VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYNTÉZA TEXTUR

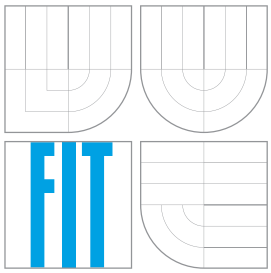
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROBERT HAVELKA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYNTÉZA TEXTUR

TEXTURE SYNTHESIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ROBERT HAVELKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MIROSLAV ŠVUB
SUPERVISOR

BRNO 2008

Zadání bakalářské práce/5677/2007/xhavel25

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Havelka Robert**
Obor: Informační technologie
Téma: **Syntéza textur**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte základy zpracování obrazu. Zaměřte se zejména na problematiku textur a metod jejich reprezentace.
2. Prostudujte dostupné materiály na téma syntéza textur (vytváření složitejších a rozsáhlých textur z jednodušších příkladů).
3. Vyberte vhodné metody a navrhněte jednoduchou knihovnu pro popis a syntézu textur.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace metod.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje. Zvažte další pokračování v rámci diplomové práce.
6. Vytvořte stručný plakát prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních tří bodů zadaní.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Švub Miroslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

Licenční smlouva je uvedena v archivním výtisku uloženém v knihovně FIT VUT v Brně.

Abstrakt

Práce čtenáři objasní význam texturování a obsahuje přehled základních postupů při syntetizování textur. Generování textur je popsáno především z pohledu procedurálních generátorů. Vysvětlen je význam použití Fourierovy transformace a její zjednodušené verze, diskrétní kosinové transformace. Při modifikacích textur je použit Gaussův šum a Gaussův filtr. Pro mísení textur je v práci použit konvoluční teorém. V rámci práce je také popsán návrh a implementace aplikace, která umožňuje experimentovat s vytvářením textur.

Klíčová slova

procedurální texturování, OpenCV, Fourierova transformace, DCT, gaussovský šum, konvoluční teorém

Abstract

This thesis deals with texturing, its importance and basic list of the processes during texture synthetising is included. Texture generating is mainly described from the procedural generator point of view. There is also explained the sense of using Fourier's transformation and its simplified version, the discreet cosine transformation. For texture modiffication, Gaussian noise and gaussian filter is used. The convolution theorem is applied for the texture mixing. Within the scope of this thesis, there is also described the application implementation, which enables experimenting with texture creating.

Keywords

procedural texturing, OpenCV, Fourier transform, DCT, Gaussian noise, convolution theorem

Citace

Robert Havelka: Syntéza textur, bakalářská práce, Brno, FIT VUT v Brně, 2008

Syntéza textur

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Miroslava Švuba.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Robert Havelka
13. května 2008

Poděkování

Děkuji panu Ing. Miroslavu Švubovi za čas, který mi věnoval při řešení této bakalářské práce.

Obsah

Kapitola 1

Úvod

Pojem syntéza textur zastřešuje obrovské množství způsobů pomocí kterých lze získávat textury a následně s nimi pracovat. Cílem této bakalářské práce je uvedení do problematiky syntézy textur, nastínění principů jak je lze upravovat a možnosti jejich použití. Pro demonstraci jsem vytvořil aplikaci, díky které má uživatel možnost vyzkoušet si generovat vlastní textury a pomocí různých funkcí je může upravovat.

V kapitole 2 pokusím vysvětlit obecné použití texturování. Principy používané při texturování a pojmy s těmito postupy spojené. V kapitole 3 osvětlím postupy, které jsou použity v samotné aplikaci. Návrh aplikace a výběr použitých knihoven je uveden v kapitole 4. Kapitola 5 pojednává o návrhu ovládání aplikace a vlastním kódu aplikace. Experimenty a získané textury nalezneme v kapitole 6. Zhodnocení práce je v kapitole 7.

Kapitola 2

Úvod do texturování

V počítačové grafice se používají modely a vizualizace různých jevů. Modely jsou pouze množinou trojúhelníku a jiných plošných útvarů, které ohraničují objekt. Při jejich zobrazení je třeba tyto plochy vyplnit texturou. Ty mohou být již dané nebo závislé na určitých okolnostech. To nás přivádí k pojmu syntéza textur, který zahrnuje činnosti spojené s vytvářením textur. V této kapitole uvedu základní pojmy spojené s využitím textur a texturováním a dále pojmy spojené s generováním a úpravou textur.

2.1 Textury

2.1.1 Co je textura?

V úvodu této kapitoly jsem uvedl pojem textura. Pro ujasnění tohoto pojmu uvedu jednoduchou definici, která vychází z [7]: *Textura je n-rozměrná pixelová mapa.* Jak definice napovídá, jedná se o matici či matici matic, kde má každý pixel přiřazenu barvu. Pixely se u textur nazývají *textely.*

2.1.2 Datové textury

Do této skupiny textur zařazujeme textury, které jsou například vytvořeny lidskou rukou jako malba a fotoaparátem či jiným způsobem převedeny do elektronické podoby. Dále textury, které jsou získány přímo pomocí kreslícího programu, ve kterém můžeme texturu nakreslit a upravovat ji po jednotlivých bodech. V neposlední řadě nesmíme zapomenout na rozmanité krásy přírody, které nám dávají neomezený zdroj textur a při použití vhodné zaznamenávací techniky je můžeme využít při texturování.

Výhoda datových textur spočívá v jejich vytváření. Zkušený grafik může lehce vytvořit nádherné textury, které budou okamžitě vypadat podle jeho přání. Takto vytvořené textury pak snadno uloží do vhodného formátu (BMP, JPG,...) a jsou okamžitě připraveny k použití. Další jejich výhoda je, že při použití zatěžují procesor pouze v míře, jaká je potřebná pro načtení souboru textury do paměti.

Textura však může způsobit problém při jejím skladování. Velikost textury je dána jejím rozměrem a kvalitou uložení při použití komprese. Z toho vyplývá, že v rozsáhlejších aplikacích, které vyžadují desítky, stovky i více vzorků textur, se výsledné paměťové nároky mohou stát neúnosnými.

Použití datových textur je tedy vhodné když:

- textura se nebude měnit nebo její změny jsou lehce uskutečnitelné pomocí programu
- vytvoření shodné textury pomocí algoritmu by velmi vytížilo procesor
- pomocí algoritmů není možné dosáhnout požadovaných výsledků
- potřebujeme malý vzorek textury pro pokročilé generátory jako předlohu

2.1.3 Generované textury

Generované textury vznikají pomocí vhodně implementovaných algoritmů. V některých případech i za použití již vytvořeného vzorku textury.

Výhoda tohoto přístupu získávání textur je v tom, že lze poměrně jednoduše vytvářet textury, které závisí například na matematických funkcích. Pro představu si každý může zkusit nakreslit přesnou křivku, která vznikne pomocí funkce $y = \sin x$. Dále paměťové nároky na uložení takové textury jsou menší, protože je uložený kód, který texturu dokáže vygenerovat v nejvyšší kvalitě a na libovolně velkou plochu. Při generování jsme pouze omezeni pamětí, kterou má program dovoleno použít pro uložení výsledku.

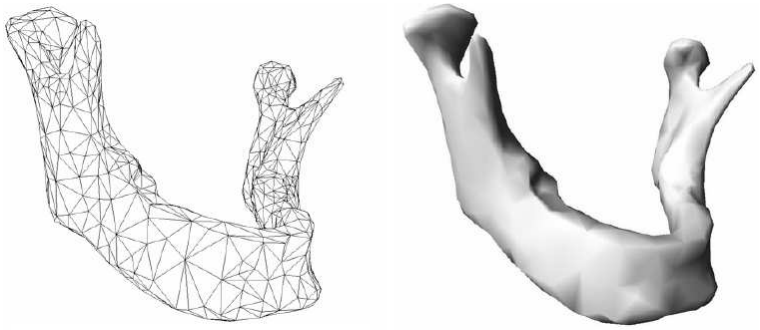
Tento přístup je však naprosto nevyhovující v případě, že programátor nedokáže správně implementovat posloupnost potřebných úkonů, aby docílil požadovaného vzhledu.

Použití generovaných textur je tedy vhodné když:

- potřebujeme ušetřit místo (např. aplikace pro mobilní telefon)
- textura má mít pseudounikátní[1] vzhled
- textura je závislá na množině proměnných (vizualizace jevů)

2.1.4 Texturování

Samotná textura však bez využití nemá význam. Chceme-li textury použít při zobrazení, procházejí procesem, který se nazývá *texturování*. Opět uvedu definici vycházející z [7]: *Texturování je proces, při kterém jsou jednotlivé textely nanášeny na povrch předmětu při rasterizaci*. Pro představu uvádím obrázek 2.1.



Obrázek 2.1: Příklad modelu (vlevo) bez textury a po provedení texturování (vpravo).

Více o texturování se můžeme dočíst například v práci [12]

[1] V návaznosti na téma generátorů náhodných čísel se vedou diskuze zda vůbec existují náhodná čísla. Vzhled textur, které jsou založeny na těchto generátorech proto nemůžeme jednoznačně označit na unikátní.

2.2 Fourierova transformace

Protože textura je n-rozměrná matice, můžeme si ji představit jako diskrétní signál a použít Fourierovu transformaci. Použití transformace nám umožňuje upravovat textury v oblasti spektra. Díky tomu můžeme utlumit či posílit námi zvolené složky a tím měnit vlastnosti zpětně syntetizované textury. Více podrobností o použití vhodné Fourierovy transformace naleznete v 3.3.

2.3 Konvoluční teorém

Při operacích s jednotlivými texturami se používá také princip vycházející z *konvolučního teorému*. Ten říká, že pokud se aplikuje konvoluce ([8]) na dva signály, může se stejného výsledku dosáhnout násobením jejich spekter, které získáme po Fourierově transformaci.

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

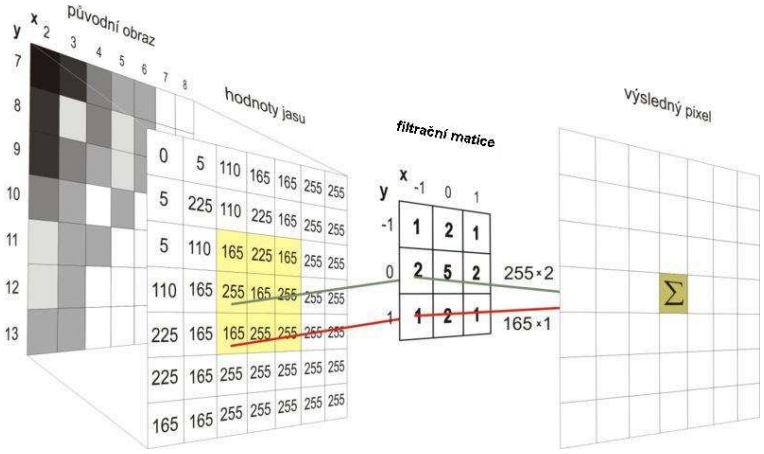
kde $\mathcal{F}$ a $\mathcal{F}^{-1}$ je Fourierova transformace a transformace k ní inverzní

f a g jsou signály se kterými chceme provést konvoluci

Pokud je tedy požadována konvoluce dvou signálů, je možné pomocí využití tohoto teorému výpočet zjednodušit. Podmínkou je však jednoduché získání spekter signálu. Informace vychází z [3].

2.4 Filtrování

Další operace s texturou mohou být realizovány pomocí filtrů. Řeč je o filtrech, které prochází texturu bod po bodu v závislosti na filtrační matici vytvoří novou texturu kde jednotlivé textely jsou vypočteny pomocí referenčního bodu a jeho okolí. Filtrační matice také udává rozměr okolí a přiřazuje každému bodu okolí váhu jakou ovlivní výslednou barvu vypočítaného bodu.



Obrázek 2.2: Příklad použití filtrační matice, jinak také zvané konvoluční jádro.

Při použití vhodné filtrační matice můžeme:

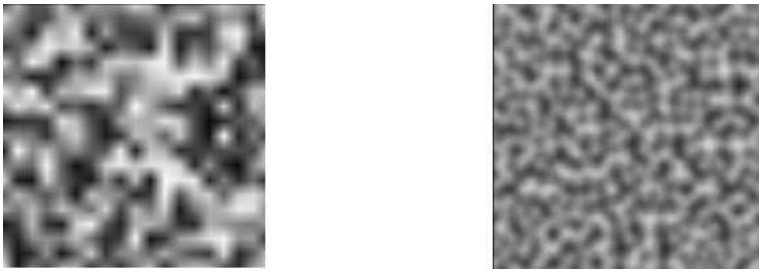
- ostřit texturu - např. přibližující se objekt, který je texturován
- rozostřit texturu - např. oddalující se objekt ([6])
- zvýraznit hrany (u obrázků se používá pro detekci hran [4])
- a další ...

2.5 Použití šumu při tvorbě textur

Vzhledem k myšlence zmíněné v 2.1.3, musíme nějakým způsobem dosáhnout prostředků, které nám umožní do textur vložit náhodnost. Způsobů se nabízí celá řada. Nejpoužívanější jsou pravděpodobně generátory pseudonáhodných čísel a generátory šumu.

Šumy, které generátory tvoří mohou mít charakteristické znaky (viz [14]), které jsou patrné na pohled. V jiných případech můžou šumy vypadat velmi podobně a abychom poznali rozdíl mezi nimi, je třeba nahlédnout do jejich spektra. Několik příkladů druhů šumu

- Bílý šum (White Noise)
- Mřížkový šum (Lattice Value Noise)
- Perlinův šum (Perlin Noise)
- Bodový šum (Spot Noise)



Obrázek 2.3: Ukázka mřížkového šumu (vlevo) a Perlinova šumu (vpravo)

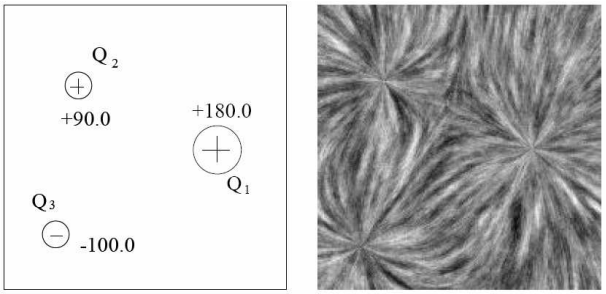
Každý šum má jiné využití. Hojně používaný Perlinův šum se využívá například při vizualizaci mraků. Více podrobností naleznete v diplomové práci [9]. Šum, který je součástí mojí aplikace, je podrobněji popsán v kapitole 3.2.

2.6 Praktické využití texturování

Shrneme-li možnosti získávání textur a také možnosti jejich úprav, můžeme po zamyšlení nalézt mnoho způsobů jak textury prakticky použít. Opět zde hraje roli i způsob jejich získávání.

Textury nejsou pouze pro vyplnění prázdných stěn v počítačové hře. Díky syntéze a následném zobrazení můžeme vidět výsledky měření a výpočtů, které jsou za normálních

okolností lidskému oku skryty a nebo není způsob jak je v reálném prostředí sledovat. Například zobrazení elektromagnetického pole, směr působení magnetického pole, vizualizace namáhání materiálu a mnoho dalších.



Obrázek 2.4: Příklad generované textury, která je závislá na hodnotách měření.

Více informací o použití a aplikaci můžeme nalézt například v práci [10].

Kapitola 3

Přehled problematiky

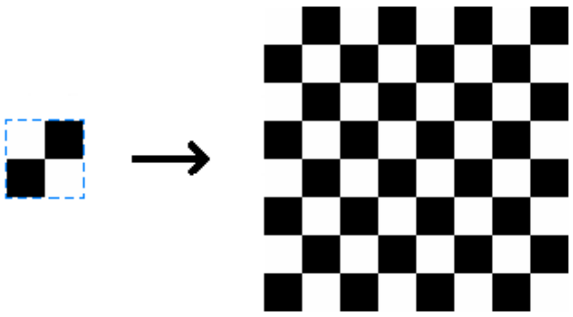
Tato kapitola se bude podrobněji zabývat pojmy, které jsou podstatné pro pochopení funkčnosti navrhované aplikace. Čerpáno bylo z [5] a [11].

3.1 Procedurální generování textur

Procedurální generátor textur může být naprosto nezávislý na proměnných a generovat stále stejnou texturu. Použití procedurálního generování textur ale skrývá daleko vetší možnosti. Pomocí těchto generátorů vznikají textury jaké jsou například vidět na obrázcích 2.3, 2.4 a 3.2.

Získávání textur touto cestou může být jednou z jednodušších variant. Výsledné textury závisí na zkušenostech programátora a jeho schopnosti reprezentovat vzory pomocí matematických zápisů a jiných postupů, které ovlivní výsledný vzhled textury.

Jako jednoduchý procedurální generátor je možné uvést příklad generátoru šachovnicového vzoru. Algoritmus klade vhodně připravený vzorek vedle sebe a nad sebe a díky tomu může v paměti vytvořit libovolně velikou texturu(obr. 3.1).

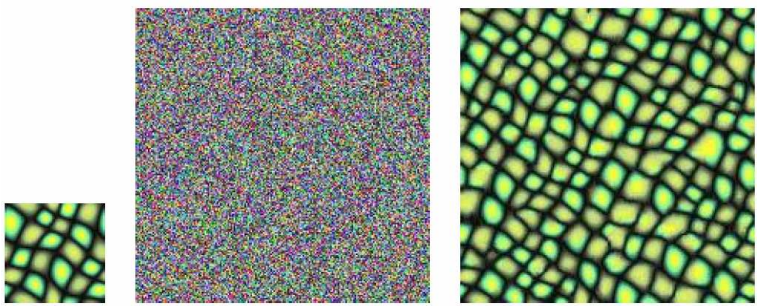


Obrázek 3.1: Vytvoření šachovnice z malého vzorku

Stejný šachovnicový vzor ovšem může zkušenější programátor kompletně vytvořit pomoci vhodě zvolených cyklů a správných podmínek. Tím se stane nezávislým na vzorové textuře.

Stejného cíle jako na obrázku 3.1 se snaží dosáhnout pokročilejší metody generování textur. Vzorek textury je však je pouze předlohou. Tato předloha nemusí být přizpůsobena k jakékoliv vzájemné návaznosti. To znamená, že pokud bychom vzorek kladli vedle sebe,

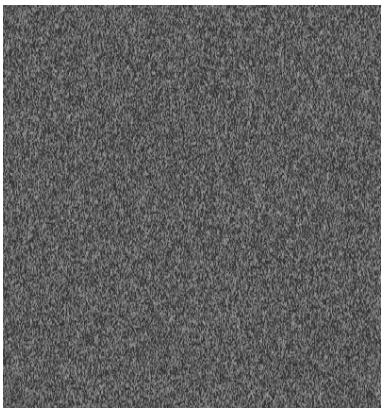
jako v příkladě s šachovnicí, budou jasně vidět přechody. Algoritmus dovede tuto texturu rozvinout (expandovat) na libovolně velikou plochu. Jak je na obrázku 3.2 patrné, generátor, který je použit, rozšiřuje texturu na plochu, která je vyplněná šumem. Tím se do procesu rozvinutí textury zavádí náhodnost. Více podrobností se můžeme dočíst v práci [13].



Obrázek 3.2: Vytvoření textury (vpravo) za pomoci malého vzorku (vlevo) a plochy vyplněné šumem (uprostřed)

3.2 Generátor Gaussovského šumu

Gaussovský nebo-li bílý šum se na pohled téměř neliší od ostatních šumů. Chceme-li zjistit čím je tento šum specifický, je třeba nahlédnout do jeho spektra. Definice (viz [1]) je založená právě na údajích, které lze vyčíst z jeho spektra a zní: *Bílý šum je náhodný signál s rovnoměrnou výkonovou spektrální hustotou.* Ve spektru bílého šumu jsou tedy rovnoměrně zastoupeny všechny složky a jeho Fourierova transformace může vypadat jako na obrázku 3.3.



Obrázek 3.3: Příklad vzhledu spektra bílého šumu.

Jak je na obrázku 3.3 zřejmé, v jeho spektru jsou zastoupeny všechny složky ve stejné míře, takže žádná svojí hodnotou nikterak významně nepřevyšuje ostatní. Což je žádoucí pro naše další použití v této práci.

3.3 Diskrétní kosinová transformace

Jak jsem zmínil v odstavci 2.2 a následně i vyplývá z podkapitoly 3.2, s texturami je možné pracovat jako s n-rozměrným signálem. To nám umožňuje použít diskrétní Fourierovu transformaci pro dvourozměrný signál.

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

kde M a N udávají šířku a výšku signálu textury

u a v jsou souřadnicemi počítaného vzorku

Textury pak můžeme měnit pomocí úprav jejího spektra. Pokud potlačíme vzorky s vysokou frekvencí, zpětně syntetizovaná textura nebude obsahovat ostré přechody. Naopak pokud potlačíme nízké frekvence, textura bude obsahovat ostré přechody.

Nesmíme však zapomenout na fakt, že obecně Fourierova transformace počítá složky jako komplexní čísla, což je pro nás nepraktické. V důsledku používání Fourierovy transformace na obrázky, tedy i textury, vznikly upravené verze Fourierovy transformace. Jak napovídá název této sekce, jedna z nejpoužívanějších úprav se nazývá *diskrétní kosinová transformace*. Jde o Fourierovu transformaci, která počítá s diskrétním signálem a vrací pouze reálné hodnoty jednotlivých složek.

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]$$

kde $N1$ a $N2$ udávají šířku a výšku signálu textury

$k1$ a $k2$ udávají souřadnice ve spektru

S takto získanými hodnotami se lépe pracuje a není potřeba je upravovat.

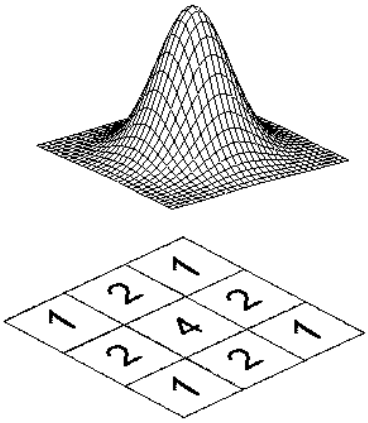
3.4 Gaussovský filtr

Gaussovský filtr můžeme též nazvat rozostřovací filtr. Jeho filtrační matice může být libovolně veliká, ale musí splňovat podmínku Gaussova rozložení.

Při použití se přikládá filtr středovým bodem na bod, jehož hodnotu, chceme filtrací získat. Použijeme-li filtr, který je uvedený na obrázku 3.4 každý bod vynásobíme jemu přiřazenou váhou (1, 2 nebo 4) a navzájem je sečteme. Na závěr musíme jejich součet podělit poměrem zastoupení všech bodů. To v našem případě znamená hodnotou 16. Z filtru je patrné, že nejvyšší váhu na výsledný bod bude mít v případě Gaussovského rozložení poměrů referenční bod, který se počítá.

Filtry jsou také navrhovány tak, že se vypouští závěrečné dělení, to v případě, že filtr vypadá jako na obrázku 3.5. Závěrečné dělení je již zahrnuto v jednotlivých váhových koeficientech.

Jedno z úskalí při používání filtrů jsou okraje textury. Filtr se může snažit počítat s hodnotami, které jsou neznámé, tedy neexistuji, protože filtrační matice je příliš veliká (viz obrázek 3.5). Zde se mohou aplikovat různé postupy, které filtru připraví vhodná čísla pro počítání. Pravděpodobně nejlepší řešení je zrcadlení textury za její okraj. Tím zůstane nejvěrohodněji zachované okolí bodu.



Obrázek 3.4: Znázornění zachování Gaussova rozložení při návrhu filtrační matice 3x3.

0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00038771	0.01330373	0.11098164	**0.22508352**	0.11098164	0.01330373	0.00038771
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067

Obrázek 3.5: Gaussovský filtr o rozměrech 7x7 u kterého není potřeba závěrečného dělení.

Kapitola 4

Návrh aplikace

Aplikace má za úkol umožnit uživateli vytvořit procedurální textury se kterými díky sadě jednoduchých operací bude moci dále experimentovat. Dále má dovolit tyto procedurálně vytvořené textury měnit změnou jejich spektra pomocí šumu. Také bude umožněno použití jednoduchého Gaussova filtru.

Celá aplikace by měla mít jednoduché ovládání a zobrazovat výsledky zvolených operací.

4.1 Použití knihoven a prostředků systému

Při návrhu aplikace je důležité vybrat vhodné knihovny a prostředky pro práci s texturami. Také použití prostředků operačního systému nám usnadní práci při tvorbě grafického uživatelského rozhraní aplikace (GUI[1]).

4.1.1 OpenCV

OpenCV[2] je volně dostupná knihovna vytvořená společností Intel pro usnadnění práce při grafických operacích. Záměr této knihovny je shrnout stále se opakující činnosti, které jsou spojeny se zpracováním médií. Sama knihovna je tvořena několika dílčími částmi, které je možné samostatně použít v závislosti na požadavcích programátora. Díky použití knihovny se programátor může zaměřit na stěžejní části jeho práce. Operace, které v knihovně můžeme nalézt jsou například:

- načítání a ukládání obrázků a videí
- práce s okny v prostředí MS Windows a Linux
- segmentace a rozpoznávání útvarů
- rozpoznávání obličeje
- sledování pohybu
- operace s maticemi
- Fourierova transformace a filtry

Podrobnější popis použitých funkcí nalezneme v kapitole 5.

[1] Graphics User Interface
[2] OpenCV - Open Computer Vision Library

4.1.2 WinAPI

Aplikaci bude vyvíjena pod systémem MS Windows, proto bude vhodné při implementaci grafického rozhraní aplikace použít prostředky operačního systému pro práci s okny. WinAPI[3] umožňuje snadno vytvářet grafická uživatelská rozhraní obsahující různé ovládací prvky.

4.2 Barevně nebo černobíle?

Důležitým rozhodnutím bylo zda s texturami pracovat barevně nebo černobíle. Kvůli jednoduššímu přístupu při diskrétních kosinových transformacích a dalších navrhovaných procedurách bude postačovat vyjádření obrázku v odstínech šedi.

4.3 Uložení textur v paměti

Protože se aplikace zabývá tvorbou textur, je důležité správně zvolit datovou strukturu pro uložení textury v paměti. Vhodnou volbou mohou odpadnout problémy spojené s převody na jiné datové struktury. Tyto konverze mohou po několikanásobném použití způsobit degeneraci vzorku. To znamená, že mnohokrát převáděné textely nebudou mít korektní odstín nebo barvu.

V zájmu vyhnutí se zbytečným problémům jsem zvolit pro uložení strukturu, která je definovaná v knihovně OpenCV - `IplImage`. Výhoda spočívá v jednodušším použití jednotlivých funkcí, které jsou v OpenCV obsaženy a které je vhodné v aplikaci použít a tedy nebudeme muset strukturu převádět.

4.4 Procedurální textury a jejich úprava

V aplikaci chceme vytvářet textury pomocí sady jednoduchých funkcí. Je nutné tedy zvolit vhodné funkce, které zajistí dostatečnou flexibilitu při generování, ale nebudou pro uživatele příliš složité.

4.4.1 Generátor obdélníkového vzoru

Myšlenka tohoto generátoru je poměrně jednoduchá. Za základ považuje obdélník, který vyplňuje polovinu cílové velikosti vzorku. Generátoru určíme kolikrát se tento obdélník bude ve výsledném vzorku[4] opakovat (viz obrázek 4.1).

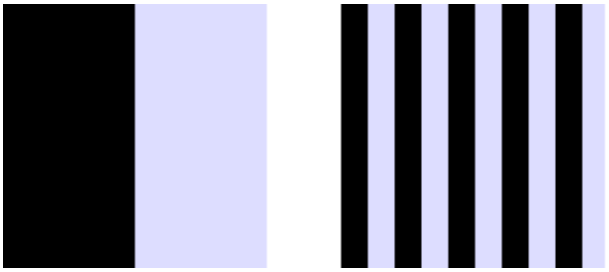
Další ovlivnění vzhledu vzorku bude možné pomocí rovnoměrné změny odstínu a postupného zužování jednotlivých obdélníků. Obojí názorně objasní obrázek 4.2

4.4.2 Generátor založený na matematických funkcích sinus a kosinus

Další generátor, který bude poskytovat základní vzorek pro další operace bude pracovat s matematickými funkcemi sinus a kosinus. Zde bude generátor ovlivňovat pouze výběr použité funkce a počet půlperiod opakujících se ve vzorku. Pro názornost uvádím příklad na obrázcích 4.3 a 4.4.

[3] WinAPI - Windows Application Programming Interface - neboli rozhraní pro programování aplikací v prostředí Windows

[4] Ve většině zobrazených vzorků je záměrně upravená jejich bílá barva, aby byl zřejmý jejich okraj.



Obrázek 4.1: Počet vygenerovaných obdélníku ve vzorku.

Obrázek 4.2: Rovnoměrná změna odstínu (vlevo), zužování obdélníků (vpravo).

4.4.3 Funkce pro úpravu a rozšíření

Samotné generátory popsané v 4.4.1 a 4.4.2 sice již generují vzorky, které jsou texturami, avšak cílem práce je ukázat něco více. Návrh následujících čtyř funkcí bude uvažovat již existující vzorky. Princip většiny bude spočívat v použití originálního vzorku a nového vzorku, který se vytvoří použitím originálu po jeho otočení o 90°. Následně budeme moci vzorek upravovat pomocí principů, které jsou uvedeny níže.

Součtová úprava vzorku

Funkce projde originální texturu a její o 90° otočenou kopii přičemž sečte hodnoty bodů, které leží na stejné pozici. Více podrobností je uvedeno v části 5.4.

$$vzorek_{vystupni}[x,y] = \left(vzorek_{original}[x,y] + vzorek_{otoceny}[x,y]\right)$$
$$x = \{0,1,\dots M\}, y = \{0,1,\dots N\}$$

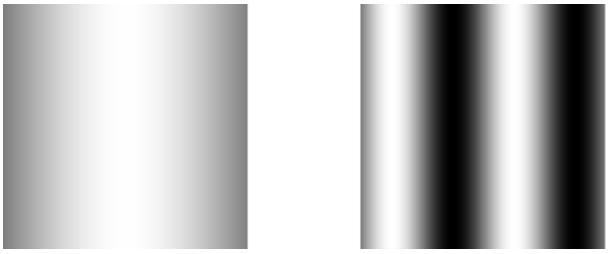
kde M a N jsou rozměry vzorku

Rozdílová úprava vzorku

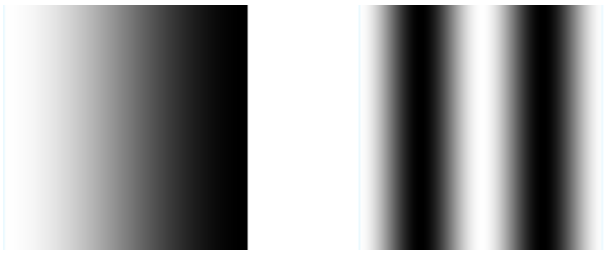
Tato funkce od sebe bude body ležící na stejné pozici odčítá. Více v 5.4

$$vzorek_{vystupni}[x,y] = \left(vzorek_{original}[x,y] - vzorek_{otoceny}[x,y]\right)$$
$$x = \{0,1,\dots M\}, y = \{0,1,\dots N\}$$

kde M a N jsou rozměry vzorku



Obrázek 4.3: Sinus – jedna půlperioda (vlevo), čtyři půlperiody (vpravo).



Obrázek 4.4: Kosinus – jedna půlperioda (vlevo), čtyři půlperiody (vpravo).

Úprava vzorku získáním maxima

Jak nadpis napovídá, funkce projde opět oba vzorky a ve výsledku se projeví vždy vyšší hodnota z dvojice bodů. Zápis by mohl vypadat následovně.

$$vzorek_{vystupni}[x,y] = MAXIMUM(vzorek_{original}[x,y], vzorek_{otoceny}[x,y])$$
$$x = \{0,1,\dots M\}, y = \{0,1,\dots N\}$$

kde M a N jsou rozměry vzorku

Úprava vzorku získáním minima

Obrácená funkce k předchozí je při použití minima a je použitá nižší hodnota z dvojice bodů.

$$vzorek_{vystupni}[x,y] = MINIMUM(vzorek_{original}[x,y], vzorek_{otoceny}[x,y])$$
$$x = \{0,1,\dots M\}, y = \{0,1,\dots N\}$$

kde M a N jsou rozměry vzorku

Zrcadlové převrácení textury

Tato funkce pouze umožní vzorek textury převrátit podle vertikální osy, která leží ve středu textury.

Rozostření textury pomocí Gaussovského filtru

Zde je v rámci ukázky naprogramován jednoduchý Gaussovský filtr. Jeho filtrační matice je 3x3 a je shodná s maticí, která je vyobrazená na obrázku 3.4.

4.5 Generování textury pomocí úprav spektra

Nejzajímavější částí aplikace je použití úprav spektra generovaných vzorků pomocí šumu. Tento princip vychází z *konvolučního teorému*, který je popsán v 2.3.

Zde se tedy v aplikaci objeví generátor Gaussovského šumu, funkce zprostředkovávající diskrétní kosinovou transformaci a její inverzní varianta.

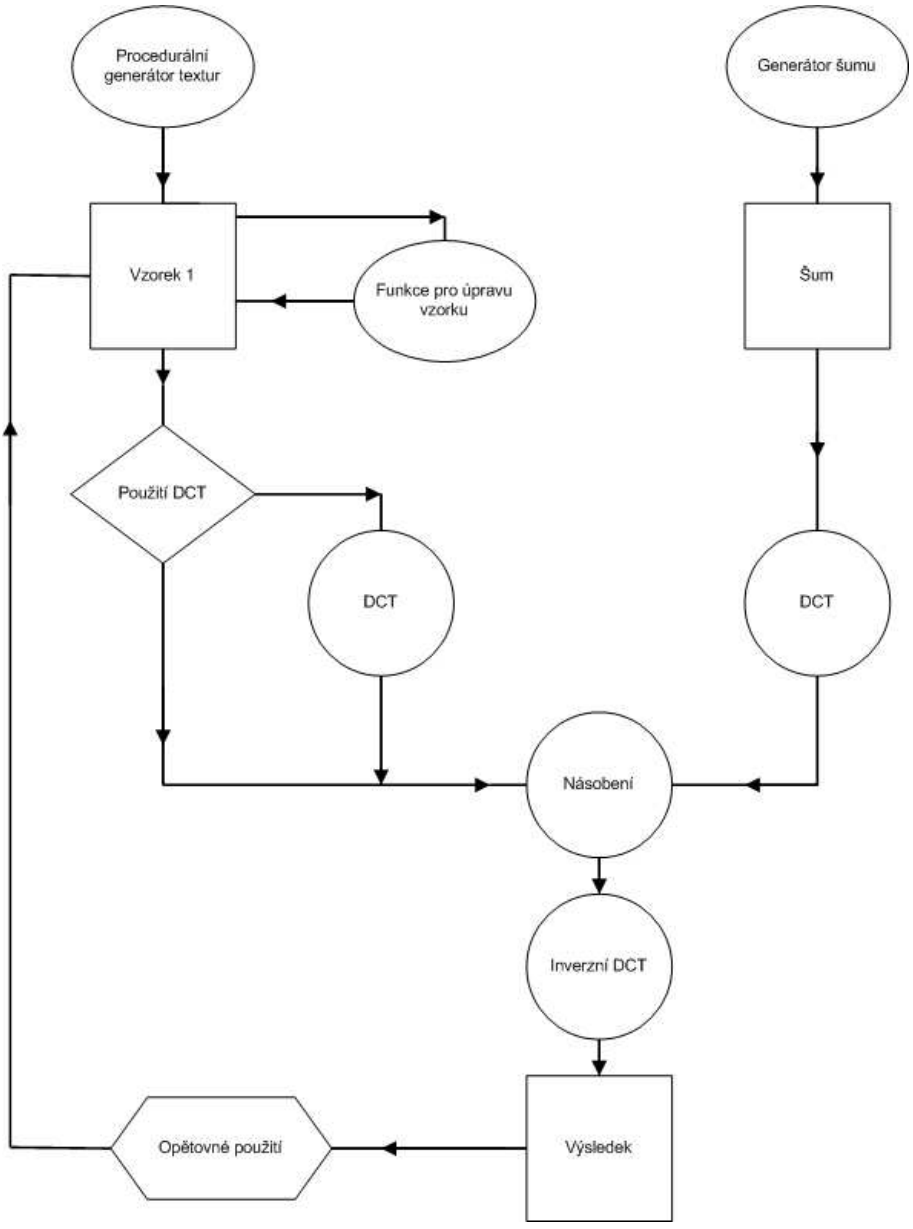
Generátor šumu

Jak již bylo zmíněno potřebujeme získat šum, který bude vhodné mísit se vzorky. Kvůli velmi klidnému průběhu spektra je tedy ideální použít generátor bílého šumu[5]. V praxi tedy získáme vhodné spektrum, kterým můžeme vynásobit spektrum našeho vzorku aniž by některé složky ovlivnil výrazněji než ostatní. To nám zaručí specifický vzor v každé textuře v závislosti na jejím vzorku.

4.6 Provázanost aplikace

Aby aplikace dovolovala co nejvíce možností, budeme při implementaci sledovat myšlenku obsaženou v diagramu na obrázku 4.5. Diagram nastiňuje možnost přenosu vzorků mezi jednotlivými etapami a jeho ovlivnitelnost funkcemi.

[5] Jak bylo již popsáno v dřívějších kapitolách, má rovnoměrné výkonové rozložení po celém spektru.



Obrázek 4.5: Uvažované propojení programu.

Kapitola 5

Implementace

V této části popíšu jak jsem implementoval navržené funkce a uvedu nejdůležitější postupy jakými řeší daný problém. Také zde budou deklarace funkcí a uvedu deklarace některých funkcí, které jsou obsaženy v knihovně OpenCV a výrazně zjednodušili vývoj aplikace.

5.1 Programovací jazyk

Při výběru programovacího jazyka jsem přihlédl k jeho jednoduché použitelnosti v kombinaci s OpenCV a WinAPI. To velmi jednoznačně určilo vítěze, kterým se stal programovací jazyk C++.

5.2 Textura ve struktuře `IplImage`

Jak jsem uvedl v odstavci 4.2, budu vytvářet textury v odstínech šedi. To znamená použití jednoho kanálu. Datový typ, pro uložení jednotlivých bodů ve struktuře `IplImage` jsem zvolil `IPL_DEPTH_32F`[1].

Protože se jedná o velmi veliký a přesný rozsah, který datový typ `IPL_DEPTH_32F` nabízí, určil jsem, že hodnota černého bodu bude dána hranicí $0,0$ a bílý bod bude reprezentován hodnotou $1,0$. To nám dává rozmezí $0,0$-$1,0$ avšak s obrovskou přesností, která se ani zdaleka nevyužije. Při zobrazení bodu pak jednoduše každý bod vynásobíme hodnotou maximální úrovně jasu.

Dále texturu potřebuji uložit do paměti. Je potřebné pro ni tuto paměť vyhradit a při ukončení práce s touto pamětí ji zase uvolnit. Zde se projevuje výhoda použití OpenCV. Jeho funkce pro práci s formátem `IplImage` mi tyto činnosti zajistí. Využil jsem tyto funkce:

```
IplImage* cvCreateImage(CvSize rozmery, int presnost, int pocet_kanalu);
```

`rozmery` – Pro zadávání rozměrů je praktické uvést konstruktor `cvSize(sirka, vyska)`.

`presnost` – Dostačující znalost `IPL_DEPTH_32F` a `IPL_DEPTH_8U`[2]

`pocet_kanalu` – Protože pracuji s odstíny šedi je dostačující jeden kanál.

```
void cvReleaseImage( IplImage** obraz);
```

[1] 32-bitové číslo float definované v OpenCV

[2] bezznaménkové 8bitové celé číslo

`obraz` – ukazatel na strukturu, kterou chci uvolnit

Vytvoření struktury a alokaci paměti provede funkce `cvCreateImage`. Stejně tak uvolnění paměti není díky funkci `cvReleaseImage` žádný problém. Zůstává tedy prostor pro plnění struktury generovanými daty. Protože pracuji s dvourozměrnými texturami poslouží funkce z repertoáru OpenCV pro nastavení a získání hodnoty určeného bodu. V mém případě se bude jednat o získání hodnoty odstínu šedi textelu.

```
void cvSetReal2D( CvArr* arr, int x, int y, double value );
```

`arr` – Ukazatel struktury s texturou.

`x, y` – Souřadnice bodu(počítáno od nuly).

`value` – Hodnota, kterou chci uložit.

```
double cvGetReal2D( const CvArr* arr, int x, int y );
```

`arr` – Ukazatel struktury s texturou.

`x, y` – Souřadnice bodu(počítáno od nuly).

Funkce vrací hodnotu požadovaného bodu.

5.3 Procedurální generátory

Je-li vyřešený způsob ukládání textury a způsob zápisu a četní jednotlivých textelů, můžu přistoupit k jejich generování.

5.3.1 Generátor obdélníkového základu

Generátor pracuje tak, že na základě rozměrů nejdříve vypočte počet zobrazených párů (dvojice černého a bílého obdélníku). Provede zaokrouhlení, aby vzorek vždy končil bílým obdélníkem a nestalo se, že v rámci zaokrouhlování bude zobrazen pouze černý obdélník z páru.

Další krok je vypočtení útlumu šedi pro každý následující obdélník, čehož je použito pokud generátor má obdélníky zesvětlovat. První obdélník bude mít vždy hodnotu $0,0$.

Nakonec se spustí samotné generování, které kontroluje šířku černého obdélníku a pokud je nastaveno zužování upravuje parametr cyklu. Při každém dokončeném páru je navýšena hodnota odstínu o hodnotu, kterou funkce předem vypočítala.

Deklarace funkce

```
void createRectangleBase(IplImage *sample, int pocet, int zuzeni , int stinovani);
```

`sample` – Ukazatel na existující strukturu s již alokovanou pamětí a nastavenými rozměry.

`pocet` – Počet obdélníků v generované textuře.

`zuzeni` – Určuje zúžení pokud je obsaženo více obdélníků.

`stinovani` – Hodnota 0 vypíná změnu odstínů generovaných obdélníků.

5.3.2 Generátor sinusového a kosinusového základu

Tento generátor pro svoji funkci požaduje výběr matematické funkce a zadání počtu půlperiod, které se mají ve výsledné textuře objevit.

Deklarace funkce

```
void createSineBase(IplImage *sample, int pocet, int unused , int funkce);
```

`sample` – Ukazatel na existující strukturu s již alokovanou pamětí a nastavenými rozměry.

`pocet` – Půlperiod obsažených ve vzorku.

`unused` – Nevyužito.

`funkce` – 0 – sinusový základ, 1 – kosinusový základ

5.4 Funkce pro úpravu vzorků

Aplikace má dovolit úpravy textur pomocí několika jednoduchých procedur, které jsou následující.

expandPlus

Protože je textura uchovávána v datovém typu float a použijeme pro reprezentaci hodnoty $0,0$ (černá) -$1,0$ (bílá), v případě součtu dvou bílých bodů získáme hodnotu $2,0$ což je chyba, která nám neudává korektní barvu. Jednoduchou úpravou však výsledek můžu normalizovat do požadovaného rozsahu a rovnice tedy bude vypadat.

$$vzorek_{vystupni}[x,y] = \frac{1}{2}\big(vzorek_{original}[x,y] + vzorek_{otoceny}[x,y]\big)$$
$$x = \{0,1,\dots M\}, y = \{0,1,\dots N\}$$

kde M a N jsou rozměry vzorku

Deklarace funkce

```
void expandPlus(IplImage *sample);
```

`sample` - Ukazatel na existující strukturu s texturou.

expandMinus

Opět musím ošetřit aby výsledná struktura uchovávající texturu obsahovala pouze hodnoty $0,0$-$1,0$. Tento problém řeší posunutí výsledku přičtením hodnoty $1,0$ a následné vydělení viz následující rovnice.

$$vzorek_{vystupni}[x,y] = \frac{1}{2}\Big(1 + \big(vzorek_{original}[x,y] - vzorek_{otoceny}[x,y]\big)\Big)$$
$$x = \{0,1,\dots M\}, y = \{0,1,\dots N\}$$

kde M a N jsou rozměry vzorku

Deklarace funkce

```
void expandMinus(IplImage *sample);
```

`sample` - Ukazatel na existující strukturu s texturou.

expandMax

Při implementaci jsou pouze porovnány body, které leží na stejné pozici a do výsledku je zahrnut bod s vyšší hodnotou. Po použití se tedy textura zesvětlí.

Deklarace funkce

```
void expandMax(IplImage *sample);
```

`sample` – Ukazatel na existující strukturu s texturou.

expandMin

Jako opačná funkce k funkci `expandMax` tato funkce vybírá do výsledku bod s nižší hodnotou. V praxi tedy dochází ke ztmavnutí textury.

Deklarace funkce

```
void expandMin(IplImage *sample);
```

`sample` – Ukazatel na existující strukturu s texturou.

invertSample

Funkce pomocí dvou vnořených cyklů projde vzorek od krajních sloupců ke středovým, přičemž body těchto sloupců prohazuje.

Deklarace funkce

```
void invertSample(IplImage *sample);
```

`sample` – Ukazatel na existující strukturu s texturou.

gaussianFilter

Zde jsem implementoval jednoduchý Gaussův filtr, který používá filtrační matici, jaká je vyobrazena na obrázku 3.4. Abych vyřešil problém s okrajovými body, je jejich výpočet proveden samostatně. poté dochází k průchodu zbytku obrazu.

Výpočet okrajových hodnot bodů je řešen tak, že chybějící body jsou zrcadleny.

Deklarace funkce

```
void gaussianFilter(IplImage *sample);
```

`sample` – Ukazatel na existující strukturu s texturou.

5.5 Generování textury pomocí úpravy spektra

Nejzajímavější výsledky vykazuje implementace generování textur, které je založené na úpravě spektra.

Abych toho docílí napsal jsem generátor bílého šumu založený na [2], který vyplní zadanou strukturu `IplImage`.

```
void noiseSample(IplImage *sample);
```

`sample` – Ukazatel na existující strukturu do které generujeme šum.

Usnadnění jsem nalezl v podobě funkce z knihovny OpenCV

```
void cvDCT( const CvArr* zdroj, CvArr* cil, int priznaky );
```

`zdroj` – Ukazatel struktury s texturou.

`cil` – Ukazatel, kam se má výsledek uložit. Musí mít stejnou velikost a typ.

`priznaky` – pro mě důležité parametry jsou následující:

- `CV_DXT_FORWARD` – vypočítá dopřednou transformaci
- `CV_DXT_INVERSE` – vypočítá inverzní transformaci.

Při aplikaci je tedy vygenerována šumová textura o rozměrech, které odpovídají vzorku, který budu upravovat. Poté na oba vzorky (vzorek textury a vzorek šumu) použiji funkci `cvDCT` který vrátí jejich spektra. Tyto spektra poté mezi sebou vynásobím. A výsledek opět předám funkci `cvDCT`, která se správným parametrem provede inverzní transformaci a tím získám požadovaný výsledek.

5.6 Normalizace textury po inverzní kosinové transformaci

Změna textury v oblasti spektra může přinést při zpětné transformaci úskalí v podobě posunu hodnot pro vyjádření barvy (v mém případě vyjádření odstínů šedi). Proto je třeba zpětně získaný vzorek normalizovat.

Normalizace je v aplikaci provedena jako lineární konverze hodnot do rozsahu $0,0$-$1,0$. Jako první krok je zjištěna největší a nejmenší hodnota nacházející se ve vzorku. Jako druhý krok je celý vzorek posunut tak aby nejmenší hodnota splynula s nulou. Nakonec je celý vzorek bod po bodu vynásoben konstantou $\frac{1}{2}(Maximum - minimum)$.

Výhoda této normalizace spočívá v tom, že se ve výsledné textuře viditelně zobrazí všechny odstíny. Pokud bych však normalizaci nepoužil, výsledek sice bude správný, avšak při zobrazení, kdy uvažuji vyjádření barev v rozsahu $0,0$-$1,0$, může výsledek vypadat téměř černý nebo naopak velmi světlý.Prakticky vzato po této normalizaci bude výsledná textura obsahovat vždy alespoň jeden bílý $(1,0)$ a jeden černý $(0,0)$ textel.

5.7 Rozhraní

Aplikace má sloužit k experimentování a tedy i ovládání by mělo být jednoduché, dovolit rychlé ovládání a do jisté míry i podávat informaci o postupech, které uživatel při generování textur použil.

Samotné umístění ovládacích prvků je možné jednoduše určit pomocí nástroje pro tvorbu formulářů a dialogových oken (builder, form-builder). Tento nástroj je obsažen

například ve vývojovém prostředí Visual Studio či Borland C++ Builder. Při implementaci vlastní aplikaci jsem použil Visual Studio.

Kód pro trasování tvorby textur

Tento kód je vytvořen pro možnost zaznamenání postupu při tvorbě vzorů pomocí procedurálních generátorů a dalších pomocných operací pro jejich úpravu.

Kód je zaznamenáván tak aby, po ukončení generování či dosažení zajímavého výsledku, mohl uživatel z rozhraní zpětně přečíst použité funkce, kterými k výslednému vzoru (textuře) dospěl.

- $+$ – Použití součtové úpravy vzorku
- $-$ – Použití rozdílové úpravy vzorku
- $\wedge$ – Použití úpravy vzorku použitím maxima
- $\vee$ – Použití úpravy vzorku použitím minima
- I – Zrcadlení textury podle středu přes vertikální osu
- G – Použití Gaussovského filtru
- C – Vzorek byl zkopírován[3]

Příklad:

CI$\vee$I$\vee$ – Vzorek je zkopírovaný (nebyl použit generátor) a byli na něj uplatněny operace převrácení, rozšíření minimem, opětovné převrácení a rozšíření minimem.

$-$I$\wedge$I$\wedge$ – Prvotní vzorek je generovaný, informaci o parametrech lze vyčíst ze zadaných hodnot. Poté na vzorek bylo použito rozšíření rozdílem, převrácení, rozšíření maximem, převrácení a opět rozšíření maximem.

5.8 Další možnosti aplikace

Opětovné použití výsledku

Abych zachoval myšlenku na obrázku 4.5 je do rozhraní, přidáno tlačítko, které texturu vygenerovanou mísením textury se šumem, překopíruje do části, ve které s ní můžeme provádět stejné úpravy jako s výchozími vzorky. To nám automaticky začne tvořit nový trasovací kód, který vždy začíná znakem C (více o významu v 5.7).

Vypnutí DCT

Další možnost je vypnutí diskrétní kosinové transformace na vygenerovaném vzorku před jeho vynásobením se šumem. Prakticky to znamená, že vygenerovaný vzor bude tvořit filtr. Prakticky tato možnost není ke generování příliš vhodná, ale při použití se dá dosáhnout také zajímavých výsledků.

[3] Vždy se nachází na začátku řetězce.

Ukládání výsledků

Aplikaci jsem napsal tak, že pokaždé uloží do souborů (`vzorek1.bmp, vzorek2.bmp, vysledek.bmp`, které se nachází v kořenové složce aplikace) aktuálně zobrazený obsah.

Kapitola 6

Experimenty

Tato kapitola především obsahuje ukázky vzorků, které je aplikace schopná vygenerovat.

6.1 Konvence označení vzorků

Pro co nejjednodušší označení vzorků, bude u obrázků uveden kód, který bude udávat použité operace v souladu s předpisem popsaným v odstavci 5.7. Uvozen však bude informací, která určuje ručně zadané parametry (počet obdélníků, počet půlperiod, atd.).

6.2 Přehled získaných vzorků pomocí programu

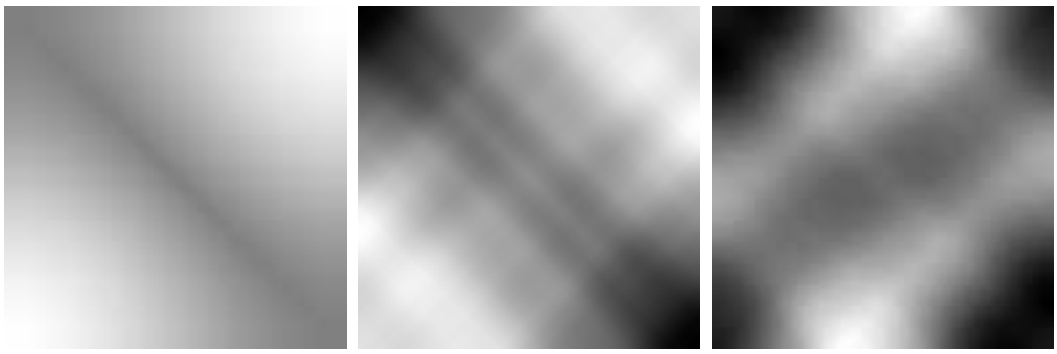
Zde jsou uvedeny vzorky textur, které aplikace vygenerovala. Nebude-li popis udávat jinak, jsou získané trojice textur uvedeny v pořadí:

Vlevo – vzorek získaný pomocí procedurálního generátoru (v popisku je uveden kód)

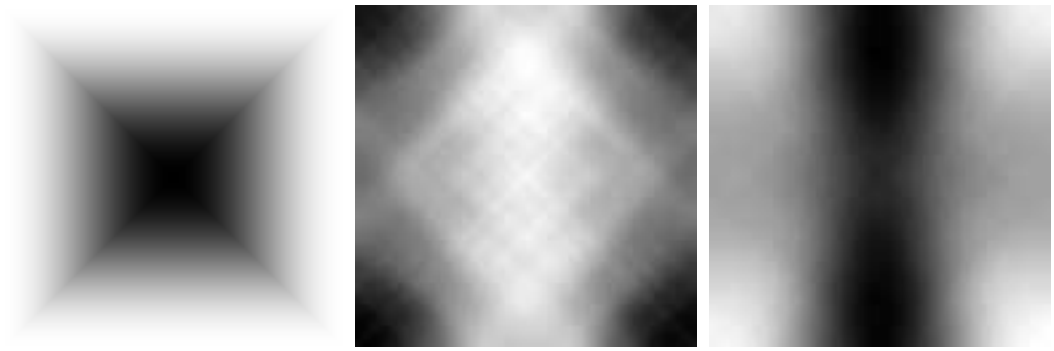
Uprostřed – vzorek získaný pomocí mísení vzorku vlevo se šumem v oblasti spektra

Vpravo – druhý vzorek získaný pomocí mísení vzorku vlevo se šumem v oblasti spektra

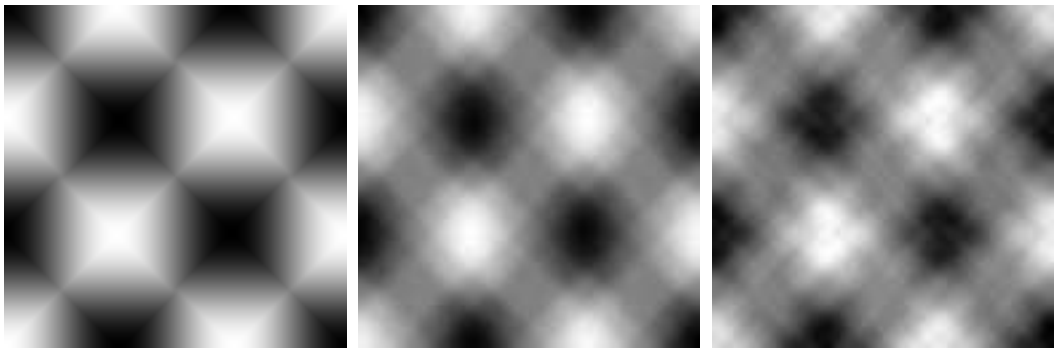
6.2.1 Vzorky získané na základě použití funkce `createSineBase`



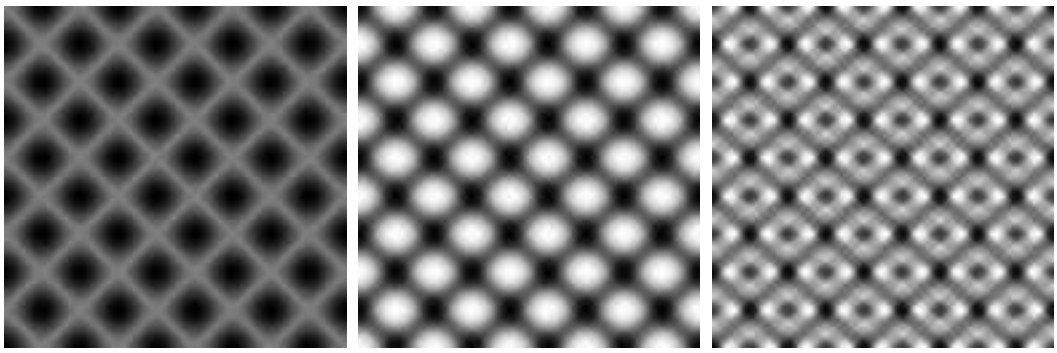
Použitá funkce – kosinus – 1 perioda. Kód `+I∧`.



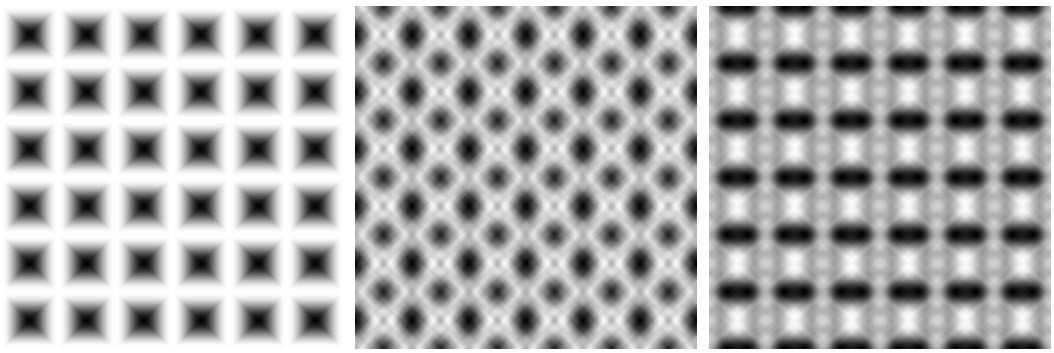
Použitá funkce – kosinus – 2 periody. Kód ∧.



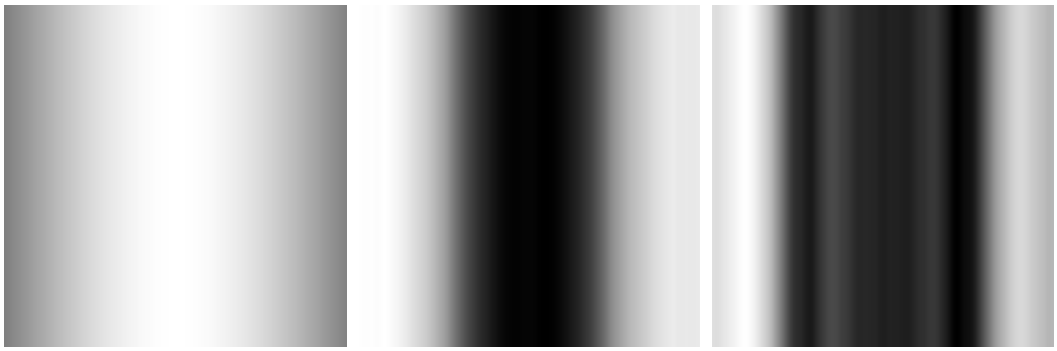
Použitá funkce – kosinus – 3 periody. Kód ∨I∧.



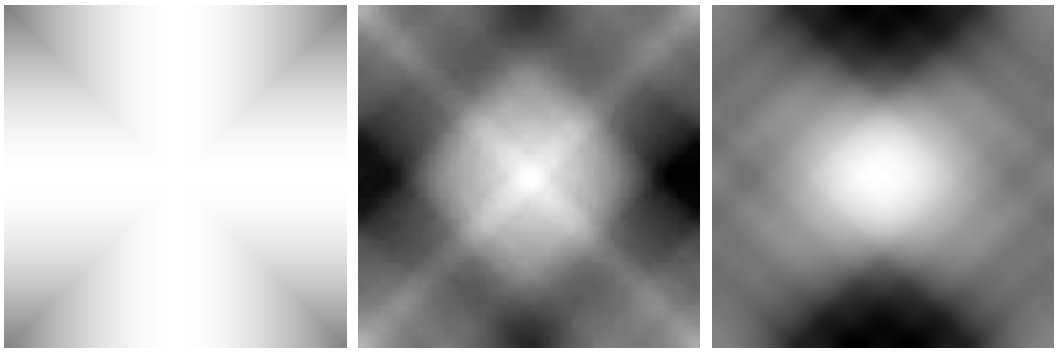
Použitá funkce – kosinus – 9 period. Kód -I∧-∨.



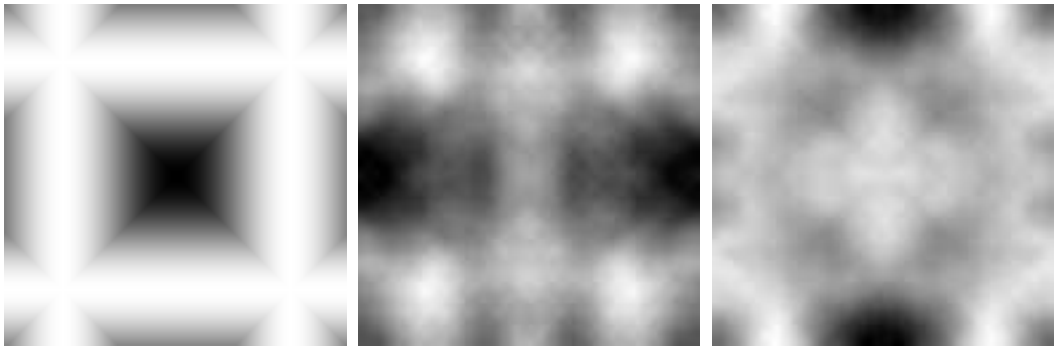
Použitá funkce – kosinus – 12 period. Kód ∧I∧.



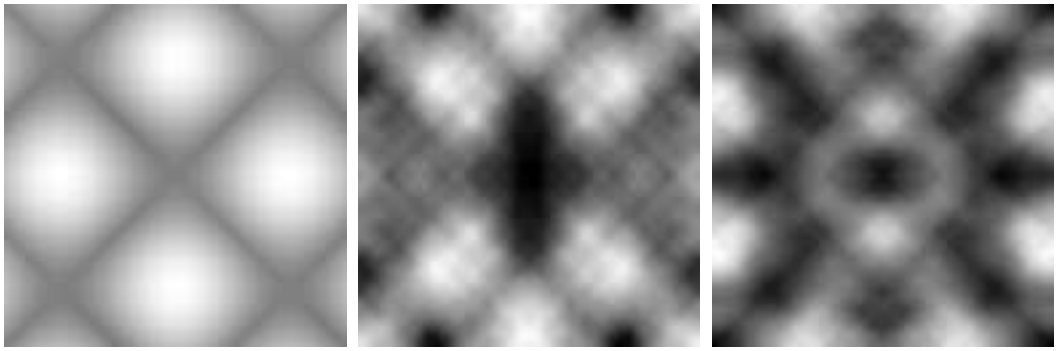
Použitá funkce – sinus – 1 perioda. Bez dalších úprav.



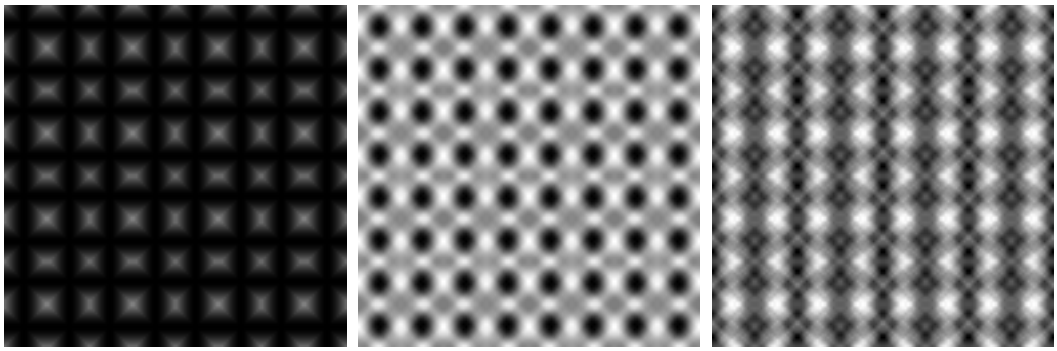
Použitá funkce – sinus – 1 perioda. Kód ∧.



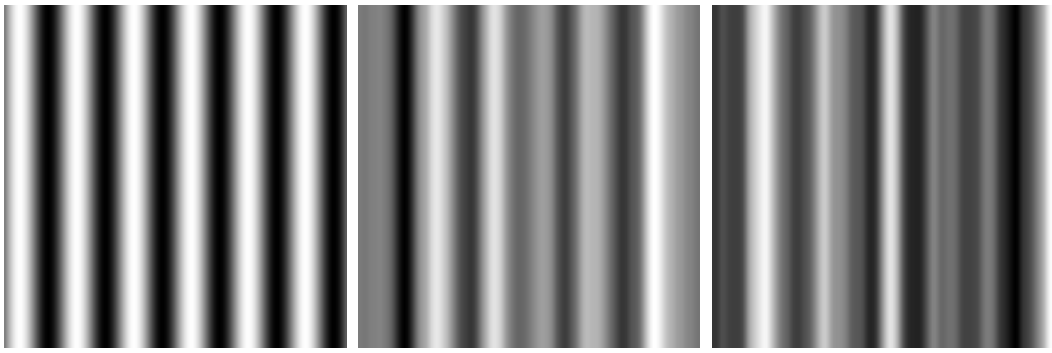
Použitá funkce – sinus – 3 periody. Kód ∧.



Použitá funkce – sinus – 3 perioda. Kód `-I`∧.

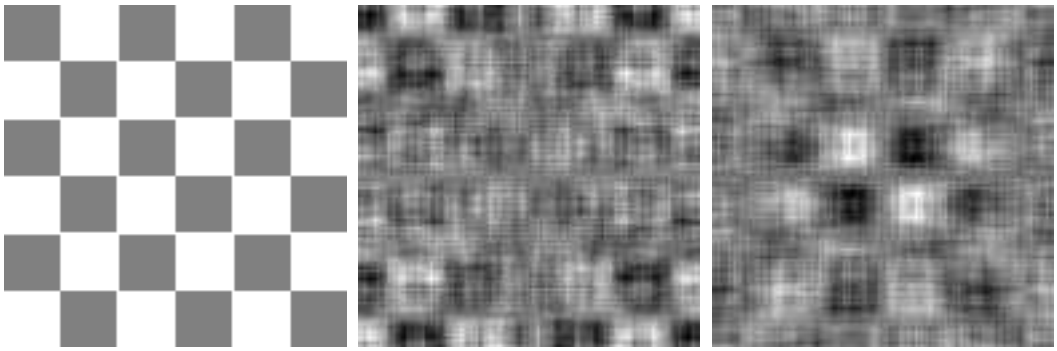


Použitá funkce – sinus – 8 period. Kód `∨I∨I`.

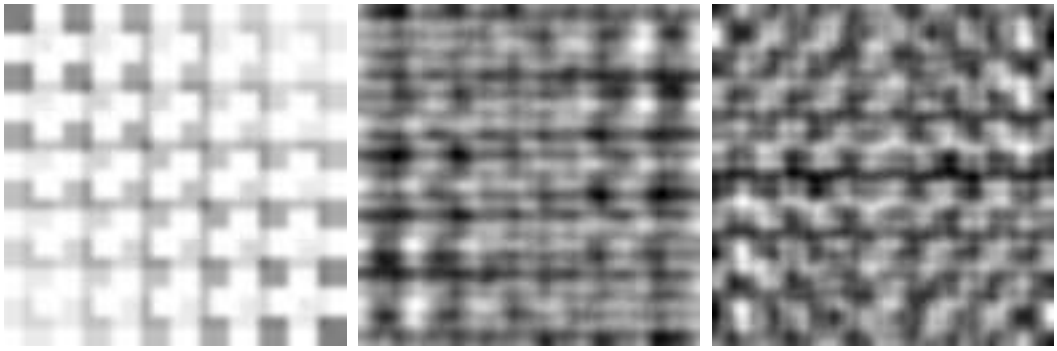


Použitá funkce – sinus – 12 period. Bez dalších úprav.

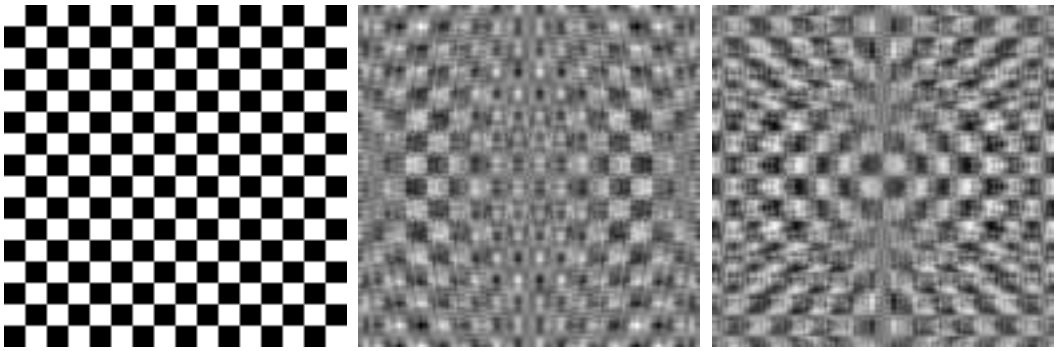
6.2.2 Vzorky získané na základě použití funkce `createRectangleBase`



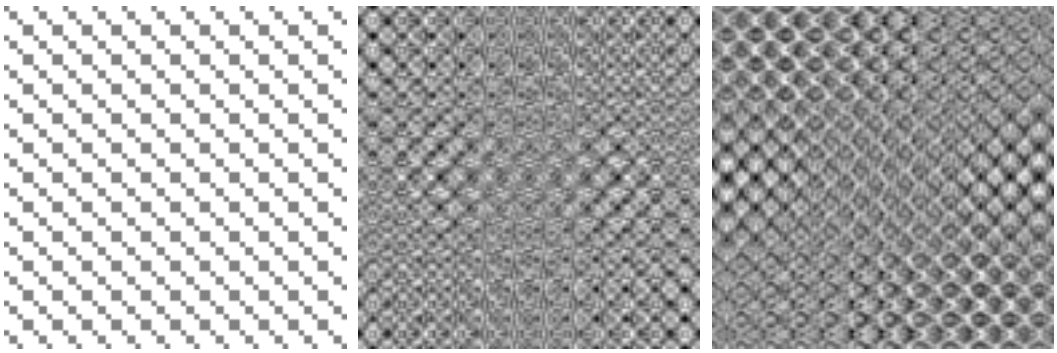
Počet obdélníků – 4. Bez zesvětlení. Kód `+I∧`.



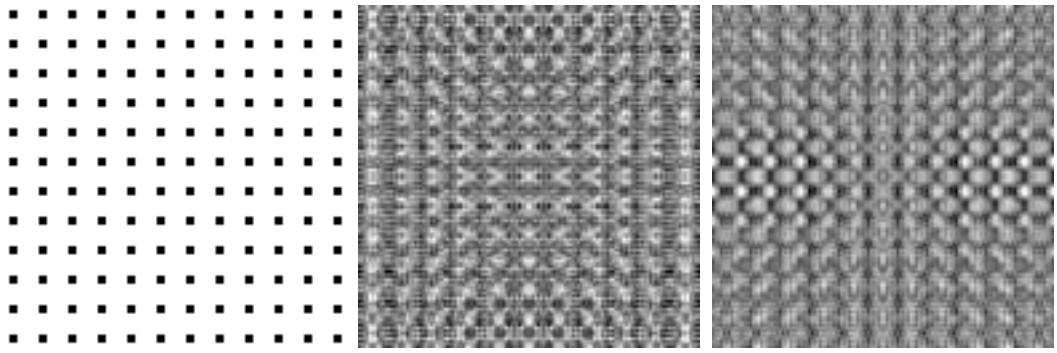
Počet obdélníků – 6. Se zesvětlením. Kód `+I∧GGG`.



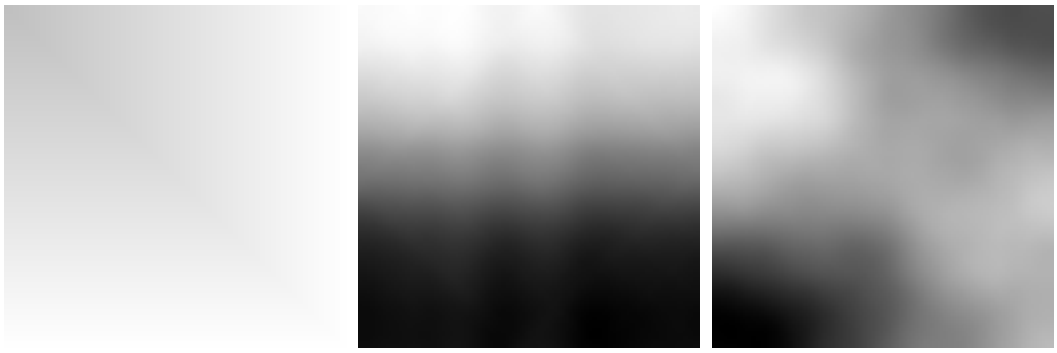
Počet obdélníků – 8. Bez zesvětlení. Kód ∧I∨.



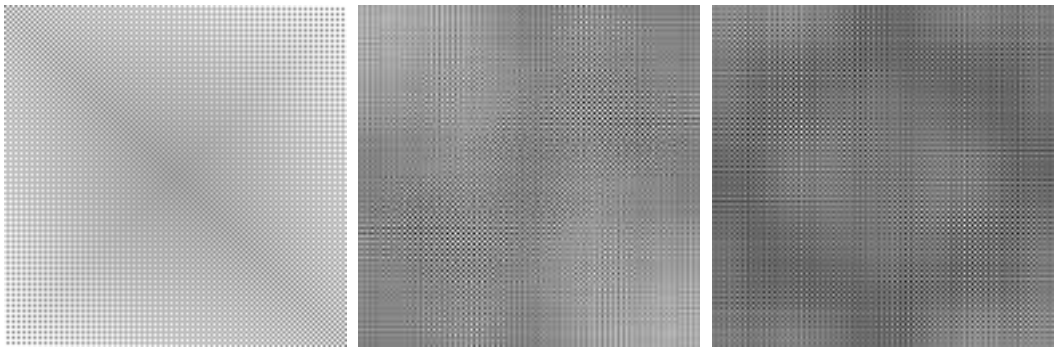
Počet obdélníků – 12. Bez zesvětlení. Kód -I∧I∧.



Počet obdélníků – 12. Bez zesvětlení. Kód I∧I∧.



Počet obdélníků – 64. Se zesvětlením. Kód ∧G.



Počet obdélníků – 64. Se zesvětlením. Kód `-I∧I∧`.

6.3 Pozorování

Vidíme, že vzorek po smísení se šumem (prostřední a pravý obrázek) vykazuje specifické vlasnosti.

Zajímavé je poukázat na fakt, že každý vzorek, který je tvořen sudým počtem půlperiod (obsahuje celé sinusovky a kosinusovky), je vzájemně návazný. To znamená, že je možné ho umisťovat vedle sebe aniž by byly viditelné přechody na rozhraních (švy).

Kapitola 7

Závěr

Při zpracování této práce jsem prostudoval techniky použitelné pro generování textur (např. procedurální generátory, generátory šumu) a jejich zpracování (např. filtrování, konvoluce).

Výsledkem mojí práce je aplikace, která zahrnuje jednoduché funkce vycházející z informací v kapitole 3. Aplikace umožní generovat vzorky textur pomocí grafického uživatelského rozhraní a výsledky zobrazí. Textury jsou v odstínech šedi a lze je dále náhodně upravovat díky použití mísení se šumem v oblasti spektra.

Jak vyplývá z experimentů (viz kapitola 6), aplikace generuje velmi rozmanité vzory. Proto jako další rozšíření aplikace může být generování textur v barvách a přidání množství dalších operací pro úpravu.

Literatura

[1] Barvy šumu. [online], [cit. 2008-5-11].
URL http://cs.wikipedia.org/wiki/Barvy_%C5%A1umu

[2] C++ gaussian noise generation. [online], [cit. 2008-4-11].
URL http://www.musicdsp.org/showone.php?id=168

[3] Convolution theorem. [online], [cit. 2008-5-11].
URL http://en.wikipedia.org/wiki/Convolution_theorem

[4] Detekce hran. [online], [cit. 2008-5-11].
URL http://cs.wikipedia.org/wiki/Detekce_hran

[5] Diskrétní kosinová transformace. [online], [cit. 2008-5-11].
URL http:
//cs.wikipedia.org/wiki/Diskr%C3%A9tn%C3%AD_kosinov%C3%A1_transformace

[6] Gaussian blur. [online], [cit. 2008-5-11].
URL http://en.wikipedia.org/wiki/Gaussian_blur

[7] Grafická knihovna OpenGL – Texturování, 2005, materiály k přednáškám, UPGM FIT VUT v Brně.

[8] Konvoluce. [online], [cit. 2008-5-11].
URL http://cs.wikipedia.org/wiki/Konvoluce

[9] Kučera, O.: *Knihovna pro výpočet šumů používaných v procedurálním texturování.* Diplomová práce, FIT VUT v Brně, Brno, 2007.

[10] Lai, C.-C.: Texture Synthesis for Data Visualization Using Spot Noise. 1994.

[11] Procedurální texturování, 2005, materiály k přednáškám, UPGM FIT VUT v Brně.

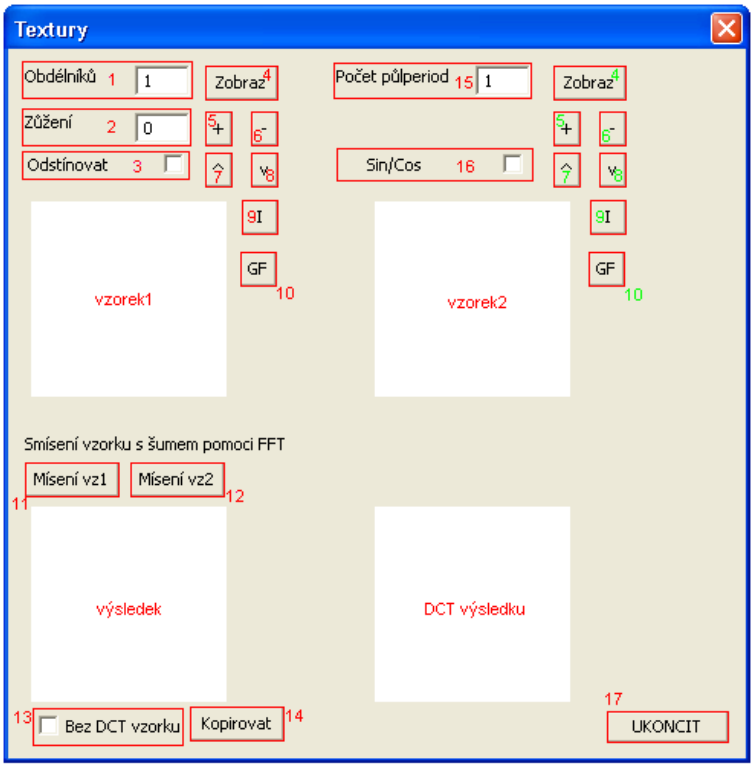
[12] Turk, G.: Texture Synthesis on Surfaces. In *Siggraph 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, 2001, s. 347–354.

[13] Wei, L.-Y.; Levoy, M.: Fast Texture Synthesis Using Tree-Structured Vector Quantization. In *Siggraph 2000, Computer Graphics Proceedings*, editace K. Akeley, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000, s. 479–488.
URL http://graphics.stanford.edu/papers/texture-synthesis-sig00/

[14] White noise. [online], [cit. 2008-5-11].
URL http://en.wikipedia.org/wiki/White_noise

Dodatek A

Popis prvků uživatelského rozhraní



1 – pole pro zadání počtu obdélníků

2 – zužování generovaných obdélníků podle zadané hodnoty

3 – zapnutí/vypnutí změny stupňů šedi u generovaných obdélníků

4 – vygenerování textury podle hodnot v polích (1,2,3) a jeho zobrazení do plochy `vzorek1`

5 – použití funkce `expandPlus` na obsah plochy `vzorek1`

6 – použití funkce `expandMinus` na obsah plochy `vzorek1`

7 – použití funkce `expandMaximu` na obsah plochy `vzorek1`

8 – použití funkce `expandMinimum` na obsah plochy `vzorek1`

9 – otočení vzorku zobrazeném na ploše `vzorek1` pomocí `invertSample`

10 – použití Gaussova filtru na obsah plochy `vzorek1`

11 – mísení obsahu plochy `vzorek1` se šumem a zobrazení výsledku do `výsledek`

12 – mísení obsahu plochy `vzorek2` se šumem a zobrazení výsledku do `výsledek`

13 – vypíná aplikování DCT na obsah plochy `vzorek1` nebo `vzorek2` před mísením se šumem

14 – zkopíruje obsah `výsledek` do plochy `vzorek1` pro možnost dalších experimentů

15 – pole pro zadání počtu půlperiod

16 – použití funkce sinu (nezatrženo) nebo kosinu při generování textury

17 – ukončí aplikaci

zelená čísla – mají shodné funkce jako čísla červená, jen pracují s sin./kosin. generátorem a výsledky operací ukládají do plochy `vzorek2`

Dodatek B

Obsah přiloženého CD/DVD

Dokumentace

- `doc/xhavel25.pdf` – soubor obsahující dokumentaci
- `doc/bp_tex.zip` – zdrojové kódy pro LaTeX s obrázky

Zdrojové kódy aplikace

- `app/src` – adresář obsahuje projekt aplikace pro Visual Studio 2005
- `app/bin/textury.exe` – spustitelný soubor aplikace (vyžaduje nainstalovanou knihovnu OpenCV)

Knihovny

- `lib` – adresář obsahuje knihovny použité v aplikaci