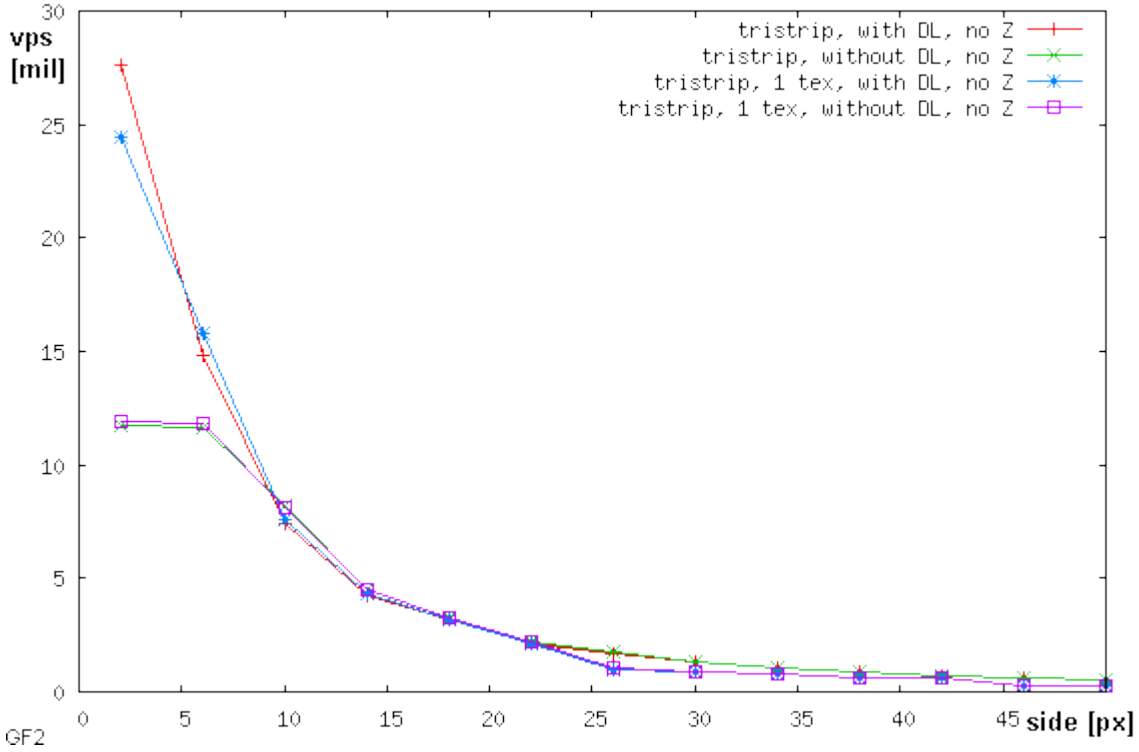
Test přepínání barev (bez textur)



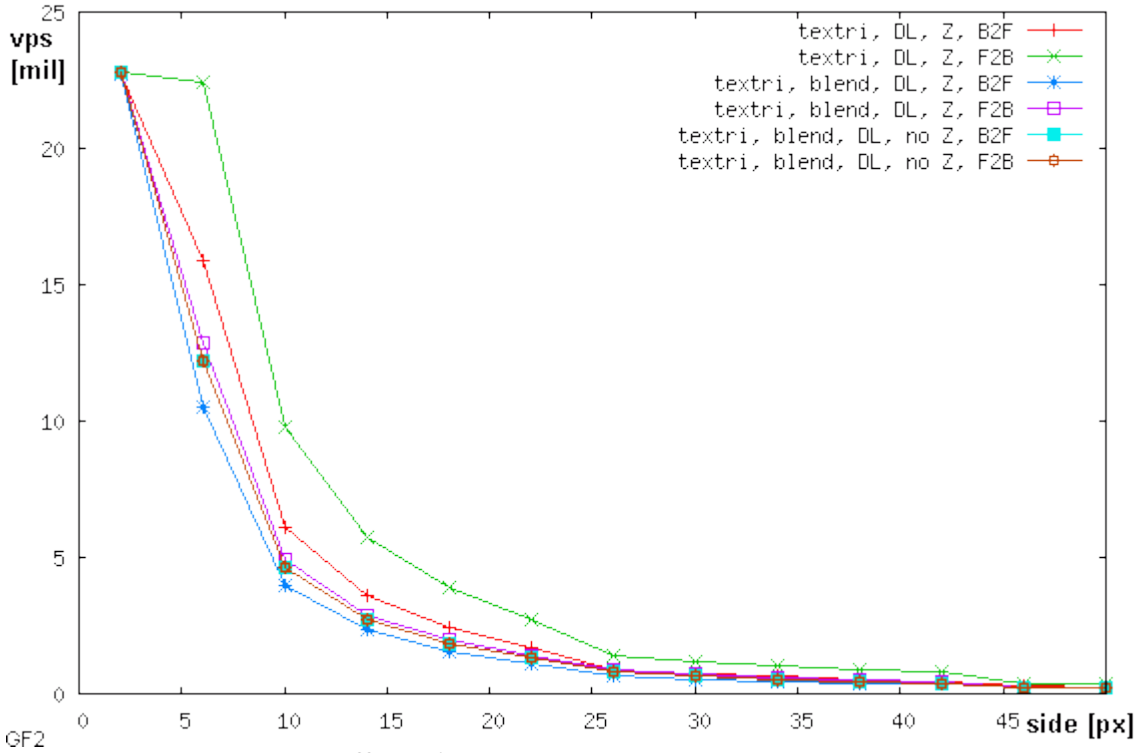
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



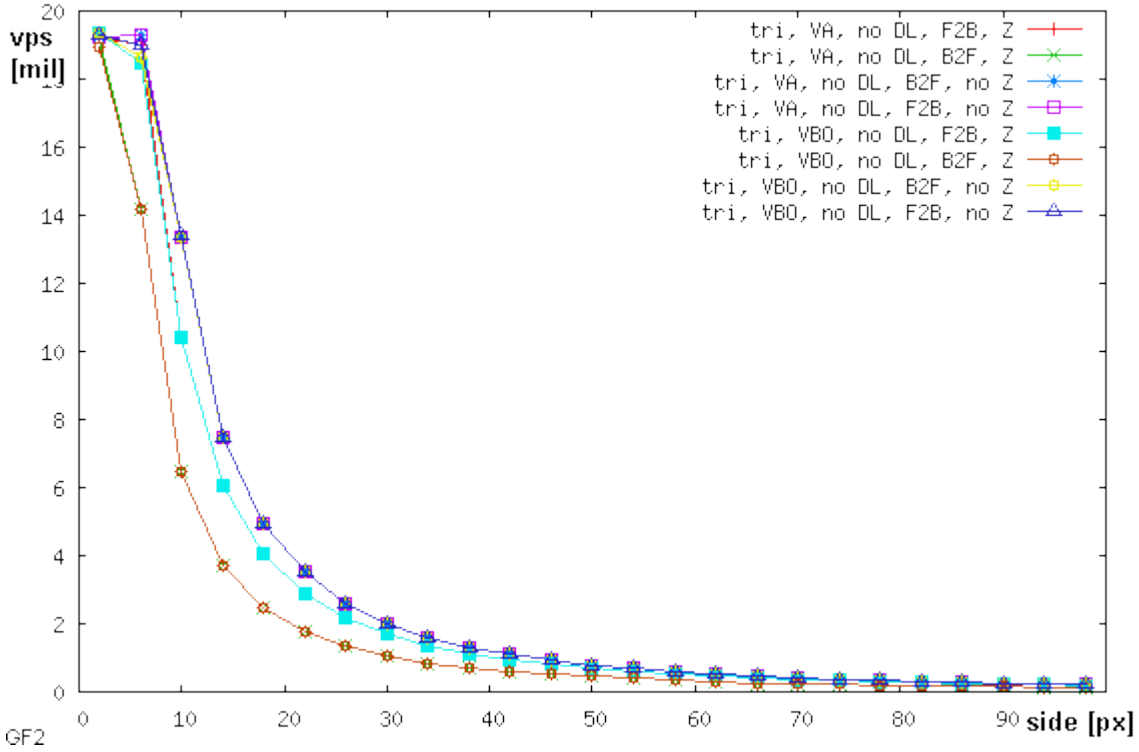
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

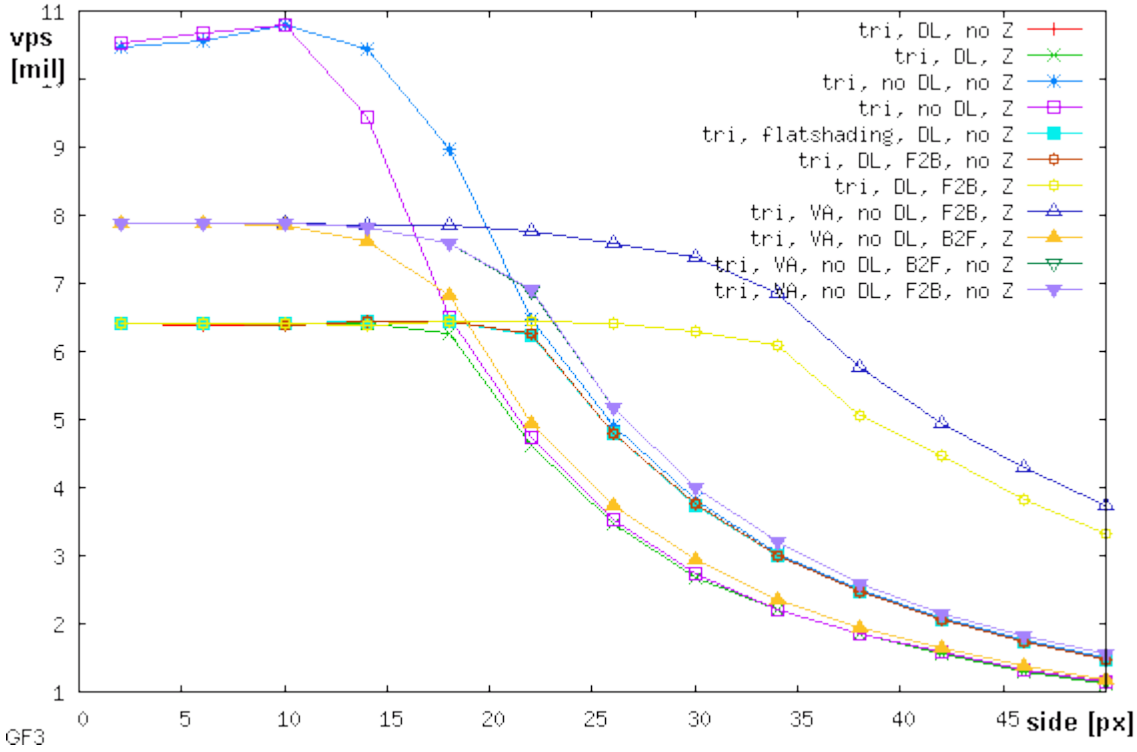


Test vertex arrays, vertex buffer objects

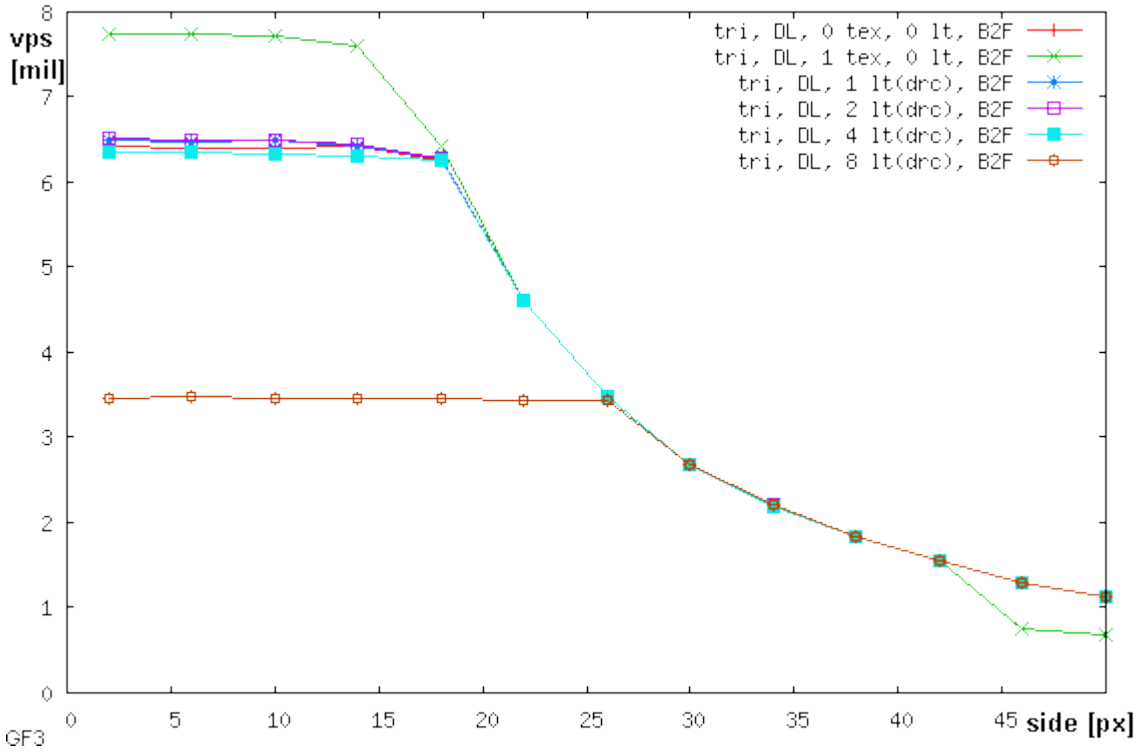


10.2.2 NVIDIA GeForce3 AGP/SSE

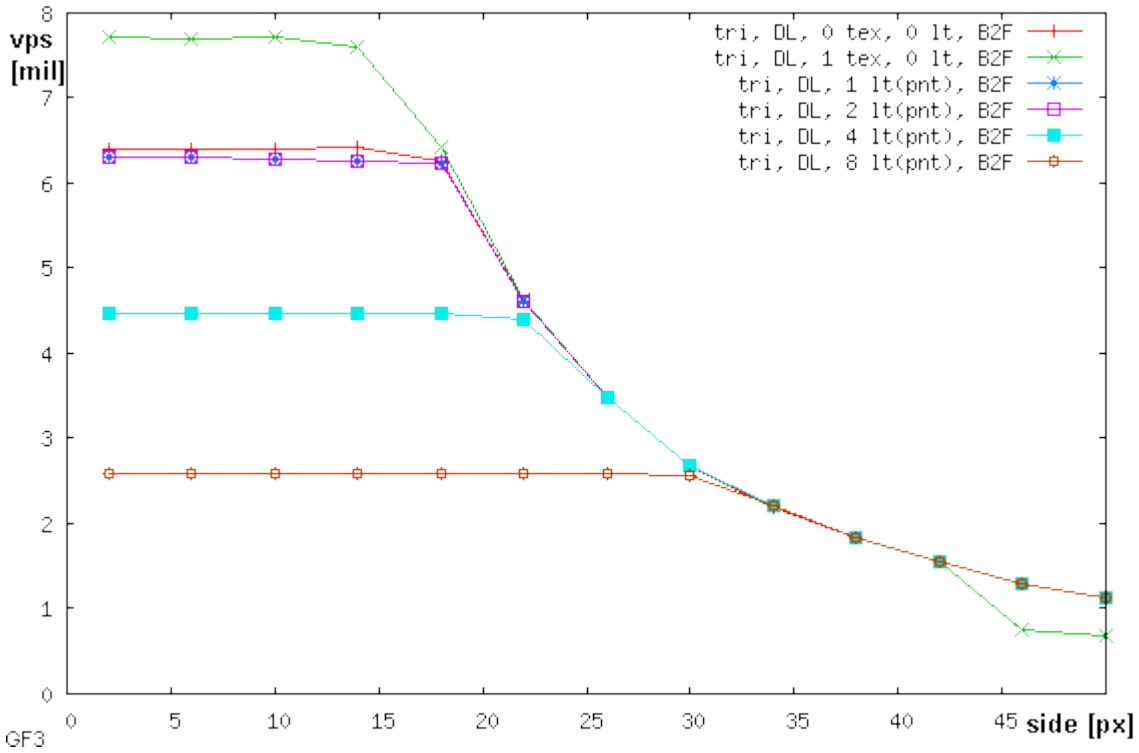
Test display listu a Z-bufferu (bez textur)



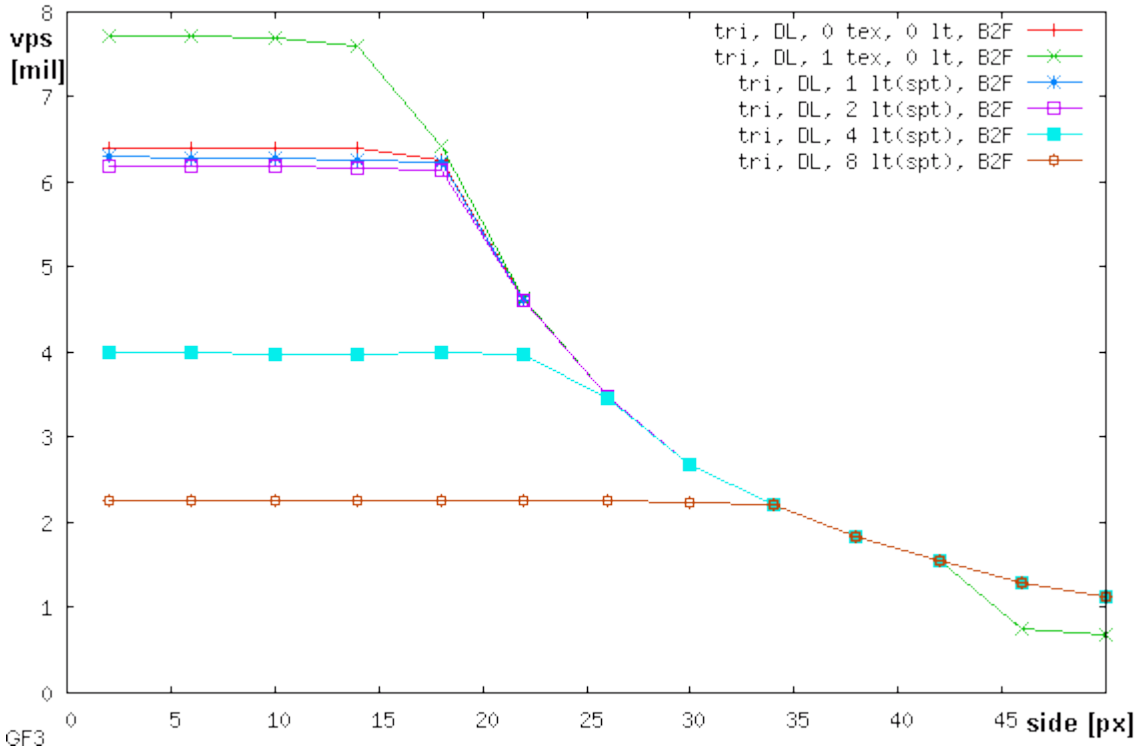
Test směrových světel (bez textur)



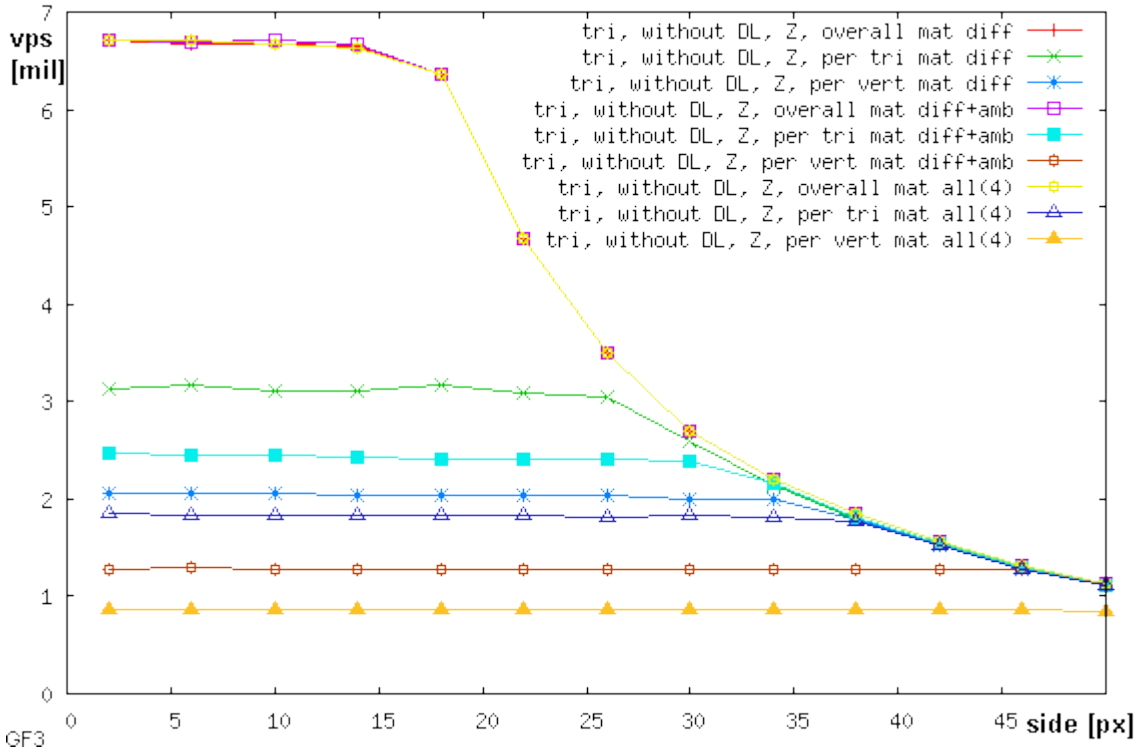
Test bodových světel (bez textur)



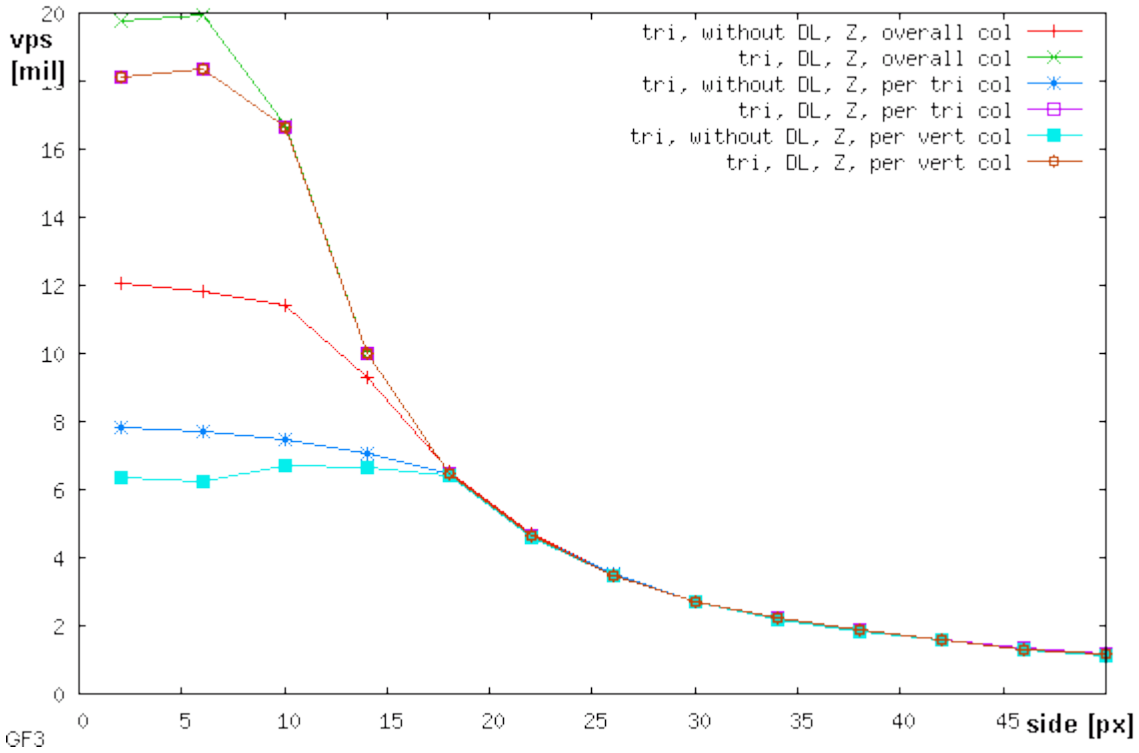
Test reflektorových světel (bez textur)



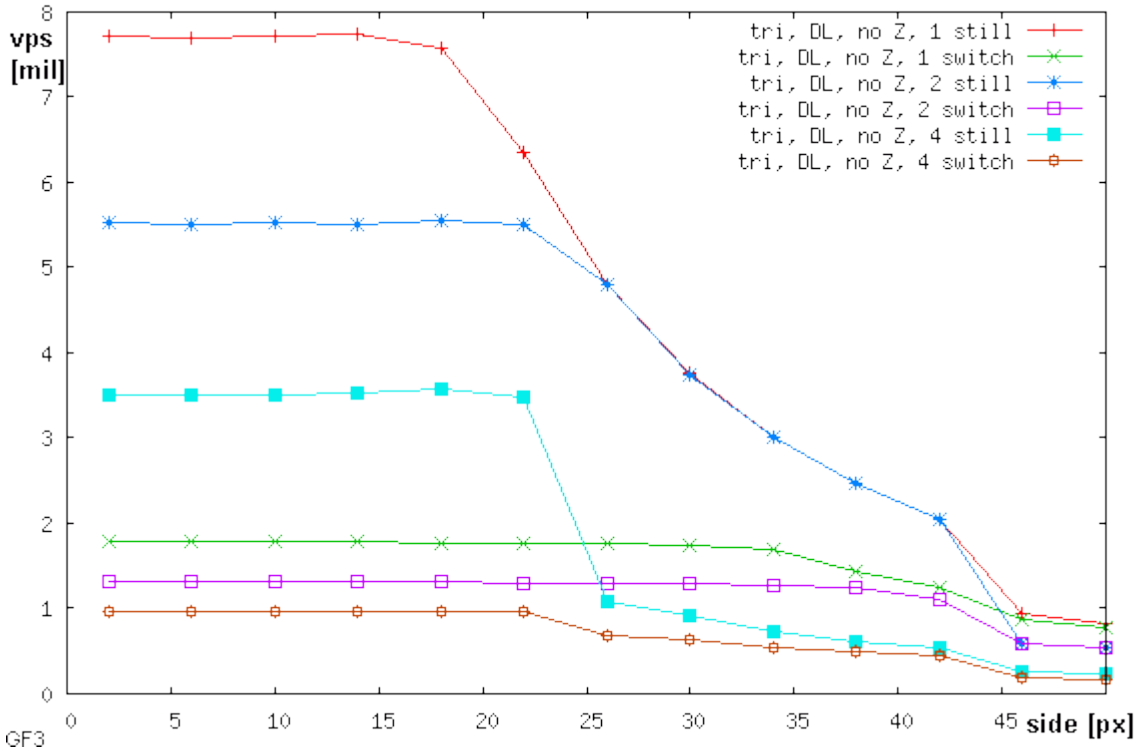
Test přepínání materiálů (bez textur)



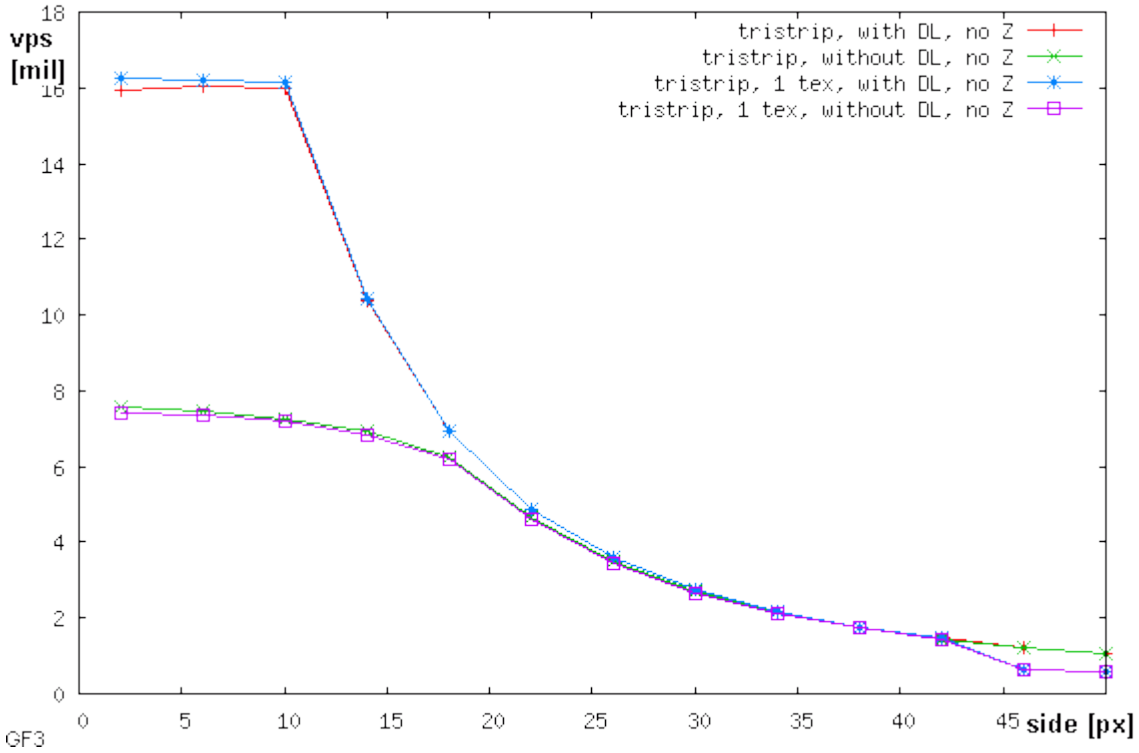
Test přepínání barev (bez textur)



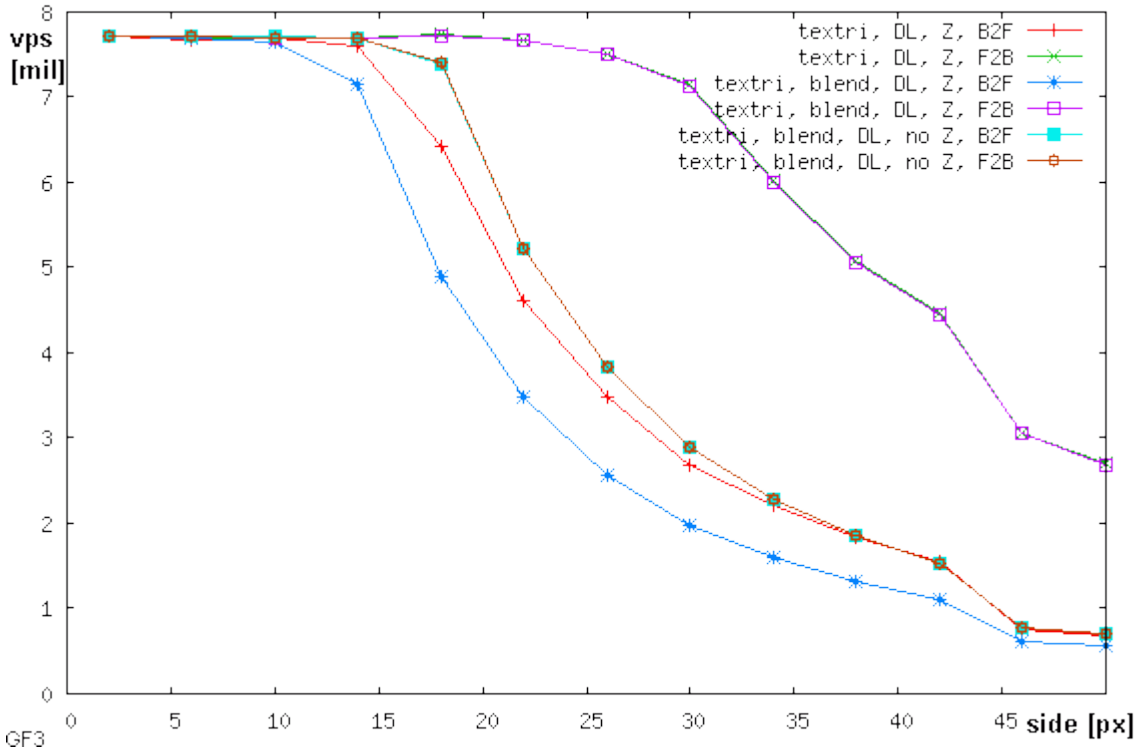
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



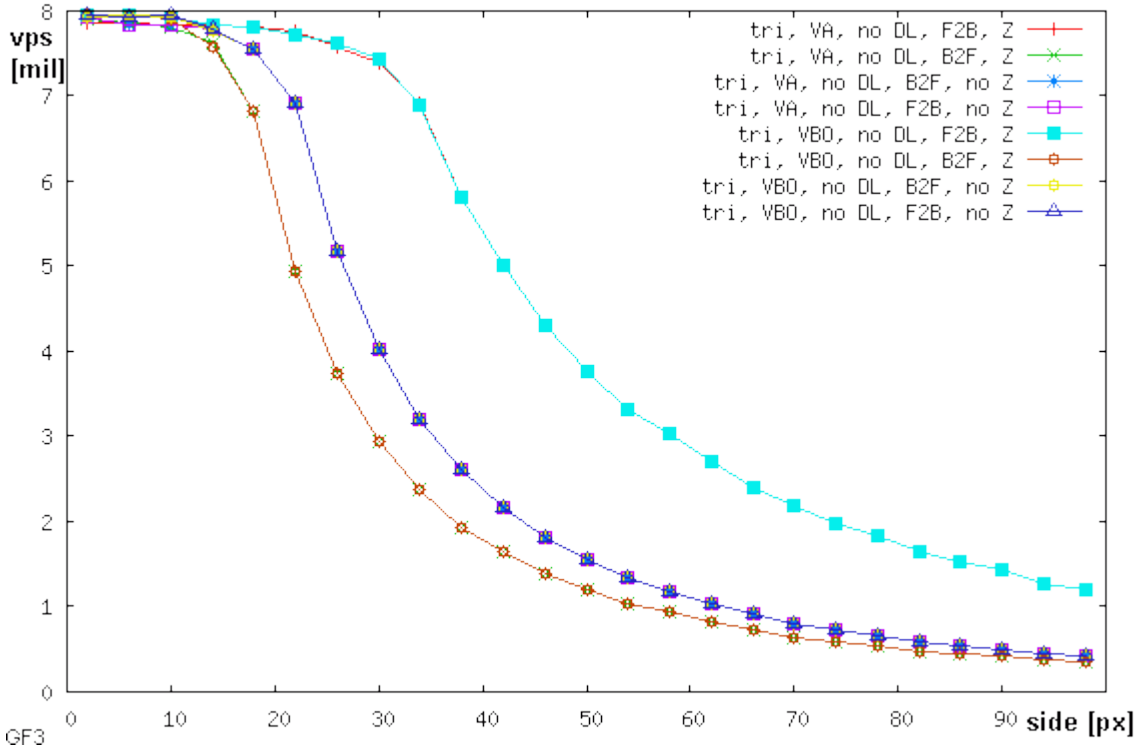
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

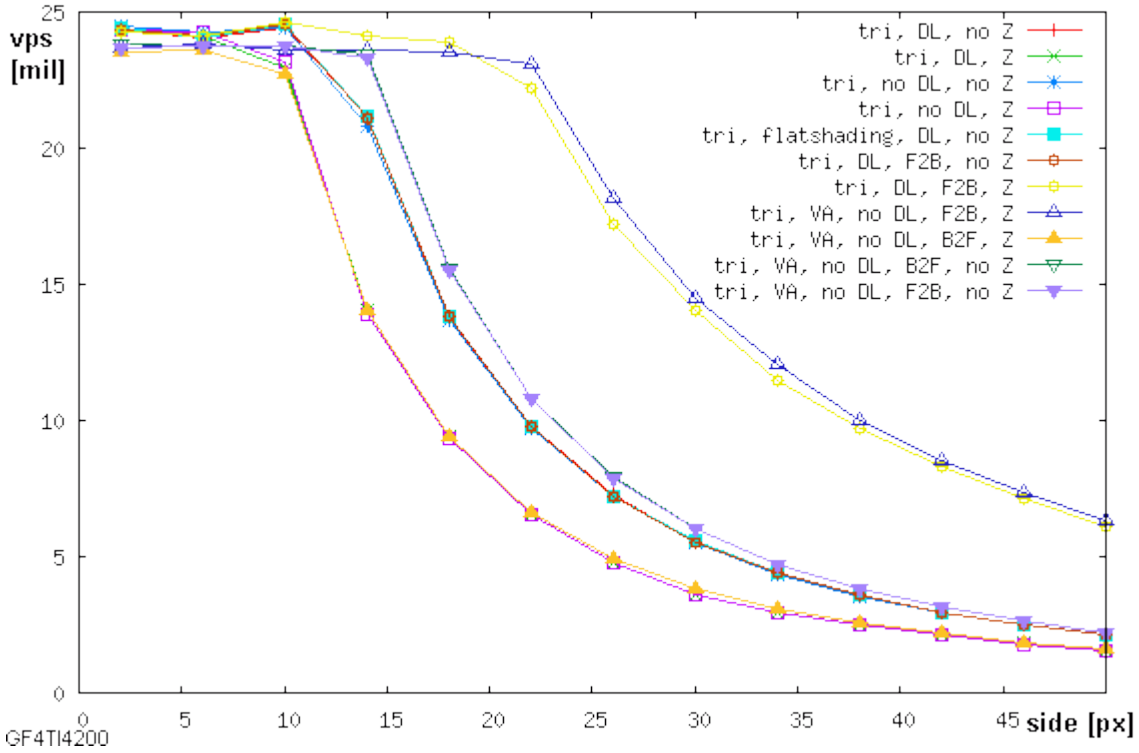


Test vertex arrays, vertex buffer objects

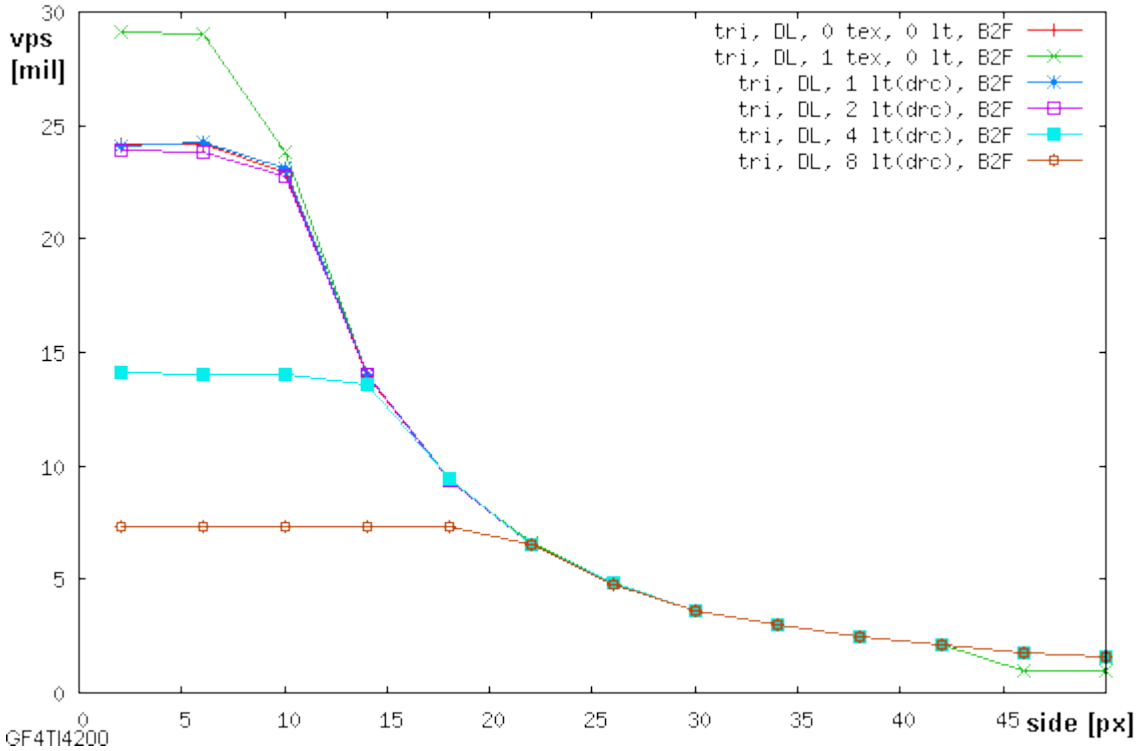


10.2.3 NVIDIA GeForce4 Ti 4200/AGP/SSE2 (první)

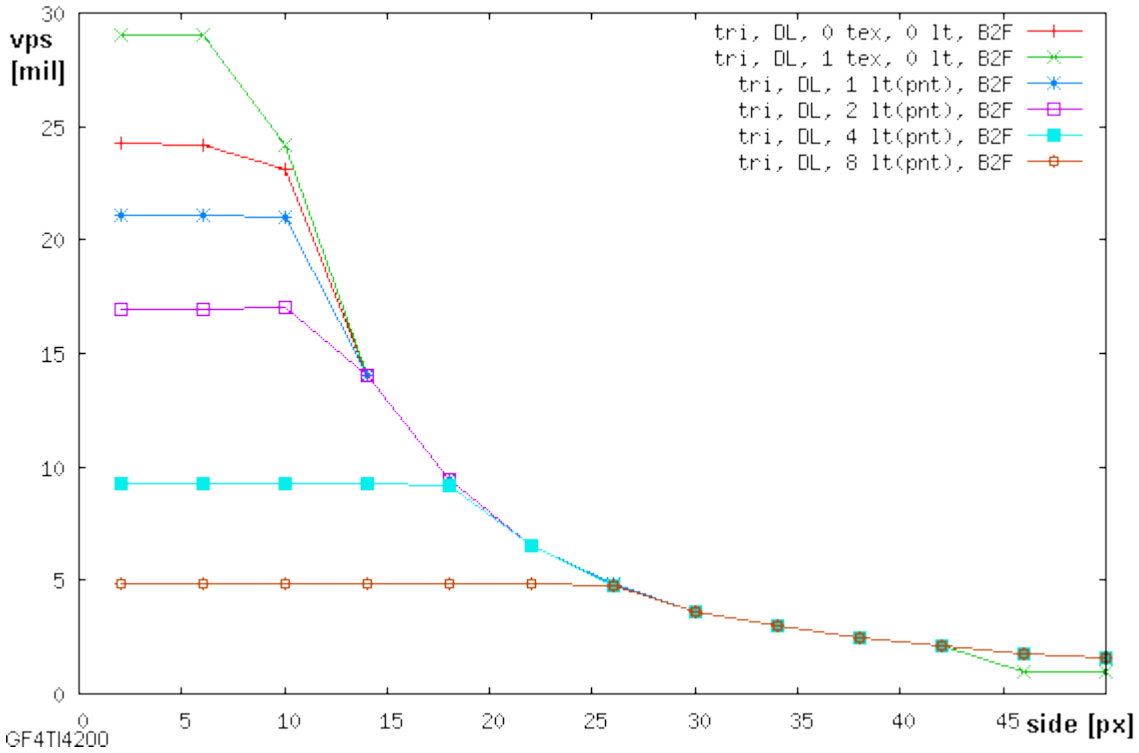
Test display listu a Z-bufferu (bez textur)



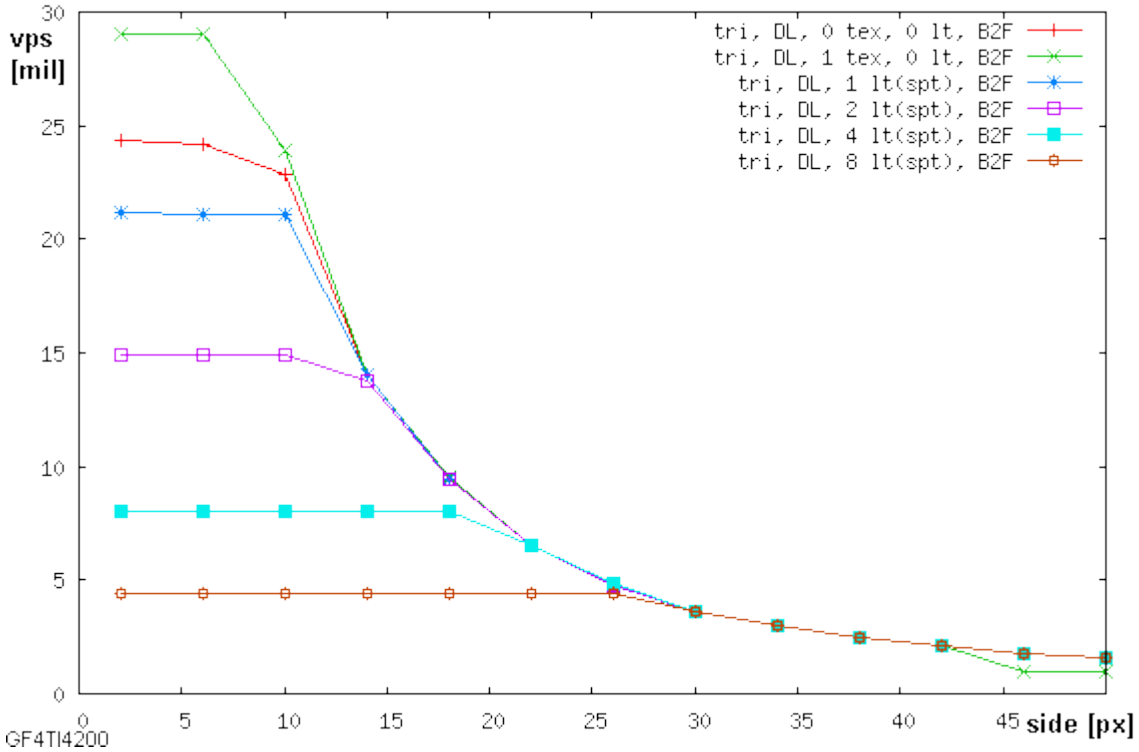
Test směrových světel (bez textur)



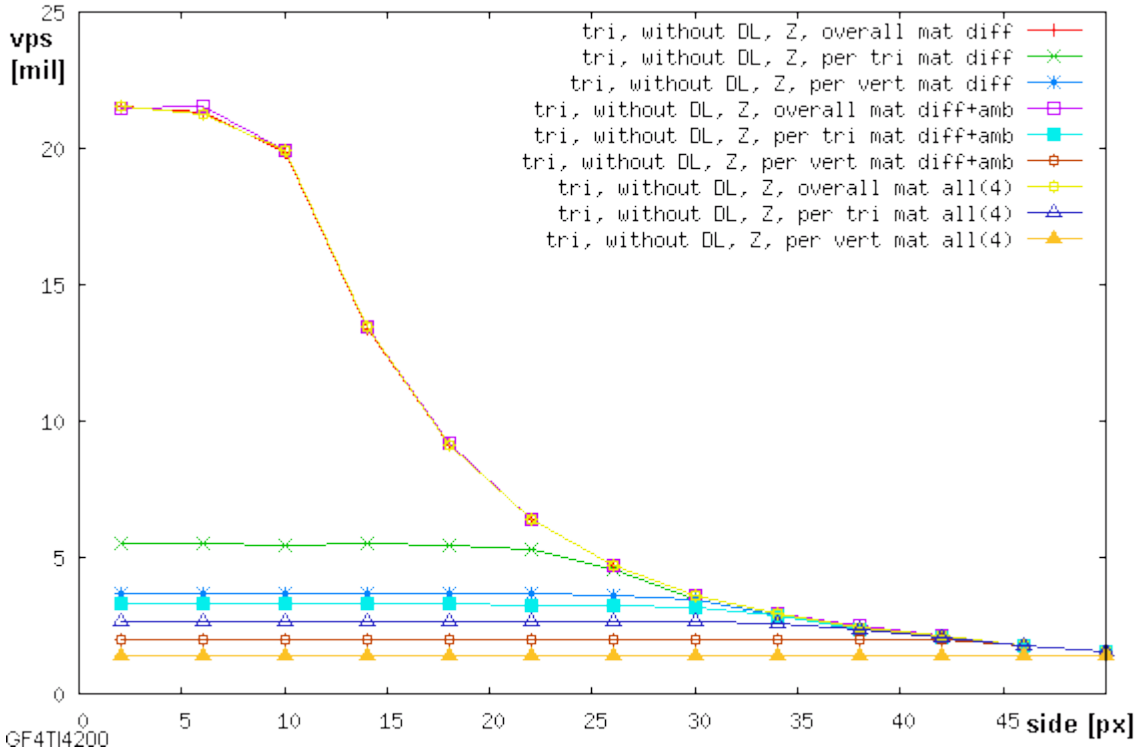
Test bodových světel (bez textur)



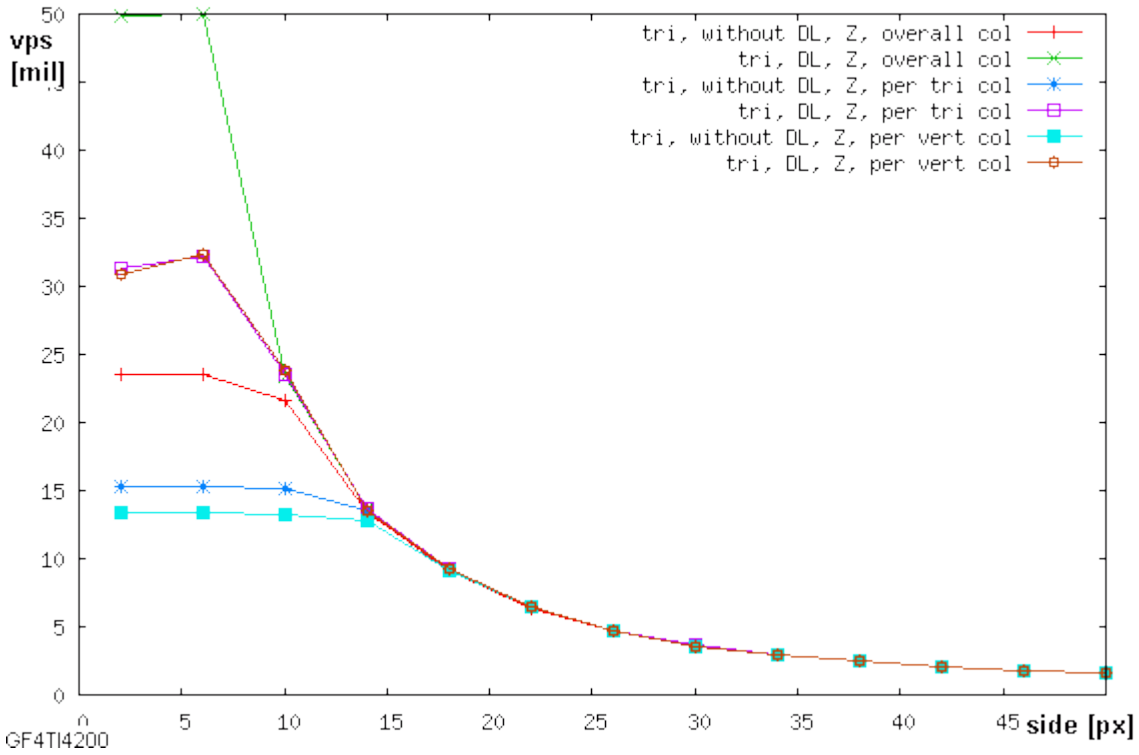
Test reflektorových světel (bez textur)



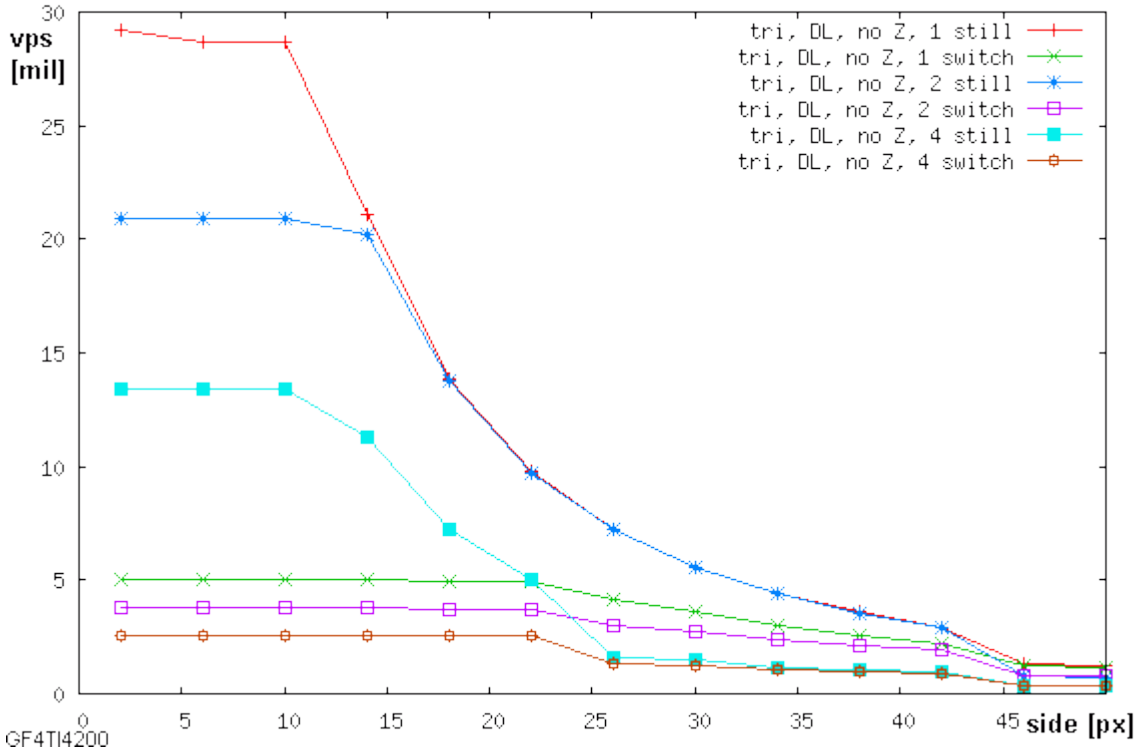
Test přepínání materiálů (bez textur)



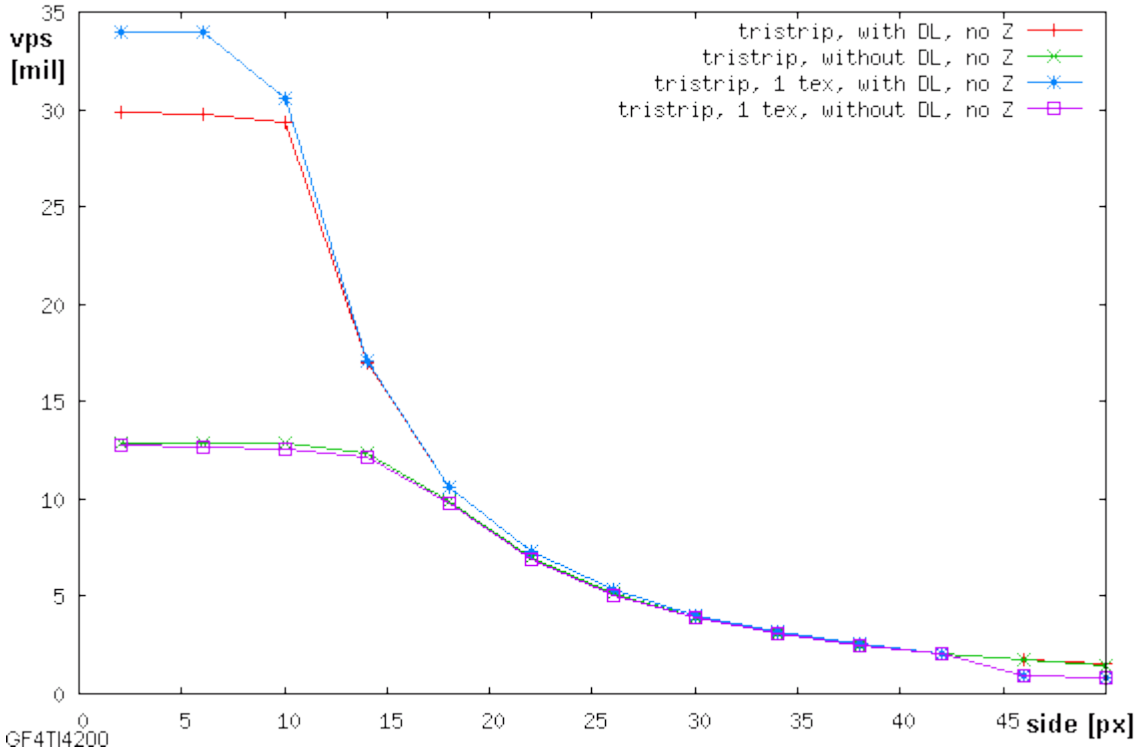
Test přepínání barev (bez textur)



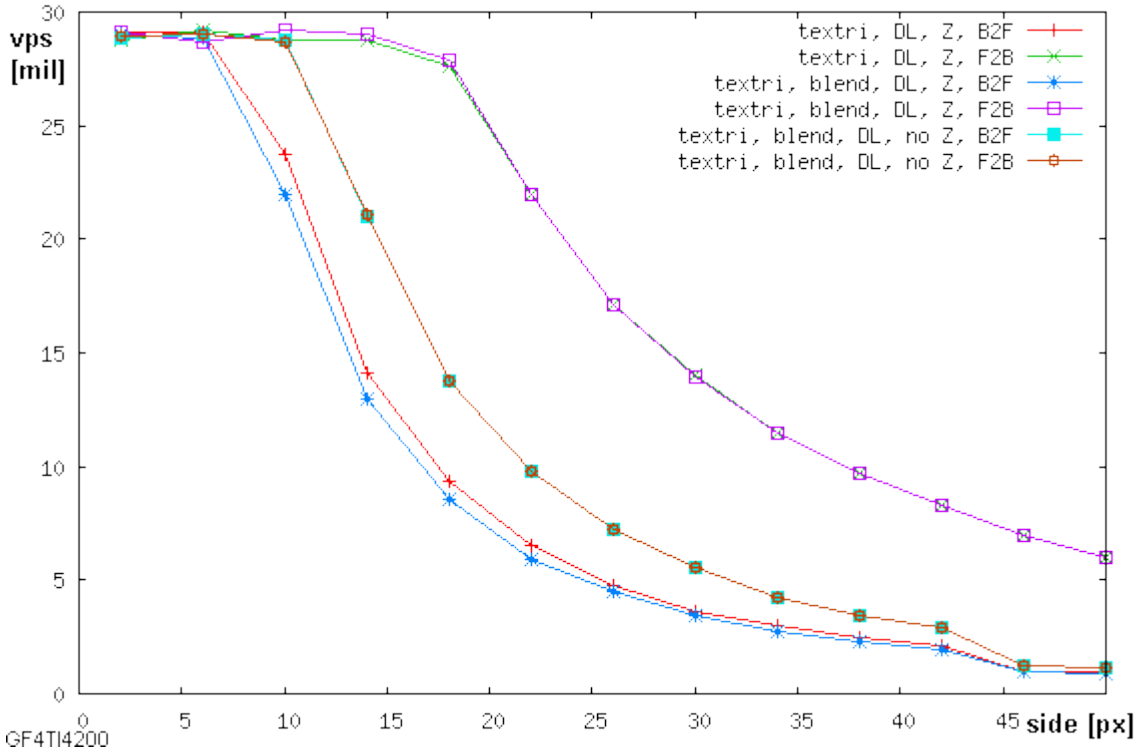
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



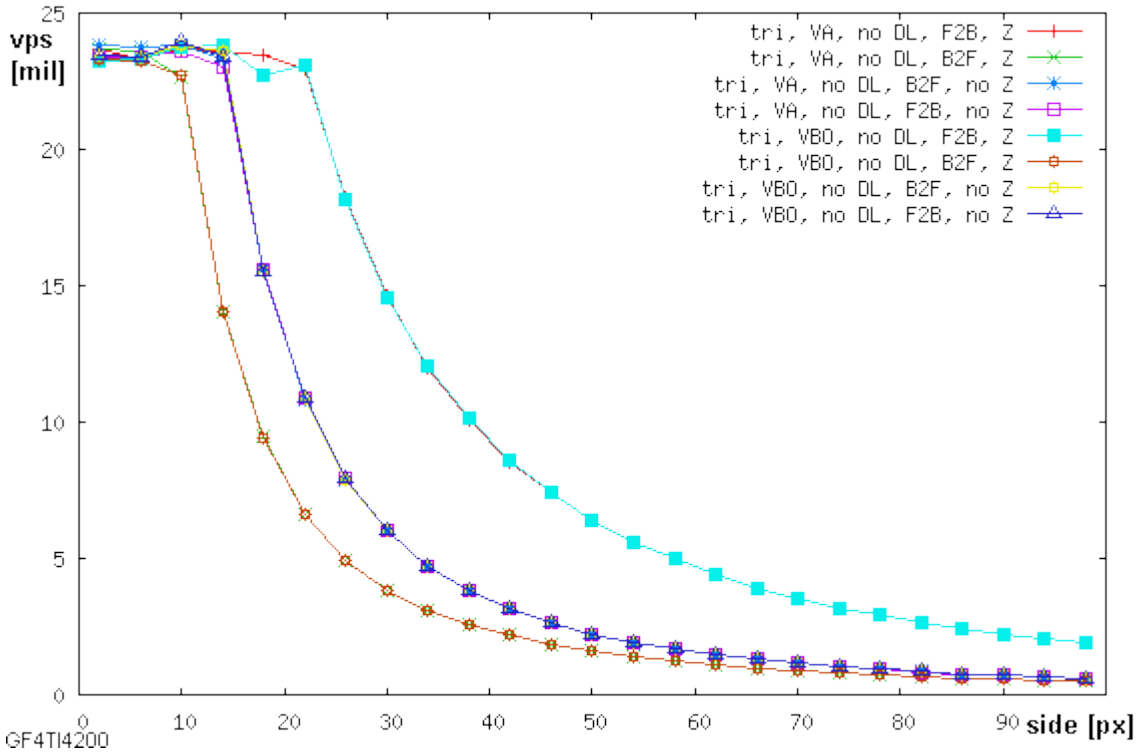
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

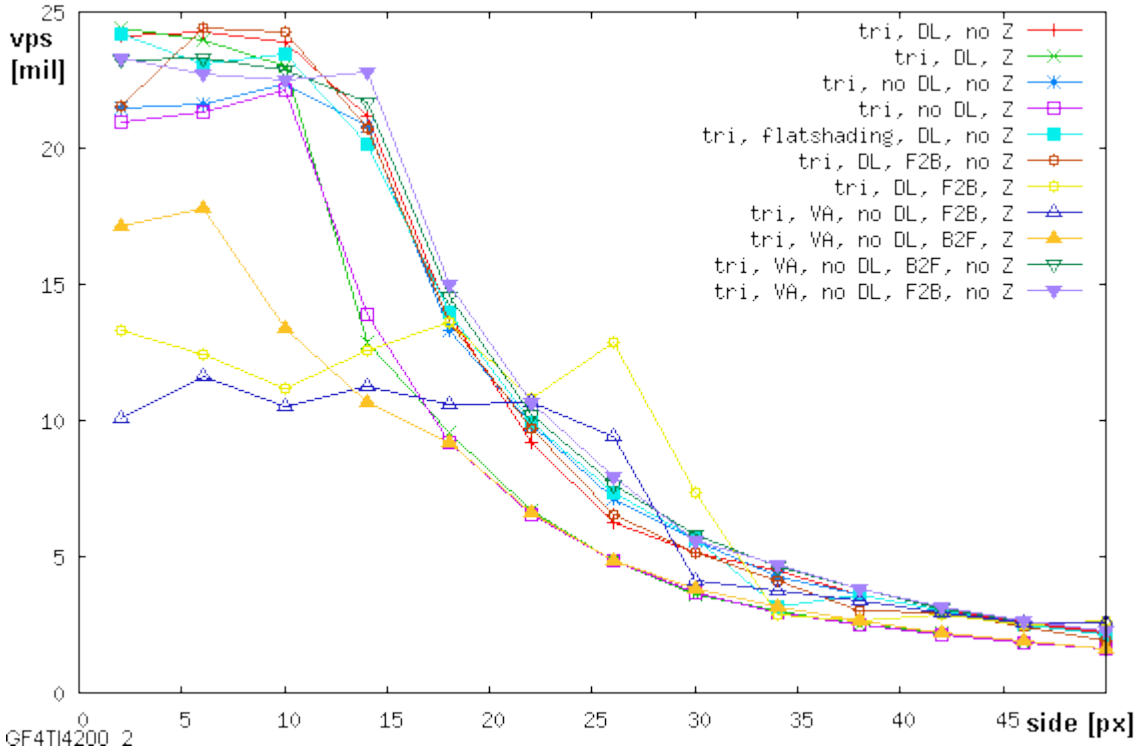


Test vertex arrays, vertex buffer objects

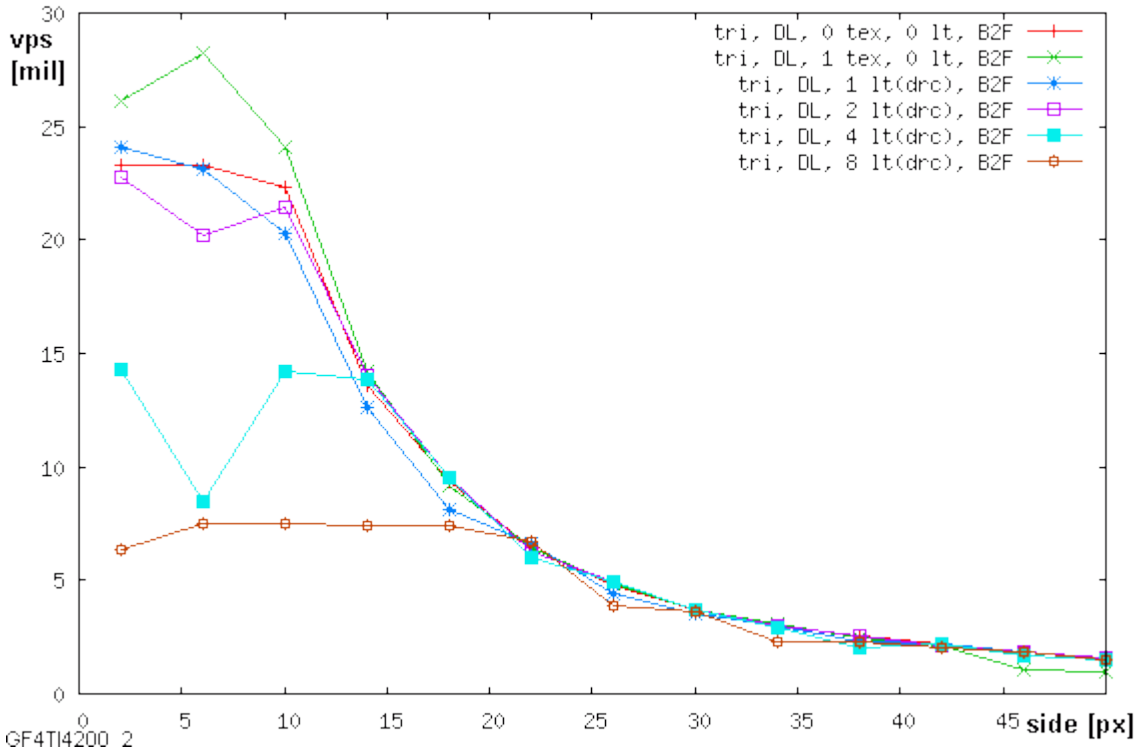


10.2.4 NVIDIA GeForce4 Ti 4200/AGP/SSE2 (druhá)

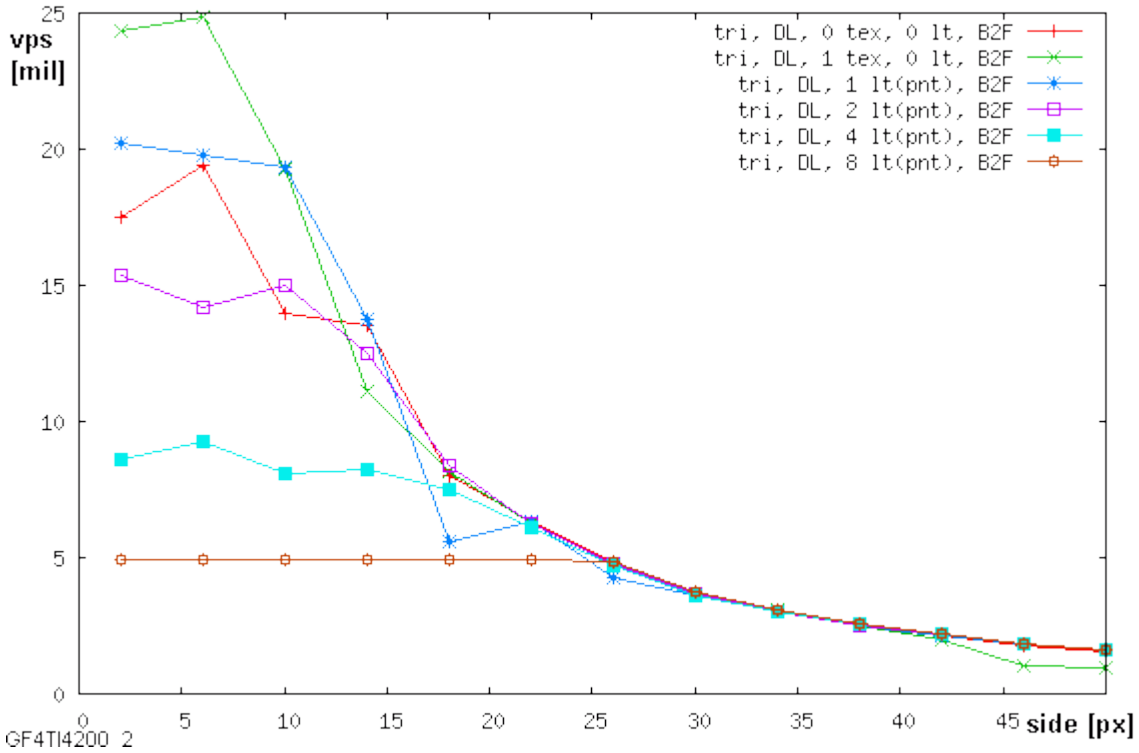
Test display listu a Z-bufferu (bez textur)



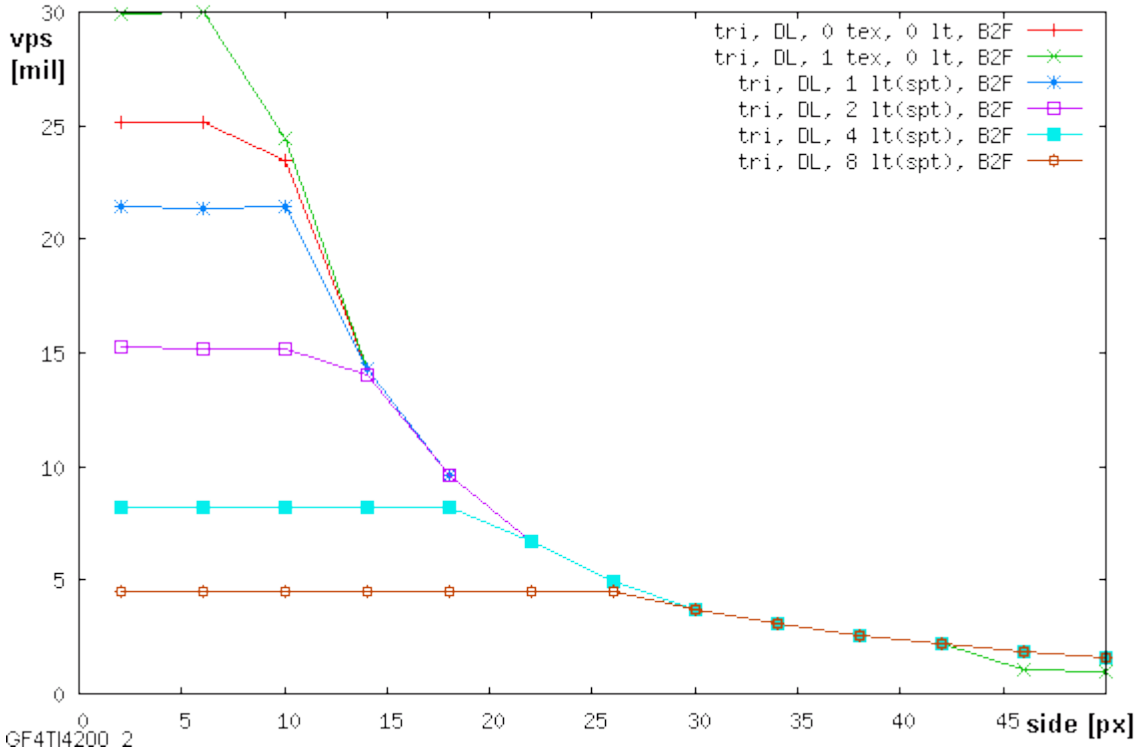
Test směrových světel (bez textur)



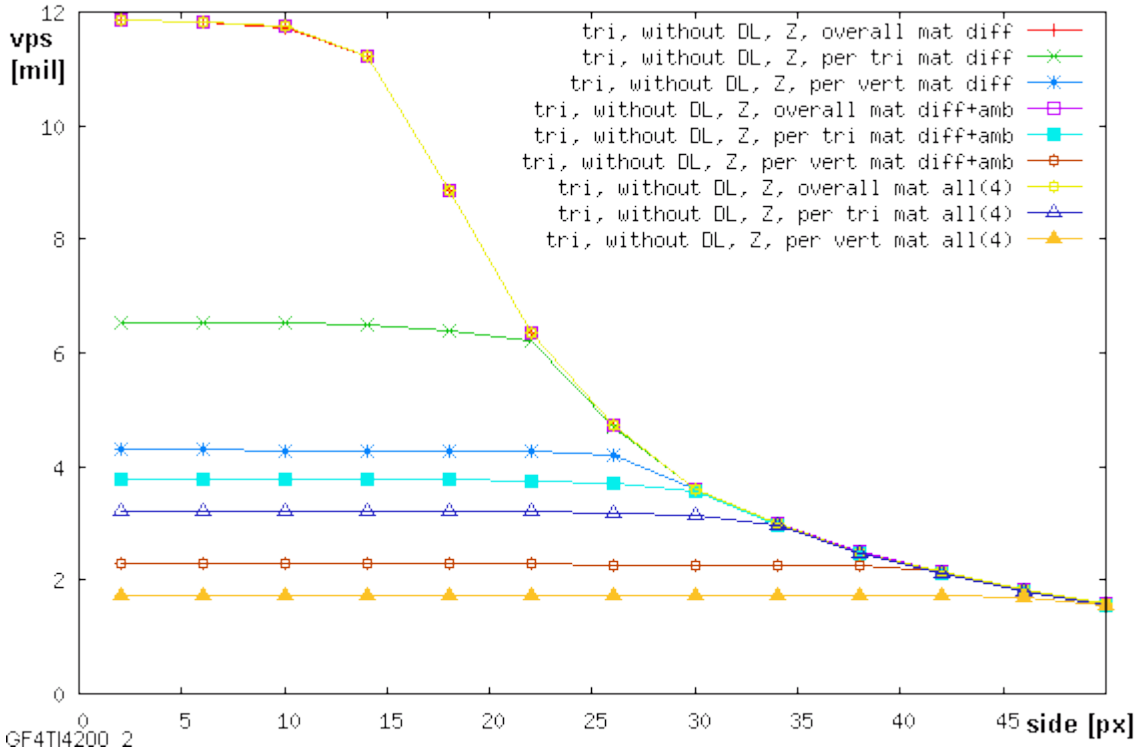
Test bodových světel (bez textur)



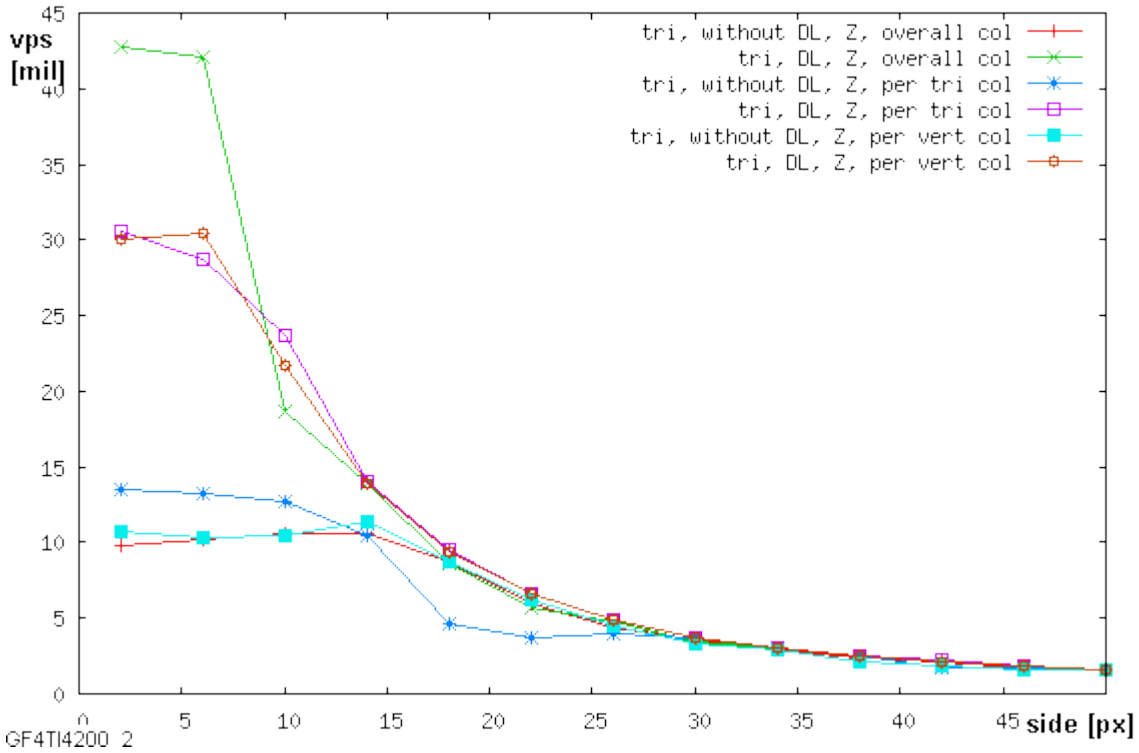
Test reflektorových světel (bez textur)



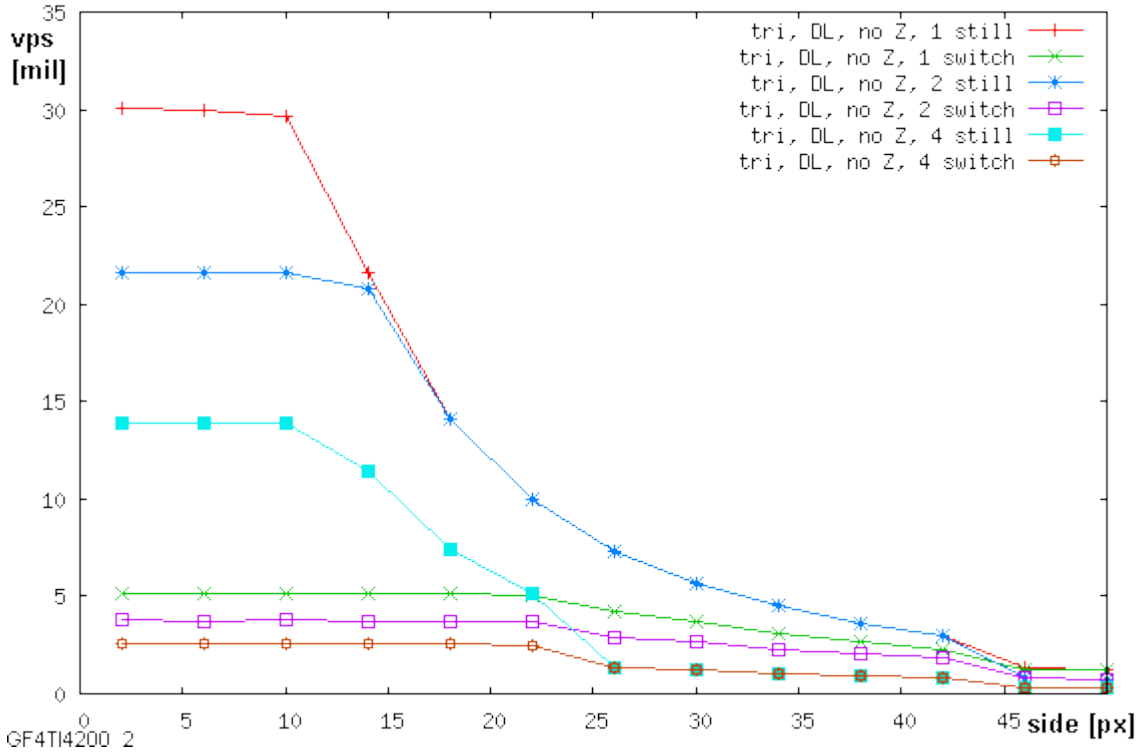
Test přepínání materiálů (bez textur)



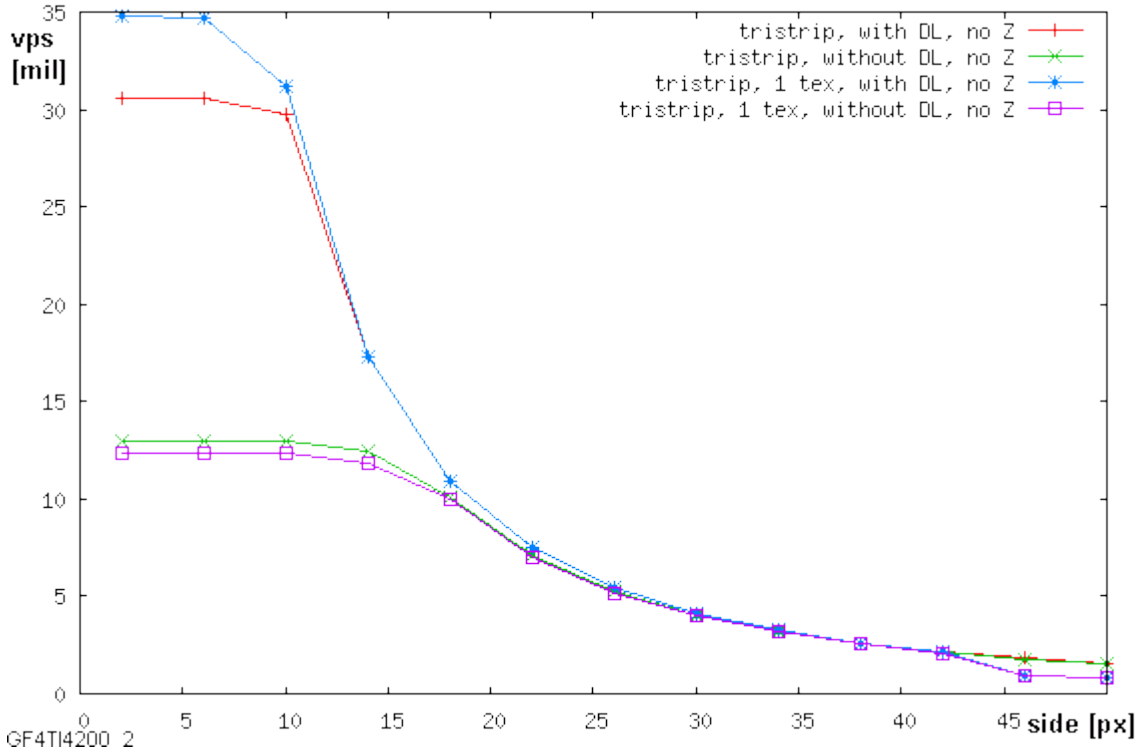
Test přepínání barev (bez textur)



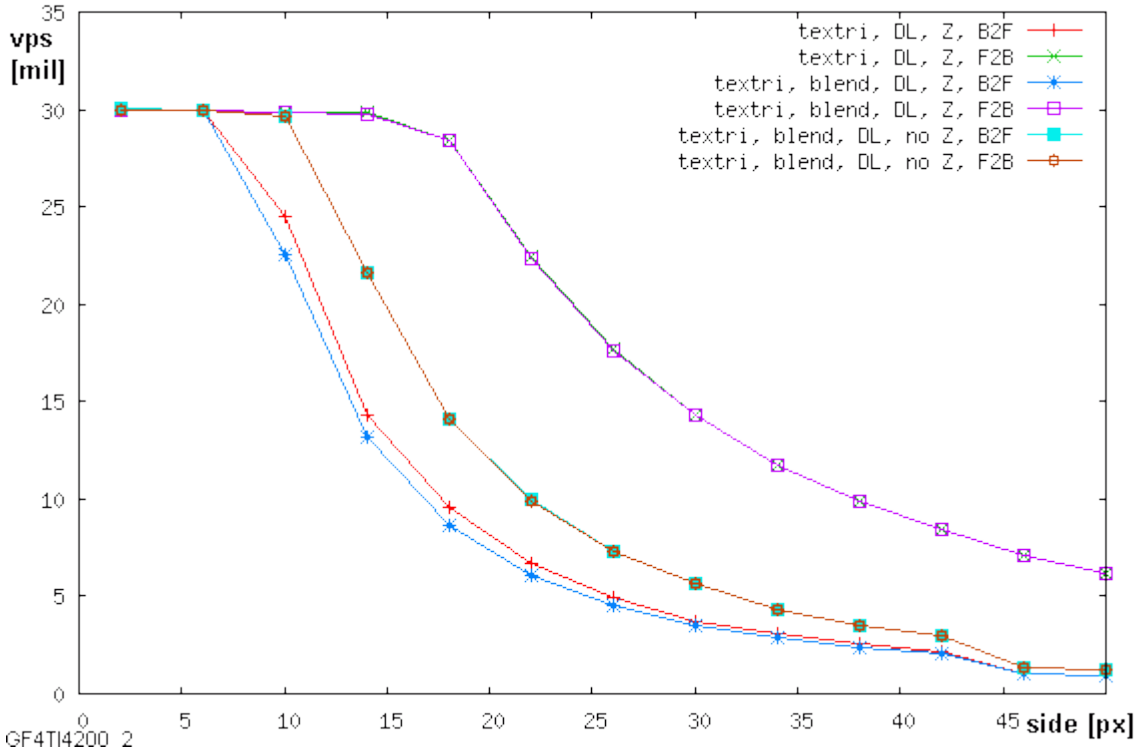
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



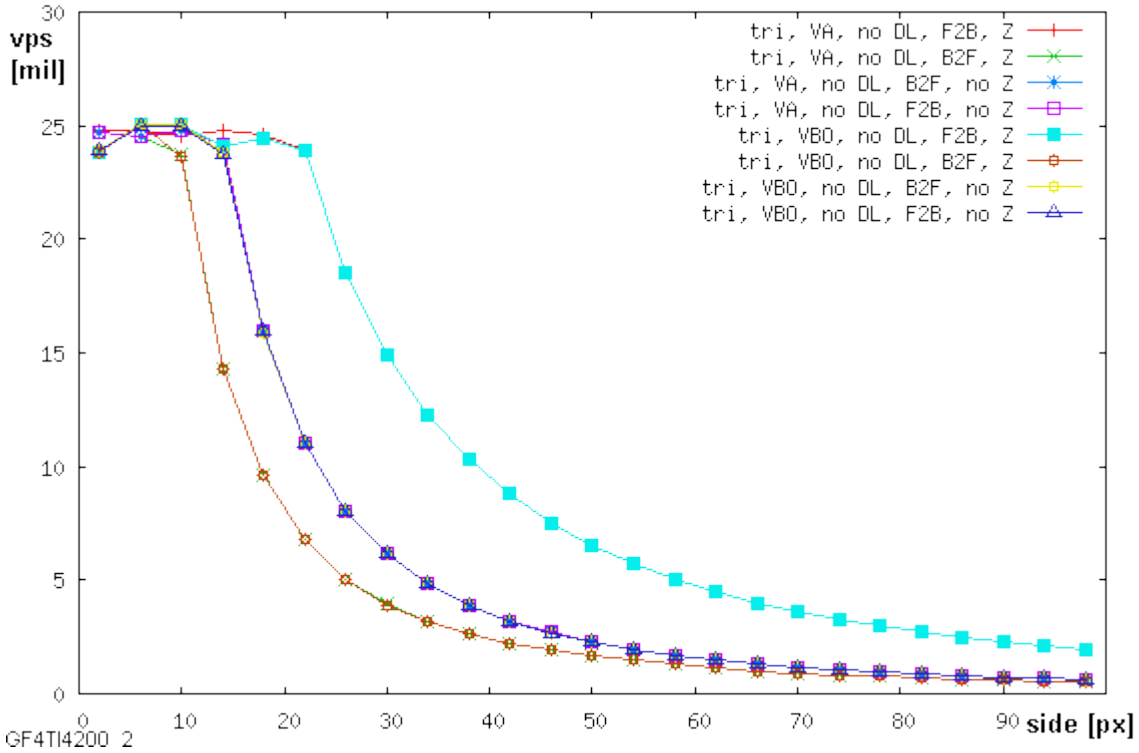
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

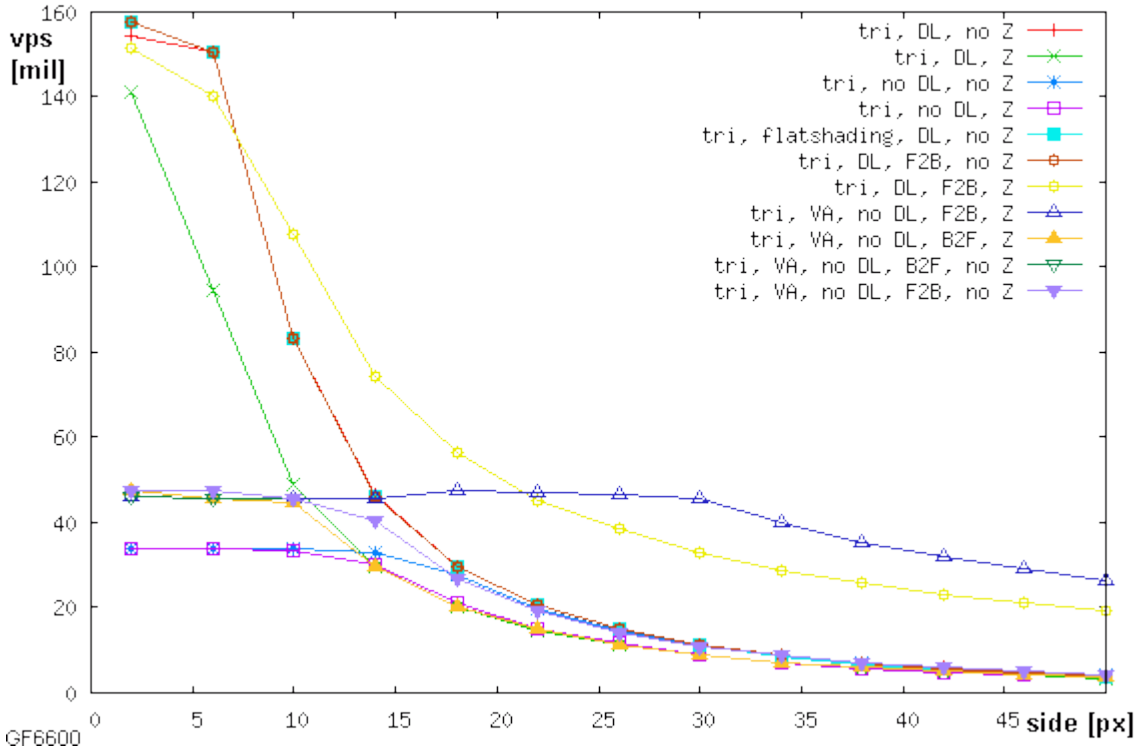


Test vertex arrays, vertex buffer objects

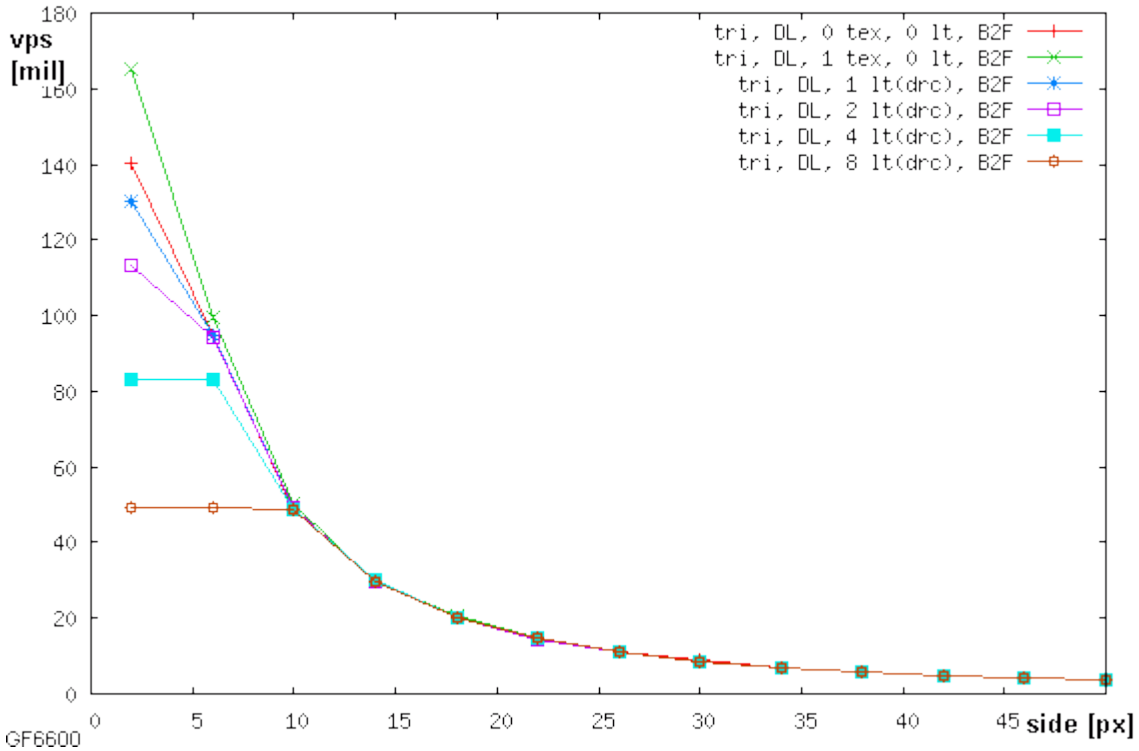


10.2.5 NVIDIA GeForce 6600 GT/PCI/SSE2/3DNOW!

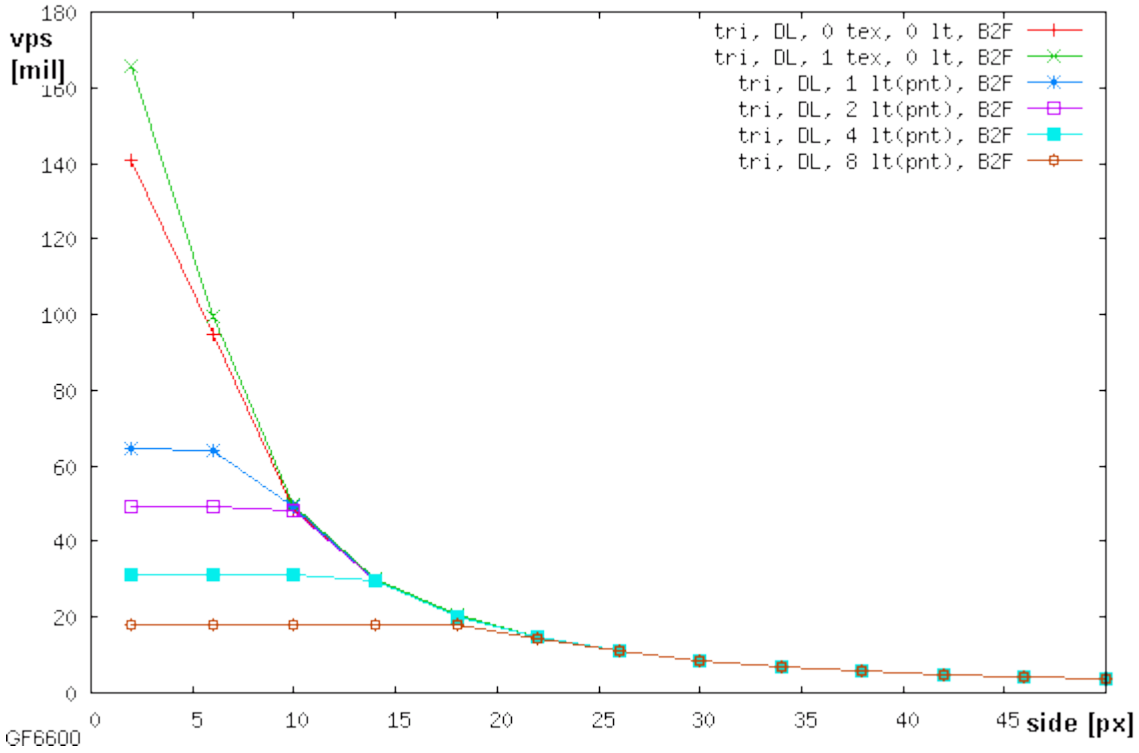
Test display listu a Z-bufferu (bez textur)



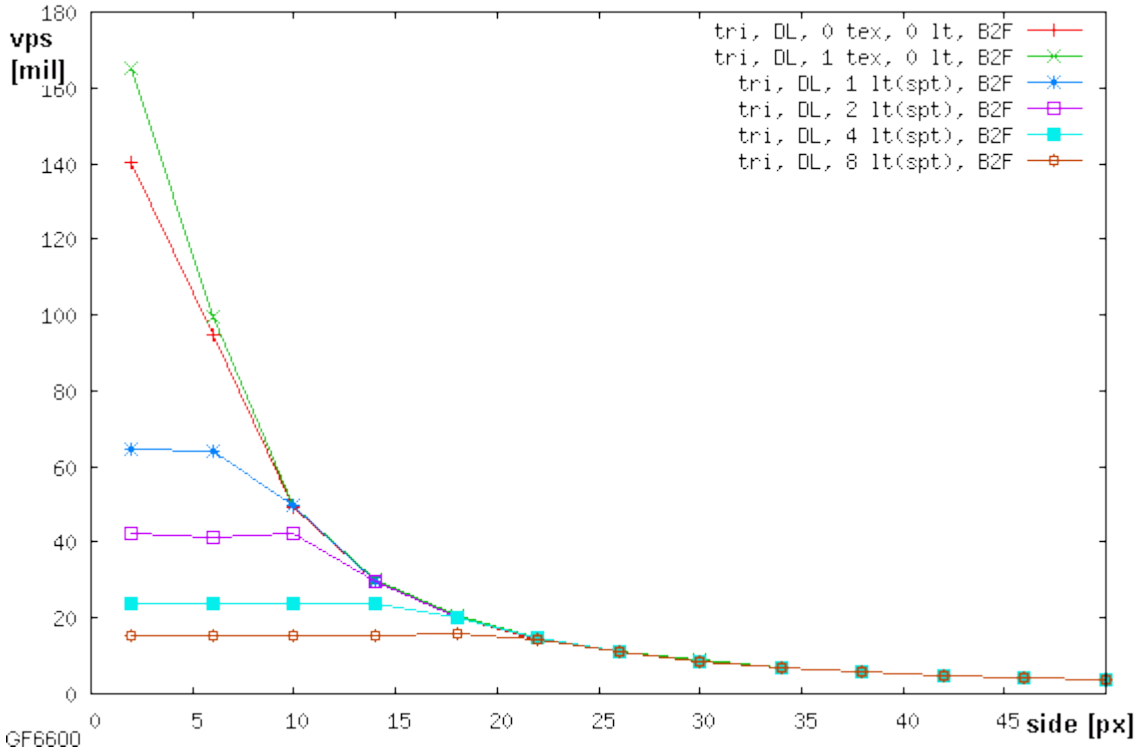
Test směrových světel (bez textur)



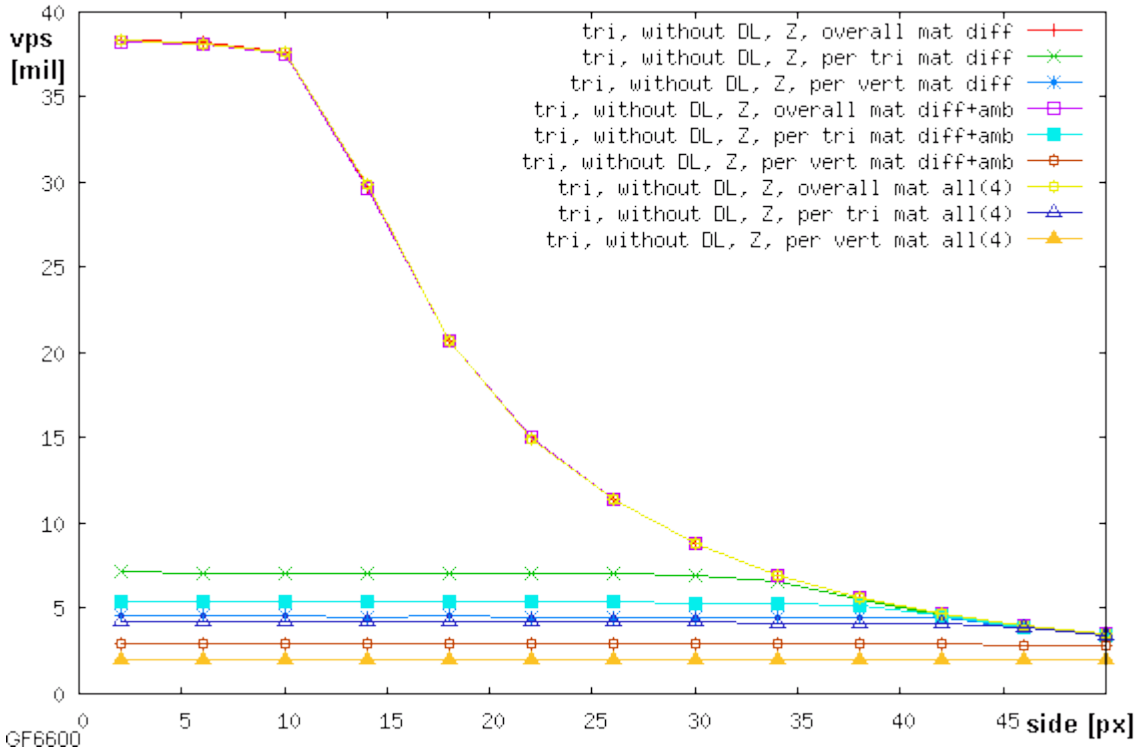
Test bodových světel (bez textur)



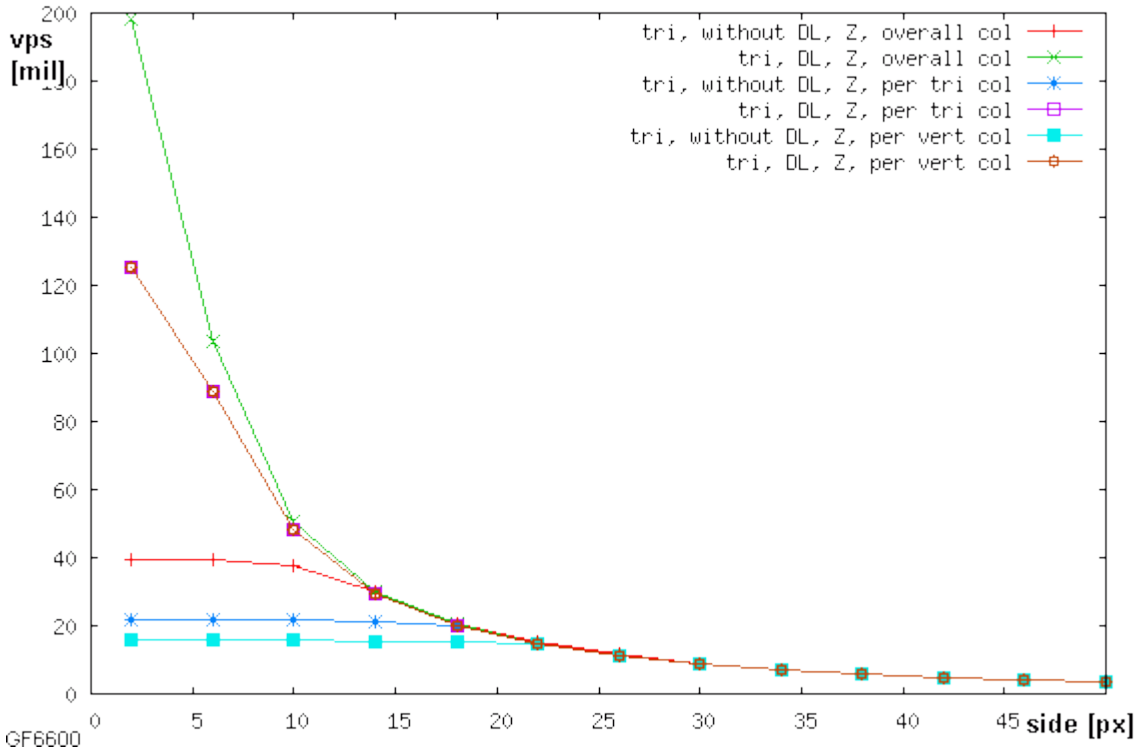
Test reflektorových světel (bez textur)



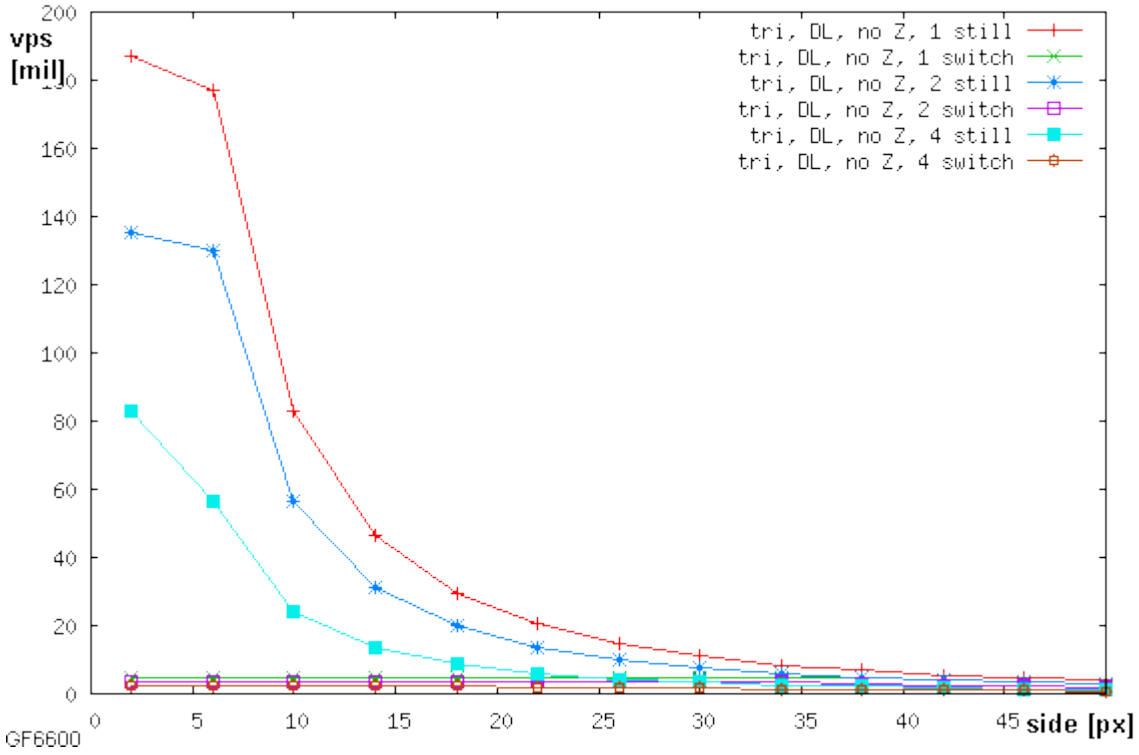
Test přepínání materiálů (bez textur)



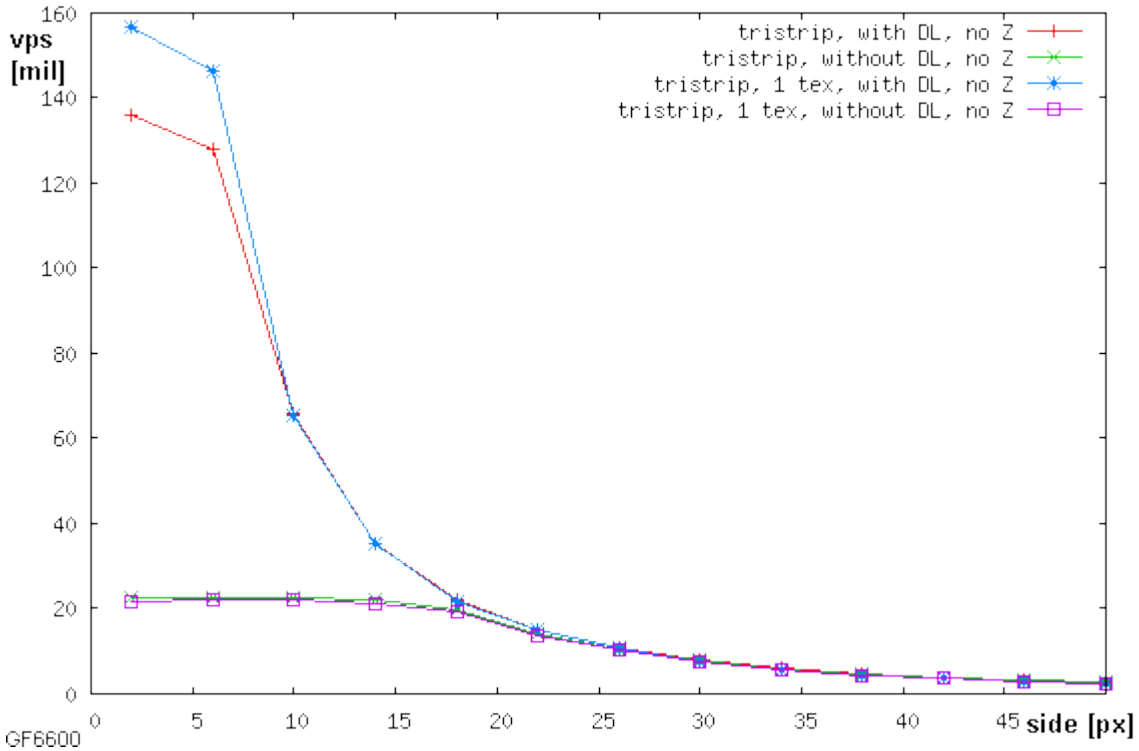
Test přepínání barev (bez textur)



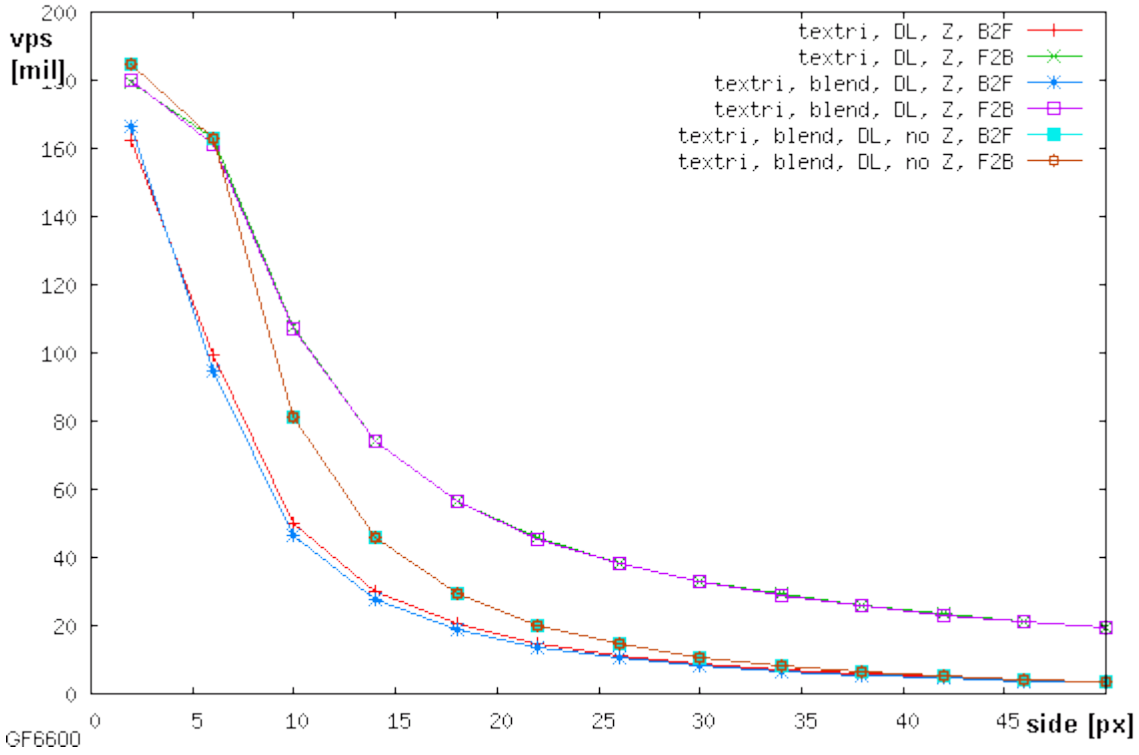
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



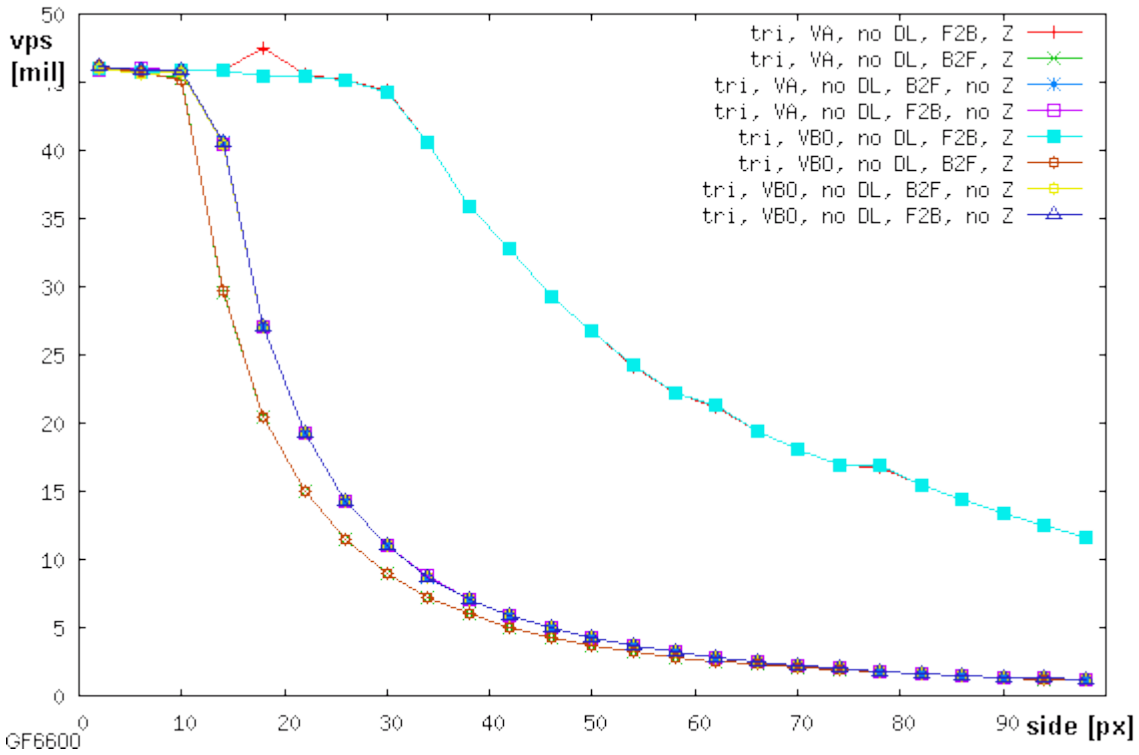
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

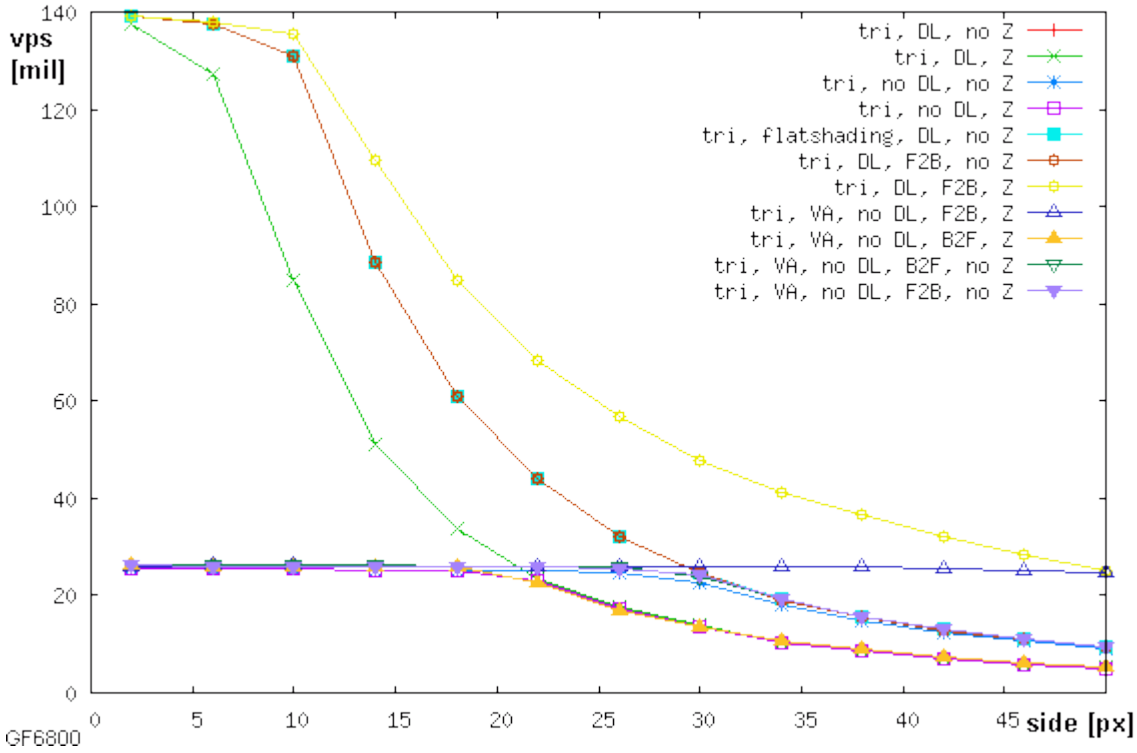


Test vertex arrays, vertex buffer objects

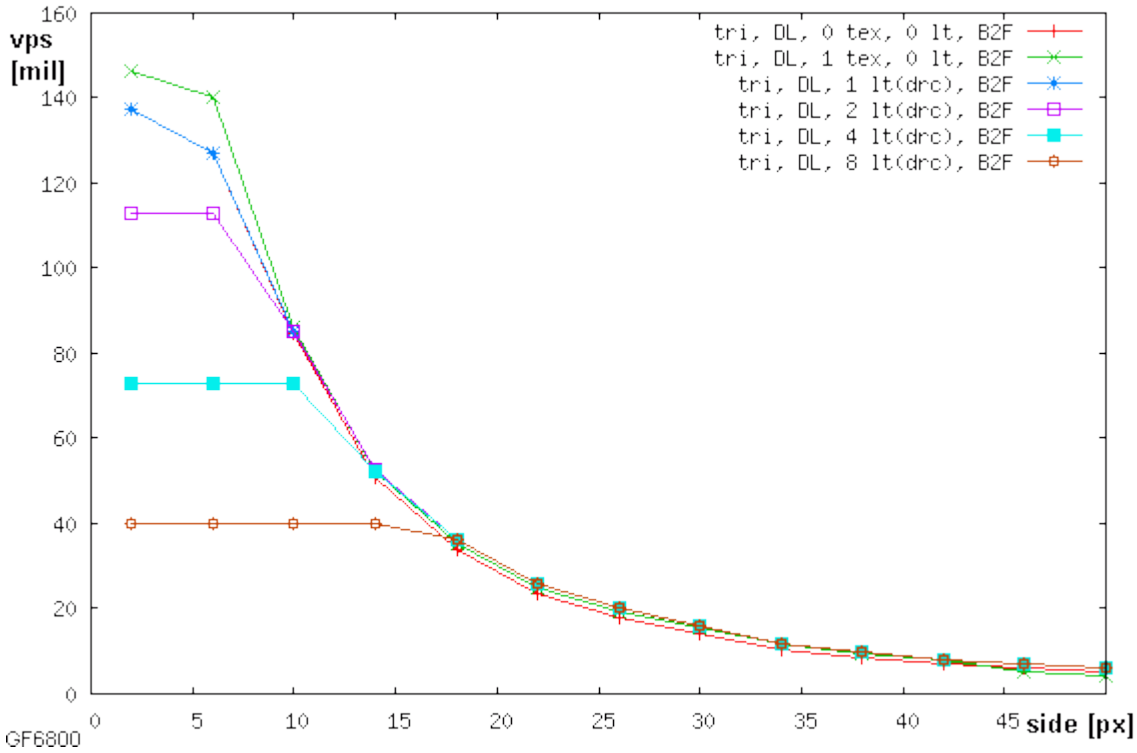


10.2.6 NVIDIA GeForce 6800/AGP/SSE2

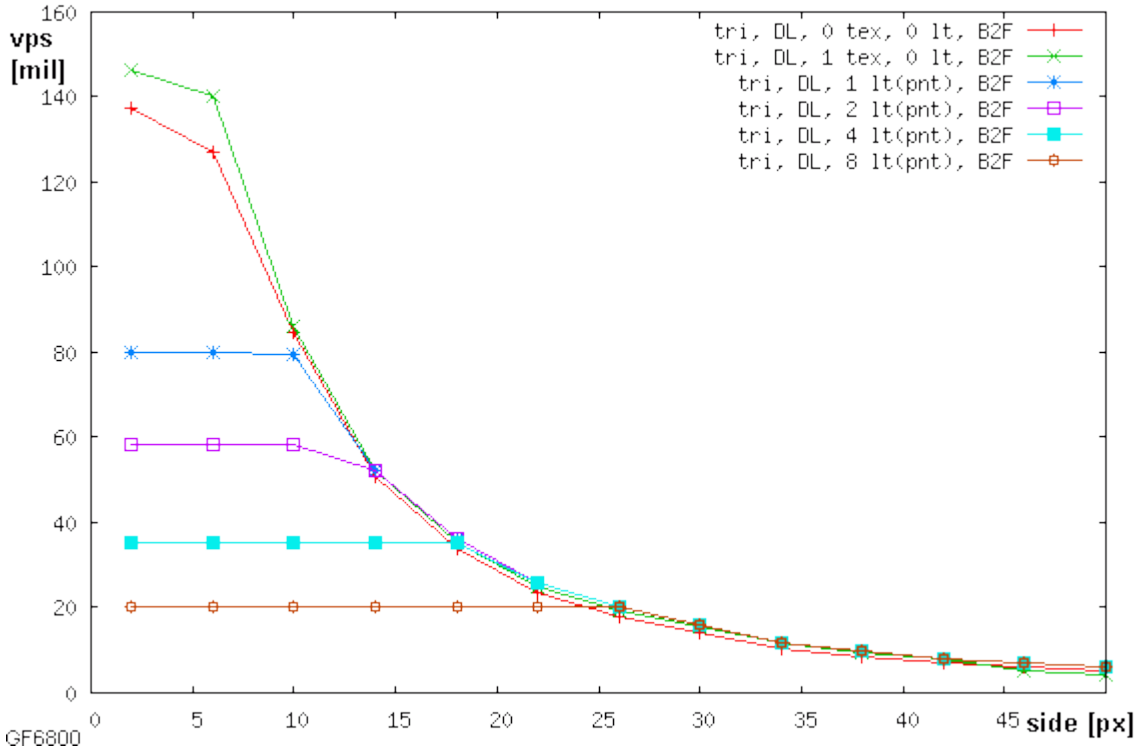
Test display listu a Z-bufferu (bez textur)



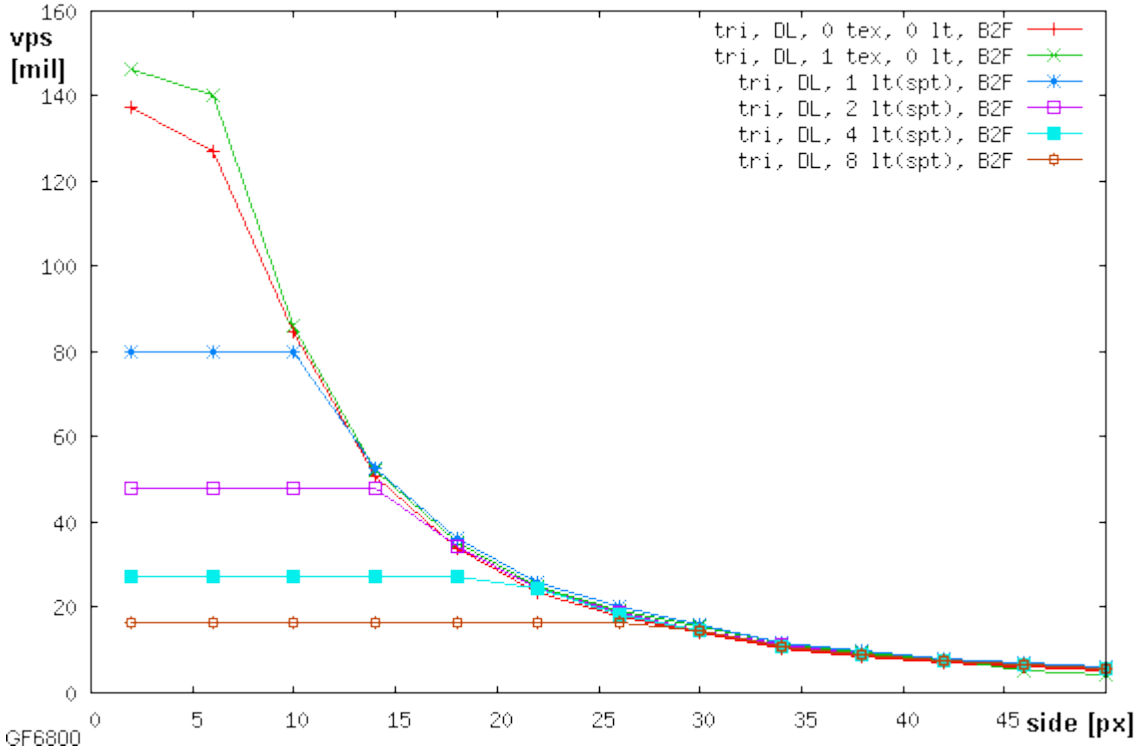
Test směrových světel (bez textur)



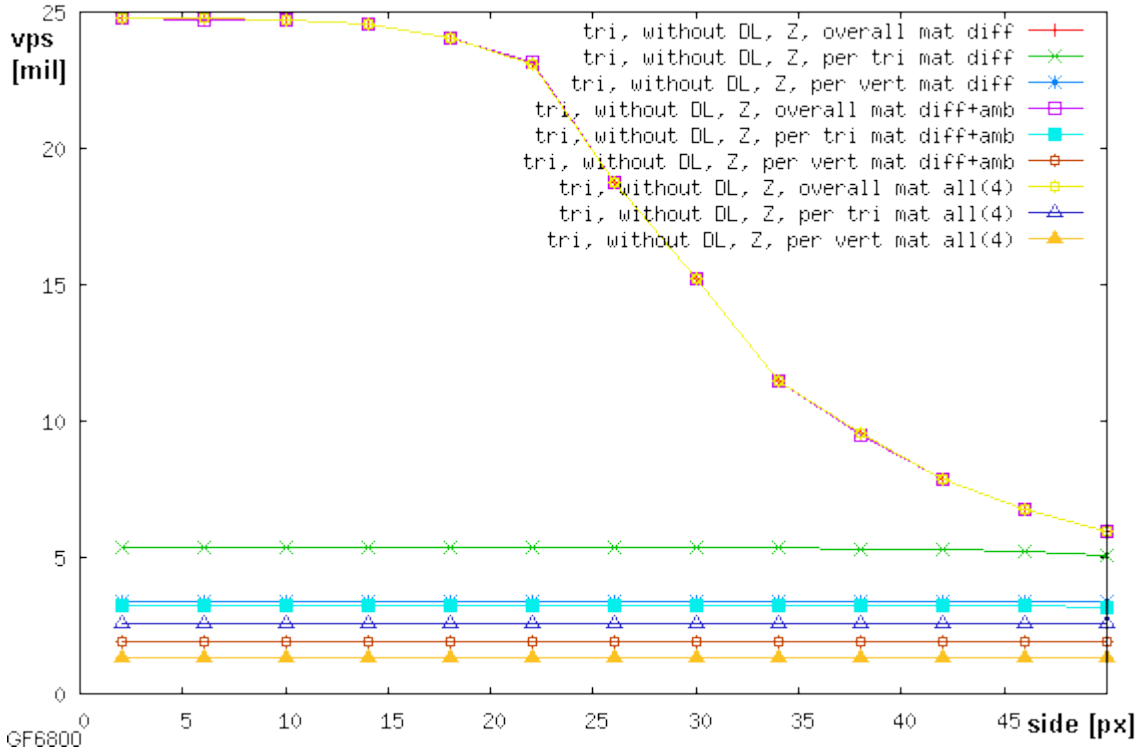
Test bodových světel (bez textur)



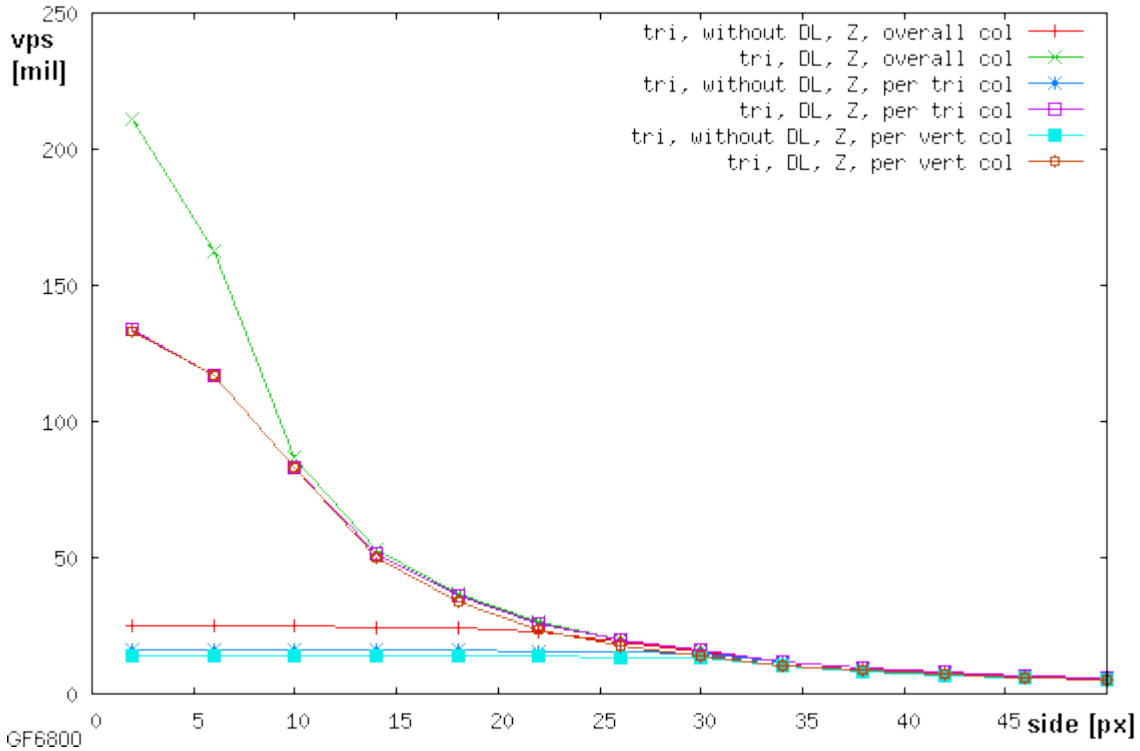
Test reflektorových světel (bez textur)



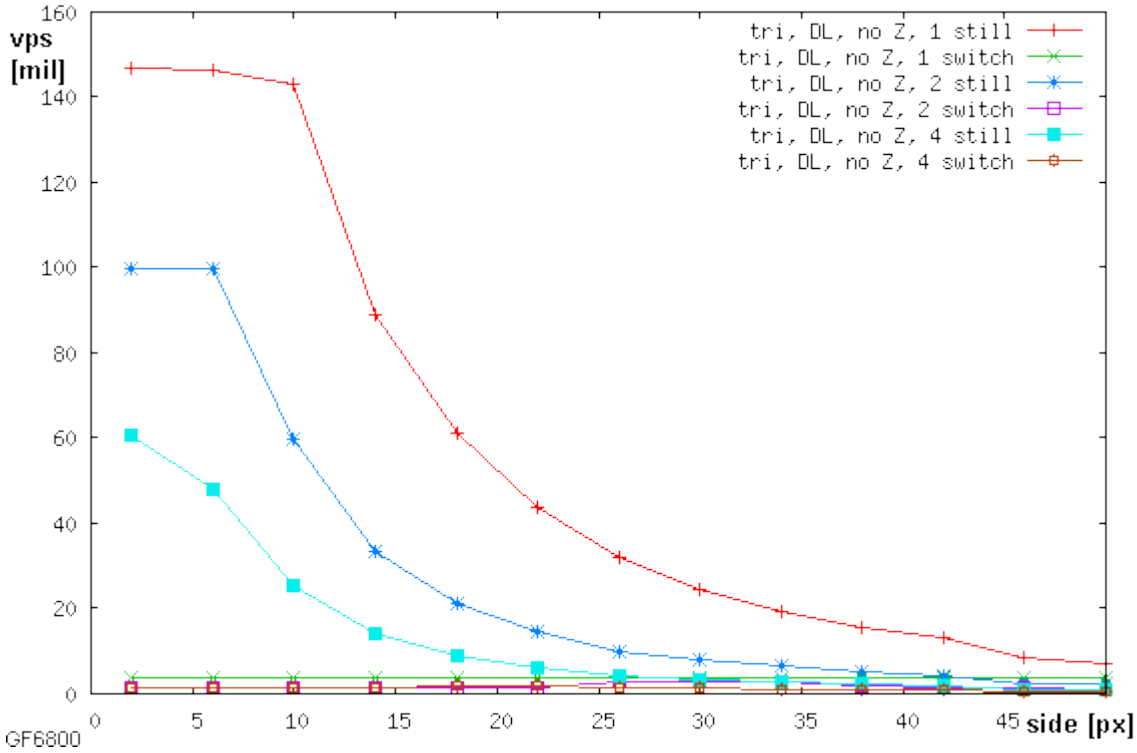
Test přepínání materiálů (bez textur)



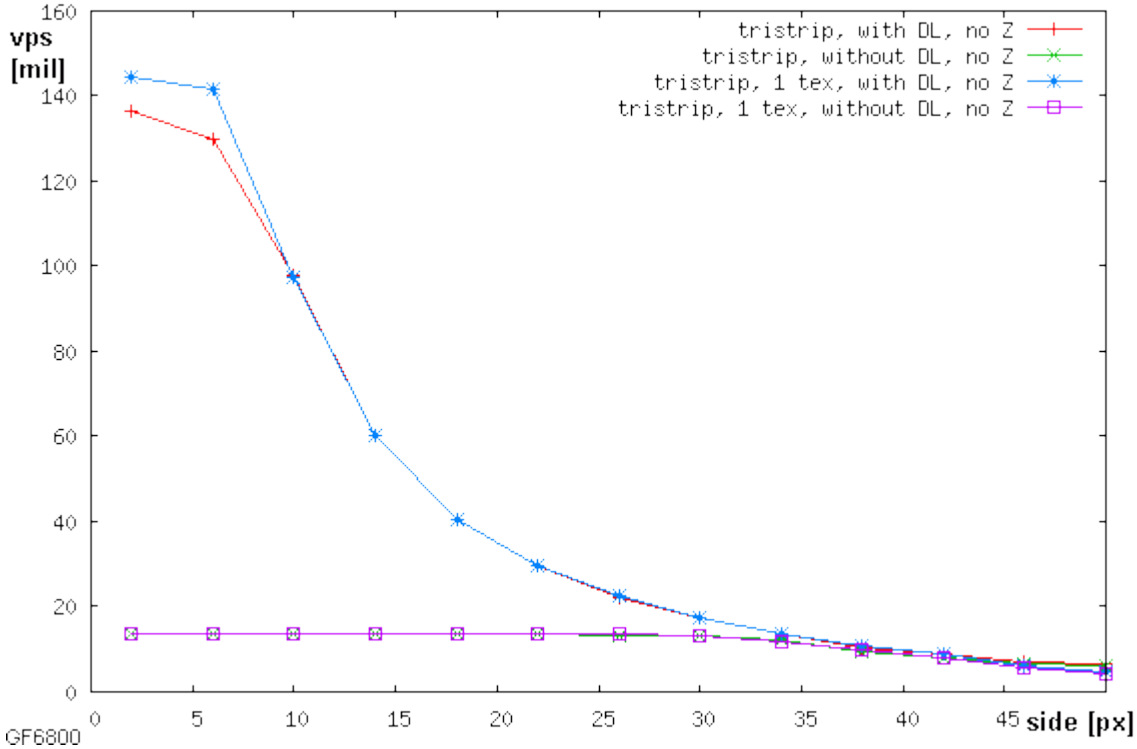
Test přepínání barev (bez textur)



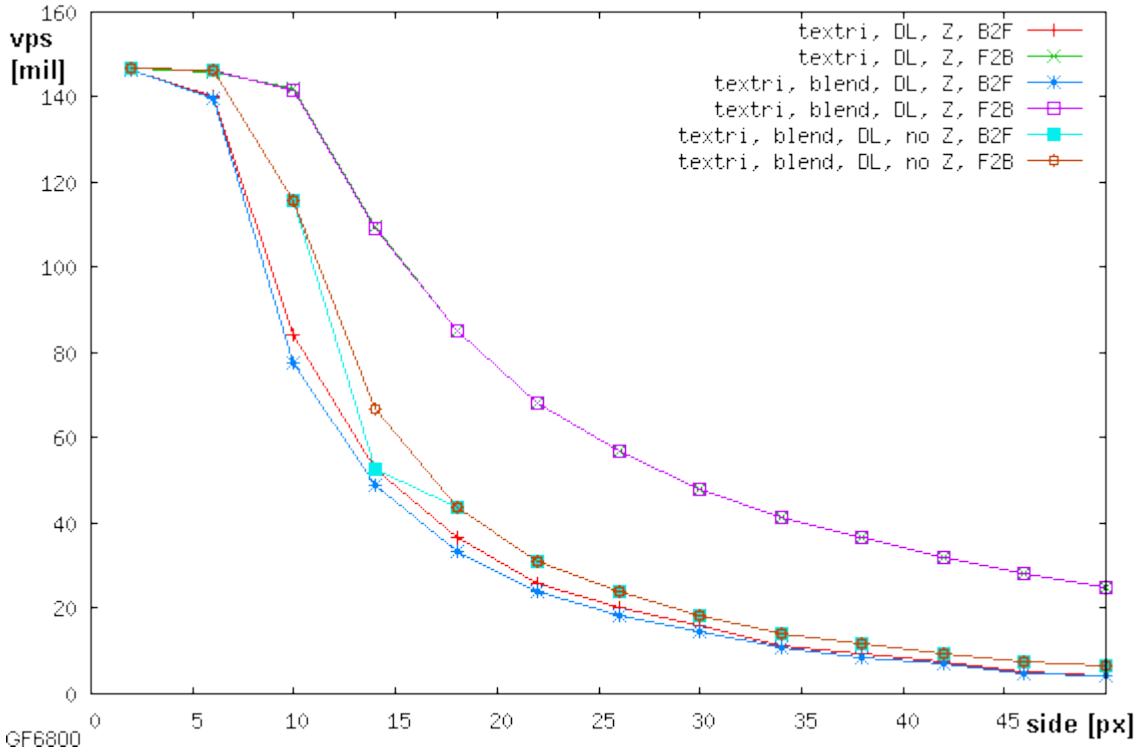
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



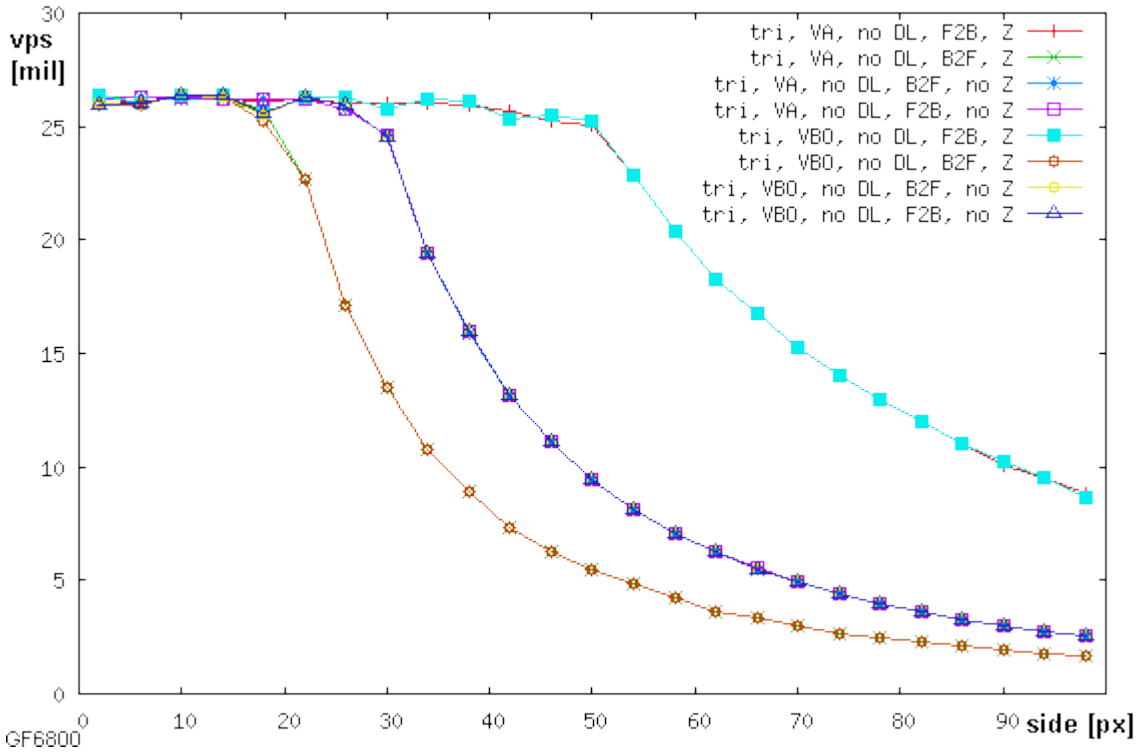
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

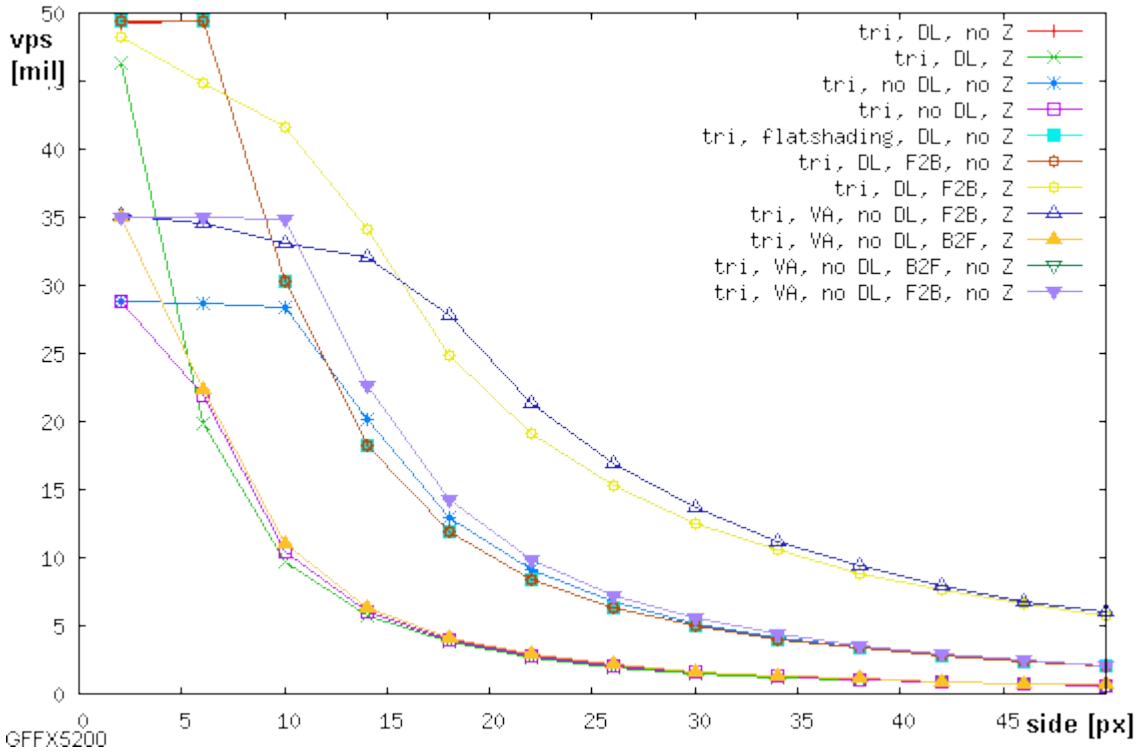


Test vertex arrays, vertex buffer objects

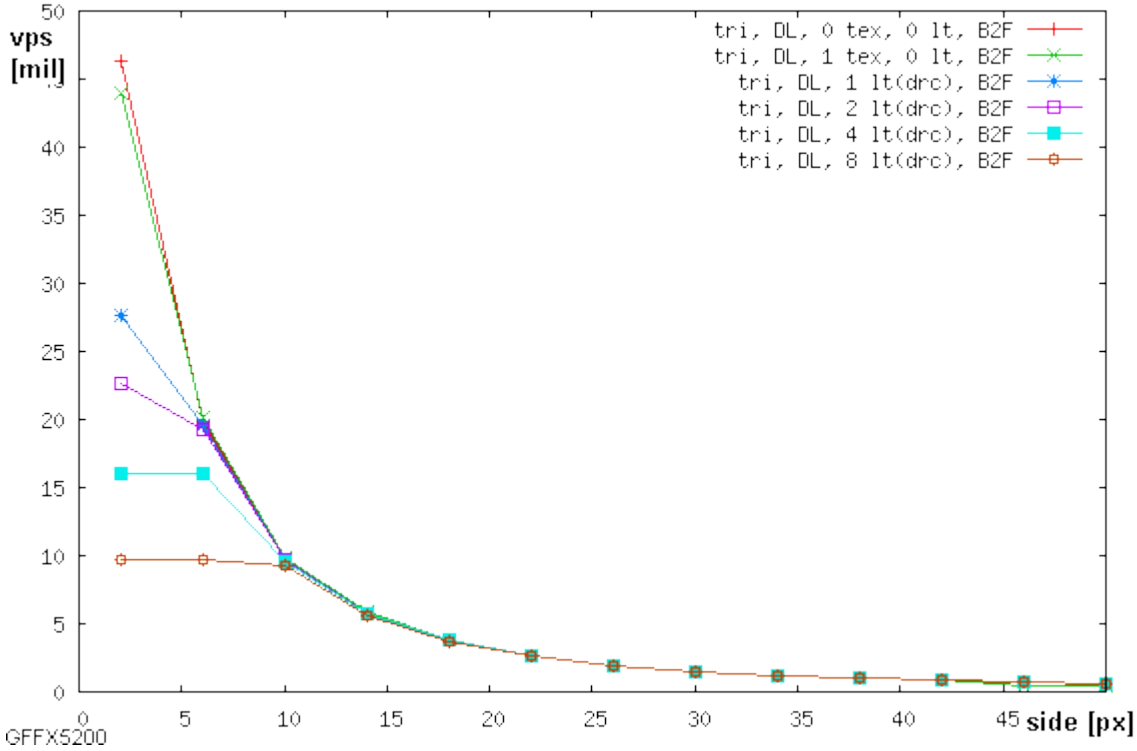


10.2.7 NVIDIA GeForce FX 5200/AGP/SSE2

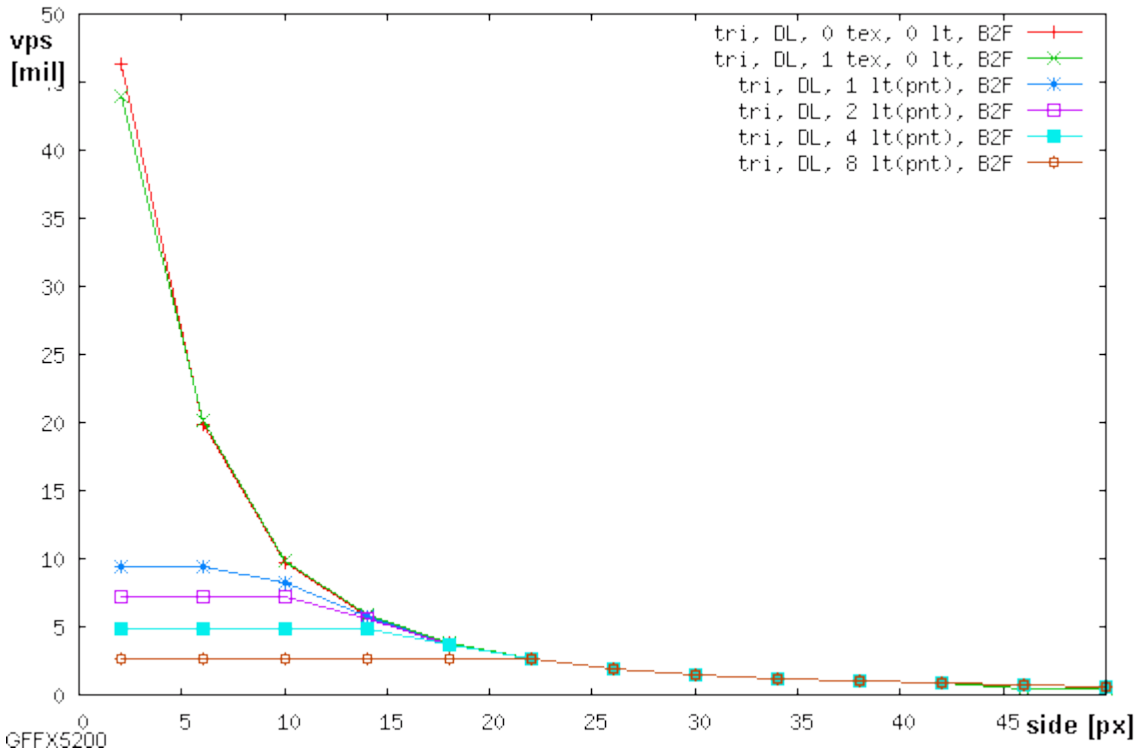
Test display listu a Z-bufferu (bez textur)



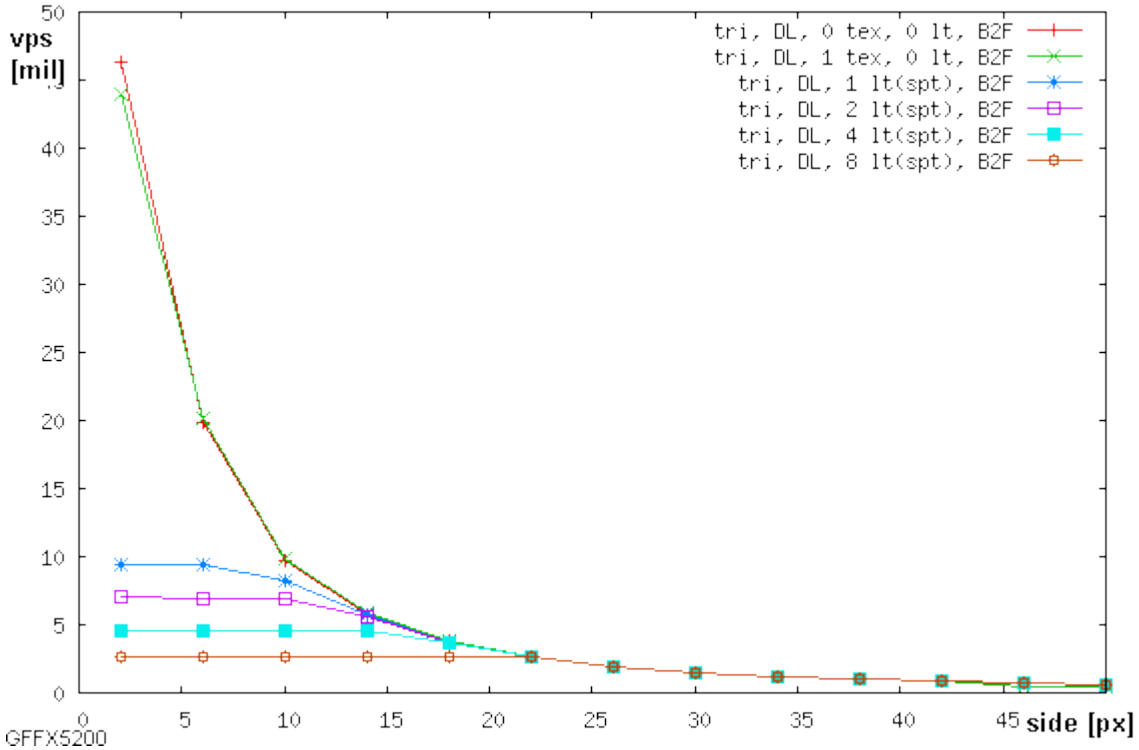
Test směrových světel (bez textur)



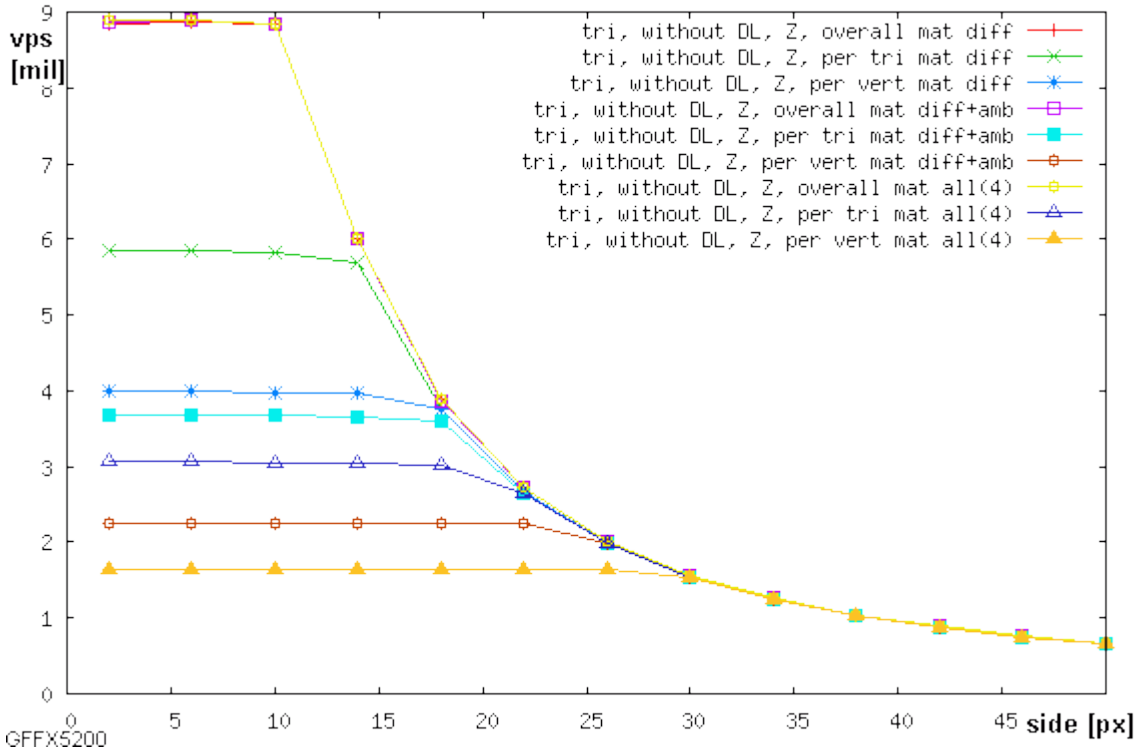
Test bodových světel (bez textur)



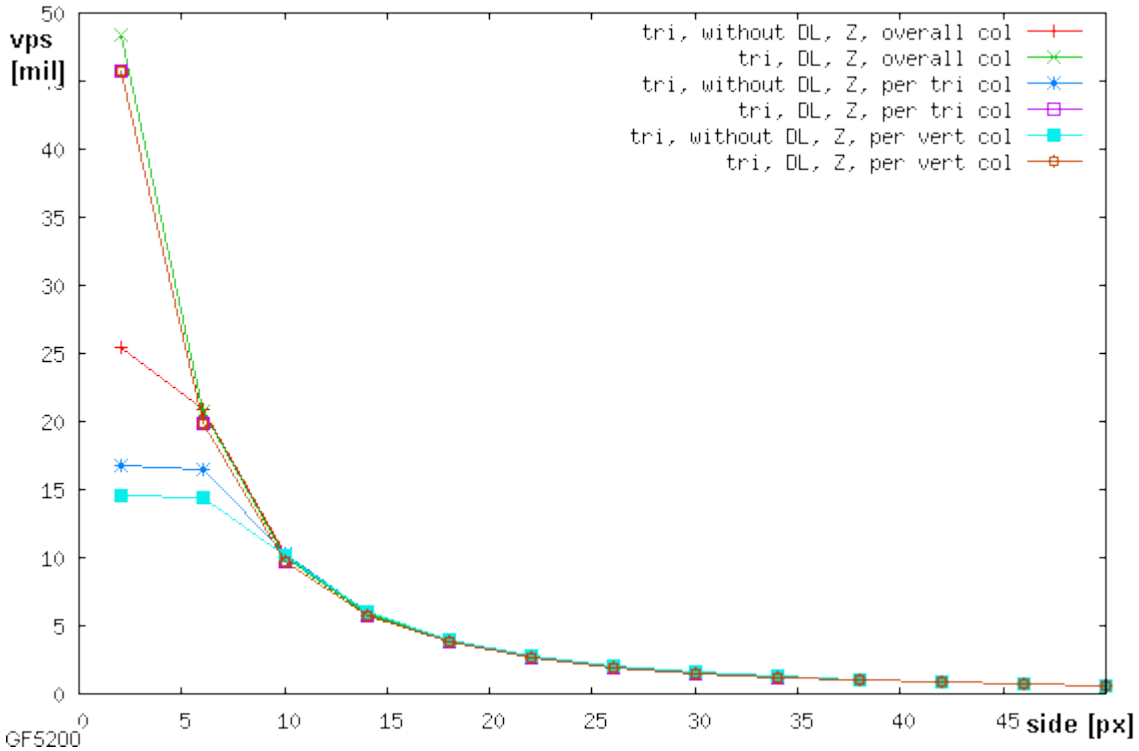
Test reflektorových světel (bez textur)



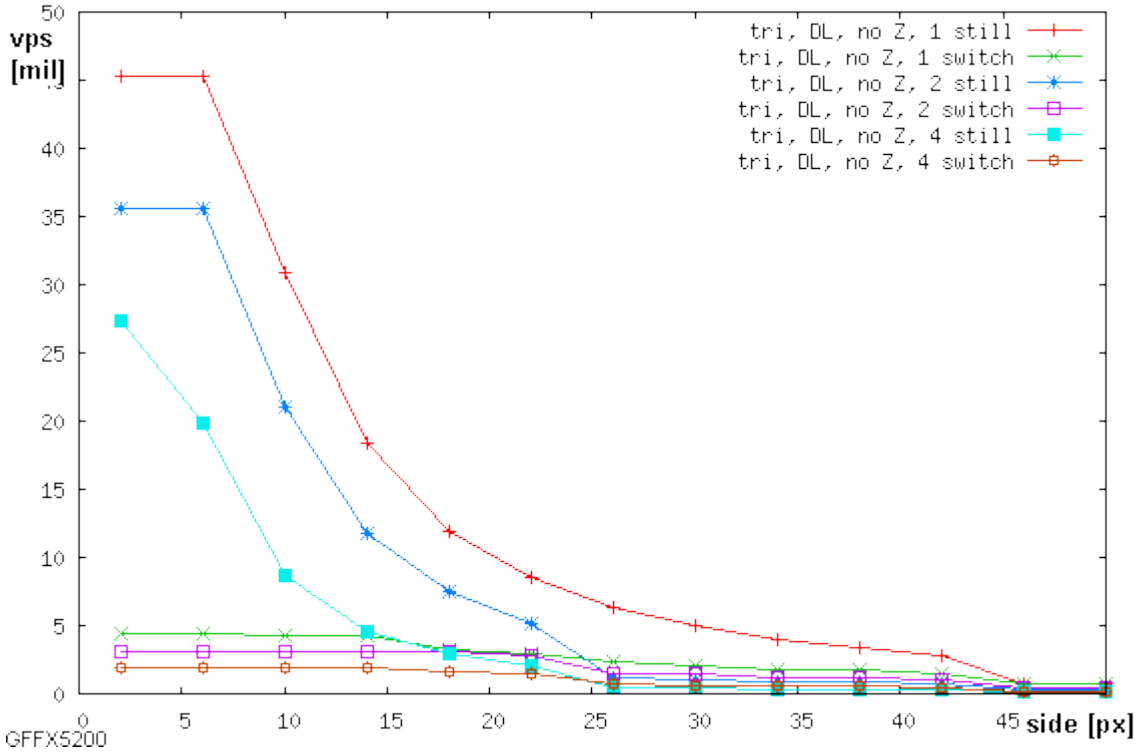
Test přepínání materiálů (bez textur)



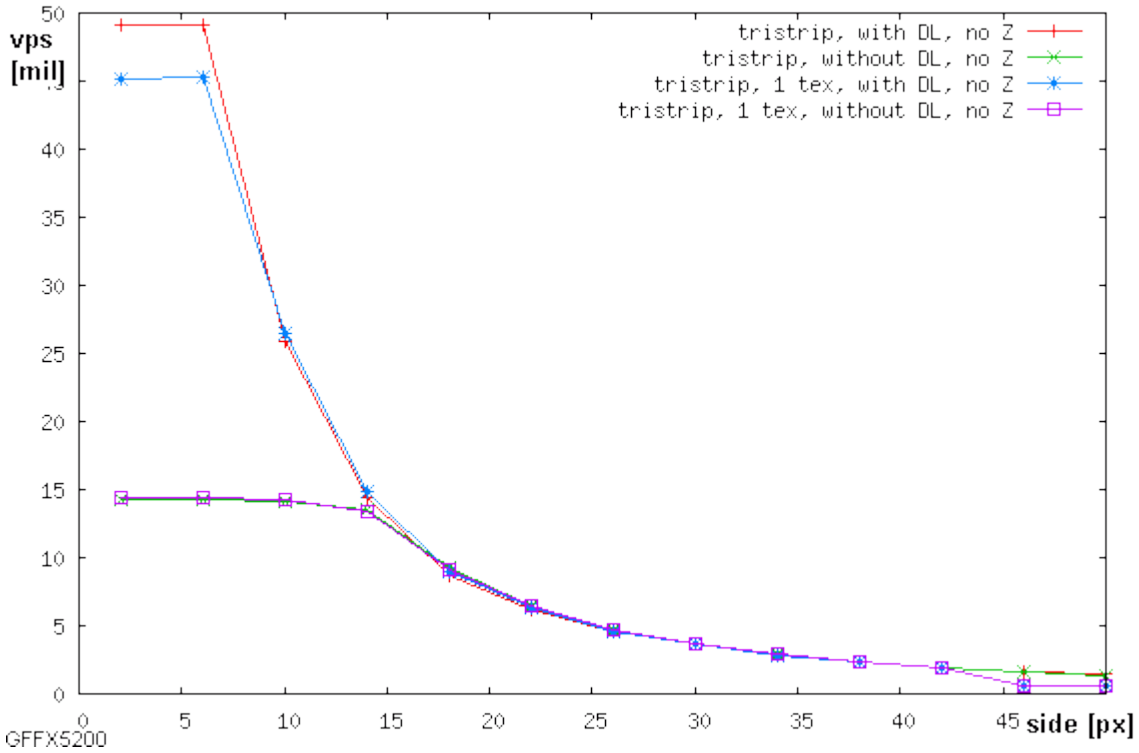
Test přepínání barev (bez textur)



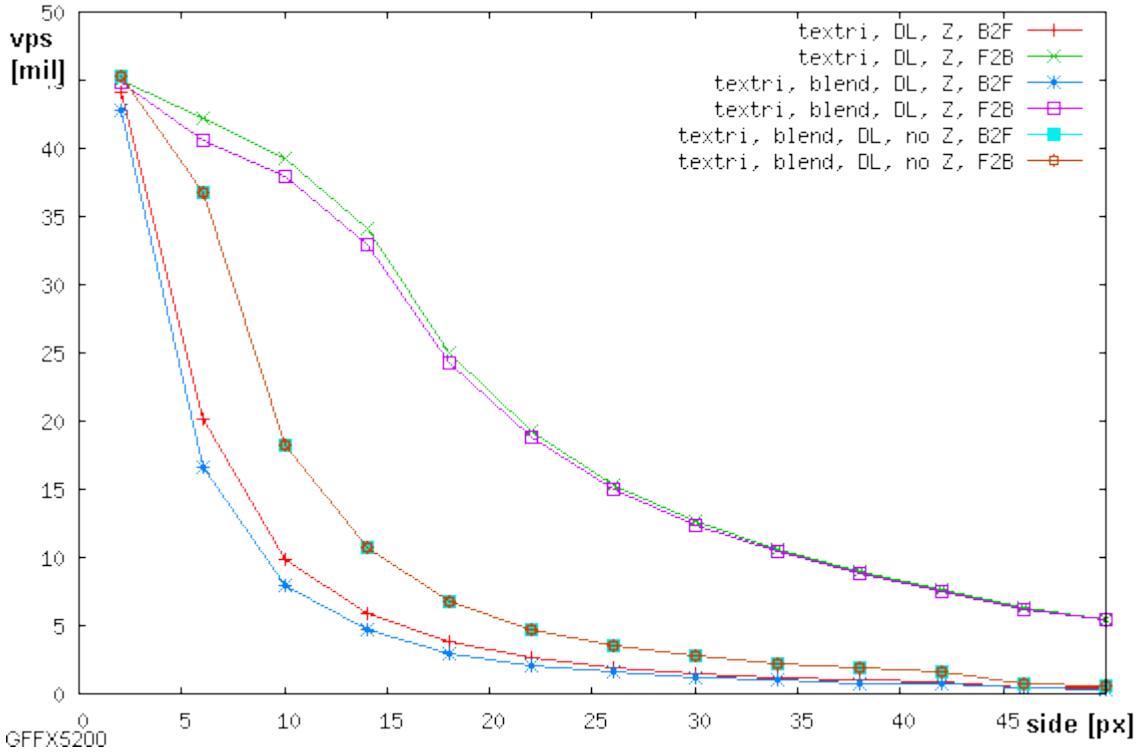
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



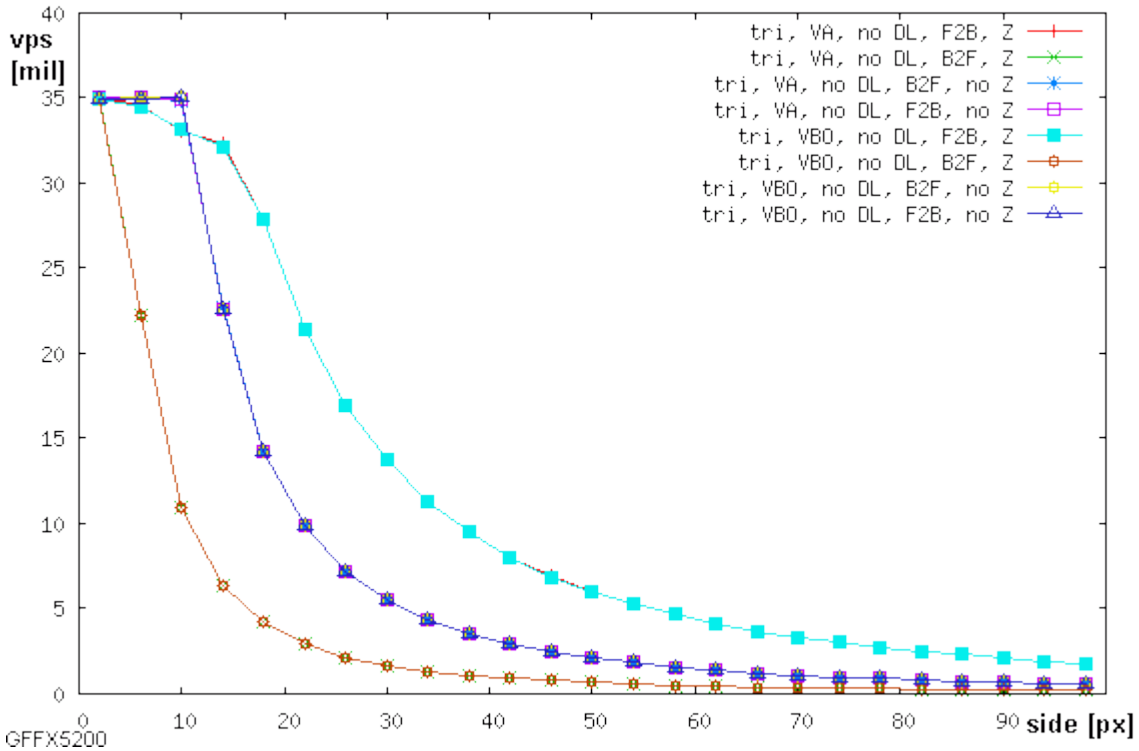
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

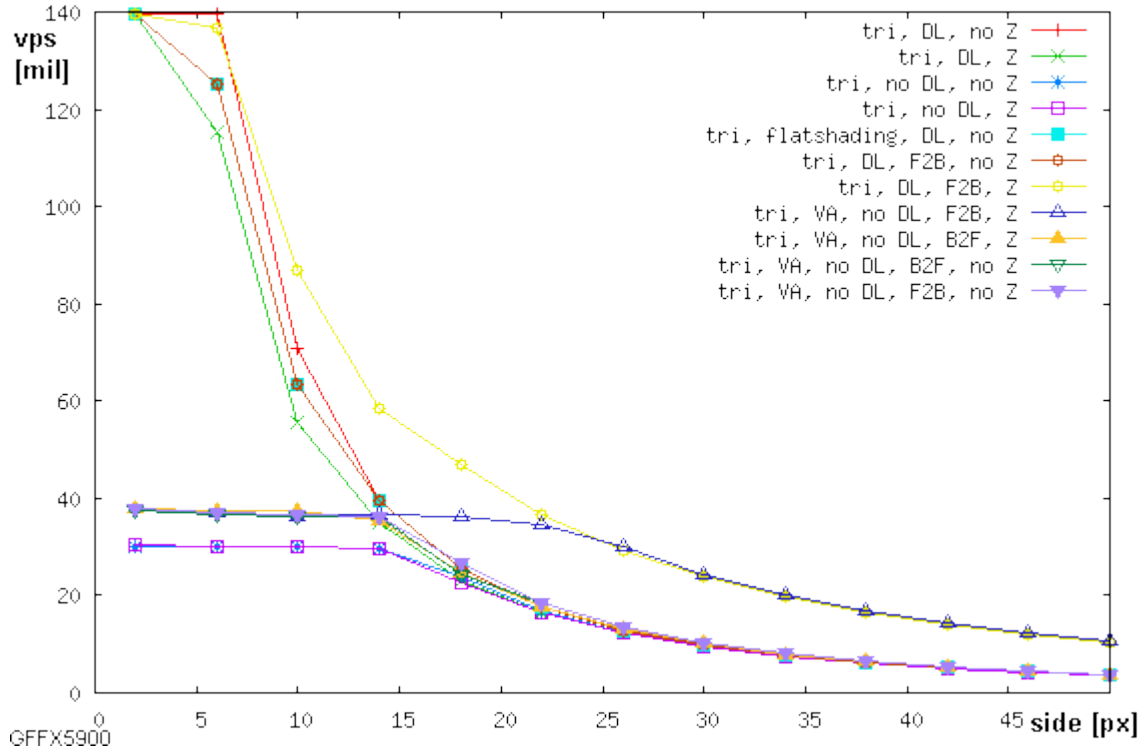


Test vertex arrays, vertex buffer objects

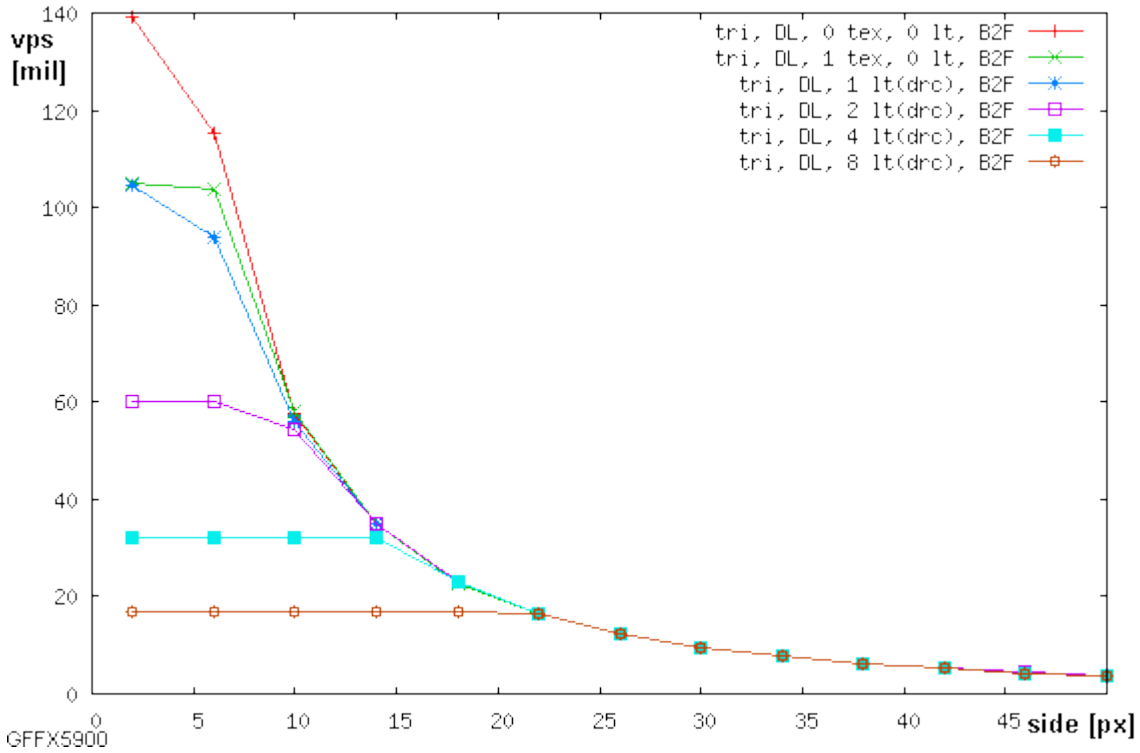


10.2.8 NVIDIA GeForce FX 5900XT/AGP/SSE2

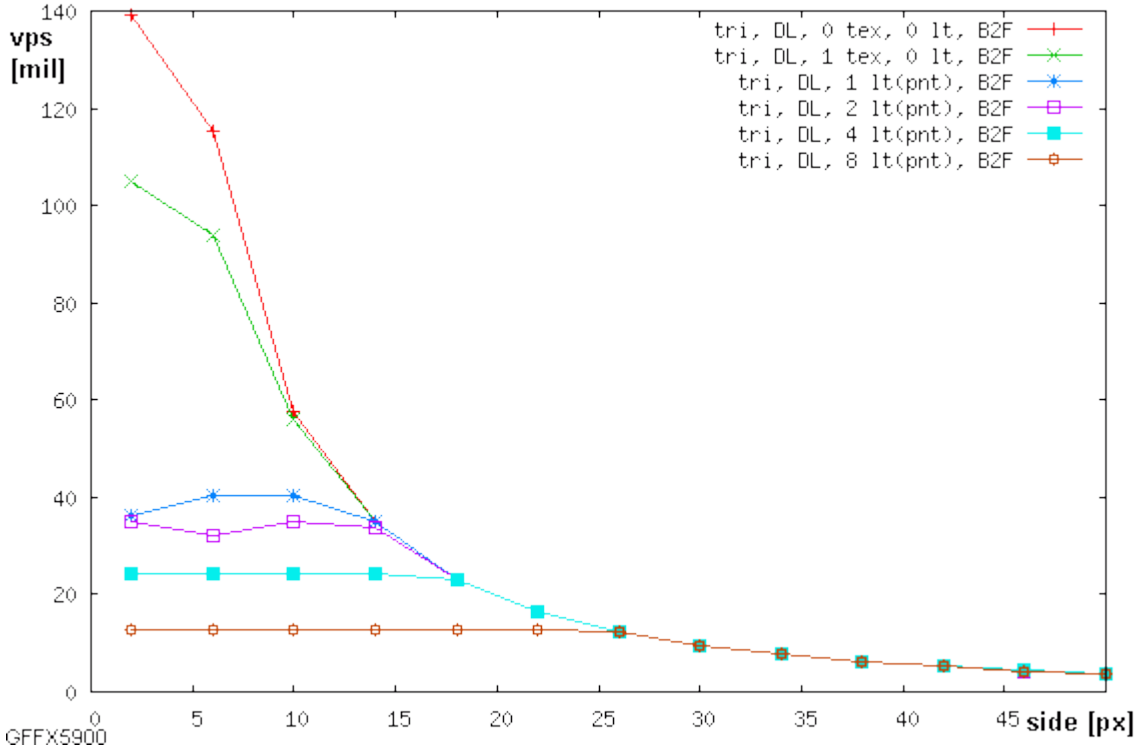
Test display listu a Z-bufferu (bez textur)



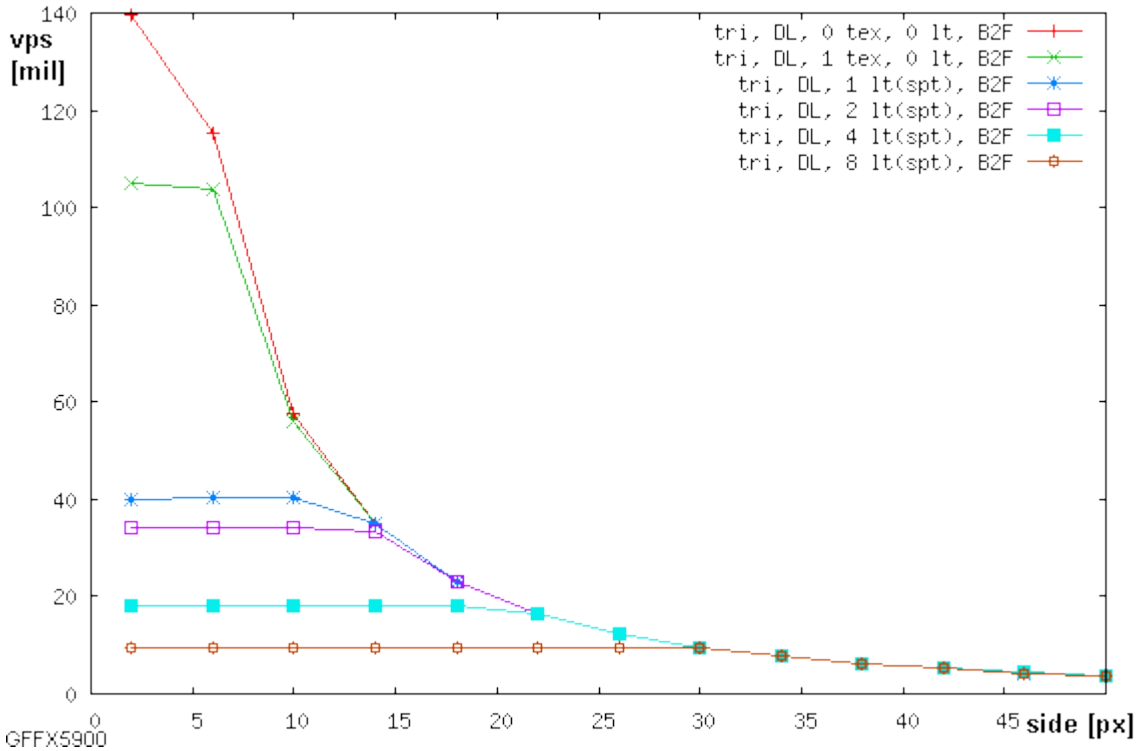
Test směrových světel (bez textur)



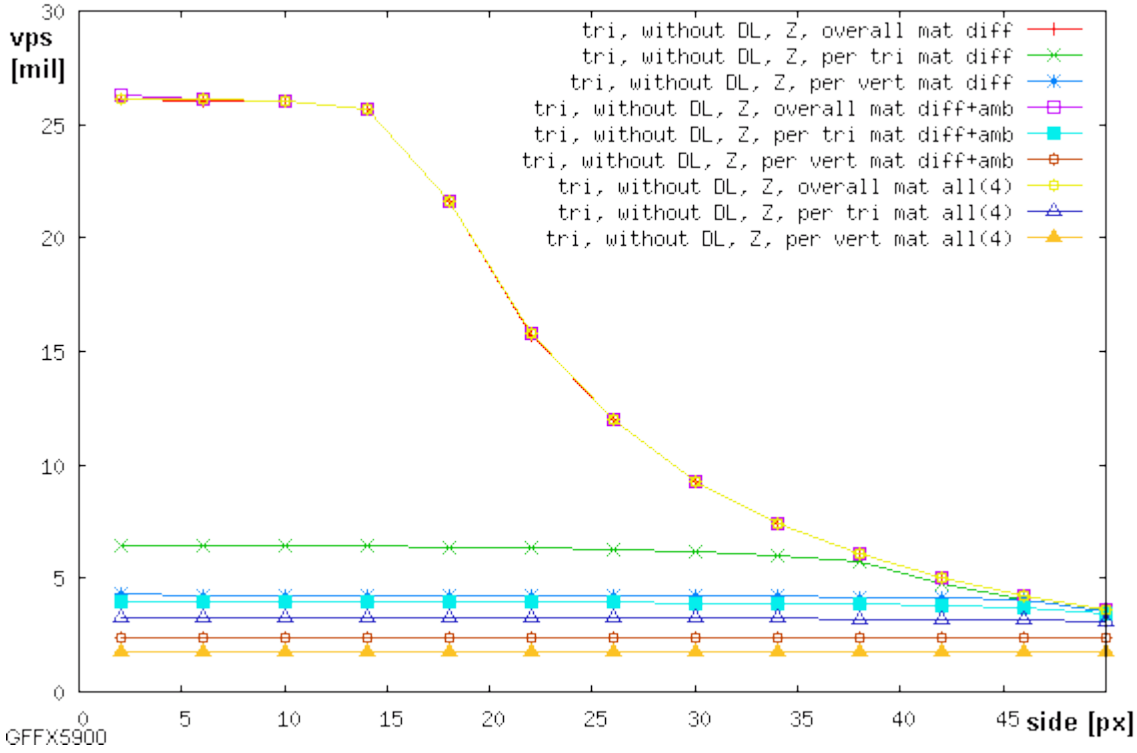
Test bodových světel (bez textur)



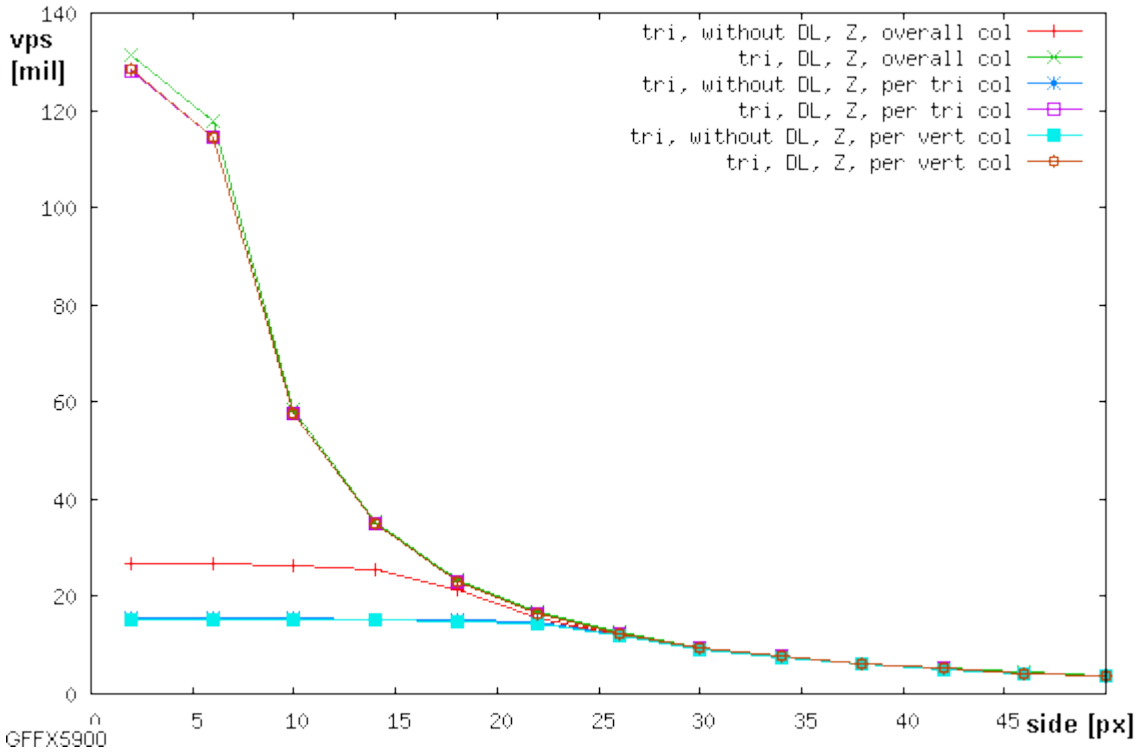
Test reflektorových světel (bez textur)



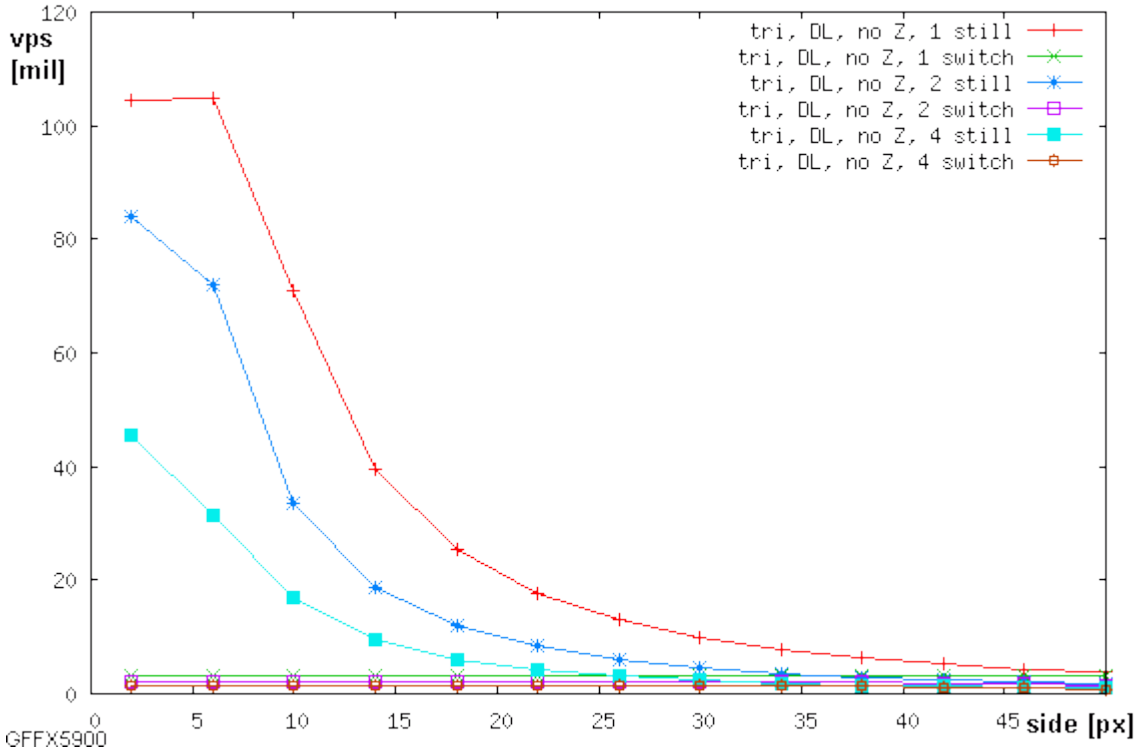
Test přepínání materiálů (bez textur)



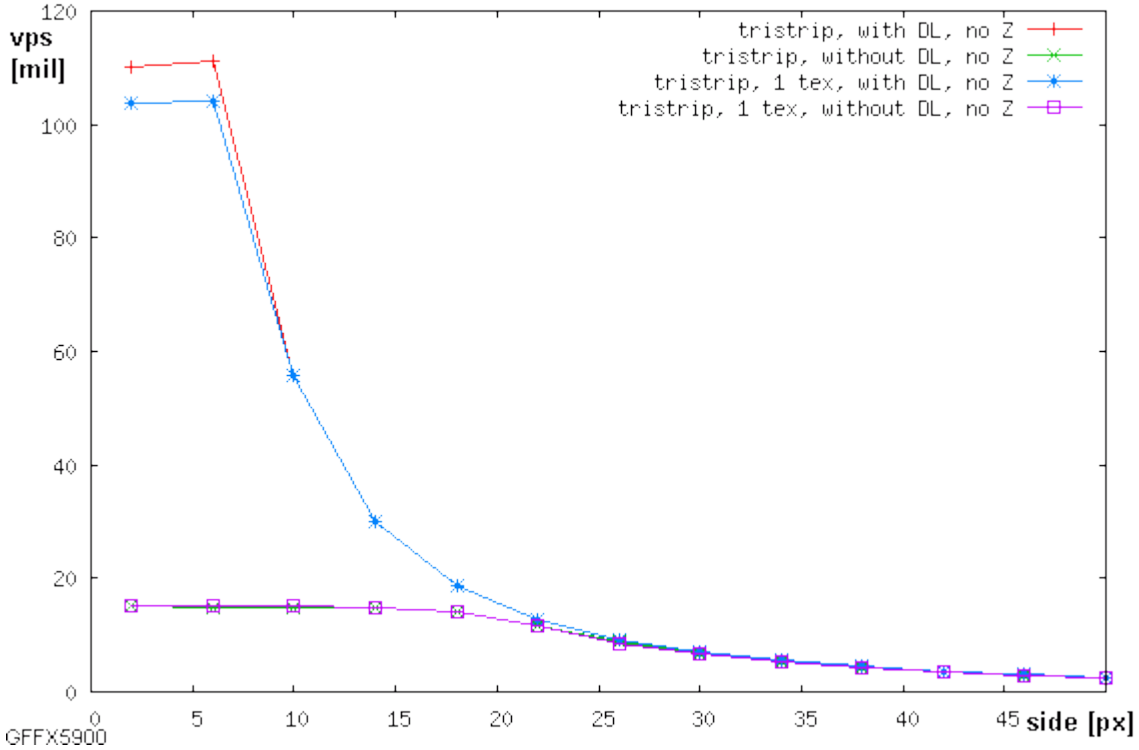
Test přepínání barev (bez textur)



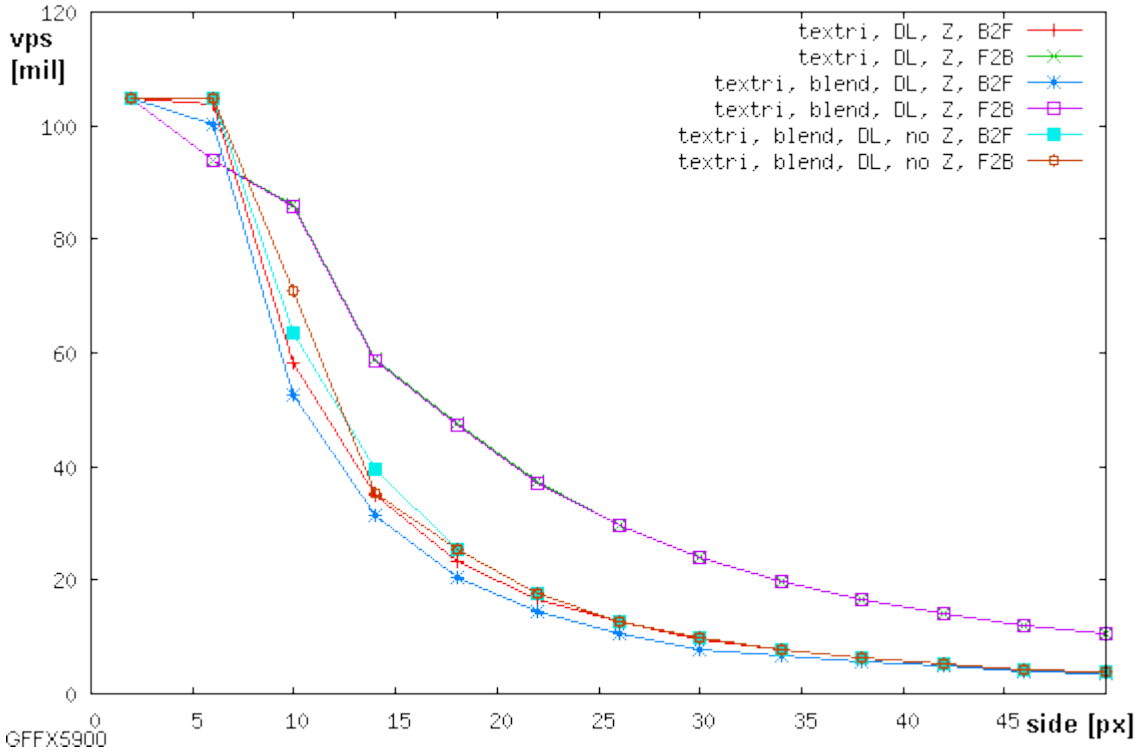
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



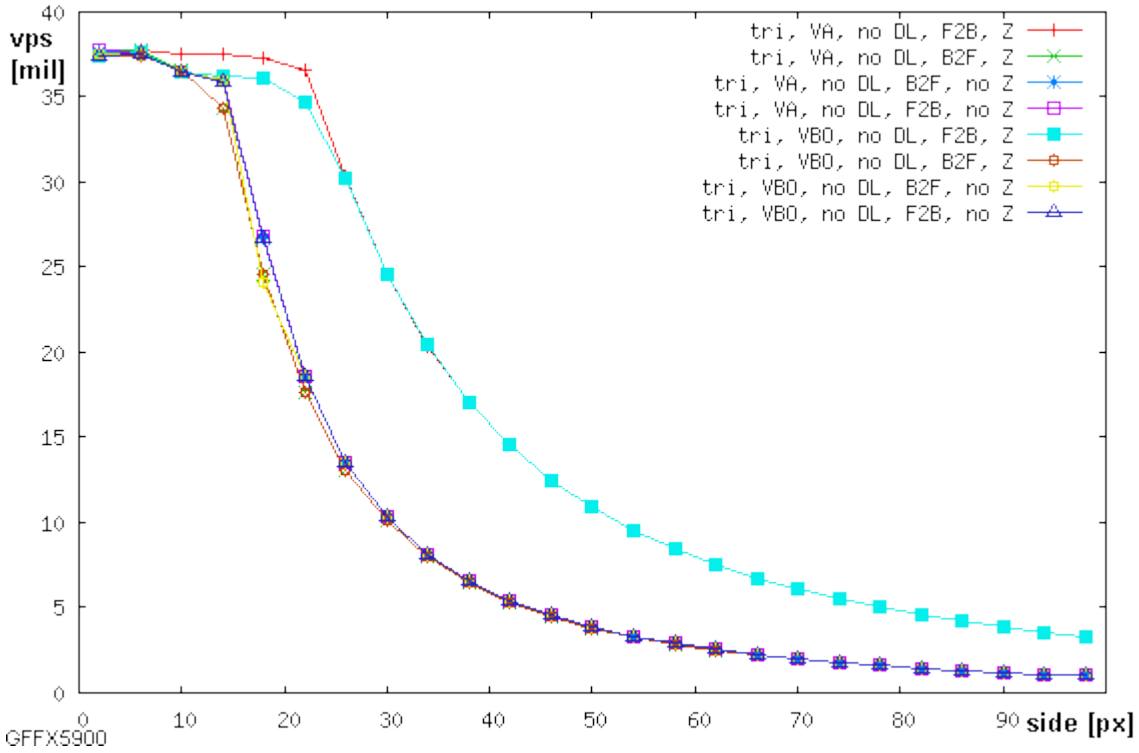
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

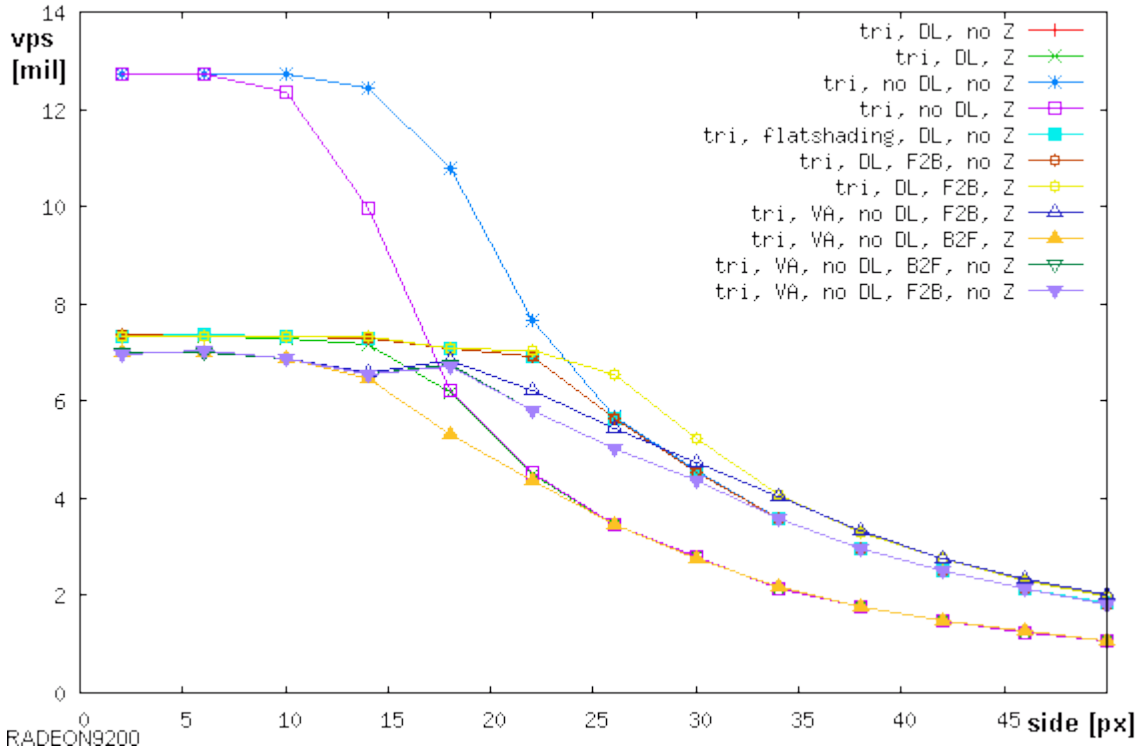


Test vertex arrays, vertex buffer objects

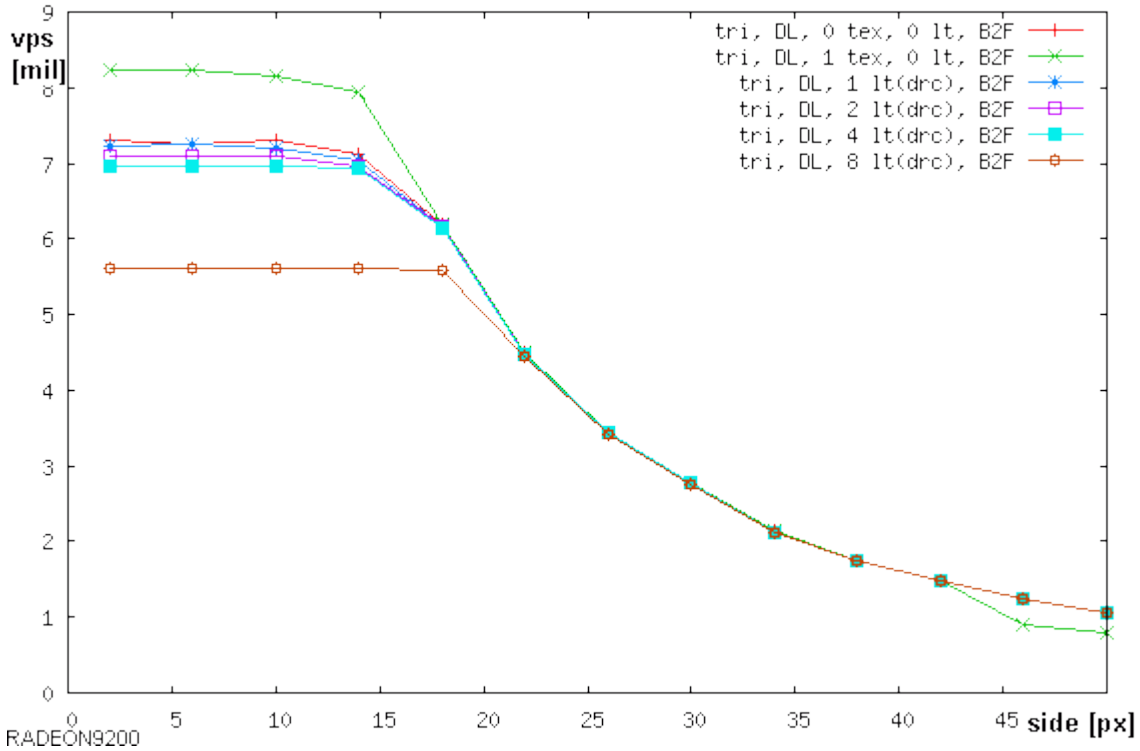


10.2.9 ATI Radeon 9200 DDR x86/MMX/3DNow!/SSE

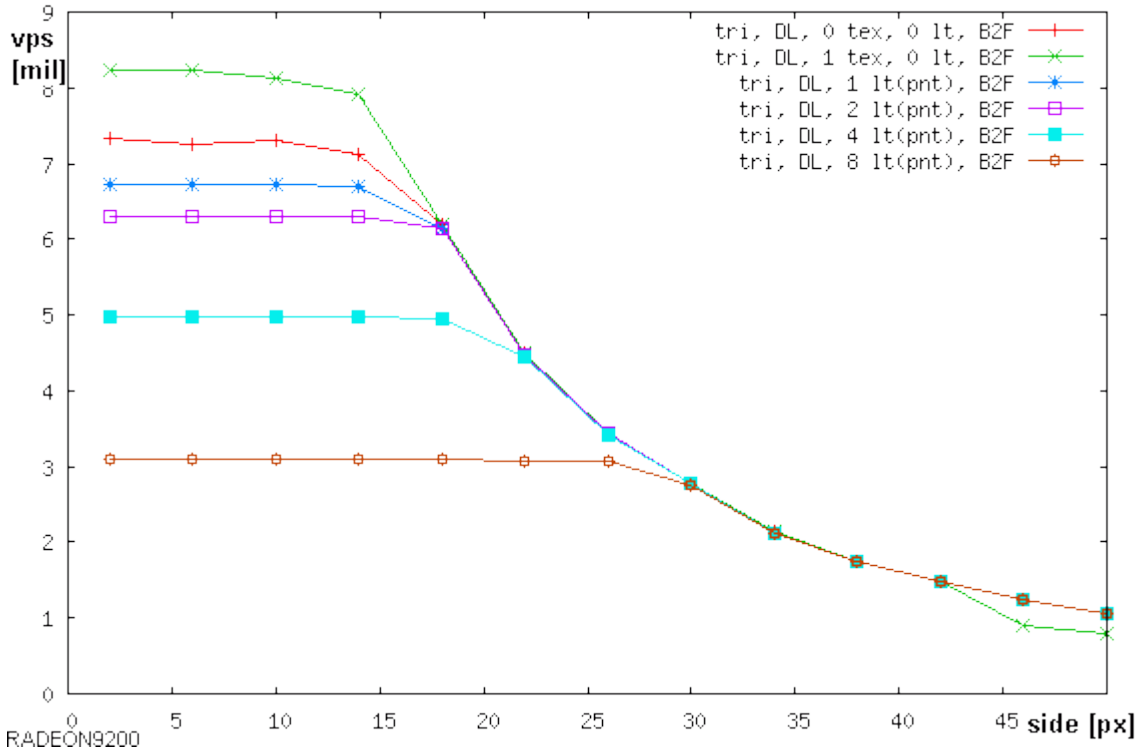
Test display listu a Z-bufferu (bez textur)



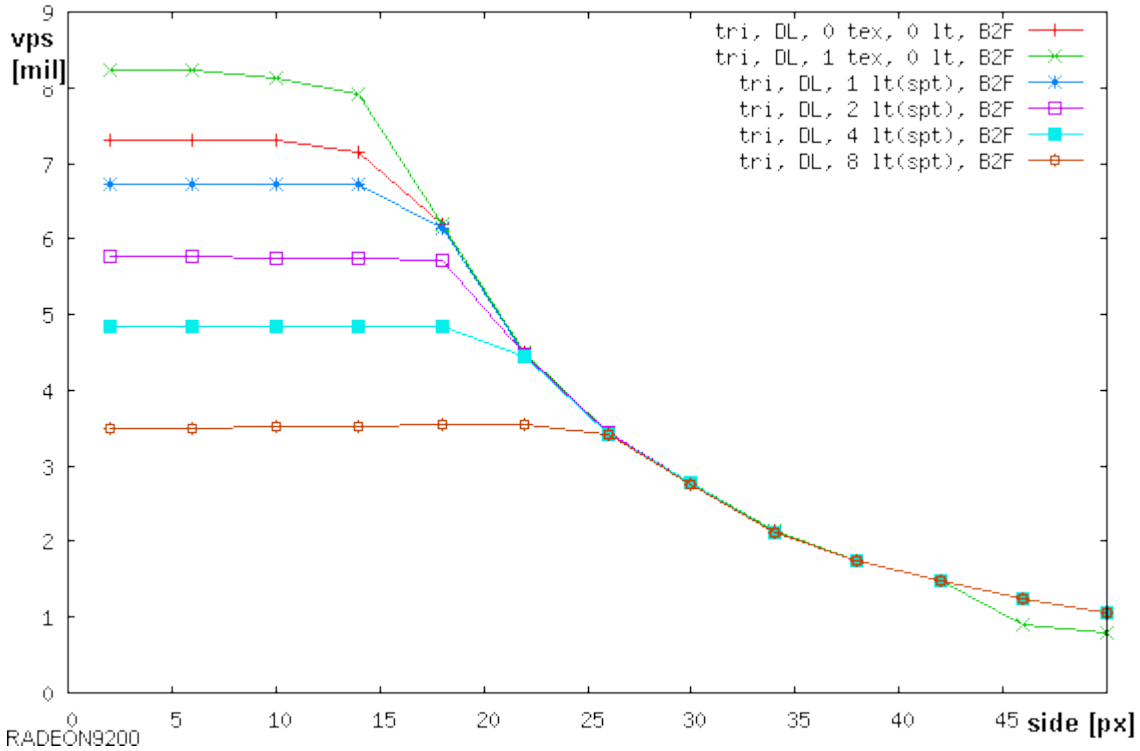
Test směrových světel (bez textur)



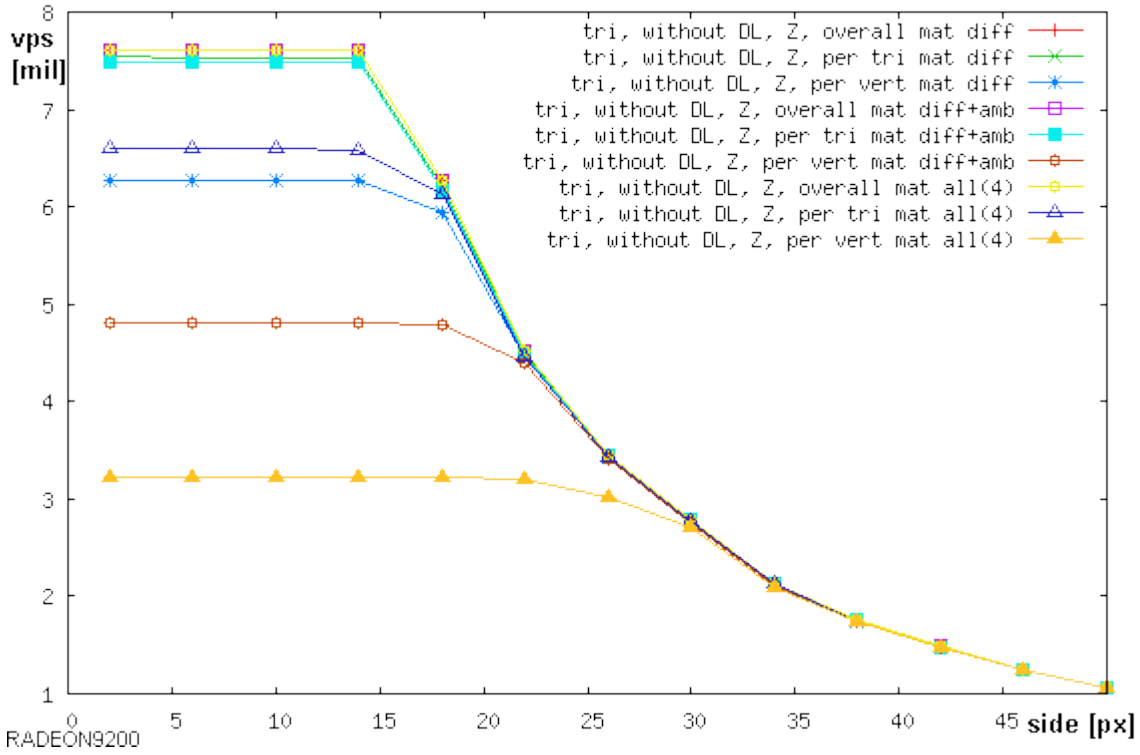
Test bodových světel (bez textur)



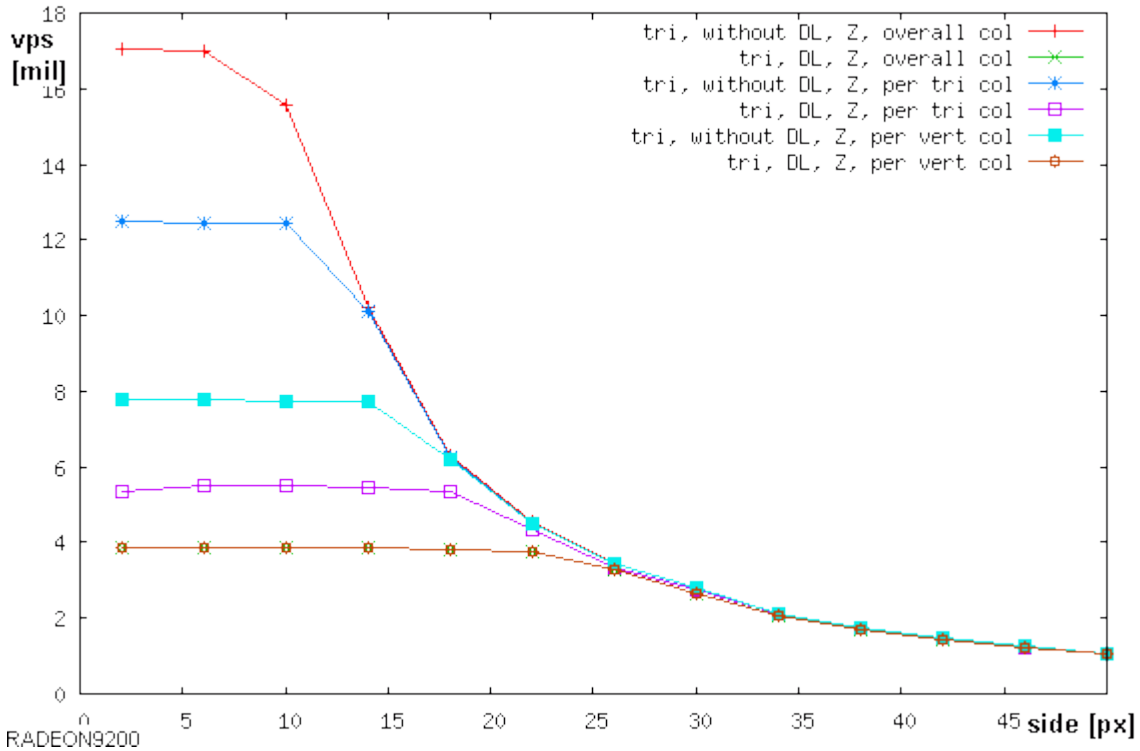
Test reflektorových světel (bez textur)



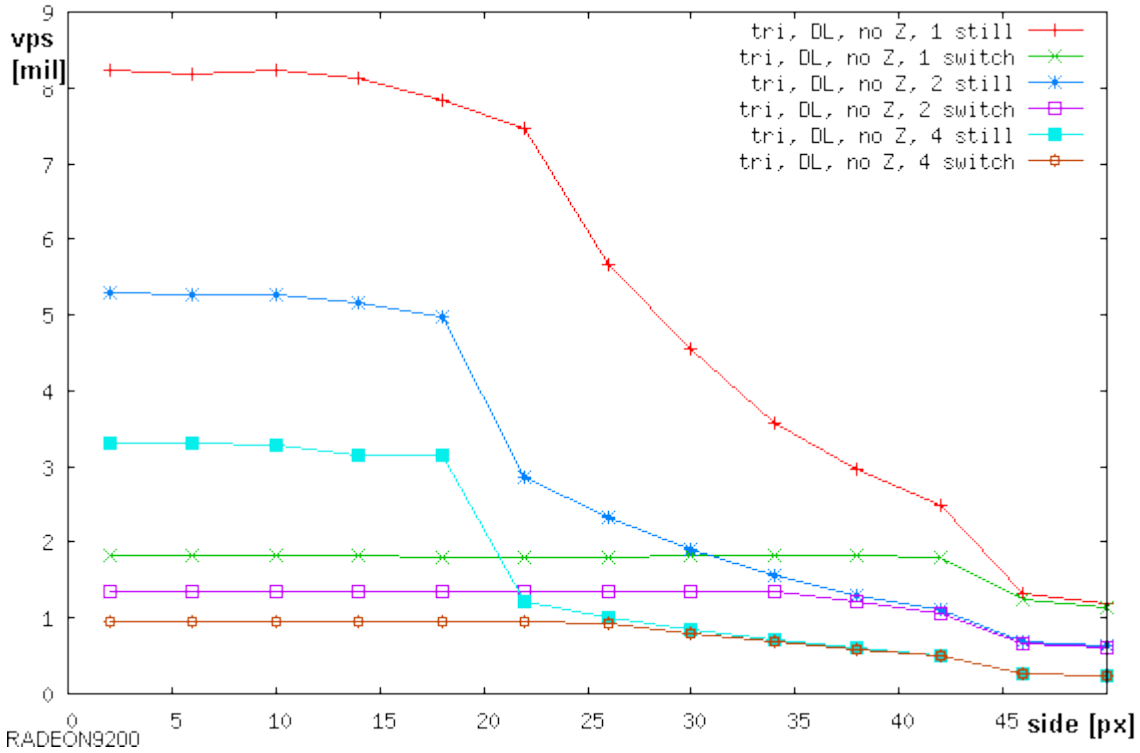
Test přepínání materiálů (bez textur)



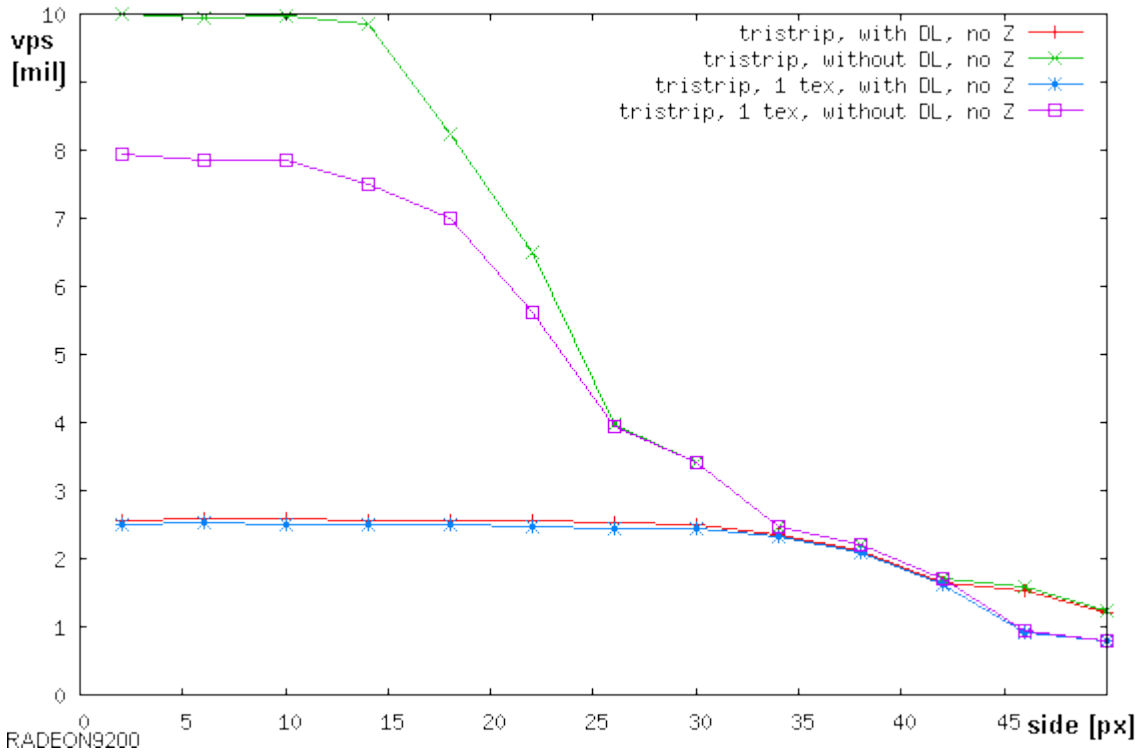
Test přepínání barev (bez textur)



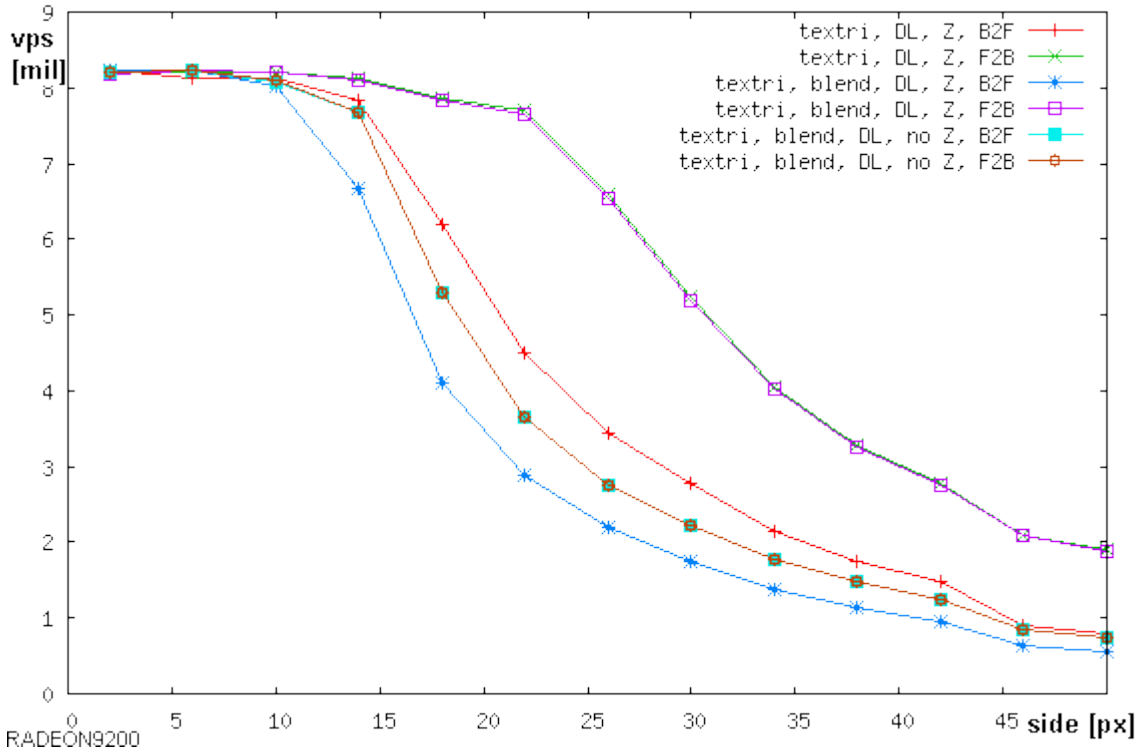
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



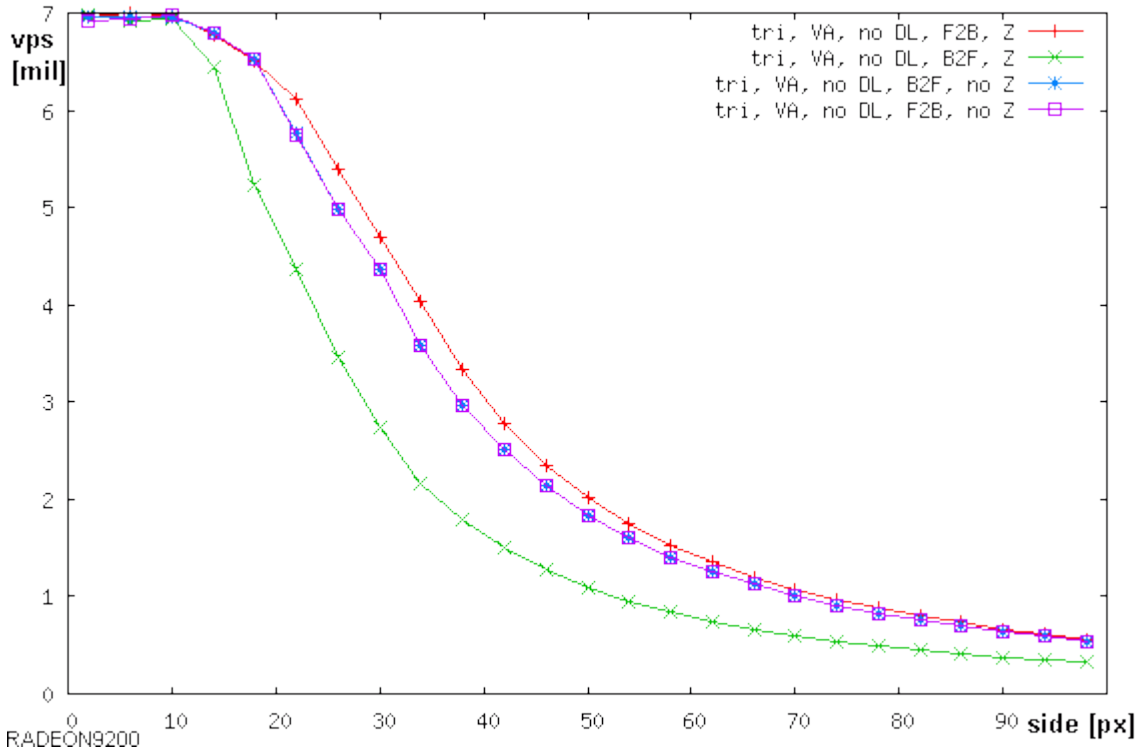
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)

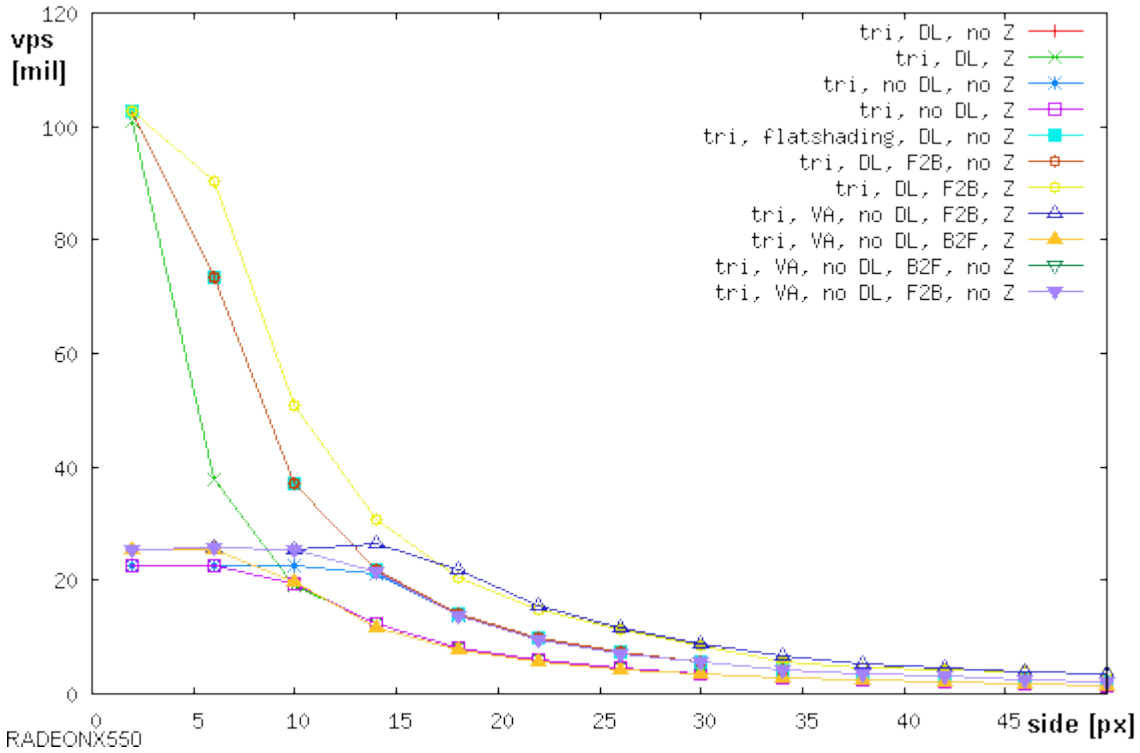


Test vertex arrays, vertex buffer objects

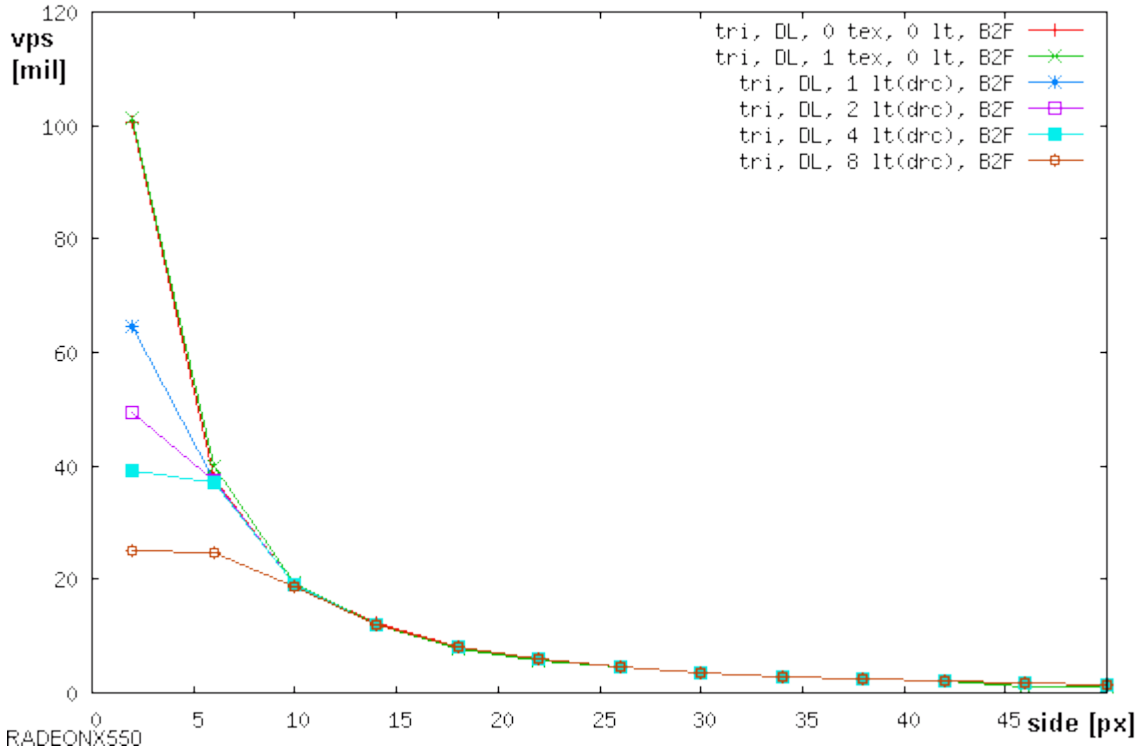


10.2.10 ATI Radeon X550 x86/SSE2

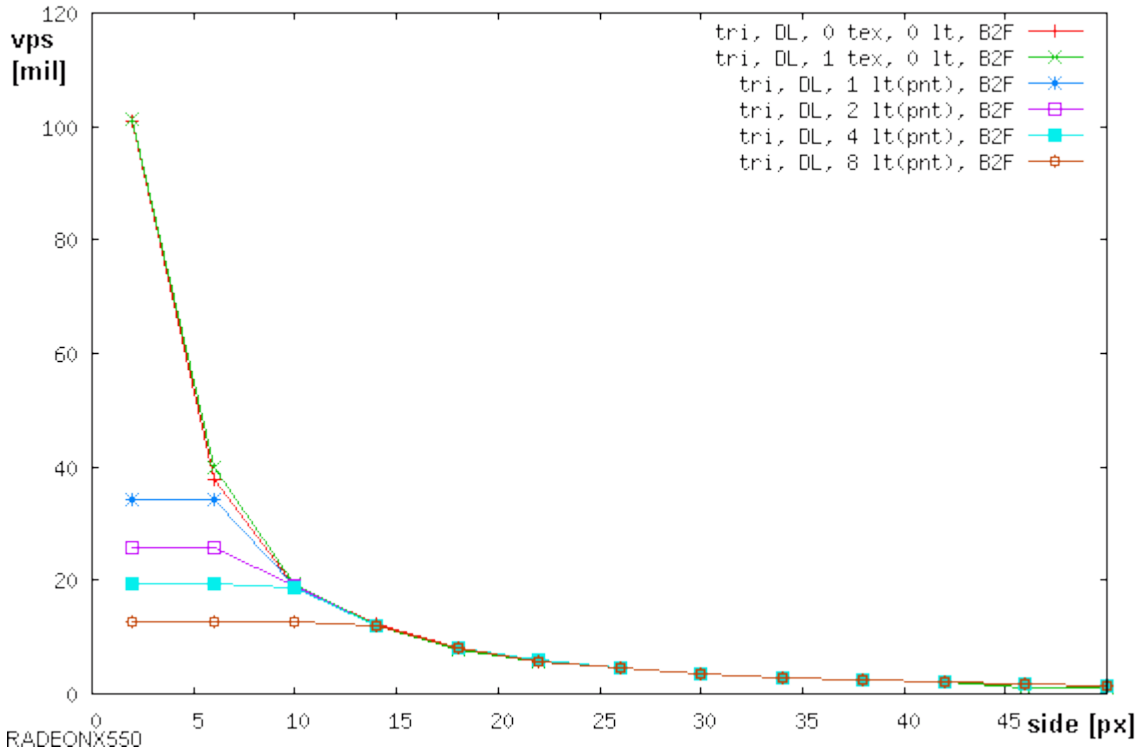
Test display listu a Z-bufferu (bez textur)



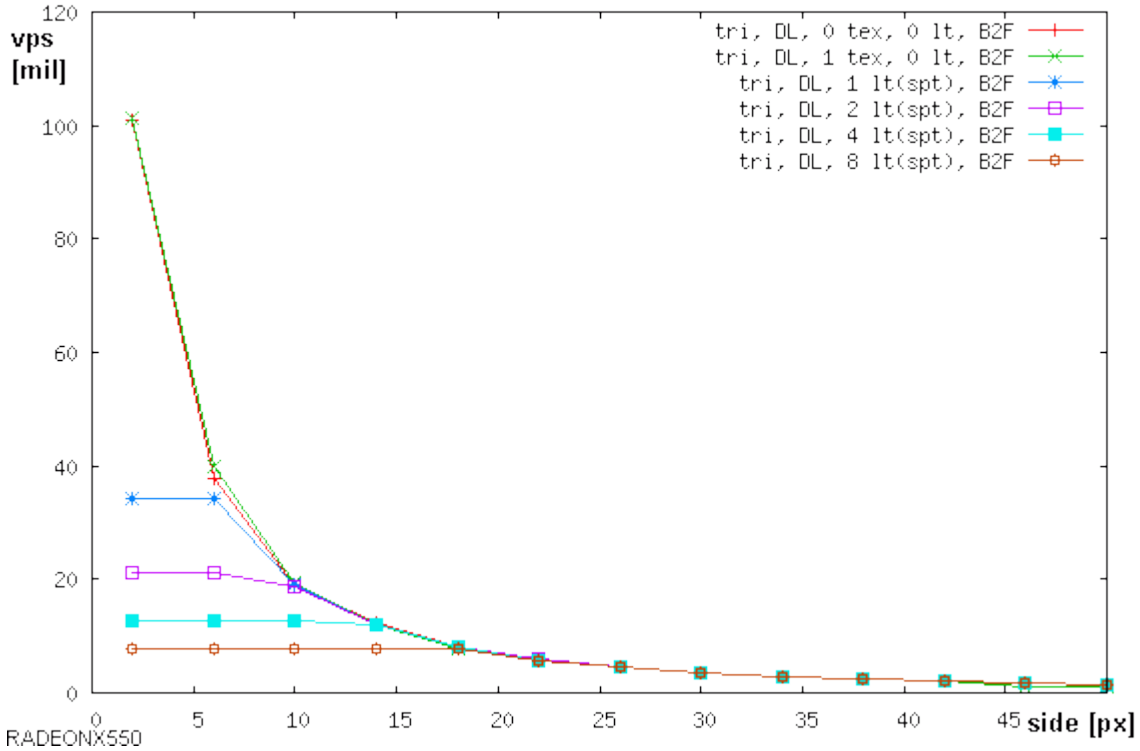
Test směrových světel (bez textur)



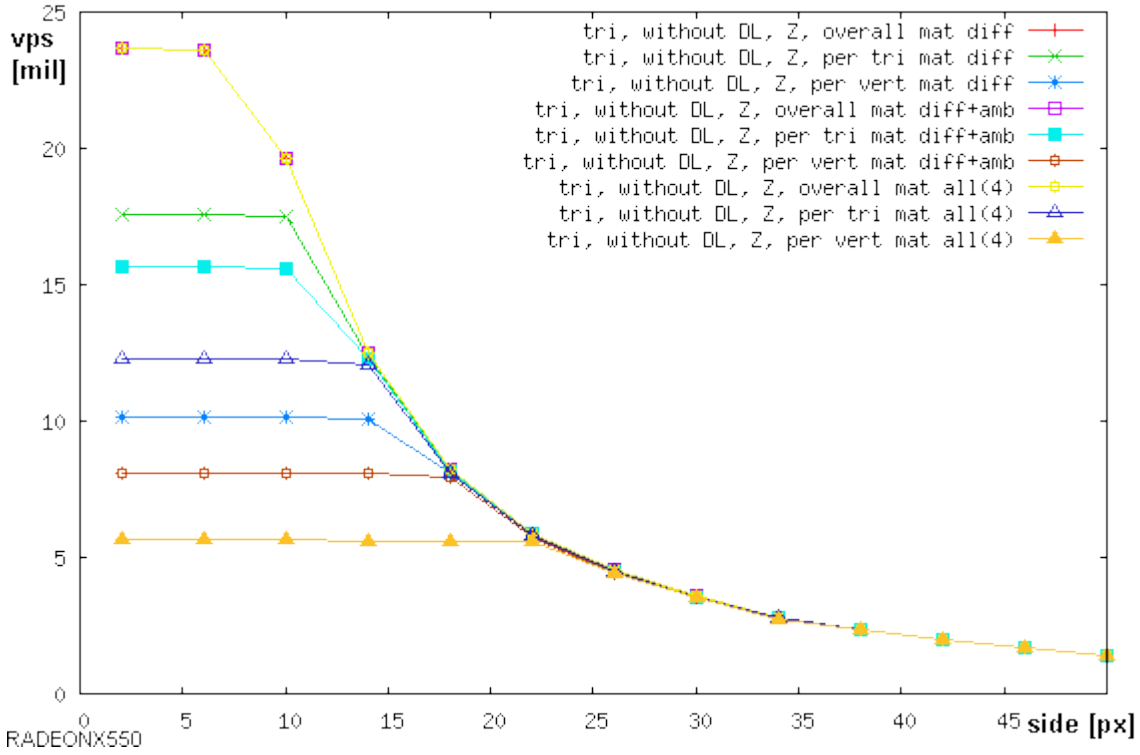
Test bodových světel (bez textur)



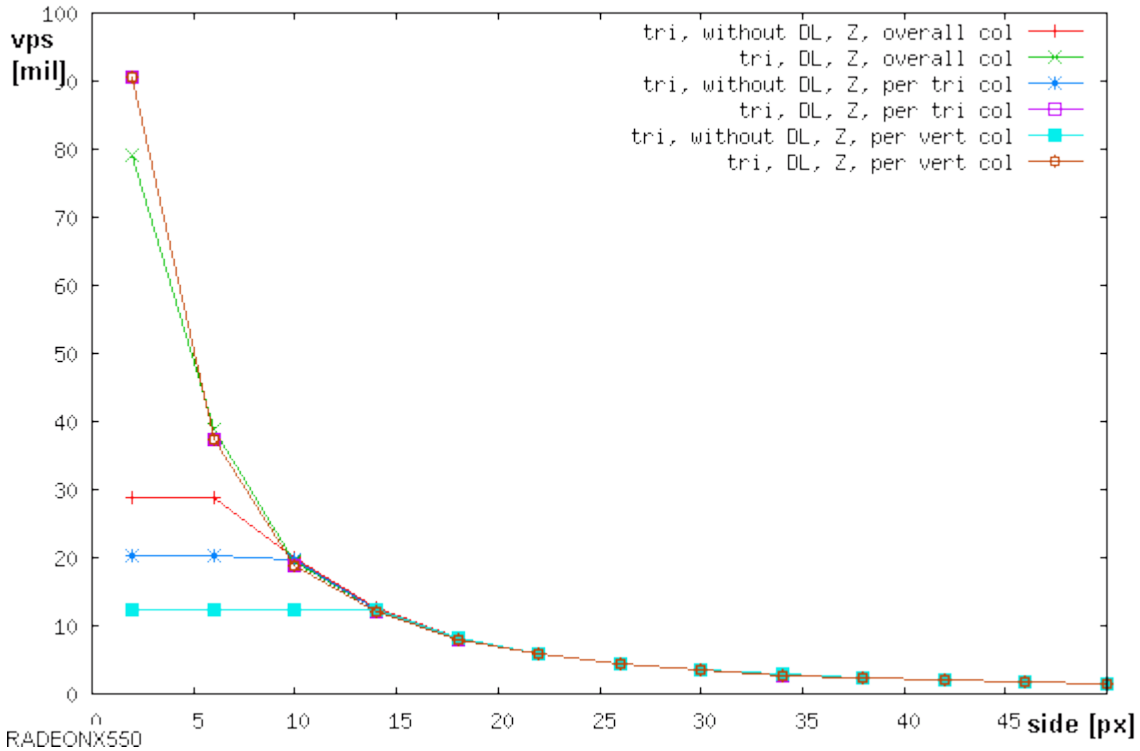
Test reflektorových světel (bez textur)



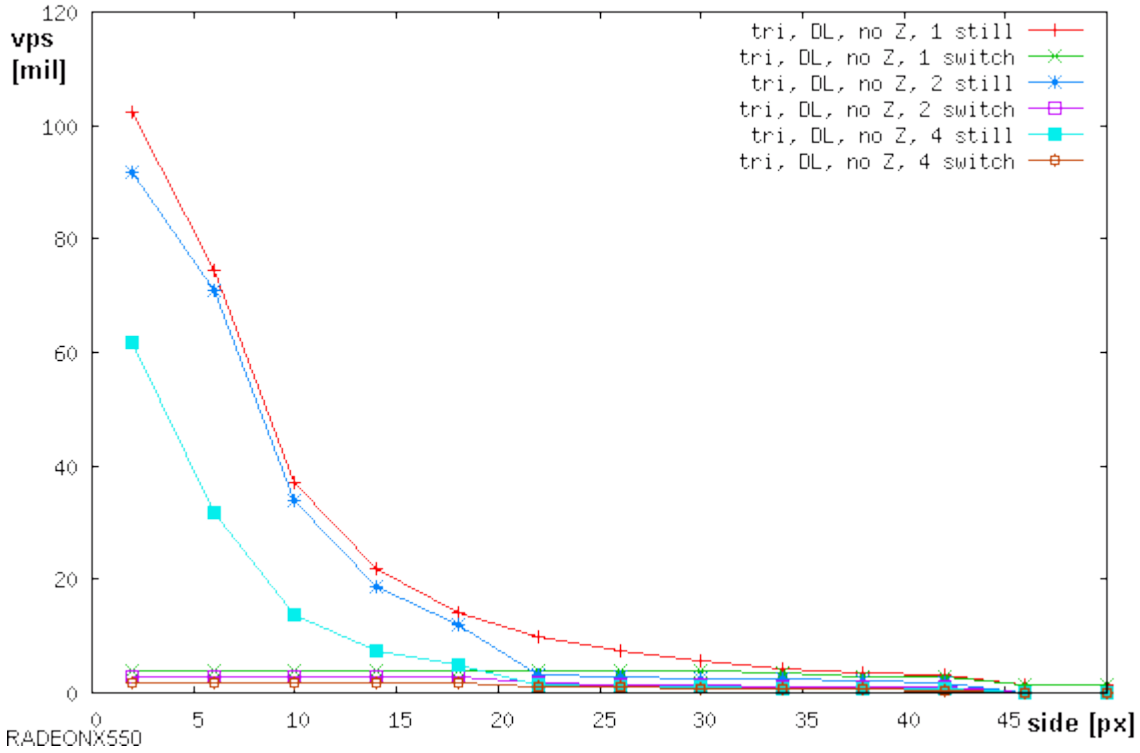
Test přepínání materiálů (bez textur)



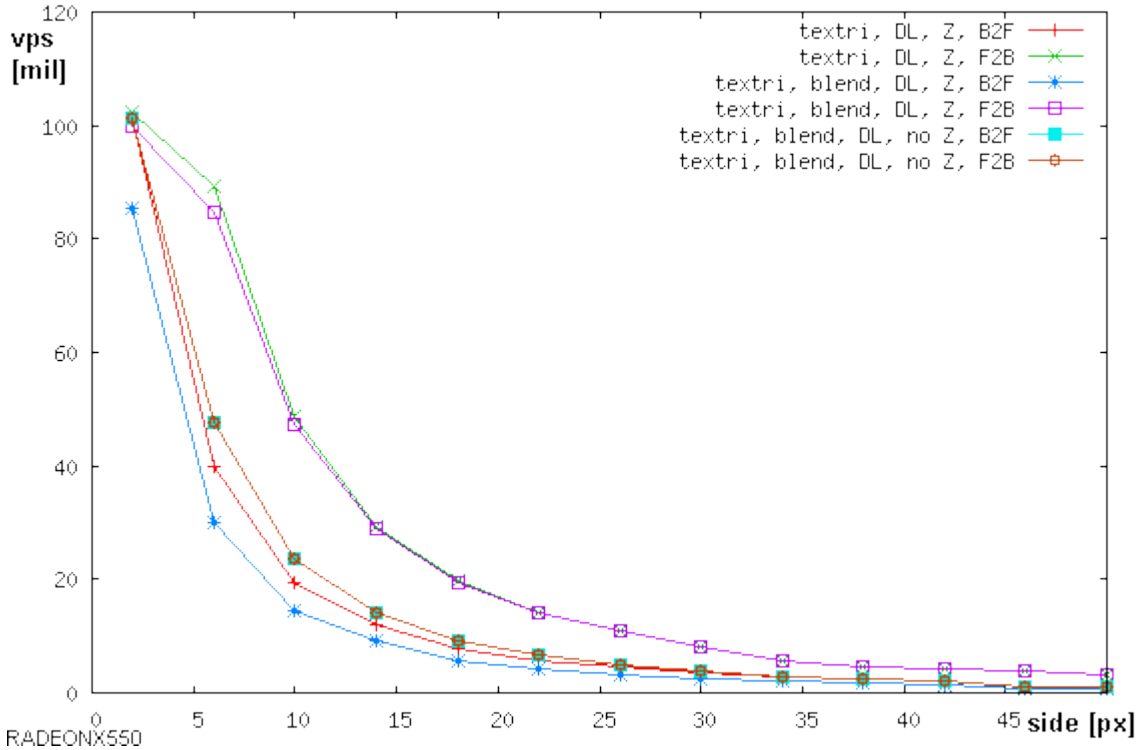
Test přepínání barev (bez textur)



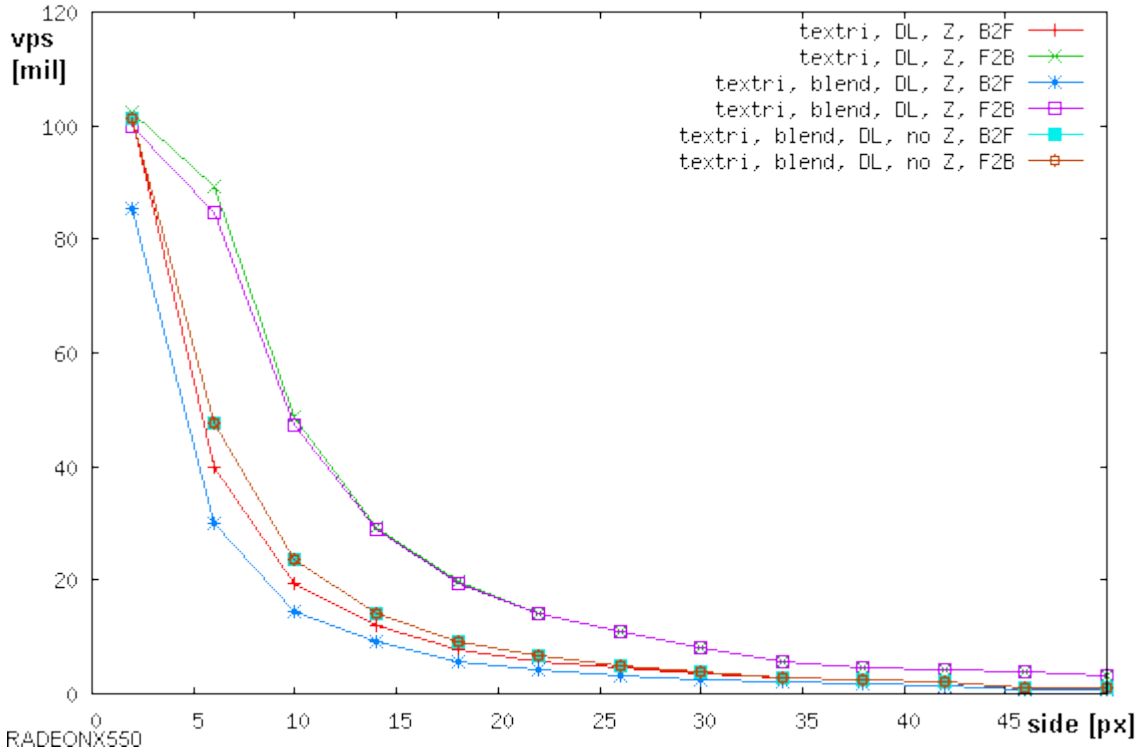
Test přepínání textur (1 textura, 2 multi-textury, 4 multi-textury)



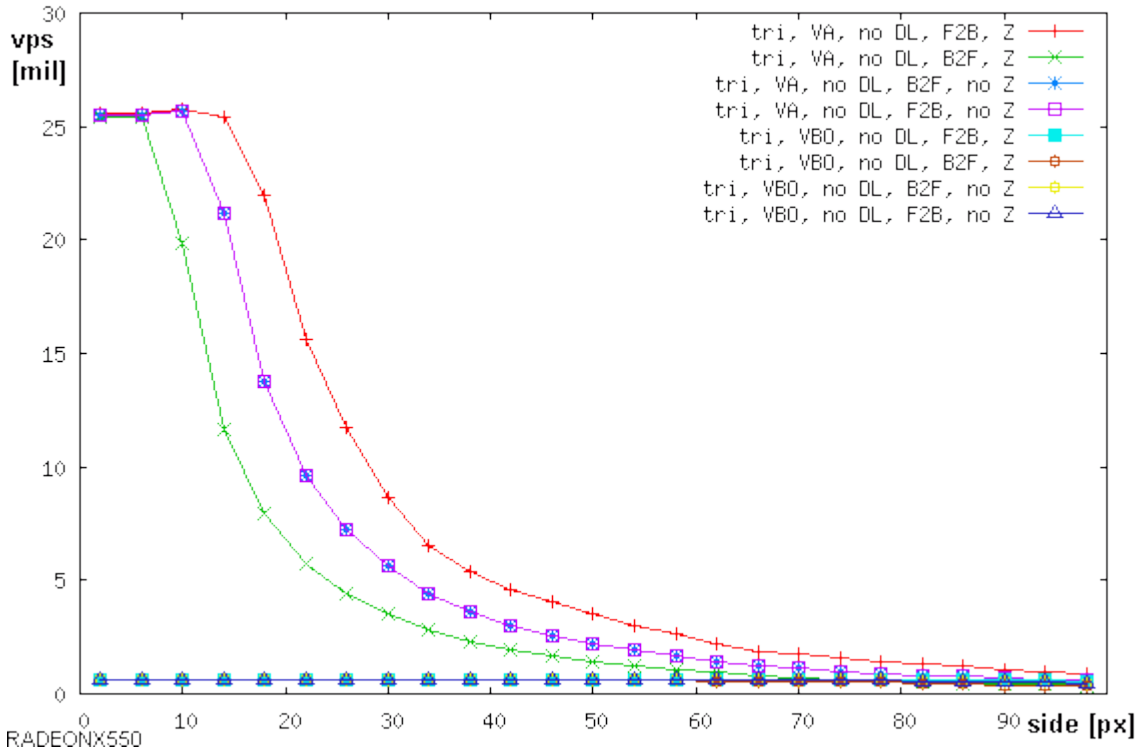
Test pásů trojúhelníků (bez textury, s jednou texturou)



Test typické scény (display list, 1 textura)



Test vertex arrays, vertex buffer objects



10.2.11 Komentář k výkonnostním charakteristikám

Přepínání materiálů a textur

Přepínání materiálů a textur lze směle zařadit mezi nejnáročnější testy. Podle dramatického poklesu výkonnosti je lepší se v praktických aplikacích této problematice vyhnout a zvolit jiný postup popř. velmi pečlivě provádět vykreslování scény.

Kreslení zepředu dozadu

Výsledky testů dokazují, že se vyplatí ve scéně provádět seřazení objektů podle vzdálenosti od pozorovatele a podle tohoto pořadí vykreslovat. Je zřejmé, že výkonost bude závislá na rozmístění a velikosti objektů ve scéně. Naopak jako nevýhodné je kreslení zezadu dopředu.

Vertex arrays, vertex buffer objects

Obecně se uvádí, že vykreslování pomocí vertex arrays a vertex buffer objects by mělo být výkonově výhodnější. Bohužel, ani jeden z provedených testů tento fakt nepotvrdil. Navíc se podle testů jeví obě metody stejně výkonné.

GF2

U GF2 se nevyplatí pro neotexturované trojúhelníky použití displaylistů – výkon s displaylisty je srovnatelný s kreslením klasických trojúhelníků. Akcelerátor paradoxně vykazuje vyšší výkon pro otexturované trojúhelníky (viz test světel).

GF3

V porovnání s GF2 nedosahuje takových maximálních hodnot (první hodnota je zpravidla o polovinu nižší), ale na druhou stranu je zlom na charakteristice posunut až třikrát dále. Obdobně jako GF2 vykazuje vyšší výkon pro otexturované trojúhelníky. Za povšimnutí stojí téměř stejná výkonnost pro jedno a dvě bodová nebo reflektorová světla. U směrových světel může být dokonce povolena až čtveřice bez viditelné ztráty výkonu.

GF4TI4200

Výkonnost kreslení displaylistu, vertex arrays a klasických trojúhelníku je překvapivě vyrovnaná. Vyšší nárůst výkonu je pozorovatelný až při kreslení pásů trojúhelníků pomocí displaylistu.

Některé charakteristiky druhé karty nejsou stabilní a v porovnání s první je výkon nižší (displaylist a Z-buffer, bodová světla, přepínání materiálů a barev). Rozdíl je příliš velký než by se dalo odůvodnit drobnou nepřesností měření. Připadají v úvahu tyto příčiny:

1) byl ovlivněn program v průběhu měření
2) jiný ovladač grafického akcelerátoru
3) různý hardware

Chybu způsobenou různým hardwarem lze vyloučit, test běžel na stejné konfiguraci. Druhou možnost vyloučit nelze, verze ovladačů byla skutečně jiná (1.5.2 oproti 1.4.2). Tím je možné vysvětlit i nižší výkon. Zůstává vysvětlit nestabilitu hodnot. Ta musela být zapříčiněna ovlivněním programu během měření (např. spuštěna náročná operace na pozadí). Dostatečným důkazem by měl být fakt, že je ovlivněna pouze polovina testů podle pořadí jak byly prováděny.

GF6600, GF6800, GFFX5900XT

Patří k nejvýkonnějším akcelerátorům v testu. Akcelerátory těží z velmi velkého nástupního výkonu (v některých testech více jak 120 milionů vertexů za sekundu), který pak i pro větší stranu objektu neklesne pod výkonnost žádné z konkurentů.

Radeon9200

Akcelerátor nezvládá práci s displaylistem. Výkonnost s displaylistem je výrazně nižší (až podezřele) než klasické kreslení. Podobně jako GF2 vykazuje vyšší výkon pro otexturované trojúhelníky. Ve scéně může být povolena čtveřice směrových světel bez ztráty výkonu. Akcelerátor si nevede špatně při testu přepínání materiálů.

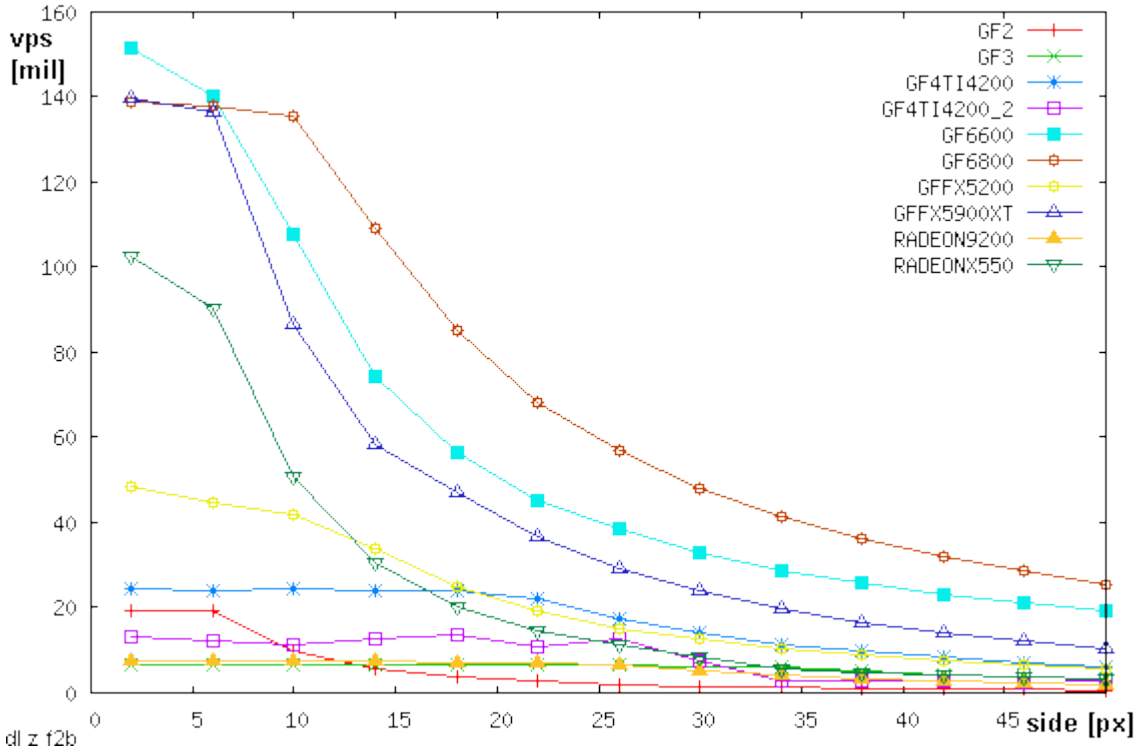
Z blíže neobjasněných důvodů chybí měření vertex buffer objects.

RadeonX550

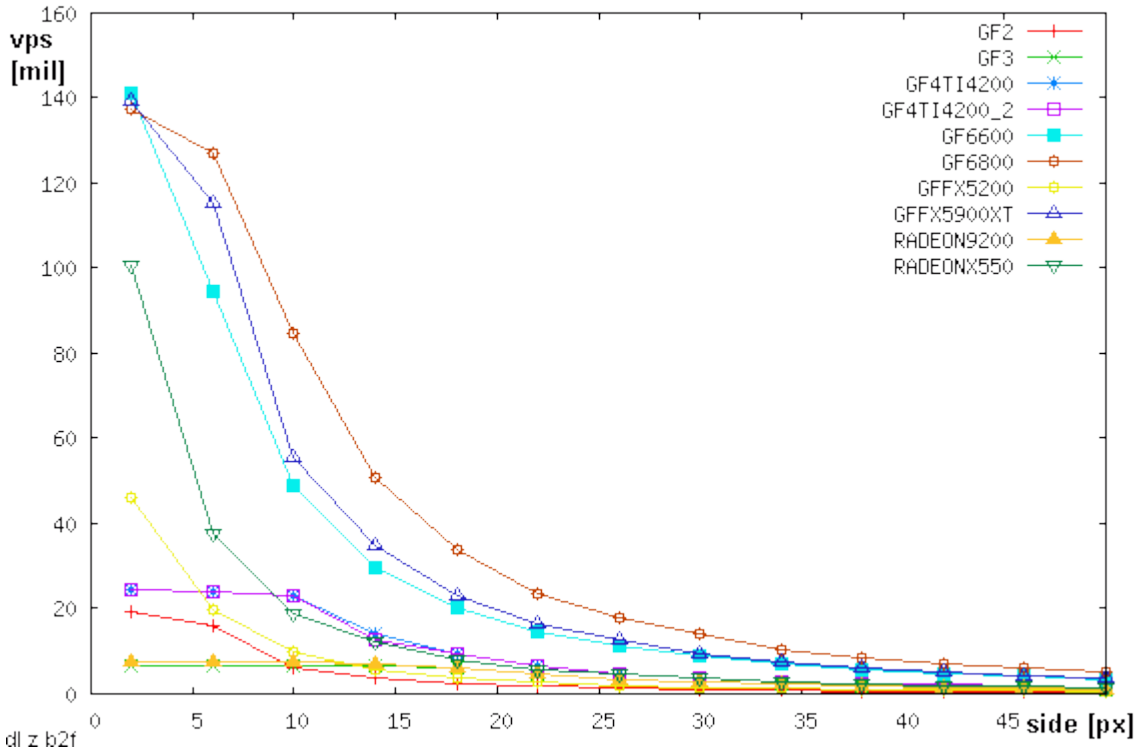
Akcelerátor překvapil velmi vysokým výkonem při přepínání materiálů pro menší velikost strany objektu. Překvapením, tentokráte však v negativním slova smyslu, byl test vertex buffer objects. Výkonnost je ustálena na 0.5 miliónech vertexů. Chyba je pravděpodobně způsobena špatným ovladačem.

10.3 Porovnání výkonnostních charakteristik

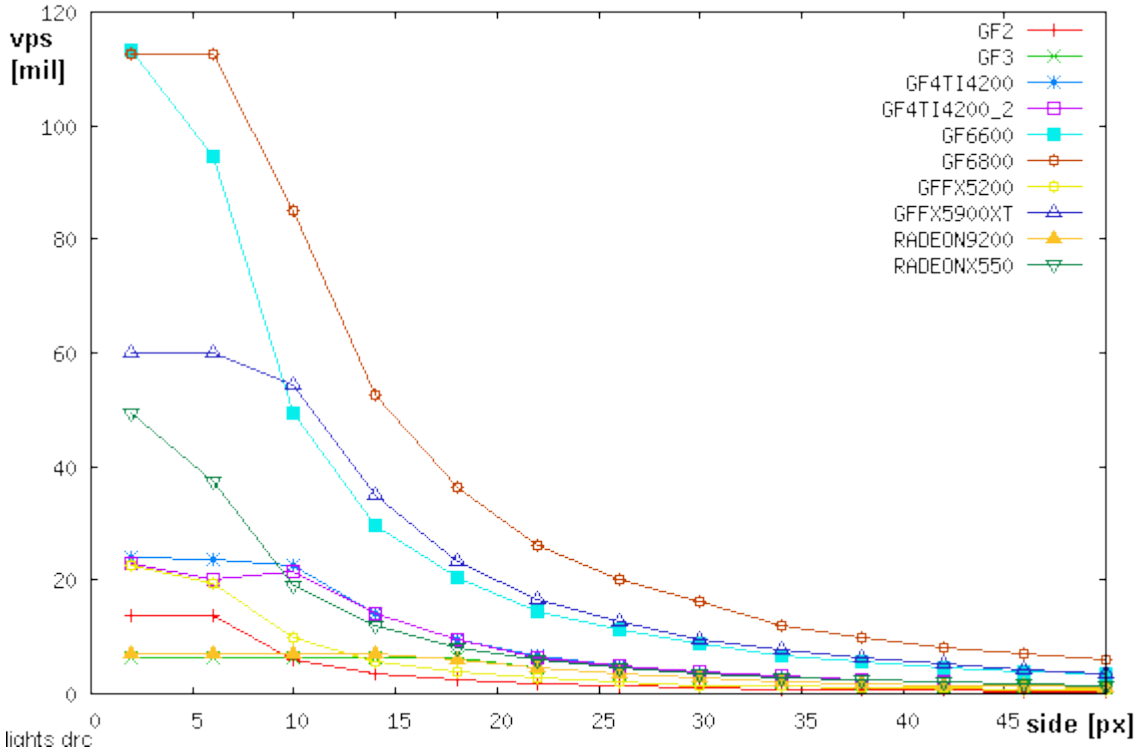
Test display listu a Z-bufferu při kreslení zepředu dozadu



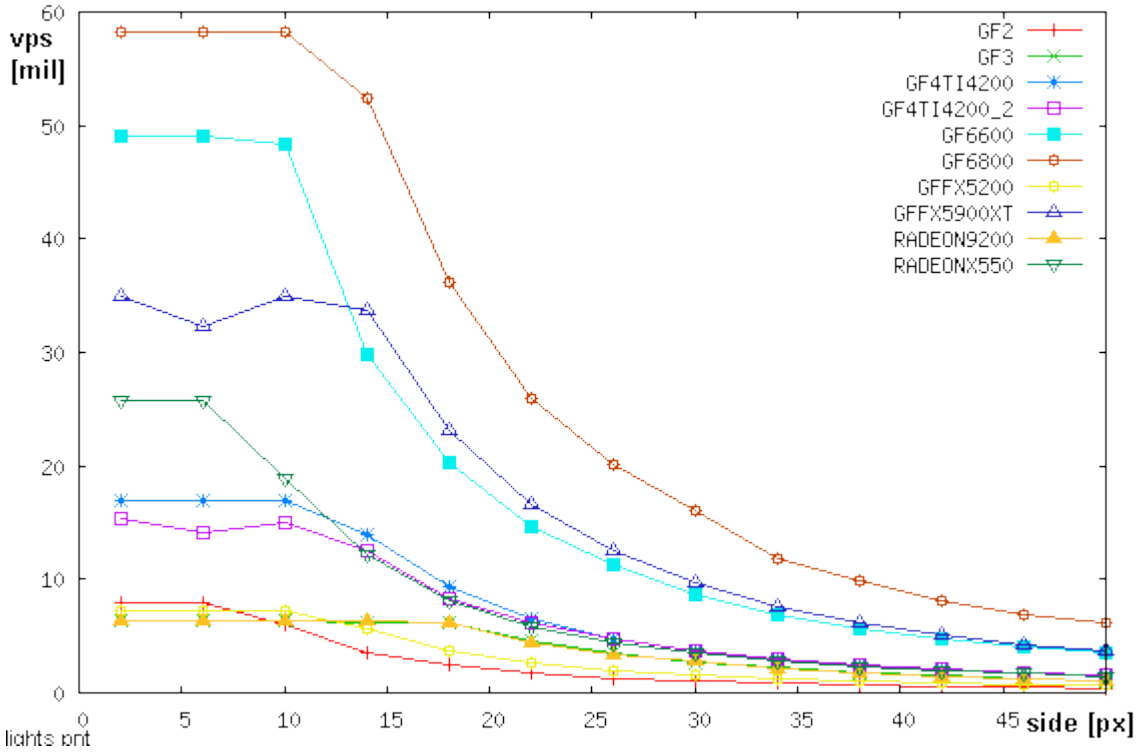
Test display listu a Z-bufferu při kreslení odzadu dopředu



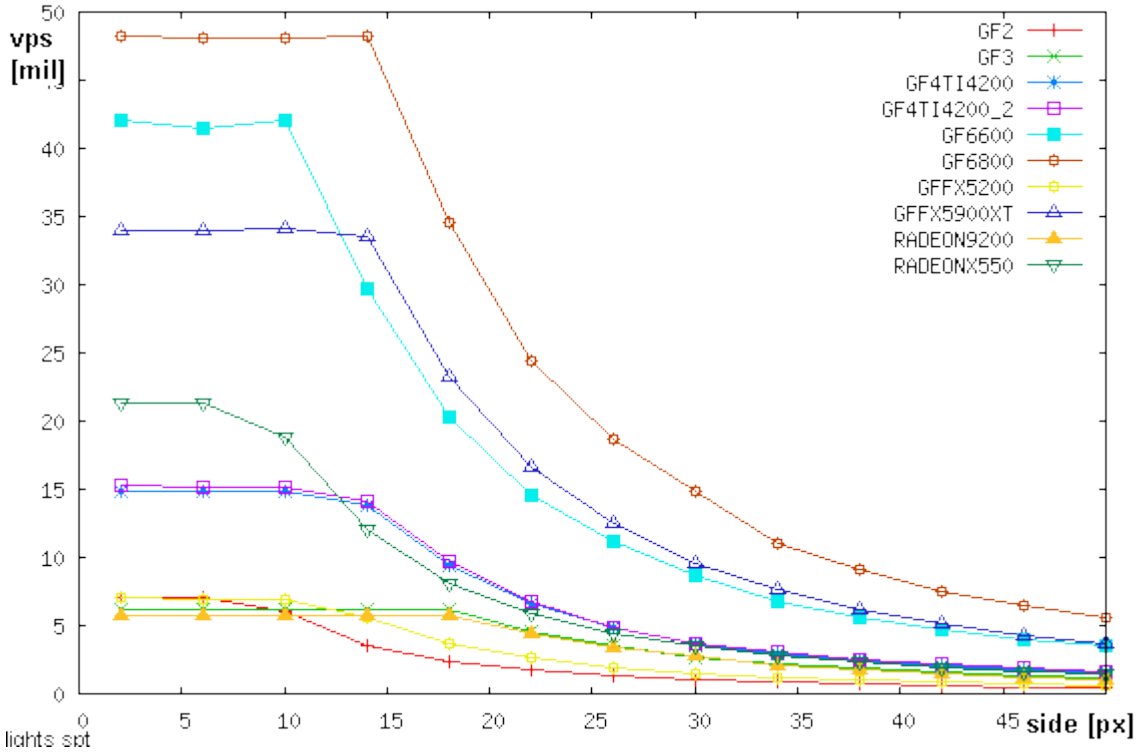
Test směrových světel pro 2 světla



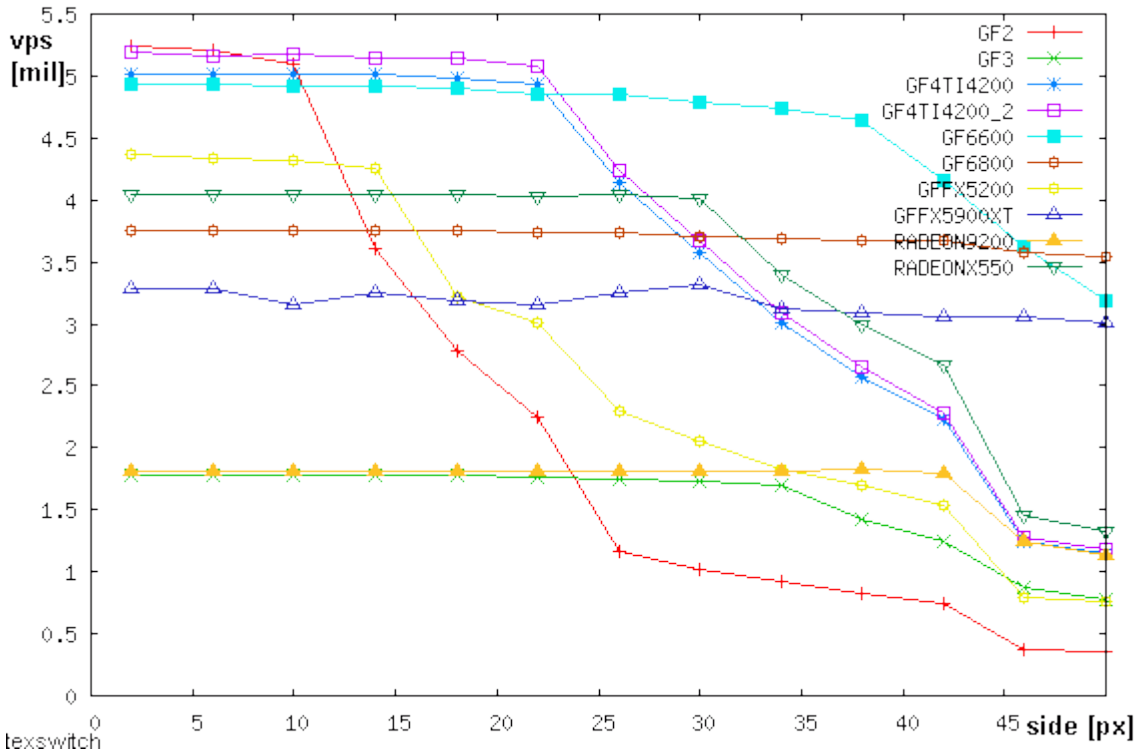
Test bodových světel pro 2 světla



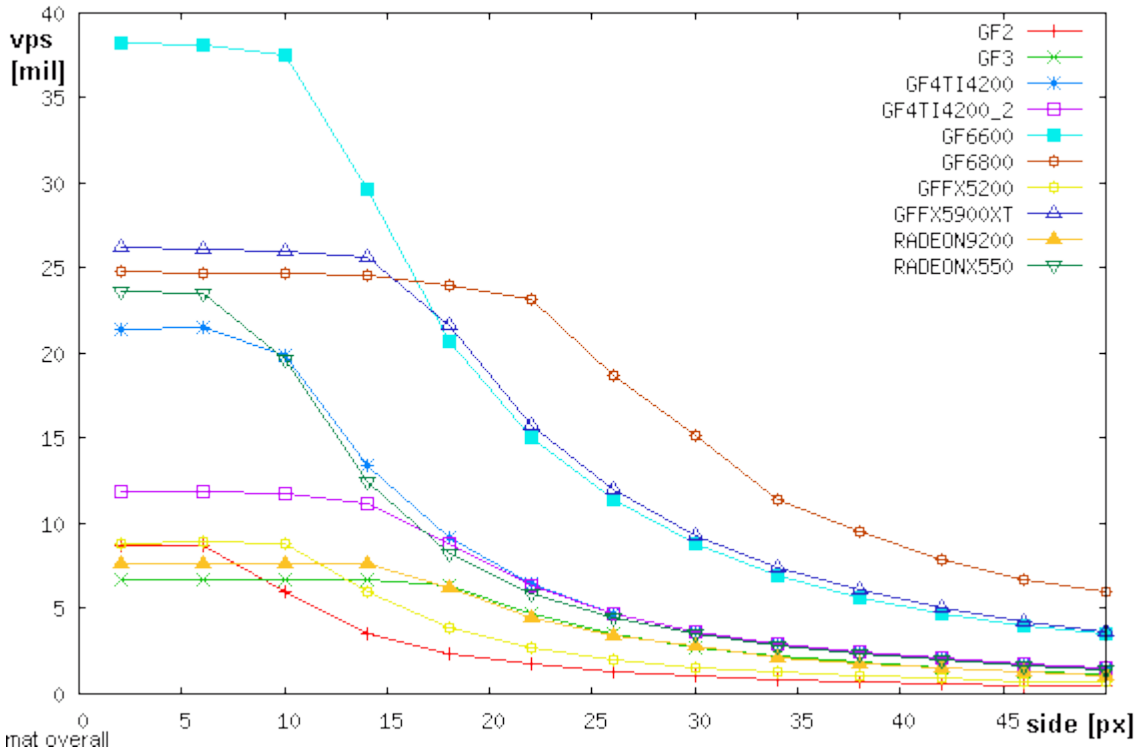
Test reflektorových světel pro 2 světla



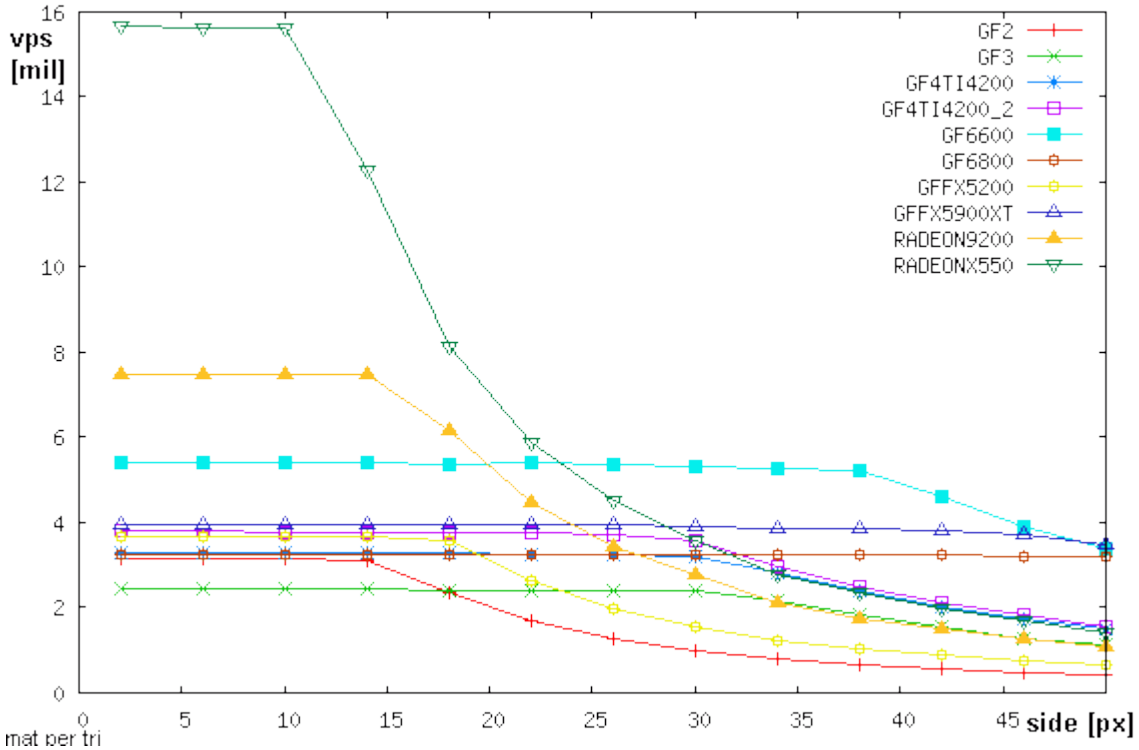
Test přepínání textur pro 1 texturu



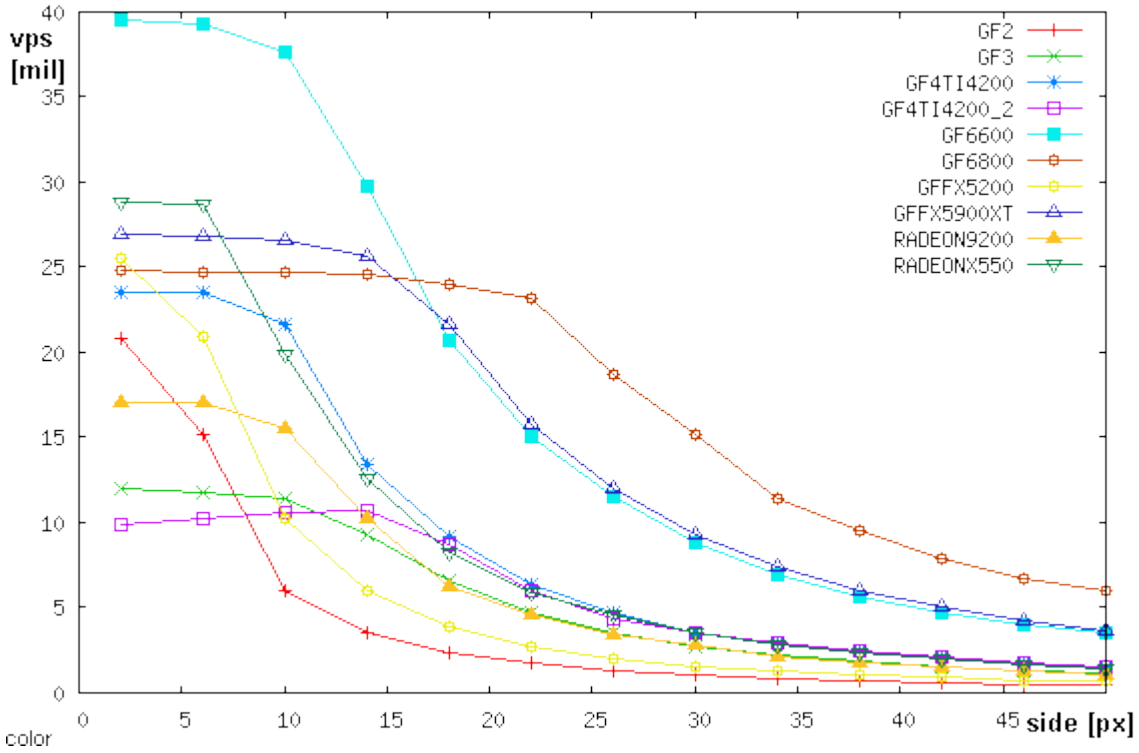
Test přepínání materiálů (celá scéna stejným materiálem)



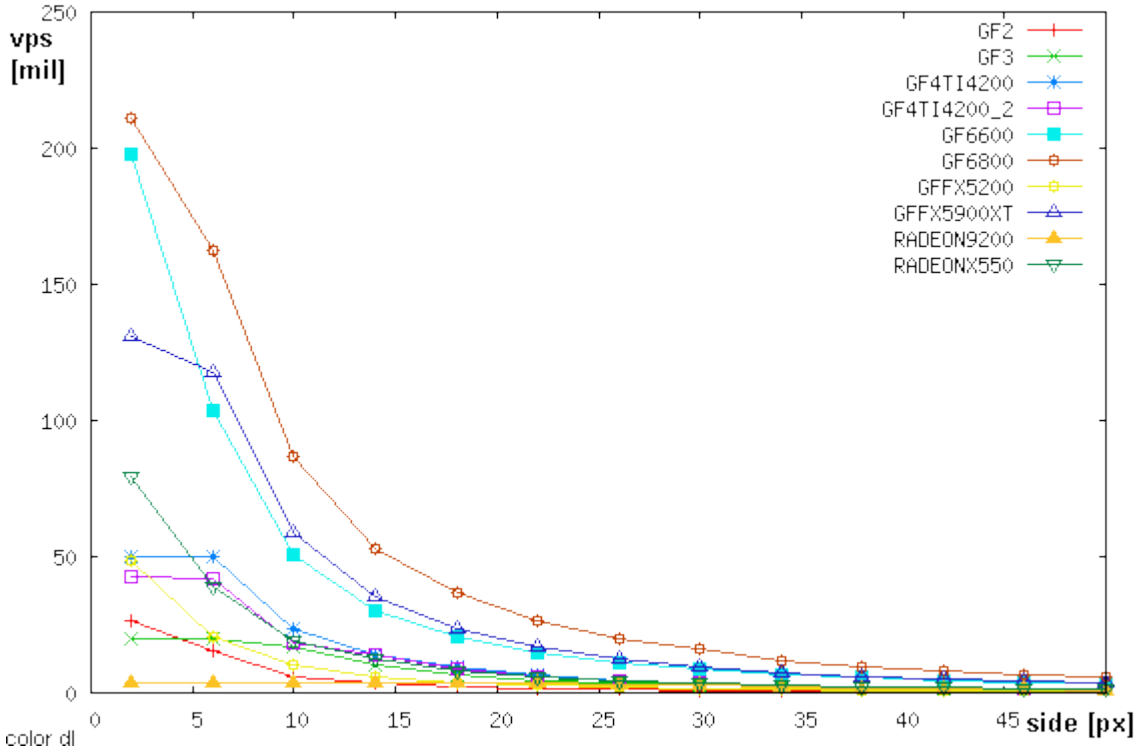
Test přepínání materiálů (každý trojúhelník jiným materiálem)



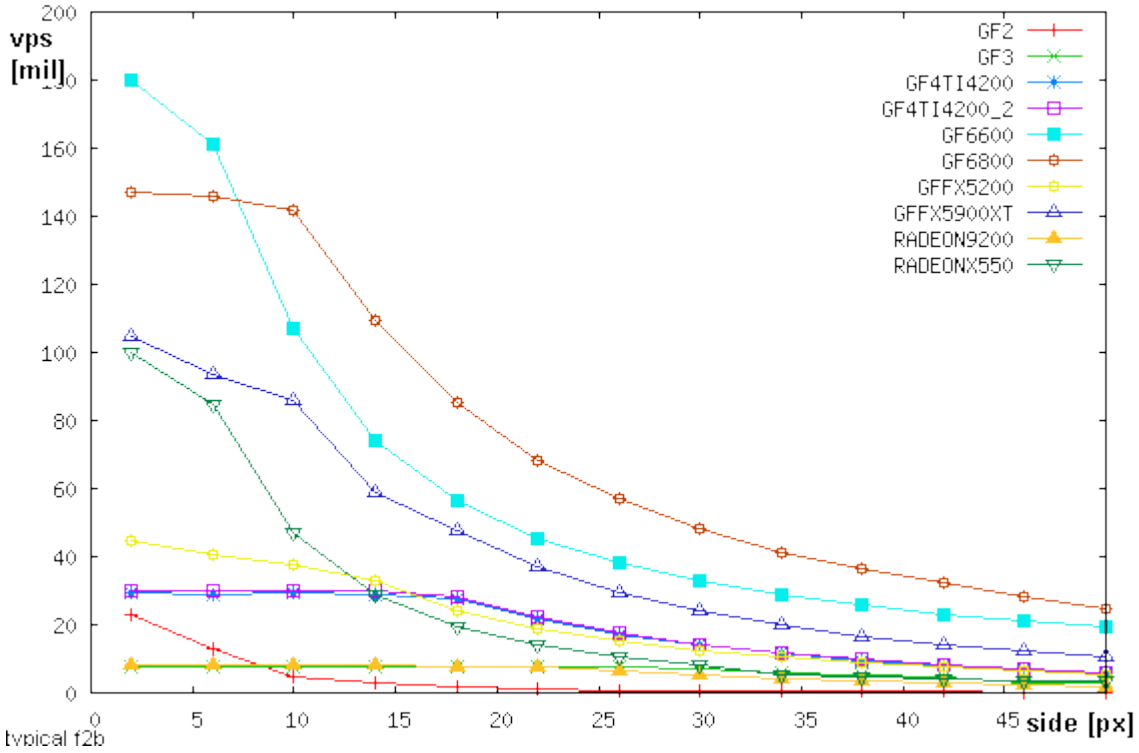
Test přepínání barev bez displaylistu (celá scéna jednou barvou)



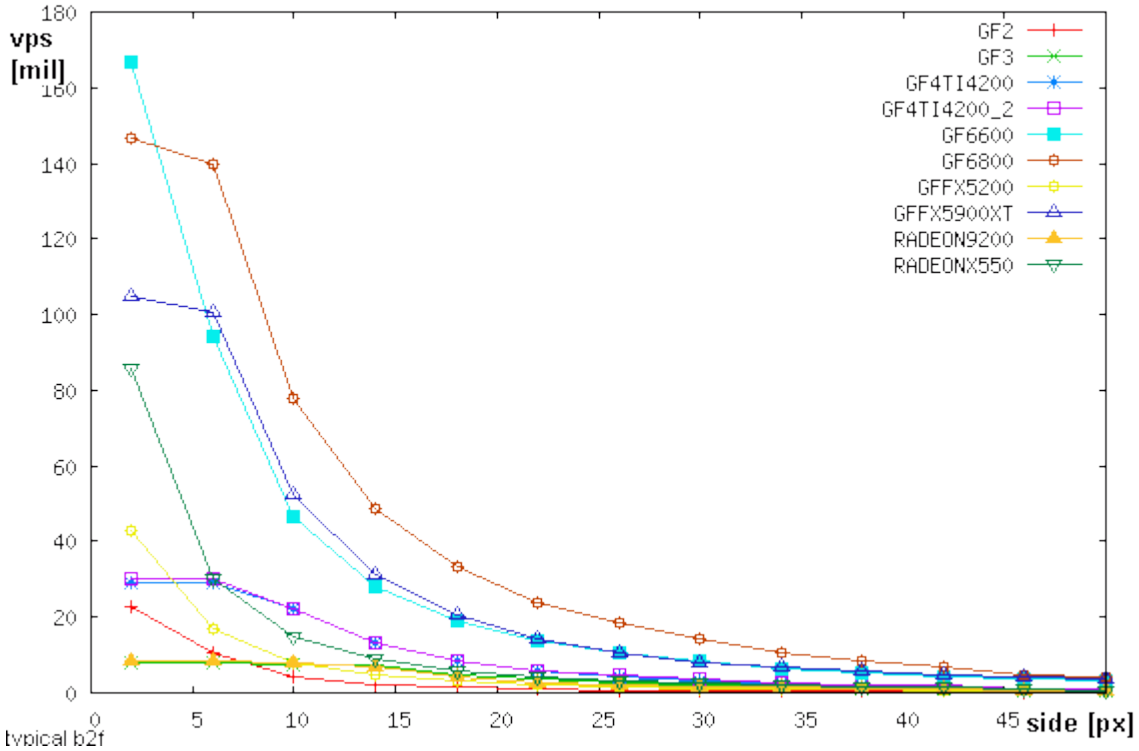
Test přepínání barev s displaylistem (celá scéna jednou barvou)



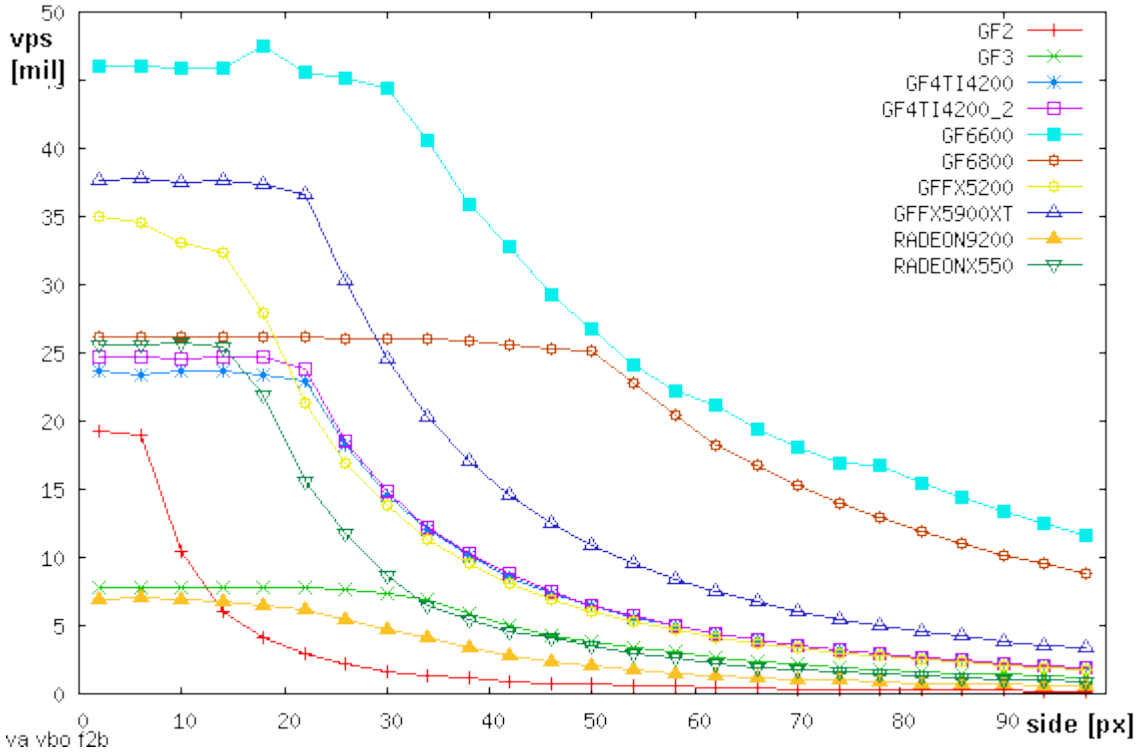
Test typické scény při kreslení zepředu dozadu



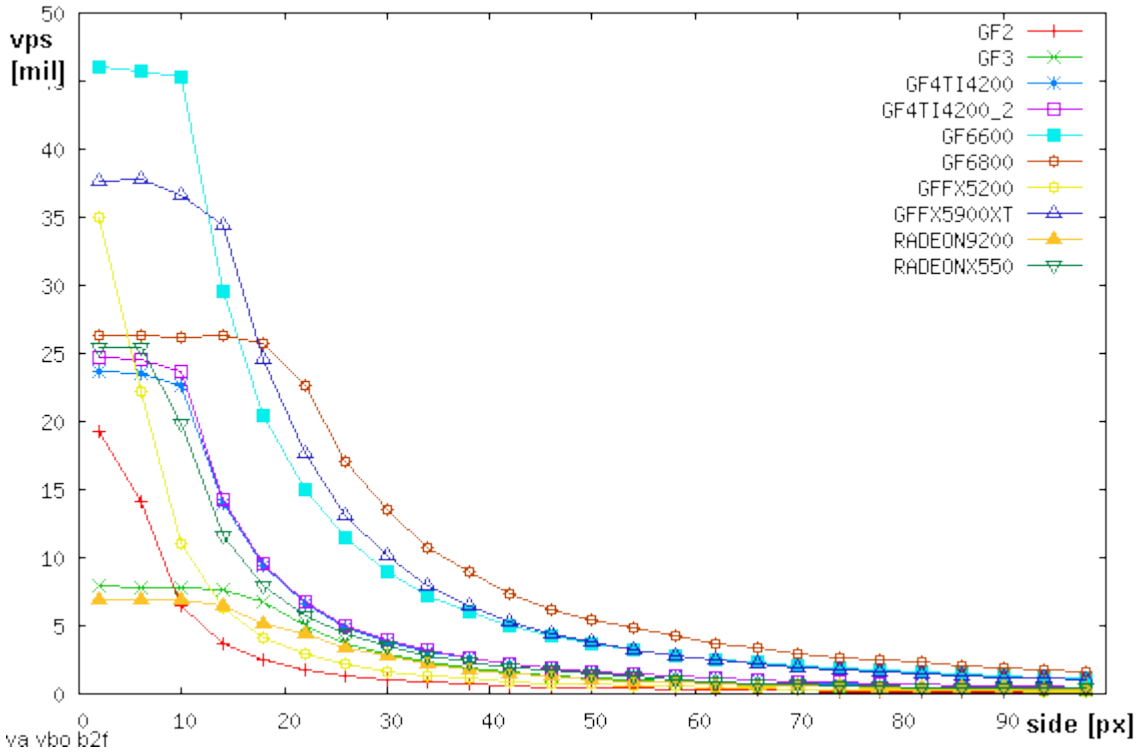
Test typické scény při kreslení odzadu dopředu



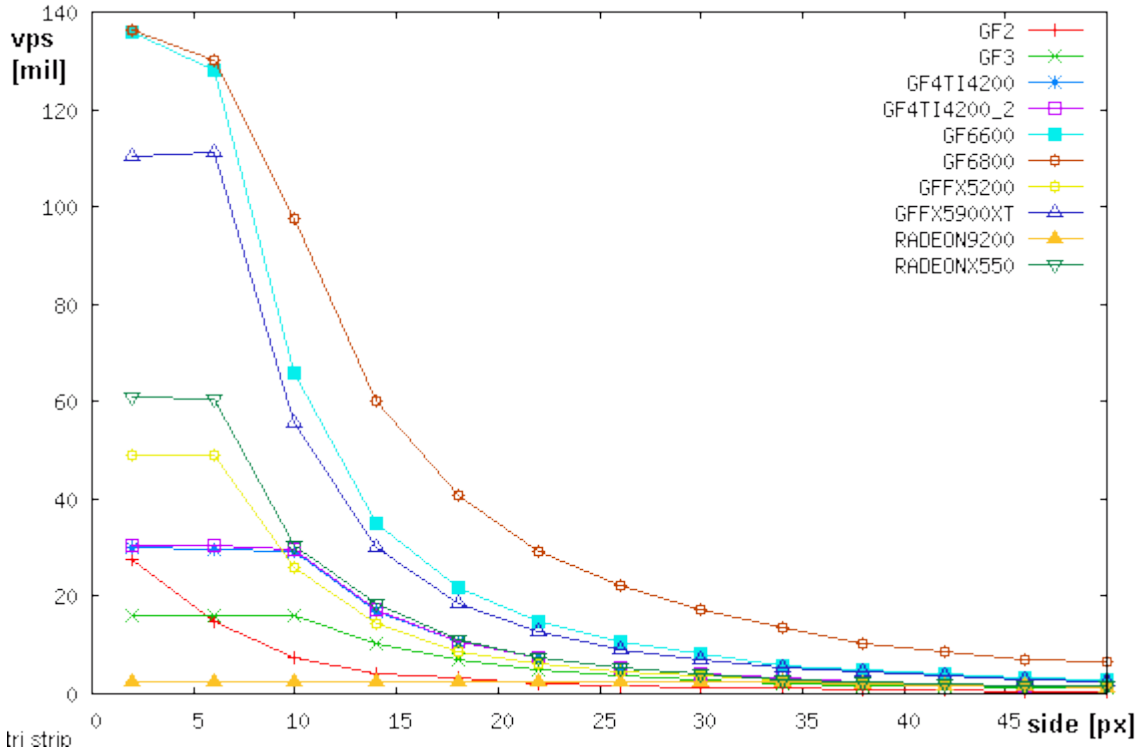
Test vertex arrays při kreslení zepředu dozadu



Test vertex arrays při kreslení odzadu dopředu



Test pásů trojúhelníků (bez textury, s jednou texturou)



Komentář ke srovnávacím charakteristikám

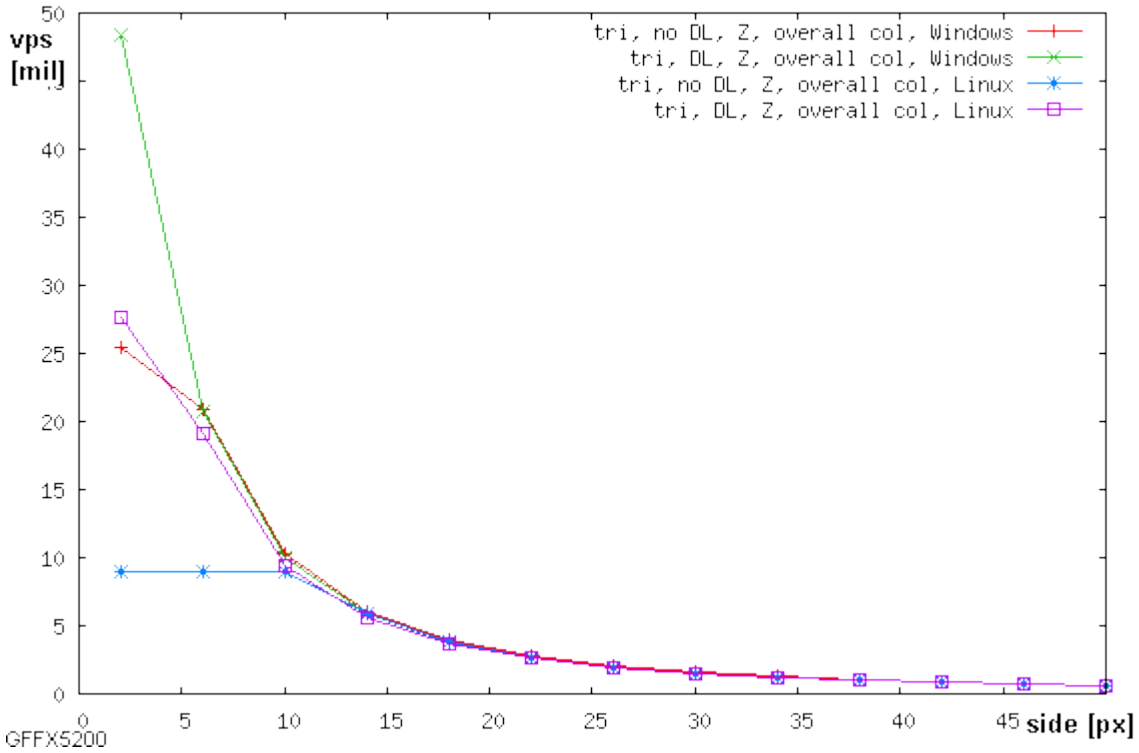
Podle naměřených charakteristik se jako nejvýkonnější jeví GF6800, GF6600, GFFX5200 a GF4TI4200 spolu s RadeonX550 (v tomto pořadí). Za povšimnutí stojí test přepínání textur, kde jsou favorité předstiženi kartou GF4TI4200 u ní však výkonost záhy klesá.

Záhadně vysokým nástupním výkonem překvapil Radeon X550 při testu přepínání materiálů s každým trojúhelníkem. Jeho výkonnost je ve srovnání s ostatními až několikanásobně vyšší. Na druhém místě v témže testu by se umístil Radeon 9200. Těžko říct, jestli se jedná o vlastnost této rodiny akcelerátorů nebo jen ojedinělý jev za určitých podmínek.

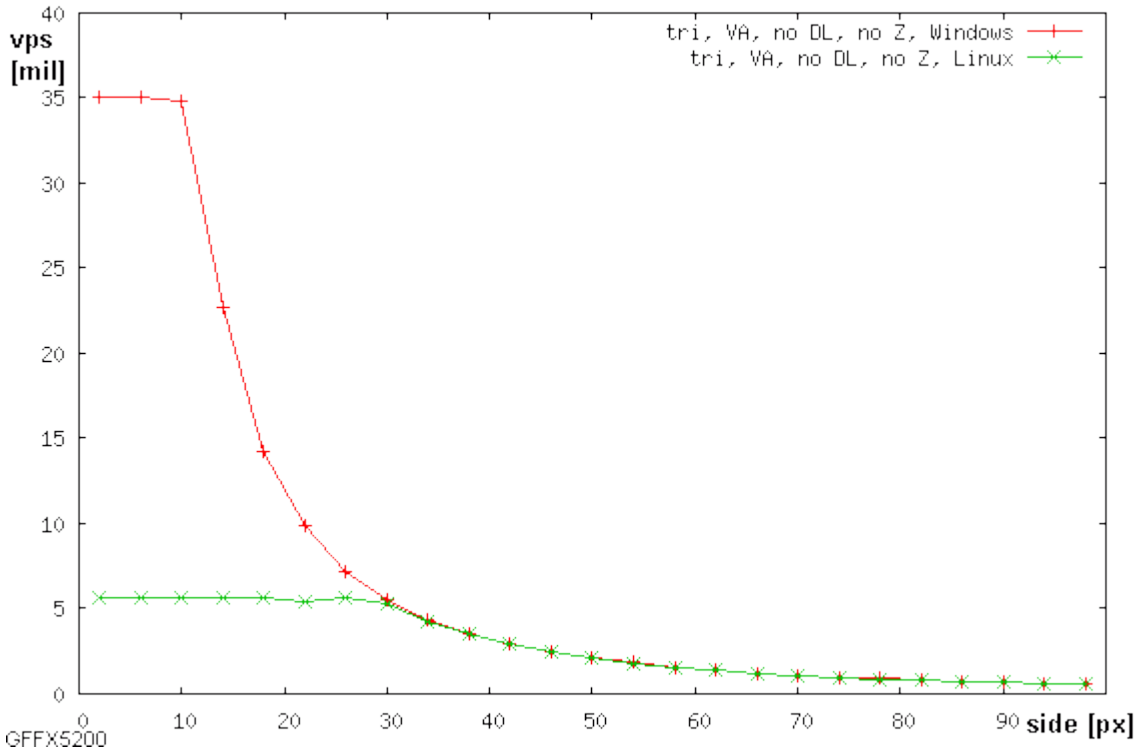
Dvojice karet GF4TI4200 se výrazně rozcházejí při testu displaylistu a Z-bufferu a při přepínání materiálů a barev. Možné příčiny jsou uvedeny na str. 92.

10.4 Výkonnost Linux vs Windows pro GFFX5200

Test přepínání barev



Test vertex arrays



Komentář k charakteristikám

Záměrně byly vybrány nejvíce se lišící charakteristiky (ostatní testy se zdají být přibližně stejné). První charakteristika zachycuje situaci, kdy výkonnost klasického vykreslení scény jednou barvou ve Windows je srovnatelné s výkonností vykreslení téže scény pomocí displaylistu v Linuxu.
V obdobném duchu je druhá charakteristika – rozdíl je ještě propastnější. Jediné možné vysvětlení je různá kvalita ovladačů.