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

DEMONSTRAČNÍ APLIKACE PRO DISPLACEMENT MAPPING A ANALÝZU VÝKONNOSTI OPENGL PIPELINE

DEMONSTRATION APPLICATION FOR DISPLACEMENT MAPPING AND OPENGL PERFORMANCE ANALYSIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. MARTIN KUČERŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. JAN PEČIVA, Ph.D.

BRNO 2014

Abstrakt

Tato diplomová práce se zabývá technikou displacement mappingu a analýzou výkonosti OpenGL pipeline s využitím volně dostupné proprietární knihovny GPUperfAPI, vytvořené společností AMD. V první části stručně popisuje OpenGL pipeline, její části jak hardwarové, tak softwarové. Následuje popis samotného displacement mappingu a jeho výhody, jemu příbuzných metod, a zmiňuje některé aplikace využívající nebo demonstrující displacement mapping. V další části popisuje rozhraní GPUperfAPI, implementační otázky jednotlivých metod a generování displacement map. V závěrečné části popisuje implementovanou aplikaci, její povahu, a účel.

Abstract

This thesis describes displacement mapping and performance analysis of OpenGL pipeline, with use of proprietary freeware library GPUperfAPI, made by AMD. First part briefly describes OpenGL pipeline, its parts, and then its counterparts in HW. Then its describes displacement mapping and similiar mapping methods, advantages of displacement mapping, and mentions few applications using displacement mapping.Next part describes interface of GPUperfAPI, implementation details of described methods and tools for generating of displacement maps. Last part describes nature of demonstration appliaction.

Klíčová slova

Teselace, Teselační shader, Displacement mapping, GPUperfAPI, OpenGL, OpenGL pipeline

Keywords

Tessellation, Tessellation shader, Displacement mapping, GPUperfAPI, OpenGL, OpenGL pipeline

Citace

Martin Kučerňák: Demonstrační aplikace pro displacement mapping a analýzu výkonnosti OpenGL pipeline, diplomová práce, Brno, FIT VUT v Brně, 2014

Demonstrační aplikace pro displacement mapping a analýzu výkonnosti OpenGL pipeline

Prohlášení

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

....
Martin Kučerňák
2. června 2014

Poděkování

Rád bych tímto poděkoval svému vedoucímu, Ing. Janu Pečivovi, Ph.D., jeho kolegům, a všem kteří mě v této práci podporovali a drželi mi palce.

Obsah

Kapitola 1

Úvod

Ačkoliv existuje už velké množství způsobů a technik jak zlepšit vzhled vykreslovaného objektu, ať jde o bump mapping, parallax mapping, parallax occlusion mapping, používání specular textury, odlesky, tak žádná z těchto metod nemění geometrii objektu. To u některých objektů není problém, u jiných už ano. Displacement mapping, ačkoliv jde o poměrně obecný pojem, ve verzi zde prezentované umožňuje modifikovat geometrii objektu. Tyto zmiňované efekty je ale třeba vykreslovat pokud možno v reálném čase, proto se druhá část práce zabývá analýzou výkonnosti OpenGL. K této analýze slouží knihovna GPUperfAPI od společnosti AMD, která umožňuje sbírat data přímo z grafické karty. Také se zabývá knihovnou OpenSceneGraph která je využita pro vykreslování.

Kapitola 2

Popis OpenGL pipeline

OpenGL vykreslovací pipeline je sekvence kroků, které provádí OpenGL při vykreslování objektů [5]. Její blokové znázornění je na obrázku 2.2, a dělí se na tyto kroky:

- Vertex Shader - je krok zpracování vrcholů vykreslovaného objektu. Vstupem je jeden vrchol (a přidružené hodnoty, jako jsou jeho normály, texturovací souřadnice, a další), výstupem také. Nemůže tedy vytvářet nové vrcholy, může ale vytvářet nové parametry, které se pak společně s aktuálně zpracovávaným vrcholem předají dále.

- Tessellation - je volitelný krok, ve kterém se, za pomoci hardwarové teselační jednotky grafické karty, aktuální polygon rozdělí na více menších dílů.

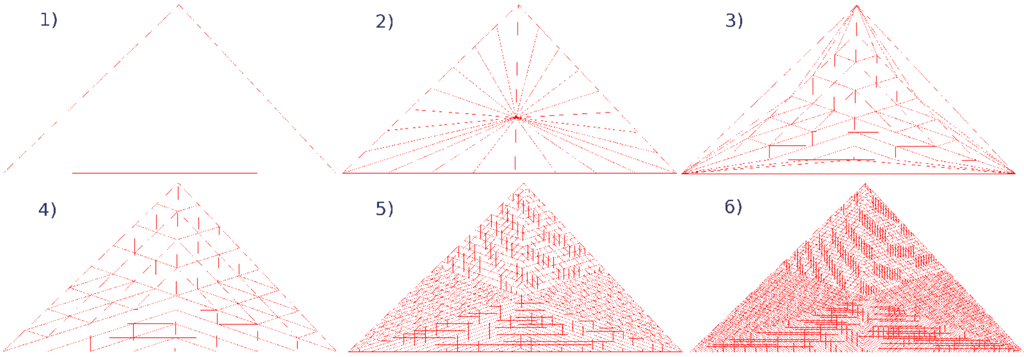
  - Tessellation Control Shader[1] - nastavuje teselační parametry pro aktuální polygon. Nastavované parametry jsou stupně vnitřní a vnější teselace. Podobně jako geometry shader vidí celý vstupní trojúhelník a může provádět podobné operace, jako je výpočet matice pro transformaci vektorů a vertexů z tangent space do object space a naopak. Při teselování mohou být podobné možnosti jako má geometry shader výhodou i v otázce výkonu, protože by vzrostl počet invokací řádové geometry shaderu, v případě maximálního stupně teselace, až 6000krát. Při testování jednoho trojúhelníku s maximálním stupněm teselace byl počet trojúhelníků vstupujících do geometry shader 6144.

    TCS je také vhodné místo pro provádění back-face cullingu - ten by se normálně prováděl až po teselační fázi, a tedy až po vypočtení všech bodů nově vzniklých teselací (tzn. po průchodu Tessellation Evaluation Shaderem). Tak by se zbytečně provádělo velké množství výpočtu, jako displacement mapping s množstvím přístupů do paměti, i samotný výpočet nových bodů - jeden nový vertex potřebuje na jednu hodnotu 14 instrukcí, a tento roste s každou další předávanou hodnotou, jako jsou normály, texturovací souřadnice, a jiné.

  - Tessellator - nejde o programovatelnou jednotku, je to fixní jednotka, která na základě parametrů nastavených v TCS vypočítá vstupními parametry určený počet barycentrických souřadnic pro nové body uvnitř aktuálního trojúhelníku (nebo čtyřúhelníku, záleží na tom, jaký typ polygonů se právě zpracovává), včetně krajních souřadnic představujících původní body. Vrcholy původního vstupního trojúhelníku jsou beze změny předány do Tessellation Evaluation Shaderu.

[1] V dalším textu bude označován zkratkou TCS

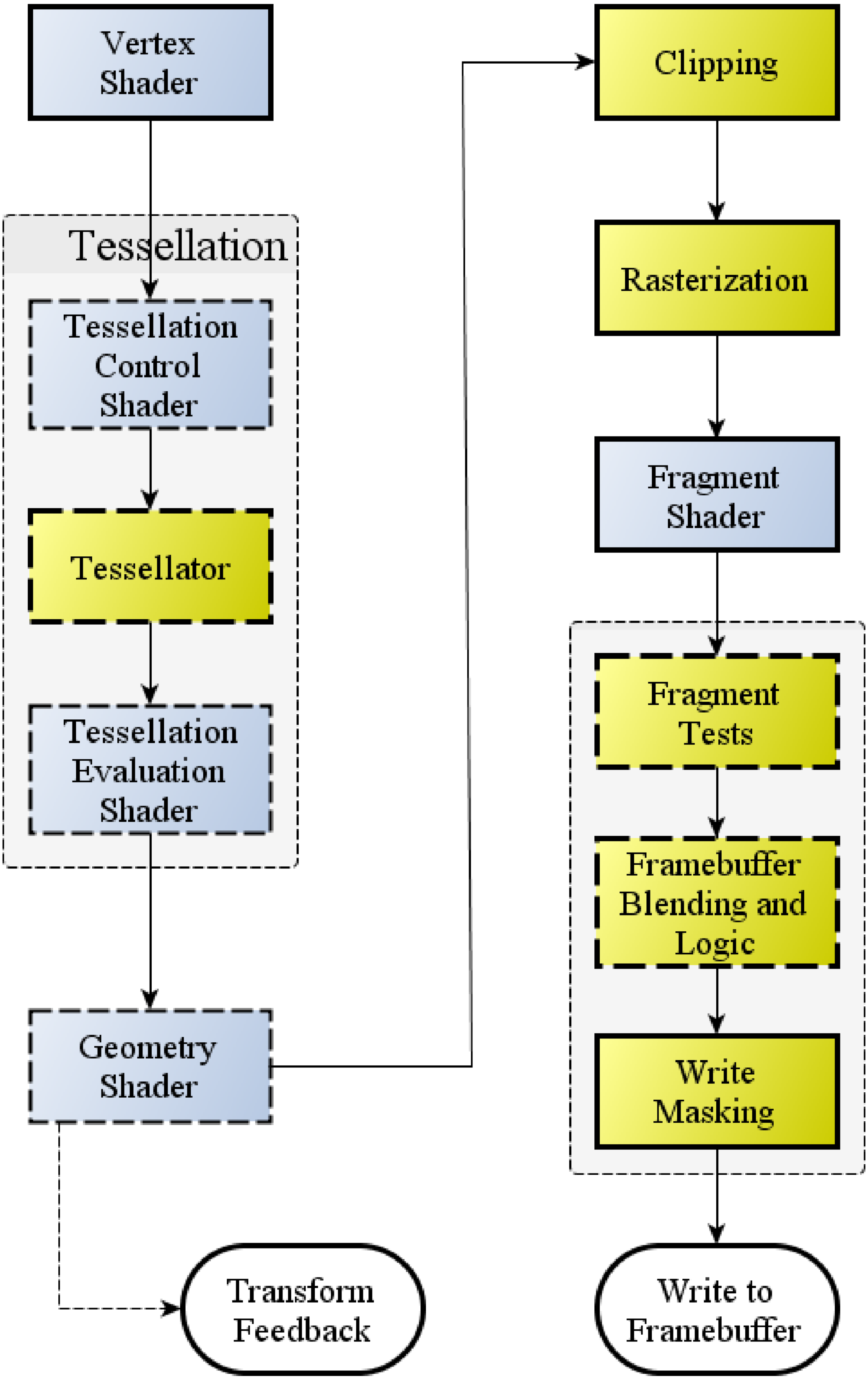
– Tessellation Evaluation Shader[2] - přepočítává barycentrické souřadnice (vygenerované teselátorem) nového vertexu na skutečné souřadnice v prostoru. Má podobné možnosti jako vertex shader. Vidí všechny body původního trojúhelníku, jak byly zpracovány v TCS, ale ne už barycentrické souřadnice jiných bodů v aktuálním trojúhelníku.



Obrázek 2.1: Příklad teselace jednoho trojúhelníku: 1) bez teselace 2) vnitřní/vnější teselace 1/8 3) vnitřní/vnější teselace 8/1 4) celková teselace 8 5) celková teselace 32 6) celková teselace 64 (maximum)

- Geometry Shader - vstupem je primitivum (může to být bod, linka, nebo polygon), které se zpracovává jako celek. Výstupem je také primitivum, které je možné měnit, jak přidáním atributů (podobně jako ve vertex shaderu), tak přidáním celých vrcholů.
- Transform Feedback - je možnost zapsat výstup geometry shader do buffer objectu, se kterým lze dále pracovat, a využít ho opakovaně při dalších vykreslovacích operacích.
- Primitive Assembly (Clipping, Culling) - v této části probíhá předpříprava primitiv na rasterizaci, tato příprava se skládá z dělení primitiv, které nejsou vidět celá, a odstranění těch která nejsou vidět (například jsou mimo obrazovku, za kamerou, mimo view frustum, nebo je povoleno face culling a polygon je odvrácený od kamery).
- Rasterization - primitiva, která prošla fází primitive assembly, jsou zde zpracovány na fragmenty, a předány fragment shaderu. Fragment je soustava parametrů potřebných pro budoucí pixel (hodnota normály, texturovací souřadnice, pozice na obrazovce, pozice ve scéně).
- Fragment Shader - je krok zpracování jednotlivých fragmentů, ty jsou převáděna na hodnoty daného pixelu pro jednotlivé buffery, jako je framebuffer (barva pixelu), z-buffer (hloubka pixelu ve scéně).
- Fragment Tests, Framebuffer Blending and Logic, Write Masking - fáze kontroly a úprav výstupního pixelu, určuje se, zda se má zapsat do framebufferu (zda není zakryt jiným pixelem, nebo není v části primitiva, které leželo částečně mimo obrazovku), pokud je průhledný, tak se v případě, že se má zobrazit, zkombinuje jeho barva s již zapsaným pixelem. Pokud není pixel zahozen, tak se zapíše do framebufferu, a jeho hloubka do z-bufferu.

[2] V dalším textu bude označován zkratkou TES



Obrázek 2.2: Znázornění vykreslovací OpenGL pipeline

Kapitola 3

Historie displacement mappingu a podobných metod

3.1 Teselátor

Předchůdce dnešní hardwarové teselace se objevil v roce 2001 na kartách řady Radeon 8500, od tehdejší společnosti ATI, jako technologie TruForm[3]. Ta spočívala v teselaci vykreslovaných trojúhelníků a posunu vnitřních bodů tak, aby ležely na pomyslné zakřivené rovině procházející vrcholy původního trojúhelníku a zakřivené podle směru jejich normálových vektorů. Výsledkem byl méně hranatý povrch. Tato technologie byla přijata do OpenGL

Obrázek 3.1: Ukázka technologie TruForm (zdroj: http://wololo.net/2013/05/09/the-photo-realism-challenge-polygons/)

jako rozšíření ATI_pn_triangles[1], ale nikdy nebyla standardizována v DirectX, a NVidia nenabízela odpovídající alternativu (kromě Quintic-RT[1] patches, ale ty nebyly využity

[1]http://www.opengl.org/registry/specs/ATI/pn_triangles.txt

snad v žádném programu). Důsledek, společně s některými nedostatky, které TruForm měl, jako zakulacení i objektů kde byly hrany úmyslem, byla podpora v nízkém počtu aplikací, a ATI proto v pozdějších kartách (od karet řady x700 a x800 není uváděn jako podporovaná technologie) od podpory této technologie upustilo.

Před příchodem DirectX 11 se hardwarová teselace objevila v grafickém čipu Xenos (z herní konzole Xbox 360)[1], a od něj odvozené rodiny čipů R600 (například Radeon HD2900). Tato teselace se dočkala podpory v OpenGL i DirectX formou rozšíření. Později se ukázalo, že tato implementace není kompatibilní s tou, která byla použita v DirectX 11 a OpenGL, nelze tedy použít teselátor skrze tato API a je třeba využít původní rozšíření. Naštěstí ale současné grafické karty ATI (s podporou DirectX 11) podporují přístup k teselátoru přes toto rozšíření, takže je možné aplikace jej využívající stále používat.

Od čipů GF100 u Nvidie (GeForce GT280) a R700 u AMD (Radeon HD58xx) je přítomna podpora DirectX11/OpenGL 4.0, a s ní jejich implementace teselačních shaderů.

3.2 Další podobné metody

Obohacením vzhledu objektu o dojem plastičnosti nebo prostorovosti se zabývá více metod, než pouze displacement mapping. Zde jsou krátce uvedeny některé z nich.

3.2.1 Bump mapping

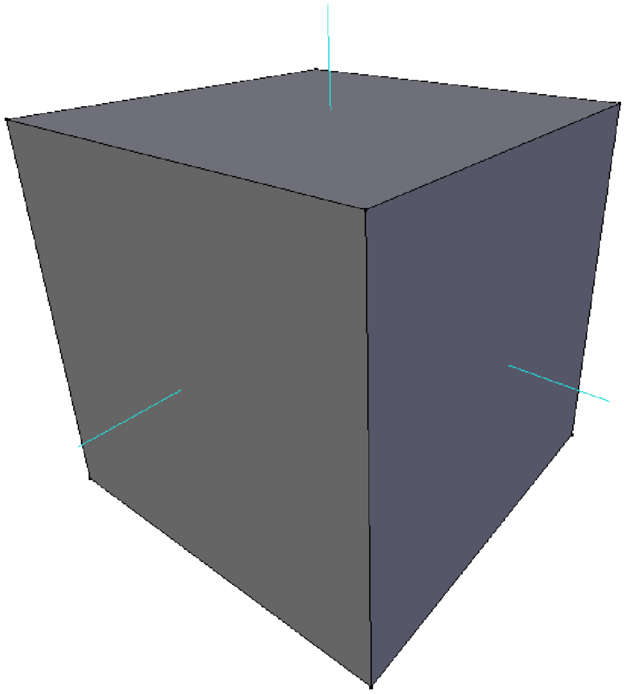
Bump mapping je metoda navržená J. F. Blinnem[7] v roce 1978, ve své původní podobě spočívala v úpravě normály na základě výškové hodnoty získané z textury. Narozdíl od Normal mappingu byla tato hodnota jednorozměrná, podobně jako u Displacement mappingu. Samotná hodnota ovšem neumožňuje určit zamýšlené natočení povrchu, a bylo nutné získat vzorky z okolních bodů a na základě tohoto okolí určit nový směr okolí. Důvodem pro tento postup bylo způsob generování map, kdy se využívaly hodnoty z-bufferu pro vykreslení detailního modelu. V roce 1984 byl představen Normal mapping v článku R. L. Cooka[10]. Oproti původnímu bump mappingu zde není třeba počítat směr normály z okolních bodů, ale normály jsou přímo uloženy v textuře, čímž se urychlí vykreslování. Později byly představeny metody jak jednoduše normálové mapy generovat.

Způsobů jak do textury uložit normály je více. Nejpoužívanější je ukládání v tangent space, které je sice na výpočet náročnější než object space, je ale o něco flexibilnější, nevadí mu například deformace způsobené animací modelu. Možné prostory jsou tyto:

- Tangent space - je prostor určený polygonem kterého se týká[2]. Tento polygon představuje vlastní bázi, takže v tangent space by normálových vektorů zobrazené na obrázku 3.2.1 směřovaly přímo nahoru, jejich xyz složky by tedy byly 0.0, 0.0, 1.0.

- Object space - je prostor, ve kterém je umístěný objekt. Normály v tomto prostoru mají stejnou bázi jako vrcholy, takže ty normály zobrazené na obrázku 3.2.1 v object space budou směřovat přesně tak jak jsou zobrazeny

- World space - uveden pro úplnost, protože bude zmíněn v jiné souvislosti dále. Všechny vektory a body v něm mají souřadnice a směrnice tak jak jsou umístěny ve scéně.



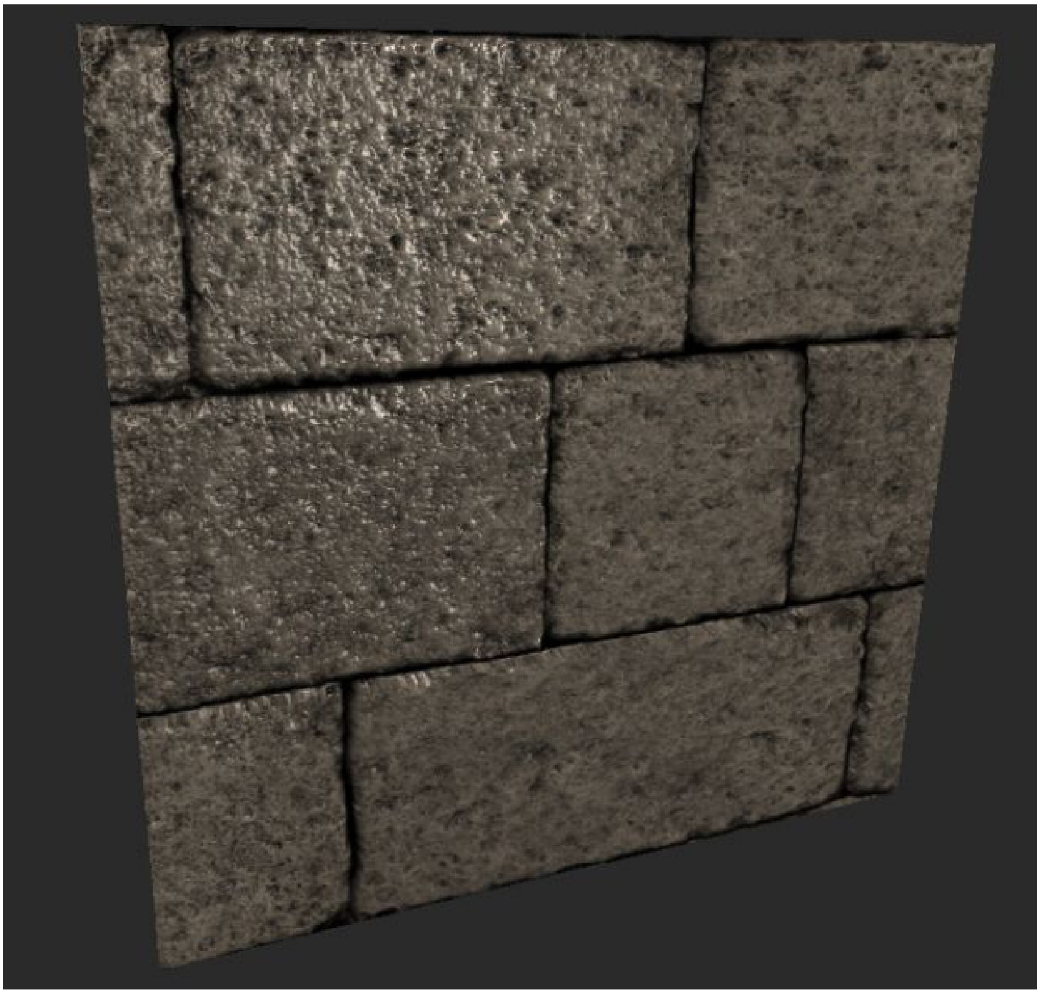
Obrázek 3.2: Směřování normál pro jednoduchý objekt

3.2.2 Parallax mapping

Paralax mapping je metoda, kterou v roce 2001 navrhl Tomomichi Kaneko[11]. Jejím cílem je navodit dojem plastičnosti objektu. Za tímto účelem si při vykreslení trojúhelníku spočítá úhel jeho normály a pohledového vektoru kamery, ten následně vynásobí koeficientem získaným z výškové mapy, a o tuto výslednou hodnotu posune texturovací souřadnice. Neumožňuje ale, aby se prvky textury překrývaly. Příklad využití displacement mappingu je na obrázku 5.1.

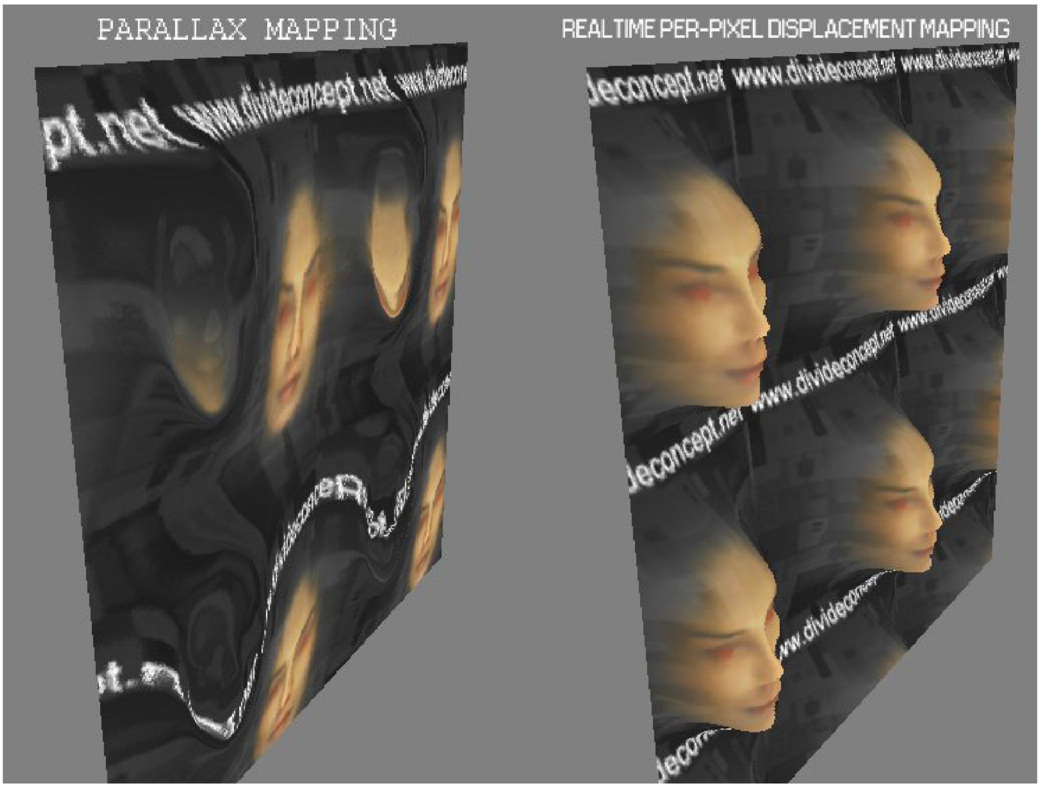
3.2.3 Per-pixel displacement mapping

Na per-pixel displacement mapping[4] lze pohlížet jako na inverzní parallax mapping. Stejně jako běžný displacement mapping používá výškovou mapu, ale s ní si dále spočítá vlastní displacement texturu, podle jejíchž hodnot upravuju texturovací souřadnice pro aktuální pixel. Oproti parallax umožňuje, aby se nějaký objekt zakreslený do textury překryl jiný, toho je docíleno tak, že se displacement mapa nepočítá tak, aby se posunul aktuální pixel, ale přiřazuje bodům textury body objektu. Nevýhodou je nutnost tuto mapu předpočítat pro každou změnu pozice a rotace vůči kameře. Jeho implementace a detaily popisovány nebudou, protože v praxi se tato metoda, z důvodu potřeby přepočítat procesorem nevyužívá. Technicky si je blízká s parallax occlusion mappingem, ten ale probíhá jen v grafické kartě.



Obrázek 3.3: Příklad aplikace bump mappingu na zeď (zdroj: http://www.moddb.com/ /mods/new-vision/images/bump-mapping)

Obrázek 3.4: Ukázka parallax mappingu (zdroj: tomshardware.it)



Obrázek 3.5: Příklad aplikace per-pixel displacement mappingu a jeho srovnání s parallax mappingem (zdroj: [12])

Kapitola 4

Teselační shadery

Z pohledu programátora se použití teselátoru skládá z Tessellation Control Shader (dále TCS; v DirectX se označuje jako Hull Shader) a Tessellation Evaluation Shader (dále TES; v DirectX se označuje jako Domain Shader). Také je potřeba použít na straně CPU pro vykreslování typ primitiv PATCHES, ať v OpenGL volání glDrawElements, nebo v případě využití OpenSceneGraphu při vytváření instance osg::Geometry při volání addPrimitiveSet. Samotnou geometrii, pokud jde například o trojúhelníkovou síť, lze použít beze změn i pro vykreslování bez využití teselátoru. Primitiva typu patches je označení pro typ primitiv zpracovávaných teselační částí pipeline, nemají pevně daný počet bodů, ten se nastavuje v kvalifikátoru layout. Patche mají v shaderu pak různý

4.1 Tessellation Control Shader

TCS je odpovědný za nastavení úrovní teselace, a jak už bylo zmíněno, má přístup ke všem vrcholům aktuálního patche. TCS v minimální podobě slouží k nastavení jednotlivých úrovní teselace, a počtu vrcholů daného patche. TCS pro použité teselátoru není povinný, v případě jeho nepřítomnosti se pro nastavení teselátoru, tj počet vrcholů v patchi a úrovně teselace, použijí přednastavené hodnoty. Ty lze změnit pomocí volání glPatchParameter na straně CPU.

Počet vrcholů se nastavuje kvalifikátorem layout

```
layout(vertices = 3) out;
```

kde *vertices* je počet vrcholů daného patche. Jak naznačuje řetězec *out*, jde o označení určené pro výstup - TCS funguje tak, že zpracovává příchozí vertexy, a po naplnění požadovaného počtu pošle celý patch dále do teselačních jednotek. V případě že po zpracování všech vertexů zpracovávaného objektu zůstane nedokončený patch (za nedokončený patch je považován takový, pro nějž zbývá menší počet vstupních vertexů, než bylo nastaveno do proměnné *vertices* v kvantifikátoru layout), je zahozen. Dalším důležitým prvkem TCS je hodnota gl_InvocationID. Tu OpenGL vytváří automaticky, není ji tedy třeba ručně definovat, a obsahuje pořadový index právě zpracovávaného vertex v aktuálně zpracovávaném patchi.

Dále je potřeba předat pro další zpracování do TES data (souřadnice vertexů, texturovací souřadnice, normály, a další). Zde, pokud není důvod pro nějaké další úpravy, stačí použít konstrukci podobou této

```
tcPosition[gl_InvocationID] = vPosition[gl_InvocationID];
```

Pro zpřehlednění kódu může být užitečné nahradit gl_InvocationID pomocí *#define* za jiný, kratší řetězec, například *ID*. Ten bude nadále používán v dalším textu.

Pak už jen nutné nastavit úrovně teselace. Ty se nastavují do proměnných gl_TessLevelInner a gl_TessLevelOuter. Tyto proměnné jsou opět vytvářené OpenGL, jsou definovaná jako pole, a obsahují úrovně teselací pro aktuální patch. Jejich hodnoty postačí nastavit jen v jedné invokaci TCS pro daný patch, takže může být užitečné je nastavovat jen například v průběhu zpracování prvního vertexu aktuálního patche (to lze zajistit kontrolou ID na hodnotu 0), zejména pokud se místo plošného nastavení úrovní používá nějaká forma jeho výpočtu.

Velikost polí gl_TessLevelInner a gl_TessLevelOuter se liší podle toho jaký typ patche se zpracovává. Přesnější určení typu patche se nastavuje v TES, který bude popsán dále. Korektní označení těchto patchů je abstraktní patch, protože teselátor nezpracovává vrcholy které prošly skrz TCS pro daný patch, pouze na základě informace o úrovních teselace a typu abstraktního patche vygeneruje relativní souřadnice určující polohu uvnitř patche. Rozdělení úrovní teselace na vnitřní a vnější je poměrně užitečná vlastnost, kterou lze využít například k navazování objektů s různými stupni teselace, jak je ukázáno v jednom z připravených příkladů, popsaném v jedné z pozdějších kapitol.

Typy patchů jsou následující:

4.1.1 Patche typu triangles

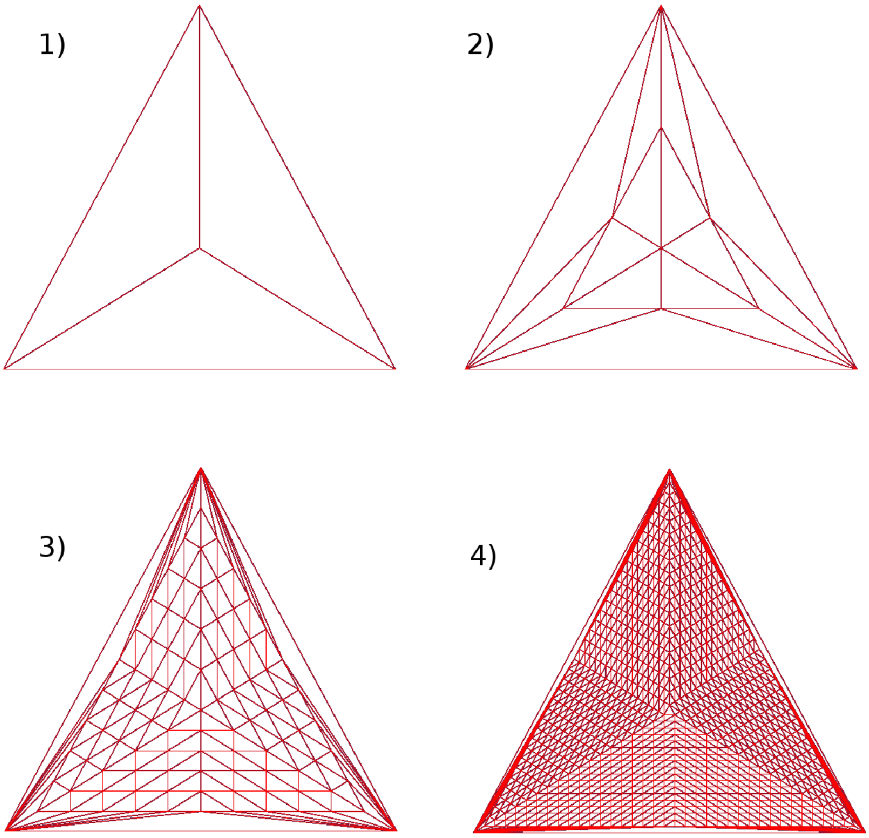
Protože trojúhelníky jsou nejrozšířenější reprezentace 3D modelů určených pro vykreslování v reálném čase na běžných grafických kartách, lze předpokládat, že tento typ patchů bude nejrozšířenější, protože lze bez dalších úprav takovýto model snadno vykreslit jak s teselací, tak i bez ní.

Pro tento typ patchů mají pole pro nastavení úrovně teselace velikost 1 prvek pro vnitřní teselaci, a 3 prvky pro vnější teselaci.

Vnitřní teselace se týká bodů uvnitř patche, určuje tedy kolik nových bodů vznikne uvnitř patche. Body uvnitř patche se rozumí takové, které neleží na spojnici mezi počátečními vrcholy patche. Příklad různých stupní vnitřní teselace jde vidět na obrázku 4.1, vnější teselace je nastavena na hodnotu 1. Při úrovni teselace 1 je sice patch nebo jeho část teselován(a), ale nevzniknou tak žádné nové vrcholy které by už nebyly v původním patchi. Na tomto obrázku je znázorněn jeden teselovaný trojúhelník. Bod s číslem 1) na tomto obrázku je znázornění stupně vnitřní teselace 2, v patchi tak vznikl jeden nový vrchol, kterým je původní trojúhelník rozdělen na tři. Při vyšších stupních nových obrázku vnikne již více, bod 2) znázorňuje vnitřní teselaci úrovně 8, bod 3) vnitřní teselaci stupně 12, a bod 4) vnitřní teselaci úrovně 37. Na bodě 4) je již trojúhelníková síť vzniklá teselací poměrně hustá, u nejvyššího stupně teselace 64, by byla opticky hustá téměř dvakrát tak, počet polygonů by byl vyšší ještě víc, protože s úrovní teselace roste nelineárně.

Podobně funguje vnější teselace. Na obrázku 4.2 je znázorněn stejný trojúhelník, tentokrát s úrovní vnitřní teselace 1, a úrovní vnější teselace nastavenou na hodnotu 8 pro všechny hrany.

Pro nastavování vnějších úrovní teselace je třeba mít na paměti vztah nastavené úrovně a pořadí v jakém jsou zpracovávány vrcholy. Jak ukazuje obrázek 4.3, vnější úroveň teselace se stejným ID jako aktuální vrchol nastavuje teselaci hrany která je v trojúhelníku protilehlá. Pokud tedy chceme nastavovat úroveň teselace pro hranu ležící mezi vrcholy s ID 0 a 1, je třeba nastavit úroveň vnější teselace pro ID 2. To, kterou úroveň teselace chceme nastavovat, naštěstí není závislé na aktuálním ID, jak bylo zmíněno výše, a proto to nepředstavuje



Obrázek 4.1: Různé stupně vnitřní teselace

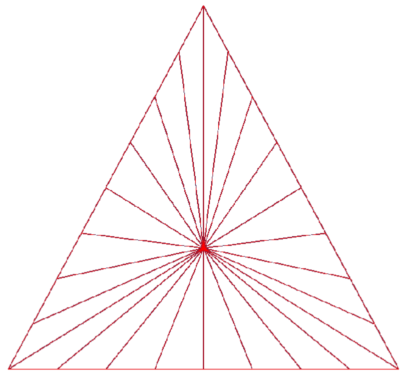
problém.

4.1.2 Patche typu quads

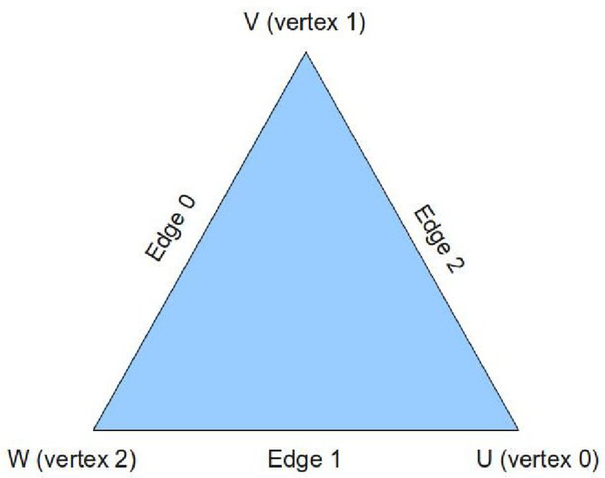
Dalším typem teselačních patchů jsou quads - čtyřúhelníky. Oproti trojúhelníkům se nastavují dvě různé vnitřní úrovně teselace, a čtyři vnější (pro všechny hrany čtyřúhelníku). V tomto případě se úrovně teselace se nastavují následovně

- Vnitřní úroveň s ID 0 nastavuje úroveň v ose x
- Vnitřní úroveň s ID 1 nastavuje úroveň v ose y
- Vnější úroveň s daným ID nastavuje úroveň teselace pro hranu spojující vrchol se stejným ID s jejím předchůdcem. Tedy, vnější úroveň pro ID 0 nastavuje hranu mezi vrcholy s ID 0 a 3, vnější úroveň pro ID 1 nastavuje hranu mezi vrcholy s ID 1 a 0, stejně tak pro zbývající hrany.

Na obrázku 4.4 lze vidět příklad teselace jednoho quadu. Na levé straně je nastavena úroveň vnitřní teselace na hodnotu 8, vnější pak na 32. Na pravém obrázku je to pak úroveň 32 pro vnitřní i vnější teselaci. Na obrázku lze také vidět, že výstupem teselace quadu jsou trojúhelníky



Obrázek 4.2: Vnější teselace

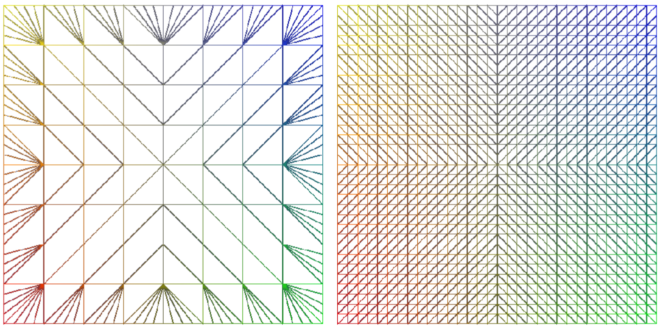


Obrázek 4.3: Organizace ID úrovní teselace a zpracovávaných vrcholů
zdroj obrázku: http://ogldev.atspace.co.uk/www/tutorial30/tutorial30.html

Ačkoliv se tento typ patchů nazývá quads, tedy čtyřúhelník, je možné nastavit vyšší počet vrcholů (v kvalifikátoru layout), je ale potřeba tomu přizpůsobit výpočet nového vrcholu v TES.

4.1.3 Patche typu isoline

Isoline jsou zde zmíněny pro úplnost. Jde o soustavu přímek v oblasti vyznačené čtyřúhelníkem. Lze je vidět na obrázku 4.5



Obrázek 4.4: Teselace quadů (čtyřúhelníků

Vyjma možností přesnějšího nastavení úrovní teselace je možné použít TCS ke cullingu patchů předtím než dojde k jejich teselaci. Normálně by culling probíhal až po zpracování polygonu v TES, který by tak v případě polygonu který by následně neprošel běžným cullingem prováděl spoustu zbytečných operací - v TES je třeba interpolovat s pomocí souřadnic vygenerovaných teselátorem všechny průchozí hodnoty abychom tak mohli získat vrchol ležící uvnitř patche, jako jsou normály, texturovací souřadnice, barvy, vektory potřebné pro normálové mapování, a případné další hodnoty. V praxi lze culling provést nastavením úrovně teselace na 0. Takové patche teselátor nezpracovává a pipeline je zahodí

4.2 Teselátor

Teselační jednotky jsou fixní neprogramovatelné části pipeline. Jejich funkci lze ovlivnit změnou úrovně teselace popsanou v předchozí části popisující TCS, a typem patchů popsaným ve stejné části. Souřadnice je ještě možno ovlivnit nastavením spacingu v TES, to bude popsáno v následující podkapitole, která jej bude popisovat. Teselační jednotky samy o sobě nevytvářejí vrcholy, pouze generují relativní souřadnice určující polohu nového vrcholu uvnitř patche. Souřadnice jsou využívány v TES, který se spustí pro každé souřadnice vygenerované teselátorem, pro vypočítání nového vrcholu. Teselační jednotky při své funkci ani neuvažují kolik má nastavený daný patch vrcholů, ke generování souřadnic tuto informaci nepotřebují.

4.3 Tessellation Evaluation Shader

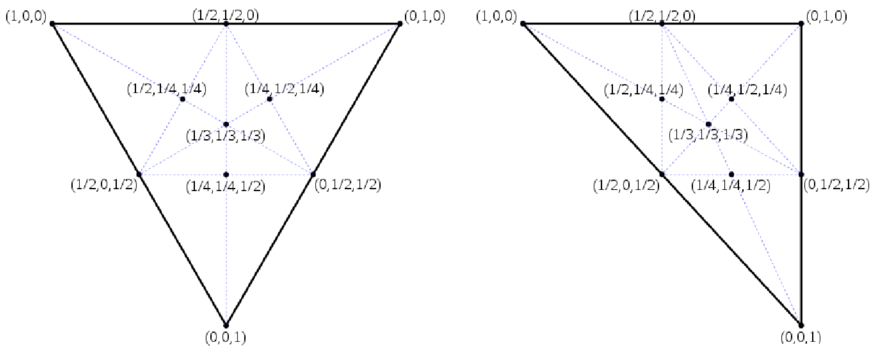
Na rozdíl od TCS je to pro využití teselace nutná část pipeline. Jejím vstupem jsou relativní souřadnice vygenerované teselátorem, určující polohu nového vrcholu uvnitř patche. Souřadnice jsou obsaženy ve vektoru gl_TessCoord (ten není potřeba deklarovat, je to vestavěná proměnná), jednotlivé složky mohou mít hodnoty od 0.0 do 1.0. Souřadnice jsou v závislosti na typu patche dvojího typu. Pro trojúhelníky jsou to barycentrické souřadnice. To jsou tříprkové souřadnice udávající relativní váhu jednotlivých původních vrcholů. Součet jednotlivých složek takovýchto souřadnic je roven 1, takže například souřadnice určující střed mají všechny tři složky o stejné hodnotě 1/3, souřadnice určující že nový vrchol leží na jedné z hran původního patche, tedy na spojnici dvou vrcholů, bude mít složku určující

Obrázek 4.5: Teselace isolines

váhu protilehlého vrcholu rovnou nule, a tak dále. Ilustrace je na obrázku 4.6. Výpočet nového vrcholu z původních vrcholů patche může mít následující podobu:

```
vec3 p0 = gl_TessCoord.x * tcPosition[0];
vec3 p1 = gl_TessCoord.y * tcPosition[1];
vec3 p2 = gl_TessCoord.z * tcPosition[2];
vec3 p_sum = (p0 + p1 + p2);
```

Zde jsou v poli tcPosition uloženy vrcholy patche, ty jsou vynásobeny jednotlivými složkami vygenerovaných souřadnic. Součtem jednotlivých složek získáme nový vrchol, ležící na pozici vygenerované teselátorem. Stejný postup lze použít i pro normály, texturovací souřadnice, a další.



Obrázek 4.6: Barycentrické souřadnice
zdroj obrázku: http://en.wikipedia.org/wiki/Barycentric_coordinate_system

Pro čtyřúhelníky a isolines jsou to 2D souřadnice udávající polohu v čtyřúhelníkovém patchi. Pouze dvě složky gl_TessCoord jsou využity, x a y. Výpočet nového vrcholu z těchto souřadnic je možný například následujícím způsobem:

```
vec3 a = mix(tcPosition[0], tcPosition[1], gl_TessCoord.x);
vec3 b = mix(tcPosition[2], tcPosition[3], gl_TessCoord.x);
vec3 p_sum = mix(a, b, gl_TessCoord.y);
```

funkce mix je lineární interpolace dvou vektorů, například v případě prvního řádku vytvoří nový bod s hodnotou

$$tcPosition[0] \cdot gl\_TessCoord.x + tcPosition[1] \cdot (1 - gl\_TessCoord.x)$$

Pro ilustraci lze použít obrázek 4.4. Na tomto obrázku je vrchol s indexem 0 znázorněn červeně, vrchol s indexem 1 zeleně, vrchol s indexem 2 žlutě a vrchol s indexem 3 modře. Tento výpočet tedy nejprve určí dva vrcholy uvnitř spodní a horní hrany vstupního patche, a poté určí bod na hraně mezi těmito dvěma novými body. Pokud bychom měli nastavený čtyřúhelník, ale z nějakého důvodu by bylo potřeba použít jako patch šestiúhelník, výpočet by bylo třeba upravit určitým způsobem, například uvažovat jej jako dva spojené čtyřúhelníky, a podle hodnoty jedné ze souřadnic umísťovat nový vrchol do některé z těchto polovin. Stejným způsobem fungují isolines, liší se jen v nastavení, která je popsáno v následující části. Stejně jako TCS, i Tessellation Evaluation Shader obsahuje kvalifikátor layout. V TES obsahuje jiné položky než v TCS, a jeho zápis může mít následující podobu:

```
layout(triangles, equal_spacing, cw) in;
```

První položka je typ patche. Tato položka je povinná, a určuje, jaké souřadnice budou generovány. Možné typy patche byly rozepsány výše. Druhou položkou je spacing. Tato položka je volitelná, a určuje vzájemnou vzdálenost generovaných souřadnic. Možné hodnoty jsou tyto:

- equal_spacing - slouží jako defaultní hodnota. Mezi vnějšími vrcholy bude konstantní vzdálenost, stejně tak mezi vrcholy vnitřními. Úroveň teselace je zaokrouhlována na celé číslo.
- fractional_even_spacing - teselovaný patch bude mít sudý počet segmentů, a všechny (opět samostatně vnější a samostatně vnitřní) segmenty kromě dvou budou mít konstantní vzdálenost. Úroveň teselace je desetinné číslo, a délka dvou segmentů s různou vzdáleností od ostatních bude daná hodnotou úrovně teselace ležící za desetinou čárkou.
- fractiona_odd_spacing - podobně jako fractional_even_spacing, ale počet segmentů je lichý.

Pro ilustraci rozdílů ve spacing je zde obrázek 4.7. Na úrovni teselace která je opačná jejich typu (například sudá úroveň teselace u lichého fractional_odd_spacing) lze vidět dva poloviční segmenty jak ve vnějších oblastech, tak ve vnitřní oblasti. Výhodou obou fractional spacingu je, že při zvyšování plynulé změně teselace se sice v určitém bodě zvýší skokově počet vrcholů, ale nezmění se skokově jejich poloha, ta se mění plynule. Tímto se zabývá jedna z pozdějších kapitol.

Poslední, také volitelnou, položkou layout v TES je zda mají být trojúhelníky, jejichž body jsou generovaný otočeny po směru hodinových ručiček (cw), nebo proti směru (ccw). To ovlivňuje případný culling před rasterizací. Defaultní hodnota je ccw.

Obrázek 4.7: Teselace isolines

4.4 Využití teselačních shaderů

Pokud by nově vzniklé vrcholy nebyly dále upravovány a teselace by byla použita pouze ve své nejjednodušší formě, výsledkem by byl pouze původní patch v rovinně rozdělený na menší, v případě trojúhelníku by po rasterizaci nebyla viditelná žádná změna. Proto je vhodné vzniklé vrcholy dále upravit nebo posunout. Pole možností je poměrně široké:

- Beziérův povrch [6], případně jiný podobný typ parametrického povrch - protože vrcholy, které jsou výstupem TES, nemusí mít žádný vztah k vrcholům vstupního patche, je možné jako vrcholy patche použít řídící body zvoleného parametrického povrchu, a z nich a ze souřadnic vygenerovaných teselačními jednotkami získat tak požadovaný povrch tvořený trojúhelníky.
- Vodní hladina nebo jiný vlnící se povrch - takového to efektu lze dosáhnout tak, že se budou vrcholy vygenerované v TES posouvat na základě výsledku připravené šumové funkce. Sama o sobě není vlnící se vodní hladina nebo jiný povrch novinkou, teselace a možnost měnit úroveň teselace na základě vypočtené vzdálenosti od kamery nabízí možnost zlepšit vzhled objektů blízko kamery, aniž by byla potřeba mít připravenou složitější geometrii než je použita pro objekty v dálce.
- Phongova teselace [8] - je metoda, která za pomocí normal, tangent, a bitangent vektorů umožňuje dopočítat pozici vektoru tak aby geometrie modelu odpovídala osvětlení. Výsledkem je zakulacení povrchů.
- Displacement Mapping - tato metoda bude blíže popsána v dalších částech této práce.

Přínosy teselace lze shrnout do následujících bodů

- Komprese objektu - lze vykreslit detailnější objekt než jaký je uložený v paměti, tím že buď vykreslovaný objekt vypočítáme z jeho uložené geometrie, nebo, v případě displacement mappingu, se část informace uloží do úspornější podoby, například jednobarevné textury, a z kombinace redukované uložené geometrie a této textury se získá původní objekt s žádnou nebo minimální odchylkou.

- Level of Detail - lze měnit složitost geometrie objektu aniž by bylo třeba mít uloženo víc geometrií s různým detailem pro jeden objekt. Změnu lze určit na základě vzdálenosti objektu.

- Urychlení vykreslení - snížení velikost geometrie může mít kladný vliv na snížení doby potřebné na vykreslení objektu, například snížením cache miss, nebo počtu přístupů do paměti.

Kapitola 5

Displacement Mapping, displacement textury, a jejich generování

Displacement Mapping je metoda vykreslování objektu využívající teselátoru grafické karty. Používá připravenou texturu osahující výškovou informaci k tomu, aby změnila pozici nových vrcholů získaných teselací původní geometrie, a dala tak vykreslovanému objektu detailnější vzhled.

5.1 Displacement mapa

Displacement mapa je textura obsahující výškovou informaci. Obvykle se získává porovnáním objektu s redukovanou geometrií s původním neredukovaným objektem. Taková mapa může být jednobarevná, za cenu nižšího rozlišení výškové informace, nebo může být informace rozdělena do více barev. K dané mapě je také potřeba vědět jaký posun představuje změna hodnoty o 1 bod, a jaká hodnota představuje nulový posun. Nulový posun bývá standardně na hodnotě 127 (v shaderu jako 0.5), ale může to být i jiná hodnota, pokud má například původní geometrie vyšší výstupky a mělčí prohlubně, a je tak užitečné mít vyšší rozlišení u výstupků než u prohlubní, nebo naopak.

Tvorbě displacement mapy a práci s ní je vhodné věnovat zvýšenou pozornost. Jak bude ukázáno v jedné z následujících kapitol, na hranách texturovací sítě jinak mohou vznikat mezery v geometrii teselovaného objektu.

5.2 Posun vrcholů

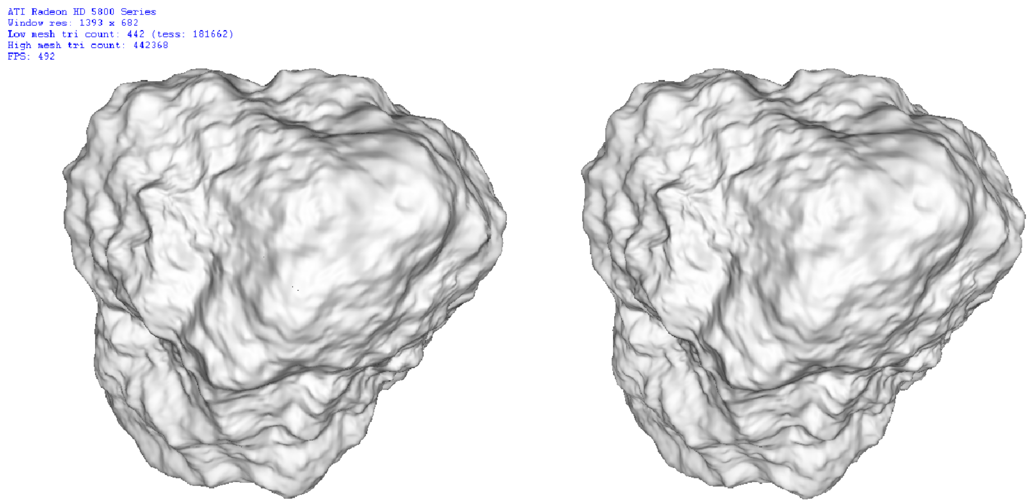
Dále je nutné vědět, kterým směrem se má vertex posunout. V příkladech připravených v této práci se používají vyhlazené (smooth) normálové vektory, a k určení směru posunu se využívá normálový vektor vypočítaný pro nový vrchol v TES. Výhodou je to že takto je jisté že vrcholy ležící na hraně patche budou mít stejný směr posunu jako vrcholy na této hraně sousedního patche.

5.3 Generování displacement map

Jak bylo zmíněno výše, mapy jsou generovány jako rozdíl redukované a neredukované geometrie. V rámci této práce byly vyzkoušeny dvě různé aplikace, program Blender, a program GPU MeshMapper

5.3.1 GPU MeshMapper

GPU MeshMapper je volně dostupný nástroj určený pro generování normálových, displacement, a ambient occlusion map. Má oproti Blenderu výhodu při generování displacement map, a to že určí rozsah posunu a středovou hodnotu, pomocí kterých je možné věrně napodobit vzhled původního modelu. Snaží se filtrovat texturu podél hrany, tak aby vznikaly minimální, pokud možno žádné, díry v teselovaném modelu. Výstup má ale stále menší nedostatky.



Obrázek 5.1: Napravo je původní objekt, nalevo je zjednodušený objekt vykreslený s využitím displacement a normal mappingu

5.3.2 Blender

Blender je open source nástroj umožňující vytváření 3D modelů, jejich nejrůznější modifikaci, a vytváření videí. Byl využit v této práci pro vytváření objektů použitých v ukázkách, které budou popsány dále. Blender obsahuje funkcionalitu pro generování různých typů map ať už porovnáním detailního a redukovaného objektu, nebo z nastaveného materiálu. Příkladem takto generovaných mapy mohou být normálové mapy, displacement mapy, specular mapy, diffuse mapy (běžné barevné textury). Nevýhodou generování displacement map v Blenderu je obtížně zjistitelná hodnota určující jaký posun představuje změna barvy o 1 bod. Zatímco GPU MeshMapper při generování vrátí informaci o středu a o posunu, Blender nikoliv, a je nutné je ručně získat z nastavení použitého pro vygenerování displacement mapy.

Kapitola 6

GPUperfAPI

GPUperfAPI[1] je knihovna vytvořená společností AMD, pro použití s grafickýma kartami Radeon. Slouží k inicializaci, nastavení a přistupování k různým čítačům grafické karty, a umožňuje tak zkoumat zatížení jednotlivých částí karty při vykreslení dané scény. Umožňuje pracovat s DirectX (od verze 10), OpenGL, OpenCL.

Čítače jsou rozdělené do skupin podle jejich povahy, jednu skupinu představují čítače časového vytížení karty a jednotlivých bloků a shaderů, dále jsou k dispozici podrobnější čítače jednotlivých bloků (například počet přístupů do paměti, čas strávený přístupy do paměti, počet zpracovaných prvků, vytížení ALU jednotky, atd.)

U shaderů se zaznamenává počet přístupů k textuře a čas strávený těmito instrukcemi, průměrný počet aritmetických instrukcí, poměr aritmetických a texturovacích instrukcí, počet vstupních a výstupních prvků, vytížení shaderu, a efektivitu.

Skupiny přístupných čítačů jsou tyto

- Timing - Celkový čas výpočtu, procento využití shaderů shaderů a rozložení zátěže mezi ně. Lze tak zjistit která z části pipeline je přetížená, a který z shaderů je třeba zoptimalizovat. Také slouží jako vodítko k určení části pipeline, na jejíž podrobné čítače pomohou analyzovat zátěž.

- VertexShader - Kromě údajů, které jsou i u ostatních shaderů nabízí i počet vertexů, které nebyly v cachi, a čas strávený jejich načítáním z paměti. Do počtu zpracovaných vertexů se započítává i počet vertexů zpracovaných domain shaderem.

- HullShader - Hull Shader je DirectX označení pro TCS, patří sem jeho čítače, které jsou základní shaderové čítače, popsané výše.

- DomainShader - Domain shader je DirectX označení pro TES.

- GeometryShader - Čítače geometry shaderu, kromě standardních čítačů nabízí i čas strávený exportováním primitiv.

- PrimitiveAssembly - Čítače operací týkajících se primitiv, tedy počet ořezaných primitiv, doba čekání na pixel shader s odesláním nového fragmentu, a počet fragmentů na trojúhelník.

- PixelShader - Kromě standardních čítačů přítomných u shaderů nabízí i čas strávený čekáním, až bude možné poslat fragment dále na Z-Test a do ColorBufferu

[1] http://developer.amd.com/tools/graphics-development/gpuperfapi/

- TextureUnit - Celkový počet přístupu k texturám, čekání na texture cache, čas strávený filtrováním.
- TextureFormat - Čas strávený prací s texturami jednotlivých formátů.
- ComputeShader - Čítače pro OpenCL.
- DepthAndStencil - Z-test a Stencil test. Počet fragmentů, které prošly testováním CHECK
- ColorBuffer - Počet paměťových přístupů

Protože není možné zaznamenávat hodnoty ze všech čítačů současně, jsou nezávisle na výše uvedeném rozdělení rozděleny do dalších skupin, podle toho, které je možné zaznamenávat současně. Pro zaznamenání hodnot více čítačů je tedy třeba určit, kolikrát musíme scénu vykreslit pro získání všech dat. Toto rozdělení je závislé na ovladači, API a konkrétní architektuře grafické karty, pro jeho získání je třeba použít jednu z funkcí této knihovny, a scénu vykreslit vícekrát, v závislosti na této hodnotě. Postup při použití této knihovny má přibližně následující podobu

- preinicialize - před vytvořením vykreslovacího kontextu se voláním funkce GPA_Initialize připraví ovladač na vytvoření kontextu s čítači
- incializace - po získání handle na vytvořený vykreslovací kontext se funkcí GPA_Initialize nastaví tento kontext jako aktuální, poté je třeba nastavit čítače (po spuštění jsou všechny deaktivované)
- analýza - pro celý průběh se nejprve vytvoří session, následně se získá počet potřebných průchodů (závislý na grafické kartě a kombinaci povolených čítačů; Průchod se zahájí a ukončí dvojicí GPA_BeginPass, GPA_EndPass). V každém průchodu se musí vykreslit celá scéna, toto vykreslení se dále zaobalí dvojicí GPA_BeginSample(x), GPA_EndSample(). Pomocí proměnné X je možno označit zaznamenaný vzorek čítačů, a mít tak více dat. Nevýhodou je potřeba dalšího vykreslení scény.
- získání údajů - než je možné přistoupit k hodnotám čítačů, je třeba ukončit session, a následně čekat než bude uzavřená session připravená k čtení (GPA_IsSessionReady). Protože GPUperfAPI umožňuje pamatovat si jen omezený počet session, je následně třeba hodnoty ze session zaznamenat. Pro jejich čtení je třeba kontrolovat jakého typu je daný čítač (int32/64, float32/64), protože pro každý typ je možné číst hodnotu jen funkcí pro daný typ určenou.
- ukončení - před ukončením aplikace je třeba zavolat funkci GPA_Destroy, která deaktivuje čítače. Pokud by nebyly deaktivovány, může být negativně ovlivněn výkon dalších aplikací pracujících s grafickou kartou.

6.1 GPU PerfStudio 2

Je freeware aplikace založená na GPUperfAPI, vyvíjená společností AMD. Umožňuje analyzovat již existující 3D aplikaci bez úpravy kódu nebo nutnosti ji znovu zkompilovat, může sloužit jako 3D debugger, nebo poskytovat za běhu informace z jednotlivých čítačů.

6.2 NVPerfKIT

Knihovna NVPerfKIT je, podobně jako GPUperfAPI, rozhraní sloužící pro přístup k výkonovým čítačům grafické karty. Naneštěstí se s tímto rozhraním objevily problémy, jejichž původ se nepodařilo blíže objasnit. NVPerfKIT i k němu přiložená demonstrační aplikace bylo zkoušeno na dvou různých počítačích, a na obou měly čítače při odečítání hodnot téměř vždy nulovou hodnotu.

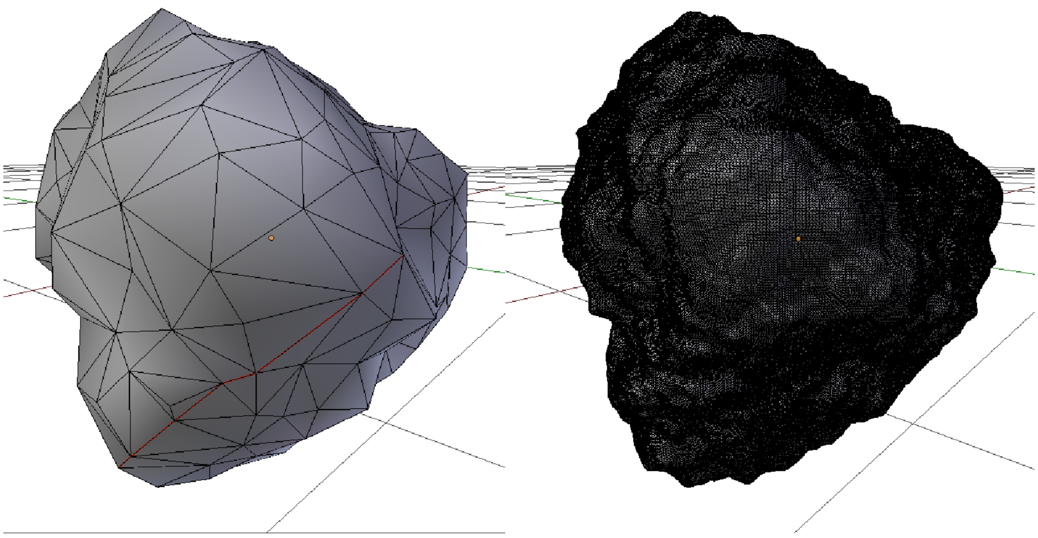
Kapitola 7

Příklady displacement mappingu

V rámci práce bylo připraveno několik příkladů demonstrujících možnosti displacement mappingu, a problémy které při něm mohou nastat. Podrobněji jsou rozepsány v následujících podkapitolách. Za účelem demonstrace bylo připraveno několik objektů, a to dlážděná plocha, a model asteroidu vytvořený v Blenderu.

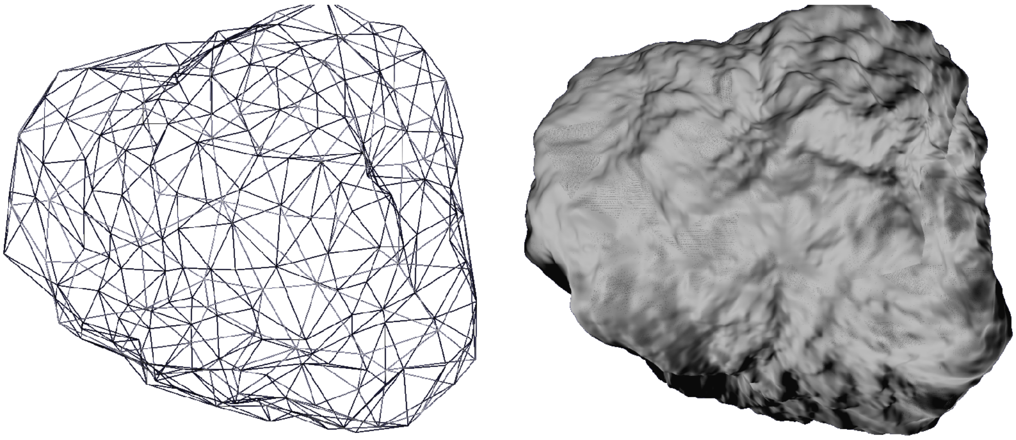
7.1 Asteroid

Model asteroidu vznikl redukcí původního modelu s vysokým počtem polygonů, a jeho doplněním o displacement a normálovou texturu. Na obrázku 7.1 lze vidět srovnání redukovaného modelu s původním. Červenou barvou je na obrázku znázorněno kudy vede na redukovaném modelu šev v síti texturovacích souřadnic



Obrázek 7.1: Zobrazení použitých modelů - na pravé straně je původní model, na levé straně je model zjednodušený

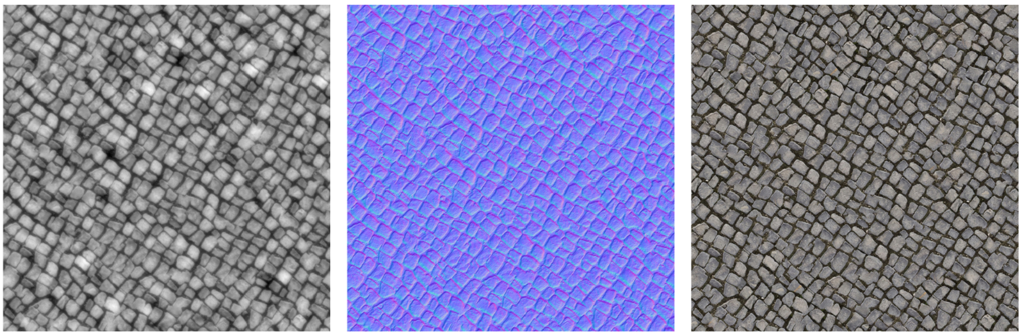
Teselací tohoto zjednodušeného modelu lze, při nejvyšším stupni teselace, získat model s ještě vyšší hustotou vyšší hustotou geometrie než měl původní model. Na obrázku 7.2 je zobrazení drátového modelu redukovaného modelu asteroidu s aplikovaným displacement mappingem. Na levé straně obrázku je model s úrovní teselace 1, lze vidět geometrii redukovaného modelu. Na pravé straně je model s úrovní teselace 48. Ačkoliv se na první pohled může zdát, že tento model není vykreslován jako drátový model, zejména v levé části obrázku si lze povšimnout, že v některých patchích se vyskytuje šum, který je způsobený prosvítajícím pozadím.



Obrázek 7.2: Ilustrace modelu s úrovní teselace 1 (nalevo), a úrovní teselace 48 (napravo)

7.2 Dlážděná plocha

Druhým modelem je model dlážděné plochy. Kromě displacement mapy dále používá normálovou mapu a diffuse mapu. Texturu, která je zobrazena na obrázku 7.3, lze použít na dlaždice, tzn. horní a spodní okraj textury na sebe navazuje, stejně tak levý a pravý.



Obrázek 7.3: Textury dlážděné plochy. Z leva: displacement mapa, normálová mapa, diffuzní mapa

7.3 Základní displacement mapping na jednom samostatném objektu

Displacement mapping v minimální podobě se provádí v Tessellation Evaluation Shader, a stačí k němu model se smooth normálami a displacement mapa. Nejprve je třeba získat texturovací souřadnici pro aktuální vrchol (tvořený z relativních souřadnic vygenerovaných teselátorem) způsobem který byl popsán v části 4.3. Pomocí vypočítaných texturovacích souřadnic se získá hodnota z displacement mapy:

```
float displacement = (texture(displacement_texture, teTexcoord).b - 0.045489)*0.106226;
```

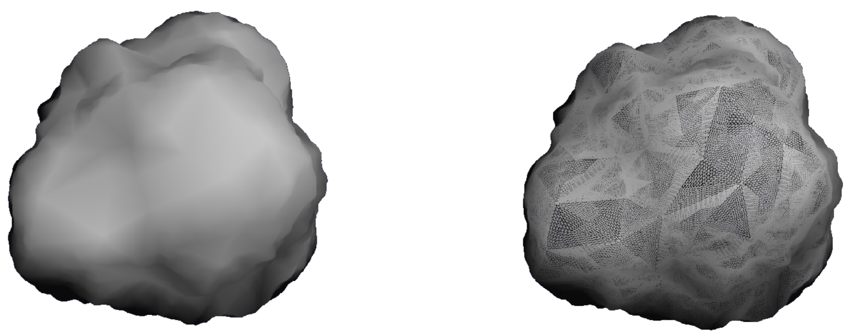
kde teTexcoord jsou vypočítané texturovací souřadnice. Textury použité v těchto ukázkách jsou uloženy jako RGBA, v případě displacement textur jsou všechny hodnoty stejné, složka *b* je zde zvolena pouze pro zjednodušení. Při praktické aplikaci displacement mappingu může být hodnota displacement mapy například uložena v alfa složce diffuzní textury. Hodnota -0.045489 je zde pro nastavení hodnoty získané z mapy, která představuje nulový posun. V tomto případě nulový posun představuje hodnota 0.045489 získaná z textury, většina hodnot v textuře tedy slouží pro posun směrem z objektu.

Tuto hodnotu je dále třeba aplikovat na vypočítaný vertex. Pro aplikaci je třeba znát směr posunu vertexu, v těchto příklad je jako směr využit normálový vektor vypočítaný z normálových vrcholů patche.

```
vec3 displacedVertex = teNormal*displacement+teVertex;
```

Určitou nevýhodou displacement mappingu je nutnost přesunout MVP transformaci z Vertex Shaderu do Tessellation Evaluation Shaderu, a transformovat až posunutý vrchol, čímž se zvýší počet transformací. V opačném případě by posunutý transformovaný vrchol neodpovídal vrcholu z neredukovaného modelu, provádět i transformaci posunu by sice možné bylo, ale nezabránilo by se tím zvýšení počtu transformací oproti prvnímu příkladu, naopak by jejich počet ještě vzrostl. Výsledný vrchol se přiřadí do proměnné gl_Position, nebo předá dále do případného geometry shaderu.

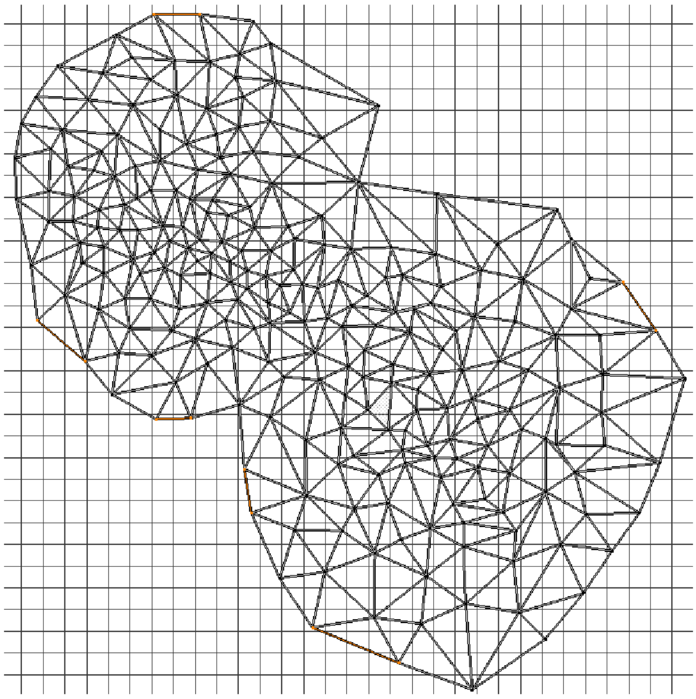
Displacement mapping je také více než vhodné doplnit o normal mapping - protože se pro displacement mapping používají modely s jednodušší geometrií, měly by jen málo normálových vektorů, ze kterých by se počítalo osvětlení. Jako příklad lze použít obrázek 7.4.



Obrázek 7.4: Asteroid po aplikaci displacement mappingu bez normal mappingu

7.4 Příčina vzniku děr v objektu a jejich řešení

Displacement mapping sebou nese jedno riziko, a to vznik děr v geometrii modelu. Ty obecně mohou vzniknout, pokud není každé volání TES pro všechny zpracovávané vrcholy ležící v jednom bodě prostoru deterministické. V případě displacement mappingu to může být způsobeno tím že body v sousedících patchích, které leží na hraně, kterou spolu tyto patche sousedí, mohou mít různé texturovací souřadnice a hodnoty z displacement mapy které získají mohou být různá. Jako příklad lze uvést obrázek 7.1, konkrétně redukovaný model - zde jsou takovými sousedícími patchi ty trojúhelníky, které se dotýkají jednou z hran červeně vyznačeného švu v síti texturovacích souřadnic. První použitá verze měla jednoduchou síť texturovacích souřadnic, ta je znázorněná na obrázku 7.5. Lze vidět, že dvojice hran tvořících švy mají různé úhly, a různou délku.

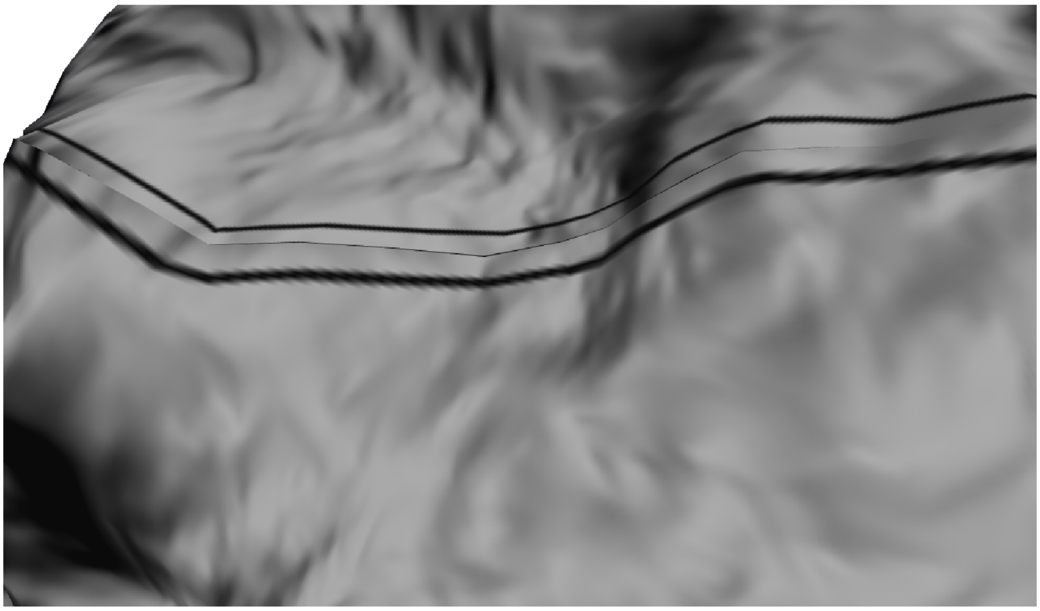


Obrázek 7.5: První verze sítě texturovacích souřadnic.

Model s těmito souřadnicemi měl tak po vykreslení poměrně velké díry v geometrii, které kazily výsledný efekt. To platí pro mapu generovanou Blenderem, i pro tu generovanou MeshMaperem, jak lze vidět na obrázku 7.6

Z tohoto důvodu byla síť texturovacích souřadnic upravena, ve snaze zamezit tak různým úhlům a délkám hran v texturovací síti, které spolu v geometrii sousedí. Očekávaný výsledek byl ten, že tato změna aplikaci generující displacement mapu umožní generovat displacement mapu tak, aby si hodnoty v textuře ve švech odpovídaly. Texturovací souřadnice byly upraveny ve dvou vlnách, nejprve byly hrany sítě texturovacích souřadnic ručně nastaveny na kraje mapy, pak byla použita funkce Blenderu na narovnání vnitřních trojúhelníků sítě. Tyto souřadnice lze vidět na obrázku 7.7.

Díry v geometrii se bohužel objevily opět. Nakonec bylo nutné ručně upravit texturu, aby si hrany ležící ve švech po aplikaci displacement mappingu odpovídaly. Toto řešení již bylo úspěšné.



Obrázek 7.6: Díry v geometrii po aplikaci displacementu s první verzí texturovacích souřadnic

7.4.1 Shrnutí

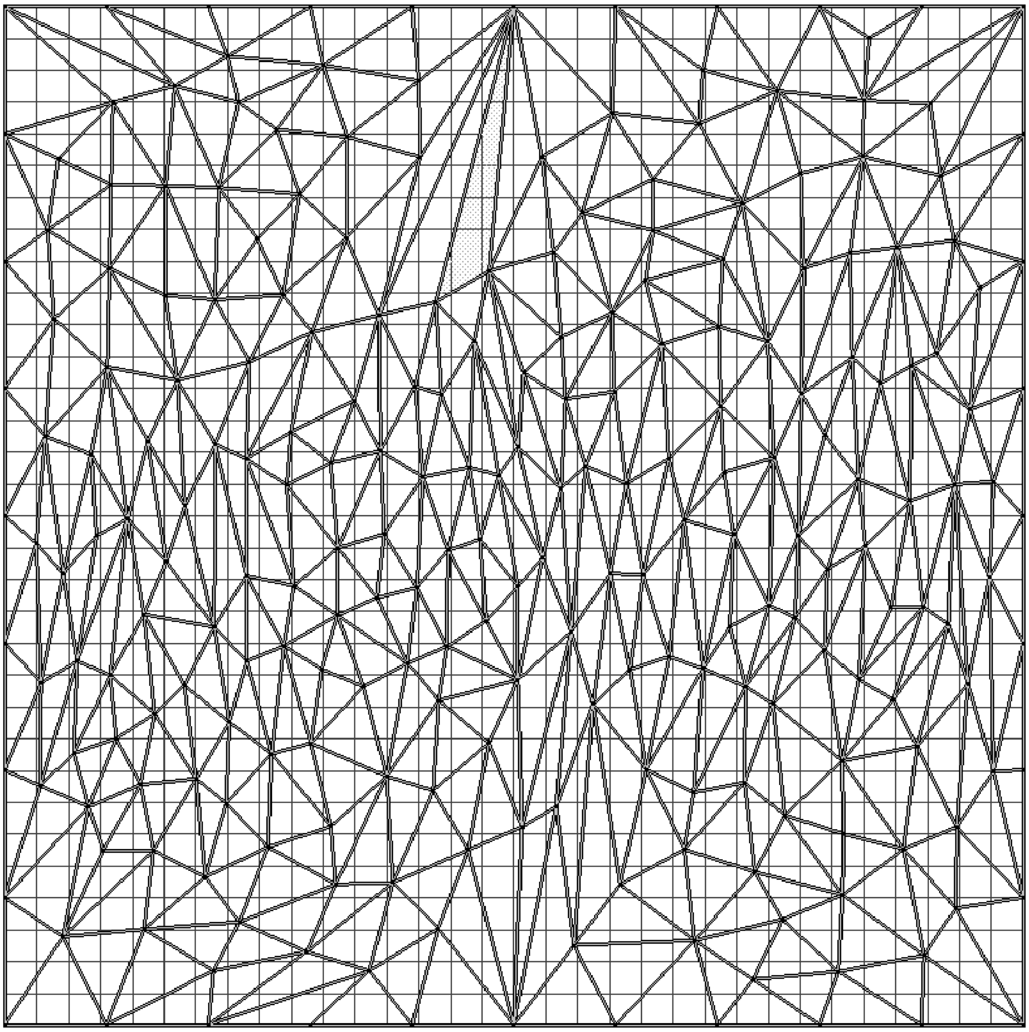
Lze předpokládat, že bez předpřípravy modelu bude běžně při displacement mappingu docházet ke vzniku děr ve výsledné geometrii. Naneštěstí ne všude bude jednoduše možné ručně upravovat texturovací síť, ve většině případů pro to že pro model již ostatní textury existují, nebo je není možné jednoduše generovat. Lze předpokládat, že toto se bude týkat zejména modelů s ručně upravovanou diffusní texturou. Řešením může být přidat druhou verzi texturovacích souřadnic extra pro displacement mapping. V průběhu vývoje bylo také experimentováno s metodou, která pracovala s druhou sadou texturovacích souřadnic, vytvořenou při předzpracování modelu a pomocí kódu přidaného do shaderů detekovala nové vrcholy ležící ve švech sítě texturovacích souřadnic, a s pomocí druhé sady texturovacích souřadnic vždy jedna z hran ze švu použila texturovací souřadnice protilehlé hrany. Ačkoliv toto řešení bylo funkční, představovalo velký nárůst instrukci v shaderu, zejména větvení kódu, a také ji nelze účinně použít pro texturovací sítě, kde se šev může dělit na více různých (například T-spoje) bez přidání dalších sad texturovacích souřadnic.

7.5 Level of Detail asteroidu

Jednou z výhod displacement mappingu je možnost měnit počet zpracovávaných vrcholů v kódu shaderu, bez nutnosti spolupráce s CPU.

První použitá, základní metoda používala pro zjištění vzdálenosti modelu od kamery transformaci "falešného"vrcholu ležícího v bodě [0.0,0.0,0.0]. Po transformaci tohoto bodu pomocí ModelView matice se určila vzdálenost transformovaného vrcholu od počátku ve View space, kde v počátku leží kamera. Tato vzdálenost se pak přepočítala na koeficient, který se použil pro změnu úrovně teselace

```
vec4 pos = osg_ModelViewMatrix * vec4(0.0, 0.0, 0.0, 1.0);
float degree = length(pos);
```



Obrázek 7.7: Upravené texturovací souřadnice

```
degree = 1/degree;
gl_TessLevelInner[0] = degree*TessLevelInner;
gl_TessLevelOuter[0] = degree*TessLevelOuter;
gl_TessLevelOuter[1] = degree*TessLevelOuter;
gl_TessLevelOuter[2] = degree*TessLevelOuter;
```

Při přípravě větší scény a použití OpenSceneGraph nodu PositionAttitudeTransform[1] pak nastal problém, kdy zřejmě transformační matice vytvářená tímto nodem deformovala původní výpočet vzdálenosti změnou vzdáleností. Z tohoto důvodu byl použit další, méně efektivní výpočet, využívající výpočet vzdálenosti od středu podobně jako v prvním příkladě, ale zároveň má druhý bod, který je před transformací v konstantní vzdálenosti od bodu prvního.

```
vec4 pos = osg_ModelViewMatrix * vec4(0.0,0.0,0.0,1.0);
```

[1]Tento node mění podle nastavených parametrů pozici, rotaci a měřítko dalších nodů, které jsou k němu připojeny.

```
vec4 correction = osg_ModelViewMatrix * vec4(0.0,0.0,100.0,1.0);

float degree = length(pos);
float correction_length = distance(pos, correction);

degree = 100.0*(degree / correction_length);   // this should restore modelspace distan

float tessLevel = max(TessLevelInner/degree,1.0);
gl_TessLevelInner[0] = tessLevel;
gl_TessLevelOuter[0] = tessLevel;
gl_TessLevelOuter[1] = tessLevel;
gl_TessLevelOuter[2] = tessLevel;
```

V tomto výpočtu se tedy získá opět vzdálenost počátku v Model space od počátku ve View space, a pak pro získání korekci měřítka deformovaného PositionAttitudeTransform se totéž provede pro pomocný vrchol posunutý o 100 bodů v Model space oproti středu. Z těchto dvou transformovaných vrcholů se určí jaké vzdálenosti ve View space odpovídá vzdálenost 100 bodů v Model space. Hodnota 100.0 ve výpočtu proměnné degree slouží jako koeficient pro nastavení míry změny úrovně teselace podle potřeb modelu.

Druhou, programátorsky čistější variantou, je vytvořit upravený Node v OpenSceneGraphu. V něm vytvořit funkci traverse, která se provádí při vykreslení každého snímku, a získat z instance třídy NodeVisitor (prochází při vykreslování strom, obsahuje potřebné hodnoty, jako jsou transformační matice) vzdálenost uzlu od kamery pomocí funkce NodeVisitoru

```
getDistanceToViewPoint(getBound().center(),true)
```

Tuto vzdálenost pak lze předat do Tessellation Control Shaderu jako uniform, nebo rovnou vypočítat stupeň teselace a ten předat. To může ale představovat u složitějších scén zbytečnou zátěž na CPU.

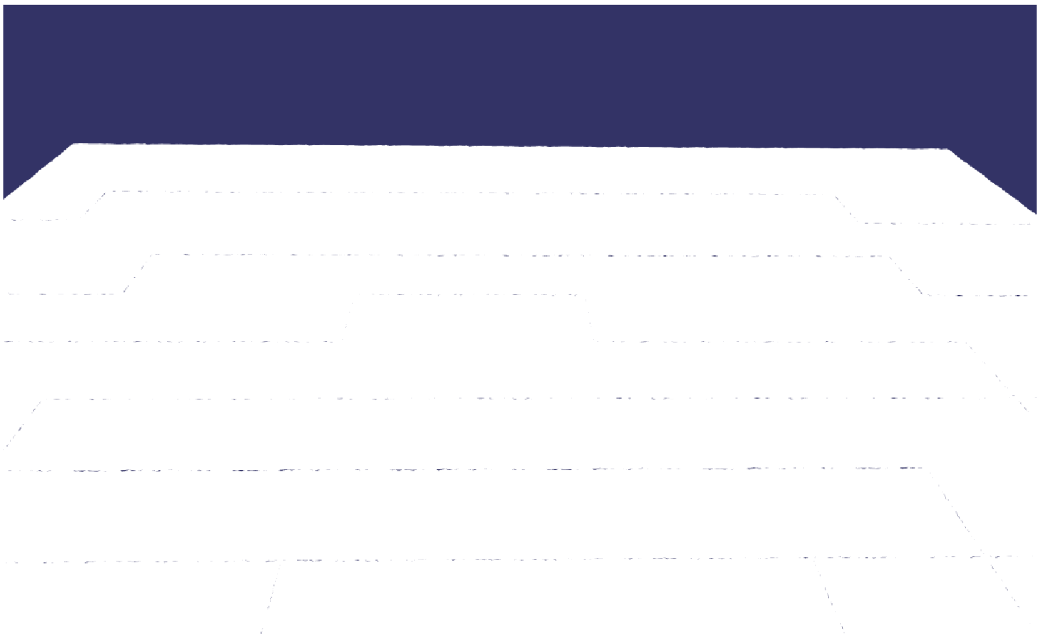
7.6 Level of Detail dlážděné plochy a vliv spacingu

Jedním z možných využití displacement mappingu je vykreslování terénu například v počítačových hrách. Ten často bývá poměrně rozsáhlý[2] a běžně se některá forma LOD používá. Problémem LOD terénu pomocí displacement mappingu je změna úrovně teselace se vzdáleností - pokud má každý díl terénu uniformní úroveň teselace, a sousedící díly terénu mohou mít různý stupeň teselace, nemusí odpovídat počet pixelů v hranách, kterými díly terénu sousedí. Pokud je použit výpočet popsaný v 7.4, každý díl terénu má jinou transformační matici, je tak jiná vzdálenost od středu View space, a je vypočten jiný stupeň teselace.

Výsledek lze vidět na obrázku 7.8. Pro potřeby lepší ilustrace chování byl upraven fragment shader tak, že všechny fragmenty mají bílou barvu. Modrá barva, která tvoří pozadí pak prosvítá v místech, kde jsou díry ve výsledné geometrii.

Použité řešení tohoto problému spočívá v přípravě speciální geometrie, s určitou podobou geometrické sítě. Tato síť s vyznačenými důležitými prvky které budou v následujícím odstavci popsány je znázorněna na obrázku 7.10

[2]Jako příklad lze použít hry The Elders Scroll: Skyrim a The Elders Scroll: Oblivion, kde se ale k LOD nepoužívá displacement mapping. Kde se displacement mapping k LOD používá je například Tom Clancy's H.A.W.X. 2



Obrázek 7.8: Díry v terénu s výpočtem stupně teselace dle vzdálenosti modelu od kamery

V této geometrii je podstatné pořadí vrcholů v každém trojúhelníku - trojúhelníky jsou tvořeny tak, aby žádný z vnitřních trojúhelníků nezačínal vrcholem ležícím na hraně, a zároveň aby trojúhelníky ležící na jednotlivých krajích geometrie začínaly vrcholem s barvou vyznačenou pro příslušný okraj. Pořadí indexů na jednotlivých krajích je vyznačeno na obrázku. Důvodem této zvláštní organizace je, že oproti LODu popsaném v 7.5 se nepoužívá pouze pevně nastavený střed pro celý model, ale každý vrchol má přiřazený svůj střed, podle něhož se počítá vzdálenost dodatečná vzdálenost od středu View space. Vrcholy, které nejsou vyznačeny barevně mají nastaven střed [0.0, 0.0, 0.0], vrcholy které barevně vyznačeny jsou, mají střed nastavený ve středu jejich okraje geometrie. Úroveň vnitřní teselace a vnější pro hrany s ID 0 a 1 se počítá ze standardního středu, úroveň vnější teselace pro hranu s ID 2 se počítá od upraveného středu (proč ID 2 lze nalézt na obrázku 4.3)

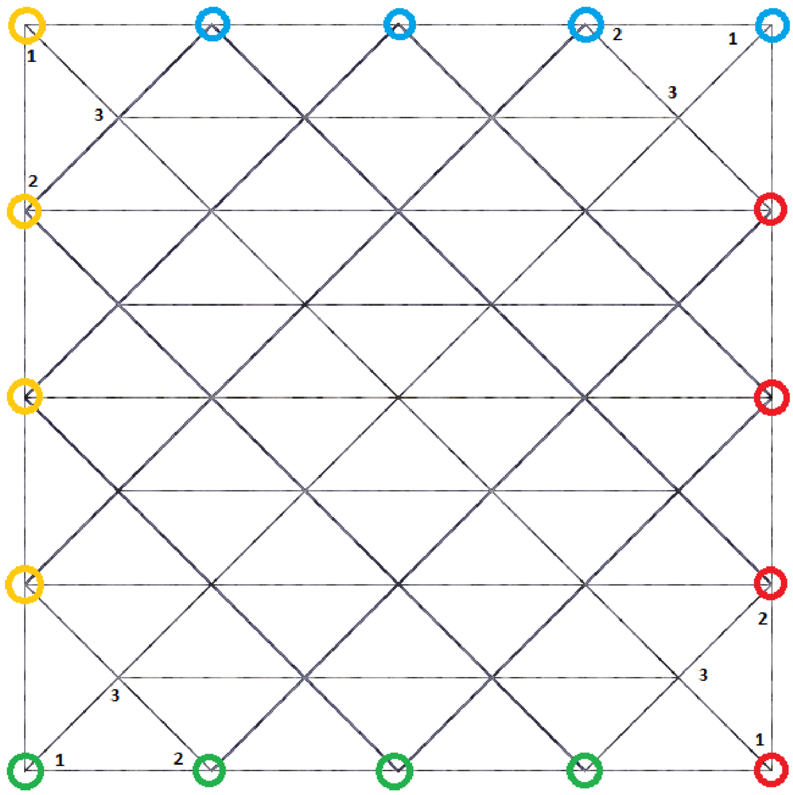
Tímto se docílí toho, že hrany sousedících dílů mají stejnou úroveň teselace, a v terénu pak nevznikají trhliny. Pokud si vykreslíme výsledný povrch s podobnou úpravou, jako to bylo na obrázku 7.8, lze vidět že výsledná geometrie už neobsahuje žádné díry.

Vliv na vzhled geometrie má i nastavení spacingu. Ten byl popsán v kapitole 4.3. Pokud je použito nastavení equal_spacing, je při pohybu objektu vidět při změně celočíselné úrovně teselace že se jednotlivé pixely vykresleného oVrcholy které viděžbjektu posunuly o jeden nebo více pixelů. V případě fractional_even_spacing a fractional_odd_spacing nejsou přítomny žádné výrazné skoky, ale stále lze pozorovat změny polohy pixelů, v tomto případě připomínají vlnění. Toto chování zejména vynikne na modelu hrubé dlažby při pohledu pod úhlem, na asteroidu již tolik výrazné není.

0

7.7 Větší scéna s více objekty a se skyboxem

Dalším připraveným příkladem je větší scéna s více objekty, a skyboxem na pozadí. Tato scéna využívá dva typy modelů asteroidů, které jsou vykreslovány s využitím displacement



Obrázek 7.9: Ilustrace speciální geometrie pro terén

mappingu, a dynamickou změnou úrovně teselace, tak jak byla popsána v části 7.5.

Modely byly opět vytvořeny v programu Blender.

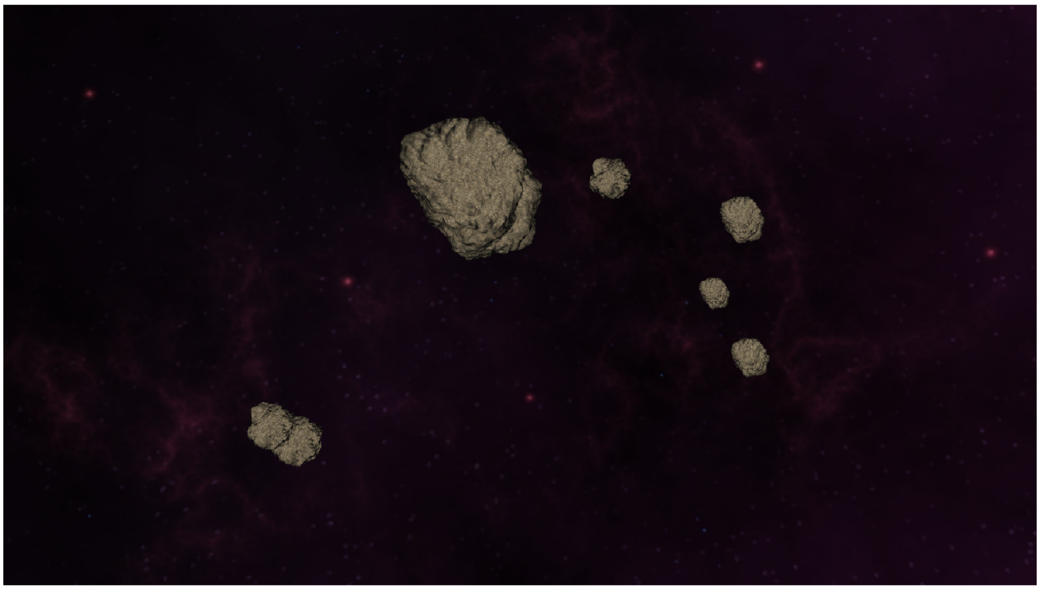
7.7.1 Skybox

Knihovna OpenSceneGraph neobsahuje standardní řešení pro skybox. Ten byl pro potřeby této ukázky vytvořen jako třída dědící ze třídy Geode, obsahující 6 samostatných objektů představující jednotlivé stěny skyboxu (byl zvolen krychlový skybox)

Umístění skyboxu do scény řeší část kódu vertex shaderu:

```
vec4 tempPosition = osg_ProjectionMatrix*vec4(osg_NormalMatrix*myVertex,0.0001);
tempPosition.z = tempPosition.w - 0.00001;
gl_Position = tempPosition.xyzw;
```

První řádek řeší transformaci a projekci. Protože je potřeba zabránit posunu skyboxu, je použita matice pro transformaci normálových vektorů. Výpočet na druhém řádku pak umísťuje skybox hluboko do scény, aby nepřekrýval jiné objekty. Odečet hodnoty 0.00001 pak řeší zaokrouhlovací chyby, které se bez něj objevily.



Obrázek 7.10: Ilustrace speciální geometrie pro terén

Kromě této části vertex shaderu bylo pro skybox dále potřeba vypnout pro skybox culling (Ačkoliv se skybox vykresluje na místě, OpenSceneGraph ho stále udržuje jako objekt se kterým by bylo možné pohybovat, a mohl by tak zabránit jeho vykreslení pokud by při provádění cullingu detekoval skybox mimo kameru. Tohoto se také týká jeden bug OpenSceneGraphu, který způsoboval že OpenSceneGraph stále při cullingu mohl vynechat jeho vykreslení. Tento bug byl již dříve popsán na stránkách OpenSceneGraphu [3], je způsoben špatným výpočtem BoundingBoxu. Pro jeho obejití je potřeba použít novou instanci třídy ComputeBoundingBoxCallback, například

```
struct ComputeSkyBoxBoundingBoxCallback :
    public osg::Drawable::ComputeBoundingBoxCallback
{
META_Object(osg,ComputeBoundingBoxCallback);

virtual BoundingBox computeBound(const osg::Drawable&) const
{ return BoundingBox(); }
};
```

[3] http://forum.openscenegraph.org/viewtopic.php?p=55946#55946

Kapitola 8

Displacement mapping a dopady na výkon

Jednou z nevýhod displacement mappingu je, že pro vykreslení redukovaného modelu se stejným detailem jaký měl původní, neredukovaný model, je potřeba provádět více operací - je potřeba vypočítat hodnoty nových vrcholů a souvisejících hodnot (texturovací souřadnice, normálové vektory), dále je potřeba provádět samotnou teselaci, a přistupovat do textur. Tato kapitola se proto zabývá některými experimenty a pozorováními provedenými v průběhu tvorby této práce. Pro získání hodnot bylo využito GPUperfAPI, testovací sestava byla osazena grafickou kartou AMD Radeon HD5830. Před zahájením měření bylo vždy provedeno několik cvičných měření, za účelem ustálení hodnot (které mohou být v ten moment ještě ovlivněny například načítáním textury), a pro každé měření byla nastavena stejná pozice kamery - v tabulce lze vidět, že velký vliv na výkon má Fragment Shader. Backface culling je vypnut za účelem zvýšení počtu zpracovávaných vrcholů a zvýšení přesnosti měření - cílem je minimalizovat vnější vlivy, čemuž počet zpracovávaných vrcholů může napomoct.

8.1 Náročnost vykreslení modelu s displacement mappingem oproti modelu bez něj

Jedním z účelů využití principu Level Of Detail (LOD) je snížení časové náročnosti vykreslování. Na následujících tabulkách je srovnání náročnosti vykreslování původního neredukovaného modelu asteroidu (tabulka 8.1), který sloužil jako základ pro vygenerování displacement mapy redukovaného modelu, a redukovaného modelu s aplikovaným displacement mappingem (tabulka 8.2). Úroveň teselace byla nastavená tak, aby byl počet vrcholů co nejbližší.

	Median	Nejnižší naměřená hodnota
GPUTime	1.43089 s	1.42856 s
ShaderBusy	99.0897%	77.1909%
ShaderBusyVS	17.5784%	17.5017%
ShaderBusyPS	82.4233%	76.5203%
VSVerticesIn	321311	321311

Tabulka 8.1: Tabulka hodnot pro původní neredukovaný model

	Median	Nejnižší naměřená hodnota
GPUTime	1.79289	1.787
TessellatorBusy	93.2832	79.7148
ShaderBusy	99.3409%	77.9819%
ShaderBusyVS	0.0846583%	0.0759169%
ShaderBusyHS	0.0841043%	0.0797171%
ShaderBusyDS	29.1237%	28.4889%
ShaderBusyPS	71.0369%	65.5093%
VSVerticesIn	324432	324432

Tabulka 8.2: Tabulka naměřených hodnot pro redukovaný model s využitím displacement mappingu

Z tabulek lze vidět, že model redukovaný model s aplikovaným displacement mappingem je v tomto případě přibližně o 25% časově náročnější- Lze předpokládat že důvodem je nutnost počítat nové vrcholy dle souřadnic vygenerovaných teselátorem.

8.2 Náročnost výpočtu stupně teselace s měnící se vzdáleností od kamery

V části 7.5 byly popsány dva možné způsoby výpočtu stupně teselace v závislosti na vzdálenosti objektu od kamery. Jeden z nich (tabulka 8.3) je založen čistě na výpočtu využívajícím transformaci pomocných bodů ModelView maticí uvnitř shaderu, druhý (tabulka 8.4) vypočítá na straně procesoru vzdálenost objektu od kamery a následně tuto hodnotu předá jako uniform do shaderu.

	Median	Nejnižší naměřená hodnota
GPUTime	2.90867	2.895
TessellatorBusy	94.0982	88.3225
ShaderBusyVS	0.175211	0.100129
ShaderBusyHS	0.195765	0.115978
ShaderBusyDS	17.9301	17.2039
ShaderBusyPS	81.707	74.1085
VSVerticesIn	393508	393508

Tabulka 8.3: Tabulka naměřených hodnot pro dlážděnou plochu s výpočtem v shaderu

	Median	Nejnižší naměřená hodnota
GPUTime	2.95722	2.94011
TessellatorBusy	94.2512	94.19
ShaderBusyVS	0.179546	0.106648
ShaderBusyHS	0.185464	0.115613
ShaderBusyDS	17.429	11.6239
ShaderBusyPS	82.2193	81.9276
VSVerticesIn	393176	393176

Tabulka 8.4: Tabulka naměřených hodnot pro dlážděnou plochu s kombinovaným výpočtem

Oproti předpokladu se ukázala být verze s výpočtem s pomocí transformace pomocných bodů o několik ms rychlejší. S přihlédnutím k tomu že tato verze přitom provádí více výpočtů lze v tuto chvíli jen spekulovat, co může být příčinou, je ale možné že změna uniformu při vykreslení každého snímku způsobuje toto zvýšení časové náročnosti.

Kapitola 9

Popis prostředků použitých pro demonstrační aplikaci

Demonstrační aplikace je vytvořená za pomocí knihoven Qt a OpenSceneGraph. Navzdory platformové nezávislosti těchto knihoven aplikace samotná nezávislá není, z důvodu využití Windows API funkce pro získání vykreslovacího kontextu (bude ale upravena tak aby pro přizpůsobení jiné platformě ji stačilo na této platformě pouze znovu přeložit).

9.1 Qt

Qt je multiplatformní knihovna poskytující prostředky pro zobrazení a spravování uživatelského rozhraní aplikace. Poskytuje třídy pro základní uživatelské prvky, používá vlastní systém signálů a slotů pro komunikaci mezi těmito prvky a odvozenými třídami.

9.2 OpenSceneGraph

OpenSceneGraph je mutliplatformní opensource knihovna určená pro 3D aplikace. Slouží jako abstrakční vrstva nad OpenGL, takže není třeba vykreslovat jednotlivé modely, ale postačí scéna ke které jsou přiřazeny, a poskytuje metody pro načítání obsahu (modely, textury). Její hlavní přínos spočívá v uložení objektů v podobě grafu scény[1], kde jsou objekty seskupeny podle polohy, a podle vzájemného vztahu (například aby bylo auto ve stejném podstromu jako jeho kola). Kromě této organizace objektů poskytuje OpenSceneGraph i optimalizaci scény, kdy nevykresluje objekty které leží mimo scénu, a seskupí je podle podobnosti jejich StateSetů (StateSet je třída seskupující OpenGL parametry a nastavení pro daný objekt, například shader program).

[1] Graf ve smyslu grafové struktury, v tomto případě stromu

Kapitola 10

Jiné již existující aplikace využívající displacement mapping

Ačkoliv se využívání displacement mappingu v počítačových hrách zatím příliš nerozšířilo, existuje již slušné množství demonstračních aplikací i reálných. Některé z nich budou zmíněny v této kapitole, společně s několika hrami.

10.1 March of the Froblins

March of the Froblin [13] je demo vytvořené společností AMD pro grafické karty řady Radeon 4800 a rozhraní DirectX 10.1. Bylo součástí prezentace na konferenci SIGGRAPH 2008. Využívalo tehdejší teselaci, zpracovanou jako nestandardní rozšíření 3.1. Kromě teselace využívalo grafickou kartu pro další výpočty, jako pathfinding v prostoru. Jeho kladem je možnost předvedení jednotlivých prvků s předpřipraveným komentářem, a přítomnost rozsáhlého článku popisujícího jednotlivé detaily. I přes využití před- DirectX 11 / OpenGL 4.0 teselace lze spustit i na pozdějších grafických kartách Radeon.

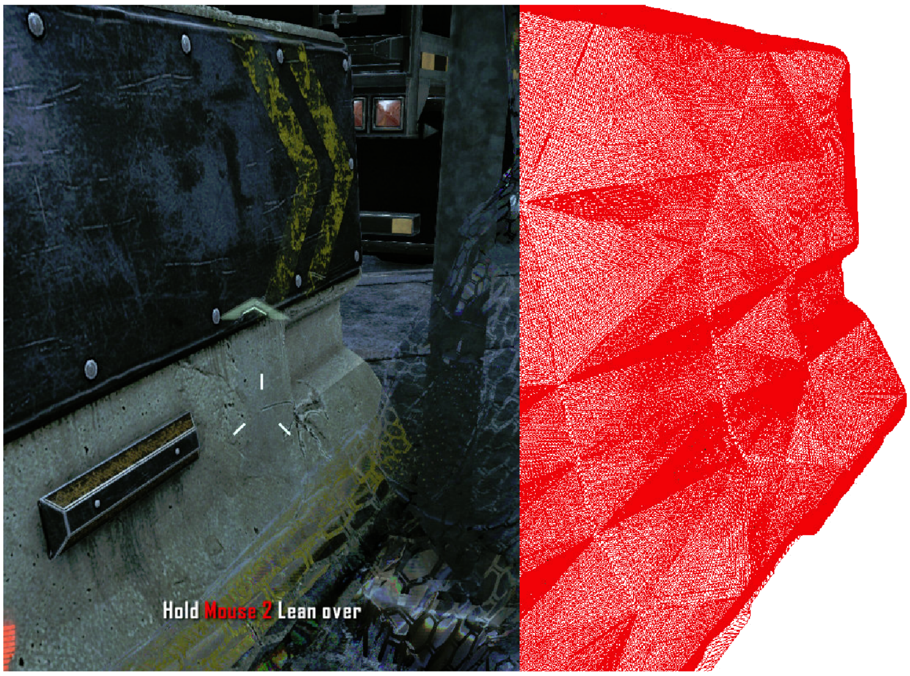
10.2 Crysis 2

Crysis 2 je počítačová hra vydaná a vyvinutá neměckou společností Crytek v roce 2011. Verze vydaná na pc běží pod DirectX 9.0c, ale byl vydán patch pro podporu DirectX 11, společně s dodatečnými texturami. Jak ale ve svém článku rozebral S. Wasson[14], implementace teselace a displacement mappingu má určité nedostatky. Objekty se teselují s uniformním stupněm teselace, takže jsou rovné plochy často zbytečně teselovány, aniž by z toho byl faktický užitek, a některé prvky, jako třeba voda, se zbytečně teselují i když je jejich geometrie skryta, a jsou pak v primitive assembly zahozeny.

10.3 Tom Clancy's H.A.W.X 2

Tom Clancy's H.A.W.X 2 je počítačová hra vydaná společností Ubisoft a vyvinutá společností Ubisoft Romania. Displacement mapping využívá pro vykreslování detailnějšího terénu. Způsob, jakým tak dělá, popsal ve svém článku I. Cantlay[9]. Využívá dvou úrovní displacement map, kdy jedna určuje základní tvar terénu (kopce, údolí), a druhá jim dodá detail.

Obrázek 10.1: March of the Froblins - jedna z komentovaných demonstrací



Obrázek 10.2: Crysis 2 - ukázka zbytečně teselovaných oblastí modelu. zdroj: [14]

Obrázek 10.3: Ukázka teselace a displacementu terénu metodou využitou v Tom Clancy's H.A.W.X 2. zdroj: [9]

Kapitola 11

Závěr

V rámci práce byly ukázány vlastnosti displacement mappingu, jeho možnosti, a úskalí. Je to metoda s poměrně specifickým využitím. Při stejné složitosti výsledné geometrie je pomalejší než vykreslování původního objektu, potřebuje moderní grafickou kartu, a více času na přípravu modelu, pokud se ve výsledné geometrii modelu nemají vyskytovat mezery. Jeho výhodou je nižší paměťová náročnost jak pro grafickou kartu, tak pro uložení na disku, případně přenos po internetu. V případě modelů v testu v části 8.1 měl neredukovaný model uložený na disku ve formátu *ive* 12 MB, kdežto redukovaný model měl jen několik desítek KB, a textura ve formátu TGA 1 MB. Protože pro samotné vykreslování nepotřebuje model žádné úpravy, lze stejnou uloženou geometrii použít pro vykreslení bez displacement mappingu jako běžný level of detail. Použití displacement mappingu jako level of detail (tedy dynamické změně úrovně teselace v závislosti na vzdálenosti od kamery) má dvě slabiny - tou menší je problém s návazností různých dílů modelu, ale to lze vyřešit podobnou metodou, jako byla ta zmíněna v části 7.6. Tou horší slabinou je samotný vzhled objektu při posunu směrem ke kameře, nebo od kamery, kde jsou na zobrazovaném modelu pozorovatelná cuknutí (equal spacing), nebo vlnění (oba typy fractional spacing). Může být také na místě otázka, nakolik užitečné je použít na level of detail, jehož cílem je obvykle snížit časovou náročnost vykreslení scény, metodu, která je časově náročnější. Otázku použití displacement mappingu je tedy vhodné důkladně posoudit, a ověřit si zda očekávané přínosy použití nebudou zastíněny jeho nevýhodami a slabinami.

GPUperfAPI se ukázalo být poměrně užitečné díky tomu, že v případě problémů s výkonem mohou některé z jeho čítačů naznačit podstatu problému. Jeho poměrně velkou nevýhodou je závislost na platformě. Krátce byla otestována i jeho obdoba NVPerfKIT od společnosti NVidia, při jehou používání se ale vyskytly přetrvávající problémy (vyskytovaly se i v úkázkové aplikaci přiložené k NVPerfKITu) které znemožnili jeho bližší prozkoumání. Ačkoliv jsou obě knihovny od různých společností pro různé značky GPU, jejich rozhraní je do určité míry podobné, a mohlo by, v případě aplikace, která by se například mohla do určité míry díky získaným údajům přizpůsobit konkrétnímu GPU, být možné vytvořit wrapper nad oběma knihovnami.

Kapitola 12

Přílohy

12.1 Obsah DVD

- Textová část
- Program - binární
- Program - zdrojový kód
- Readme.pdf

Literatura

[1] R600 (ASIC). http://en.wikipedia.org/wiki/Radeon_HD_3000_Series#Hardware_tessellation, 13 May 2013? [cit. 2013-05-24].

[2] Tangent space. http://en.wikipedia.org/wiki/Tangent_space, 19 April 2012? [cit. 2013-05-28].

[3] TruForm. https://en.wikipedia.org/wiki/TruForm, 19 April 2012? [cit. 2013-05-28].

[4] GPU Gems 2. 2005-01-01 [cit. 2013-01-08].

[5] Rendering Pipeline Overview. http://www.opengl.org/wiki/Rendering_Pipeline_Overview, 2012-11-02 [cit. 2013-01-08].

[6] Bailey, M.: Tessellation Shaders. http://web.engr.oregonstate.edu/ mjb/cs519/Handouts/tessellation.1pp.pdf, [cit. 2013-05-27].

[7] Blinn, J. F.: Simulation of wrinkled surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM Press, 1978, ISSN 0097-8930, s. 286--292, doi:10.1145/800248.507101. URL http://dx.doi.org/10.1145/800248.507101

[8] Boubekeur, T.; Alexa, M.: Phong Tessellation. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, ročník 27, 2008.

[9] Cantlay, I.: DirectX 11 Terrain Tessellation. 2011.

[10] Cook, R. L.: Shade Trees. 1984. URL http://www.gdv.informatik.uni-frankfurt.de/lehre/ss2007/GDV/Uebung/Paper/99-1-cook-shadetrees.pdf

[11] Kaneko, T.; Takahei, T.; Inami, M.; aj.: Detailed shape representation with parallax mapping. In *In Proceedings of the ICAT 2001*, 2001, s. 205--208.

[12] Lobel, R.: Realtime Per-Pixel Displacement Mapping on Arbitrary Geometry. 2004. URL http://www.divideconcept.net/papers/RPPDM-RL04.pdf

[13] Shopf, J.; Barczak, J.; Oat, C.; aj.: March of the Froblins: simulation and rendering massive crowds of intelligent and detailed creatures on GPU. In *Proceeding SIGGRAPH '08 ACM SIGGRAPH 2008 Games*, 2008, s. 52--101.

[14] Wasson, S.: Crysis 2 tessellation: too much of a good thing? 16 August 2011? [cit. 2013-05-26].
URL http://techreport.com/review/21404/crysis-2-tessellation-too-much-of-a-good-thing