VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIRTUÁLNÍ TEXTUROVÁNÍ

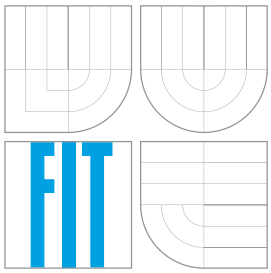
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK BIBERLE

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIRTUÁLNÍ TEXTUROVÁNÍ

VIRTUAL TEXTURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK BIBERLE

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2014

Abstrakt

Tato práce pojednává o technice známé jako Virtuální texturování či Megatexture. Popisuje způsoby implementace této techniky na běžně dostupném grafickém hardwaru. Popisuje také programovou knihovnu VortexVT, která byla vyvinuta pro zjednodušení implementace virtuálního texturování v nových či již existujících grafických systémech.

Abstract

This thesis deals with a technique known as Virtual textures or Megatextures. It describes possible ways of implementing this technique on consumer-grade graphics hardware. It also touches on the subject of the VortexVT library, a library developed to ease integration of virtual texturing into new or already existing graphics systems.

Klíčová slova

počítačová grafika, virtuální textura, megatexture, sparse texture, OpenGL

Keywords

computer graphics, virtual texture, megatexture, sparse texture, OpenGL

Citace

Zdeněk Biberle: Virtuální texturování, bakalářská práce, Brno, FIT VUT v Brně, 2014

Virtuální texturování

Prohlášení

Prohlažuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Adama Herouta, Ph.D. a že jsem uvedl všechny prameny, ze kterých jsem čerpal.

.....
Zdeněk Biberle
21. května 2014

Poděkování

Za trpělivost a nápady děkuji vedoucímu práce Doc. Ing. Adamu Heroutovi, Ph.D.

Obsah

Kapitola 1

Úvod

Trend vývoje 3D grafických technologií směřuje v současné době ke stále větším a bohatším scénám. Tento trend je zjevný primárně v počítačových hrách, ale týká se i profesionálních nástrojů pro práci s počítačovou grafikou ve třech rozměrech, které s podobně složitými, ne-li složitějšími, scénami musí taktéž umět pracovat.

Jedním z faktorů, které dělají práci s detailními scénami složitější, je omezená velikost paměti grafických akcelerátorů. Tento faktor limituje hlavně velikost použitých textur. Je ovšem důležité si povšimnout, že v typických případech je velikost zobrazovacích zařízení mnohem nižší, než celková velikost použitých textur. To nám umožňuje využívat grafickou paměť efektivně a ponechávat v ní pouze pouze omezené množství texelů.

Výše popsaná technika je známá pod názvem Virtual texture či Megatexture. O implementaci této techniky pro dosažení vykreslování scén využívajících textury s velkým množstvím texelů v reálném čase pojednává právě tato práce. Cílem je shromáždit klíčové informace pro implementaci virtuálního texturování a získat praktické zkušenosti implementací programové knihovny.

Kapitola 2

Popis metody

Jak již bylo zmíněno v úvodu, základní myšlenkou virtuálního texturování je udržování pouze důležitých dat v grafické paměti. Toho dosáhneme jednoduše rozdělením původní textury na adekvátní množství menších částí, tzv. stránek.[1]

Poté si v grafické paměti vyhradíme prostor pro dostatečně velké množství těchto stránek a při každém snímku se ujistíme, že v této paměti máme nahrány stránky, které náleží k částem textury, která jsou dle současné scény viditelné, případně budou viditelné v blízké budoucnosti. Při vykreslování scény poté za použití fragment shaderů zařídíme korektní vykreslení textury.

Celý proces virtuálního texturování lze tedy rozdělit na několik částí:

1. Shromáždění souřadnic viditelných stránek
2. Nahrání dat stránek z patřičného zdroje stránek
3. Uložení dat stránek na vhodné volné místo do grafické paměti
4. Vykreslení scény

Naivní implementace těchto kroků ovšem přináší několik dalších překážek, které je nutné překonat. Konkrétně se jedná o následující:

- Stále může nastat případ, kdy je viditelná celá plocha virtuální textury. To by znamenalo vytvoření požadavků na nahrání všech stránek do paměti a v takovém případě virtuální texturování ztrácí zcela smysl.
- Pokud budeme všechny stránky v grafické paměti shromažďovat v jedné textuře, což se dle [2] jeví jako nejefektivnější řešení, tak narazíme na problémy s filtrováním. Hardwarové filtrovací jednotky totiž zcela pochopitelně předpokládají, že dva texely, které s sebou fyzicky sousedí, s sebou sousedí i logicky. To v našem případě, kdy jednoduše ukládáme na libovolná místa v grafické paměti libovolné stránky, nebude na okrajích těchto stránek vždy platit.
- Jelikož budeme stránky zpravidla nahrávat z pevného disku, tak musíme počítat s tím, že tato operace nějakou dobu trvá. Musíme tedy řešit situaci, kdy potřebujeme vykreslit část virtuální textury, která dosud nebyla nahrána.

Řešením těchto problémů se zabývá následující kapitola. Popisem možných implementací klíčových částí virtuálního texturování se pak zabývá kapitola 4.

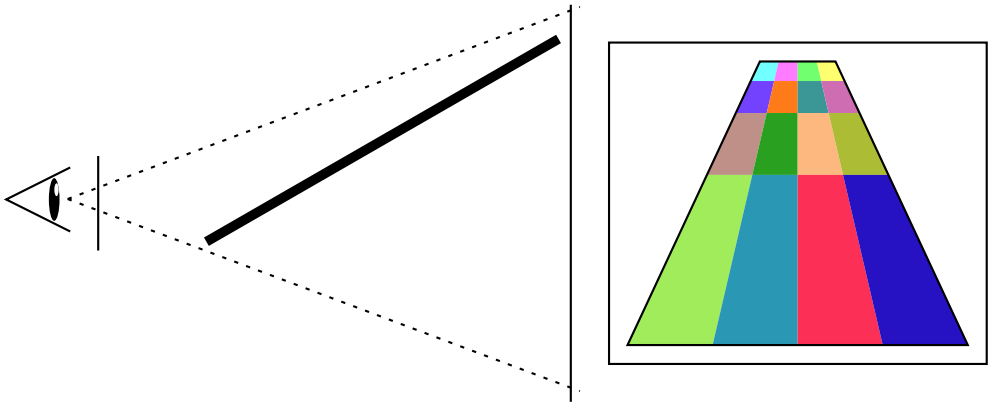
[1] Pojmenování "stránka" bylo zvoleno z důvodu podobnosti virtuálního texturování s virtuální pamětí, kde se taktéž používá tento termín.

Kapitola 3

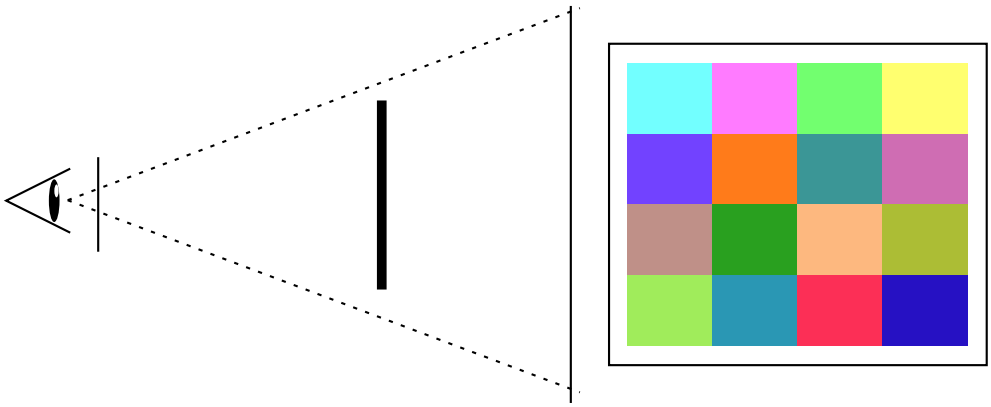
Řešení typických problémů

V kapitole 2 byly popsány některé problémy, které jsou typické pro naivní implementaci virtuálního texturování. Tato kapitola popisuje možnosti řešení těchto problémů.

3.1 Nadměrné množství nahrávaných stránek



Obrázek 3.1: Demonstrace situace, ve které je viditelná celá plocha virtuální textury



Obrázek 3.2: Další demonstrace situace, ve které je viditelná celá plocha virtuální textury

Obrázek 3.1 demontruje případ, kdy je viditelná celá plocha virtuální textury. Nahrání celé virtuální textury do paměti je ovšem nežádoucí. Řešení je ale překvapivě jednoduché.

Všimněme si, že v zobrazené situaci se značná část textury nachází poblíž kamery a vyžaduje tedy větší množství detailů, než zbytek textury. Dále si povšimněme, že v případě, že se všechny části textury nachází stejně daleko od kamery (tj. tak, jak je vyobrazeno v obrázku 3.2), tak zpravidla platí, že velikost zobrazovacího zařízení je menší než velikost virtuální textury.

Pro obě tyto situace tedy platí následující fakt: některé (ne-li všechny) části virtuální textury můžeme do grafické paměti nahrát se zmenšeným množstvím detailů bez jakékoliv degradace celkové vizuální kvality.

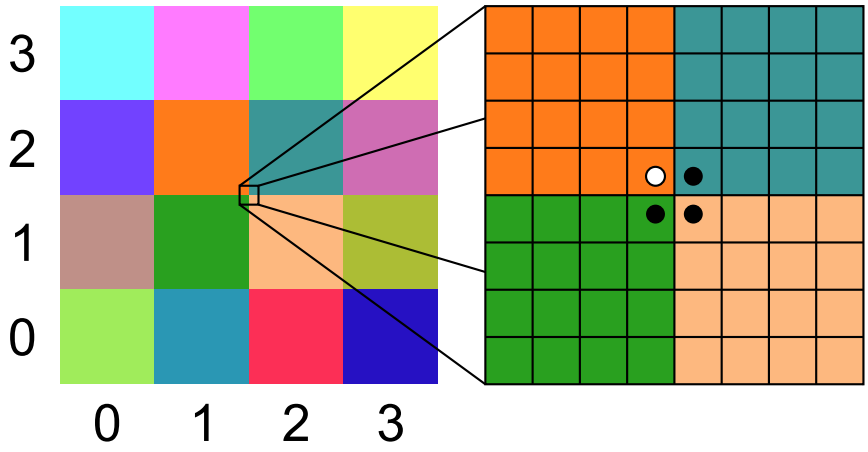
Jak ovšem definujeme zmenšené množství detailů? Stejně jako v případě běžného mipmapování použijeme jako základ pro tyto části původní virtuální texturu, ovšem zmenšenou o vhodný faktor. Tento faktor musí být mocnina dvou.

Otázkou ovšem zůstává, kde získáme a kde budeme skladovat tyto části s menšími detaily (pro jednoduchost je budeme taktéž nazývat stránkami). Rozumným řešením je pracovat s nimi stejně jako s ostatními běžnými stránkami. Z toho vyplývá, že tyto stránky budou mít i stejnou velikost jako běžné stránky. Z tohoto důvodu musíme zajistit, že celkový počet stránek virtuální textury vertikálně i horizontálně je mocnina dvou a zároveň je v obou rozměrech stejný.

Vedlejším efektem našeho snažení je, že se zbavíme nutnosti generovat mipmapy jednotlivých stránek při nahrávání, což může potenciálně být značná výkonnostní výhoda.

3.2 Nekorektní filtrování na okrajích stránek

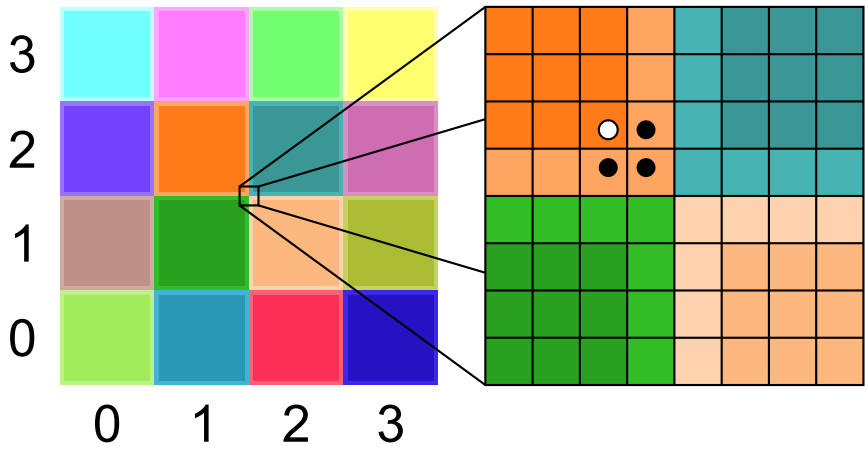
Všechny běžně používané hardwarové metody filtrování textur (tj. bilineární, trilineární a anizotropní) využívají při vzorkování textur nejen texel, který se nachází na vzorkovaných souřadnicích, ale i texely okolní. Problém, který díky tomuto vzniká, demonstruje obrázek 3.3. Na tomto obrázku vidíme v levé části texturu, která obsahuje šestnáct stránek z virtuální textury. Při vzorkování stránky na souřadnicích $(1, 2)$ došlo i ke vzorkování texelů ze stránek okolních, tj. na souřadnicích $(1, 1)$, $(2, 1)$ a $(2, 2)$. Jelikož tyto stránky spolu nemusí mít nic společného, tak nastává situace, kdy se nám do vzorkované stránky prolíná nežádoucí barva.



Obrázek 3.3: Demonstrace nevhodného filtrování textur

3.2.1 Bilineární filtrování

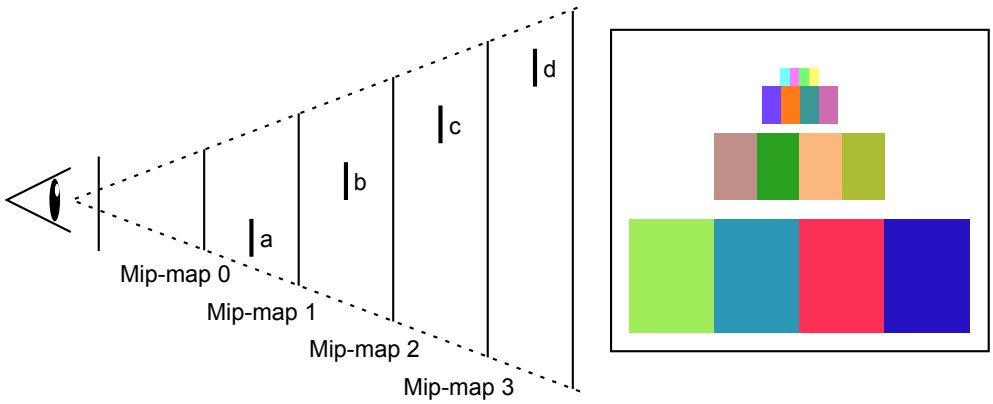
Řešení je v případě bilineárního filtrování poměrně jednoduché. [1] uvádí, že stačí ke každé stránce uložené v grafické paměti přidat dodatečný okraj šířky jednoho texelu, který obsahuje korektní okolní data, a upravit výpočet souřadnic ve fragment shaderu. Výsledek demonstruje obrázek 3.4.



Obrázek 3.4: Demonstrace vhodného bilineárního filtrování textur

3.2.2 Trilineární filtrování

Trilineární filtrování přidává k bilineárnímu filtrování prolínání mezi dvěma vhodnými mipmap úrovněmi. Jako vhodné řešení se tedy nabízí přidat k textuře se stránkami jednu mipmap úroveň. Toto řešení má ovšem lehce přehlédnutelný problém, který lze demonstrovat na obrázku 3.5. Zde vidíme, že část a se nachází mezi mipmap úrovněmi 0 a 1. Filtrování pro tuto část bude tedy fungovat správně. Bohužel v případě části b (a podobně i v případě částí c a d) dojde ke zvolení mipmap úrovní 1 a 2 (respektive 2 a 3 či 3 a 4) . Jelikož ale máme texturu se stránkami pouze s mipmap úrovněmi 0 a 1, tak dojde k redukci trilineárního filtrování na bilineární filtrování na mipmap úrovni 1 a tedy k redukci detailů.



Obrázek 3.5: Demonstrace nevhodného výběru mipmapy při trilineárním filtrování

Jádrem problému je, že výpočet úrovně použité mipmapy je prováděn výchozím způsobem grafické karty a je tudíž založen na souřadnicích, které se nachází v prostoru textury se stránkami. Správný výpočet úrovně mipmapy by ale měl být založen na souřadnicích v prostoru virtuální textury. Musíme tedy tento výpočet provést sami ve fragment shaderu za použití derivace souřadnic virtuální textury (v GLSL shaderech lze použít funkce `dFdx` a `dFdy`) a vzorkování textury se stránkami provést vhodným způsobem, který nám umožňuje specifikovat úroveň detailů (v GLSL shaderech tedy pomocí funkce `texture2DGrad`).

Jelikož trilineární filtrování vychází z filtrování bilineárního, tak musíme taktéž ke každé stránce přidat vhodný okraj, ale narozdíl od bilineárního filtrování potřebujeme okraj alespoň jeden texel široký i na mipmap úrovní 1. Z toho vyplývá, že šířka okraje v mipmap úrovni 0 musí být alespoň dva texely.

3.2.3 Anizotropní filtrování

Dle [3] lze anizotropní filtrování virtuálních textur implementovat velice podobně jako trilineární filtrování v předchozí části. Jediným rozdílem je nutnost většího okraje stránek.

3.2.4 Shrnutí

Jelikož máme většinou každou stránku nahranou s úrovní detailů, která je blízká tomu, co scéna právě v daný okamžik potřebuje, tak se nemusíme zabývat mipmapováním a v naprosté většině případů nám stačí využít bilineárního filtrování. Využití lepší filtrovací metody, například anizotropního filtrování, by přineslo užitek pouze, pokud máme značné množství geometrie, na kterou je shlíženo pod ostrým úhlem.

3.3 Vykreslování dosud nenahraných stránek

V případě, že nahráváme stránky z relativně pomalého zdroje jako je například pevný disk počítače či optické médium, tak nelze přehlédnout, že tato operace může trvat nadměrně dlouhou dobu a provádění těchto operací blokujícím způsobem tedy znemožňuje dosažení interaktivního počtu snímků za sekundu. Přirozeným řešením tedy je nahrávání stránek asynchronně. V tomto případě ovšem musíme řešit stav, kdy ke korektnímu vykreslení scény potřebujeme stránky, které dosud nejsou nahrané.

Naštěstí máme díky činnosti z části 3.1 již existující systém, který můžeme pro tyto případy využít.

Řešením je tedy mít neustále nahránu alespoň stránku s celou virtuální texturou s nejmenším množstvím detailů. Tato stránka bude vykreslována tak dlouho, dokud nebude dokončeno nahrání vhodnější stránky.

Kapitola 4

Možnosti implementace klíčových částí

Tato kapitola popisuje možné způsoby implementace klíčových částí systému virtuálního texturování.

4.1 Shromáždění souřadnic viditelných stránek

Pro zajištění korektního vykreslení scény je nutné vědět, které části virtuální textury jsou v současnosti viditelné a které je tedy nutné nahrát do grafické paměti.

[2] a [4] uvádí jako řešení vykreslení scény do zpětnovazebního bufferu za použití speciálního fragment shaderu. Další možností je analytické řešení založené na znalosti vykreslované scény. V dalších částech se podíváme na obě možnosti blíže.

4.1.1 Zpětnovazební buffer

Tato metoda využívá dodatečného bufferu a speciálního fragment shaderu pro vykreslení scény a vyplnění zmíněného bufferu souřadnicemi vykreslovaných stránek a požadovanou úrovní detailů. Tento postup má několik výhod:

- Jedná se o dostatečně univerzální řešení, které lze bez velkých problémů použít pro prakticky libovolnou geometrii.
- Díky použití z-bufferu nemusíme nahrávat stránky, které jsou skryty jinou geometrií.
- Je možné měnit tendenci systému nahrávat některé stránky změnou projekční matice či jiných parametrů scény.
- Pokud vhodně upravíme vykreslení scény, tak můžeme provádět jednoduchou predikci v budoucnu potřebných stránek, čímž můžeme kompenzovat vyšší latenci nahrávání stránek. Jednoduchým způsobem jak tuto predikci provádět je například odhad budoucí pozice objektů ve scéně na základě jejich aktuální pozice, směru a rychlosti.

Některé tyto výhody jsou ovšem vyváženy následujícími nevýhodami:

- Celý zpětnovazební buffer musíme analyzovat na CPU, což může být v případě velkého zpětnovazebního bufferu časově velmi náročná operace.

- Částečně průhledné objekty představují problém, jelikož jeden fragment zpětnovazebního bufferu může nést informaci pouze o jedné stránce.

Ovšem uvědomme si, že scéna vykreslovaná při zpětnovazební operaci prakticky vůbec nemusí připomínat finální zobrazovanou scénu. Musí pouze obsahovat stejné souřadnice textur. Máme tedy hodně volnosti, což nám umožňuje experimentovat a najít tak vhodný způsob vykreslení této scény pro naši konkrétní aplikaci.

Kupříkladu efekty obou zmíněných nevýhod lze různými postupy minimalizovat. Velikost zpětnovazebního bufferu totiž může být mnohem menší, než velikost zobrazovaného framebufferu aplikace. Tímto podstatně snížíme množství práce na CPU. Vedlejším efektem navíc je, že stránky, které v obraze zabírají velmi malou plochu (obecně asi v řádech jednotek pixelů) budou ignorovány. Díky tomuto nemusíme nahrávat stránky, které příliš nepřispívají k vykreslení finální scény, což nepředstavuje problém díky mechanismu z 3.3.

Druhou nevýhodu, tj. problémy s průhledností, lze minimalizovat například vykreslením částečně průhledné geometrie za použití tečkování (například v OpenGL lze použít `glPolygonStipple`).

Nyní se ale přesuňme ke zmíněnému fragment shaderu, který bude použit pro vykreslení scény do zpětnovazebního bufferu. Tento shader musí provést dvě klíčové operace:

1. Převést vstupní souřadnice vzorku na souřadnice stránky v prostoru virtuální textury.
2. Zjistit potřebnou úroveň detailů této stránky.

Převod souřadnic

Pro převod standardních texturovacích souřadnic v rozsahu $\langle 0, 1)$ lze použít velice jednoduchý výraz:

$$\vec{s} = \lfloor \vec{v} \cdot p \rfloor \tag{4.1}$$

Zde $\vec{s}$ je vektor souřadnic stránky, $\vec{v}$ je vektor souřadnic vzorku a p je počet stránek na straně virtuální textury.[1]

Zjištění úrovně detailů

Abychom nenahrávali pouze stránky s nejvyšším množstvím detailů, tak potřebujeme zjistit, jakou úroveň detailů jednotlivých stránek právě potřebujeme. Pro označení úrovně detailů zvolíme zkratku LOD. LOD 0 bude náležet nejvyššímu množství detailů a každé další vyšší celé číslo bude značit o polovinu méně detailů. Celkový počet úrovní detailů je pak omezen počtem stránek virtuální textury.

Pro zjištění LOD lze použít derivaci souřadnic vzorku virtuální textury (v OpenGL dostupné pomocí funkcí `dFdx` a `dFdy`). Pokud tyto hodnoty vynásobíme počtem texelů na straně virtuální textury, tak získáme počet texelů, které spadají do jednoho fragmentu. Z této hodnoty již můžeme vypočítat LOD:

$$LOD = \max\left(0, \log_2\left(\min\left(\left|\frac{d\vec{v}}{dx}\right|, \left|\frac{d\vec{v}}{dy}\right|\right) \cdot r\right)\right) \tag{4.2}$$

Zde $\vec{v}$ je vektor souřadnic vzorku v prostoru virtuální textury, r je rozměr strany virtuální textury v texelech. Derivace podle x a podle y jsou derivace dle os framebufferu.

[1] Jak již bylo vysvětleno v kapitole 3.1, tak vertikální i horizontální počet stránek ve virtuální textuře je stejný.

4.1.2 Analytické řešení

V určitých případech můžeme vygenerovat souřadnice potřebných stránek sami na základě znalostí vykreslované geometrie.

Jednou z možných implementací je mít přiřazené souřadnice stránek k jednotlivým částem geometrie a na základě pozice a směru kamery tyto souřadnice vybrat a dle vzdálenosti kamery od geometrie patřičně upravit hodnotu úrovně detailů.

Jelikož se ovšem jedná o velice neobecnou metodu s omezeným použitím, tak se jí dále zabývat nebudeme.

4.2 Nahrávání dat stránek ze zdroje stránek

Vyžadujeme jednoduchý a poměrně rychlý přístup k jednotlivým stránkám, tudíž se jako rozumné řešení jeví skladovat každou stránku v samostatném souboru na pevném disku či optickém médiu. Pokud navíc zachováme stránky dostatečně malé, tak si můžeme dovolit i jejich kompresi (zřejmě některým z běžně dostupných kompresních formátů jako PNG či JPEG).

Vhodným dodatkem může být i vyrovnávací paměť umístěná v operační paměti. Tato vyrovnávací paměť může být užitečná především v případě, že máme k dispozici nepříliš velké množství grafické paměti a nechceme se nadbytečně spoléhat na pevný disk či optické médium, jelikož tyto mají zpravidla nadměrně vysokou přístupovou dobu.

Další možností zlepšení nahrávání stránek je provádění této činnosti asynchronně. To pomáhá především v případech, kdy dojde k náhlému pohybu či zobrazení většího množství nové geometrie, jelikož potřebujeme pro tuto geometrii nahrát stránky. Pokud by toto nahrávání běželo synchronně jako součást hlavní smyčky programu, tak by vykreslení aktuálního snímku muselo počkat, než tato operace skončí, což může trvat i několik stovek milisekund, což je z pochopitelných důvodů příliš dlouho.

Při asynchronním nahrávání stránek bude sice tato činnost opticky viditelná[2], ovšem minimalizujeme skoky počtu snímků za sekundu.

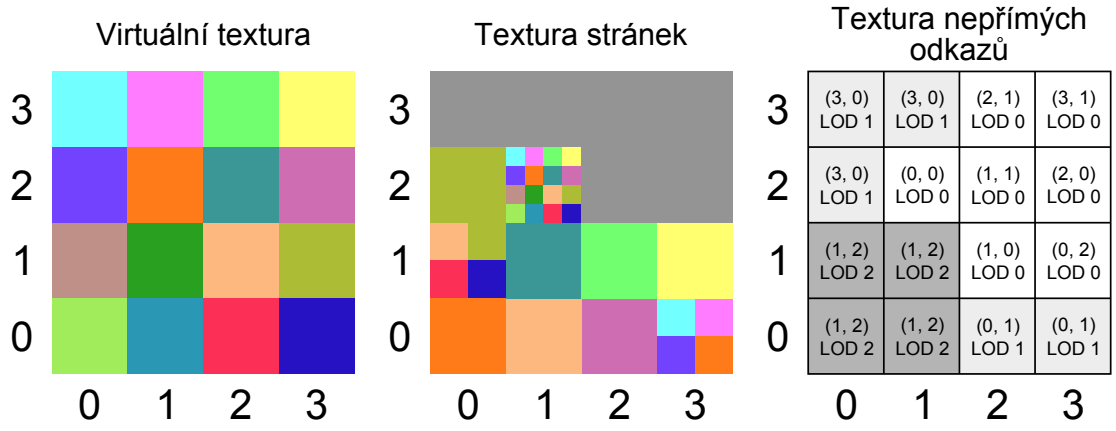
4.3 Uložení dat stránek do grafické paměti

Při ukládání stránek do grafické paměti máme pouze několik možností. Můžeme každou stránku uložit do samostatné textury a při vykreslování použít pole textur pro přístup k těmto texturám z fragment shaderu. Druhou možností je ukládat stránky do jedné, dostatečně velké textury. Tento přístup je dle [2] podstatně vhodnější co se týče výkonu. Další výhodou je větší flexibilita ohledně rozměrů stránek. To je užitečně obzvlášť v případě, kdy chceme jednoduše měnit šířku dodatečných okrajů stránek.[3]

Když tedy máme stránky umístěné v grafické paměti, ke které můžeme přistupovat z fragment shaderu pro vykreslení scény, tak ještě musíme zajistit, aby tento fragment shader věděl, kde se nachází konkrétní stránky. Pro tento účel můžeme využít hash mapy, quadtree nebo jednoduchou look-up tabulku. Prohledávání hash map a quadtree ve fragment shaderu by ovšem vyžadovalo zbytečně složité a pro fragment shader nevhodné kontrolní struktury. Proto se uchýlíme k look-up tabulce, kterou implementujeme pomocí textury.

[2] Tzv. texture popping.

[3] Účel dodatečných okrajů stránek byl vysvětlen v kapitole 3.2.



Obrázek 4.1: Příklad demonstrující texturu nepřímých odkazů

Tuto texturu budeme nazývat textura nepřímých odkazů. Každý texel v této textuře se nachází na stejném relativním místě, jako jemu přiřazená stránka ve virtuální textuře. Každý tento texel obsahuje informaci o umístění přiřazené stránky v textuře stránek a o její úrovni detailů.

Finální výsledek je demonstrován na obrázku 4.1. Zde vidíme původní virtuální texturu, několik stránek z této textury nahraných do textury stránek a vidíme také texturu nepřímých odkazů.

4.4 Vykreslení scény

Pro finální vykreslení použijeme fragment shader, který provede transformaci souřadnic, které se nachází v prostoru virtuální textury, na souřadnice v prostoru textury stránek. K tomuto využije texturu nepřímých odkazů, která byla zmíněna v předchozí části. Celý postup může vypadat následovně:

$$\begin{aligned} \vec{s}, lod &= \text{textura nepřímých odkazů}\,(\vec{v}) \\ \vec{f} &= \frac{\text{fract}\left(\frac{\vec{v} \cdot p}{lod}\right) + \vec{s}}{p} \end{aligned} \tag{4.3}$$

Zde $\vec{v}$ je vstupní vektor se souřadnicemi v prostoru virtuální textury, $\vec{s}$ jsou souřadnice stránky v textuře stránek, p je počet stránek na straně virtuální textury a $\vec{f}$ je vektor s finálními souřadnicemi použitelný pro přístup k textuře stránek.

Postup použitelný pro přístup ke stránkám s dodatečnými okraji se od tohoto postupu příliš neliší a může vypadat následovně:

$$\begin{aligned} \vec{s}, lod &= \text{textura nepřímých odkazů}\,(\vec{v}) \\ \vec{f} &= \frac{\text{fract}\left(\frac{\vec{v} \cdot p}{lod}\right) \cdot a + \vec{s} + (o, o\ )}{p} \end{aligned} \tag{4.4}$$

Tento vzore byl doplněn pouze o dvě další hodnoty: a je poměr velikosti samotné stránky vůči velikosti stránky včetně okrajů a o je poměr šířky okraje vůču velikosti stránky včetně okrajů.

4.5 Shrnutí kapitoly

Tato kapitola se zabývala čtyřmi hlavními částmi implementace virtuálního texturování. Schopný čtenář by nyní měl být schopný implementovat alespoň základní funkcionalitu virtuálního texturování, a to i na běžně dostupném grafickém hardwaru. Zmíněné postupy totiž nevyužívají prakticky žádné pokročilé vlastnosti grafických akcelerátorů.

Kapitola 5

Knihovna VortexVT

Zaměřením této kapitoly je knihovna VortexVT, jejíž vytvoření bylo hlavním cílem této bakalářské práce. Tato knihovna zjednodušuje většinu kroků popsaných v předchozích kapitolách při implementaci virtuálního texturování pomocí grafické knihovny OpenGL. Knihovna obsahuje generické komponenty především v oblasti nahrávání stránek, ze kterých lze kompozicí tvořit složité a pro konkrétní aplikaci vhodné stránky-nahrávající řetězce.

5.1 Přehled knihovny

Knihovna je psána v programovacím jazyce D2[1] a tudíž je možné ji použít z libovolného prostředí, které umožňuje volání nativního kódu, ovšem nejjednodušší je ji použít opět z jazyka D2.

Převážná část knihovny je tvořena šablonami, což umožňuje specializaci jednotlivých komponent pro konkrétní aplikaci bez nadbytečné run-time režie.

Šablony jsou také použity pro dosažení statického polymorfismu. Statickým polymorfismem je omezeno množství nepřímých volání a zároveň je zvýšena lokalita dat, což vede k potenciálnímu zvýšení výkonu.

Knihovna řeší všechny klíčové části popsané v kapitole 2 pomocí několika modulů. Tyto moduly jsou popsány v následujících částech.

5.1.1 Modul `vortexvt.tilerequestprovider`

Tento modul obsahuje strukturu `FboTileRequestProvider`, která za použití PBO[2] a FBO[3], tj. pomocí metody zpětnovazebního bufferu popsané v kapitole 4.1.1 generuje požadavky na nahrávání stránek.

Použitý framebuffer využívá běžného BGRA formátu s osmi bity na každý kanál, celkově je tedy dostupných 32 bitů. Rozložení těchto bitů je následující:

- 12 bitů - vertikální souřadnice stránky
- 12 bitů - horizontální souřadnice stránky
- 4 bity - LOD

[1] Viz http://dlang.org/.
[2] Pixel Buffer Object
[3] Framebuffer Object

- 4 bity - ID virtuální textury

Toto rozdělní bylo zvoleno tak, aby bylo dosaženo virtuálních textur o velikosti až 4096 × 4096 stránek. Tomu odpovídá i zvolený počet bitů pro úroveň detailů, tj. čtyři bity pro šestnáct úrovní detailů. Z těchto šestnácti úrovní je třináct (tj. úrovně 0 až 12) využito přímo jako LOD. Úroveň 15 je využita jako označení pixelu, který nevyžaduje nahrání žádné stránky. Ostatní úrovně jsou rezervovány pro budoucí použití.

Poslední čtyři bity jsou použity pro identifikaci virtuální textury. Je tedy možné dohromady používat až šestnáct různých virtuálních textur.

Typické použití struktury `FboTileRequestProvider` je následující:

```
1  // V~inicializační sekci aplikace vytvoříme instanci.
2  // Parametry konstruktoru určují velikost zpětnovazebního bufferu,
3  // počet stránek virtuální textury na straně virtuální textury
4  // a počet úrovní detailů.
5  // Pro získání posledních dvou parametrů můžeme použít vlastnosti
6  // struktur z~další části.
7  auto requestProvider = FboTileRequestProvider(
8      256,
9      256,
10     tileProvider.tileCount,
11     tileProvider.lodCount
12 );
13
14 // Do hlavní smyčky aplikace přidáme dodatečné vykreslení scény do
15 // zpětnovazebnáho bufferu.
16 requestProvider.prepareTileRequestRendering();
17
18 // Zde provedeme aktivaci vhodných shaderů a vykreslení scény
19
20 requestProvider.finishTileRequestRendering();
21
22 // Nyní můžeme využít vlastnost data poskytovatele požadavků na
23 // stránky k~získání dvourozměrného pole těchto požadavků. První
24 // úroveň tohoto pole je indexována dle ID virtuální textury.
25 // Druhá úroveň tohoto pole je jednoduchý seznam požadavků.
26 // Typicky tuto druhou úroveň předáme přímo do kompozitoru
27 // textury stránek.
28 compositor.loadTiles(requestProvider.data.front);
29
30 // Na konci aplikace je nutné uvolnit prostředky
31 requestProvider.destroy();
```

5.1.2 Modul `vortexvt.tileprovider`

Tento modul obsahuje hned několik struktur, které dohromady tvoří komplexní systém nahrávání stránek. Významnými strukturami jsou `SimpleDevilTileProvider`, `CachingTileProviderWrapper`, `BorderedTileProviderWrapper` a `AsyncTileProviderWrapper`.

`SimpleDevilTileProvider` je určen pro jednoduché nahrávání stránek ze souborů za použití knihovny DevIL[4]. Jako zdroj stránek využívá složku s obrazovými soubory a in-

[4] Viz http://openil.sourceforge.net/.

formační soubor uložený v téže složce, který specifikuje různé vlastnosti dané virtuální textury.[5]

`CachingTileProviderWrapper` obaluje libovolný jiný poskytovatel stránek a tvoří nad ním cache stránek v paměti RAM s nastavitelnou velikostí a LRU[6] strategií.

`BorderedTileProviderWrapper` taktéž obaluje jiný poskytovatel stránek a generuje pomocí něj stránky s libovolně širokým okrajem, což umožňuje využí filtrování textur.

A konečně `AsyncTileProviderWrapper` opět obaluje libovolný jiný poskytovatel stránek a přidává asynchronní nahrávání stránek.

Dodatečně modul také obsahuje rozhraní `ITileProvider` a třídu `TileProviderWrapper`, která implementuje toto rozhraní a zároveň obaluje jiný poskytovatel stránek. To umožňuje využít polymorfismu a referenční sémantiky v případě, že je to v dané situaci vhodné.

Komponovatelnosti těchto datových typů je dosaženo použitím šablon. Struktury `CachingTileProviderWrapper`, `BorderedTileProviderWrapper`, `AsyncTileProviderWrapper` i třída `TileProviderWrapper` mají šablonový parametr (zpravidla pojmenovaný `WrappedProvider`), který určuje typ obaleného poskytovatele stránek. Instance tohoto poskytovatele bude poté umístěna přímo do poskytovatele obalujícího poskytovatele. Chceme-li tedy například poskytovatel stránek založený na `SimpleDevilTileProvider` ovšem s dodatečným cachováním, tak můžeme použít následující kód:

```
auto tileProvider = CachingTileProviderWrapper!SimpleDevilTileProvider(
    velikostCache,
    adresaTextury // adresa textury pro konstrukci SimpleDevilTileProvider
);
```

Zde zároveň vidíme jeden z možných způsobů konstrukce vnořených poskytovatelů stránek, tj. předání parametrů konstruktoru vnořeného poskytovatele přes konstruktor obalujícího poskytovatele. Druhou možností je přímo předání instance obaleného poskytovatele:

```
auto tileProvider = CachingTileProviderWrapper!SimpleDevilTileProvider(
    velikostCache,
    SimpleDevilTileProvider(adresaTextury)
);
```

Takto předaný poskytovatel poté bude zkopírován dovnitř obalujícího poskytovatele.

V případě, že potřebujeme indirekci mezi jednotlivými poskytovateli, tak můžeme využít rozhraní `ITileProvider` a obalující třídu `TileProviderWrapper`:

```
auto sdtp = new TileProviderWrapper!SimpleDevilTileProvider(
    adresaTextury
);
auto ctpw = CachingTileProviderWrapper!ITileProvider(
    velikostCache,
    sdtp
);
```

Alternativou je využití ukazatelů:

```
auto sdtp = new SimpleDevilTileProvider(
    adresaTextury
);
auto ctpw = CachingTileProviderWrapper!(SimpleDevilTileProvider*)(
    velikostCache,
```

[5] Více o tomto souboru viz příloha B.

[6] Least Recently Used

```
6       sdtp
7   );
```

Je tedy zjevné, že můžeme konstruovat různě složité řetězce poskytovatelů stránek bez zbytečné indirekce. Příkladem takového složitějšího řetězce může být poskytovatel stránek z demonstrační aplikace, který vypadá následovně:

```
1  auto tileProvider =
2      CachingTileProviderWrapper!(
3          AsyncTileProviderWrapper!(
4              BorderedTileProviderWrapper!(
5                  CachingTileProviderWrapper(!
6                      SimpleDevilTileProvider
7                  )
8              )
9          )
10     )(
11     512, // velikost druhé cache
12     512, // velikost asynchronní cache
13     128, // velikost první cache
14     buildPath("resources/textures", modelName) // adresa textury
15 );
```

5.1.3 Modul `vortexvt.compositor`

`vortexvt.compositor` je modul obsahující jedinou strukturu, `Compositor`. Tato třída slouží pro kompozici textury stránek a generování textury nepřímých odkazů. K nahrávání stránek `Compositor` využívá libovolný dodaný poskytovatel stránek. `Compositor` zvládá stránky libovolné velikosti i dodatečné okraje stránek.

Kompozitor skládá stránky to textury stránek postupně přímo vedle sebe. V případě naplnění textury stránek je použita strategie LRU[7] pro vybrání stránek k přepsání.

5.1.4 GLSL shadery

Knihovna VortexVT obsahuje samozřejmě i zdrojové kódy GLSL funkcí, které lze vložit do vlastní shaderové pipeline. Obsažen je fragment shader doplňující strukturu `FboTileRequestProvider` a také fragment shadery pro vzorkování virtuálních textur bez filtrování i s bilineárním filtrováním.

Shader `makeTileRequest.frag`

`makeTileRequest.frag` je fragmentový shader určený pro spolupráci s modulem `vortexvt.tilerequestprovider`. Tento shader obsahuje jedinou funkci, `makeTileRequest`, která generuje data pro výstupní pixely zpětnovazebního bufferu.

Parametry této funkce jsou:

- `vtTexCoord` - Vzorkovaná souřadnice v souřadnicovém prostoru virtuální textury.
- `vtId` - Identifikační číslo virtuální textury. Pomocí tohoto parametru lze rozlišit více (až 16) různých virtuálních textur.

[7] Least Recently Used

- `vtTileCount` - Počet stránek na straně virtuální textury.
- `vtSize` - Velikost virtuální textury v texelech. Opět se jedná o vertikální i horizontální velikost.
- `lodBias` - Násobič LOD hodnot pro kompenzaci rozdílu mezi velikostí hlavního framebufferu a zpětnovazebního bufferu. Výpočet této hodnoty lze provést následujícím způsobem:

$$lodBias = \min\left(\frac{z_1}{f_1}, \frac{z_2}{f_2}\right) \tag{5.1}$$

  Kde $\vec{z}$ je velikost zpětnovazebního bufferu a $\vec{f}$ je velikost framebufferu.

Shader `calculateVtIndirectionBorderless.frag`

Tento fragmentový shader obsahuje funkci `calculateVtIndirectionBorderless` použitelnou při finálním vykreslování geometrie s virtuální texturou. Tato funkce implementuje výpočet souřadnic vzorku v textuře stránek na základě textury nepřímých odkazů a souřadnic vzorku v prostoru virtuální textury. Tato funkce nezvládá stránky s okraji a nelze ji tedy použít v případě, že požadujeme filtrování virtuálních textur.

Parametry této funkce jsou:

- `indirectionTexture` - Sampler textury nepřímých odkazů.
- `vtTexCoord` - Souřadnice vzorku virtuální textury.
- `vtTileCount` - Počet stránek na straně virtuální textury.
- `vtTileSizeFraction` - Velikost části textury stránek, kterou zabírá jedna stránka. Tuto hodnotu lze běžně získat podělením vlastnosti `tileTextureUsedSpaceSizeMultiplier` vlastností `tileTextureTileCount` použitého kompozitoru.

Shader `calculateVtIndirectionBordered.frag`

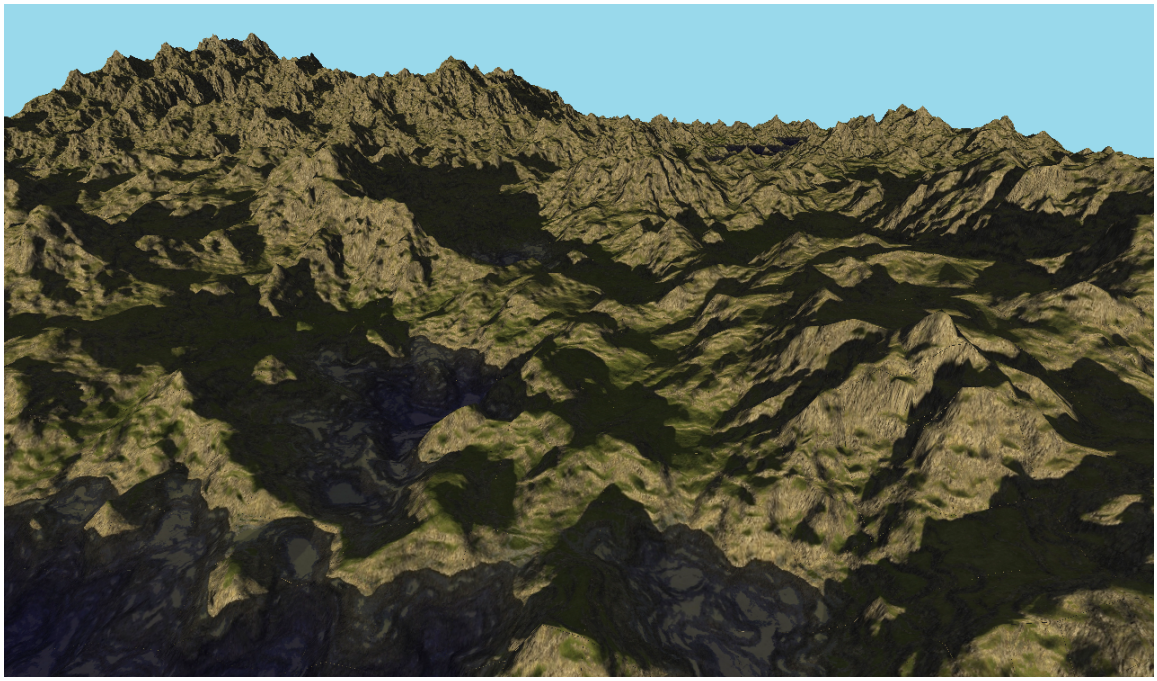
Tento fragmentový shader je velice podobný shaderu `calculateVtIndirectionBorderless.frag`, ovšem přidává podporu vzorkování textur se stránkami s okraji.

Parametry této funkce jsou:

- `indirectionTexture` - Sampler textury nepřímých odkazů.
- `vtTexCoord` - Souřadnice vzorku virtuální textury.
- `vtTileCount` - Počet stránek na straně virtuální textury.
- `vtTileSizeFraction` - Velikost části textury stránek, kterou zabírá jedna stránka. Tuto hodnotu lze běžně získat podělením vlastnosti `tileTextureUsedSpaceSizeMultiplier` vlastností `tileTextureTileCount` použitého kompozitoru.
- `vtTileOffset` - Odsazení stránek v textuře stránek kvůli okrajům těchto stránek. Tuto hodnotu lze získat přímo z vlastnosti `tileTextureTileOffset` kompozitoru.
- `vtTileBorderlessSize` - Poměr velikosti stránky a velikost stránky s přidanými okraji. Tuto hodnotu lze získat z vlastnosti `tileTextureTileBorderlessSizeFraction` kompozitoru.

5.2 Demonstrační aplikace

Nad knihovnou VortexVT byla mnou vytvořena jednoduchá demonstrační aplikace, která zobrazuje jeden model potexturovaný virtuální texturou a umožňuje pohybovat s kamerou.



Obrázek 5.1: Demonstrační aplikace `vortexvt-demo`

Výchozím demonstračním modelem je vygenerovaný terén, který má přes 150 000 trojúhelníků a který je potexturován texturou o velikosti 65536 × 65536 texelů. Celkem se tedy jedná o 16 GiB nekomprimovaných RGBA dat či 12 GiB nekomprimovaných RGB dat.

5.3 Stránkovací nástroj

Pro automatické rozřezávání velkých textur na stránky a generování stránek s nižší úrovní detailů byl vytvořen nástroj `vortexvt-tiler`. Tento nástroj bere na vstupu obrazový soubor či složku s více obrazovými soubory (vhodné v případě, že pro vytváření či modifikaci virtuální textury používáme nástroj, který sám zvládá práci s danou texturou ve formě stránek) a generuje obrazové soubory o zvolené velikosti a se všemi vhodnými úrovněmi detailů do zvolené výstupní složky. Parametry vstupu jsou dodávány pomocí parametrů příkazové řádky a parametry výstupu jsou zvoleny dle informačního souboru o virtuální textuře, který má stejný formát jako informační soubor použitý v případě poskytovatele stránek `SimpleDevilTileProvider` z modulu `vortexvt.tileprovider` a nachází se ve výstupní složce.[8]

5.3.1 Typické použití nástroje `vortexvt-tiler`

Pro jednoduché rozdělení jednoho zdrojového obrázku na větší množství stránek lze použít tento jednoduchý příkaz:

[8] Více o tomto souboru viz příloha B.

```
vortextvt-tiler --input=vstup.png --output=vystup
```

Za předpokladu, že `vystup` je složka obsahující soubor `info.json`, tak dojde k rozdělení vstupního obrázku `vstup.png` na patřičný počet stránek a současně dojde i k vygenerování všech vhodných úrovní detailů.

V případě, že vstupní data již jsou rozdělena na stránky, tak můžeme tento nástroj použít k převodu těchto stránek na stránky o jiné velikosti či k vygenerování úrovní detailů. K aktivaci módu stránkovaného vstupu je určen přepínač `--tiled`. K určení formátu názvů vstupních souborů je určen parametr `--filename-format` či `--ff`. Příklad spuštění aplikace `vortextvt-tiler` se stránkovacím vstupem je následující:

```
vortextvt-tiler --input=vstup --output=vystup \
--tiled --filename-format=x%02s-y%02s.png
```

Za předpokladu, že `vstup` je složka obsahující soubor `info.json` a `vstup` je složka obsahující vstupní stránky s formátem názvu podle parametru `--filename-format` (tj. například `x05-y15.png`), tak dojde ke spojení vstupních stránek a následnému rozdělení na výstupní stránky a vygenerování úrovní detailů.

Je vhodné poznamenat, že tato operace uchovává v paměti pouze minimální nutné množství paměti a nenahrává všechny vstupní stránky zároveň.

5.4 Interoperabilita s jinými programovacími jazyky

Ačkoliv je celá knihovna napsána v druhé verzi jazyka D, tak ji můžeme použít ze všech jazyků, které podporují volání nativního kódu. Jelikož je ovšem knihovna složena především z šablon, které nemají ekvivalentní reprezentaci v jiných jazycích, tak musíme tyto šablony instanciovat a vytvořit vhodné mezi-jazykové rozhraní a to právě v jazyce D2.

Jelikož je toto rozhraní závislé na konkrétních instancích šablon, které jsou závislé na kompilačních parametrech, tak nelze vytvořit generickou implementaci tohoto rozhraní a je nutné, aby si ji každý uživatel vytvořil sám. To přirozeně zvyšuje náročnost vývoje.

Následující případ ukazuje možný způsob vytvoření C rozhraní pro kompozitor virtuálních textur:

```
module crozhrani;

extern (C):

import vortexvt.compositor :
    Tile, Compositor;
import vortexvt.tileprovider :
    CachinTileProviderWrapper,
    SimpleDevilTileProvider;
import vortexvt.tilerequestprovider :
    FboTileRequestProvider;

// Je nutné vytvořit i rozhraní umožňující vytvoření tohoto
// poskytovatele stránek
alias TileProvider = CachingTileProviderWrapper!SimpleDevilTileProvider;
alias Comp = Compositor!TileProvider;

Comp* compCreate(
    TileProvider* tileProvider,
    uint tileTextureTileCount,
```

```
21      uint tileTextureUnit,
22      uint mappingTextureUnit,
23      GLenum internalFormat,
24      uint borderSize)
25  {
26      return new Comp(
27          tileProvider,
28          tileTextureTileCount,
29          tileTextureUnit,
30          mappingTextureUnit,
31          internalFormat,
32          borderSize);
33  }
34
35  // Metoda loadTiles je šablona parametrizovaná typem vstupního range.
36  // Musíme tedy stanovit, jaké konkrétní rozhraní zde bude použito.
37  // Jednou z možností je použít běžné pole a předat ukazatel
38  // na jeho začátek a délku tohoto pole.
39  void compLoadTiles(Comp* comp, Tile* tiles, size_t tilesLength)
40  {
41      comp.loadTiles(tiles[0 .. tilesLength]);
42  }
43
44  // Další možností je vytvoření funkce, která přímo použije vlastnost
45  // data námi používané instance struktury FboTileRequestProvider.
46  // Následující funkce předpokládá, že používáme pouze virtuální texturu
47  // s ID = 0
48  void compLoadTiles2(Comp* comp, FboTileRequestProvider* requestProvider)
49  {
50      comp.loadTiles(requestProvider.data.front);
51  }
52
53  // Ostatní metody kompozitoru lze implementovat podobně, jako
54  // ukazuje následující funkce
55  uint compTileTextureTileCount(Comp* comp)
56  {
57      return comp.tileTextureTileCount;
58  }
```

Hlavičkový soubor pro jazyk C, který nám umožní tyto funkce volat, pak může vypadat následovně:

```
1  #ifndef CROZHRANI_H
2  #define CROZHRANI_H
3
4  #include <stdint.h>
5  #include <GL/gl.h>
6
7  // Se vším budeme pracovat přes opaque ukazatele
8  struct Tile;
9  struct Comp;
10 struct TileProvider;
11 struct FboTileRequestProvider;
12
13 Comp* compCreate(
```

```
14      TileProvider* tileProvider,
15      uint32_t tileTextureTileCount,
16      uint32_t tileTextureUnit,
17      uint mappingTextureUnit,
18      GLenum internalFormat,
19      uint32_t borderSize);
20
21  void compLoadTiles(struct Comp* comp, Tile* tiles, size_t tilesLength);
22  void compLoadTiles(struct Comp* comp,
23      struct FboTileRequestProvider* requestProvider);
24  uint32_t compTileTextureTileCount(struct Comp* comp);
25
26  // atd.
27
28  #endif
```

Pro korektní běh programu v případě vzniku výjimek je samozřejmě nutné rozšířit rozhraní o zachytávání těchto výjimek a funkce pro získání informací o vzniklých chybách.

Kapitola 6

Měření výkonu knihovny VortexVT

Pro zjištění vlivu knihovny VortexVT na výkon grafických aplikací byly použity dva počítače s následujícím hardwarovým a softwarovým vybavením:

- PC1 hardware:
  - CPU: Intel Core i5-4670K 3.4 GHz
  - RAM: 8GiB DDR3 1.6 GHz
  - GPU: ATi Radeon HD 3850, 512 MB VRAM
- PC1 software:
  - OS1: Microsoft Windows 7
    - Ovladač GPU: Catalyst Display Driver 8.97.100.11
  - OS2: Arch Linux, jádro 3.15.7-1-ck
    - Ovladač GPU: xf86-video-ati 1:7.4.0-3
- PC2 hardware:
  - CPU: Intel Core i5-3230M 2.6 GHz
  - RAM: 4GiB DDR3 1.6 GHz
  - GPU: nVidia 730M
- PC2 software:
  - OS1: Microsoft Windows 8
    - Ovladač GPU: GeForce 340.52
  - OS2: Arch Linux, jádro 3.15.6-1-ck
    - Ovladač GPU: nvidia-ck 340.24-2

Měřeno bylo trvání následujících operací:

- Renderování požadavků na stránky do zpětnovazebního bufferu
- Agregace požadavků ze zpětnovazebního bufferu

- Kompozice virtuální textury a textury nepřímých odkazů
- Finální renderování scény
- Celkový čas snímku

Je třeba poznamenat, že pro kompilaci demonstrační aplikace byly použity dva různé kompilátory, konkrétně DMD na systémech Microsoft Windows[1] a LDC na systémech GNU/Linux[2].

Tento rozdíl mezi kompilátory znemožňuje porovnávat výkon knihovny mezi různými operačními systémy, což ovšem není předmětem těchto testů. Očekávaným výsledkem je tedy vyšší výkon na systémech GNU/Linux z důvodu lepšího generování kódu kompilátorem LDC.

Měření bylo provedeno dvakrát, poprvé s virtuální texturou o velikosti 256 × 256 texelů a podruhé s texturou o velikosti 65536 × 65536 texelů. Obě textury měly stránky o velikosti 128 × 128 texelů. Ve všech případech byla kamera ponechána ve výchozí pozici a aplikace byla ponechána v tomto stavu po dobu přibližně jedné minuty.

Původně byla měření prováděna dodatečně ještě s použitím mnohem jednodušší geometrie scény (konkrétně se jednalo o dva trojúhelníky tvořící čtverec), ovšem hned po několika prvních měřeních se ukázalo, že složitost geometrie scény má prakticky nulový vliv na výkon. Z tohoto důvodu tato měření nebyla dokončena a již naměřené hodnoty nebyly nijak použity.

		Renderování požadavků	Agregace požadavků	Kompozice	Finální renderování	Celkový čas snímku
256 × 256	PC1, OS1	0.15	4.06	1.98	0.05	6.4
	PC2, OS1	0.16	2.88	2.1	0.03	5.56
	PC1, OS2	2.43	0.44	0.45	0.02	16.7
	PC2, OS2	0.23	0.9	0.76	0.37	16.67
65536 × 65536	PC1, OS1	0.24	4.31	7.12	0.27	12.37
	PC2, OS1	0.18	4.79	8.21	0.02	13.5
	PC1, OS2	2.78	0.79	1.52	0.03	16.66
	PC2, OS2	0.27	1.46	3.23	0.05	16.72

Tabulka 6.1: Výsledky měření výkonu knihovny (všechny časy jsou v ms)

Výsledky samotného měření jsou v tabulce 6.1. Z těchto hodnot je zjevné, že v případě použití kompilátoru DMD agregace požadavků na stránky a kompozice zcela dominují čas snímku a v případě textury o velikosti 65536 × 65536 aplikace stěží dosahuje šedesáti snímků za sekundu.

[1] Kompilátor DMD byl zvolen z důvodu nadměrné složitosti použití kompilátoru LDC.

[2] Kompilátor LDC byl zvolen z důvodu chyby v kompiltároru DMD, která znemožňuje předávat struktury větší než 8 B do funkcí s volací konvencí jazyka C, což znemožňuje využít knihovnu SFML pro vytvoření aplikačního okna.

V případě kompilátoru LDC je situace mnohem lepší (což je očekávané)[3], ale i přesto agregace požadavků a kompozice trvá v nejhorším případě přibližně třetinu času snímku při šedesáti snímcích za sekundu.

Při nasazení knihovny VortexVT v běžné aplikaci by tedy neměl být problém dosáhnout interaktivního počtu snímku za sekundu (tj. ideálně alespoň šedesáti), ovšem vyšší časová náročnost agregace požadavků a kompozice stránek by mohla působit problémy, obzvláště v případě velikých virtuálních textur.

[3]Letmý pohled na tabulku výsledků by mohl vyvolat představu, že výkon této verze je naopak nižší, a to z důvodu vyššího celkového času snímku. Tento čas ovšem obsahuje i dobu čekání, které je způsobeno pevnou rychlostí aplikace (tj. šedesát snímků za sekundu), která je vynucena nastavením operačního systému.

Kapitola 7

Možná rozšíření či vylepšení knihovny VortexVT

Knihovna VortexVT rozhodně umožňuje implementovat solidní systém virtuálního texturování, ale i přesto by bylo vhodné ji rozšířit o dodatečnou funkcionalitu, která buď zvyšuje pohodlí vývojářů a cílových uživatelů, zvyšuje výkon či umožňuje ji efektivněji využít v moderním grafickém prostředí. Některá taková rozšíření jsou:

- Poskytovatel stránek, který nenahrává stránky z jednotlivých souborů (podobně jako `SimpleDevilTileProvider`), ale pouze z jednoho velkého souboru, který obsahuje všechny stránky. Toto řešení má výhodu v tom, že není nutné při přenášení aplikace přenášet desítky či stovky tisíc souborů, jak je tomu v současnosti u demonstrační aplikace.
- Zlepšení algoritmu aktualizace textury nepřímých odkazů v kompozitoru virtuálních textur. Dodatečným testováním jsem totiž zjistil, že právě tato operace je (obzvlášť v klidovém stavu) poměrně časově náročná, jelikož prakticky rekonstruuje celou texturu nepřímých odkazů během každého snímku, což může být problematické obzvlášť pro virtuální textury s velikým počtem stránek.
- Prozkoumat možnosti asynchronního kompozitoru či agregátoru výsledků ze zpětnovazebního bufferu, ať už za použití OpenGL Buffer Objects a DMA přenosu dat či opravdu asynchronních rutin těchto komponent. Takto by bylo možné omezit výkonnostní vliv těchto operací na celou aplikaci.
- Použít rošíření `ARB_sparse_texture`, pokud jej aktuální hardware podporuje. Toto rozšíření by mělo odstranit spoustu neduhů virtuálního texturování, přinést výkonnostní výhody a zjednodušit shadery.

Kapitola 8

Závěr

V této práci bylo navrhnuto několik způsobů implementace techniky virtuálního texturování. Bylo také poukázáno na několik problémů, které jsou pro tuto techniku typické, a bylo navrhnuto několik způsobů, jak tyto problémy řešit. Dalším výsledkem této práce je programová knihovna VortexVT, která usnadňuje zapojení virtuálního texturování do nově vznikajících či již existujících grafických systémů a poskytuje široké možnosti přizpůsobení uživatelem.

Praxí jsem také zjistil, že ačkoliv je základní myšlenka techniky virtuálního texturování velice jednoduchá, tak při implementaci lze narazit na spoustu drobných i velkých problémů. Za zmínku stojí například prolínání nežádoucích barev do okrajů vykreslovaných stránek při zapnutém bilineárním filtrování bez použití dodatečných okrajů stránek.

Spousta těchto problémů je způsobena tím, že grafický procesor pracuje s jistými předpoklady, které my nejsme schopni splnitl. Kupříkladu vzorkování textur je považováno za poměrně vyřešenou záležitost, ovšem pouze v případě, že je splněn předpoklad spojitosti textury, což v případě virtuálních textur jednoduše neplatí a proto musíme tento předpoklad obcházet více či méně složitými způsoby. Možná i z tohoto důvodu jsou techniky jako právě virtuální texturování poměrně neprobádanou oblastí počítačové grafiky.

V poslední době naštěstí vidíme naopak posun k větší flexibilitě v této oblasti – vždyť kupříkladu právě virtuální texturování není ve své podstatě nic jiného, než prostá virtuální adresace paměti, kterou grafické procesory disponují již delší dobu. Technologie jako DirectX tiled resources či OpenGL rozšíření `ARB_sparse_texture`[1] slibují právě onu chybějící flexibilitu.

Naštěstí lze i přese všechny zmíněné překážky vytvořit obstojně fungující systém, který ve většině případů vypadá dobře, zvládá interaktivní počet snímků za sekundu a běží přitom na širokém spektru hardwaru – přeci jen jsme prakticky nepoužili žádné příliš pokročilé funkce grafických procesorů, vystačili jsme si s fragment shadery a mimo-obrazovkovými framebuffery.

A ačkoliv to zatím nevypadá, že by se techniky jako virtuální texturování příliš rozmáhaly (prakticky jediný komerčně dostupný herní engine s podporou virtuálního texturování je id Tech 5 od id Software), možná uvidíme brzy něco podobného, ovšem mnohem ambicióznějšího – renderování voxelových octree je čím dál tím oblíbenější směr počítačové grafiky, který se od virtuálního texturování teoreticky liší pouze přidaným třetím rozměrem a jedná se tedy o logický krok kupředu.

[1] Viz http://www.opengl.org/registry/specs/ARB/sparse_texture.txt.

Literatura

[1] Barret, S.: Sparse Virtual Textures [online].
http://silverspaceship.com/src/svt/, 2008 [cit. 2014-05-20].

[2] Mittring, M.; Crytek GmbH: Advanced Virtual Texture Topics. In *ACM SIGGRAPH 2008 Games*, SIGGRAPH '08, New York, NY, USA: ACM, 2008, s. 23–51, doi:10.1145/1404435.1404438.
URL http://doi.acm.org/10.1145/1404435.1404438

[3] Obert, J.; van Waveren, J. M. P.; Sellers, G.: Virtual Texturing in Software and Hardware. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, New York, NY, USA: ACM, 2012, ISBN 978-1-4503-1678-1, s. 5:1–5:29, doi:10.1145/2343483.2343488.
URL http://doi.acm.org/10.1145/2343483.2343488

[4] van Waveren, J. M. P.: Software Virtual Textures.
http://mrelusive.com/publications/papers/Software-Virtual-Textures.pdf, 2013 [cit. 2014-05-20].

Příloha A

Obsah DVD

DVD dodané s prací obsahuje, mimo jiné, následující:

- Knihovna VortexVT – Zdrojové texty v jazyce D2 a shadery v jazyce GLSL patřící výhradně ke knihovně jsou umístěny ve složce vortexvt-lib
- Demonstrační aplikace – Veškere zdrojové texty v jazyce D2, shadery v jazyce GLSL, požadované knihovny, spustitelná aplikace a demonstrační model s demonstrační texturou (cca. 2,4 GiB komprimovaných JPEG souborů) jsou umístěny ve složce vortexvt-demo
- Stránkovací nástroj – zdrojové texty v jazyce D2, požadované knihovny a spustitelná aplikace jsou ve složce vortexvt-tiler

Všechny tyto složky navíc obsahují soubor `package.json` pro sestavení projektu pomocí systému DUB.[1]

[1] Viz http://code.dlang.org/.

Příloha B

Formát souboru `info.json`

`info.json` je informační soubor poskytující informace o virtuální textuře pro poskytovatel stránek `SimpleDevilTileProvider` a pro stránkovací nástroj. Soubor má jednoduchý JSON formát tvořený z jednoho JSON objektu se šesti členy. Ty jsou následující:

- `tileSize` – Vertikální i horizontální velikost jedné stránky.
- `tileCount` – Vertikální i horizontální počet stránek ve virtuální textuře
- `lodCount` – Počet úrovní detailů. Tato hodnota by v současnosti vždy měla být rovna $log_2\left(tileCount\right) + 1$
- `tileFormat` – Formát dat jednotlivých stránek. Možné hodnoty jsou:
  - `rgb`
  - `rgba`
  - `bgr`
  - `bgra`
  - `luminance`

  Nejčastější hodnotou bude zřejmě `rgb`.
- `tileType` – Datový typ barevných komponent v obrázku. Možné hodnoty jsou:
  - `signedByte`
  - `unsignedByte`
  - `signedShort`
  - `unsignedShort`
  - `signedInt`
  - `unsignedInt`

  Nejčastější hodnotou bude zřejmě `unsignedByte`.
- `filenameFormat` – Formát názvu souborů se stránkami. Formát využívá systaxe modulu `std.format` jazyka D2.[1] První parametr formátu představuje úroveň detailů,

[1] Viz http://dlang.org/phobos/std_format.html.

druhý parametr představuje horizontální souřadnici stránky a třetí parametr představuje vertikální souřadnici stránky.

Příklad souboru info.json pro přiloženou demonstrační virtuální texturu:

```
1 {
2     "tileSize" : 128,
3     "tileCount" : 512,
4     "lodCount" : 10,
5     "tileFormat" : "rgb",
6     "tileType" : "unsignedByte",
7     "filenameFormat" : "lod%1$02s-x%2$04s-y%3$04s.jpg"
8 }
```

Příloha C

Přehled použití knihovny VortexVT

Při použití knihovny VortexVT je nejprve nutné instanciovat tři komponenty:

- `FboTileRequestProvider`
- `Compositor`
- Libovolný poskytovatel stránek. Ten může být například poskládán ze struktur `SimpleDevilTileProvider` a `CachingTileProviderWrapper` pro vytvoření jednoduchého posktovatele stránek s dodatečnou cache v operační paměti.

Dále je nutné vytvořit shadery, ty mohou být založeny na GLSL funkcích dodaných s knihovnou. Tyto funkce mají několik klíčových parametrů, jejichž účel je popsán přímo v shaderech.

Pokud chceme mít kontrolu nad filtrováním, tak je vhodné vytvořit OpenGL sampler a připojit jej k textuře stránek. Textura nepřímých odkazů musí vždy být bez filtrování.

Pro zjištění stránek, které musíme nahrát, použijeme již vytvořený `FboTileRequestProvider`. Aktivujeme pro sběr souřadnic stránek určený shader a zavoláme na vytvořeném `FboTileRequestProvider` metodu `prepareTileRequestRendering`. Poté vykreslíme scénu a následně zavoláme metodu `finishTileRequestRendering`. Nyní můžeme přistoupit k vlastnosti `data` naší instance `FboTileRequestProvider` a získat tak dvourozměrné pole, jehož první úroveň reprezentuje ID virtuální textury a jehož druhá úroveň je seznam potřebných stránek.

Tento seznam můžeme předat do metody `loadTiles` našeho objektu `Compositor`. To způsobí nahrání stránek do textury strának a vhodnou úpravu textury nepřímých odkazů.

V případě použití většího počtu virtuálních textur jednoduše vytvoříme více instancí kompozitoru a poskytovatele stránek a požadavky na stránky do kompozitorů rozdělíme dle první úrovně pole z vlastnosti `data` poskytovatele požadavků.

Poté už můžeme konečně aktivovat shader pro finální vykreslení scény a scénu vykreslit.