Optimalizace textur **ZOZ**

David Sedláček **15.1.2004**

1. Zadání:

Navrhněte metodu optimalizace textur mapovaných na povrch trojrozměrných objektů. Uvažujte vstupní množinu textur rozličných velikostí, barev a kompresí, které jsou namapované na plochy v prostorové scéně. Cílem je seskupit individuální textury do jednoho velkého obrazu (optimálně pokrýt jeho plochu) a aktualizovat souřadnice pro mapování textur v prostorové scéně. Implementujte metodu do podoby filtru, který provede optimalizaci textur a jejich mapování pro zadanou scénu ve formátu VRML. Na rozsáhlé množině testovacích dat měřte časové a paměťové úspory. Uvažujte prostředí lokálního počítače i webové prezentace.

Obsah:

2. Problém a jeho řešení

Při přenosu souborů pro VRML je velkým problémem nejen velikost textur, ale hlavně jejich množství pro rozsáhlejší scény. S velikostí se dá manipulovat pouze kompresí obrázků, což má za následek zhoršení kvality. Proto jsem se pokusil řešit alespoň problém počtu přenášených obrázků.

Z tohoto důvodu se musí načíst celý VRML soubor, zpracovat informace o texturách a jejich mapování, poskládat textury do minimálního počtu obrázků a zpět přepsat mapovací koordináty.

2.1 Popis programu opTex

2.1.1 Základní informace

Program `opTex` je napsán v javě, [JAVA], s přiloženou knihovnou Java Advanced Imaging (JAI) viz. [JAI], použitý parser VRML souborů viz. [IICM]. Zkompilovaný java byte-code byl následně převeden na spustitelný soubor, čímž odpadá zbytečné vypisování spouštěcího parametru pro `.jar` balíček. Nutné pro spuštění jsou pouze dva parametry, prvním je VRML soubor i s příponou `.wrl`, druhým parametrem je cílový adresář, kde se vytvoří přeformátovaný VRML soubor stejného jména jako původní a podadresář `/img` kde jsou uloženy textury.

2.1.2 Použití programu

`opTex.exe [-parametr]* inputFile.wrl outpuDirectory`

seznam parametrů
`-help` .. nápověda
`-zip` .. výstup bude zkomprimován GZIPem
`-quality:xxx` .. komprese ukládaných JPG obrázků v procentech - default=100%
`-all:PNG|JPEG` .. všechny formáty převede na PNG nebo JPEG
`-gif|png|jpeg:PNG|JPEG` .. daný formát převede na PNG nebo JPEG

`opTex.exe -zip -all:jpg –quality:75 sample.wrl newsample`

Význam: všechny textury ulož ve formátu JPEG, komprese bude 75% a výstupní .wrl soubor bude zkomprimován gZipem.

Pro jednoduchost zápisu program zároveň podporuje jednopísmenné parametry. Jiný zápis předchozího příkladu.

`opTex.exe –z –a:j –q:75 sample.wrl newsample`

[-parametr]* znamená libovolné opakování parametrů, pokud se vlastnosti parametrů shodují, program se chová dle posledního zapsaného, viz. následující příklad:

`opTex.exe –a:j –j:p sample.wrl newsample`

Všechny textury ulož ve formátu JPEG, kromě textur JPEG, které ulož jako PNG. Pozor! Ač je příkaz nelogický, filtr opTex se tak dá nastavit.

Podpora výstupního formátu GIF není, pokud ale nastavíme –gif:gif, znamená to, s GIFy nic nedělej, pouze je zkopíruj tak, jak jsou.

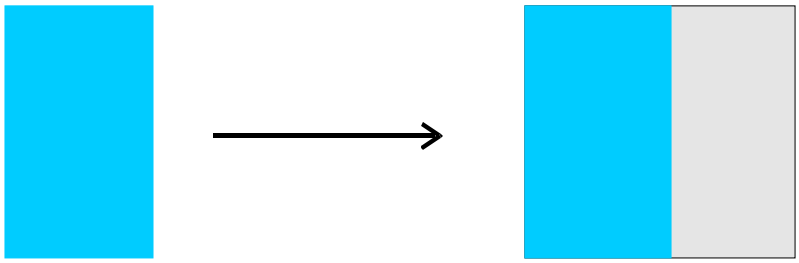
2.2 Kroky optimalizačního procesu

- Načtení vstupního VRML souboru, případné rozGzipování
    - Vše se provádí ve třídě `opTex.opTex` - hlavní třída programu
- Předání vstupu do VRML parseru, načtení do vnitřních struktur programu
    - Již zmíněný parser IICM
    - Uzly `OpShape` se ukládají do seznamu (`LinkedList items`), s kterým následně pracují všechny předzpracovávající a optimalizační metody
    - Třídy:
        - `opTex.parser.OpTexTraverser`
        - `opTex.inner.OpShape`
        - `opTex.inner.OpAppearance`
        - `opTex.inner.OpTextureCoord`
        - `opTex.inner.Vector`
- Pomocí JAI načtení základních informací o obrázcích (velikost, průhlednost…)
    - Postupně se prochází strom s uloženými uzly OpShape a načítají se obrázky z parametru URL, a pomocí metod Java Advanced Imaging se zjišťují základní informace, viz. [JAIdev], [JAItutor].
- Odstranění duplicitních obrázků
    - Je zbytečné několikrát po sobě umisťovat do výsledné textury ten samý obrázek, který je několikrát referencovaný ve vstupním souboru. Zároveň se zde provede rozřazení do stromů dle formátu bitmapy a také dle nastavených parametrů programu. Na každý strom zvlášť se pak volají následující procedury.
    - Metoda `opTex.inner.Optim.findSameText()`
- Optimalizace
    - Na vstupní data se aplikuje optimalizační metoda (aplikují se všechny dostupné a vybere se nejlepší řešení), prozatím pouze optimalizace na čtverec. Popíši dále.
- Porovnání výsledků optimalizačních metod
- Poskládání obrázků do jednoho a jejich uložení
    - `opTex.inner.SCImage.writeImg()` prochází strom s obrázky, každý SCImage má v sobě další strom s obrázky. Pokaždé se zavolá tato metoda, což má za následek vykreslování posunutých obrázků vždy o úroveň výše, až se poskládá celá výsledná textura.
    - Poté se již jen zavolá metoda pro uložení celé textury `opTex.inner.SCImage.saveJPEG/PNG()`

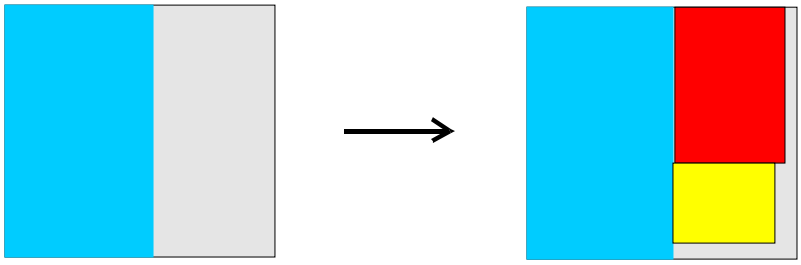
- Přepočítání mapovacích koordinátů, odstranění duplicitních referencí na stejné bitmapy
    - Zavoláme metodu `opTex.inner.SCImage.update(…)` na kořen stromu s obrázky, ona jej postupně prochází a započítává posunutí, které následně zapíše do uzlů OpAppearance.
- Zápis VRML souboru
    - `opTex.parser.OpTexTraverserSave`
- Případně nové zaGzipování

2.3 Optimalizace doplněním na čtverec

- Metoda začíná setříděním a následným vybráním nejmenšího obrázku. To má za následek minimální ztráty obsazené plochy. Kdyby se začínalo od největších obrázků, docházelo by k plýtvání místem.
- Nejmenší obrázek doplním na čtverec (zvětšení menšího okraje tak, aby plocha byla čtvercová)



- Zkusím co nejlépe (velikost pokryté plochy) doplnit zbytky po stranách
    - Toto má za následek opětovný pokus o doplnění na čtverec tentokrát menšího obrázku, postupnou rekurzí dojde k optimálnímu zaplnění plochy



- Provede se ořez přečnívajících okrajů
- Zpracuje se další obrázek

- Konec – pokud není lepší řešení s ohledem na pokrytou plochu

3. Měření

3.1 Měřící postup, použité programy

Postup:
Veškerá měření jsem provedl na osobním počítači, s přístupem na internet garantovanou linkou 64kbs/64kbs. Měřená data byla umístěna na serveru netra.`felk.cvut.cz`. Postup měření byl následující:

1) Vypnul jsem všechny programy přistupující na internet.
2) Programem Net Activity Diagram (NAD) jsem sledoval graf přenášených dat internetem. (pokud běžel jenom tento program, nepozoroval jsem a ani nezaznamenal žádnou aktivitu na internetovém spojení).
3) Do internetového prohlížeče jsem zadal adresu jednoduchého `.wrl` souboru umístěného na serveru (`start.wrl` – pouze jeden netexturovaný objekt) kterým jsem si zajišťoval zjištění IP adresy z DNS serveru a zároveň start Cortony.
4) Následoval reset počitadel přenesených dat programu NAD.
5) Zadání adresy měřeného souboru do prohlížeče s již otevřeným Cortona klientem.
6) Čas přenosu souborů jsem měřil od prvního zaznamenaného bytu po poslední (viz. grafy).
7) Opakováním bodů 3 – 6 jsem naměřil všechna data.

Použité programy a konfigurace:

Internetový prohlížeč	Maxthon (MyIE2) version 1.1.0.67 – 2004.11.09
VRML prohlížeč	Cortona VRML Client - version 4.1 (release 91), ©ParallelGraphics
Program pro měření přenesených dat	MetaProducts, Net Activity Diagram™, release 2.2.272 , trial
PC	CPU P4-3GHz, RAM 512MB, Windows XP-SP2, připojení k internetu prostřednictvím LAN, k páteřní síti připojeno přes Wi-Fi.

3.2 Ukázky měřených dat

Obr 4.2 turkwell

Obr 4.1 Graz Bell Tower

Měřená data byla vybrána s ohledem na počet a velikost textur. Toto jsou dva základní představitelé, viz.3.3- naměřené hodnoty.

3.3 Naměřené hodnoty

	Staroměstská radnice		Prašná brána		Karlův most: most		Karlův most: Staroměstská věž		Karlův most: Malostranská věž		Karlův most: celý model	
	orig	opTex	orig	opTex								
počet textur	**169**	**21**	**34**	**13**	**14**	**8**	**.--.**	**11**	**.--.**	**11**	**42**	**30**
celková velikost textur [kB]	1103	1012	458	460	48	60	.--.	136	.--.	98	285	294
velikost .wrl.gz souboru [kB]	37	38	274	268	11	11	44	42	48	48	103	102
celkem dat pro přenos [kB]	1140	1050	732	728	59	71		178		146	388	396
počet přenesených dat downLoad [kB]	1710	1095	783,15	768,53	106,08	86,27	207,61	213	130,9	166,51	443,37	454,18
počet přenesených dat upLoad [kB]	80	45	38,68	29,68	12,03	7,81	12,8	12,48	12	11,25	38,45	31,27
doba přenosu souborů / mm:ss,ms	*02:40,5*	*01:35,3*	*01:16,3*	*01:09,9*	*00:12,7*	*00:16,8*	*00:20,1*	*00:21,9*	*00:15,9*	*00:16,4*	*00:41,0*	*00:41,8*
rozdíl časů přenosu / mm:ss,ms		**01:05,2**		**00:06,4**	-	**00:04,1**	-	**00:01,8**	-	**00:00,5**	-	**00:00,9**

tabulka 3.1 - VRML data

	Maribor Synagogue		Maribor Plague		Graz Bell Tower		turkwell		sample	
	orig	opTex	orig	opTex	orig	opTex	orig	opTex	orig	opTex
počet textur	**44**	**19**	**118**	**24**	**32**	**12**	**66**	**6**	**3**	**3**
celková velikost textur [kB]	1094	786	890	631	182	196	297	290	165	165
velikost .wrl.gz souboru [kB]	24	25	476	366	93	86	18	17	2,292	1,732
celkem dat pro přenos [kB]	1118	811	1366	997	275	282	315	307	167,292	166,732
počet přenesených dat downLoad [kB]	1180	842,84	1400	1010	312,07	301,82	387,33	322,24	183,55	172
počet přenesených dat upLoad [kB]	50	34,33	97	40	27,84	16,77	53,03	13,28	7,94	8,29
doba přenosu souborů / mm:ss,ms	01:48,1	01:13,7	02:15,4	01:37,4	*00:37,3*	*00:27,1*	*00:47,7*	*00:30,5*	*00:17,3*	*00:17,5*
rozdíl časů přenosu / mm:ss,ms		**00:34,4**		**00:38,1**		**00:10,2**		**00:17,2**		**00:00,2**

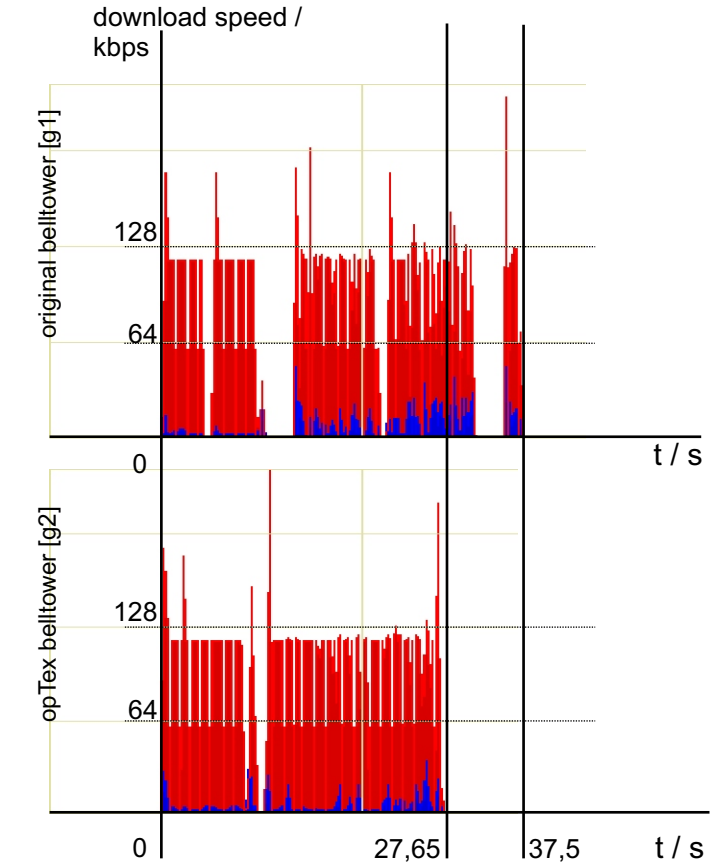
tabulka 3.2 - VRML data

	gif 4k	jpg 50k	jpg 100k	png 400k	1jpg + html	5jpg + html	16jpg + html
velikost dat pro přenos [kB]	4,901	51,057	106,256	415,173	183,183	189,369	199,435
počet přenesených dat downLoad [kB]	5,540	52,300	108,210	424,490	190,760	199,750	230,060
počet přenesených dat upLoad [kB]	0,860	1,840	3,250	11,820	6,920	8,120	17,670
doba přenosu souborů / mm:ss,ms	00:01,1	00:06,7	00:09,6	00:41,6	0:17	00:19,9	00:23,7

tabulka 3.3 - umělá data

1jpg, 5jpg, 16jpg - ten samý soubor jpg, v celku, rozdělený na 5 dílů a rozdělený na 16 dílů + *html* stránka ve které byl zobrazen

3.4 Grafy



Graf 3.1

Na grafu 3.1 je vidět rozdíl při přenosu dat s originálním počtem textur [g1] (original belltower – 34 textur) a zmenšeným počtem [g2] (opTex belltower – 13 textur). Jelikož oba přenášené vzorky jsou stejně ovlivňovány shora přenosovou rychlostí linky, mohu konstatovat, že ani jeden vzorek neměl výhodnější podmínky, tedy vyšší rychlost přenosu. Dále jsou zde vidět stažená data (download), světlejší část svislých čar a také odeslaná data (upload), spodní, tmavší část svislých čar. Hodnoty pro upload / download se překrývají. Je zřejmé, i dle naměřených dat viz. tabulka 3.2, že pro graf [g1] stoupá s počtem textur množství odeslaných dat a dále se zvyšuje množství prodlev, při čekání na další soubory s texturou. Pro ostatní data byly grafy podobné.

3.5 Závěr měření

Dle měření přenosu jednotlivých souborů (první část tabulky 3.3), jsem zjistil, že množství režijních dat je přibližně lineární (kompletní upLoad a rozdíl downLoad – skutečná velikost). Naproti tomu, když se stejné množství dat přenese rozdělené na několik částí, stoupá velikost přenesených režijních dat a tím pádem také vzrůstá čas přenosu, (druhá část tabulky 3.3). Z toho je vidět, že se rozhodně vyplatí minimalizovat počet přenášených souborů.

V druhé části, při měření na VRML datech se domněnka potvrdila. Jak je vidět na rozdílech časů v tabulkách 3.1 a 3.2. Například rozdíl při přenosu souborů pro Staroměstskou radnici je 1/3 celkového času přenosu, což činí více jak 1 minutu. Většinou jde o třetinovou až poloviční úsporu času. Pouze data Karlova mostu nejsou správně optimalizována, proto nevykazují lepší výsledky než originál (počet textur je skoro stejný jako původní a zároveň jsem nastavil špatnou kvalitu JPEG obrázků). Ale je vidět, že ačkoli byla zvolená vetší kvalita textur, při přenosu kompletní scény se časy takřka shodují (počet a velikost textur pro věže Karlova mostu v originální VRML scéně jsem nedokázal z dat určit).

4. Známé problémy, případná vylepšení

Pokud skript (např. ECMAScript) není definován jako součást jednoho z uzlů `field`, `evnetIn`, `eventOut`, parser zareaguje výjimkou, což má za následek předčasné ukončení programu.

Parser nezachovává komentáře ve VRML souboru.

Při definici dat mimo rodičovský uzel, se sice načtou a zpracují všechny textury, ale dojde následně k nesprávnému zpětnému mapování na objekt. Př:

```
DEF cs Appearance {
     Texture ImageTexture{
               url "tex/t04.jpg"
     }
}
```

Pokud by tato definice byla v uzlu `Shape`, vše by proběhlo v pořádku, na odstranění chyby aktuálně pracuji.

Nutnost dalších optimalizačních metod.

Optimalizace doplněním na čtverec zanechává v pravém dolním rohu nevyužitý prostor.

Načítání a ukládání v gZip formátu je řešeno přes odkládací soubor (`tmp.tmp`), jde o nevýhodné řešení, které trvá dlouho a program potřebuje mít povolen zápis do adresáře s programem.

5. Zdroje, potřebné knihovny, literatura

[JAI] Java Advanced Imaging - nutno nainstalovat do java sdk - http://java.sun.com/products/java-media/jai/

[IICM] knihovna *VRML parser verze 1.1* od institutu IICM - http://www.iicm.edu/vrwave/pw

[JAVA] java SDK - min. verze 1.4 - http://java.sun.com

[Žára99] J.Žára – *VRML 97, Laskavý průvodce virtuálními světy*, Computer Press, 1999, ISBN: 80-7226-143-6.

[JAIdev] http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/ - *Programming in Java Advanced Imaging* – Sun microsystems, 1999

[JAItutor] http://java.sun.com/developer/onlineTraining/javaai/jai/index.html - *JAI tutorial*