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

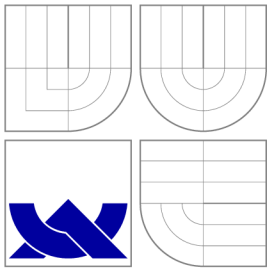
DISPLACEMENT MAPPING S VYUŽITÍM VIRTUÁLNÍCH TEXTUR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

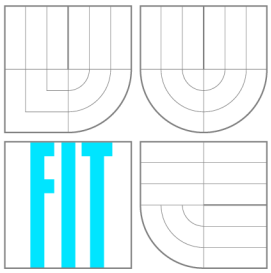
AUTOR PRÁCE
AUTHOR

JOZEF ČULEN

BRNO 2015



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

DISPLACEMENT MAPPING S VYUŽITÍM VIRTUÁLNÍCH TEXTUR

DISPLACEMENT MAPPING WITH VIRTUAL TEXTURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JOZEF ČULEN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. STARKA TOMÁŠ

BRNO 2015

Abstrakt

Tato práce se zabýva technikou přidávání nerovností povrchu modelů nazvanou displacement mapping s využitím virtuálních textur. Displacement mapping bude výkonnostně i vizuálně porovnán s normal mappingem. Práce dále obsahuje stručný přehled některých světelných modelů a předcházejících metod pro simulování nerovností povrchů.

Abstract

This thesis describes technique for adding roughness to surface of models called displacement mapping with virtual textures. Efficiency and visual quality of displacement mapping will be compared to normal mapping. Thesis also briefly describes some of lightning models and former techniques for adding details to smooth surface.

Klíčová slova

displacement mapping, virtuálni textury, OpenGL, teselace

Keywords

displacement mapping, virtual textures, OpenGL, tesselation

Citace

Jozef Čulen: Displacement mapping s využitím
virtuálních textur, bakalářská práce, Brno, FIT VUT v Brně, 2015

Displacement mapping s využitím virtuálních textur

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Starky.

.....
Jozef Čulen
20. května 2015

Obsah

Kapitola 1

Úvod

Jedným z hlavných prvkov realistickej realtimovej počítačovej grafiky je interakcia svetla s povrchom objektov. Už dlhú dobu nie je problém efektívne renderovať rôzne objekty tvorené napríklad polygonmi alebo rôznymi parametrickými povrchmi. Ich spoločný problém a celkovo počítačovej grafiky všeobecne spočíva v tom, že vyzerajú uhladene a umelo. Dôvodom je to, že objekty v reálnom svete majú makro aj mikroskopické nerovnosti, oderky a rôzne poškodenia povrchu a materiálu, z ktorého je samotný objekt tvorený. Svetlo sa na týchto nerovnostiach povrchu láme a aj tie najmenšie nedokonalosti či nečistoty povrchu majú významný vplyv na to ako človek vníma tento objekt.

Táto práca sa zaoberá tzv. displacement mappingom, ktorý mení povrch objektov a pridáva tak nerovnosti a nedokonalosti vo väčšej mierke. Táto technika je spojená s dynamickou tesseláciou topológie modelov s využitím virtuálnych textúr.

Nasledujúca kapitola sa zaoberá vybranými svetelnými modelami, ktoré sa používajú v počítačovej grafike v reálnom čase. Tretia kapitola obsahuje rozbor techník simulúcich nerovnosti povrchov modelov. Štvrtá kapitola popisuje techniku displacement mapping. Piata kapitola rozoberá problematikou virtuálnych textúr. V šiestej kapitole je rozbrány návrh a implementácia. Dosiahnuté výsledky sú prezentované v Siedmej kapitole.

Kapitola 2

Svetelné modely

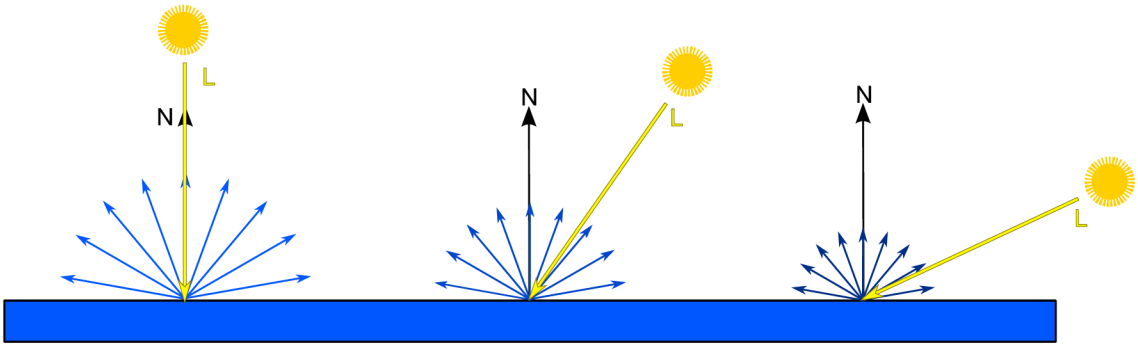
Úroveň fotorealistickosti realtimovej grafiky sa za posledných pár rokov, vďaka hernému priemyslu, značne zvýšila ale stále sme ďaleko od skutočnej fotorealistickosti. Príklad súčasnej úrovne realtimovej grafiky je zobrazený na obrázku 2.1. Vzhľadom na to, že súčasné grafické karty ešte nie sú schopné realtimovo simulovať emitovanie a interakciu fotónov v inej než triviálnej scéne, sme odkázaní používať na túto úlohu tzv. offline renderovacie programy. Offline renderovacie programy nerenderujú v reálnom čase ale výpočet osvetlenia scény pre jednu snímku môže trvať aj niekoľko hodín.

Obrázek 2.1: Scéna z hry Kingdom come: Deliverance vytvorenej v hernom engine Cryengine 3.

Pri renderovaní v reálnom čase je potrebné využívať techniky, ktoré aproximujú interakciu fotónov svetla so svetom. Nevýhodou tohto prístupu je fyzikálna nepresnoť, prípadne výsledok absolútne nezlučitelný s reálnym chovaním svetla. Na druhú stranu nám to dovoľuje renderovať komplexné scény, ktoré s typickým offline renderovacím programom môžu trvať niekoľko hodín renderovacieho času na snímku.

2.1 Lambertov model

Tento model je vhodný na simulovanie materiálov s hrubým alebo matným povrchom, taký ktorý nemá viditeľné odlesky. Princíp celého modelu je založený na Labertovom kosínusovom zákone, ktorý hovorí že intenzita svetla je proporcinálna ku kosínusu uhlu medzi vektorom svetla a normálou povrchu. To znamená, že pokiaľ svetlo svieti na povrch pod malým uhlom, osvetlí väčšiu časť povrchu ako keď svieti naň kolmo. Tým pádom intenzita svetla na jednotku plochy je menšia. Intenzita nasvetlenia je závislá len na polohe svetla a nie na polohe pozorovateľa.



Obrázek 2.2: Vplyv intenzity odrazu svetla v závislosti od uhlu dopadnutého svetla.

Na výpočet je teda potrebné mať vektory L a N, ktoré musia byť normalizované. Odrazivosť materiálu je označená ako R a L je intezita svetla. Skalárny súčin vektorov N a L je potrebné orezať len na kladné hodnoty aby sme nezískavali záporné intezity svetla. Príklad je na obrázku 2.3. Vzorec je:

$$I = LRmax(0, (N \cdot L)) \tag{2.1}$$

Obrázek 2.3: Ukážka lambertovho svetelného modelu.

2.2 Phongov model

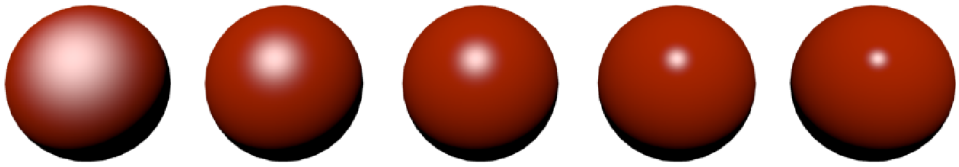
Jeden z najznámejších svetelných modelov, ktorý ma výborný pomer výpočetná náročnost ku kvalite zobrazenia avšak za cenu toho, že výsledok je fyzikálne nepresný. Predstavil ho B.T. Phong v roku 1975[6]. Materiály majú tri zložky - ambient, difúzna, zrkadlový odraz

(specular reflection). V nich sú uložené farby, ktoré materiál odráža. Rovnaké vlastnosti majú aj svetlá.

Ambient zložka materiálu je v podstate snaha o simuláciu globálneho osvetlenia. Dôvodom použitia tejto zložky je ten, že ak máme niektoré polygony mimo dosahu svetla tj. nie sú priamo osvetlené žiadnym svetelným zdrojom, boli by úplne čierne. Ambient zložka sa tak pričíta k finálnej farbe pixelu ak je difúzna zložka čierna a výsledkom je svetlejšia scéna ale bez čiernych oblastí kam nedočiahne svetlo.

Difúzna zložka materiálu je farba, ktorú materiál odráža ak naň svieti svetelný zdroj.

Zložka zrkadlovej odrazivosti materiálu určuje farbu zrkadlového odrazu. V skutočnosti sa táto zložka rozdeľuje na dve časti a to farbu a silu odrazu. V reálnom svete má každý bežný materiál, pokiaľ nepovažujeme čiernu dieru za bežný materiál, nejakú mieru odrazivosti. Farba je jednoznačná, určuje farbu odrazu. Miera odrazivosti nám pomáha rozlišovať či sa jedná o lesklý plast alebo koženú bundu. Na obráku 2.4 je vidieť vplyv tejto vlastnosti na vzhľad materiálu. Zjednodušene sa dá miera odrazivosti označiť za ostrosť odrazu.



Obrázek 2.4: Vplyv ostrosti odrazu materiálu na jeho vzhľad. Smerom zľava od nízkej po vysokú hodnotu.

Výsledná farba pixelu I sa teda rovná súčtu uvedených 6 zložiek. Rovnica na tento výpočet vyzerá takto:

$$I = I_a + I_d + I_s \tag{2.2}$$

Hodnoty I_a, I_d a I_s predstavujú zložky ambient, diffuse a specular reflection už sčítané zo zložiek materiálu a svetiel. Ambient zložka I_a sa teda vypočítá nasledovne:

$$I_a = R_a * L_a, \qquad \text{kde} \quad 0 \leq R_a \leq 1$$

Hodnota R_a predstavuje ambient zložku materiálu a L_a ambient zložku svetla.

Diffuse zložka I_d sa počíta ako:

$$I_d = R_d * L_d max(0, N \cdot L)$$

Hodnoty R_d a L_d sú difúzne zložky materiálu a svetla. Vypočíta sa skalárny súčin vektorov L a N ak je vyššsí ako nula vynásobí sa ním výsledna farba. Táto časť je totožná s Lambertovým modelom.

Specular reflection zložka I_s sa v praxi teda získa:

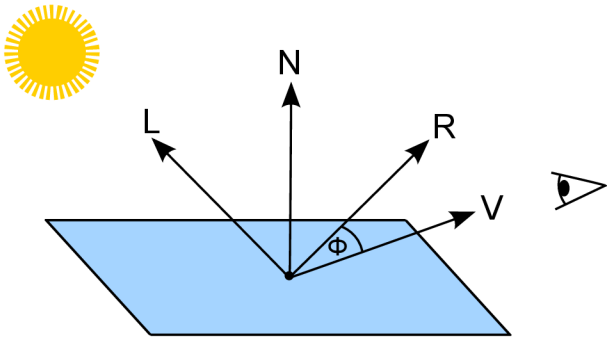
$$I_s = R_s * L_s max(0, R \cdot V)^{\alpha}$$

Hodnoty R_s a L_s sú farby odrazov modelu a svetla. α je phong exponent, ktorý určuje ostrosť odrazu a $R \cdot V = \phi$ a to je uhol medzi vektormi R a V.

Výsledná rozpísaná rovnica na výpočet phongovho osvetlovacieho modelu je teda:

$$I = R_a * L_a + R_d * L_d max(0, N \cdot L) + R_s * L_s max(0, R \cdot V)^{\alpha} \tag{2.3}$$

Spomínané vektory sú zobrazené na obrázku 2.5.



Obrázek 2.5: Phongov nasvetlovací model.

V praxi sa väčšinou používa alternatíva phongovho modelu, ktorú upravil James F. Blinn[2]. Blinn-Phong metóda sa líši pri výpočte specular reflection zložky, kde sa namiesto skalárneho súčinu vekorov R a V vypočíta skalárny súčin vektorov H a N. Vektor H sa nachádza v strede medzi vektormi L a V a vypočíta sa nasledovne:

$$H = \frac{L + V}{|L + V|}$$

V takomto prípade nie je potrebné prepočívať vektor R, ktorý je zavislý na vektore N. Vektor H nie je závislý na vektore N, tým pádom ušetrí veľa výpočtovov pri smerových svetlách a konštantných uhloch pohľadu.

2.3 Cook-Torrance model

Narozdiel od spomínaných modelov, tento nasvetlovací model sa snaží byť fyzikálne presnejší. Autori tohto modelu Robert L. Cook a Kenneth E. Torrance ho predstavili vo svojom článku "A reflectance model for computer graphics"[4].

Princíp tohto modelu spočíva v tom, že sa s plochami narába ako keby pozostávali z veľkeho množstva malých plošiek. Orientácia týchto plošiek určuje odrazivosť materálu. Lesklé materiály majú podobnú orientáciu normál týchto plošiek medzi sebou a pri matných materiáloch je ich orientácia veľmi rozdielna. Materiál s takýtmo hrubým povrchom odráža dopadané svetlo do rôznych smerov a preto sa zdá, že nie je tak lesklý.

Zjednodušená rovnica na vypočet jednoduchého Cook-Torrance svetelného je nasledovná:

$$r = ambient + \sum_{l} (N \cdot L) * \Big(K + (1 - K) * r_s\Big) \tag{2.4}$$

Písmeno K je svetlo, ktoré sa odráža z diffúznej zložky materiálu a r_s je zložka zrkadlového odrazu, ktorá sa počíta pre každé svetlo. Výpočet r_s je nasledovný:

$$r_s = \frac{F * D * G}{\pi * (N \cdot L)(N \cdot V)}$$

Normála je označená ako N, L je vektor svetla, V je pozorovací vektor. Zrkadlový odraz teda závisi od Fresnel faktoru (F), hrubosti povrchu (D) a od geometrického útlmu (G).

Fresnel faktor (F) určuje, koľko svetla sa odrazí od povrchu a koľko svetla prenikne povrchom. Vzorec navrhnutý na výpočet fresnel zložky v pôvodnom článku je výpočetne náročný a preto sa často používa aproximácia, ktorú predstavil Ch. Schlick[7]. Schlickov vzorec vyzerá nasledovne:

$$F_\lambda(u) = f_\lambda + (1 - f_\lambda)(1 - u)^5 \tag{2.5}$$

Hrubosť povrchu (D), teda distribúcia orientácie mikroplošiek určuje hrubosť odrazu svetla. Túto distribúciu možeme vypočítať napríklad Beckmannovov metódou [1]. Vzorec vyzerá nasledovne:

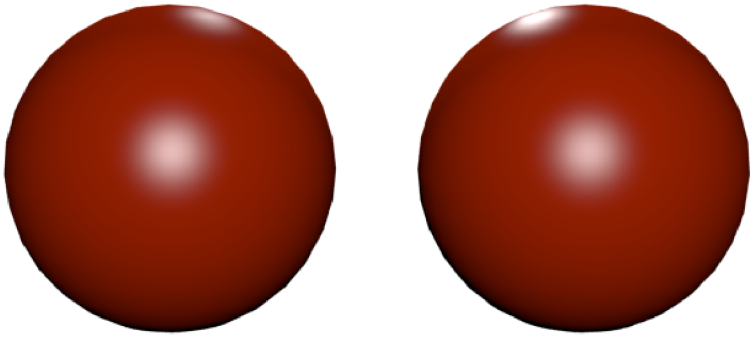
$$D = \frac{1}{\pi * m^2 * (N \cdot H)^4} e^{\left(\frac{(N \cdot H)^2 - 1}{m^2 * (N \cdot H)^2}\right)} \tag{2.6}$$

Písmeno m určuje hrubosť povrchu. Polovičný vektor H sa vypočíta rovnako ako pri ostatných modeloch a to sčítaním vektorov L a V.

Niektoré mikroplošky môžu odrážené svetlo zablokovať, prípadne znižiť jeho intenzitu. Geometrický útlm (G) angl.Geometric attenuation je teda číslo od 0 do 1, ktoré predstavuje proporcionálne množstvo svetla, ktoré zostalo po týchto interferenciách. Vzorec na výpočet je nasledovný:

$$G = min\left(1, \frac{2 * (N \cdot H) * (N \cdot V)}{V \cdot H}, \frac{2 * (N \cdot H) * (N \cdot L)}{L \cdot H}\right) \tag{2.7}$$

Rozdiel je možné pozorovať na obrázku 2.6. Odrazy svetla na okraji objektu sú viditeľne jasnejšie.



Obrázek 2.6: Porovnanie Phongovho model (vľavo) a Cook-Torrance modelu (vpravo) s dvomi bodovými svetlami umiestnenými medzi guľami, jedno v predu a jedno vzadu.

Kapitola 3

Metódy simulovania nerovností povrchu

Pre požiadavky realistickejšej realtime grafickej reprezentácie, a zároveň pre relátivne nízky výkon grafických kariet pri renderovaní obrovského množstva polygonov museli vzniknúť metódy, ktoré simulujú nerovný povrch geometrie modelov s relatívne nízkym počtom polygonov. Nerovnosti povrchu sú vačsinou definované textúrou. Vzhľadom na to že nerovnosti sú brané do úvahy len pri výpočte osvetlenia, sú tieto metódy menej náročné na pamäť a výkon grafickej karty.

Nevýhoda týchto metód je v tom, že ajkeď to vyzerá, že povrchy su detialné, geometria navyše tam stále nie je, a teda procesy ako napríklad vypočet fyzikálnych interkacií nebude fungovať tak ako by bolo očakávané.

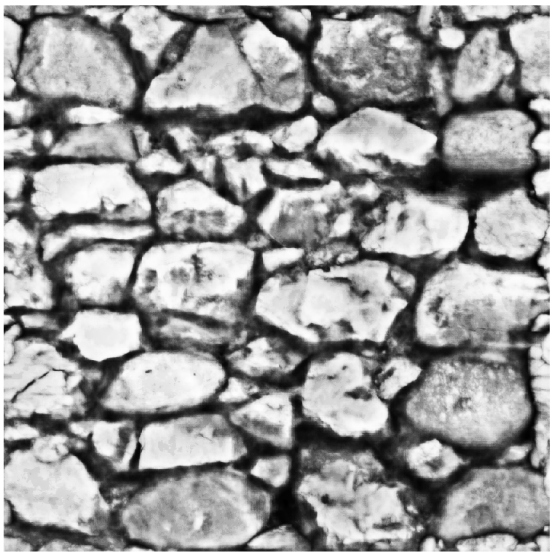
3.1 Bump mapping

Základnú formu tejto metódy publikoval James F. Blinn vo svojom článku "Simulation of wrinkled surface"[3]. Cieľom jeho článku bolo predviesť metódu pre napodobovanie vysokofrekvnečných detailov a nerovností na parametrických povrchoch(patch), ktoré vyzerajú veľmi umelo vzhľadom na ich hladkosť. Definovať tieto nerovnosti pomocou patchov by bolo hardwarovo veľmi náročné. Blinn navrhuje použiť funkciu alebo textúru nerovnstí tzv. bump funkciu alebo textúru pre uchovanie týchto detailov. Táto metóda sa da využiť aj v dnešnej realtimovej počítačovej grafike, ktorá využíva trojuholníky resp. polygony na definíciu modelov.

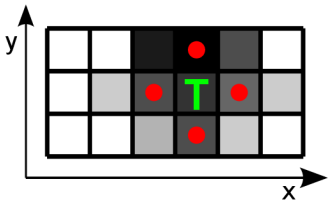
Princíp tejto metódy spočíva v pozmenení normál povrchu pomocou bump textúry vo fragment shadery pri výpočte osvetlenia. Bump textúra je čiernobiela viď. obrázok 3.1 a teda intezita každého texelu určuje výšku nerovnosti, preto sa môže nazývať ako tzv.výšková textúra teda v angl. height map. Majme texel T ku ktorému je potrebné vypočítať normálu vzhľadom na zmenu výšky. Výšku získame vypočítaním rozdielov susedných pixelov viď. obrázok 3.2 a vzorec na výpočet je:

$$\begin{aligned} rozdiel\_x &= texel[T.x+1][T.y] - texel[T.x-1][T.y]; \\ rozdiel\_y &= texel[T.x][T.y+1] - texel[T.x][T.y-1]; \\ T.normala &= povodna\_normala + rozdiel\_x * rozdiel\_y. \end{aligned} \tag{3.1}$$

Výsledkom bude nová normála $T.normala$ vypočítaná z intenzity pixelov bump textúry na výpošet osvetlenia. Normála sa dá vypočítať aj z väčšieho okolia.



Obrázek 3.1: Príklad bump textúry. Vyššia intenzita farby určuje vačšiu výšku texelu.



Obrázek 3.2: Výpočet normály z bump textúry.

Výhodou tejto metódy je rýchlosť a jednoduchá tvorba bump textúr. Nevýhodou je nemožnosť uloženia mikro detailov, pretože dnešné grafické karty sú obmedzené maximálnou veľkosťou textúry s ktorou dokáže pracovať. Súčasné grafické karty dokážu maximálne pracovat s textúrami veľkosti 8192*8192 prípadne 16384*16384 pixelov. Ďalšou nevýhodou je, že silueta objektu sa nemení a ani výpočet tieňov, ktoré by mali nerovnosti vrhať, nie je možný. Preto je vhodné používať túto metódu na veľmi jemných nerovnostiach, kde k zmene siluety neprichádza.

3.2 Normal mapping

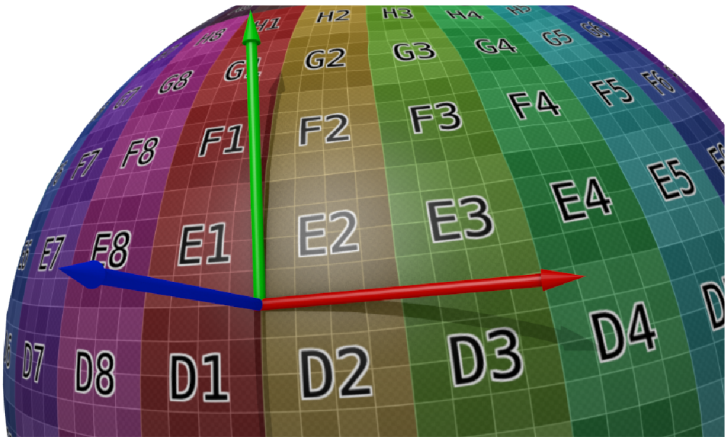
Normal mapping je rozšírenie bump mappingu, kde sa namiesto bump textúry používa normálová textúra. Normálová textúra uchováva v každom texely vektor normály v RGB komponentoch obrázku. Ide o farebný obrázok viď obrázok 3.4. Týmto sa vyhneme výpočtu normály z bump textúry. Červený kanál obrázku uchováva X-ovú súradnicu, zelený kanál uchováva Y-ovú súradnicu a modrý kanál Z-ovú súradnicu vektoru normály. Pretože väčšina normál smeruje k pozorovateľovi, normálové textúry majú vždy modrý odtieň.

V drvivej väčšine prípadov sú súradnice uložené v tzv. dotyčnicovom priestore angl. tangent space. Je to jeden zo súradnicových systémov tak ako world space alebo object space. Tento priestor si môžeme predstaviť ako plochu kolmú na vrchol tvorenú vektormi T a B, ktoré vlastne korešpondujú s textúrovacími vektormi U a V. Nákres tejto plochy a vektorov ktoré ju tvoria je na obrázku 3.3. Na ňom sú znázornené vektory N (modrý),

ktorý je normálou vrcholu, T (čerevný) čo je tangent, ktorý je zarovnaný s textúrovacou osou U, B (zelený) čo je bitangent a je zarovnaný texturovacou osou V.

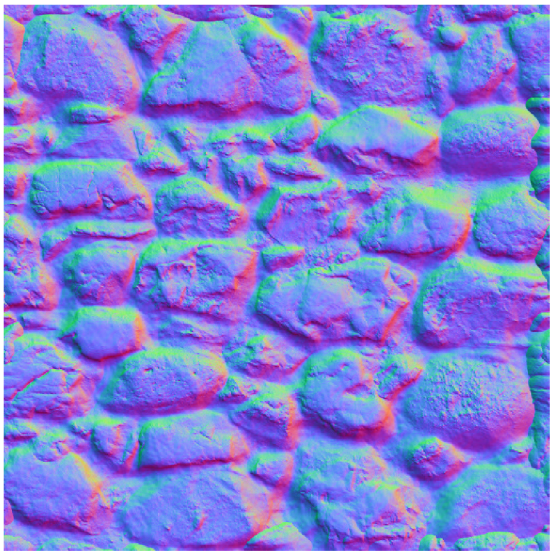
Na prepočet do dotyčnicového súradnicového systému sa využíva tzv. TBN matica. Skratka TBN je odvodená od slov tangent, bitangent a normal. To napovedá ako je táto matica vytvorená:

$$TBN = \begin{pmatrix} T.x & T.y & T.z \\ B.x & B.y & B.z \\ N.x & N.y & N.z \end{pmatrix}$$



Obrázek 3.3: Znázornenie dotyčnicového priestoru.

Dôvod použitia tohto súradnicového systému pri ukladaní normál do normálových textúr, je flexibilita. Namiesto toho aby sa prepočítavali všetky normály pri každom pohybe kamery alebo objektu sa tak budú prepočítavať svetlá a kamery do dotyčnicového priestoru, čo je omnoho efektívnejšie.

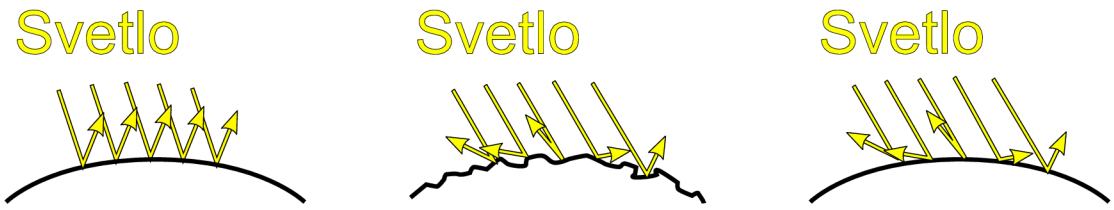


Obrázek 3.4: Príklad normálovej textúry. Modrý kanál obsahuje X-ovú súradnicu, zelený kanál obsahuje Y-ovú súradnicu a modrý kanál obsahuje Z-ovú súradnicu.

Jednotlivé farebné komponenty nadobúdajú v OpenGL hodnoty 0 až 1. Pretože súradnice môžu byť aj záporné, je potrebné ich previesť do intervalu -1 až 1. Na to slúži vzorec:

$$2 * texel.rgb - 1.0 \tag{3.2}$$

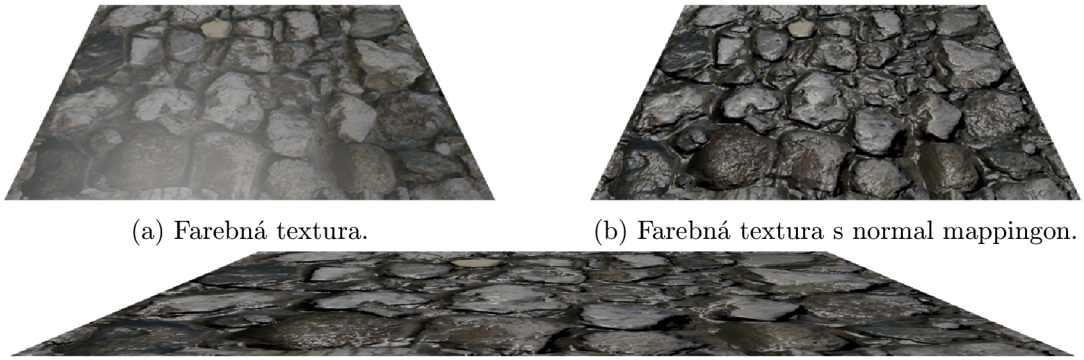
Po prepočte získame 3D vektor, ktorý určuje smer normály. S touto novo vypočítanou normálou sa následne vypočíta osvetlenie. Diagram tejto techniky je na obrázku 3.5.



(a) Hladký povrch objektu (b) Nerovný povrch objektu (c) normal mapping

Obrázek 3.5: Interakcia svetla s rôznym povrchom objektov.

Výhodou tejto metódy je väčsia flexibilita oproti bump mappingu, pretože je možné explicitne určiť smer normály pre každý texel textúry. Nevýhoda normalových textúr, je v tom že sa nedajú jednoducho kresliť a spravidla bývaju generované nástrojom z bump textúr, z rovnakého modelu s vysokým počtom polygonov obsahujúci požadované detaily alebo z reálnych fotografií. V bump a normal mappingu, pri pohľade na povrch pod ostrým uhlom je vidieť, že detaily povrchu tam nie sú a povrch je plochý viď. obrázok 3.6.



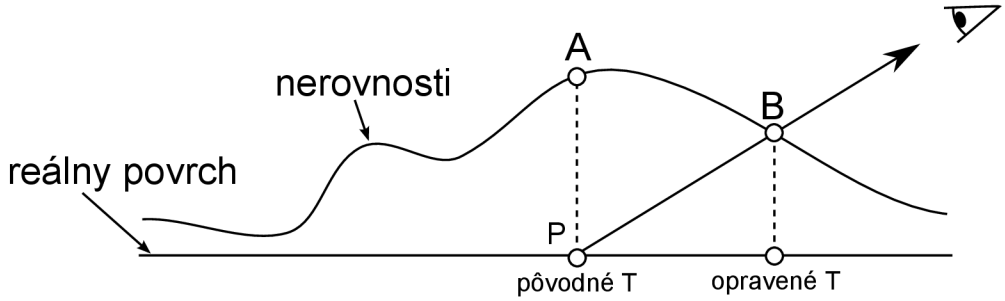
(a) Farebná textura. (b) Farebná textura s normal mappingon.

(c) Plochý povrch s normal mappingom.

Obrázek 3.6: Účinnosť normal mappingu po ostrým uhlom.

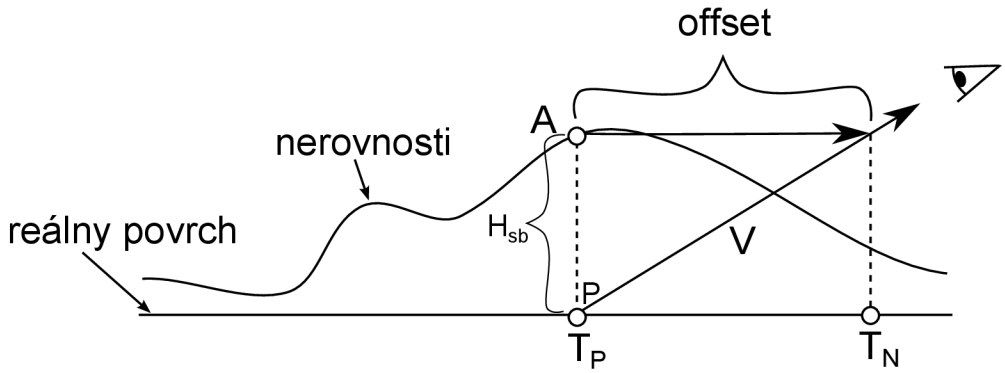
3.3 Parallax mapping

Pri použití predchádzajúcich techník pozorujeme nedostatok, ktorý sa prejavuje "sploštovaním"detialov pri ostrých uhloch pohľadu na plochu. Problém spôsobuje fakt, že ak by bol povrch naozaj nerovný, tak pri pohľade na bod P, by sme mali vidiet bod B ale namiestoho toho vidíme bod A. Tento fakt spôsobuje "plochosť"ostatných techník. Parallax mapping sa snaží tomuto predísť posunom texturovacích súradníc. Technika bola predstavená v práci "Detailed shape representation with parallax mapping"[5]. Cieľom je teda posunúť textúrovacie súradnice T bodu A na textúrovacie súradnice bodu B. Princíp tohto fenoménu je na obrázku 3.7.



Obrázek 3.7: Princíp parallax mappingu.

Pre výpočet tohto posunu (offset) sú potrebné tri komponenty: pôvodná textúrovacia súradnica, výška nerovnosti v pôvodnom bode a vektor v dotyčnicovom priestore smerujúci od daného pixelu k pozorovateľovi. Na obrázku 3.8 je naznačený princíp výpočtu offsetu. Na reprezentáciu výšky nerovností sa využíva štandadná výšková textúra, ktorá obsahuje hodnoty 0 až 1 pre každý texel.



Obrázek 3.8: Princíp parallax mappingu.

Pre výpočet offsetu textúrovacích súradniciach v bode P, vektor pohľadu V musí byť najprv normalizovaný. Výška H pôvodného bodu T_P je získaná z výškovej textúry. Pretože výšky v texeloch nadobúdaju hodnoty od 0 po 1, je potrebné ich zväčšiť (scale) a posunúť (bias) aby lepšie reprezentovali fyzickú štruktúru povrchu materiálu, ktorý sa snažíme napodobiť. Vzorec na výšku teda vyzerá nasledovne:

$$H_sb = height_texel.r * scale + bias \tag{3.3}$$

Pre výpočet je teda potrebné preložiť vektor paralélne s reálnym povrchom modelu presne nad bod P a od toho bodu až do vektoru pohľadu V. Tento nový vektor je offset, ktorý je možný pripočítať k pôvodným textúrovacím súradniciam a získame posunuté textúrovacie súradnice. Vypočet podľa:

$$T_N = T_P + (H_{sb} \cdot V_{\{x,y\}}/V_{\{x\}}) \tag{3.4}$$

Toto je najzákladnejšia forma parallax mappingu. Nie je bezchybná, technika predpokladá, že body T_P a T_N majú rovnakú výšku a stále sa objavujú artefakty pri pohľadoch pod nízkym uhlom. Preto vznikli metódy, ktoré sa sa snažía toto minimalizovať. Parallax mapping s obmedzovaním offsetu, nedovoľuje aby bol offset váčší ako výška bodu. Vznikli aj

iteratívne metódy, ktoré sa snažia uhádnuť správny posun vysielaním lúča. Na tomto princípe napríklad fungujú iteratívne varianty parallax mapping-u. Neskôr sa objavili metódy na aproximáciu tieňov, ktoré by vrhali nerovnosti povrchu. Porovnanie normal mappingu a parallax occlusion mappingu je na obrázku 3.9.

Obrázek 3.9: Porovnanie parallax mappingu a normal mappingu (zdroj[1]). Pri pohľade na normal mapping napravo je vidieť, že dlážka plochá. Pri parallax occlusion mappingu-u je jasne vidieť, že dlaždice sú vyvýšené.

[1] http://www.amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/I3D2006-Tatarchuk-POM.pdf

Kapitola 4

Displacement mapping

Táto technika sa odlišuje od predchádzajúcich tým, že reálne mení pozíciu vrcholov. Nejde teda o žiadne aproximácie pri výpočte osvetlenia. Vrchol sa posunie po normále o relatívnu vzdialenosť získanú z displacement textúry. Táto textúra je identická s bump textúrou. Intenzita pixelu určuje úroveň posunutia.

Samotný displacement mapping bez dostatočného množstva vrcholov neurobí nič viac, ako že existujúce vrcholy posune po normále. To znamená, že z jedného trojuholníka tvoreného tromi vrcholmi získame len trojuholník s posunutými vrcholmi. Preto je potrebné mať dostupný dostatočný počet vrcholov, ktoré majú reprezentovať požadované detaily z height textúry. Z toho vyplýva, že model už dostatočný počet vrcholov musí obsahovať alebo sa vrcholy budu generovať za behu programu.

4.1 Tesselácia

OpenGL 4.0 a vyššie verzie nám umožnujú deliť primitíva na viac menších primitív. Tento proces sa nazýva teselácia angl. tesselation. Skupina vrcholov vytvorených pri teselácii sa nazýva patch. Vrcholy pôvodnéheho primitíva sa nazývaju kontrolné vrcholy. Teselácia sa skladá z troch úrovní, ktoré ovládajú ako a čo sa bude teselovať:

1. Tesselation control shader
2. Tessellation primitive generation
3. Tesselation evaluation shader

Tesselation control shader (TCS) je nepovinná úroveň a má dve úlohy. Určuje množtvo teselácie, ktoré by patch mal mať a vykonáva špeciálne transformácie na kontrolných bodoch. TCS pridáva alebo uberá počet vrcholov. OpenGL špecifikácia vyžaduje aby minimálny podporovaný počet vrcholov v jednom patchi bol 32. Maximálny počet nie je určený.

Tessellation primitive generation (tesselator) vykonáva samotnú tvorbu nových vrcholov podla parametrov definovaných v TCS. Je to pevná úroveň a nedá sa upravovať. Výsledkom je nový patch, ktorý je odoslaný do tesselation evaluation shader (TES). Vrcholy sa pri generovaní umiestňujú podľa nastavení rozloženia v TES.

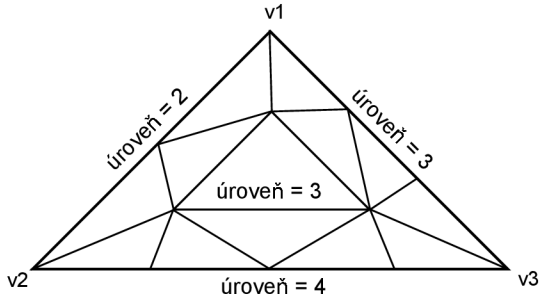
Tesselation mód určuje na aký typ primitíva sa patch rozdelí pred odoslaním na rasterizáciu. Tento mód sa nastavuje v TES a može byť jeden z *quads*(štvoruholníky), *triangles*(trojuholníky) alebo *isolines*(úsečky). Tesselation mód okrem určovania ako sa majú primitíva generovať, určuje aj to ako sa má interpretovať vstupná premenná `gl_TessCord`

v TES. V tomto shadery je vhodné vykonávať samotný displacement mapping a počítať pomocné vektory pre výpočet osvetlenia. Pri teselácii sa posúva nový bod po interpolovanej normále pôvodných bodov. Vzorec pre tento posun je nasledovný:

$$nova\_pozicia = (int\_normala * posun * uroven) + povodna\_pozicia \tag{4.1}$$

Kde *int_normala* je interpolovaná normála, *posun* je vzdialenosť posunu po normále získaná z texelu displacement textúry, *uroven* je uživateľský faktor násobenia posunu a *povodna_pozicia* predstavuje pôvodnú pozíciu vrcholu.

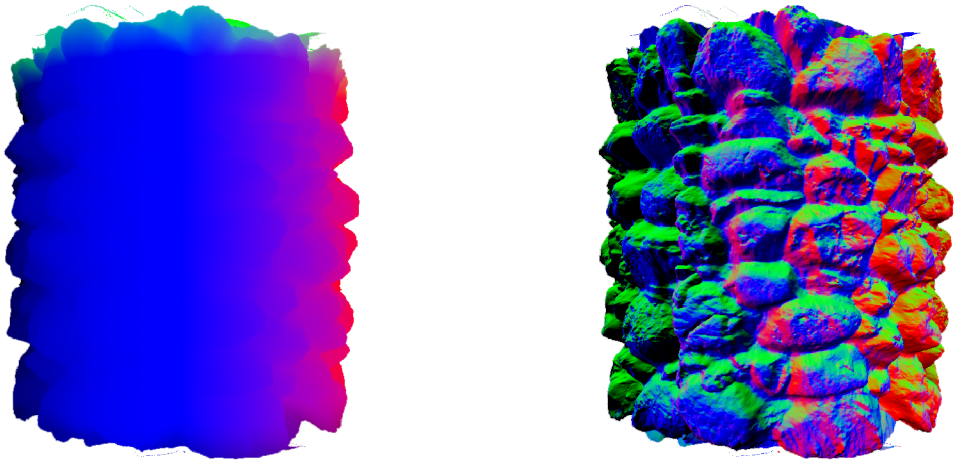
Množstvo teselácie sa nastavuje zvlášť pre každú hranu primitíva a zároveň aj pre vnútro primitíva. Nastavenie úrovne teselácie hrán sa nazýva vonkajšia úroveň teselácie a pre nastavenie vnútornej úrovne tesselácie slúži vnútorná úroveň teselácie. Kedže hrán je viac, obidve úrovne majú viac hodnôt. V závislosti od zvoleného typu primitíva sa bere do úvahy relevantný počet hodnot. Pri triangle sú to 3 hodnoty vonkajšej úrovne teselácie a jedna hodnota vnútornej teselácie. Pri quad sú to 4 hodnoty vonkajšej úrovne tesselácie a dve hodnoty vnútornej teselácie. Príklad teselácie trojuholníka zbrazuje obrázok 4.1.



Obrázek 4.1: Príklad tesselácie trojuholníka.

4.2 Výpočet normál

Veľkým problémom je výpočet normál teselovaných a posunutých vrcholov, pretože normály pôvodných vrcholov už nie sú platné. Vzhľadom na to, že žiadny typ shader-u nemá prístup k susedným vrcholom tak sa naskytujú dve možnosti ako normály vypočítať. Využiť bump alebo normal mapping. Kedže sa budú počítať normály pri výpočte osvetlenia, úroveň teselácie nie je podstatná.



(a) Normály po tesselácii a displacement mappingu.

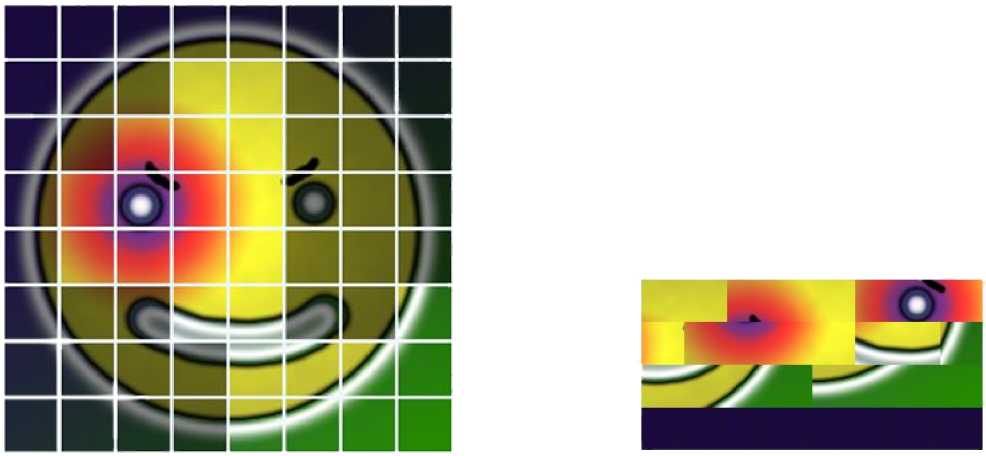
(b) Normály po výpočte z height textúry.

Obrázek 4.2: Porovnanie normál pred a po prepočítaní.

Kapitola 5

Virtuálne textúrovanie

Termín *virtuálne textúrovanie* onzačuje techniku v ktorej sa využíva textúra(virtuálna textúra) obrovských[1] rozmerov, ktorá veľkosťou presahuje kapacitu grafickej pamäte. Virtuálna textúra môže obsahovať textúry viacerých objektov alebo textúru jedného objektu vo vysokom rozlíšení. Druhý prípad je zaujimavejší z toho dôvodu, že je možné dosahovať detailnosť textúr, ktorá pred touto technikou nebola možná vzhľadom na limitáciu maximálnej veľkosti jednej textúry. Tento limit sa časom zväčšuje, ale aj pri súčasných moderných grafických kartách je maximálna veľkosť 16384 x 16384 texelov.



(a) Viditeľné stránky virtuálnej textúry.

(b) Načítané viditeľné stránky vo fyzickej textúre.

Obrázek 5.1: Porovnanie virtuálnej a fyzickej textúry. zdroj[2]

Princíp tejto techniky spočíva v rozdelení virtuálnej textúry na tzv. stránky (veľkosť je typicky 128x128 texelov) a načítaní len potrebných stránok textúry do grafickej pamäte na korektné vyrenderovanie scény. Potrebné stránky sa načítavajú do *fyzickej textúry* ku ktorej má prístup textúrovací vzorkovač v shadery. Veľkosť tejto textúry je typicky 4096 x 4096 texelov resp. 1024 stránok. Čo má za následok efektívnejšie využívanie grafickej pamäte, pretože v pamäti sú vždy len potrebné dáta. Na druhú stranu vzhľadom na to, že nie sú dostupné všetky dáta textúr a samotné stránky sú náhodne umiestnené je potrebná réžia,

[1] Počítačová hra RAGE využíva 128k x 128k texelové textúry.

[2] http://silverspaceship.com/src/svt/

ktorá sa bude starať o načítavanie a mazanie textúr z fyzickej textúry a prekladať virtiuálne textúrovacie súradnice objektu na fyzické tak aby ukazovali správne časti z fyzickej textúry. Problém s nekombatibilitou virtuálnych a fyzických textúrovacích súradnic je zobrazený na obrázku 5.1.

5.1 Softvérové virtuálne textury

Softwarové metódy virtuálneho textúrovania sú používané v hernom priemysle už od roku 2007[3]. Softvérová technika bola popularizovaná Johnom Carmackom z ID Software. Implementácia firmy ID software bola nazvaná Megatexture[8], čo v preklade znamená obrovská textúra.

5.1.1 Preklad adresy

Kedže sa jedná o softvérovú implementáciu, všetko je ovládané programom samotným. Jeden z hlavných problémov nastáva pri otázke ako preložiť virtuálnu textúrovaciu súradnicu (adresa) do fyzickej. Ako je vidieť na obrázku 5.1, virtuálne a fyzické adresy sú úplne rozdielne.

Najednoduchšia formu prekladu adresy z virtuálnej na fyzickú je ekvivalentná k procesu vyhľadávania požadovanej úrovne detialov (angl. level of detail - LOD) pomocou adresy virtuálnej textúry na prechod cez stromovú štruktúru, ktorá reprezentuje mip-map hierarchiu stránok momentálne nahratých v grefickej pamäti. Každý list stromu určuje scale a bias, ktoré zmenia virtuálnu adresu na fyzickú. Scale je pomer medzi veľkosťou virtuálnej mip-map úrovne a veľkosťou fyzickej textúry. Bias je zväčšený posun k virtuálnej stránke vo virtuálnej mip-map úrovni odpočítaný od posunu k fyzickej stránke vo fyzickej textúre.

Počas prechádzania stromu môžu nastať dve situácie: virtuálna adresa sa nájde alebo nie a použije sa scale a bias posledného listu. V druhom prípade sa preklad adresy prehodí do vyššej mip-map úrovne, pretože požadovaná stránka sa v grafickej pamäti nenachádza. Táto metóda má veľmi nízku naročnosť na pamäť ale zároveň najhoršiu výkonnosť čo sa týka odozvy.

Ďalšou metódou je využiť textúru, ktorá bude mať jeden texel pre každú stránku virtuálnej textúry. Nevýhodou tejto metódy je veľkosť stránkovacej tabuľky.

5.1.2 Filtrovanie

Veľkým problémom pri softvérovej implementácii virtuálnych textúr je filtrovanie. Problém spočíva v tom, že hardware nevie o tom, že textúra nahratá v pamäti sa skladá zo stránok a preto by pri hardwarovom filtrovaní vznikali artefakty na hraniciach stránok. Softvérová implementácia filtrovania by bola príliš drahá na výpočetný výkon.

Aby sa dala využiť hardvérové bi-lineárne filtrovanie, je potrebné aby každá stránka mala rámček texelov okolo seba. Pre pokročilejšie filtrovacie metódy ako napríklad anisotropické filtrovanie je dôležité aby ramček texelov mal viac ako jeden texel.

5.1.3 Aktuálnosť stránok

Stránky vo virtuálnej textúre sa budú stále meniť pri každej zmene v renderovanej scéne. Preto je potrebné najprv zistiť, ktoré stránky sa budú využívať a je potrebné nahrať ich do

[3]Enemy Territory: Quake Wars od vývojára Splash Damage.

grafickej pamäte a až potom je možné vyrenderovať scénu.

Vyrenderovaním scény do špeciálneho bufferu (je vhodné použiť menšie rozlíšenie ako rozlíšenie frame bufferu), ktorý bude uchovávať súradnice vo virtuálnej textúre, požadovanú mip-map úroveň a ID virtuálnej textúry (nie sme obmedzený na jednu virtuálnu textúru), budeme vedieť kde je aká stránka potrebná. Výhoda tohto prístupu je že renderovanie prechádza cez hĺbkovú kontrolu (depth test), takže virtuálne texturovanie nie je zbytočne zaťažované požiadavkami na stránky, ktoré nie sú vidieť.

5.2 Hardvérové virtuálne textury

Hardvérové virtuálne textúry nazývane aj partially resident textures (PRT) prinášajú priamu hardvérovú podporu pre vačšinu úkonov, ktoré sú potrebné pri softvérovej variante. Vývojár už nemusí vytváarať a spravovať tabuľku stránok ani prekladať adresy. Toto rieši hardware za vývojára.

Pôvodne boli PRT podporované len na grafických kartách od spoločnosti AMD od HD7XXX série cez OpenGL rozšírenie AMD_sparse_texture. Neskôr však vzniklo aj rozšírenie ARB_sparse_texture a EXT_sparse_texture2 pre ostatné grafické moderné grafické karty.

5.2.1 Výhody

Keď sa shader pokusí získať texel z textúry použitím UV súradníc, tak dedikovaná časť na GPU najprv vypočíta virtuálnu adresu texelu. Výpočet adresy zavisí od typu textúry, formátu, UV hodnôt, offsetu a pod. Virtuálna adresa je nasledovne vložená do tzv. hardware virtual memory subsystem (HVMS). HVMS nasledovne vykonáva preklady z virtuálnej do fyzickej adresy, spustí operáciu čítania texelu z fyzickej textúry a vráti hodnoty načítaného texelu. HVMS obsahuje dedikovanú hardvérovu tabuľku stránok, ktorá prináša niekoľko výhod v porovnaní so softvérovou variantou.

Pri sotfvérovej implementácii bolo potrebné rozhodnúť o veľkosti stránok, formáte a pod. Hardvérová stránkovacia tabuľka využíva zjednotený formát stránok pre všetky formáty textúr. Nevýhoda tohto prístupu je, že veľkosť jednej stránky je pevne určená na 64kB, čo znamená že 32-bit formát textury RGBA8 bude mať veľkosť 128 x 128 texelov.

Sofvérová implementácia nemá žiadnu hardvérovú podporu pre filtrovanie bez úpravy stránok pridaním rámčeka texelov. Na druhú stranu hardvérová implementácia má podporu pre hardvérové filtrovanie stránok virtuálnej textúry.

Hardvérová implementácia dokáže využívať špecialne cache na zrýchlenie požiadaviek na texely a preklad virtuálnych adries na fyzické adresy. Toto nie je možné pri softvérovej implementácii.

5.2.2 Chyba stránky

Chyba stránky je udalosť, ktorá nastane, keď shader zažiada texel virtuálnej textúry, ktorý nemá záznam v tabulke stránok čo znamená, že stránka nie je vo fyzickej textúre. V softvérovej implementácii je dotaz na existenciu texelu vo fyzickej textúre veľmi jednoduchý. Shader zažiada o textúru zo stránkovej tabuľky a analyzuje validnosť vráteného výsledku.

Pri hardvérovej variante nie je potrebné čítať údaj z tabuľky, pretože podporuje priamu propagáciu informácie o mapovaní stránky do fyzickej textúry. Požiadavka na texel priamo vráti chybu. Toto ale neplatí pre rozšírenie ARB_sparse_texture, EXT_sparse_texture2 ho rozširuje o túto funkcionalitu.

Kapitola 6

Návrh a implementácia

V tejto kapitole budú popísané algoritmy a postupy riešenia problémov, ktoré nastali počas vývoja programu. Na implementáciu programu bolo použité OpenGL 4.5 so GLSL vo verzii 450 core a jazyk C++ vo vývojovom prostredí Microsoft Visual Studio 2013. Vytvorenie kontextu, okna a vstup od užívateľa zaisťuje knižnica Simple Direct Media (SDL[1]) spolu s knižnicou GLEW[2] vo verzii 1.12.0 na sprístupnenie OpenGL fukncionality. Na načítavanie modelov bola použitá open source knižnica Assimp[3] vo verzii 3.1.1. Pre načítavanie a prácu s textúrami bola použitá knižnica DevIL[4] vo verzii 1.7.8. Knižnica OpenGL mathematics (GLM[5]) vo verzii 0.9.6.1 bola využívaná na prácu s vektormi a maticami mimo shaderov.

6.1 Nastavenie a renderovanie scény

Trieda Display zapúzdruje vytvorenie okna, pridelenie kontextu a základné nastavenie OpenGL.

Systém obsahuje triedy:

Camera - Predstavuje kameru, ktorá zobrazuje svoj pohľad do okna a spravuje matice relevantné k pohľadom.

Transform - Stará sa o transformácie objektu a udržiava transformačné matice.

Shader - Načítavá, linkuje a kompiluje zdrojové súbory shaderov.

Texture - Načítavá obrázkové súbory a nahráva ich do grafickej pamäte.

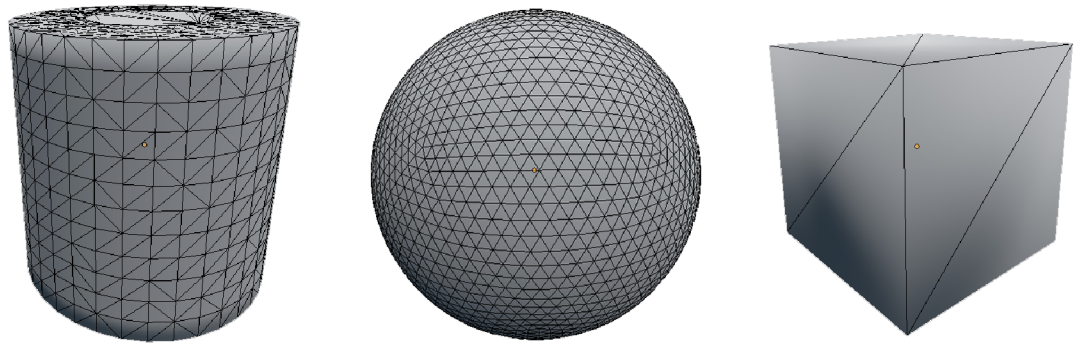
Mesh - Načítavá zdrojové súbory modelov, dopočítavá normály a tangenty.

Pre jednoduchosť, svetlo je reprezentované 3D vektormi pre polohu a farbu svetla jednou float hodnotou pre intenzitu ambient svetla.

[1] SDL knižnica je dostupná na https://www.libsdl.org/.
[2] GLEW knižnica je dostupná na https://http://glew.sourceforge.net/.
[3] Assimp knižnica je dostupná na http://assimp.sourceforge.net/.
[4] DevIL knižnica je dostupná na http://openil.sourceforge.net/.
[5] GLM knižnica je dostupná na http://glm.g-truc.net/.

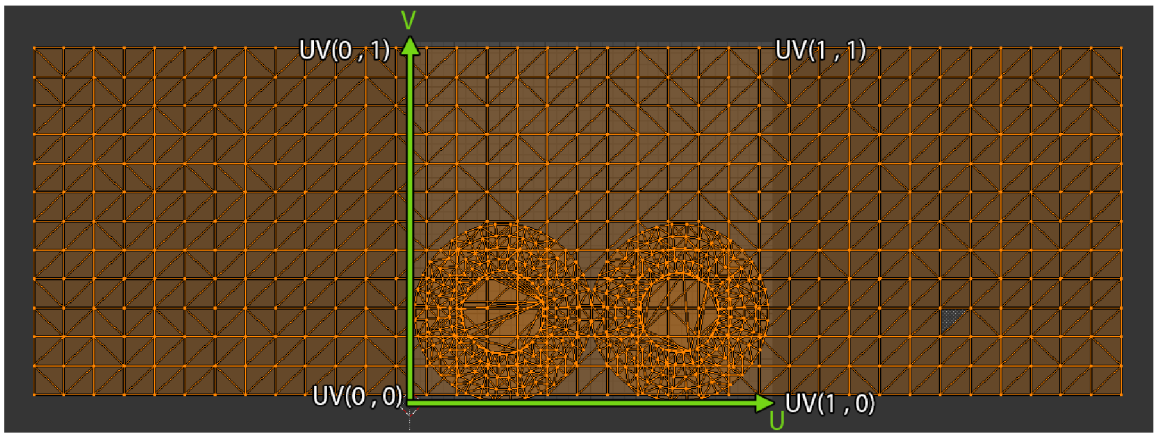
6.2 Tvorba a načítavanie modelov

Pre testovacie účely boli vytvorené 3 modely (cylinder, sphere a cube), ktoré postačujú na demonštráciu prezentovaných techník. Model "cube"má 12 trojuholníkov, model "cylinder"má 1508, model "sphere"má 5120 trojuholníkov. Boli zvolené jednoduché primitíva, vzhľadom na jednoduché mapovanie bezošvých dlaždicových textúr angl. seamless tile textures. Dlaždicové textúry sú textúry, ktoré sú upravené tak aby sa dali naskladať nad seba alebo vedľa seba bez toho aby bol viditelný spoj. Vybrané primitíva sú zobrazené na obrázku 6.1. Na tvorbu 3D modelov bol využitý open source software Blender 3D. Neodeliteľnou



Obrázek 6.1: Testovacie modely: zľava cylinder, sphere a cube

súčastou tvorby modelov bolo vytvorenie tzv. UV máp, ktoré predstavujú textúrovacie súradnice pre každý vrchol modelu. UV mapy museli byť správne zarovnané, keďže od textúrovacich súradnic závisí výpočet tangent, ktoré sa využívajú pri výpočte TBN matice v normal mappingu. Príklad UV mapy použitej v testovacom objekte "valec"je na obrázku 6.2



Obrázek 6.2: Príklad UV mapy testovacieho modelu "cylinder"vytvorenej v programe Blender 3D.

Knižnica assimp umožnuje načítavať rôzne formáty modelov, ale rozhodol som sa využívať len modely vo formáte OBJ, pre ich jednoduchosť a podporu v rôznych 3D modelovacích softwaroch.

Modely sa ukladajú do 5 rôznych array bufferov pre jednotlivé elementy, ktoré ho definujú (vrchol, textúrovacie súradnice, normála, dotyčnica, index). Pre prípad, keď model neobsahoval normály alebo dotyčnice, assimp ich automaticky dopočíta. Pre jednotnosť načítaných modelov a jednoduchosť riešenia je ešte striagulovaný (všetky polygony sú rozdelené na trojuholníky). Dôvodom triangulácie je, že teselačný engine pracuje vždy s jedným druhom primitíva.

Načítanie modelu rovno ukladá model do grafickej pamäte a vyvolaním metódy render() resp. render_displace() sa vykreslí model na obrazovku. Je potrebné mať 2 rôzne metódy na vykreslovanie klasickej geometrie a teselovanej, pretože teselovaná geometria sa dá vykreslovať len cez primitívum GL_PATCHES a obyčajná geometria je vykreslovaná cez GL_TRIANGLES.

6.3 Načítavanie textúr

Trieda Texture načítavá súbory textúry pomocou knižnice DevIL. Po úspešnom načítaní súboru je potrebné nastaviť formát dát obrázku na formát, ktorý sa bude používať v OpenGL. V tomto pripáde je použitý formát texelu `RGBA8` to znamená, že textúra má 4 kanály a každý z nich má 8 bitov pre uchovanie informácie o texely. Datový typ na uchovanie jedného texelu je `unsigned byte`.

Vzhľadom na to, že OpenGL považuje počiatočný bod angl. origin point textúry v ľavom dolnom rohu je potrebné obrázok prevrátiť po x osi. Dôvod tohto prevrátenia je, že DevIL načítavá textury s počiatočným bodom v ľavom hornom rohu. Vyhneme sa tak prevráteným textúram.

Následne sa načítane dáta nahrajú do grafickej pamäte. Paramater sparse v konštruktory triedy Texture určuje či sa bude jednať o partially resident texturu (PRT) alebo o klasickú textúru. Pre použitie tejto textúry je už len potrebné pripojiť textúru na textúrovaciu jednotku a to pomocou metódy Bind().

Pri PRT textúrach je potrebné aby veľkosť strán textúry, do ktorej sa budú načítavať stránky, bola násobkom veľkosti stránky. Veľkosť stránky pre konkrétný formát sa dá zistiť ako je uvedené v kóde 6.1.

```
1 GLint tileX, tileY, tileZ;
2 glGetInternalformativ(GL_TEXTURE_2D, GL_RGBA, GL_VIRTUAL_PAGE_SIZE_X_ARB, 1,
    &tileX);
3 glGetInternalformativ(GL_TEXTURE_2D, GL_RGBA, GL_VIRTUAL_PAGE_SIZE_Y_ARB, 1,
    &tileY);
4 glGetInternalformativ(GL_TEXTURE_2D, GL_RGBA, GL_VIRTUAL_PAGE_SIZE_Z_ARB, 1,
    &tileZ);
```

Kód 6.1: Dotaz na podporvané veľkosti stránok

6.4 Jednoduchý shader manager

Pre potreby realizácie tohto projektu musel byť vytvorený jednouchý shader manager, ktorý sa bude starať o načítavanie zdrojových súborov shaderov a následné ich kompilovať a linkovať. Mal by byť schopnosť načítavať skupiny shaderov, ktoré tvoria samostatný celok. Každý tento celok môže mať rôzne množstvo a typy zdrojových súborov.

Pre tieto účely bola vytvorená trieda Shader. Každá inštancia tejto triedy zapúzdruje viacero shader zdrojových súborov tvoriacich jeden celok. Konštruktor tejto metódy očákava

jediný parameter, ktorý určuje názov tohto shader programu. Zdrojové súbory sa potom načítavajú pridaním pripony za názov shader programu. Vertex shader (prípona .vs) a fragment shader (prípona .fs) sú povinné shadery. Tesselation evaluation shader (prípona .tes), tesselation control shader (prípona .tcs) a geometry shader (prípona .gs) sú nepovinné shadery. Po načítaní shaderu sa skompiluje. Vzhľadom na to, že bolo potrebné využívať aj hardware tesselator, teselačne časti shaderu su nepovinné. Pred samotným linkovaním shader programu sa nastavia lokácie pre array buffery obsahujúce pozíciu vrcholov, textúrovacie súradnice, normály a dotyčnice. Následne je program zliknovaný a kontroluje sa, či nenastali chyby pri linkovaní.

Posledná schopnosť tejto triedy je nastavovať uniform premenné, ktoré budú dostupné v shaderoch. Z toho vyplýva, že nám zaisťujú komunikáciu z CPU do GPU. Metóda sa nazýva `setUniform()` a pomocou preťažovania je možné ňou nastavovať uniform premenné základných GLSL datových typov ako napríklad vektory, matice, integer, float a bool. K tomuto pomohla knižnica GLM, ktorá umožnuje pracovať s vektormi a maticami presne tak ako s nimi pracuje GLSL, takže nie je potrebné nijak dáta upravovať a je možné ich ihneď odoslať do GLSL cez uniform premennú.

Scéna sa vždy vykresluje s jedným shaderom aktívným. Preto pri využívaní viacero shaderov je potrebné ich prehadzovať. Aktívny shader sa nastaví metódou `Bind()`.

6.5 Displacement mapping shader

Vertex shader funguje len ako tzv. pass-trough shader, čo znamená, že len prenáša vstupné hodnoty na výstup. V tomto prípade je potrebné preniesť pozíciu vrcholu, textúrovacie súradnice, normálu a tangentu. Náhľad je v kóde 6.2.

```
#version 450 core
in vec4 position;
in vec2 texCoord;
in vec3 normal;
in vec3 tangent;

out vec2 tex_coord_CS_in;
out vec3 normal_CS_in;
out vec3 tangent_CS_in;

void main () {
  //pass through shader, len posuva vstupy dalej
  tex_coord_CS_in = texCoord;
  normal_CS_in = normal;
  tangent_CS_in = tangent;
  gl_Position = position;
}
```

Kód 6.2: Vertex shader - displacement mapping

Naleduje tesselation control shader, ktorý ma za úlohu nastavovať úroveň tesselácie. Všetky úrovne teselácie sa nastavia na rovnakú hodnotu aby sme dosiahli rovnomernú teseláciu. Zároveň je potrebné zase posunúť ďalej textúrovacie súradnice, normály a dotyčnice. Tesselation control shader je v kóde 6.3.

```
#version 450 core
layout(vertices = 3) out;
uniform float tess_level;

in vec2 tex_coord_CS_in[];
in vec3 normal_CS_in[];
in vec3 tangent_CS_in[];

out vec2 tex_coord_ES_in[];
out vec3 normal_ES_in[];
out vec3 tangent_ES_in[];

void main()
{
  //nastavovanie urovne tesselacie pre vonkajsie hrany
  gl_TessLevelOuter[0] = tess_level;
  gl_TessLevelOuter[1] = tess_level;
  gl_TessLevelOuter[2] = tess_level;
  //...a pre vnutorne
  gl_TessLevelInner[0] = tess_level;
  //pozicie vertexov
  gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
  //posun udajov dalej
  normal_ES_in[gl_InvocationID] = normal_CS_in[gl_InvocationID];
  tangent_ES_in[gl_InvocationID] = tangent_CS_in[gl_InvocationID];
  tex_coord_ES_in[gl_InvocationID] = tex_coord_CS_in[gl_InvocationID];
}
```

Kód 6.3: Tesselation control shader - displacement mapping

Využíva sa tu uniform premenná `tess_level`, ktorá určuje úroveň teselácie. Toto nám dovoľuje nastavovať úroveň teselácie počas spusteného programu. Taktiež je potrebné nastaviť počet vrcholov, ktoré tvoria patch. V tomto prípade sú to tri, pretože pracujeme s trojuholníkmi za každých okolností. V tejto úrovni sú všetky vstupné aj výstupne premenné polia.

Ďalšia úroveň je tesselation evaluation shader, v ktorej sa prepočítavajú všetky vstupné premenné na už tesselované časti, ktoré tvoria patch. Táto úroveň sa spúšta nad každým novo tesseláciou vytvoreným trouholníkom.

```
#version 450 core
layout(triangles, equal_spacing, ccw) in;
...
vec2 interpolate2D(vec2 v0, vec2 v1, vec2 v2){
    return vec2(gl_TessCoord.x) * v0 + vec2(gl_TessCoord.y) * v1 + vec2(
    gl_TessCoord.z) * v2;
}
vec3 interpolate3D(vec3 v0, vec3 v1, vec3 v2){
    return vec3(gl_TessCoord.x) * v0 + vec3(gl_TessCoord.y) * v1 + vec3(
    gl_TessCoord.z) * v2;
}
void main()
{
//interpolovanie normály pre tesselovaný trojuholník
normal_FS_in = interpolate3D(normal_ES_in[0], normal_ES_in[1], normal_ES_in
    [2]);
normal_FS_in = normalize(normal_FS_in);
//interpolovanie tangenty pre tesselovaný trojuholník
tangent_FS_in = interpolate3D(tangent_ES_in[0], tangent_ES_in[1],
    tangent_ES_in[2]);
```

```
tangent_FS_in = normalize(tangent_FS_in);
//interpolovanie textúrovacích súradníc pre tesselovaný trojuholník
tex_coord_FS_in = interpolate2D(tex_coord_ES_in[0], tex_coord_ES_in[1],
    tex_coord_ES_in[2]);
//čítanie výšky posunu vrcholu z displacement textúry
float h = texture(height_tex, tex_coord_FS_in.xy).r;
//výpočet normálového vektoru, z ktorého sa bude počítať osvetlenie
vec3 N = normalize(mat3(MV_mat) * normal_FS_in);
vs_out.N = N;
//výpočet tangent vektoru, z ktorého sa bude počítať osvetlenie
vec3 T = normalize(mat3(MV_mat) * tangent_FS_in);
vs_out.T = T;
//výpočet bitangent vektoru, z ktorého sa bude počítať osvetlenie
vec3 B = cross(T ,N);
vs_out.B = B;
//získanie pozície tesselovaného
vec4 position =gl_in[0].gl_Position.xyzw * gl_TessCoord.x +
        gl_in[1].gl_Position.xyzw * gl_TessCoord.y +
        gl_in[2].gl_Position.xyzw * gl_TessCoord.z;
//posun vertexu po normále o vzdialenost h a displace faktorom
vec4 newVertexPos = vec4(normal_FS_in * h * displace_ratio,0.0) + position;
gl_Position = MVP_mat * newVertexPos;
//Výstupný vektor svetla
vs_out.L = mat3(V_mat)*light.vDirection - vec3(MV_mat * newVertexPos);
//výstupný vektor pohľadu
vs_out.V = vec3(-MV_mat * newVertexPos);
}
```

Kód 6.4: Úryvok z tesselation evaluation shader - displacement mapping

Poslednou úrovňou je fragment shader. Úlohou tejto úrovne, je vykresliť teselovaný model s farebnou textúrou, specular textúrou a normálami vypočítanými z displacement textúry. Je využívaný Blinn-Phong svetelný model a normály sú vypočítané metódou bump mapping. Vzhľadom na to, že úroveň posunu bodov je možné užívateľsky zmeniť bolo potrebné podľa toho aj meniť úroveň bump mappingu. Na určenie posunu slúži uniform premenná `displace_ratio`, podľa ktorej sa počíta úroveň bump mappingu. Pre testované modely som testovaním zvolil konštantu 200, ktorou sa násobí `displace_ratio` pri výpočte rozdielov susedných texelov. Vplyv tohto faktoru na nízke a vyššie úrovne posunu je zobrazený na obrázku 6.3. Pri zápornej hodnote v `displace_ratio` sa vrcholu posúvaju smerom nadol po normále a displacement mapping má opačný efekt tj. tam kde by mali byť vyvýšeniny bude povrch znížený. Fragment shader je zobrazený v kóde 6.5.

```
#version 450 core
uniform sampler2D color_tex;
uniform sampler2D normal_tex;
uniform sampler2D specular_tex;
uniform sampler2D height_tex;
...
void main()
{
//rozmery textury
float WIDTH= 1024.0f;
float HEIGHT= 1024.0f;
//uroven
float SCALE =displace_ratio;
vec2 uv =  tex_coord_FS_in.xy;
vec2 du = vec2(1/WIDTH, 0);
vec2 dv= vec2(0, 1/HEIGHT);
```

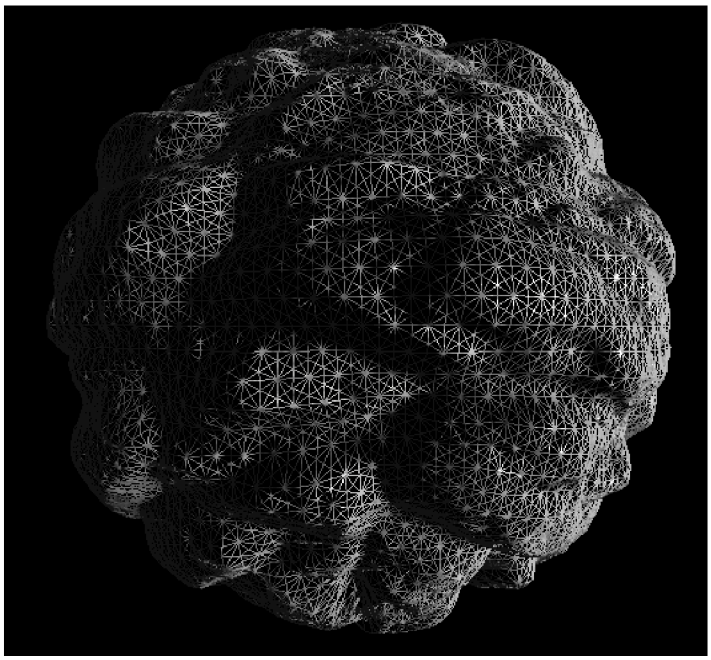
(a) *displace_ratio* = 0.06 (b) *displace_ratio* = 0.30

Obrázek 6.3: Vplyv hodnoty displace_ratio na výpočet normál.

```
17 //pozri susedov a vypocitaj rozdiel
18 float dhdu = (SCALE*200) * (texture(height_tex, uv+du).r - texture(height_tex
       , uv-du).r);
19 float dhdv = (SCALE*200) * (texture(height_tex, uv+dv).r - texture(height_tex
       , uv-dv).r);
20 //vypocet normaly z vyskovej textury a normalizacia vektorov
21 vec3 N = normalize(fs_in.N -fs_in.T*dhdu +fs_in.B*dhdv);
22 vec3 L = normalize(fs_in.L);
23 vec3 V = normalize(fs_in.V);
24 vec3 H = normalize(L + V);
25 vec3 diffuse_albedo = material.color;
26 //pokial je potreba, nacita farebny texel
27 if(use_color_texture == true)
28   diffuse_albedo =texture(color_tex, tex_coord_FS_in).rgb;
29 //ambient zlozka
30 vec3 ambient = light.fAmbientIntensity * diffuse_albedo;
31 //lambert
32 vec3 diffuse = max(dot(N, L), 0.0) * diffuse_albedo;
33 //specular mapping
34 vec3 spec_factor = vec3(texture2D(specular_tex, tex_coord_FS_in));
35 //specular zlozka
36 vec3 specular = max(pow(dot(H, N), material.specular_power), 0.0) * material.
       specular_albedo * spec_factor;
37 //vystup
38 colorOut = vec4(ambient+  diffuse + specular, 1.0);
39 //zobraz normaly
40 if(show_normals)
41   colorOut = vec4(N, 1.0);
42 }
```

Kód 6.5: Úryvok z fragment shader - displacement mapping

Na renderovanie drôtového modelu bola využitá funkcia `glPolygonMode()`, ktorá umožňuje nastaviť ako chceme polygony reprezentovať. Pôvodné nastavenie je na `G_FILL`, čo v praxi znamená, že polygon sa vykresľuje ako plocha tvorená v našom prípade tromi bodmi. Nastavením hodnoty `GL_LINE` získame drôtový model, avšak fragment shader bude zapisovať farby len na miesta kde sa nachádza čiara, pretože len táto časť je rasterizovaná. Náhľad drôtového modelu je zobrazený na obrázku 6.4.



Obrázek 6.4: Drôtový model teselované objektu.

6.6 Ostatné shadery

Pre porovnávacie účely je implementovaný samostatný Blinn-Phong svetelný model aj v kombinácii s normal mappingom. Všetky tri shadery su načítané a skompilované na začiatku behu programu, čo umožňuje okamžite meniť shadery počas renderovania.

6.7 Užívateľský vstup a výstup aplikácie

Na spracovanie vstupu je použitá knižnica SDL, ktorá umožnuje odchytávať udalosti vstupných zariadení ako klávesnica, myš alebo gamepad. Interaktivita programu spočíva v prepínaní aktívneho shaderu medzi Blinn-Phong, Blinn-Phong s normal mappingom a displacement mapping využívajúci Blinn-Phong svetelný model s bump mappingom. Ďalšou možnosťou je vypínať alebo zapínať zobrazenie farebnej textúry a farebnej vizualizácie normál polygonov. Špecificky pre displacement mapping je možné nastaviť úroveň tesselácie a rozsah posunu vrcholov. Pre pohyb v 3d priestore je implementovaná lietajúca kamera s pohľadom z prvej osoby s možnosťou meniť uhol zorného pole.

SDL knižnica dokáže zistiť počet pixelov o koľko sa sa myš pohla a túto hodnotu určuje relatívne od poslednej. Vzdialenosť je uložená v premenných `motion.xrel` a `motion.yrel`,

pre obdive osi, ktoré obsahuje udalosť SDL event. Vďaka tejto hodnote, sme schopný otáčať kamerou. Náhľad algoritmu, ktorý spracováva vstupy z myši je zobrazená v kóde 6.6.

Pohyb kamerou horintálne sa realizuje vytvorením rotačnej matice cez GLM funkciu `rotate()`, ktorá vyžaduje v parametroch uhol otočenia a vektor smerújuci od kamery nahor, niekedy nazývany angl. up vector. Rotačnou maticou pre horizontálny pohyb sa vynásobí vektor smerujúci dopredu z kamery (forward vector) a up vector.

Pohyb kamerou vertikálne je trošku zlozitejší. Ako prvé je potrebné zistiť vektor, ktorý je pravoúhly od `forwardaj` `up` vektoru. Tento vektor nazvyme `right` a získame ho cez cross produkt vektorov `forward` a `up`. Nový `forwad` vektor sa získa vynásobením rotačnou maticou s osou otáčania okolo vektoru `right`. Nasledovne je potrebné vypočítať nový `up` vektor, kedže kamera už pozoruje iný bod. Nový `up` vektor sa vypočíta cross produktom medzi vektormi `right` a `forwad`. Implementácia pre obidve metódy otočenia kamery je zobrazená v kóde 6.7.

```
case SDL_MOUSEBUTTONDOWN:{
  if (e.button.button == SDL_BUTTON_RIGHT){
    if (drag == false)
    drag = true;
  }
}
break;
case SDL_MOUSEMOTION:{
  if (drag == true){
  int x = e.motion.xrel;
  int y = e.motion.yrel;
  cam.LookHorizontal((float) (-x)/ 70);
  cam.LookVertical((float) y/70);
  }
}
break;
case SDL_MOUSEBUTTONUP:{
  if (e.button.button == SDL_BUTTON_RIGHT)
  {
    if (drag == true)
    drag = false;
    }
  }
break;
case SDL_MOUSEWHEEL:{
  cam.Fov(e.wheel.y * .01f);
}
break;
```

Kód 6.6: Ovládanie kamery.

```
void LookVertical(float angle){
  glm::vec3 right = glm::normalize(glm::cross(up, forward));
  forward = glm::vec3(glm::normalize(glm::rotate(angle, right) * glm::vec4(
    forward, 0.0)));
  up = glm::normalize(glm::cross(forward, right));
}
void LookHorizontal(float angle){
  glm::mat4 rotation = glm::rotate(angle, up);
  forward = glm::vec3(glm::normalize(rotation * glm::vec4(forward, 0.0)));
  up = glm::vec3(glm::normalize(rotation * glm::vec4(up, 0.0)));
}
```

Kód 6.7: Ovládanie kamery.

V programe sa nachádza premenná `activeShader`, ktorá obsahuje ukazateľ na aktívny shader. Bolo potrebné mať nejakú indikáciu, či aktívny shader využíva teseláciu. Keďže sa tesselácia využíva len v displacement shadery premenná má názov `displace`. Princíp prepínania aktívneho shaderu je v kóde 6.8.

```
case SDLK_1:
  blinn.Bind();
  activeShader = &blinn;
  displace = false;
  break;
case SDLK_2:
  normal.Bind();
  activeShader = &normal;
  displace = false;
  break;
case SDLK_3:
  shader.Bind();
  activeShader = &displacement;
  displace = true;
  break;
```

Kód 6.8: Úryvok zo spracovani vstupu, ktorý rieši prepínanie medzi shaderami.

Program neobsahuje žiadne grafické užívateľské rozhranie, preto sú reakcie na užívateľské akcie vypisované na štandartný výstup do konzole. Chybové hlásenia sú vypisované na chybový výstup.

Kapitola 7

Výsledky a merania

Výsledné rendery modelu "cylinder"so všetkými shaderami sú zobrazené na obrázku 7.4. Všetky testy boli vykonávané na operačnom systéme Microsoft Windows 8.1 s aktuálnymi grafickými driverami. Pre testovania boli použité 2 desktopové počítačové zostavy a jeden notebook:

Zostava 1
- Intel Core i7 3770K @3,5Ghz
- 16GB RAM 1600Mhz
- Nvidia GTX 660ti 2GB VRAM

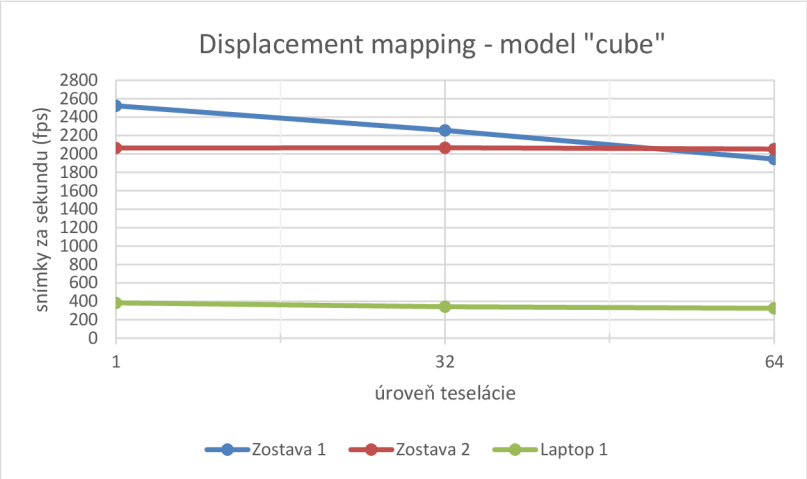
Zostava 2
- Intel Core i7 920 @2,67Ghz
- 12GB RAM 1333Mhz
- Nvidia GTX 970 4GB VRAM

Laptop 1
- Intel Core i5 4210M @2,6Ghz
- 8GB RAM 1600Mhz
- Nvidia GT 820M 2GB VRAM

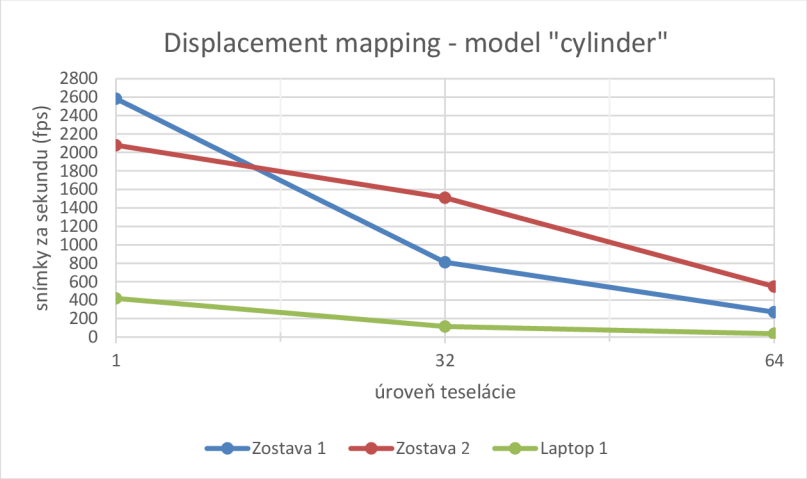
Bude sa testovať počet snímkov za sekundu (fps) so shaderami normal mapping a displacement mapping na každom z troch testovacích objektov. Pri druhom sa budú testovať 3 úrovne tesselácie (1,32,64). Každý test bude vykonaný 10 krát a výsledky budú spriemerované na jednu hodnotu. Počet vyrenderovaných snímkov je pevne daný na 10240 pre každý z testov. Cieľom je zistiť celkový dopad na výkon pri použití normal mappingu a displacement mappingu s rôznymi úrovňami tesselácie. Výsledky testovania sú v tabuľke 7.1.

Pri teste s modelom "cube"nastal zaujimavý a neočakávaný stav. Zostava 1 s grafickou kartou Nvidia GTX 660ti podáva vyšší výkon ako novšia a výkonnejšia Nvidia GTX 970 v zostave 2. Tento jav nastáva pri testoch s úrovňou teselácie 1 a 32. Nie je to chyba merania, testy boli zopakované niekoľko krát mimo hlavných testov. Procesor aj RAM v zostave 2 majú nižší výkon ako tie v zostave 1 ale nemali by mať vplyv na výkon aplikácie, vzhľadom na to, že všetky dôležité výpočty sa spracovávajú na grafickom procesore. Pretaktovaním procesora v zostave 2 sa potvrdilo, že nemá veľký vplyv na výkon aplikácie.

Ďalší zaujímavý jav je, že zostava 1 ma klesajúci výkon narozdiel od ďalších dvoch zostáv. Graf je zobrazený na obrázku 7.1.



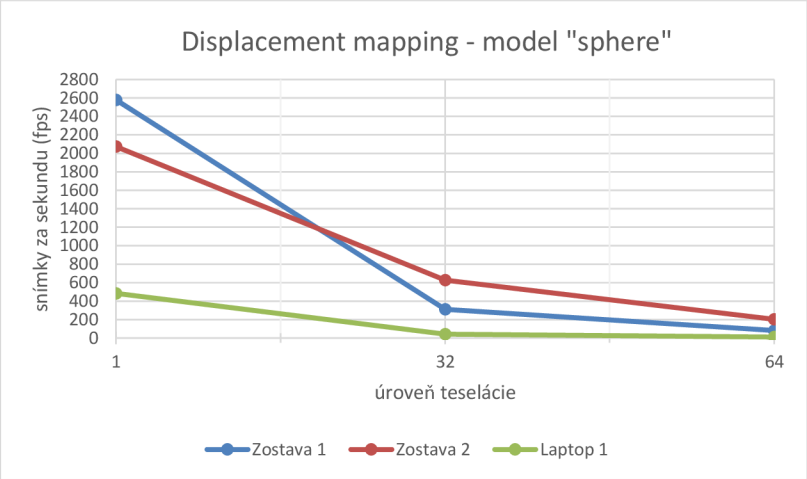
Obrázek 7.1: Graf výkonu zostáv s objektom ”cube”.



Obrázek 7.2: Graf výkonu zostáv s objektom ”cylinder”.

Pri teste s modelom ”cylinder”nastal podobný jav ako v predchádzajúcom teste, kde zostava 1 so slabšou grafickou kartou má vyšší výkon pri displacement mappingu bez teselácie. Neobjavuje sa však jav s obdobným výkonom nezávisle na úrovni teselácie pri zostave 2 a laptope 1. Výkon viditeľne klesá na všetkých zostavách. Graf tohto testu je na obrázku 7.2.

Test s modelom ”sphere”je najnáročnejší, pretože pôvodný model ma najvyšší počet polygonov zo všetkých testovacích modelov. Laptop 1 nezvláda obrovský počet trojuholníkov pri teselácii. Záťaž je obrovská aj na zostavu 1, kde výkon klesol zhruba o 97% pri maximálnej teselácii. Výsledky sú v grafe na obrázku 7.3.



Obrázek 7.3: Graf výkonu zostáv s objektom ”sphere”.

Názov testu	**Zostava 1**	**Zostava 2**	**Laptop 1**
Cube			
Normal mapping	2536 fps	2026 fps	397 fps
Displacement mapping (úroveň = 1)	2522 fps	2064 fps	383 fps
Displacement mapping (úroveň = 32)	2255 fps	2067 fps	341 fps
Displacement mapping (úroveň = 64)	1944 fps	2053 fps	325 fps
Cylinder			
Normal mapping	2536 fps	2074 fps	438 fps
Displacement mapping (úroveň = 1)	2584 fps	2079 fps	419 fps
Displacement mapping (úroveň = 32)	810 fps	1511 fps	115 fps
Displacement mapping (úroveň = 64)	270 fps	548 fps	36 fps
Sphere			
Normal mapping	2578 fps	2077 fps	485 fps
Displacement mapping (úroveň = 1)	2579 fps	2073 fps	484 fps
Displacement mapping (úroveň = 32)	310 fps	627 fps	42 fps
Displacement mapping (úroveň = 64)	82 fps	203 fps	11 fps

Tabulka 7.1: Tabuľka výsledkov testovania

(a) Len farebná textúra.

(b) Normal mapping.

(c) Displacement mapping bez teselácie.

(d) Displacement mapping s maximálnou teseláciou.

Obrázek 7.4: Výstupy aplikácie.

Kapitola 8

Záver

Cieľom bakalárskej práce bolo implementovať v OpenGL techniku tvorby nerovností povrchu displacement mapping s virtualnými textúrami. Bolo potrebné naštudovať princíp fungovania GLSL shaderov v OpenGL, základne svetelné modely, predchádzajúce techniky simulovania nerovností povrchu a virtuálne textury.

Bol vytvorený systém na správu shaderov, ktorý umožnuje jednoduchú prácu so shaderami. Implementovaný displacement mapping shader s flexibilnou úrovňou teselácie je relatívne nenáročný a pri rozumných hodnotách sa príliš neodlišuje od bump mappingu. Bez teselácie je v niektorých prípadoch rýchlejší ako normal mapping. Samotná kvalita výstupu je ale omnoho vyšia, keďže sa mení silueta objektu, čo pri normal mappingu nenastáva. V praxi je teda táto metóda využitelná bez viditeľne nižieho výkonu ale zároveň s viditeľným vizaulným zlepšením. Všetky textúry boli implementované ako partially resident textures.

V testovaní nastali niektoré neočakváne situácie ako napríklad, keď podľa špecifikácie slabšia grafická karta bola lepšia v niektorých testoch ako novšia a výkonnejšia grafická karta rovnakého výrobcu a s rovnakými ovládačmi.

Práca by sa mala rozšíriť o optimalizáciu teselácie, ktorá je v tomto štádiu fixná (uživateľsky nastaviteľná). Naprikíklad meniť úroveň teselácie na základe vzdialenosti vrcholu od kamery. Ďalšou optimalizáciou by bolo teselovať len tie vrcholu, ktoré ma význam teselovať. Pokiaľ displacement textúra obsahuje len jeden maly kopček v strede textúry nemá význam aby sa teselovali časti, ktoré sú ploché.

Literatura

[1] Beckmann, P.; Spizzichino, A.: The scattering of electromagnetic waves from rough surfaces. *Norwood, MA, Artech House, Inc., 1987, 511 p.*, ročník 1, 1987.

[2] Blinn, J. F.: Models of light reflection for computer synthesized pictures. In *ACM SIGGRAPH Computer Graphics*, ročník 11, ACM, 1977, s. 192–198.

[3] Blinn, J. F.: Simulation of wrinkled surfaces. *ACM SIGGRAPH Computer Graphics*, ročník 12, č. 3, 1978: s. 286–292.

[4] Cook, R. L.; Torrance, K. E.: A reflectance model for computer graphics. *ACM Transactions on Graphics (TOG)*, ročník 1, č. 1, 1982: s. 7–24.

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

[6] Phong, B. T.: Illumination for computer generated pictures. *Communications of the ACM*, ročník 18, č. 6, 1975: s. 311–317.

[7] Schlick, C.: An Inexpensive BRDF Model for Physically-based Rendering. In *Computer graphics forum*, ročník 13, Wiley Online Library, 1994, s. 233–246.

[8] van Waveren, J.: Software Virtual Textures. 2012.