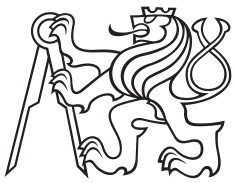
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Program pro tvorbu normálových map z jedné fotografie

Šimon Sedláček
Otevřená Informatika

Květen 2017
Vedoucí práce: David Sedláček

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Sedláček Šimon

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: Program pro tvorbu normálových map z jedné fotografie

Pokyny pro vypracování:

Navrhněte a implementujte program, který vytvoří normálovou mapu z jedné fotografie. Normálová mapa bude aplikovatelná jako textura pro 3D vykreslování - obsahuje informaci o normále objektu v daném bodě povrchu. Finální program otestujte minimálně na 10-ti vzorcích rozdílných vstupních fotografií a zhodnoťte kvalitu vytvořené mapy a i finálního vykresleného 3D obrázku s jejím použitím.

Seznam odborné literatury:

[1] Computer Assisted Relief Generation-A Survey. J. Kerber, M. Wang, J. Chang, J. J. Zhang, A. Belyaev and H.-P. Seidel. COMPUTER GRAPHICS forum 2012, Volume 31, Issue 8, December 2012, Pages 2363-2377.

[2] Interactive normal reconstruction from a single image. Tai-Pang Wu, Jian Sun, Chi-Keung Tang, Heung-Yeung Shum. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2008, Volume 27 Issue 5, December 2008.

Vedoucí: Ing. David Sedláček, Ph.D.

Platnost zadání do konce letního semestru 2017/2018

L S

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 12.1.2017

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 12. 5. 2017

Abstrakt / Abstract

V této práci popisuji využití Shape from Shading algoritmů na vytvoření výškových a normálových map z jediné fotografie, následně je zde popsána implementace algoritmu Interactive Normal Reconstruction from a Single Image [1], který dokáže ze vstupní fotografie vypočítat výškovou mapu. Pro dosažení lepších výsledků jsem algoritmus upravil. Pro výpočet normálových map z map výškových jsem využil Sobel-Feldmanův operátor konvoluce. Následně jsem implementaci algoritmu Interactive Normal Reconstruction from a Single Image testoval na různých vstupních obrázcích a srovnával jsem výstupy s výstupy autorů článku [1]. Algoritmus jsem také testoval vzhledem k počtu výpočetních kroků, velikosti vstupního obrázku a vzhledem k různorodosti dat na vstupu. Implementovaný algoritmus jsem použil v programu NormalMAPP, který je výsledkem mé práce. Program NormalMAPP dokáže vypočítat výškovou a normálovou mapu z jediné fotografie. Uživatelské prostředí jsem testoval s uživately a nalezené problémy jsem zpět reflektoval do implementace programu.

Klíčová slova: rekonstruce povrchu; rekonstruce povrchu z jediného obrázku; Shape from Shading; výškové mapy; hloubkové mapy; normálové mapy; textury; Interactive Normal Reconstruction from a Single Image; editor pro tvorbu textur; editor pro tvorbu výškových map.

In this document I describe the usage of Shape from Shading algorithms with goal of obtaining height maps and normal maps from single image. As a next part of the document, there is described an implementation of algorithm Interactive Normal Reconstruction from a Single Image [1], which can calculate height map from single image. I altered the algorithm for obtaining better results. I used convolution for calculating normal maps from height map. As convolution mask I used Sobel-Feldman operator. I tested my implementation of Interactive Normal Reconstruction from a Single Image on different input images and I compared my outputs with outputs from authors of the algorithm. I also tested the algorithm due to calculation steps, size of input image and diversity of input images. Implementation of this algorithm has been used in software called NormalMAPP, which is the result of my work. NormalMAPP can calculate height map and normal map from single image. User interface has been tested with users. Problems which arised from the testing has been resolved in the final aplication.

Keywords: surface reconstruction;surface reconstruction from single image; Shape from Shading; height maps; depth maps; normal maps; textures; Interactive Normal Reconstruction from a Single Image; texture editor; editor for height map creation.

Title translation: Software for generating normal maps from single image

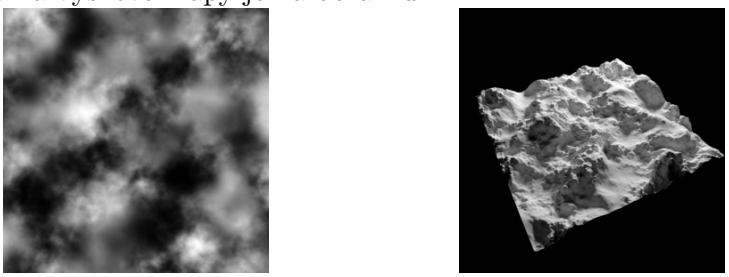
Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

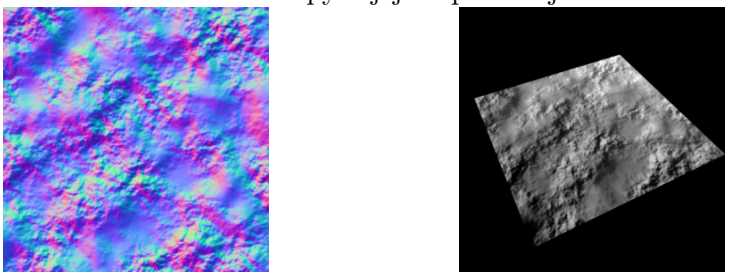
V počítačové grafice se používají textury pro dosažení kvalitnějšího výstupu vykresleného trojdimenzionálního povrchu bez nutnosti obrovského výpočetního výkonu. Textury se používají zejména na simulaci barevnosti povrchu, úpravy lesklosti materiálů, úpravy normálových vektorů a podobně. V mé práci se budu zabývat dvěma typy textur, které spolu úzce souvisí.

Prvním typem je výšková mapa (někdy hloubková mapa). Jak název napovídá, výšková mapa nám popisuje relativní výšku v daném bodě vůči nulové hladině. Tato textura využívá jen hodnot šedi, kde nulová hladina je znázorněna šedou barvou (polovina spektra, většinou 127). Černá barva je nejnižší bod, bílá barva je bod nejvýše položený, viz [4]. Pomocí této textury lze popsat celý trojdimenzionální povrch a povch díky ní i generovat. Ukázka výškové mapy je na obrázku 1.1.



Obrázek 1.1. Vlevo výšková mapa (převzato z [1]), vpravo výšková mapa použitá na zdeformování objektu.

Druhým typem textury je normálová mapa. Normálová mapa nám v každém pixelu udržuje informaci o normálovém vektoru (vektoru kolmém k povrchu v daném bodě). Díky této textuře jsme v počítačové grafice schopni vykreslit zvrásněné povrchy, které ale nemusíme modelovat. Normálová mapa mění směr odrazu světla v daném bodě, ale nemění tvar objektu. Tato textura nám poskytne vizuální kvalitu bez nutnosti výpočetního výkonu. Příklad normálové mapy a jejího použití je vidět na obrázku 1.2.



Obrázek 1.2. Vlevo normálová mapa, vpravo normálová mapa použitá na vykreslený čtverec.

[1]) `http://playerstage.sourceforge.net/doc/Gazebo-manual-0.8.0-pre1-html/terrain.png`

Při tvorbě prostorových modelů si modelář často kreslí barevné textury sám nebo je nalezne na internetu. K takovýmto texturám pak většinou nemá ani výškové ani normálové mapy. V dnešní době jsou ale již výškové a normálové mapy v prostorové grafice naprostým základem (zejména normálové mapy). Proto jsem se v mé práci rozhodl vytvořit program, který dokáže k vložené fotografii (obrázku) vypočítat výškovou a normálovou mapu.

Jak jsem již zmiňoval, výškové a normálové mapy nám popisují tvar povrchu. Proto jsem hledal algoritmus, který dokáže z jediného obrázku zrekonstruovat jeho povrch. Se svou studií jsem začal v článku [5].

Povrch můžeme rekonstruovat s takovou kvalitou, jak kvalitní máme vstup. Můžeme algoritmy rozdělit do tří skupin, viz [5], dle jejich vstupních informací:

Modelování: celý povrch se znovu nakreslí v některém 3D editoru

Vstupní fotografie: používá 2D obrázek jako předlohu / vzor

Vstupní povrch: používá 3D geometrii jako vstup

V mé práci se zaměříme pouze na druhou skupinu - algoritmy, které rekonstruují povrch ze vstupní fotografie. Rekonstrukce povrchu z jediného obrázku je náročná úloha, neboť informace, kterou potřebujeme pro náš třetí rozměr nemáme dostupnou, máme k dispozici pouze barvu pixelu umístěnou do 2D prostoru. Tímto náročným úkolem se zabývá vědecká disciplína zvaná Shape From Shading, viz [5] .

V kapitole 2 ukáži několik algoritmů, které slouží k rekonstrukci povrchu z jediného snímku. V dalších kapitolách se již budu zabývat pouze algoritmem Interactive Normal Reconstruction from a Single Image, viz [1], který jsem ve své práci implementoval. Na konci budou následovat ukázky z programu NormalMAPP, který jsem vytvářel v rámci práce a který poskytuje uživatelské prostředí pro výpočet normálových (a výškových) map pomocí implementovaného algoritmu.

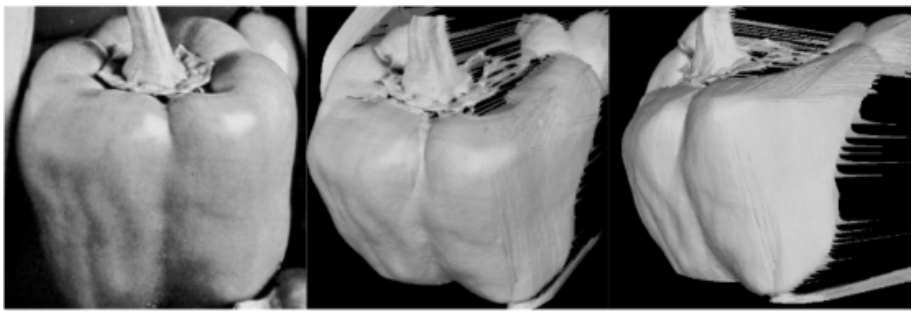
Kapitola 2
Algoritmy na rekonstrukci povrchu

Obtížnost samotného problému rekonstrukce povrchu z jediného obrázku byla popsána Berthold K. P. Horn, Richard, S. Szeliski, a Alan L. Yuille v práci [6]. Je zde popsáno, že nelze zcela jednoznačně u každého povrchu určit jeho tvar z jediné fotografie. Tento fakt má za následek, že všechny algoritmy, které zde uvádím, pouze odhadují, jak by mohl povrch vypadat, ale nezaručují, že povrch skutečně takto vypadal.

Nyní si ukážeme několik algoritmů na rekonstrukci povrchu.

2.1 Interactive shape from shading

Tento algoritmus požaduje na vstupu více než jen vstupní obraz. Požaduje, aby uživatel určil směr normálového vektoru v minimálně třech bodech. Díky této dodané informaci lze odhadnout směr nasvícení dané scény, kde byla fotografie pořízena. Algoritmus byl navržen na povrchy, které se jsou velice podobné Lambertovu stínovacímu modelu (více v kapitole 3.1.1), pro tento model byly výsledky velice dobré, bohužel algoritmus není použitelný pro jiné nasvětlovací modely. Algoritmus rekonstruuje povrch tím, že podle vstupních parametrů na obrázku nalezne vrcholy (kandidáty na lokální extrémy). Tyto vrcholy následně začne propojovat křivkami. Více viz [2]. Ukázka výstupů algoritmu je na obrázku 2.1.



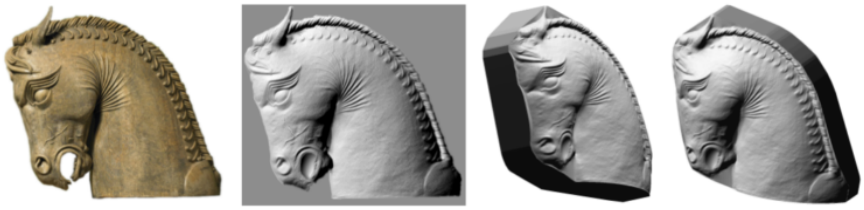
Obrázek 2.1. Ukázka převzata z článku [2]. Vlevo je vstupní fotografie, prostřední a pravý obrázek je rekonstrukce z různých úhlů.

2.2 Interactive Normal Reconstruction from a Single Image

Algoritmus Interactive Normal Reconstruction from a Single Image navazuje na práci [2] a vylepšuje algoritmus Interactive shape from shading. Tento algoritmus uvedl způsob, jak opravit malé chyby při rekonstrukci povrchu interaktivní metodou. Algoritmus předpokládá, že materiál povrchu a jeho nasvícení je podobné Lambertovu stínovacímu modelu. Ze vstupních hodnot se opět odhadne vektor pozice nasvětlení scény.

Normálové vektory se poté spočítají tak, aby splňovaly rovnici Lambertova stínování s co nejmenší chybou. Tato metoda bohužel zavádí do spočtených vektorů chybu, která má tendenci natáčet veškeré vektory směrem, jakým byl vektor pozice nasvětlení scény.

Tuto chybu algoritmus řeší prokládáním kružnic mezi jednotlivými vektory tak, aby vektory byly tečnými vektory kružnice. Z těchto vztahů algoritmus poté spočítá výšku povrchu v každém pixelu, tudíž je výstupem výšková mapa. Druhá část algoritmu spočívá v opravě vzniklé výškové mapy pomocí dalšího uživatelského vstupu. Uživatel nakreslí na povrchu koule úsečku a tuto úsečku zopakuje na vzniklé výškové mapě, algoritmus následně přepočítá výškovou mapu, aby povrch výškové mapy v místě úsečky odpovídal povrchu koule v místě úsečky na kouli. Ukázka výstupů algoritmu je na obrázku 2.2. Více viz [1].



Obrázek 2.2. Ukázka převzata z článku [1]. Vlevo je vstupní fotografie, dále pak rekonstruovaný povrch.

2.3 Relief as Images

Relief as Images je plně automatický algoritmus, nepožaduje kromě vstupní fotografie žádné dodatečné informace. Autoři algoritmu se snažili vytvořit takový algoritmus, který by nebyl závislý na počtu světelných zdrojů, na barvě světla ani na materiálu povrchu. Relief as Images vytváří reliéfy tak, že pro daný obrázek vytvoří vektorovou síť složenou z poligonů (čtverce rozdělené na 4 trojúhelníky) a tuto vektorovou síť ohýbá tak, aby její povrch po vykreslení měl obdobnou hodnotu stupně šedi v daném bodě jako původní fotografie. Více viz [3]. Ukázka výstupů algoritmu je na obrázku 2.3.

Obrázek 2.3. Ukázka z článku [3]. Obrázky v pozadí jsou rekonstruované reliéfy. Obrázky v popředí jsou vstupní fotografie

2.4 Výběr algoritmu

Pro svůj program na rekonstrukci povrchu z jediné vstupní fotografie jsem si vybral algoritmus Interactive Normal Reconstruction from a Single Image, neboť nejen že nabízel dobrý základ pro zajímavou práci (interaktivní zadávání vstupních údajů a interaktivní opravování chyb výpočtu), ale navíc výstupy rekonstruovaných povrchů, které tento algoritmus spočítal, byly velice kvalitní. Porovnání Interactive Normal Reconstruction from a Single Image oproti ostatním algoritmům je popsáno v článku [1].

V mé práci implementuji pouze první část algoritmu, druhou část (v článku [1] zvanou RotationPalette: Interactive Normal Manipulation), jsem již pro obtížnost první části neimplementoval.

Kapitola 3
Interactive Normal Reconstruction from a Single Image

Tato kapitola popisuje funkci algoritmu Interactive Normal Reconstruction from a Single Image, viz [1]. Algoritmus dokáže odhadnout rekonstrukci trojdimenziálního povrchu z jediného obrázku. Vstupem je jediný obrázek daného povrchu a tři nebo více vektorů. Pomocí vstupních hodnot algoritmus spočítá vektor pozice nasvětlení scény. Tento vektor se následně použije do rovnice Lambertova stínovacího modelu, z této rovnice se dopočítají normálové vektory v každém bodě. Druhá část algoritmu opravuje chybu, která vzniká při výpočtu normálových vektorů (vektory mají tendenci být natočené směrem ke světelnému zdroji). Chybu algoritmus opravuje tím způsobem, že mezi dvojici vektorů proloží kružnici tak, aby vektory tvořily tečné vektory dané kružnice. Tímto získáme relativní výšky mezi jednotlivými body. Z relativních výšek se poté spočítají absolutní výšky, což je výsledná výšková mapa.

3.1 Omezení algoritmu

Pokud má být výsledek rekonstrukce věrohodný, je vhodné, aby povrch ve vstupním obrázku byl jednoho odstínu barvy, viz [1].

Dále jsou požadované souřadnice tří nebo více vektorů, které uživatel musí před výpočtem zadat.

Algoritmus předpokládá, že scéna, která je předložena v obrázku, celá odpovídá Lambertovu stínovacímu modelu. Z tohoto faktu následně vycházejí všechny hlavní rovnice algoritmu.

3.1.1 Lambertovo stínovací model

Lambertovo stínování lze popsat jednoduchou rovnicí $I = \rho N^T L$, kde $I \in R$ je hodnota šedi daného pixelu,$\rho \in R$ je tzvn. albedo neboli odrazovost scény, $N \in R^3$ je normálový vektor v daném pixelu a $L \in R^3$ je vektor, který ukazuje ve směru nasvícení scény. Více viz [7].

3.2 Postup výpočtu

Výpočet lze rozložit do tří základních kroků (v článku [1] jsou to rovnice E1, E2 a E3), tyto kroky si probereme zvlášť:

1) Výpočet vektoru nasvětlení scény
2) Výpočet normálových vektorů pro jednotlivé pixely
3) Rekonstrukce výškové mapy z normálových vektorů

3.2.1 Výpočet vektoru nasvětlení scény

Hledáme: $L = (l_x, l_y, l_z); L \in R^3; \|L\| = 1$

Vstup: $A \in R^{n \times m}$, kde A je matice vstupního obrázku výšky n a šířky m, $a_{ij} \in R^3$ je vektor, který po složkách obsahuje hodnoty r,g,b.

Jako první potřebujeme z původního obrázku získat mapu šedi, označme si její matici I. Odstín šedi $i \in R$ pro pixel a vypočítáme, dle článku [8], následovně:

$$i = 0.2126 \times a_r + 0.7152 \times a_g + 0.0722 \times a_b$$

a_r je intenzita červené barvy v pixelu a, a_g je intenzita zelené barvy v pixelu a, a_b je intenzita modré barvy v pixelu a. Zvolený model výpočtu hodnoty šedi jsem zvolil z toho důvodu, že v konstantách reflektuje citlivost lidského oka na různé vlnové délky viditelného spektra, například na zelenou barvu je lidské oko nejcitlivější, proto má složka se zelenou barvou konstantu s největší hodnotou.

Pak pro matici I platí následující: $I \in R^{n \times m}$ je matice odstínu šedi původního obrázku,$i_{ij} \in R$ je odstín šedi v daném pixelu (i, j).

Nyní potřebujeme druhý vstupní parametr algoritmu - tři nebo více vektorů. Pro každý zadaný vektor n od uživatele musí platit následující:

a) Vektor musí mít zadanou svou pozici na obrázku - pixel, k jakému je vektor přiřazen.

b) Vektor musí co nejvíce odpovídat normálovému vektoru původního povrchu v daném pixelu.

c) Dané hodnotě šedi $i_{ij} \in R$ musí být přiřazen pouze jeden vektor, respektive musí platit následující:

Mějme zadané vektory u a v. Vektor u má zadanou svou pozici na pixelu a, vektor v má zadanou svou pozici na pixelu b. Pixel a má v matici I svou hodnotu šedi i_a a pixel b má v matici I svou hodnotu šedi i_b. Pokud platí $i_a = i_b + \delta$, kde $\delta \in R$ je odchylka, pak pro zadané vektory u a v musí platit $u = v$. Jinak nejsou zadané vektory validní a rekonstrukce povrchu může být díky závislosti vektorů značně degenerovaná.

d) Všechny vektory jsou jednotkové délky. Neboli platí pro každý vektor u, že $\|u\| = 1$.

Naši úlohu získání vektoru směru nasvětlení lze získat vyřešením přeurčené soustavy rovnic: $i_a - \rho \times n_a^T L = 0$

kde i_a značí hodnotu šedi v pixelu a a n_a značí zadaný normálový vektor pro pixel a. Rovnice přímo vychází z rovnice pro Lambertovo stínovací model $I = \rho N^T L$. Těchto rovnic bude Q, neboli tolik, kolik uživatel zadal vektorů na vstupu. Bez újmy na obecnosti předpokládejme, že soustava nemá řešení.

Budeme tedy hledat optimální řešení ve smyslu nejmenších čtverců. Rovnice přepíšeme jako optimalizační úlohu: $min(\sum_a^Q \left\| i_a - \rho \times n_a^T L' \right\|^2)$, kde naše neznámá je vektor L'.

Toto lze přepsat maticově: $min(\sum_a^N \left\| I - \rho \times N^T L' \right\|^2)$, kde $N \in R^{N \times 3}$ je matice, která v řádcích obsahuje složky x, y, z daných normálových vektorů.

Neboli řešíme soustavu $N^T L' = \frac{1}{\rho} \times I$. Získaný vektor L' znormalizujeme: $L = \frac{L'}{\|L'\|}$

3.2.2 Výpočet normálových vektorů pro jednotlivé pixely

V této části potřebujeme vypočítat normálové vektory pro jednotlivé pixely, neboli z rovnice pro Lambertovo stínovací model $I = \rho N^T L$ je naše neznámá vektor N. Po normálových vektorech bychom měli požadovat jednotkovou velikost, tudíž bychom měli řešit minimalizační úlohu s omezením, že všechny vypočítané normálové vektory musí být normalizované, řešení takovéto úlohy ale není vůbec jednoduché. V článku [1] obešli problém s omezující podmínkou tím, že minimalizaci počítali bez omezení a každý vektor N normalizovali až po jeho spočtení. My to uděláme obdobným způsobem. K rovnici $I = \rho N^T L$ potřebujeme přidat tzvn. regularizační člen, který má tvar:

$$\lambda min(\sum_{i,j} \|N_i - N_j\|^2)$$

,kde N_i a N_j jsou normálové vektory sousedících pixelů. Regularizační člen nám zajistí, že normálové vektory na sebe budou "navazovat" a tvořit tak souvislý povrch, jedná se vlastně o uhlazovací funkci, která nám určuje, jak moc si mají sousední vektory být podobné. Míru zahlazení nám určuje člen λ, kde $\lambda \in \langle 0, 1 \rangle$.

Regularizaci použijeme od daného pixelu do čtyř směrů. Označme si náš aktuální pixel jako $a = (x, y)$, kde x je souřadnice pixelu v ose x a y je souřadnice pixelu v ose y, potom regularizační normy pro tento pixel budou celkem čtyři, neboli bude regularizován se čtyřmi okolními pixely. Ony čtyři okolní pixely jsou $(x+1, y); (x-1, y); (x, y+1); (x, y-1)$. Regularizační rovnice pro pixel a vypadá poté následovně:

$$\lambda min(\|a - (x+1, y)\|^2 + \|a - (x-1, y)\|^2 + \|a - (x, y+1)\|^2 + \|a - (x, y-1)\|^2)$$

Jelikož zde počítáme s vektorem L', který jsme si spočítali v minulém kroku (vektor ukazující směr osvětlení scény) a tento vektor byl určen minimalizací přeurčené soustavy, můžeme bez újmy na obecnosti opět očekávat, že soustava rovnic nebude mít řešení, proto si jí opět převedeme na řešení soustavy ve smyslu nejmenších čtverců:

$$min(\sum_{a}^{n \times m} \left\| \frac{1}{\rho} i_a - n_a^T L' \right\|^2) + \lambda min(\sum_{i,j} \|n_i - n_j\|^2)$$

Minimalizační úlohu převedeme do maticového tvaru: $A \times N = \frac{1}{\rho} I$, kde matice $N \in R^{1 \times (nm)}$ je matice obsahující v jednom sloupci jednotlivé složky všech normálových vektorů. Složky jsou rozmístěny následovně: $N = [n_x^1 n_y^1 n_z^1 ... n_x^{n \times m} n_y^{n \times m} n_z^{n \times m}]^T$. Matice $I \in R^{1 \times (nm)}$ je matice do sloupců zarovnaných hodnot šedi. Poznamenejme následující: pro pixel p platí, že hodnota šedi i je stejná pro všechny složky normálového vektoru pixelu p, tudíž matice I má tvar: $I = [i_1 i_1 i_1 ... i_{n \times m} i_{n \times m} i_{n \times m}]^T$.

Matice $A \in R^{(nm) \times (nm)}$ nám popisuje vztah pixelu s jeho hodnotou šedi a také jeho vztah k sousedním pixelům (dáno regularizační funkcí). Nalézt minimum této funkce není výpočetně jednoduché (matice A má rozměr třikrát počtu pixelů na druhou), proto použijeme iterační algoritmus, takzvanou Gauss-Seidelovu metodu.

Gauss-Seidelova metoda je metoda na řešení optimalizačních úloh iteračním zlepšováním parciálního optima. Fungování Gauss-Seidelovy metody si uvedeme na následujícím příkladě.

Mějme přeurčenou soustavu lineárních rovnic, která je dána vztahem $Ax = b$. Soustava nemá řešení, proto chceme nalézt přibližné řešení ve smyslu nejmenších čtverců, neboli řešit soustavu $min(\|Ax - b\|^2)$. Tato soustava by se dala řešit pomocí pseudoinverze nebo QR rozkladu, ale kvůli množství rovnic a neznámých nelze tuto metodu použít. V tomto případě použijeme iterační Gauss-Seidelovu metodu, viz [9].

Nechť $x = [x_1..x_n]^T$, předpokládejme, že členy $x_1, ..., x_{i-1}$ a $x_{i+1}, ..., x_n$ mají již svou optimální hodnotu. Poté budeme řešit pouze minimalizaci úlohy $min(\|Ax - b\|^2)$ pro jednu neznámou (to jest pro x_i). Tato rovnice je snadno algoritmicky řešitelná.

Jakmile vypočítáme naše parciálně optimální x_i , tak tento člen zafixujeme, to jest: budeme předpokládat, že nabývá optimální hodnoty. Nyní budeme předpokládat, že neznámé $x_1, ..., x_i$ a $x_{i+2}, ..., x_n$ mají již svou optimální hodnotu. A řešíme rovnici $min(\|Ax - b\|^2)$ nyní pro neznámou x_{i+1}.

Tento algoritmus konverguje k optimální hodnotě vektoru x. Na inicializačních hodnotách $x_1, ..., x_n$ nezáleží (respektive ovlivní pouze rychlost konvergence).

3.2.3 Rekonstrukce výškové mapy z normálových vektorů

Z předchozí části jsme jako výstup získali mapu, ve které jsme pro každý pixel získali normálovou složku $n = (x, y, z)$. Poslední část algoritmu spočívá v tom, že z normálových vektorů jednotlivých pixelů zjistíme relativní výšky mezi sousedními pixely. Z těchto získaných relativních výšek následně spočítáme pro každý pixel jeho výšku absolutní. Začneme s výpočtem relativních výšek. Pro pixel $a = (x, y)$ (to jest je na pozici x a y) budeme počítat dvě relativní výšky a to relativní výšku mezi pixelem a a pixelem $b = (x + 1, y)$ a mezi pixelem a a pixelem $c = (x, y + 1)$. Výpočet relativní výšky mezi pixelem a a pixelem b vypadá následovně:

Pixel b je od pixelu a vpravo. Normálové vektory obou dvou pixelů si promítneme na dvourozměrnou plochu, kde průmět povedeme ve směru osy y (pixel b je vpravo od a, tudíž na plynulý přechod povrchu, který bude normálami tvořen, nemá složka y žádný vliv). Dostaneme tedy následující informace:

Pro pixel a máme normálový vektor $n_a = (x_a, y_a)$ a pro pixel b máme normálový vektor $n_b = (x_b, y_b)$. Jako vzálenost sousedících pixelů zvolme 1. Bez újmy na obecnosti předpokládejme, že vektor n_a je umístěn tak, že začíná z nulové výšky, to jest $h_a = 0$ a navíc leží v počátku souřadného systému. Dále víme, že vektor n_b je od n_a vzdálen o 1.

A my chceme určit výšku h_b tak, aby mezi vektory n_a a n_b byl co nejplynulejší přechod. Jako náš plynulý přechod je přirozené zvolit kružnici, proto definujme úlohu tak, že hledáme takovou kružnici, kde vektory n_a a n_b jsou jejími normálovými vektory. Po proložení kružnice těmito vektory nám vyjde výška h_b, ve které musí vektor n_b být, aby byl normálovým vektorem dané kružnice. Pokud zachováme orientaci vektorů ($n_b \neq -n_b$), pak tento výsledek je pouze jeden, viz [1].

Dále mějme pixel c, zde bude výpočet téměř totožný až na fakt, že c je umístěno pod pixelem a (o řádek níže), proto promítneme normály obou pixelů ve směru osy x, tudíž budeme složku x ignorovat.

S takto vypočítaným polem relativních výšek přejdeme na rekonstrukci samotného povrchu, neboli na počítání absolutních výšek. Při výpočtu relativních výšek je více než pravděpodobné, že jsme si do spočítaných výšek zanesli chybu. Abychom rozložili chybu rovnoměrně po celém povrchu musíme použít minimalizační funkci následujícího tvaru:

$$min(\sum_{i,j}((h_i - h_j) - q_{ij})^2)$$

, kde h_i je absolutní výška pixelu i , h_j je absolutní výška pixelu j a q_{ij} je relativní výška mezi pixelem i a j.

Tato funkce nám říká, že pokud odečteme absolutní výšky daných sousedících pixelů, tak jejich rozdíl se musí rovnat jejich relativní výšce. Zde jsou neznámé všechny absolutní výšky h_i, hodnoty q_{ij} jsme si spočítali v předchozím kroku. Tuto sumu snadno upravíme do maticového tvaru $min(\|AH - Q\|^2)$, kde matice H je sloupcový vektor, který má v sobě hodnoty absolutních výšek h_i, matice Q je sloupcový vektor, který má v sobě hodnoty relativních výšek. Matice A nám definuje vztah odečtu absolutních výšek $(h_i - h_j)$ sousedících pixelů. Pro tuto minimalizaci opět použijeme Gauss-Seidelovu iterační metodu.

Tímto jsme získali výškovou mapu našeho povrchu. Nyní přejdeme na výpočet normálové mapy z mapy výškové.

Kapitola 4
Výpočet normálových map za pomoci Sobel-Feldmanova operátoru

V předchozím kroku jsme vypočítali výškovou mapu ze zadaného obrázku, nyní převedeme informaci výšek na normálové vektory povrchu a tím vytvoříme normálovou mapu. Pro převod použijeme Sobel-Feldmanův operátor, který využívá masku konvoluce. Nejdříve popíšeme, co konvoluce je a jak jí využijeme.

4.1 Konvoluce

Konvoluce je matematický operátor pro dvě zobrazení. Tento operátor značíme $*$ a je dán následujícím vztahem: $(f*g)(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da$, kde funkce $g(x)$ se nazývá konvoluční jádro, viz [10]. V našem případě nám funkce f definuje vstupní obrázek (výškovou mapu) a funkce g náš normalizační filtr (konvoluční masku Sobel-Feldmanova operátoru). Jelikož v obraze pracujeme s diskrétním množstvím veličin, můžeme vztah přepsat do následující podoby: $(f*g)(x) = \sum_{i=-n}^{n} f(i)g(x-i)$. Tento vztah znamená, že pro každý pixel i z výchozího obrázku, je funkcí g jasně stanovena jeho hodnota, tuto hodnotu funkce g a výchozí hodnotu funkce f následně vynásobíme, sečteme s ostatním okolím bodu a dostaneme novou hodnotu pixelu na pozici x. Sumou od -n do n procházíme tzv. konvoluční masku pixelu x. Tato maska je většinou okolí pixelu x o velikosti 3×3 pixely. Podle nastavení různých hodnot této masky, ale i velikosti této masky, můžeme zcela měnit vlastnosti algoritmu. My použijeme algoritmus konvoluce, který je popsán v následujícím kroku.

4.2 Konvoluční maska

Pro bod a na souřadnicích (x, y) je v tabulce 4.1 dán předpis pro konvoluční masku 3×3.

(x-1,y-1)	(x,y-1)	(x+1,y-1)
(x-1,y)	(x,y)	(x+1,y)
(x-1,y+1)	(x,y+1)	(x+1,y+1)

Tabulka 4.1. Vztahy sousednosti v konvoluční masce 3×3.

Konvoluční maska 3×3 má následně tvar 4.2.

a_1	a_2	a_3
a_4	a_5	a_6
a_7	a_8	a_9

Tabulka 4.2. Konvoluční maska 3×3.

, kde $a_i \in R$. Masku aplikujeme na bod a následujícím způsobem (funkce $f(a_i)$ vrací hodnotu šedi v daném pixelu): $\bar{a} = \sum_{i=1}^{9} a_i \times f(a_i)$. Hodnota $\bar{a}$ je pak naše nová hodnota pro bod na souřadnicích (x, y).

4.3 Výběr konvolučního operátoru

Při výběru správného operátoru jsem se řídil knihou Image Processing, Analysis and Machine Vision, viz [11]. Kniha poskytla informace o několika operátorech pro detekci hran, právě tyto operátory lze využít k výpočtu normálových map.

4.3.1 Robertsův operátor

Tento operátor používá dvě konvoluční masky o velikosti 2×2 pixely, první maska má tvar 4.3 a druhá 4.4.

1	0
0	-1

Tabulka 4.3. Robertsův operátor, 1. maska.

0	1
-1	0

Tabulka 4.4. Robertsův operátor, 2. maska.

Hodnota hrany se následně vypočítá jako:

$$| g(i,j) - g(i+1,j+1) | + | g(i,j+1) - g(i+1,j) |$$

Jednou z hlavních nevýhod Robertsonova operátoru je jeho náchylnost vůči odchylkám, zapříčiněná malou konvoluční maskou, která aproximuje gradient v daném bodě, viz [11].

4.3.2 Laplaceův operátor

Laplaceův operátor používá dvě masky o rozměrech 3×3, první maska má tvar 4.5 a druhá 4.6.

0	1	0
1	-4	1
0	1	0

Tabulka 4.5. Laplaceův operátor, 1. maska.

0	1	0
1	-8	1
0	1	0

Tabulka 4.6. Laplaceův operátor, 2. maska.

Laplaceův operátor aproximuje gradient druhou derivací, což zaručuje větší stabilitu vůči výchylkám v obraze. Laplaceův operátor jsem si nevybral kvůli tomu, že některé hrany v obrázku dostanou větší váhu, než by měly, viz [11].

4.3.3 Sobel-Feldmanův operátor

Sobel-Feldmanův operátor používá dvě masky o rozměrech 3×3. První maska, pojmenujme ji Gx, je uvedena v tabulce 4.7.

-1	0	1
-2	0	2
-1	0	1

Tabulka 4.7. Sobel-Feldmanův operátor, 1. maska.

Druhá maska, pojmenujme ji Gy, je uvedena v tabulce 4.8

1	2	1
0	0	0
-1	-2	-1

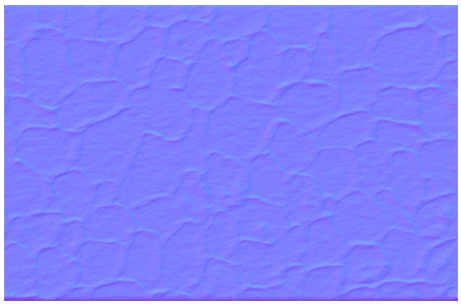
Tabulka 4.8. Sobel-Feldmanův operátor, 2. maska.

Sobelův operátor jsem si vybral kvůli tomu, že nemá nevýhody výše zmíněných operátorů a je velice rychlý, například oproti Canny edge detektoru, viz [12]

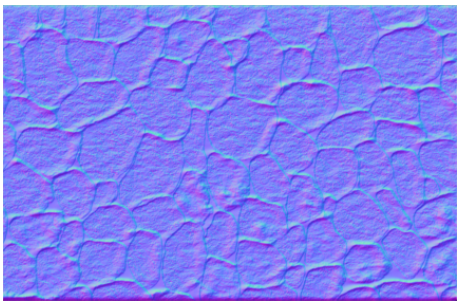
4.4 Výpočet normálové mapy

Pro výpočet normálové mapy použijeme konvoluční masky Sobel-Feldmanova operátoru (4.7 a 4.8). Vezměme si pixel z naší výškové mapy a, pro tento pixel budeme počítat jeho normálový pixel $b = (r, g, b)$, b je pixel o třech barevných hodnotách r, g, b které nabývají hodnot 0 až 255. Víme, že normálový vektor je definován třemi složkami: x, y, z, tuto informaci chceme zapsat do obrázku, proto použijeme přiřazení $b = (r, g, b) = (x, y, z)$, neboli červená složka pixelu b bude představovat hodnotu x pro daný normálový vektor, zelená složka bude představovat hodnotu y a modrá složka hodnotu z.

Nyní použijeme na pixel a konvoluční masku Gx, výsledná hodnota bude naše složka x. Obdobně použijeme konvoluční masku Gy, výsledná hodnota bude naše hodnota y v normálovém pixelu b. Poslední složka v našem pixelu, která nám chybí, je složka z. Tato hodnota nám určuje, jakou bude mít normálový vektor "výšku" (velikost ve směru osy z), pro ukázku uvádím dva případy 4.1 a 4.2.



Obrázek 4.1. Normálová mapa s hodnotou z = 150



Obrázek 4.2. Normálová mapa s hodnotou z = 230

Lze si povšimnout, že normálová mapa vpravo má mnohem výraznější okraje a rysy povrchu než mapa vlevo. Toto má za následek právě složka z v našem vektoru. Po získání všech složek je musíme normalizovat a promítnout na prostor $\{i|i \in N, i \in \langle 0, 255\rangle\}$. Následně zapíšeme do pixelu b. Tento proces použijeme na všechny pixely.

Všimněme si, že konvoluční maska předpokládá, že pro každý pixel jsou definovány sousední pixely ve všech osmi směrech, což ale obecně neplatí (například v rohu obrázku, kde jsou pouze tři). Pro tyto případy normály buďto nepočítáme (viz [13]), popřípadě můžeme přidat hranici. Tato hranice bude okolo celého obrázku a její tloušťka bude jeden pixel (pro naší konvoluční masku 3×3). Hodnotu pixelů v hranici nastavíme na šedou barvu, respektive na polovinu spektra, kterého náš pixel může nabývat, více v článku [14]. V našem případě může pixel nabývat hodnot od 0 do 255, hodnotu pixelů v hranici bychom tedy nastavili na 127 (nebo 128 dle zaokrouhlování).

Kapitola 5
Implementace algoritmu

V této kapitole popíši, jak jsem implementoval algoritmus Interactive Normal Reconstruction from a Single Image. Celý algoritmus jsem implementoval v jazyku Java 8. Využíval jsem jen standartních knihoven Javy, pouze v kapitole 5.1 jsem použil knihovnu Jama, viz [15].

5.1 Výpočet vektoru nasvětlení scény ze zadaných hodnot

Pro výpočet jsem použil vzorec $I = \rho N^T L$, od uživatele jsou zadány minimálně 3 normálové vektory N a já potřebuji spočítat vektor L. Rovnici jsem přepsal do následujícího tvaru $N^T L = \frac{I}{\rho}$. Tato rovnice v maticovém tvaru pro všechny vektory má nasledující tvar: $[N_1^T ... N_n^T]^T L = [\frac{I_1}{\rho} ... \frac{I_n}{\rho}]^T$. Normálové vektory od uživatele jsem před výpočtem normalizoval. Hodnoty I jsem vypočítal z původního obrázku následujícím způsobem:

$$I = 0.2126 \times r + 0.7152 \times g + 0.0722 \times b$$

Výsledný vektor L jsem opět normalizoval.

5.2 Výpočet normálových vektorů

Po vypočtení vektoru nasvětlení popíši výpočet normálových vektorů. Pro tento účel jsem použil rovnici E2, viz [1]: $min(\sum_a^{n \times m} \left\| \frac{1}{\rho} i_a - n_a^T L' \right\|^2) + \lambda min(\sum_{i,j} \|n_i - n_j\|^2)$.

Zprvu jsem zkoušel vyřešit tento případ stejně, jako jsem řešil výpočet vektoru nasvětlení, tudíž pomocí matic. Tento postup se však ukázal okamžitě jako nevhodný, neboť Java hlásila nedostatek paměti už při obrázku, který měl velikost 150×150 pixelů. Proto jsem matice nepoužil a vyjádřil jsem si rovnici následujícím způsobem. Při použití Gauss-Seidelovy metody zůstane v rovnici pouze jeden neznámý vektor (n_x^i, n_y^i, n_z^i). Po vyjádření neznámé jsem obdržel následující vztah:

$$(\frac{1}{\rho} - n_x^i L_x - n_y^i L_y - n_z^i L_z)^2 + \lambda min(\sum_{i,j} \|n_i - n_j\|^2) + c$$

, kde S je množina sousedících pixelů (minimálně 2, maximálně 4) a c označuje konstantu, kterou tvoří ostatní prvky. Ze vztahu je vidět, že se jedná o kvadratickou funkci. Já jsem potřeboval najít minimum této funkce. Díky faktu, že funce je funcí paraboloidu, lze najít minimum velice jednoduše. Rovnici jsem zderivoval podle všech proměnných (n_x^i, n_y^i, n_z^i) a tím dostal její parciální derivace.

Následně jsem položil parciální derivace rovné nule a vyřešil soustavu tří rovnic o třech neznámých:

$$n_x^i = \frac{L_x(\frac{1}{\rho}I_i - n_y^i L_y - n_z^i L_z) + \lambda \sum_{s\in S} n_x^s}{L_x^2 + \lambda S}$$

$$n_y^i = \frac{L_y(\frac{1}{\rho}I_i - n_x^i L_x - n_z^i L_z) + \lambda \sum_{s\in S} n_y^s}{L_y^2 + \lambda S}$$

$$n_z^i = \frac{L_z(\frac{1}{\rho}I_i - n_x^i L_x - n_y^i L_y) + \lambda \sum_{s\in S} n_z^s}{L_z^2 + \lambda S}$$

Vím, že nalezený vektor bude minimem funkce, neboť funce je konvexní, tudíž má pouze jeden extrémní bod a tento bod je jejím minimem. Pro snazší popis výsledného vzorce jsem si zavedl několik konstant:

$$d = \frac{L_x L_y}{L_x^2 + \lambda S}, a = \frac{L_x L_y}{L_y^2 + \lambda S - L_x L_y d}, b = \frac{L_z L_y}{L_y^2 + \lambda S - L_x L_y d}, g = \frac{L_z L_x}{L_x^2 + \lambda S},$$

$$s_x = \sum_{s\in S} n_x^s, s_y = \sum_{s\in S} n_y^s, s_z = \sum_{s\in S} n_z^s$$

Následují vzorce pro výpočet všech složek vektoru:

$$n_z^i = \frac{\frac{1}{\rho}I_i(L_z - L_x g + L_y g a - L_y b - L_x d g a + L_x d b)}{L_z^2 + \lambda S + \delta - L_z L_y b + L_z L_x d b + L_z L_y a g - L_z L_x a d g - L_z L_x g}$$
$$+ \frac{\lambda(s_x a b - s_y b - s_x g + s_y a g - s_x d a g)}{L_z^2 + \lambda S + \delta - L_z L_y b + L_z L_x d b + L_z L_y a g - L_z L_x a d g - L_z L_x g}$$

$$n_y^i = \frac{\frac{1}{\rho}I_i(L_y - L_x d) + n_z^i L_z(L_x d - L_y) + \lambda(s_y - s_x d)}{L_y^2 + \lambda S - L_y L_x d}$$

$$n_x^i = \frac{L_x(\frac{1}{\rho}I_i - n_y^i L_y - n_z^i L_z) + \lambda s_x}{L_x^2 + \lambda S}$$

Díky tomu, že jsem mohl původní funkci zderivovat, jsem nemusel počítat součet všech rovnic pro celý obrázek, ale stačilo pouze dosadit do těchto vzorců (odpadlo vše skryté v konstantě c).

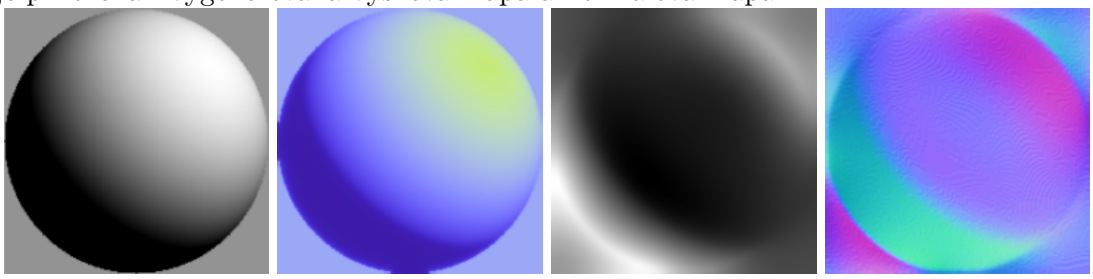
Po vyjádření vztahů jsem začal s implementací. Před započetím výpočtu jsem nastavil hodnotu všech vektorů na hodnotu validního jednotkového vektoru, to jest na hodnotu $(x, y, z) = (0, 0, 1)$. Tento krok jsem učinil, neboť po algoritmu vyžaduji validní jednotkové vektory, které nemají zápornou souřadnici z a nechtěl jsem, aby algoritmus vstupoval do výpočtu s nevalidními hodnotami, které nedávají smysl. Zejména kvůli regularizační části algoritmu je tento krok důležitý, neboť by se nově vypočítaný vektor snažil v prvním kole výpočtu „napodobit“ nulové vektory, které ale ve výsledku nechci (výstup musí mít všechny vektory jednotkové délky).

Po úvodním nastavení začal samotný výpočet normálových vektorů. Algoritmus počítá každý cyklus Gauss-Seidelovy metody pro každý pixel se stejně “starými” hodnotami. Dejme tomu, že počítá k-tý cyklus a je na pixelu i. Poté se jako hodnoty sousedních pixelů berou hodnoty z k-1 cyklu. Pro aplikování této vlastnosti jsem použil přepisovací buffer, aby se mi stará data nekryla s nově vypočítanými.

V bufferu jsem si zachovával nová data a propisoval je zpět do pole až do chvíle, kdy jsem si byl jist, že jejich hodnoty již nebudou v tomto cyklu výpočtu použity. Po každém spočítaném vektoru (to jest složek n_x^i, n_y^i a n_z^i) jsem si daný vektor normalizoval před jeho zápisem do bufferu. Takto sestavený algoritmus už nezpůsoboval problémy s nedostatkem paměti.

Při výpočtu se na normálové mapě v rozích obrázku vyskytovaly hodnoty, které byly zdaleka odlišné od zbytku obrázku a kazily výsledky. Usoudil jsem, že tyto hodnoty pocházejí z toho faktu, že v celém obrázku je $S = 4$ (4 sousedící pixely), pouze na okrajích je menší, proto se tam chyby mohou projevit razantněji (hodnoty se počítají z menší množiny sousedů). Tyto rohové hodnoty jsem proto nepočítal a do krajních pixelů jsem po spočtení středu obrázku zapsal hodnotu nejbližších pixelů, které už měly všechny 4 sousední pixely.

Zde 5.1 jsou ukázky vygenerované normálové mapy z tohoto algoritmu a jeho vstupu, je přiložena i vygenerovaná výšková mapa a normálová mapa.



Obrázek 5.1. Zleva doprava: vstupní obrázek, výstup z rovnice E2, vypočítaná výšková mapa, normálová mapa.

Je vidět již z výstupu, že jsou zde zaznamenány pouze dva směry: z pravého horního rohu směrem do levého dolního rohu. Tento fakt se potvrzuje i na vygenerované výškové mapě, která zcela postrádá klesání (stoupání) v levém horním a pravém dolním rohu. Algoritmus má takto pracovat, výstupy jsou správné, jen neposkytují dostatečnou informaci pro vygenerování správné výškové mapy. Abych zanesl do obrázku další dva směry, které zde chybí, rozhodl jsem se, že použiji přímo vstupní vektory od uživatele. Vstupní vektory jsem použil pro vytvoření podkladu, kterým by se měly vektory na výstupu algoritmu podobat. Pro vytvoření podkladu jsem použil poupravenou interpolaci. Je zadán set vstupních vektorů od uživatele, u každého vstupního vektoru je známá jeho souřadnice (x, y, z) a jeho pozice na obrázku (x_I, y_I). Chci vypočítat souřadnice vektoru $a = (x, y, z)$, který má pozici na obrázku (x_a, y_a). Všechny vstupní vektory od uživatele jsem si seřadil dle jejich vzdálenosti od vektoru a (neboli pomocí Euklidovské vzdálenosti $\|(x_a - x_I, y_a - y_I)\|$). Ten vektor, který je od a nejblíže (nazvu jej vektorem f) má vzdálenost h_f od vektoru a. Další vektory, které mají od a větší vzdálenost než h_f, jsem započítával do interpolace jen tehdy, jestliže platí následující:

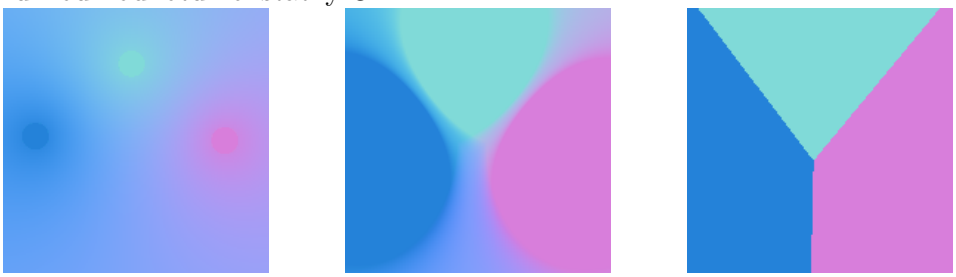
Mějme vektor g a jeho vzdálenost od a rovnou h_g, nechť platí $h_g \geq h_f$. Vektor g zahrneme do výpočtů tehdy, jestliže platí $h_g - h_f \leq C$, kde C je konstanta.

Nastavení konstanty C prodiskutuji později.

Měl jsem tedy vybranou sadu vektorů, které jsem chtěl zahrnout do výpočtu vektoru a. Chtěl jsem, aby nejbližší vektor měl při výpočtu nejvyšší váhu a nejvzdálenější aby měl váhu co nejmenší, proto jejich vzdálenosti od vektoru a dal rovny $h' = \frac{1}{h}$. Pro každý vektor jsem si spočetl sílu, s jakou bude mít vliv na výsledný vektor a. Pro nejbližší vektor f je jeho síla rovna $s_f = h'_f$. Pro ostatní vektory jsem vypočítal jejich sílu (vliv na výsledný vektor a) následovně $s_g = s_f - (\frac{s_f - h'_g}{C})s_f$. Po spočtení sil všech vektorů jsem sečetl jejich hodnoty $l = \sum_i s_i$.

Nyní popíši výpočet souřadnic samotného vektoru a.

Souřadnici x vektoru a jsem vypočetl vzorcem $x = \sum_i (\frac{s_i}{l})x_i$. Výsledné souřadnice vektoru a jsem normalizoval. Zde 5.2 jsou výsledky pro tři nastavené vstupní vektory a pro různou hodnotu konstatny C.



Obrázek 5.2. Zleva doprava: C = 0.1, C = 0.005, C = 0.0.

Je vidět, že čím je konstanta vyšší, tím je interpolační pás větší a plocha zaplněná barvou nejbližšího vektoru je menší. Naopak čím nižší konstanta, tím větší plocha je zaplněna barvou nejbližšího vektoru (interpolační pás je užší). Experimentálně jsem konstantu C nastavil na hodnotu $C = 0.005$. Přechody i po interpolaci byli relativně ostré, proto jsem ještě použil konvoluci, o velikosti masky 10×10, která zapisovala do každého pixelu průměr ze všech hodnot z masky. Jedná se rozmazání obrazu, výsledek pro $C = 0.005$ je na obrázku 5.3.

Obrázek 5.3. Interpolované vstupní vektory s rozmazáním.

Takto sestavený podklad jsem použil do upravené rovnice E2. Do původních rovnice E2 jsem musel přidat další regularizační člen:

$$min(\sum_{a}^{n\times m} \left\| \frac{1}{\rho} i_a - n_a^T L' \right\|^2) + \lambda min(\sum_{i,j} \|n_i - n_j\|^2) + \delta min(\sum_{i} \|n_i - d_i\|^2)$$

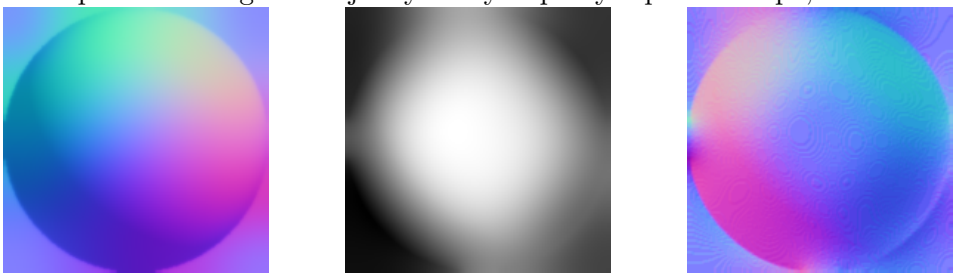
, tento poslední regularizační člen nám určuje (dle hodnoty konstanty δ), jak moc má být náš vektor n_i podobný vektoru d_i, který pochází z interpolace vstupních vektorů. Po novém vyjádření vztahů dostaneme následující rovnice:

$$n_z^i = \frac{\frac{1}{\rho} I_i(L_z - L_x g + L_y g a - L_y b - L_x d g a + L_x d b)}{L_z^2 + \lambda S + \delta - L_z L_y b + L_z L_x d b + L_z L_y a g - L_z L_x a d g - L_z L_x g}$$
$$+ \frac{\lambda(s_x a b - s_y b - s_x g + s_y a g - s_x d a g) + \delta(d_x a b - d_y b - d_x g + d_y a g - d_x d a g)}{L_z^2 + \lambda S + \delta - L_z L_y b + L_z L_x d b + L_z L_y a g - L_z L_x a d g - L_z L_x g}$$

$$n_y^i = \frac{\frac{1}{\rho} I_i(L_y - L_x d) + n_z^i L_z(L_x d - L_y) + \lambda(s_y - s_x d) + \delta(d_y - d_x d)}{L_y^2 + \lambda S + \delta - L_y L_x d}$$

$$n_x^i = \frac{\frac{1}{\rho} I_i - n_y^i L_y - n_z^i L_z) + \lambda s_x + \delta d_x}{L_x^2 + \lambda S + \delta}$$

Z takto upraveného algoritmu již výsledky dopadly o poznání lépe, viz 5.4.



Obrázek 5.4. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa.

Na výsledcích je vidět, že levý spodní roh je stále mnohem výraznější, než zbylé tři rohy okolo koule, nicméně tato nová úprava kompenzovala nedostatek informací o tvaru koule a výsledná výšková mapa již mnohem více připomíná výškovou mapu koule. Uživatel může dosáhnout lepších výsledků, pokud zadá více vstupních vektorů.

5.3 Výpočet výškové mapy

Výpočet jsem rozdělil do dvou částí. V první části pro každý pixel, který má sousední pixel pod sebou a vpravo od sebe, to jest pro (x, y) existuje $(x+1, y)$ nebo $(x, y+1)$, počítám relativní výšky mezi tímto pixelem a jeho dvěma sousedy (v krajních hodnotách s jeho jedním nebo žádným sousedem). V druhé části pak z těchto získaných relativních výšek mezi pixely vypočítám výšky absolutní, neboli výškovou mapu.

5.3.1 Výpočet relativních výšek mezi pixely

Postupoval jsem dle zadání v [1]. Při výpočtu relativní výšky mezi aktuálním pixelem a sousedním pixelem vpravo od něj jsem promítl jejich vektory ve směru osy y (tudíž hodnotu v ose y jsem zanedbal). Měl jsem tedy dva vektory $a = (a_x, a_z)$ a $b = (b_x, b_z)$. Tyto dva vektory jsem proložil kružnicí tak, aby tvořili normálové vektory dané kružnice. Vektor a jsem fixoval se začátkem v počátku, vektor b byl posunutý od a o 1 doprava. Výška vektoru b bude poté určovat relativní výšku mezi oběma vektory, tudíž mezi oběma pixely, kterým vektory přísluší. Výšku vektoru jsem spočítal následujícím vzorcem:

$$q_{ab}^2\left(\frac{-a_z b_x}{a_x} - b_z\right) + q_{ab}\left(\frac{2a_z b_z}{a_x} - 2b_x\right) + \left(\frac{a_z b_x b_z}{a_x}\right) = 0$$

Bohužel výsledky poskytnuté z této rovnice nelze použít přímo, neboť pro určité vektory mohou výsledky skončit v nekonečnu. Pro představu uvádím příklad:

Označme si úhel, který svírá vektor a s osou x jako α a úhel, který svírá vektor b s osou x jako β. Pokud například zvolíme úhel $\alpha = 180°$ a $\beta = 180°$, pak kružnice by musela mít nekonečně veliký poloměr, aby oba vektory byly jejími tečnými vektory, tudíž i relativní výška mezi vektory by byla rovna nekonečnu.

Kvůli těmto vlastnostem jsem zavedl tři omezující konstanty: $ROUND\_ANGLE$, $FLAT\_ANGLE$ a $MAX\_RELATIVE\_HEIGHT$. První konstanta nám určuje, jak moc se může úhel α odchýlit od úhlu β, abychom je stále považovali za stejný úhel, podmínka vypadala takto: $(\beta > (\alpha - ROUND\_ANGLE)) \wedge (\beta < (\alpha + ROUND\_ANGLE))$

Konstanty jsem testoval v kapitole 5.3.2. Pokud si úhly byly podobné, pak jsou dostupné dvě možnosti: zaprvé jsou oba dva úhly skoro rovny $0°$ (resp. $180°$) nebo zadruhé jsou jiné. Podmínku, zda jsou skoro rovny $0°$ či $180°$, jsem ověřoval pomocí druhé konstanty FLAT_ANGLE.

Pokud se ukázalo že jsou skoro rovny 0° (resp. 180°), pak jsem použil jako relativní výšku konstantu -MAX_RELATIVE_HEIGHT (resp. MAX_RELATIVE_HEIGHT). Pokud nebyly podobné 0° nebo 180° ale sobě byly podobné, pak jsem se nesnažil je proložit kružnicí, ale proložil jsem jimi přímku, neboli jsem použil vzorec $q_{ab} = -\frac{a_x}{a_z}$. Vzorec vychází z následující úvahy:

Mějme vektor t, který je kolmý na vektor $a = (a_x, a_z)$*, tudíž platí* $t = (-a_z, a_x)$*. Chceme nalézt bod B na přímce, která začíná v počátku a pokračuje ve směru vektoru t. Tento bod má souřadnice* $B = (1, q_{ab})$*. Dále víme z parametrického vyjádření přímky, že* $1 = -a_z k$*, kde k je parametr přímky. Poté musí platit* $q_{ab} = a_x k$*, vyjádříme* $k = -\frac{1}{a_z}$ *a po dosazení máme výsledný vzorec.*

Popsal jsem co algoritmus dělá, pokud nefunguje proložení na kružnici. Nyní popíši co se děje, když na kružnici vektory proložit lze. Vyjádřil jsem si proměnnou q_{ab} z kvadratické rovnice uvedené výše:

$$q_{ab} = -\frac{b_x - \frac{a_z b_z}{a_x}}{\frac{a_z b_x}{a_x} + b_z} \pm \frac{\sqrt{(\frac{a_z b_z}{a_x} - b_x)^2 + (\frac{a_z b_x}{a_x} + b_z)^2}}{-\frac{a_z b_x}{a_x} + b_z}$$

Aby mohla kvadratická rovnice vyjít, pak musí platit následující podmínka $\frac{a_z b_x}{a_x} + b_z \neq 0$. Pokud toto platí, pak má rovnice dvě řešení. Pouze jedno z řešení je správné, a to takové, které u dané kružnice zachová směr vektorů (neotočí je o 180°). To, jaký z oněch dvou výsledků to je, jsem zjistil jednoduše. Ze spočtených relativních výšek jsem si zpátky spočetl úhly α a β:

$$\beta_1 = \frac{q_{ab1} - \frac{a_z}{a_x}}{\frac{a_z b_x}{a_x} - b_z}, \beta_2 = \frac{q_{ab2} - \frac{a_z}{a_x}}{\frac{a_z b_x}{a_x} - b_z}$$

$$\alpha_1 = \frac{\beta_1 b_x + 1}{a_x}, \alpha_2 = \frac{\beta_2 b_x + 1}{a_x}$$

Správná relativní výška je pak taková, pro kterou vyšlo $\alpha_i \geq 0$ a $\beta_i \geq 0$.

Pokud neplatí podmínka $\frac{a_z b_x}{a_x} + b_z \neq 0$ kvadratická rovnice se zjednodušší na výraz $q_{ab}(\frac{2a_z b_z}{a_x} - 2b_x) = 0$ z čehož je vidět, že $q_{ab} = 0$.

Při výpočtu se stávalo, že i když jsem proložil hodnoty kružnicí nebo přímkou, tak překročily hodnotu MAX_RELATIVE_HEIGHT, proto jsem na konec ještě zavedl podmínku, která hodnoty větší než konstanta (menší než mínus konstanta) zaokrouhlí na hodnotu konstanty (mínus konstanty). Poté jsem spočítal ještě relativní výšku mezi pixelem a a pixelem $c = (x, y + 1)$. Zde jsem promítnul vektory ve směru osy x (x-ovou složku jsem zanedbal). Zbytek výpočtů byl totožný. Hodnoty jsem ukládal do pole s hodnotami ve formátu double, pro každý pixel zde byla připravena dvě čísla double, první číslo pro relativní výšku mezi naším pixelem a pixelem vpravo a druhé číslo pro relativní výšku mezi naším pixelem a pixelem pod ním.

5.3.2 Testování konstant

Nyní uvedu testování samotných konstant. Nastavení konstatny ROUND_ANGLE jsem odvodil z experimentálního testování. Konstantu jsem testoval na obrázku 5.5.

Obrázek 5.5. Obrázek mince pro testování konstant. Převzato z adresy [1]).

Výstupy jsou výškové mapy z již přepočtených relativních výšek na výšky absolutní. Při malém ROUND_ANGLE se na výstupu objevovaly artefakty, hlavně na líci hlavy, viz obrázek 5.6.

Obrázek 5.6. Vypálená místa na vygenerované normálové mapě, ROUND_ANGLE = 0.1

Při moc velkém ROUND_ANGLE se začal deformovat povrch nesprávným směrem (opět nejzřetelnější na líci hlavy), jak je vidět zde 5.7.

Obrázek 5.7. Vypálená místa na vygenerované normálové mapě, ROUND_ANGLE = 45.0

Konstantu ROUND_ANGLE jsem experimentálně nastavil na hodnotu 2.0, která byla kompromisem mezi obojím.

[1]) `https://s-media-cache-ak0.pinimg.com/originals/a4/07/0c/a4070cc369aa9b475f107f49ffb7ba78.jpg`

Nastavení konstatny FLAT_ANGLE jsem taktéž odvodil z experimentálního testování. Konstantu jsem opět testoval na obrázku mince. Při malém FLAT_ANGLE tval výpočet mnohem déle, neboť se téměř všechny vektory prokládaly kružnicí. Pro naší minci bylo spoždění algoritmu o 0.5 sekundy. Pro obrázek dvakrát větší byl už rozdíl 3 sekundy.

Při moc velkém FLAT_ANGLE se začaly potlačovat detaily a objevil se šum, jak je vidět zde 5.8.

Obrázek 5.8. Šum při velké hodnotě FLAT_ANGLE, FLAT_ANGLE = 45.0

Konstantu FLAT_ANGLE jsem experimentálně nastavil na hodnotu 5.0. Nastavení konstatny MAX_RELATIVE_HEIGHT jsem opět odvodil z experimentálního testování. Konstantu jsem také testoval na obrázku mince. Při malém MAX_RELATIVE_HEIGHT byly skoro všechny přechody mezi pixely ohodnoceny stejně, proto obrázek nemá téměr žádné jemné přechody, ale pouze “tvrdé” skoky, ukázka na obrázku 5.9.

Obrázek 5.9. Zánik jemných přechodů, MAX_RELATIVE_HEIGHT = 0.1

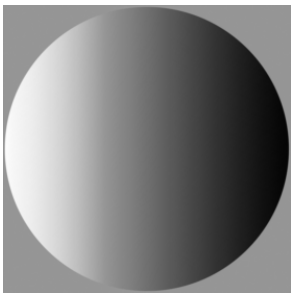
Při moc velkém MAX_RELATIVE_HEIGHT se detaily potlačily a vystoupily pouze nejvýraznější prvky, kde relativní výška právě dosahovala hodnot konstanty, viz 5.10.

Obrázek 5.10. Zánik detailů, MAX_RELATIVE_HEIGHT = 20.0

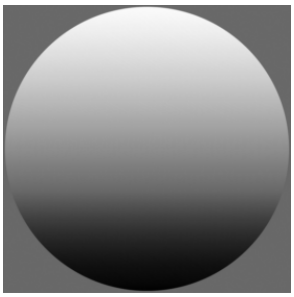
Konstantu MAX_RELATIVE_HEIGHT jsem nastavil na hodnotu 5.0.

Správné fungování algoritmu jsem ověřil tak, že jsem na vstup dal normálovou mapu 5.11 a z dané normálové mapy jsem si nechal spočítat relativní výšky. Tyto relativní výšky jsem poté zapsal do obrázků, v prvním obrázku 5.12 je pro každý pixel zapsána relativní výška mezi tímto pixelem a pixelem vpravo od něj, v druhém obrázku 5.13 je poté zapsána relativní výška mezi naším pixelem a pixelem pod ním.

Obrázek 5.11. Vstupní obrázek normálové mapy koule (obrázek převzat z adresy [1]).



Obrázek 5.12. Obrázek popisující relativní výšky směrem doprava



Obrázek 5.13. Obrázek popisující relativní výšky směrem dolů

Výsledky správně popisují povrch koule. Pokud je v pixelu i zapsána světlá barva, pak to znamená, že pixel j (což je pixel vpravo respektive pod ním) je víš než pixel i. Opačně to platí pro černou barvu, šedá barva znamená, že se relativní výška nemění.

5.3.3 Výpočet absolutních výšek

Po vypočtení relativních výšek mezi jednotlivými pixely jsem přepočítal relativní výšky na absolutní, abych získal výslednou výškovou mapu. Tato část algoritmu implementuje rovnici E3 z [1]: $min \sum_{i,j} ((h_i - h_j) - q_{ij})^2$. Minimalizaci jsem opět řešil Gauss-Seidelovou iterační minimalizací. Všechny proměnné h_j jsem si zafixoval až na jednu, označím si ji h_i, toto bude nyní naše proměnná. Pokud se toto učiní, tak z původní minimalizační úlohy vznikne jednodušší kvadratická rovnice. Použiji stejnou vlastnost jakou jsem již použil u výpočtu normálových vektorů. Jestliže potřebuji najít minimum takovéto rovnice, tak použiji fakt, že se jedná o konvexní funkci. Pokud funkci zderivuji a najdu její nulový bod (bude jen jeden), tak platí, že jsem našel minimum této funkce.

Po derivaci rovnice jsem obdržel následující výraz:

$$h_i = \frac{\sum_{j \in J} (h_j + q_{ij}) + \sum_{k \in K} (h_k + q_{ki})}{S}$$

, kde množina J obsahuje pixely, které jsou od našeho pixelu i vpravo a dole, množina K obsahuje pixely, které jsou od našeho pixelu i vlevo a nahoře. S je počet sousedících pixelů. Z výrazu je vidět, že se jedná o aritmetický průměr z hodnot $h_i = q_{ij} + h_j$ a $h_i = h_k - q_{ki}$, kde h_j je hodnota výšky pixelu vpravo (respektive dole) od naše pixelu i a h_k je hodnota výšky pixelu vlevo (respektive nahoře) od naše pixelu i. Při výpočtu výškové mapy tímto způsobem se mi v obrázku na okrajích v mnoha případech objevily extrémní hodnoty (buďto zcela bílá, či zcela černá barva) jak je vidět na ukázce 5.14.

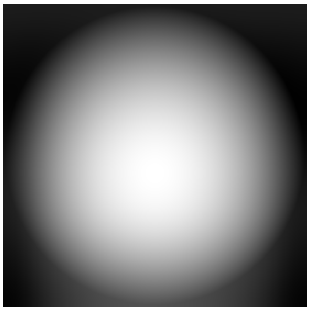
Obrázek 5.14. Extrémy na okrajích obrázku

Tento nechtěnný efekt byl pravděpodobně způsoben tím, jakým způsobem jsem počítal absolutní výšky v krajních bodech. V krajních bodech se počítal aritmetický průměr jenom ze dvou (respektive ze tří hodnot). Důsledek efektu byl takový, že při mapování vypočtených hodnot výšek na hodnoty 0 – 255 (při zápisu do obrázku) byly hodnoty uvnitř obrázku mnohé zaokrouhleny do stejného čísla, neboť dané extrémy tvořily hranice rozsahu. Již na vykreslené výškové mapě je zřejmé, že vykreslený střed výškové mapy nevyužívá celého rozsahu odstínů šedi, ale je zúžen jen na velmi malý rozsah. Extrémy na okrajích jsem vyřešil tím, že výšku v krajních bodech nepočítám a pouze do daného krajního bodu zapíši hodnotu výšky nejbližšího sousedního pixelu, který již má všechny čtyři sousedy. Po této úpravě efekt prakticky zmizel a namapování hodnot výšek na prostor 0 – 255 již dopadlo o poznání lépe, viz 5.15.

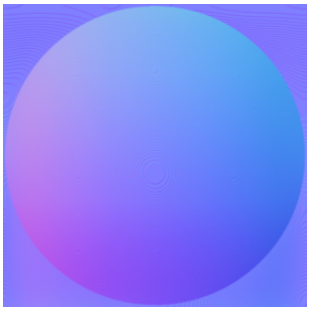
Obrázek 5.15. Opravené extrémy na okrajích obrázku

Opět jsem použil přepisovací buffer (stejně jako při výpočtu normálových vektorů v kapitole 5.2), do kterého si ukládám nově vypočítané hodnoty a dané hodnoty přepisuji zpět do pole obrázku až ve chvíly, kdy jsem si jist, že se s nimi v daném kole výpočtu nebude již počítat. Tímto přepisovacím bufferem zajistím, že se mi stará data nebudou plést s nově vypočítanými a že každý pixel v poli bude počítat se stejně „starými“ daty.

Algoritmus jsem opět otestoval na normálové mapě koule jako v případě výpočtu relativních výšek. Vstupem algoritmu bylo pole relativních výšek vypočítané v předchozí části (5.12 a 5.13). Výstupem algoritmu je výšková mapa 5.16, tuto mapu použijeme v následující kapitole pro výpočet mapy normálové 5.17. Tím jsem dokončil popis implementace algoritmu Interactive normal reconstruction from single image.



Obrázek 5.16. Vypočítaná výšková mapa



Obrázek 5.17. Vypočítaná normálová mapa

5.4 Výpočet normálové mapy

V předchozí kapitole jsem popsal postup pro výpočet výškové mapy. Tuto mapu nyní použiji pro výpočet normálové mapy. Pro výpočet použiji Sobel-Feldmanův operátor. Budu se věnovat jenom pixelům, které mají všechny čtyři sousední pixely, krajní hodnoty jsem nepočítal, vždy jsem do krajního pixelu jen zapsal hodnotu nejbližšího pixelu, který již měl všechny čtyři sousedy. Jako vstup do tohoto algoritmu budou tři věci. První je výšková mapa, druhá je z-tová složka vektorů (výška mapy, kterou určuje uživatel) a třetí, poslední vstup, je úhel normálových vektorů, který také zadá uživatel. Musíme tedy pro každý pixel spočítat složky x a y. Pixel má hodnoty ve složkách r, g, b a přiřazení složek x, y, z bude následující $(x, y, z) = (r, g, b)$.

Je dán pixel normálové mapy $a = (x, y, z) = (r, g, b)$, pro spočtení x-ové složky pixelu a jsem použil následující konvoluční masku 4.7 ze Sobel-Feldmanova operátoru, kterou jsem aplikoval na hodnoty výškové mapy. Konvoluční maska pro složku x vyjadřuje tento vzorec:

$$x = -h_{x-1,y-1} + h_{x+1,y-1} - 2h_{x-1,y} + 2h_{x+1,y} - h_{x-1,y+1} + h_{x+1,y+1}$$

Obdobně pro složku y jsem ze Sobel-Feldmanova operátoru použil konvoluční masku 4.8. Vzorec pro y-ovou složku je:

$$y = h_{x-1,y-1} + 2h_{x,y-1} + h_{x+1,y-1} - h_{x-1,y+1} - 2h_{x,y+1} - h_{x+1,y+1}$$

Normálové vektory jsem poté otočil v úhlu α, který uživatel zadal na vstupu:

$$x' = xcos(\alpha) - ysin(\alpha)$$
$$y' = ycos(\alpha) + xsin(\alpha)$$

Před zápisem vektoru do normálové mapy jsem vektor normalizoval a jednotlivé složky promítnul do rozsahu hodnot 0 až 255.

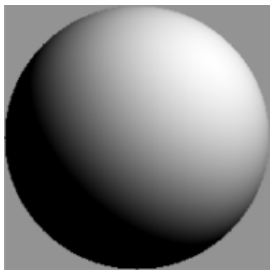
Kapitola 6
Testování algoritmu

Algoritmus jsem testoval na deseti různých vstupních fotografiích. Z každého testu je několik výstupů: výstup z rovnice E2, výstupní výšková mapa, spočítaná normálová mapa a povrch s použitou výškovou a normálovou mapou vykreslený ve 3D editoru Blender. První testovaný povrch je pro srovnání výstupů z mé implementace algoritmu a z implementace algoritmu autorů článku [1]. Pro další dva testované obrázky jsem nechal vykreslit postupně povrch se světelným zdrojem umístěným: vpravo nahoře, vpravo dole, vlevo dole a vlevo nahoře. Zbývající testované povrchy jsem již vykresloval s nasvícením jenom z jedné strany. Za výstupními obrázky následuje vždy diskuse nad výsledky.

Po testech výstupu jsem algoritmus otestoval ještě na rychlost výpočtu, viz kapitola 6.11.

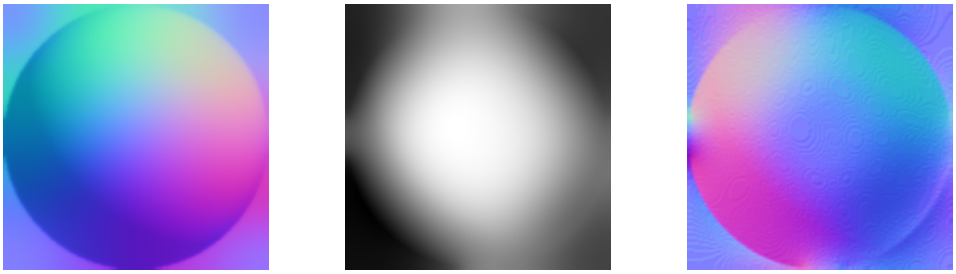
6.1 Koule

První povrch na testování byl obrázek koule, na tomto obrázku testovali algoritmus i v článku [1]. V článku, dle mého názoru počítali povrch koule pouze tam, kde koule skutečně byla, proto se u jejich výsledků vyskytuje šum na okraji koule, zatímco já jsem vypočítal povrch koule z celého vstupního obrázku a poté jsem v editoru pro vykreslení použil jenom kouli bez okrajů.



Obrázek 6.1. Koule: Vstupní obrázek

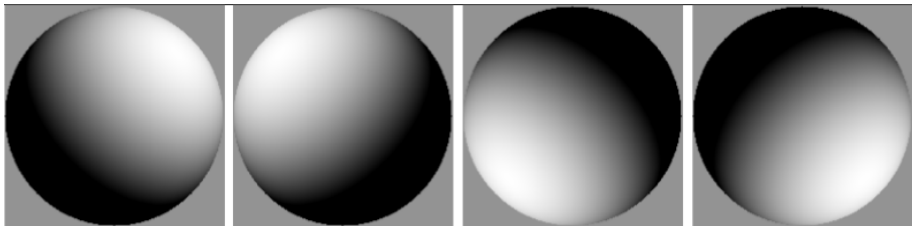
6.1.1 Výstupy



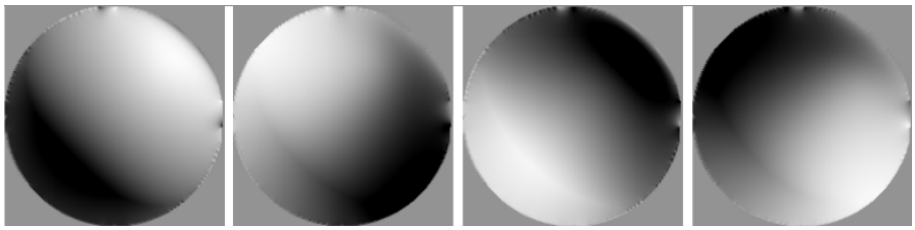
Obrázek 6.2. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa.

6.1.2 Porovnání výstupů s článkem [1]

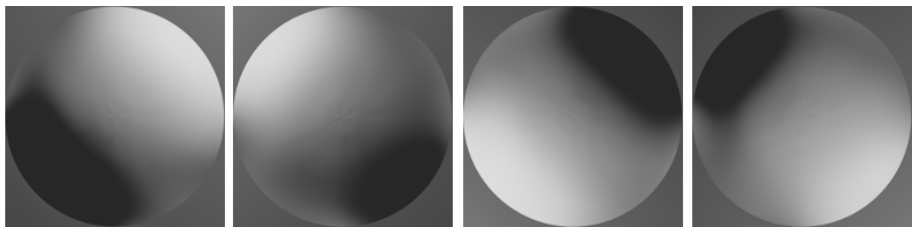
První uvádím, jak by nasvícení koule mělo skutečně dopadnout. Jedná se o obrázek převzatý ze článku [1]. Druhý obrázek ukazuje výsledky autorů článku a pod ním jsou mé výsledky.



Obrázek 6.3. Koule: Skutečný povrch



Obrázek 6.4. Koule: Výstup autorů článku [1]



Obrázek 6.5. Koule: Můj výstup

6.1.3 Diskuse nad výstupy

Koule byl jediný povrch, kde jsem použil interpolovanou masku ze vstupních vektorů. Z výškové mapy je vidět, že je koule trochu hranatá a nemá dokonale zaoblené hrany, ale po vykreslení povrchu se tento efekt neprojeví natolik, aby to kazilo dojem kulatého povrchu. Na výškové mapě je vidět, že pravý dolní roh je zcela černý, ale ostatní rohy jsou jen zašedlé, toto je způsobeno tím, že i přes použití interpolované masky algoritmus prosazuje tvar, který je určený původními rovnicemi.

Oproti výstupu z článku [1] je vidět na mém výstupu znatelné zlepšení, zejména na kouli nasvícené z levého horního rohu, u mé koule se neobjevily artefakty způsobené vstupním obrázkem, jako u výstupu autorů článku.

6.2 Mince

Obrázek jsem zvolil neboť není diffuzní a chtěl jsem zjistit, jak si algoritmus poradí s ne čistě diffuzním povrchem.

Obrázek 6.6. Mince: Vstupní obrázek.Tento obrázek byl převzat z adresy [1]).

6.2.1 Výstupy

Obrázek 6.7. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa.

Obrázek 6.8. Vykreslený povrch v různých směrech nasvícení

6.2.2 Diskuse nad výstupy

U složitých povrchů, jako je tento, je nemyslitelné, aby uživatel zadával směr normál pro většinu povrchu. Pro vykreslení tohoto (i všech dalších) povrchu jsem tudíž použil jen 3 vstupní vektory, pomocí nichž se spočítal vektor nasvětlení. Na minci je vidět, že má tendenci zachovávat správný tvar ve směru nasvětlení (pravý horní roh), zatímco v opačném směru je tato informace poškozena nebo zcela chybí. Okraje mince jsou zkroucené, toto může být způsobeno silnými odlesky nediffuzního povrchu nebo bílým pozadím.

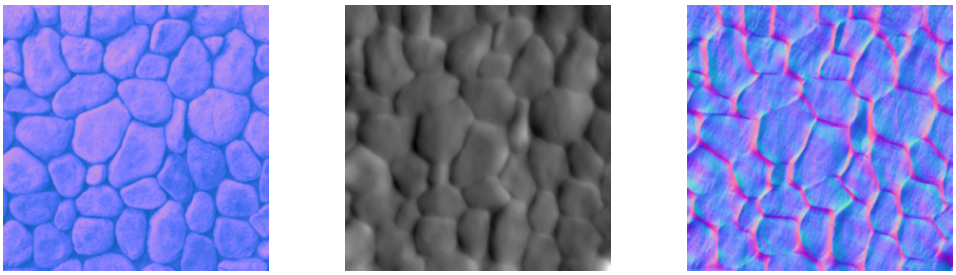
[1]) `http://d3au2jhdi7kksl.cloudfront.net/content/image/2/product/998870116`

6.3 Kamenný povrch

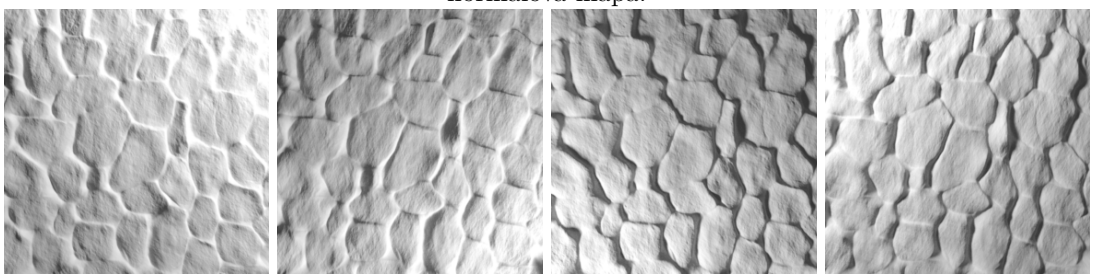
Tento testovací obrázek jsem zvolil kvůli nejednotě barev a velkému šumu (nejedná se o hladký povrch). Obrázek jsem ořízl, aby strany měly stejnou velikost.

Obrázek 6.9. Kamenný povrch: Vstupní obrázek.Tento obrázek byl převzat z adresy [1])

6.3.1 Výstupy



Obrázek 6.10. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa.



Obrázek 6.11. Vykreslený povrch v různých směrech nasvícení

6.3.2 Diskuse nad výstupy

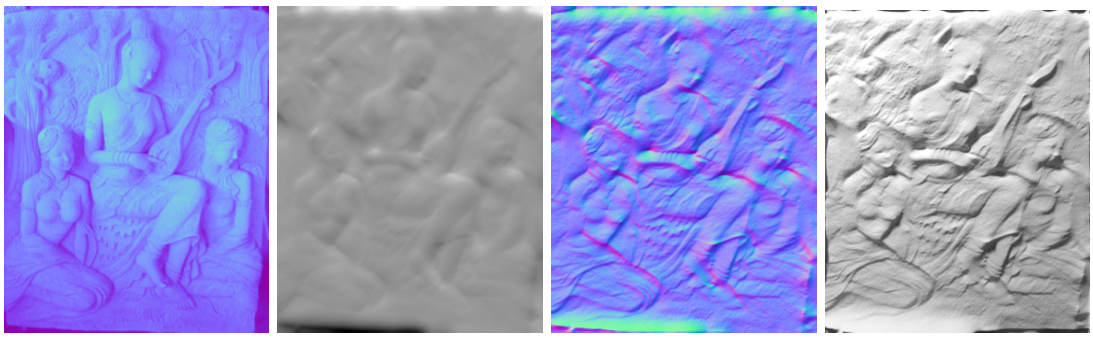
Z vykreslených povrchů je vidět, že kameny nejsou odděleny, nýbrž se z levého dolního rohu do pravého horního rohu postupně zvedají. I přes tento omyl, byl algoritmus schopen ustát nejednotnost barev v obrázku a věrně napodobit tvar jednotlivých kamenů.

[1]) `http://weknowyourdreams.com/image.php?pic=/images/stone/stone-06.jpg`

6.4 Fotografie soch

Obrázek 6.12. Fotografie soch: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.4.1 Výstupy



Obrázek 6.13. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

6.4.2 Diskuse nad výstupy

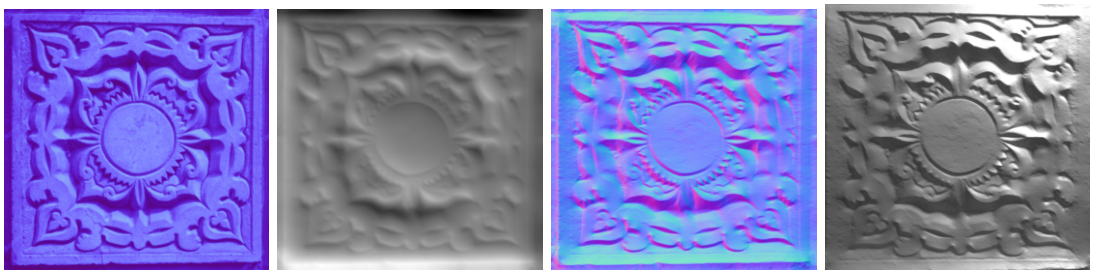
Algoritmus zcela potlačil jakékoliv detaily, zejména na okrajích obrázku (jako například strom v levém horním rohu). Fotografie měla kromě vyfoceného reliéfu také na okrajích kousek pozadí (zejména pravý dolní roh), předpokládal jsem, že tyto okraje zcela zkazí výsledek, ale výsledek byl překvapivě dobrý i přes nesourodost povrchů.

[1]) `http://image.ec21.com/image/poraween/oimg_GC04347445_CA04347449/Lady_Stone_Relief_Wall_Hanging.jpg`

6.5 Fotografie kamenného reliéfu

Obrázek 6.14. Fotografie kamenného reliéfu: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.5.1 Výstupy



Obrázek 6.15. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

6.5.2 Diskuse nad výstupy

Tento povrch jsem testoval zejména z domněnky, že algoritmus nebude umět zachovat jasné a ostré rysy. Na výsledném vykresleném povrchu je vidět, že jsem se částečně mýlil. Výsledek ale není dokonalý, v originále jsou rýhy a výstupky, které algoritmus zcela potlačil.

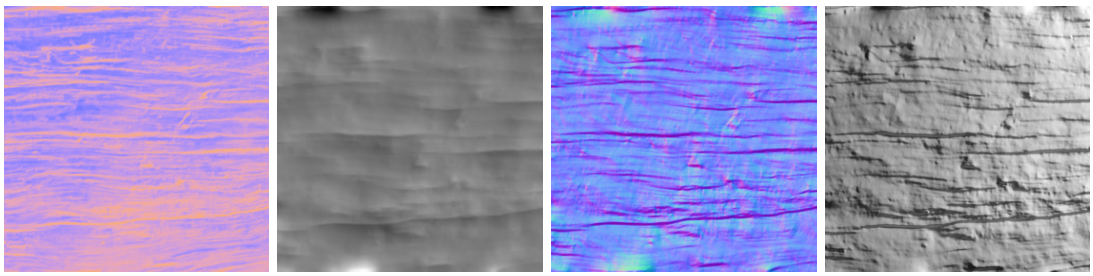
[1]) `http://static6.depositphotos.com/1039538/576/i/950/depositphotos_5763635-stock-photo-old-abstract-bas-relief.jpg`

6.6 Fotografie dřeva

Fotografii jsem ořízl, aby měla čtvercový tvar.

Obrázek 6.16. Fotografie dřeva: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.6.1 Výstupy



Obrázek 6.17. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

6.6.2 Diskuse nad výstupy

S výsledkem tohoto testu jsem velice spokojen, neboť jsem neočekával, že na takto nejasném a zašuměném povrchu dokáže algoritmus produkovat takové výsledky. Algoritmus dokázal do výškové mapy zanést nejen hrubé kontury dřeva, ale i jemné nerovnosti na povrchu.

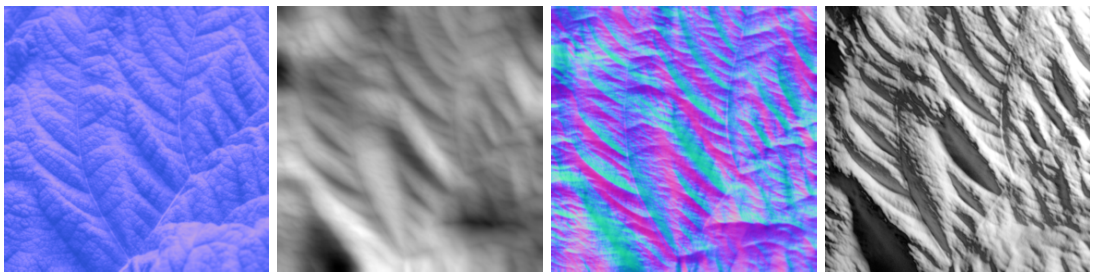
[1]) `http://etc.usf.edu/clippix/pix/a-flying-insect-on-a-wood-surface_medium.jpg`

6.7 Fotografie listu

Obrázek jsem ořízl do čtverce.

Obrázek 6.18. Fotografie listu: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.7.1 Výstupy



Obrázek 6.19. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

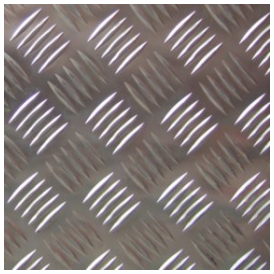
6.7.2 Diskuse nad výstupy

Tento vstupní obrázek nebyl příliž vhodný, neboť je vidět, že obrázek obsahuje části listu, které jsou mnohem blíže ke kameře než jiné (pravý spodní roh). Tuto část algoritmus neodhadl správně, ale to jsem očekával. Zajímavé je, že zbytek povrchu byl vypočítán prakticky bez chyby, a to i když byl list ve vstupním obrázku zkroucený a nebyl zcela rovně před kamerou. S výsledky tohoto testu jsem spokojen.

[1]) `http://www.publicdomainpictures.net/pictures/170000/velka/leaf-texture-1463652587zED.jpg`

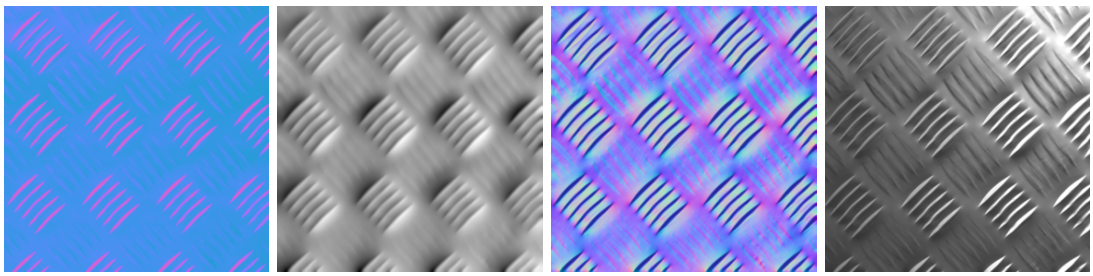
6.8 Kovový plát

Obrázek jsem opět ořízl do tvaru čtverce.



Obrázek 6.20. Kovový plát: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.8.1 Výstupy



Obrázek 6.21. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

6.8.2 Diskuse nad výstupy

Z výstupu z rovnice E2 je vidět, že algoritmus zachytil rýhy jen v jednom směru a kolmé rýhy zcela potlačil. Tento fakt se projeví i na výsledném vykresleném povrchu. Nutno si ale uvědomit, že algoritmus je určený na diffuzní povrchy a tento povrch je lesklý kov, proto jsem s výsledky spokojen.

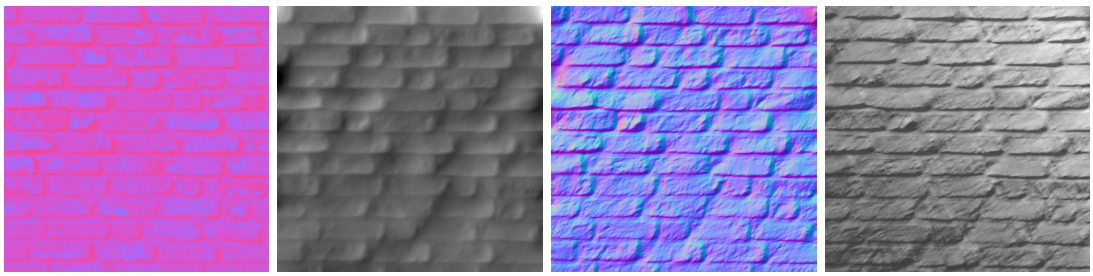
[1]) `http://www.imageafter.com/dbase/textures/metals/b21tabus1159.jpg`

6.9 Cihlová zeď

Obrázek jsem ořízl do čtvercového tvaru.

Obrázek 6.22. Cihlová zeď: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.9.1 Výstupy



Obrázek 6.23. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

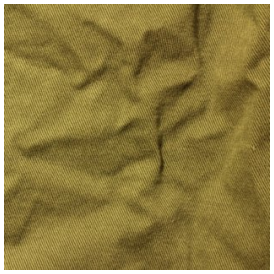
6.9.2 Diskuse nad výstupy

Výšková mapa nesprávně popisuje vertikální spáry mezi cihlami, ale po vykreslení samotného povrchu je tato chyba zanedbatelná a výsledek je dle mého názoru přijatelný.

[1]) `http://img.huffingtonpost.com/asset/2000_1000/573af82a1600002a00f937b7.jpeg`

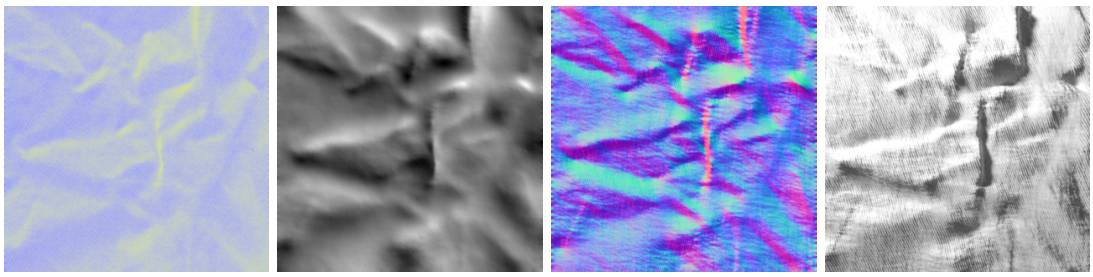
6.10 Látka

Fotografii jsem ořízl do tvaru čtverce.



Obrázek 6.24. Látka: Vstupní obrázek. Tento obrázek byl převzat z adresy [1]).

6.10.1 Výstupy



Obrázek 6.25. Zleva doprava: výstup z upravené rovnice E2, vypočítaná výšková mapa, normálová mapa, vykreslený povrch.

6.10.2 Diskuse nad výstupy

Látka měla velice matoucí povrch i vzhledem k jejímu hrubému povrchu. Po vykreslení si lze povšimnout, že vykreslený povrch se značně liší od povrchu originálního, nicméně algoritmus byl schopen popsat jak hlavní tvarové rysy látky (jak je látka zmačkaná), tak i jednotlivé rýhy v látce.

[1]) `http://www.mayang.com/textures/Fabric/images/Fine`

6.11 Testování rychlosti výpočtu

Algoritmus jsem otestoval vůči třem faktorům. Prvním je závislost doby výpočtu na vstupních datech. Druhým je závislost doby výpočtu na rozlišení vstupního obrázku a třetím je závislost doby výpočtu na počtu výpočetních kroků.

6.11.1 Závislost na vstupních datech

V této části jsem testoval algoritmus na všech obrázcích z kapitoly 6. Všechny obrázky jsem převedl na rozlišení 180×180 pixelů. Obrázek 6.12 jsem oříz do čtverce. Každý obrázek jsem nechal algoritmus spočítat se stejnými parametry desetkrát. Cyklus Gauss-Seidelovy metody pro rovnici E2 (viz kapitola 3.2.2) jsem nastavil na 25 průběhů a cyklus Gauss-Seidelovy metody pro rovnici E3 (viz kapitola 3.2.3) jsem nastavil na 1000 průběhů. Tabulka průměrů naměřených hodnot je v tabulce 6.1.

testovaný obrázek	průměrný čas[sekundy]
obrázek 6.1	0.5474
obrázek 6.6	0.5365
obrázek 6.9	0.5335
obrázek 6.12	0.5557
obrázek 6.14	0.4928
obrázek 6.16	0.5100
obrázek 6.18	0.5243
obrázek 6.20	0.5176
obrázek 6.22	0.5138
obrázek 6.24	0.5113

Tabulka 6.1. Testování závislosti na vstupních datech.

Testování dopadlo podle mého očekávání, data nejsou téměř závislá na vstupních datech. Jediná část algoritmu, která je ovlivněna vstupními daty, je při výpočtu relativních výšek, ovlivňuje ji konstanta FLAT_ANGLE, viz 5.3.2. Tato konstanta ale nemá tak velký vliv, aby ve výsledných časech byl výraznější rozdíl.

6.11.2 Závislost na rozlišení

V této části jsem testoval závislost na rozlišení. Pro testování jsem použil obrázek 6.20. Obrázek jsem si uložil v několika rozlišeních, které jsem poté testoval opět desetkrát. Nastavení algoritmu bylo stejné jako v kapitole 6.11.1. Cyklus Gauss-Seidelovy metody pro rovnici E2 (viz kapitola 3.2.2) jsem nastavil na 25 průběhů a cyklus Gauss-Seidelovy metody pro rovnici E3 (viz kapitola 3.2.3) jsem nastavil na 1000 průběhů. Tabulka průměrů naměřených hodnot je v tabulce 6.2.

rozlišení obrázku	průměrný čas[sekundy]
1358×1358	31.6452
1019×1019	19.7132
680×680	8.4738
340×340	2.1253
136×136	0.3557
68×68	0.0847

Tabulka 6.2. Testování závislosti na rozlišení.

Dle výsledků z tabulky 6.2 je vidět, že se vzrůstajícím rozlišením trvaly výpočty stále delší dobu. Algoritmus je závislý na rozlišení vstupního obrázku, ale tento výsledek se dal očekávat, neboť čím více pixelů, tím více proměnných ve všech použitých rovnicích algoritmu.

6.11.3 Závislost na počtu výpočetních kroků

V této části jsem testoval závislost na počtu výpočetních kroků. Testoval jsem na obrázku 6.6. Cyklus Gauss-Seidelovy metody pro rovnici E2 (viz kapitola 3.2.2) a pro rovnici E3 (viz kapitola 3.2.3) jsem nastavoval oba dva na stejný počet kroků v každém testu. Výsledky testu jsou uvedené v tabulce 6.3.

počet kroků	průměrný čas[sekundy]
200	14.4221
175	13.1764
150	13.7972
125	12.8660
100	12.2683
75	12.2327
50	11.6350
25	11.5071
1	10.7081

Tabulka 6.3. Testování závislosti na počtu výpočetních kroků.

Z výsledků v zaznamenaných v tabulce 6.3 je vidět, že se snižujícím se počtem kroků, klesá i čas potřebný k výpočtu. Tyto výsledky byly více než předvídatelné, neboť při více krocích se obrázek musí vícekrát přepočítávat. Je vidět, že cca 10 sekund trvá inicializace algoritmu, zatímco samotný průběh výpočtu není již tak časově náročný. Například rozdíl délky výpočtu mezi 200 kroky a 100 kroky jsou pouhé 2 sekundy.

Při testování algoritmu jsem si všiml že druhá část, rovnice E2 (viz kapitola 3.2.2), dokonverguje do minima velice rychle, cca po 10 cyklech se její výstupy již prakticky neměnily. Postupná konvergence je ukázána na obrázku 6.26. Mohl bych tedy nastavit 10 cyklů staticky pro každý výpočet, do výsledného programu jsem ale nastavil 25 cyklů, protože zde jsou ještě proměnné λ, δ a ρ (viz 3.2.2), které nastavuje uživatel. Pro 10 cyklů nebyla znát změna výstupů mezi nastavením proměnné $\lambda = 10$ a $\lambda = 100$, při 25ti krocích se ale již výstup skutečně začal vyhlazovat. Počet kroků pro rovnici E2 jsem proto nastavil na 25, takovouto hodnotu jsem ponechal i ve finálním programu.

Obrázek 6.26. Různé výstupy rovnice E2 pro různé počty kroků. Zleva doprava: 2 kroky, 10 kroků, 20 kroků, 100 kroků.

Zatímco u třetí části, rovnice E3 (viz kapitola 3.2.3), rozdílný počet kroků dramaticky mění výstupy algoritmu. Pro ukázku jsem algoritmus použil s různými počty kroků pro rovnici E3 na obrázek 6.6. Výsledky testování jsou zachyceny v tabulce 6.4.

počet kroků	výšková mapa	normálová mapa
10		
100		
1000		
5000		

Tabulka 6.4. Testování závislosti na počtu výpočetních kroků.

Z výsledků je vidět, že čím více kroků pro výpočet použijeme, tím plastičtější výsledek dostaneme. U výstupu pro 10 kroků je výšková mapa složena pouze z tvrdých přechodů a neobsahuje žádné informace, například o zvlnění tváře na minci, naopak u výstupů pro 5000 kroků už se nám začíná vlnit i celá mince. Parametr počtu kroků pro rovnici E3 jsem nechal nastavovatelný uživatelem, a to v rozsahu 100 až 10 000 kroků.

6.12 Testování vlivu konstant λ , δ a ρ na výstup

Algoritmus má celkem tři nastavitelné konstanty: λ , δ a ρ (viz kapitola 5.2).

6.12.1 Testování konstanty λ

První se zaměřím na konstantu λ. Konstanta λ nám určuje vliv regularizační funkce v rovnici E2 na výsledný vektor, jak jsem již popsal v kapitole 3.2.2. Tudíž čím vyšší konstanta λ, tím plynulejší by vzniklý výstup z rovnice E2 měl být. Tuto teorii jsem opět otestoval na obrázku 6.6, pro výpočet jsem u každého testu musel upravit počet kroků výpočtu, aby změny byly zřetelné. Výsledky testování pro různé hodnoty λ jsou v tabulce 6.5.

λ	výstup z E2	výšková mapa	normálová mapa
$\frac{1}{100}$			
$\frac{1}{2}$			
1			

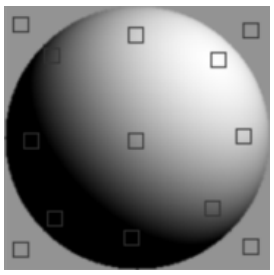
Tabulka 6.5. Testování vlivu konstanty λ na výstup.

Na výstupu z rovnice E2 není vidět rozdíl, ale na výškové mapě pro hodnotu $\lambda = 1$ je vidět, že krk postavy má (nejspíše díky nediffusním odleskům) nepřirozeně zvrásněný povrch, zejména tyto nerovnosti se při zvýšení parametru λ zahladily, bohužel je též vidět na výškové mapě pro hodnotu $\lambda = \frac{1}{100}$, že nám začínají mizet detaily z vlasů postavy.

Konstantu λ jsem nechal z uživatelského prostředí nastavitelnou v rozsahu $\frac{1}{100}$ až 1.

6.12.2 Testování konstanty δ

Konstanta δ, jak je popsáno v kapitole 5.2 určuje, jak moc budou mít vstupní vektory přímý vliv na výstup z rovnice E2. Ze vstupních vektorů se spočte interpolací maska (obrázek 5.3), která se poté použije jako podklad, kterému se budou připodobňovat výstupní vektory z rovnice E2. Pro testování této konstanty jsem si zvolil obrázek koule 6.1, na kterém bylo dobře vidět, jaký má konstanta vliv ja výstupní vektory. Na obrázku 6.27 je čtverci na vstupním obrázku znázorněno, kde jsou umístěny vstupní vektory, každý vektor jsem nastavil tak, aby odpovídal tvaru koule v daném místě. Výstupy z rovnice E2 pro různé hodnoty δ jsou pak na obrázku 6.28.



Obrázek 6.27. Rozmístění vstupních vektorů, testování konstanty δ.

Obrázek 6.28. Testování konstanty δ. Zleva doprava: $\delta = 0$, $\delta = \frac{1}{4}$, $\delta = \frac{1}{2}$

Konstantu δ jsem nechal z uživatelského prostředí nastavitelnou v rozsahu 0 až $\frac{1}{2}$.

6.12.3 Testování konstanty ρ

Konstanta ρ má reflektovat, jakou odrazivost má materiál, který bude algoritmem počítán, více je popsáno v kapitole 3.2.2. Konstantu jsem testoval opět na obrázku 6.1. Výstupy z rovnice E2 pro různé nastavení konstanty ρ jsou na obrázku 6.29.

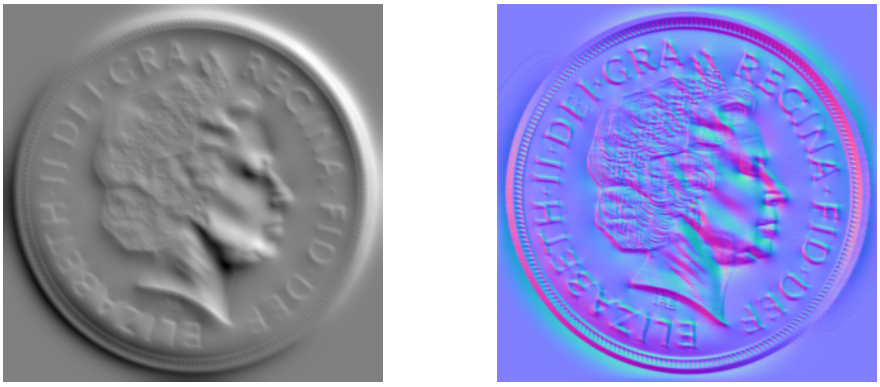
Obrázek 6.29. Testování konstanty ρ. Zleva doprava: $\rho = \frac{1}{10}$, $\rho = \frac{1}{2}$, $\rho = 1$

Z výsledků testování je vidět, že se sníženou hodnotou konstanty ρ se algoritmus začne chovat méně citlivě a již nerozlišuje takové spektrum vektorů. Do výsledku se poté dostanou jenom vektory, které měly největší zastoupení. Hodnota konstanty ρ menší než 1.0 tudíž, dle mého názoru, znehodnocovala výsledky. V algoritmu lze konstantu na vstupu nastavit, nicméně pro nedostatek místa na obrazovce při tvorbě uživatelského prostředí (více k uživatelskému prostředí v kapitole 7) a pro špatný vliv konstanty na výstupy algoritmu, jsem tuto konstantu neumožnil uživatelům nastavit z uživatelského prostředí. Hodnotu konstanty jsem nastavil na 1.0.

6.13 Porovnání výstupů algoritmu s konkurenčními programy

Výstupy algoritmu jsem porovnával s konkurenčními programy, které se zabívají stejnou problematikou. Programů, které dokáží zrekonstruovat povrch z jediného obrázku je opravdu pomálu, po dlouhém hledání jsem našel dva programy, které toto dokáží. Pro porovnání jsem použil obrázek mince 6.6, na kterém jsem testoval algoritmus v kapitole 6.2. Všechny programy, které jsem vyzkoušel generovaly jak výškové tak i normálové mapy, proto jsem obě tyto mapy použil pro porovnání. Nejprve uvádím výstupy z mé implementace algoritmu, následně jsou uvedeny výstupy z jednotlivých programů.

6.13.1 Výstupy z mé implementace algoritmu



Obrázek 6.30. Výstupy z mé implementace algoritmu, zleva doprava: výšková mapa, normálová mapa

6.13.2 Výstupy z programu InsaneBump

InsaneBump je volně stažitelné rozšíření do programu Gimp. Plně automatický algoritmus. Dostupný na adrese [1])

Obrázek 6.31. Výstupy z InsaneBump, zleva doprava: výšková mapa, normálová mapa

[1]) `http://registry.gimp.org/node/28117`

6.13.3 Výstupy z programu CrazyBump

CrazyBump je placený program, také plně automatický. Dostupný na adrese [1])

Obrázek 6.32. Výstupy z CrazyBump, zleva doprava: výšková mapa, normálová mapa

6.13.4 Diskuze nad výstupy

Výstupy z mé implementace algoritmu dopadly překvapivě dobře. Například výstupy z InsaneBump jsou o porovnání horší. Kvalita mých výškové mapy je na stejné úrovni jako u CrazyBump, normálová mapa je u CrazyBump dle mého názoru lepší.

[1]) `http://www.crazybump.com/`

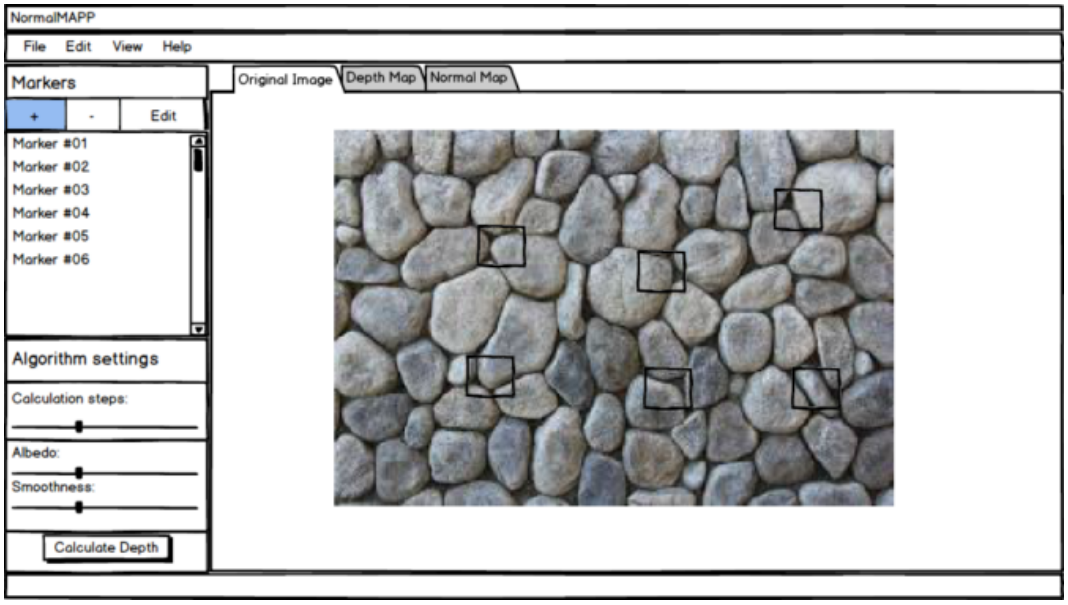
Kapitola 7
Uživatelské rozhraní aplikace

Aplikaci jsem nazval „NormalMAPP“. Uživatelské rozhraní pro aplikaci jsem navrhoval v programu Balsamiq Mockups 3 (více zde [1])). Samotnou aplikaci jsem implementoval v jazyce Java a uživatelské prostředí za pomoci balíku Swing.

7.1 Nárvh uživatelského rozhraní

7.1.1 Obrazovka se vstupním obrázkem

Tato obrazovka (obrázek 7.1) se uživateli zobrazí při spuštení programu. Uprostřed obrazovky se nachází obrázek, který si uživatel načetl z disku. Na tomto obrázku bude uživatel označovat místa pomocí čtverců. V těchto místech následně nastaví informace o normálovém vektoru (zobrazeno na obrázku 7.2). Vlevo je hlavní panel programu, v tomto panelu budou situovány hlavní ovládací prvky programu. V horní části panelu budou tlačítka na přidávání, úpravu a smazání čtverců. Pod tlačítky bude seznam přidaných čtverců. Pod seznamem následuje část pro nastavení konstant algoritmu a tlačítko na spuštění výpočtu.



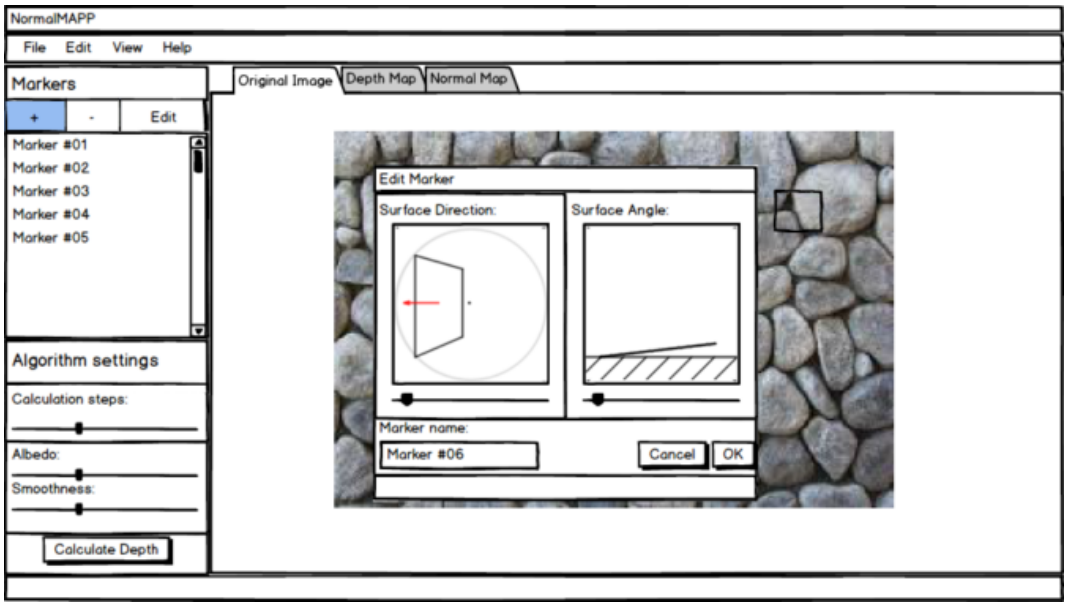
Obrázek 7.1. Obrazovka se vstupním obrázkem

[1]) https://balsamiq.com/products/mockups/

7.1.2 Zadávání vektoru normály

Tato obrazovka (obrázek 7.2) se uživateli zobrazí, pokud přidá čtverec do obrázku. Obrazovka se skládá ze dvou hlavních částí, v levé části uživatel posuvníkem nastavuje rotaci normálového vektoru ve směru kolmém k obrázku (ve směru osy z). V pravé části pak uživatel nastavuje úhel vektoru od osy z a obrázkem.

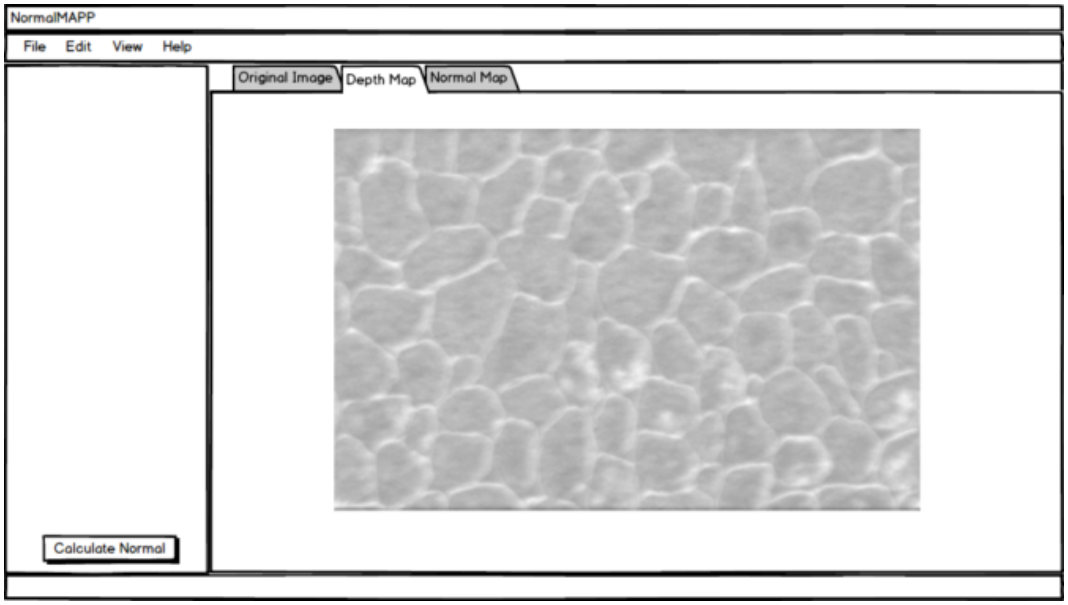
Uživatel si může vybrat i název, jaký chce značce dát.



Obrázek 7.2. Zadávání vektoru normály

7.1.3 Obrazovka s výškovou mapou

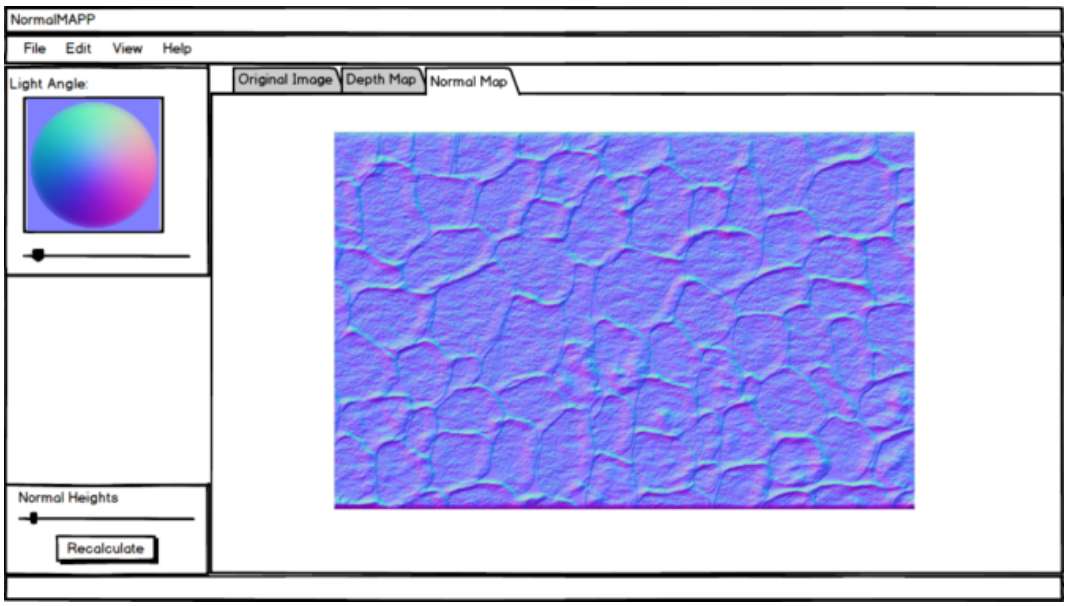
V této obrazovce (obrázek 7.3) se zobrazí vypočítaná výšková mapa, v pravé části je pak tlačítko na spočítání normálové mapy.



Obrázek 7.3. Obrazovka s výškovou mapou

7.1.4 Obrazovka s normálovou mapou

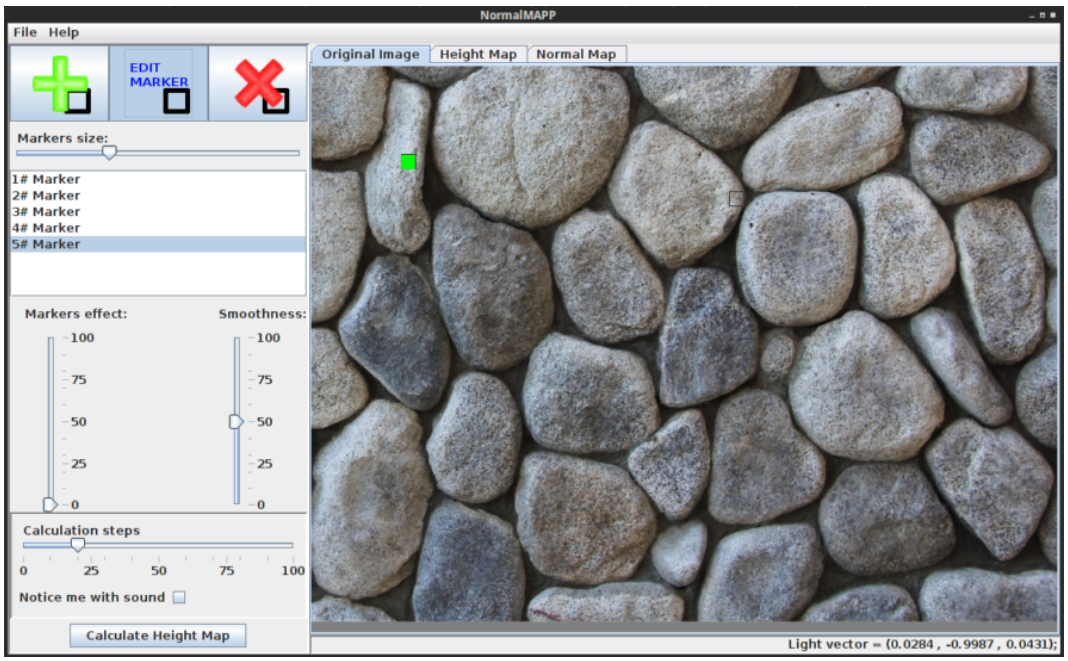
Na této obrazovce (obrázek 7.4) se uživateli zobrazí spočtená normálová mapa. V pravém panelu má uživatel možnost rotovat normálové vektory kolem jejich osy z a nastavit si výšku normálové mapy. U nastavování rotace normálových vektorů je uživateli postkytnut náhled koule, jak budou vektory na přepočítané mapě vypadat. Všechny provedené změny pak může aplikovat na normálovou mapu stiskem tlačítka `Recalculate`.



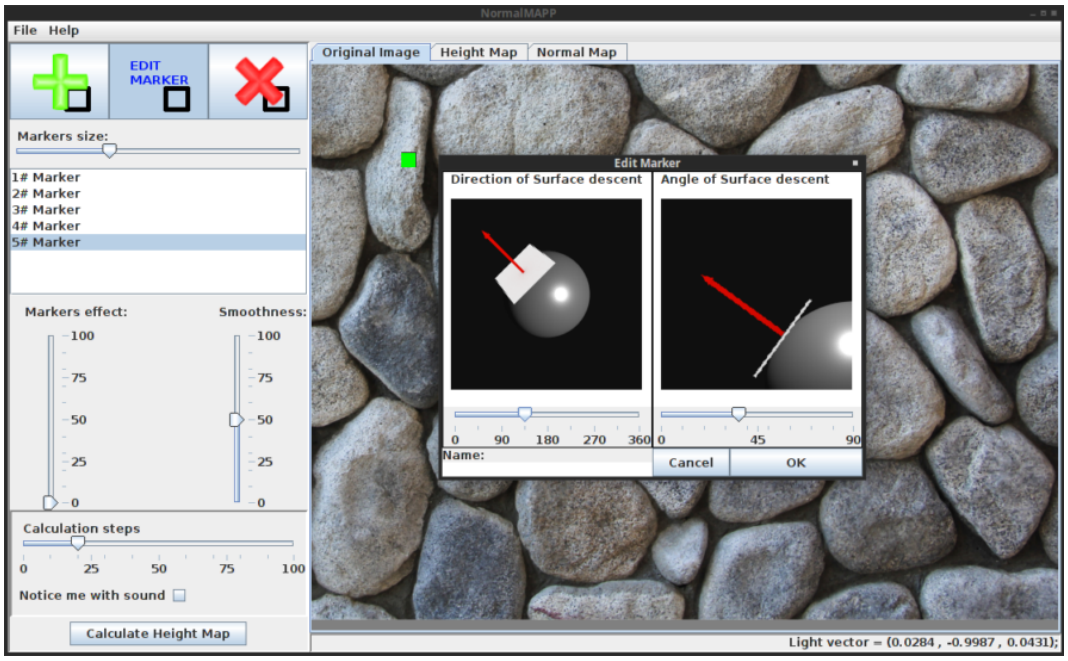
Obrázek 7.4. Obrazovka s normálovou mapou

7.2 Výsledné uživatelské rozhraní

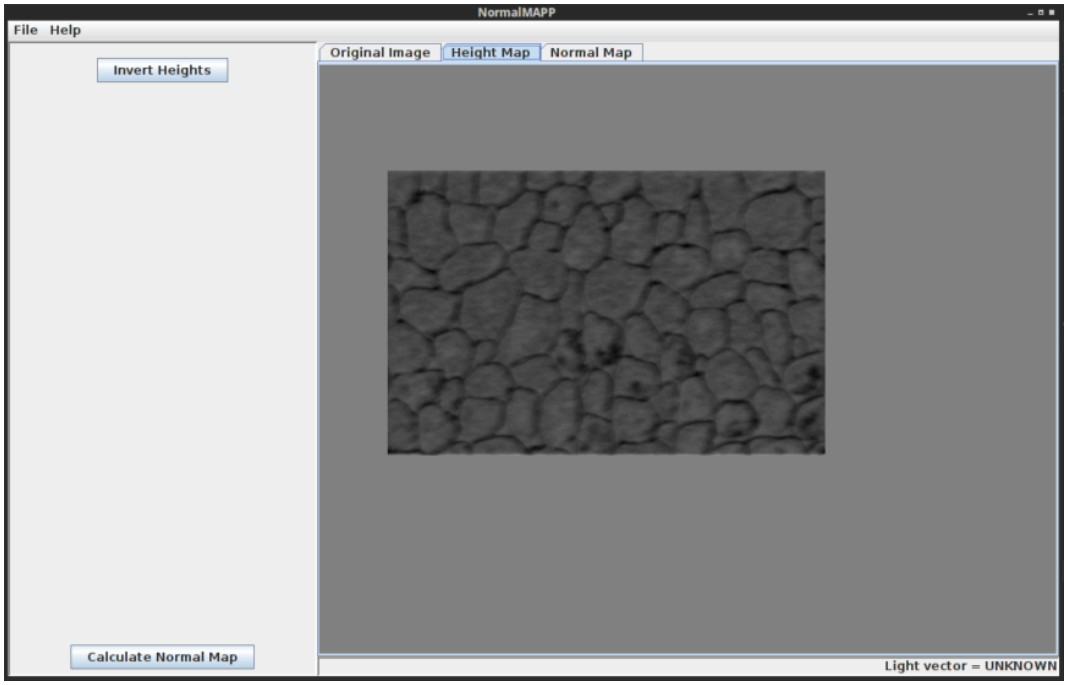
V obrazovce se vstupním obrázkem jsem od návhu do konečné podoby aplikace změnil vstupní konstanty programu, důvod k tomuto rozhodnutí je popsán v kapitole 6.12.3. Posuvník *Smoothness* nastavuje hodnotu konstanty λ a posuvník *Markers effect* nastavuje hodnotu konstanty δ, více o konstantách v kapitole 6.12. Posuvník *Calculation steps* nastavuje počet výpočetních kroků algoritmu, více popsáno v kapitole 6.11.3. Funkce zaškrtávacího tlačítka *Notice me with sound*, stejně jako oznamovací lišta, jsou popsány v kapitole 8.



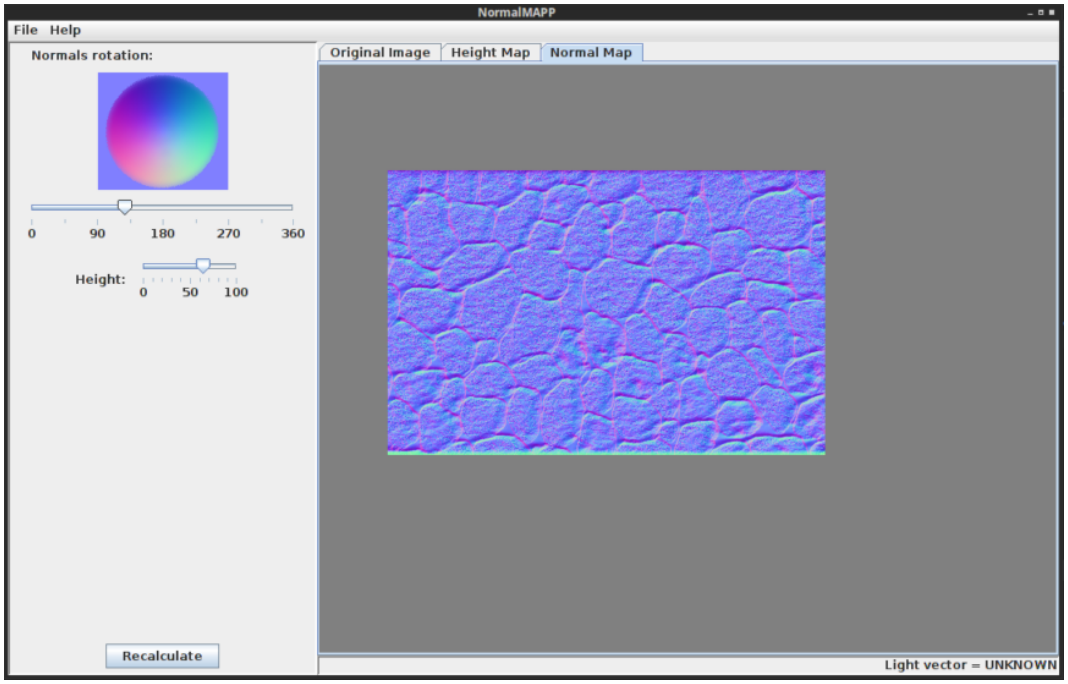
Obrázek 7.5. Obrazovka se vstupním obrázkem



Obrázek 7.6. Zadávání vektoru normály



Obrázek 7.7. Obrazovka s výškovou mapou



Obrázek 7.8. Obrazovka s normálovou mapou

Kapitola 8
Ostatní funkce programu

8.1 Návod na použití

Do programu jsem zabudoval jednoduchý tutoriál, abych si byl jist, že uživatel bude umět program používat. Zde jsou záběry jednotlivých stránek. Tutoriál je v programu dostupný pod záložkou Help → Tutorial.

Welcome to the NormalMAPP

In this little tutorial we will teach you how to generate your own normal maps

You can go to next page by clicking **Next**.

You can always return to previous page by clicking **Previous**.

For ending the tutorial just hit the **End** button.

Obrázek 8.1. Tutoriál, 1. stránka

Opening Texture

You can either open an **texture / photograph** and let NormalMAPP generate you heigt map and normalmap or you can load an **heightmap** and generate normal map from it.

Open Texture / Photo

Press *File --> Open Texture*

Your loaded texture will then show up in the Original Image tab.

Open Height Map

Press *File --> Load --> Load Height Map*

Height map will then show up in the Height Map tab.

Obrázek 8.2. Tutoriál, 2. stránka

Add Markers

Before NormalMAPP can generate height map you have to provide at least **3** markers.

Marker will store the information of surface angle at the point where it is.

First click at the button for **adding marker**:

Now you selected the tool for adding markers, just simply **click into the picture**, where you want to place the marker.

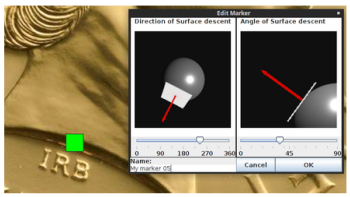
Now we will learn how to set the marker correctly.

Obrázek 8.3. Tutoriál, 3. stránka

Setting Up Marker

Marker should be representing surface shape at its placement. For this purpose you will have to set up surface descent and angle of such descent in your marker.

Example of correct setup:



You can name markers to your liking.

Calculated results will depend lot on the marker's setup so take your time and experiment for the best results.

Obrázek 8.4. Tutoriál, 4. stránka

Edit / Remove Markers

After placing marker on the picture you can change it's values or completely remove it

Edit Marker

For editing marker select tool **Edit Marker**:

Then just simply click on the marker on the screen which you want to edit.

Remove Marker

For removing marker select tool **Remove Marker**:

Then click on the marker you want to remove.

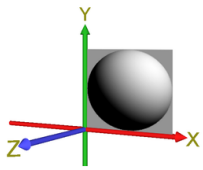
*You can **highlight marker** by selecting it in the list of markers:*

Are those markers too big? Adjust size of markers yourself by using **Markers size** slider.

Light Vector

Light vector will be calculated after 3 markers are in place, once the light vector is caltulated you can compute the height map.

Axes of light vector are following:



Light vector determines in which direction is the light shinning to on the surface. If the last figure is negative, you know you should rearrange the position of your markers.

Obrázek 8.5. Tutoriál, 5. stránka

Calculations setup

Markers effect

Markers effect determines how much your markers will effect the surface directly.

Once you've placed three markers the Light vector is calculated.

But if you want to use your markers as direct surface correction than set "Markers effect" on value higher than zero.

Use this parameter for surface correction.

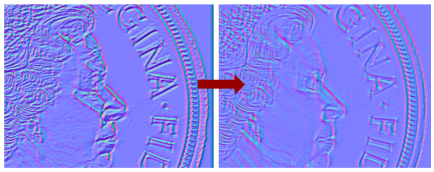
Obrázek 8.6. Tutoriál, 6. stránka

Calculations setup

Smoothness

Smoothness parameter will define how curvy the final surface should be.

Use this parameter for getting rid of height **artifacts** as is shown on the picture:



With high smoothness there will be no artifacts but less details, use it wisely.

Obrázek 8.7. Tutoriál, 7. stránka

Calculations setup

Calculations steps

This parameter determines how long will the calculations last.

For most pictures 20 - 30 steps are enough.

Beware of too many calculation steps. More steps does not equal better results!

If we assume that calculations will last for too long (more than 30 seconds), we will play sound of gong after everything is calculated.

You can press *Calculate Height Map* to start the computation.

Obrázek 8.8. Tutoriál, 8. stránka

Height map

Select **Height map** tab to show calculated height map.

Invert height map by clicking *Invert heights*

You can press *Calculate Normal Map* to calculate normal map.

Obrázek 8.9. Tutoriál, 9. stránka

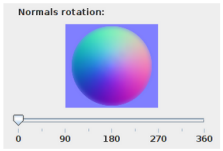
Normal map

Select **Normal map** tab to show calculated normal map.

In the last section you can set up normal map to your liking by rotating normals or by changing height of the normal map.

Rotating normals

In the side panel is section called **Normals rotation**.



Use the slider below to adjust direction of normals in your map.

Height of map

You can change the height of the map by adjusting the slider **Height**:

Press **Recalculate** button to apply your changes to the map!

Obrázek 8.10. Tutoriál, 10. stránka

Saving output

You can save height map and normal map by following:

Saving height map

Select *File --> Save --> Save Height Map*

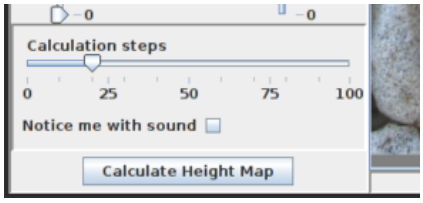
Saving normal map

Select *File --> Save --> Save Normal Map*

Obrázek 8.11. Tutoriál, 11. stránka

8.2 Zvukový signál

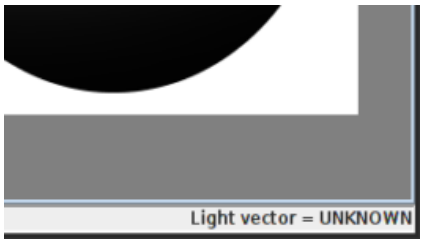
Uživatel si může v programu nastavit, zdali chce upozornit zvukovým tónem, že výpočty již skončily. Uživatel tak může učinit zaškrtávacím tlačítkem „Notice me with sound“, viz obrázek 8.12. Zvuková ukázka pochází z adresy [1]).



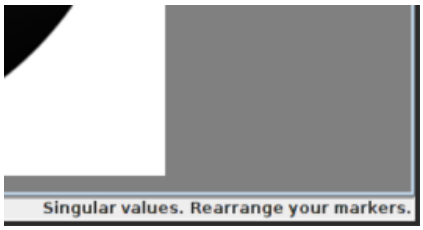
Obrázek 8.12. `Notice me with sound` zaškrtávací tlačítko

8.3 Výpočet vektoru nasvícení v průběhu zadávání vektorů

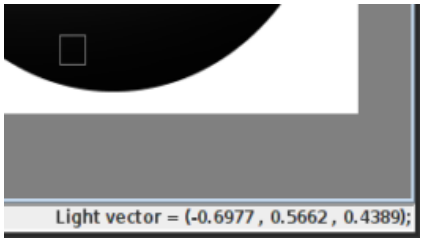
Do spodní části obrazovky jsem umístil oznamovací lištu, ve které se vypisuje vypočítaná pozice vektoru nasvětlení. Díky tomuto kroku si uživatel může zkontrolovat, zdali vektor není opačný, než by měl být. Algoritmus by v takovém případě poté spočítal výškovou mapu inverzní k té co požadoval uživatel. Vektor se počítá z prvních tří zadaných vektorů, ostatní vektory nemají na vektor světla vliv. Obrazovka je zachycena na obrázcích 8.13,8.14 a 8.15.



Obrázek 8.13. Vektor světla nebyl spočítán.



Obrázek 8.14. Špatně zadané informace.



Obrázek 8.15. Vektor světla byl spočítán a vypsán na oznamovací lištu.

[1]) `http://cmp.felk.cvut.cz/cmp/courses/OPT/cviceni/07/gong.wav`

Kapitola 9
Uživatelské testování aplikace

Uživatelské testování proběhlo na laptopu s finální aplikací. Při výběru participanta jsem se řídil následujícími kritérii:

- *základní znalost angličtiny*
- *základní znalost 3D grafiky a funkce normálových a výškových map*

Pro participanta jsem si připravil scénář:

„V 3D editoru tvoříte scénu, do které chcete zakomponovat následující motiv:

Obrázek 9.1. Testovací obrázek. Převzato z adresy [1]).

Obrázek jste ale nalezl na internetu a nemáte k němu žádné jiné podklady. Aby objekt ve scéně dobře vypadal, potřebujete použít normálovou mapu, tu ale nemáte. Rozhodnete se normálovou mapu tohoto obrázku získat pomocí programu NormalMAPP.

Vstupní obrázek je uložen na ploše pod názvem INPUT_IMG.jpg. Potřebujete uložit normálovou mapu na plochu pod názvem ‚normalmap.png'.“

Participanta jsem před začátkem testování upozornil na přítomnost tutoriálu přímo v programu. Dále jsem mu vysvětlil důvody testování a také, že testuji software a ne jeho osobu, proto pokud se něco pokazí, tak je to chyba softwaru a ne jeho. Dotázal jsem se participanta, zda souhlasí se zveřejněním záběrů z testování. Participant nesouhlasil, proto průběh testování popíši slovně.

9.1 Průběh testování

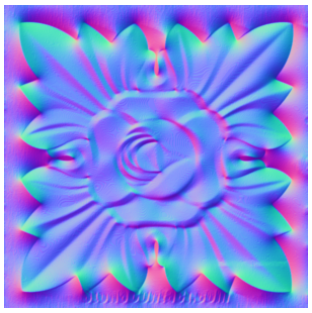
Participantovi jsem předložil scénář. Participant v programu bez problému nalezl tutoriál a začal postupovat dle návodu. Načtení vstupního obrázku nepůsobilo žádné problémy.

Participant bez problémů přidal první značku, měl problém zjistit, jak nastavit správně informace o vektoru ve značce. Zde se zdržel asi 20 sekund, překlikával z tutoriálu do aplikace. Po přidání první značky již bez problému přidal další dva. Participant při plnění úkolu přeskočil možnosti úpravy a smazání značek. Přeskočil i nastavení algoritmu a spustil výpočet výškové mapy.

[1]) `http://pic.stonecontact.com/picture/20159/40656/3d-feature-stone-wall-interior-relief-carving-tile-white-limestone-relief-carving-p378772-1b.jpg`

Po skončení výpočtu uživatel očekával, že mu program automaticky zobrazí výsledek výpočtu. Pokračoval v tutoriálu, překlikl na vypočítanou výškovou mapu a vyzkoušel si její otočení, poté stiskl na výpočet normálové mapy. V nastaveních normálové mapy se participant vyznal bez problémů. Po úpravě normálové mapy ji participant bez problémů uložil na určené místo.

Participantem vygenerovaný výsledek je na obrázku 9.2.



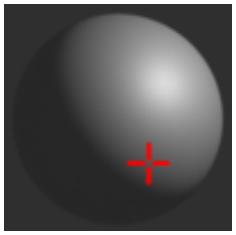
Obrázek 9.2. Normálová mapa vytvořená participantem.

9.2 Vyhodnocení výsledků testování

Hlavní problém nastal, když participant měl nastavit informace pro první značku. Participant nepochopil na první pohled, co se jakým parametrem nastavuje. Další problém nastal při skončení výpočtu, participant očekával, že se mu výšková mapa zobrazí sama.

9.3 Doporučení na zlepšení designu

Problém s nastavováním informací o vektoru by se dal řešit zjednodušením designu. Místo dvou nastavovacích ploch bych použil jen jednu. V této ploše by uživatel na obrázek koule myší položil červený zaměřovač, který by znázorňoval směr i náklon plochy v jednom. Nápad jsem zachytil do následujícího obrázku 9.3.



Obrázek 9.3. Návrh na zlepšení designu úpravy vektorů.

Takovéto řešení má výhodu ve snadném pochopení principu. Tímto způsobem ale nelze zadávat vektory s velkou přesností, což by do jisté míry znemožnovalo generovat kvalitní výsledky. Při dlouhodobém používání by mohlo dojít k frustraci uživatele, proto jsem tento návrh zamítl. Další možnost je natočení video tutoriálu, kde by se uživatelům mohl postup zadávání vektorů ve značkách vysvětlit lépe a na více příkladech. Tutoriály jsem natočil celkem dva, první vysvětluje základy programu a je dostupný na adrese [1]) a druhý popisuje pokročilejší funkce programu, ten je dostupný na adrese [2]).

Druhý nalezený problém byl, že se obrazovka po výpočtu výškové mapy nepřepla na záložku s vypočítanou výškovou mapou, tento problém jsem již odstranil.

[1]) `https://youtu.be/ANF_ZWEaMFg`
[2]) `https://youtu.be/bdK3Ef2_KI8`

Kapitola 10
Závěr

V této práci jsem rozebral téma rekonstrukce trojdimenziálního povrchu z jediného obrázku. Za tímto účelem jsem se věnoval algoritmům Shape from Shading. Pro svůj projekt jsem si vybral algoritmus Interactive Normal Reconstruction from a Single Image [1]. První jsem teoreticky rozebral rovnice algoritmu, pro pochopení jak algoritmus pracuje.

Následně jsem popsal svou implementaci algoritmu. Algoritmus jsem implementoval v jazyce Java 8. V implementaci jsem narazil na omezení algoritmu, musel jsem proto algoritmus částečně poupravit (viz kapitola 5.2). Po úpravě byly výsledky mnohem lepší, viz kapitola 5.

Po implementaci jsem testoval výstupy algoritmu. Výstupy dopadly nad mé očekávání, například u testování koule v kapitole 6.1 jsem dosáhl lepších výsledků, než byly publikovány v článku [1]. Algoritmus mile překvapil i u jiných výstupů, například u testování dřevěného povrchu v kapitole 6.6, kde vstupní obrázek nebyl dle mého názoru vůbec vhodný, přesto algoritmus dokázal vypočítat kvalitní rekonstrukci původního povrchu.

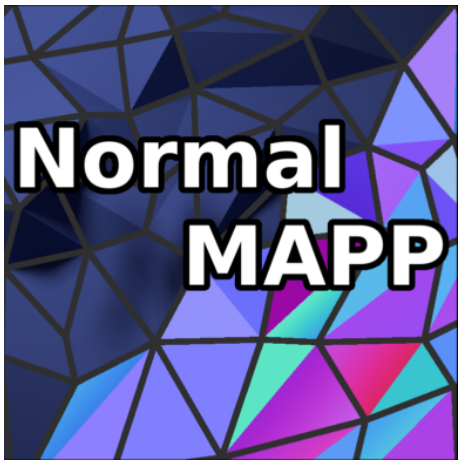
Po otestování algoritmu jsem zaznamenal vytvořené uživatelské prostředí i jeho počáteční návrhy. Uživatelské rozhraní jsem otestoval na finální aplikaci v kapitole 9. Z uživatelského testování vyvstaly dva problémy. Jako řešení na první problém jsem navrhl, že by bylo vhodné natočit video tutoriál, který by uživateli vše vysvětlil ve větším detailu. Video tutoriály jsem natočil celkem dva a jsou dostupné na adresách [1]) a [2]). Řešení druhého problému jsem již také zapracoval do výsledného programu.

V kapitole 8 jsem popsal ostatní fuknce, které program poskytuje.

[1]) `https://youtu.be/ANF_ZWEaMFg`
[2]) `https://youtu.be/bdK3Ef2_KI8`

10.1 Publikování programu

Výsledný program jsem nazval NormalMAPP. Pro program NormalMAPP jsem navrhl ikonu, která je na obrázku 10.1.



Obrázek 10.1. Ikona programu NormalMAPP

Program jsem otestoval pro operační systém Linux Xubuntu a Microsoft Windows 7. Pro Linux jsem vytvořil bashový script, který uživateli nainstaluje aplikaci na jeho systém, přiřadí aplikaci ikonu programu a přidá spouštecí script do seznamu aplikací systému. Pro Windows jsem pomocí programu Launch4j (dostupný na adrese [1])) vytvořil NormalMAPP.exe soubor, který program spouští. Do složky s výsledným programem jsem umístil soubor README.txt, kde jsou všechny instalační instrukce popsány.

NormalMAPP jsem publikoval na GitHub, repozitář se nachází na adrese [2]). Jsou zde publikovány jak zdrojové kódy aplikace tak výsledná aplikace. Všechny potřebné informace jak pro vývojáře tak pro uživatele, jsou popsány u příslušných adresářů.

[1]) `http://launch4j.sourceforge.net/`
[2]) `https://github.com/SedlaSi/NormalMAPP_repo/`

Literatura

[1] WU, Tai-Pang, et al. Interactive normal reconstruction from a single image. In: ACM Transactions on Graphics (TOG). ACM, 2008. p. 119.

[2] ZENG, Gang, et al. Interactive shape from shading. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005. p. 343-350.

[3] ALEXA, Marc; MATUSIK, Wojciech. Reliefs as images. ACM Trans. Graph., 2010, 29.4: 60:1-60:7.

[4] What is Depth Map? In i-Art Corporation. 2012
http://www.i-art3d.com/Eng/About_Depth.htm

[5] KERBER, Jens, et al. Computer assisted relief generation—A survey. In: Computer Graphics Forum. Blackwell Publishing Ltd, 2012. p. 2363-2377.

[6] HORN, Berthold K.. P.. ; SZELISKI, Richard S.; YUILLE, Alan L.. . Impossible shaded images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1993, 15.2: 166-170.

[7] COOK, Robert L.; TORRANCE, Kenneth E.. . A reflectance model for computer graphics. ACM Transactions on Graphics (TOG), 1982, 1.1: 7-24.

[8] Seven grayscale conversion algorithms. In Tanner Helland. 2011.
http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/

[9] STRONG, David M. Iterative Methods for Solving Ax = b - Gauss-Seidel Method. In Mathematical association of America.
http://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-gauss-seidel-method

[10] AHN, Song Ho. Convolution. In Song Ho. 2008.
http://www.songho.ca/dsp/convolution/convolution.html

[11] SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. Image processing, analysis, and machine vision. Cengage Learning, 2014.

[12] CHANDWADKAR, Radhika, et al. Comparison of Edge Detection Techniques. In: Proceedings of Sixth IRAJ International Conference. 2013.

[13] Convolution Filter. In Robo Realm. 2005.
http://www.roborealm.com/help/Convolution.php

[14] JAHNE, Bernd. Digital image processing: Concepts, algorithms, and scientific aplications. Berlin, Heildelberg: Springer-Verlag, 1997.

[15] Hicklin, Joe; Moler, Cleve; Webb, Peter. A Java Matrix Package. 2012
http://math.nist.gov/javanumerics/jama/

Příloha A
Instalační příručka

Aplikace byla testována pro operační systémy Linux Xubuntu a Microsoft Windows 7.

A.1 Instalace na operační systém Linux

Pro spuštění aplikace na systému Linux budete potřebovat program GraphicsMagick, který můžete nainstalovat příkazem:

$$apt - get \quad install \quad graphicsmagick$$

Stáhněte si následující soubor [1]). Soubor rozbalte a přes příkazovou řádku spusťte skript *setup_linux.sh* následujícím příkazem:

$$./setup\_linux.sh$$

Skript možná bude nutné povolit, jako spustitelný program, to lze udělat následujícím příkazem:

$$chmod \quad 755 \quad setup\_linux.sh$$

Po spuštění skriptu zadejte heslo pro uživatele root (pokud nejste již jako root přihlášeni) a stiskněte klávesu Enter. Budete dotázáni na uživatele, kterému má být aplikace nainstalována, zadejte uživatelské jméno a stiskněte Enter. Poté se program nainstaluje. Program lze následně spustit z příkazové řádky příkazem *NormalMAPP*. Po restartování systému se program zobrazí ve vašem seznamu nainstalovaných aplikací. Aplikaci lze odinstalovat spuštěním skriptu *uninstall_linux.sh*, který se nachází v adresáři */home/$USER/NormalMAPP/*.

Pro více informací čtěte přiložený README.txt soubor.

[1]) `https://github.com/SedlaSi/NormalMAPP_repo/blob/master/build/NormalMAPP.zip`

A.2 Instalace na operační systém Microsoft Windows

Stáhněte si následující soubor [1]). Soubor rozbalte a ve složce *NormalMAPP* \ *src* naleznete spouštecí soubor aplikace NormalMAPP.exe. Pokud vám nejdou načíst obrázky nainstalujte si do počítače program GraphicsMagick, ve složce *NormalMAPP* \ *src* se nachází konfigurační soubor *conf.txt*, v tomto souboru zadejte cestu ke složce s nainstalovaným programem GraphicsMagick, více informací v samotném souboru *conf.txt*. Pro další informace čtěte přiložený README.txt soubor.

A.3 Jak zkompilovat aplikaci

Projekt byl vytvořen v programovacím prostředí IntelliJ IDEA 2016.1. Ultimate. Aplikace používá pro propojení s knihovnami soubor MANIFEST.MF (NormalMAPP/src/Resources/META-INF/MANIFEST.MF), je zde zakódováno, že se externí knihovny nacházejí ve složce ./lib/. Pro přeložení projektu v IDE ale můžete použít knihovny stažené z Maven repozitáře. Závislosti knihoven jsou popsány v souboru pom.xml (NormalMAPP/pom.xml).

Pro přeložení projektu pod systémem Windows musíte ještě specifikovat cestu k programu GraphicsMagic. Tato cesta je uložena ve třídě java.gui.session.Session pod názvem $GRAPHICS\_MAGIC\_FOLDER$. Je čtena z metody *loadGraphicsMagick()*.

[1]) `https://github.com/SedlaSi/NormalMAPP_repo/blob/master/build/NormalMAPP.zip`

Příloha B
Použité knihovny a externí programy

B.1 Knihovny

- vecmath (adresa knihovny [1]))
- jama-1.0.3 (adresa knihovny [2]))
- jai-imageio-core-1.3.0 (adresa knihovny [3]))
- im4java-1.2.0 (adresa knihovny [4]))

B.2 Externí programy

- GraphicsMagick (adresa programu [5]))
- Launch4j (adresa programu [6]))

[1]) https://java.net/projects/vecmath
[2]) http://math.nist.gov/javanumerics/jama/
[3]) https://github.com/jai-imageio/jai-imageio-core
[4]) https://sourceforge.net/projects/im4java/
[5]) http://www.graphicsmagick.org/
[6]) http://launch4j.sourceforge.net/

Příloha C
Obsah přiloženého DVD