Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Syntéza textur

Texture synthesis

2014 **Václav Smutný**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Václav Smutný**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Syntéza textur
Texture Synthesis

Zásady pro vypracování:

Cílem práce je vytvoření nástroje pro generování textur na základě jednodušších snímků různých povrchů. Aplikace by měla uživateli umožňovat měnit potřebné parametry syntézy a generovat náhledy výstupní textury.

1. Seznamte se s metodami analýzy a modelování rozměrných textur z reálných snímků, zaměřte se převážně na metody založené na pravděpodobnostních modelech (náhodná pole apod.) a statistických přístupech k analýze dat.
2. Vybranou metodu detailně popište a naimplementujte.
3. Výsledek otestujte na sadě dodaných snímků nanesením výsledné textury na vhodný model.

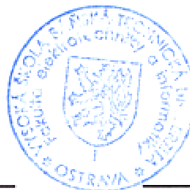
Seznam doporučené odborné literatury:

[1] Paget, Longstaff. Texture synthesis via a noncausal nonparametric multiscale Markov random field. IEEE Trans. on Image Processing. 1998.
[2] Efros, Freeman. Image Quilting. SIGGRAPH. 2001.
[3] Wei, Li-Yi. Texture Synthesis by Fixed Neighborhood Searching. PhD Thesis. 2001.
[4] http://en.wikipedia.org/wiki/Texture_synthesis.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Fabián**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *30. dubna 2014*

podpis studenta

Poděkování

Velice rád bych poděkoval Ing. Tomáši Fabiánovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

// Abstrakt

Má bakalářská práce se zabývá syntézou textur. Zpočátku popisuji různé přístupy, jaké typy algoritmů syntézy textur se používaly nebo používají a jaké výhody a nevýhody poskytují. Pro implementaci jsme vybrali algoritmus syntézy textur vytvořený skupinou Microsoft Research s názvem Parallel Controllable Texture Synthesis[1], který implementuji v jazyce C++ za pomocí knihovny OpenCV. Tento algoritmus následně do detailů popisuji a vysvětluji, na jakém principu funguje a jak zásadně mohou vstupní hodnoty uživatele ovlivňovat výstupní textury vytvořené pomocí tohoto algoritmu. V kapitole čtvrté vyhodnocuji uměle vytvořené textury, dobu zpracování, kvalitu a jaké nastavení vstupních hodnot je nejlepší použít. V poslední kapitole lehce nastiňuji, čeho by chtěli vývojáři v oblasti syntézy textur ještě v budoucnosti dosáhnout.

// Klíčová slova

Hugues Hoppe, chvění, korekce, Microsoft, Microsoft Research, OpenCV, Parallel Controllable Texture Synthesis, pod-průchod, průchod, Sylvain Lefebvre, syntéza textur, textura souřadnic, tiling, zvětšování

Abstract

My bachelor thesis deals with texture synthesis. At first, I'm describing differences between different approaches and what is good and bad about them. For my implementation and testing purposes we chose one texture synthesis algorithm invented by Microsoft Research group, called Parallel Controllable Texture Synthesis [1]. I'm explaining and describing in detail how is this algorithm working and how the user is capable of affecting the output texture by the input values for example by adding more randomness etcetera. In the fourth chapter I analyze artificial texture, processing time and what user input values are the best to use. In the last chapter I lightly outline what the developers would like to achieve in the future with texture synthesis algorithms.

Key words

Correction, Hugues Hoppe, jitter, Microsoft, Microsoft Research, OpenCV, Parallel Controllable Texture Synthesis, pass, subpass, Sylvain Lefebvre, texture of coordinates, texture synthesis, tiling, upsampling

Seznam použitých symbolů a zkratek

1. 2D – Dva rozměry
2. 3D – Tři rozměry
3. C++ – Programovací jazyk
4. CPU – Centrální výpočetní jednotka
5. GPU – Grafická výpočetní jednotka
6. CHAR – Datový typ
7. OpenCV – Knihovna funkcí pro práci s obrázky
8. UCHAR – Datový typ

Obsah

1 Úvod ... - 10 -
2 Teoretická část ... - 11 -
2.1 Textura ... - 11 -
2.2 Syntéza textur ... - 12 -
2.3 Základní metody Syntézy textur ... - 13 -
2.3.1 Tiling (spojování) ... - 13 -
2.3.2 Stochastická syntéza textur ... - 14 -
2.3.3 Jednoúčelová strukturovaná syntéza textur ... - 14 -
2.3.4 Chaotická mozaika ... - 14 -
2.3.5 Záplatově založena syntéza textur ... - 15 -
2.3.6 Pixelově založena syntéza textur ... - 15 -
3 Detailní popis algoritmu - Parallel controllable texture synthesis ... - 16 -
3.1 Uživatelské ovládání ... - 16 -
3.2 Základní schéma ... - 17 -
3.2.1 Důvod proč pracovat se souřadnicemi ... - 18 -
3.3 Princip algoritmu ... - 19 -
3.3.1 Zvětšování (upsampling) ... - 21 -
3.3.2 Chvění (Jitter) ... - 22 -
3.3.3 Korekce - oprava ... - 23 -
3.3.4 Korekční průchod pixely ... - 25 -
4 Výsledky a vyhodnocení ... - 26 -
4.1 Počet korekčních kroků ... - 26 -
4.2 Počet pod-průchodů ... - 27 -
4.3 Velikost porovnávaného okolí ... - 28 -
4.4 Hodnoty pro změnu náhodnosti textury ... - 30 -
4.5 Paměťová a časová náročnost ... - 31 -
4.6 Vstupní a výstupní textury ... - 32 -
5 Budoucí práce ... - 33 -
5.1 Vylepšení kvality ... - 33 -

5.2 Syntéza terénu ... - 33 -
5.3 Menší nároky na paměť: ... - 33 -
6 Závěr ... - 34 -
Použitá literatura ... - 35 -
Seznam příloh ... I

1 Úvod

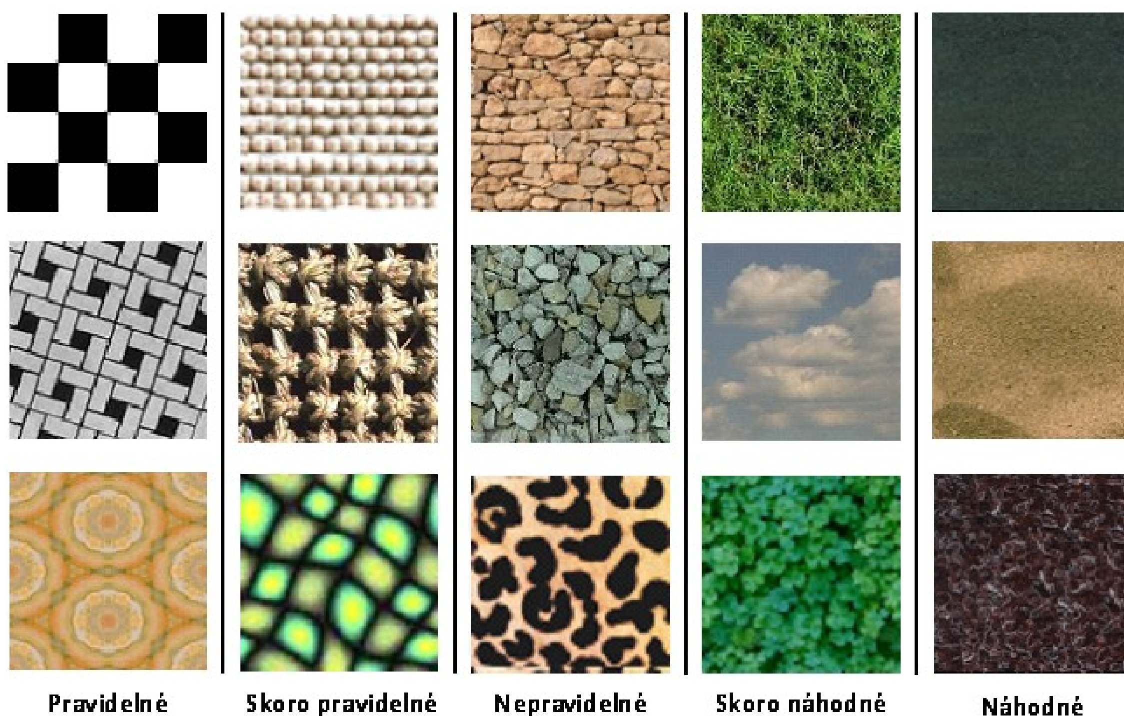
Pod názvem syntéza textur si lze představit algoritmus, podle kterého se uměle vytváří nové textury s co nejvyšší podobností textuře vstupní. V dnešní době tyto algoritmy dosahují poměrně vysoce kvalitních výsledků i zpracování v realném čase za pomocí paralelního zpracování na GPU. Syntéza textur řeší například jak zabránit okatému kopírování částí vstupní textury nebo odstranění viditelných hran, které vznikají po zvyšování rozlišení textury. Různé metody a jejich řešení popisuji v kapitole druhé. Pro implementaci a testovací účely jsme vybrali algoritmus vytvořený dvojicí Sylvain Lefebvre a Hugues Hoppe ze skupiny Microsoft Research s názvem Parallel Controllable Texture Synthesis[1], který implementuji v programovacím jazyce C++ za pomocí knihovny OpenCV. V třetí kapitole se věnuji tomuto vybranému algoritmu syntézy textur a detailně popisuji, na jakém principu tento algoritmus funguje. Po implementaci vybrané metody syntézy textur vyhodnocuji v kapitole čtvrté vygenerované výsledky a popisuji, jaké vstupní parametry jsou nejvíce optimální pro nastavení tohoto algoritmu. Nakonec v kapitole páté lehce nastiňuji budoucí práci, čeho by chtěli vývojáři syntézy textur ještě v budoucnu dosáhnout.

2 Teoretická část

V této kapitole se zabývám teoretickými základy k tématu syntéza textur. Ze začátku charakterizuji význam slova textura a jaké různé typy textur mohou být. Dále popisuji využití syntézy textur a obecné požadavky na tento algoritmus. Na konci této kapitoly jsou popsány algoritmy syntézy textur od nejjednodušších po složitější včetně jejich výhod a nevýhod.

2.1 Textura

Svět je plný textur, povrch každého viditelného objektu je v určitém smyslu textura. Textury jsou pozorovatelné jak na umělém tak přírodním objektu, jako třeba dřevo, rostliny, materiál, kůže apod. Obecně slovo textura odkazuje na charakteristický povrch a vzhled objektu o určité velikosti, tvaru, uspořádání a poměru jeho základních částí. Vzhledem k různorodosti textur ve světě je obtížná reprodukce podle společného vzoru[8].



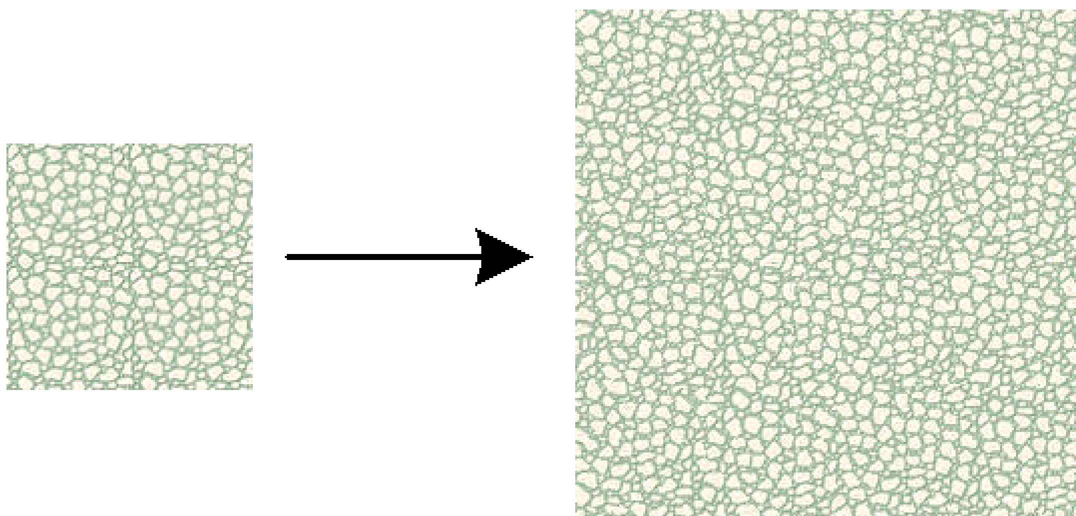
Obrázek 2.1: *Příklady různých druhů textur od pravidelných až po náhodné*

2.2 Syntéza textur

Syntéza textur má celou řadu aplikací v oblasti počítačové grafiky a zpracování obrazu. Textura je většinou snímek pocházející z fotografie. Dostupné fotografie nebo části fotografií mohou být příliš malé na překrytí celého povrchu objektu. V takovém případě, jednoduché spojování snímků způsobí nepřijatelné vzorky ve formách viditelných opakovaní a švů. Syntéza textur řeší tento problém generováním volitelných velikostí textur. Syntéza textur se dá také aplikovat ve zpracování obrazu jako například komprese obrázku nebo videa.

Algoritmus syntézy textur má vytvořit texturu, která splňuje tyto požadavky [5]:

- Výstupní textura má velikost zadanou uživatelem
- Výstupní textura by měla být co nejvíce podobná vstupní textuře
- Výstupní textura by neměla mít viditelné nenavazující hrany, bloky, vzory
- Výstupní textura by neměla opakovat stejnou strukturu na více místech



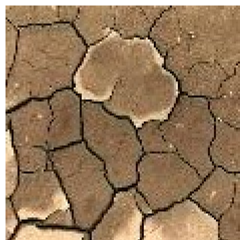
Obrázek 2.2: *Uživatel vloží vstupní texturu do programu, poté si zvolí velikost výstupní textury a spustí syntézu textur. Algoritmus uměle vytvoří novou výstupní texturu, která vypadá co nejvíce podobně jako textura vstupní*

2.3 Základní metody Syntézy textur

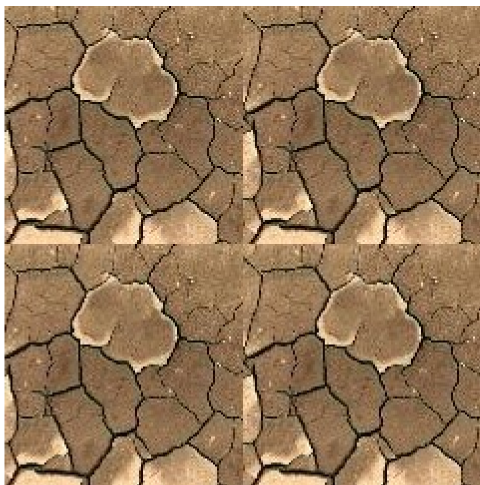
V této podkapitole se zabývám různými metodami syntézy textur od těch nejjednodušších po složitější a pro každou metodu vypisuji jejich výhody a nevýhody. Některé složitější metody dosahují i zpracování v reálném čase za pomocí GPU.

2.3.1 Tiling (spojování)

Tiling je nejjednodušší metoda jak zvýšit rozlišení textury, která generuje textury ze vzorků/bloků a staví je vedle sebe. To znamená, že více kopii čtvercových bloků jsou vkládány za sebe blok po bloku. Občas tuto metodu využívají jiné algoritmy syntézy textur na zvyšování rozlišení textury. Výsledky této metody jsou jen vzácně vyhovující. Ve většině případů budou vidět nesouvislosti mezi bloky. Výhodou této metody je vysoká rychlost algoritmu, ale nevýhodou jsou nekvalitní výsledky, kde jdou vidět nesouvisející hrany a opakující se bloky zobrazeny na obrázku 2.4.



Obrázek 2.3: *Vstupní textura vložená uživatelem*



Obrázek 2.4: *Výstupní textura při použití algoritmu tiling s viditelnými hranami*

2.3.2 Stochastická syntéza textur

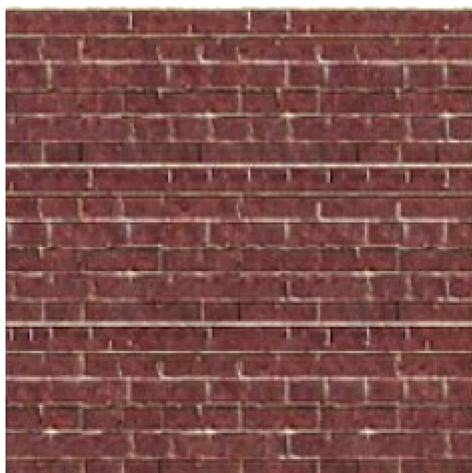
Stochastická metoda syntézy textur produkuje textury na principu náhodně vybrané barevné složky pro každý pixel, ovlivněny jsou jen základními parametry jako minimální jas, průměrná barva nebo maximální kontrast. Jedinou výhodou těchto algoritmů je, že fungují kvalitně se stochastickými texturami. Jinak produkují velice neuspokojivé výsledky a také ignorují jakékoliv struktury v rámci vybrané textury [5].

2.3.3 Jednoúčelová strukturovaná syntéza textur

Takto založené algoritmy používají pevně stanovené procedury na vytvoření výstupní textury, například jsou limitovány na jen jeden druh struktury. Tudíž tyto algoritmy mohou být aplikovány jen na textury s velmi podobnou strukturou. Jednoúčelový algoritmus by mohl například vytvořit vysoce kvalitní texturu kamenné zdi, avšak tento algoritmus by s vysokou pravděpodobností vytvořil velice nekvalitní výstupní texturu, kdyby vstupní vzor zobrazoval například oblázky. Výhodou těchto algoritmů jsou vysoce kvalitní výstupy pro jeden určitý druh textury. Avšak nevýhodou je, že algoritmus funguje kvalitně jen pro jeden druh struktury a pro každou jinou strukturu je potřeba použít nebo vyvinout algoritmus nový [5].

2.3.4 Chaotická mozaika

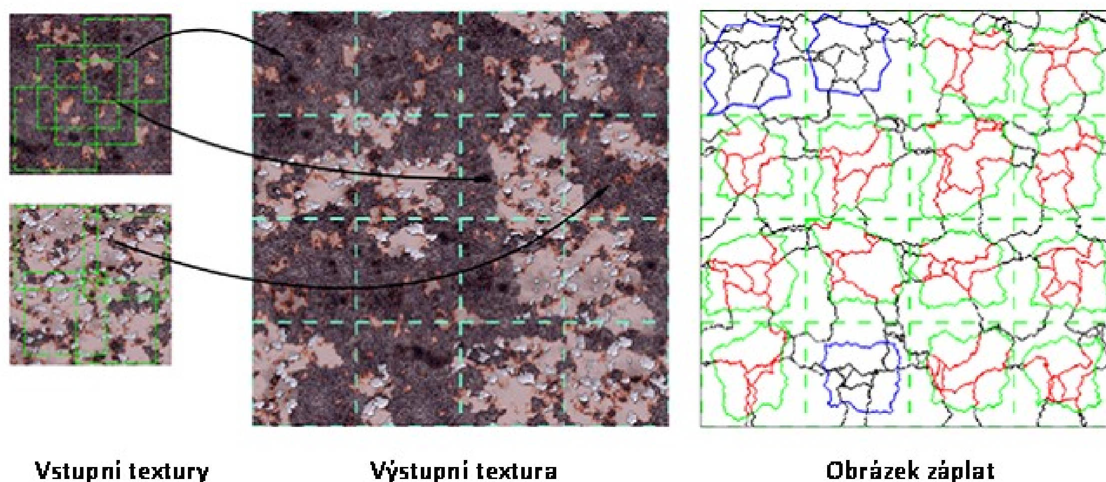
Tato metoda je navržená skupinou od společnosti Microsoft, která se zabývala internetovou grafikou. Algoritmus je jemnější verzí metody tiling a provádí 3 kroky. Výstupní textura je kompletně zaplněna metodou tiling. Nyní nově vygenerovaná textura obsahuje opakující se vstupní texturu s viditelnými švy. Poté náhodně nakopíruje různě velké bloky ze vstupní textury na texturu výstupní. Jako výsledek je spíše neopakující se textura s viditelnými švy. Výstupní textura je poté vyfiltrována, aby byly odstraněny hrany. Konečný výsledek je do určité míry akceptovatelná textura. Výhodou je vygenerování textury bez okatého opakování vstupní textury, ale kvůli filtrovacímu kroku je tato metoda stále neuspokojivá, protože dělá výstupní obrázek příliš rozmazaný. Detaily metody získány z oficiální dokumentace [4].



Obrázek 2.5: *Uměle vytvořená textura pomocí algoritmu chaotická mozaika, která je kvůli nekvalitnímu filtrovacímu kroku příliš rozmazaná. Obrázek použit z dokumentu [4]*

2.3.5 Záplatově založena syntéza textur

Záplatově založena syntéza textur je vzorkovací algoritmus používající záplaty tj. malá 2D část ze vstupní textury. V každém kroku algoritmus vloží záplatu ze vstupní textury do uměle vytvořené výstupní textury. Abychom se vyhnuli neshodujícím se hranicím dvou bloků, algoritmus vybírá záplaty na základě co nejmenšího rozdílu mezi hranami záplat. Záplatově založená syntéza textur produkuje vysoce kvalitní textury pro velkou rozmanitost textur od opakujících se až po náhodné. Některé algoritmy jsou natolik rychlé, že dokážou v reálném čase vyprodukovat novou texturu i na osobním počítači. Čerpáno z [6],[5].



Obrázek 2.6: *Na obrázku vidíme dvojici vstupních textur, ze kterých se skládá po vybraných záplatách výstupní textura. Červeně jsou vyznačeny bloky s viditelnými hranami, které potřebujeme schovat bloky vyznačené zelenou barvou. Odmítnuté bloky jsou označeny modrou barvou, protože stále produkovali viditelné švy. Obrázek použit z dokumentu [6]*

2.3.6 Pixelově založena syntéza textur

Pixelově založené algoritmy jsou jedny z nejvíce úspěšných algoritmů syntézy textur. Typicky vytváří novou texturu na principu naleznutí nejbližšího okolí pro každý pixel, ale naleznutí nejbližšího okolí může být časově velice náročná úloha. Proto se často předopočítává několik kandidátů pro každý pixel, než prohledávání celé vstupní textury. Výhodou pixelově založených algoritmů je velice kvalitní doplňování chybějících částí textury. Nevýhodou těchto metod je nedostačující napodobení určitých struktur, pokud použijeme malé okolí pro porovnávání [2].

3 Detailní popis algoritmu - Parallel controllable texture synthesis

Metoda vytvořená dvojicí Sylvain Lafebvre a Hugues Hoppe ze skupiny Microsoft Research. Odkaz na oficiální dokumentaci [1] odkud jsem čerpal všechna data k implementaci, detaily, pseudokódy a definice této metody. Pomocí tohoto algoritmu lze generovat aperiodické textury s deterministickou náhodností, kde náhodnosti dosahuje pomocí víceúrovňového zvětšování a rozhazování bloků textury. Tato metoda dosahuje i zpracování v reálném čase, ale jen pomocí výpočtu na GPU. Mé zpracování využívá více jádrové CPU.

Tato metoda by se dala zařadit malou částí mezi „Patch-Based" záplatově orientované a větší částí mezi „pixel-based" pixelově orientované algoritmy syntézy textur. Algoritmus si zakládá na víceúrovňovém rozhazování a posouváním částí textury, kde pro každou úroveň používáme jinou velikost rámce, čím získáváme obrovskou náhodnost na výstupní textuře. V každé úrovni zvětšujeme texturu dvakrát až do potřebné velkosti, počet úrovní se liší dle vstupních hodnot. Zvětšování funguje na bázi použití textury z minulé úrovně a specificky pro každý pixel vytvoří novou čtveřici pixelů. Tedy pro každý rodičovský pixel je vytvořena čtveřice potomků čímž získáme dvakrát větší texturu pro další úroveň. Posledním potřebným krokem je krok korekční, zásadní inovace je provádění korekce na každém pixelu nezávisle. To povolí korekci pixelů v libovolném pořadí. Korekce je provedena na bázi porovnávání dvou okolí, první je z rozhozené textury, která potřebuje opravit a druhé okolí pochází z textury vstupní, podle které opravujeme rozhozenou texturu.

3.1 Uživatelské ovládání

Mnoho algoritmů syntézy textur nabízí jen omezené řízení uživatelem, poskytují tedy jen malou kontrolu nad množstvím proměnlivosti textury. Typicky hledají stejné okolí, pro které lze jednoduše navázat další záplatu. Díky tomu se zmenšuje náhodnost, které mohou tyto algoritmy dosáhnout. Jedna zásadní změna tohoto algoritmu je náhodné posunování bloků textury závislých na vstupních parametrech, které mění výstupní texturu spíše nepředvídatelně. Tato metoda přináší přístup pro více explicitní a intuitivní ovládání. Klíčovým principem je ovládání funkce chvění, která pouze naruší souřadnice na každé úrovni pyramidy.

3.2 Základní schéma

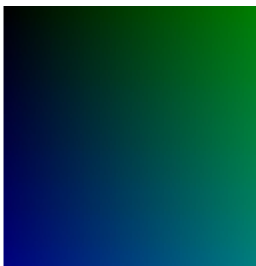
Jako první vytvoříme texturu souřadnic ze vstupní textury. Texturu souřadnic vytvoříme tak, že pro každý pixel ze vstupní textury vezme pozici, na které se pixel nachází a tuto pozici zapíšeme do pixelu v textuře souřadnic místo barevné složky, na tutéž pozici.

Definice:

Ze vstupní textury E o velikosti $m \times m$ syntéza textur vytvoří novou texturu S, ve které každý pixel $S[p]$ uloží souřadnice u pixelu ze vstupní textury E (kde $p, u \in \mathbb{Z}^2$). Tedy barva pixelu p je dána

$$E[u] = E\big[S[p]\big]. \tag{3.1}$$

Obrázek 3.1: *Vstupní textura E je textura nejčastěji malého rozlišení, kterou vkládá uživatel pro vygenerování textury většího rozměru*



Obrázek 3.2: *Na obrázku lze vidět vygenerovaná textura souřadnic za pomocí vstupní textury, kde barva pixelu určuje pozici, na které najdeme reálný pixel ve vstupní textuře*

Příklad generace textury souřadnic:

Pro pixel p na pozici $(1,1)$ s barevnou složkou (r, g, b) ze vstupní textury se vytvoří pixel u_{11}, který bude na stejné pozici $(1,1)$. Místo barvy bude mít souřadnice pod, kterými najdeme tento vybraný pixel p a barevná složka pixelu u_{11} bude jen dvoumístná $(1,1)$. Pro pixel ze vstupní textury na pozici $(1,2)$ vytvoříme pixel u_{12}, který uložíme na pozici $(1,2)$ s barvou $(1,2)$. Stejně pokračujeme pro každý pixel ve vstupní textuře.

3.2.1 Důvod proč pracovat se souřadnicemi

Pomocí této transformace dosáhneme toho, že můžeme použít malé proměnné např. 8bitové uchar, které jsou dostačující pro vstupní textury do velikosti 255×255 pixelů. Díky tomu vytvoříme texturu souřadnic, která má jen dvě složky například modrou a zelenou. Tímto můžeme dosáhnout mnohem rychlejšího zpracování a nižší náročnosti na paměť. Protože většina výpočtů probíhá nad těmito souřadnicemi krom korekčního kroku, ve kterém musíme porovnávat barvy pixelů ze vstupní textury. Z textury souřadnic můžeme zpět získat barevnou texturu.

3.2.1.1 ***Pracování jen se vstupní texturou***

Pro uložení barvy jednoho pixelu je použita proměnná typu char – 24bit

$$3 \times 24 = 3\,\text{složky} \times \text{alokace proměnné} = 72\text{bitů} \tag{3.2}$$

3.2.1.2 ***Pracování se souřadnicemi***

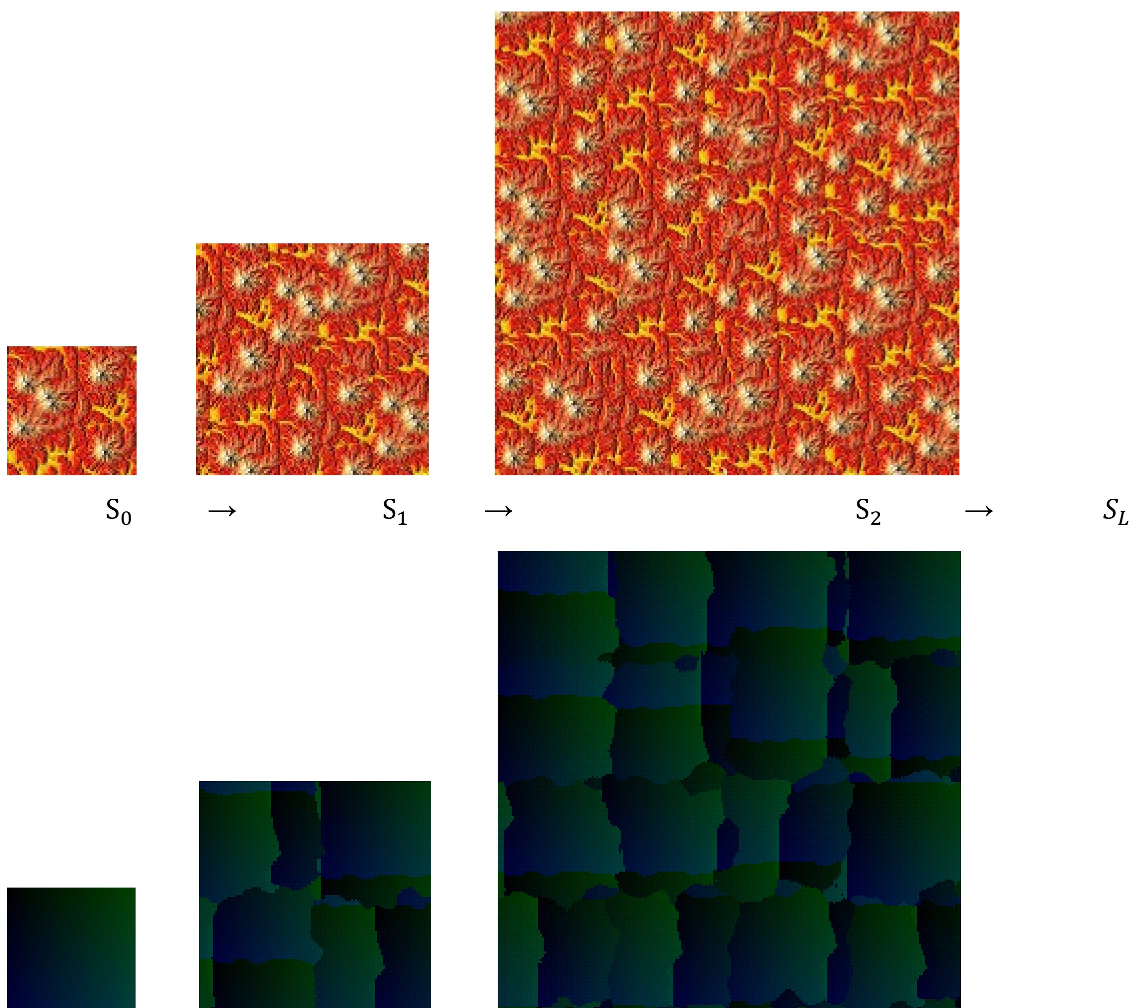
Pro uložení souřadnic jednoho pixelu do proměnné typu uchar – 8bit

$$2 \times 8 = 2\,\text{složky} \times \text{alokace proměnné} = 16\text{bitů} \tag{3.3}$$

Pomocí této transformace do textury souřadnic budeme zpracovávat až o 78% méně dat při práci s texturou souřadnic než při použití barevné textury.

3.3 Princip algoritmu

Syntéza pracuje s texturou souřadnic, která je vytvořena z textury vstupní. Jak se vytváří a k čemu tato textura souřadnic slouží je popsáno v kapitole 3.2. Syntéza zpracovává texturu v opakujících se cyklech, začíná se od nejnižší úrovně 1 až do úrovně n kde počet cyklů je závislý na velikosti výstupní textury. Na nejnižší úrovni zásadně ovlivňujeme, jak bude výstupní textura vypadat a zvyšováním úrovně ovlivňujeme jen menší a menší detaily textury.

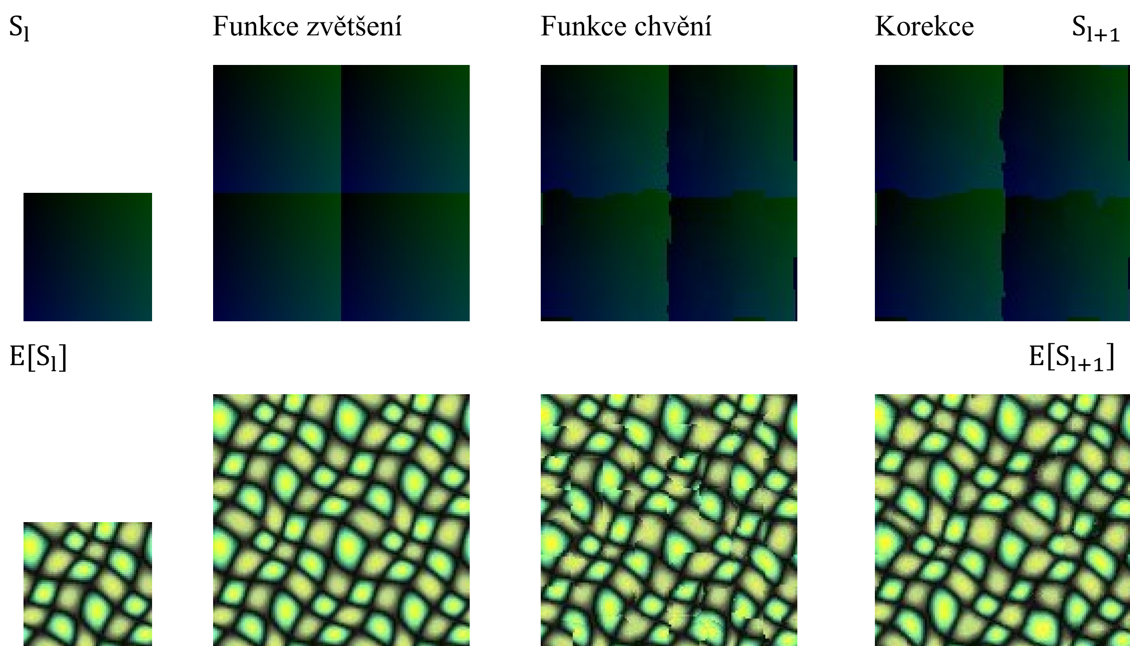


Obrázek 3.3: *aplikuje se tradiční hierarchická syntéza, z textury se vytváří jak kdyby pyramida $S_0, S_1, \dots, S_L$ (S_L = výstupní textura) v pořadí od hrubého k jemnému*

V každé úrovni jsou použity tři kroky. První krok je zvětšení textury dvakrát neboli anglicky upsamping, poté nastupuje chvění neboli anglicky jitter, kterým získáme náhodnost a nakonec aplikujeme korekci. Korekce opraví hrany a změny, které nastaly po rozhození textury pomocí funkce chvění. Všechny tři kroky jsou detailněji popsány dále v textu. Jeden průchod přes úroveň je zobrazen na obrázku 3.4.

```
Pseudokód syntézy textur

for l∈{0...L}                   // Víceúrovňový přístup
      Sl = Upsample(Sl-1)       // Použití funkce zvětšení
      Sl = Jitter(Sl)           // Použití funkce rozhození
      if (l>2)                  // Pro úrovně větší než 2
            for {1..c}          // Aplikace korekčních kroků
            Sl = Correct(Sl)    // Opravovaní rozhozené textury
return SL                       // Vrať výstupní texturu
```



Obrázek 3.4: *Pro každou úroveň pyramidy, se provádí tři kroky. Upsampling – zvětšování, jitter - chvění - funkce rozhození a nakonec korekce, která opraví hrany a různé nesrovnalosti*

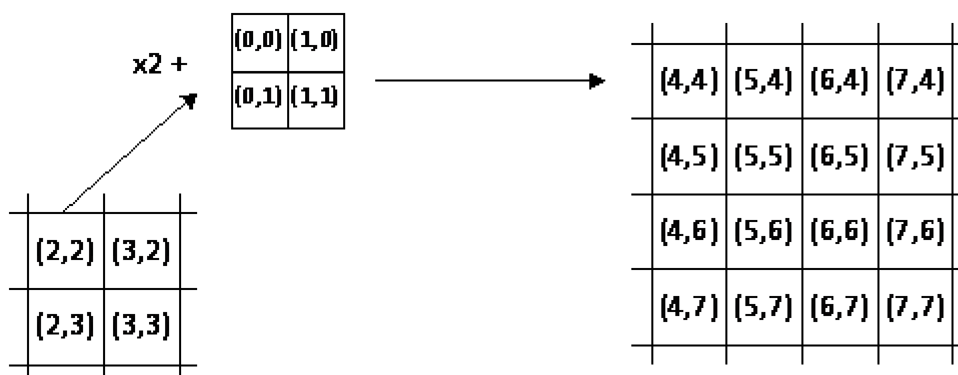
3.3.1 Zvětšování (upsampling)

Raději než používání samostatného průchodu syntézy pro vytvoření větší textury z předchozí hrubší úrovně jednoduše upravíme souřadnice rodičovských pixelů. Specificky přiřadíme ke každému pixelu čtyři potomky se zvětšenými rodičovskými souřadnicemi a kompenzaci závislou na potomkovi.

Definice:

$$S_l[2p + \Delta] := (2S_{l-1}[p] + h_l\Delta) \text{ mod m}, \qquad \Delta \epsilon \left\{\binom{0}{0}\binom{0}{1}\binom{1}{0}\binom{1}{1}\right\} \tag{3.4}$$

Jestli je vstupní textura prstencového/toroidního tvaru a funkce chvění je zakázána, postupné zvětšování textury vytvoří $S_l[p] = p$ mod m, které odpovídá spojení kopii vstupních textur neboli metoda tiling.



Obrázek 3.5: *Vlevo na obrázku vidíme souřadnice z předchozího kroku, které vynásobíme dvakrát a přičteme k nim kompenzaci pro potomky a tímto získáme dvakrát větší texturu. Souřadnice (x, y) odkazují na reálnou barvu ze vstupní textury*

3.3.2 Chvění (Jitter)

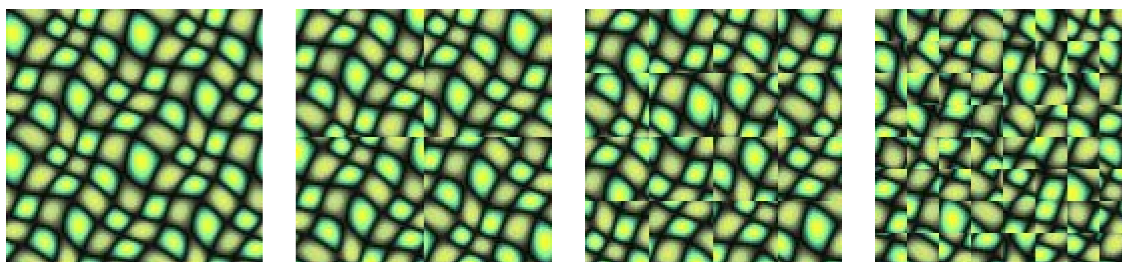
Jedná se o funkci, pomocí které přidáváme náhodnost do zpracovávané textury. V každé úrovni rozdělujeme texturu na bloky, které posuneme o vektor a pokud tento vektor není nulový tak posouváme celý blok o velikost vektoru. Texturu postupně pro každou úroveň rozdělujeme na menší a menší bloky.

Funkce rozděluje při každé úrovni texturu na čtyř-strom. Pro úroveň jedna rozsekne texturu na 4 části zobrazeny na obrázku číslo 3.6 a 3.7. Pro každou ze 4 částí se přičte k souřadnicím uloženým v textuře vektor, pomocí kterého se posunou všechny souřadnice v daném čtverci o stejnou hodnotu. Pro úroveň dvě rozsekneme všechny 4 čtverce na další 4 čtverce. Vznikne tedy nyní 16 čtverců, na které je znovu aplikován posun o vektor. Pro každou úroveň se liší počet čtverců na kolik je textura rozdělena, pro úroveň $3 = 64$, $4 = 256$ atd. Úroveň určuje, kolikrát byl aplikovaný čtyř-strom. Tato funkce pracuje jen s texturou souřadnic. Textura souřadnic je popsána v kapitole 3.2.

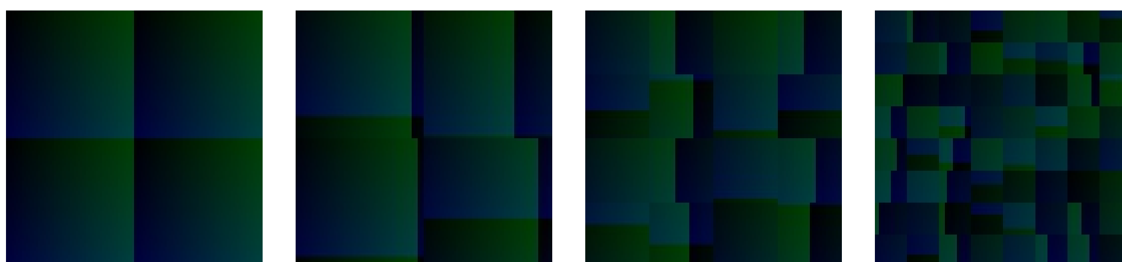
$$S_l[p] := \left(S_l[p] + J_{l(p)}\right) \bmod m, \text{ kde } J_l(p) = \left\lfloor h_l\, \mathcal{H}(p) r_l \binom{.5}{.5} \right\rfloor \qquad (3.5)$$

Výstupní koeficient h_l redukuje amplitudu chvění na jemnější úrovně, toto je obecně požadováno. Jestli bude korekční krok vypnut, chvění na každé úrovni vypadá jako čtyř-strom z překládaných bloků v konečné textuře.

Úroveň 0. 1. 2. 3.



Obrázek 3.6: *Na nejnižší úrovni zásadně ovlivňujeme texturu a postupem k vyšší úrovni měníme čím dále menší detaily textury. V tomto případě je použito extrémních hodnot posunu pro názornou ukázku*



Obrázek 3.7: *Zde jsou zobrazeny ovlivněné souřadnice funkcí chvění při daných úrovních. Textury pod sebou odpovídají stejné úrovni*

3.3.3 Korekce - oprava

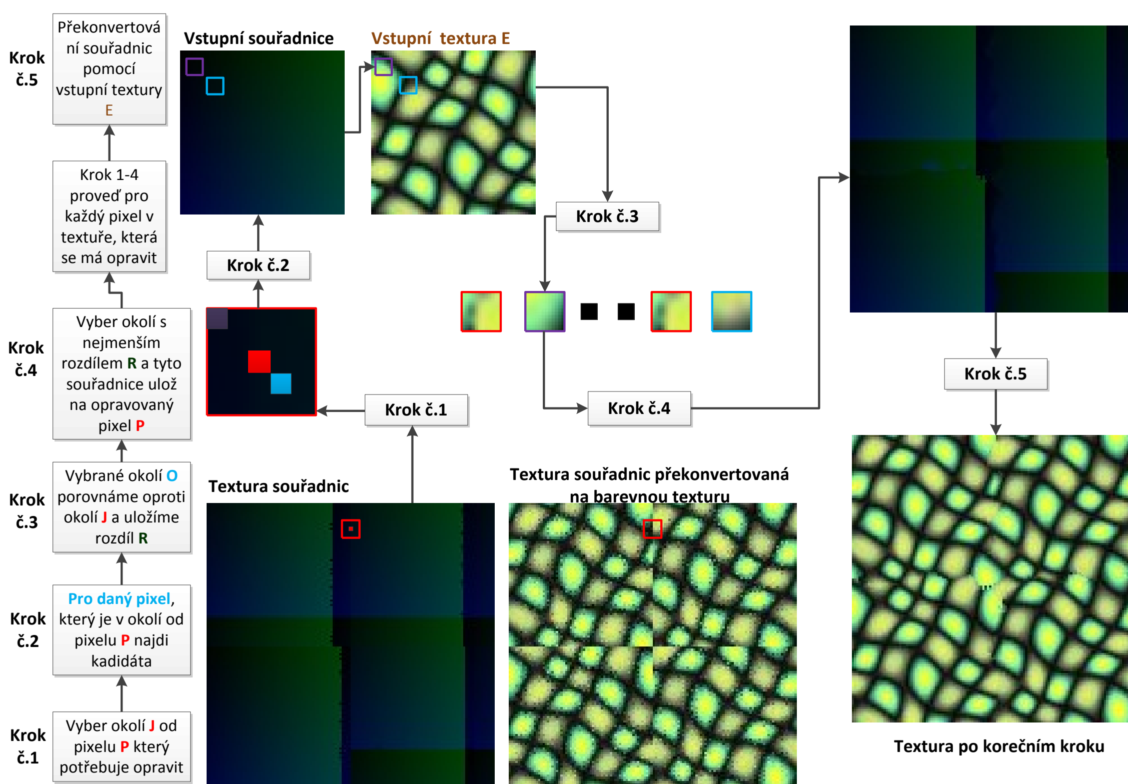
Korekce používá jako vstup texturu souřadnic z předchozího kroku spolu se vstupní texturou. Po předchozím kroku je nová textura v nekonzistentním tvaru a musí být přepracována k obrazu vstupní textury. Korekce je prováděna na každém pixelu nezávisle, to povolí korekci pixelů v libovolném pořadí. Korekce funguje na bázi porovnávání dvou okolí, jedno je z rozhozené textury z předchozího kroku, která potřebuje opravit a druhé ze vstupní textury, podle které opravujeme rozhozenou texturu. Korekce pro jeden pixel je názorně zobrazena na obrázku 3.8.

$$S_l[p] = C^l_{i_{min}}(S_l[p + \Delta_{min}] - h_l\Delta_{min}), \text{ kde} \tag{3.6}$$

$$i_{min}, \Delta_{min} = \underset{i\epsilon\{1..k\}}{\operatorname{argmin}} \left\| N_{S_l}(p) - N_{E_l}\left(C^l_i(S_l[p + \Delta] - h_l\Delta)\right)\right\| \varphi(i)$$

$$\Delta\epsilon\left\{\binom{-1}{-1}, \binom{-1}{-0}, \dots, \binom{1}{1}\right\}$$

$$\text{Ve, kterém } \varphi(i) = \left\{\begin{matrix} 1 & i = 1 \\ 1 + \kappa' & i > 1 \end{matrix}\right\}$$



Obrázek 3.8: *Zde je názorně zobrazena oprava jednoho pixelu. Kroky korekce jsou více popsány na další stránce. Textura po korekčním kroku ještě není konečný výsledek, protože postupuje do další úrovně. Tento obrázek s vyšším rozlišením je k naleznutí na DVD ve složce obrázky z dokumentu pod jménem CorrectionOverview.vsd*

3.3.3.1 ***Krok č. 1***

Celý proces je popsán jen pro jeden pixel, protože tento proces je stejný pro každý pixel v opravované textuře. Z textury souřadnic vybereme pixel P_i, který potřebuje opravit. Oprava funguje na bázi porovnávání okolí pixelů. Okolí pixelu P_i označíme jako okolí J_i. Pro okolí J_i projdeme všechny pixely jeden po jednom. Pro ukázku korekčního algoritmu vybereme světle modrý pixel M.

3.3.3.2 ***Krok č. 2***

Z pixelu M získáme souřadnice, které nás přepošlou na pozici kandidáta M_k. V našem případě používáme souřadnice vstupní textury. Textura souřadnic na pozici kandidáta M_k nás odkáže na pozici reálného pixelu ze vstupní textury. A tímto získáme návrh, zda pixel M_k může nahradit pixel P_i.

3.3.3.3 ***Krok č. 3***

Okolí kandidáta M_k porovnáváme oproti okolí J_i. Porovnání je provedeno jako rozdíl barevných složek dvou okolí. První okolí je z rozhozené textury, která potřebuje opravit a druhé je z textury vstupní, podle které novou texturu opravujeme.

```
Pseudokód porovnávacího algoritmu:
for x e {-1… 1}                           // pro okolí 3x3
      for y e {-1… 1}                     //
Vp = cvGet2D(input_image,Mky+y,Mkx+x)  // Okolí kandidáta
Jp = cvGet2D(input_image,Py+y,Px+x)    // Okolí textury po chvění
            Diff+= (Vp - Jp)2      // Celkový rozdíl dvou okolí
Return (Diff)                      // vrať rozdíl dvou celých okolí
```

3.3.3.4 ***Krok č. 4***

Projdeme všechny kandidáty na tuto pozici a vybereme ten, který měl nejméně lišící se okolí. Tento kandidát nahradí pixel, který je zrovna opravován.

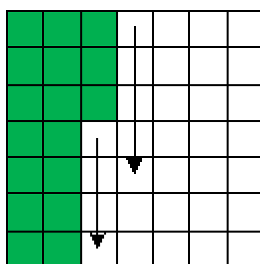
3.3.3.5 ***Krok č. 5***

Pomocí vstupní textury a textury souřadnic, zobrazíme nově vygenerovanou texturu. Tento algoritmus je víceúrovňový, tudíž po korekci se textura souřadnic pošle do další úrovně. Kde se provede funkce zvětšení, chvění a korekce znovu dokud algoritmus nedosáhne konečné úrovně.

3.3.4 Korekční průchod pixely

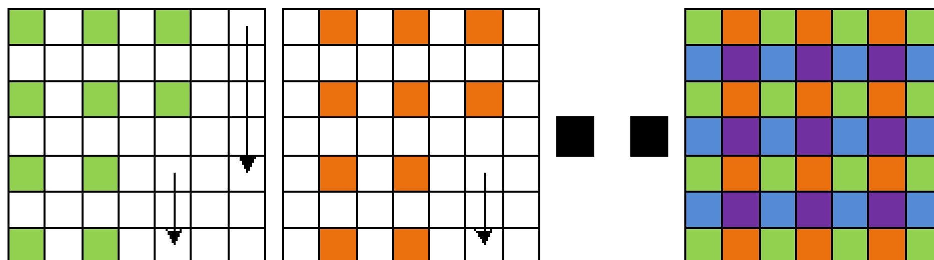
Když se opravují všechny pixely sekvenčně jeden za druhým, nastává problém. Pixely jsou opravovány vzhledem k jejich sousednímu okolí, ale to se mění. Toto může vést ke zpomalení konvergence nebo vést i k zacyklení. Řešením je rozdělení korekčního průchodu do postupných pod-průchodů nesousedících pixelů. Specificky, aplikujeme s^2 pod-průchodů. Existuje celá řada faktorů, které je nutno zvážit před výběrem počtu pod-průchodů a pořadí. Kvalita syntézy se vylepšuje s větším počtem pod-průchodů, ale ne větším než 9. Více o testování pod-průchodů je popsáno v kapitole 4.2 a obrázku 4.2. Pořadí zpracovávání pod-průchodů může být zobrazeno graficky jako matice s × s.

Pokud $s = 1$ získáme 1 pod-průchod, protože $s^2 = 1$



Obrázek 3.9: *Počet pod-průchodů je rovno jedné, tedy celá textura se projde sekvenčním přístupem*

Pokud $s = 2$ získáme 4 pod-průchody, protože $s^2 = 4$



Obrázek 3.10: *Pro $s = 2$ se průchod celou texturou rozdělí na 4 pod-průchody vyznačené různými barvami. První se provede celý pod-průchod zelenou barvou poté druhý pod-průchod oranžovou atd. Dohromady tyto čtyři pod-průchody tvoří průchod celou texturou. Vidíme, že pokud máme porovnávací okolí 3 × 3 pixely, tak zelené pixely jsou opraveny, aniž by zpracovávaly pixely, které se zrovna opravují korekčním krokem*

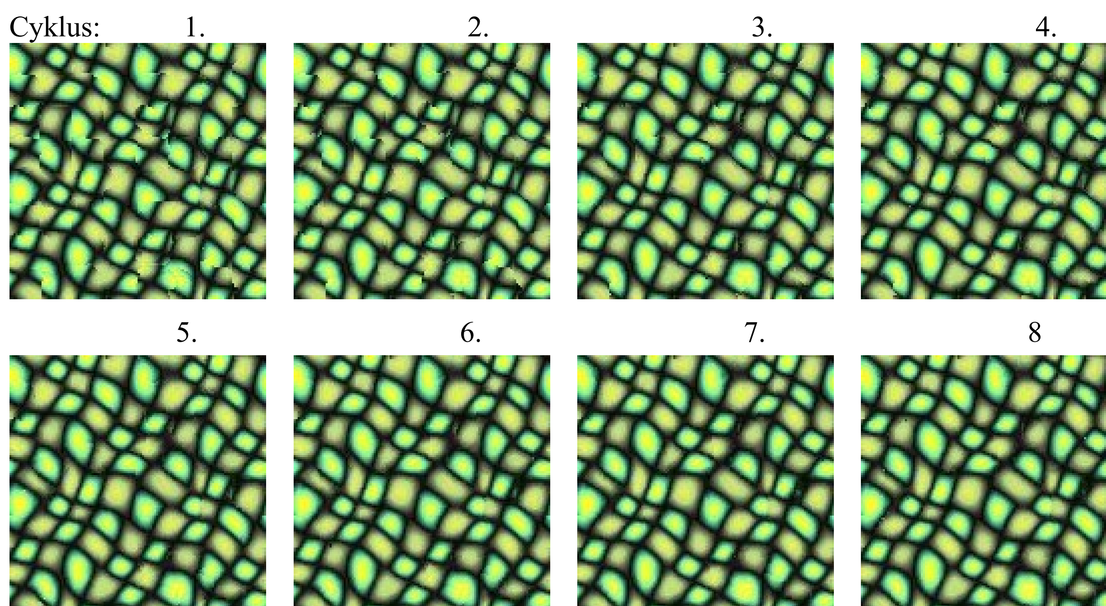
4 Výsledky a vyhodnocení

Tato kapitola se zabývá konečným vyhodnocením implementované metody syntézy textur. Názorně vysvětluji typy vstupních parametrů pro algoritmus Parallel Controlable Texture Synthesis. Pro každý vstupní parametr jsem provedl testování k naleznutí doporučené hodnoty postačující k vygenerování kvalitní výstupní textury pro většinu případů. Vstupní hodnoty jsou například počet korekčních kroků, velikost porovnávaného okolí atd. Všechny uměle vygenerované textury mou implementací jsou k naleznutí na DVD přiloženém k této bakalářské práci ve složce Výstupní textury. Pro aplikaci textur na plochu 3D objektu jsem použil software Blender a vyrenderované obrázky jsou k naleznutí na DVD ve složce Renderované obrázky.

4.1 Počet korekčních kroků

Počet korekčních kroků zásadně ovlivňuje, jak kvalitně bude textura s viditelnými hranami opravena. Korekční krok je potřeba aplikovat několikrát za sebou, protože korekce aplikovaná pouze jedenkrát nestihne opravit všechny chyby způsobené zavedením náhodnosti do textury. Podle testů je nejlepší aplikovat korekci pro kvalitní výstupy 2-5 krát. Tento algoritmus syntézy textur je více-úrovňový tedy pro každou úroveň se aplikuje stejný počet korekčních kroků. Proto není nutné, aby textura byla úplně do detailů opravena, protože ji opraví další korekční kroky.

Doporučený počet korekcí: 2-5

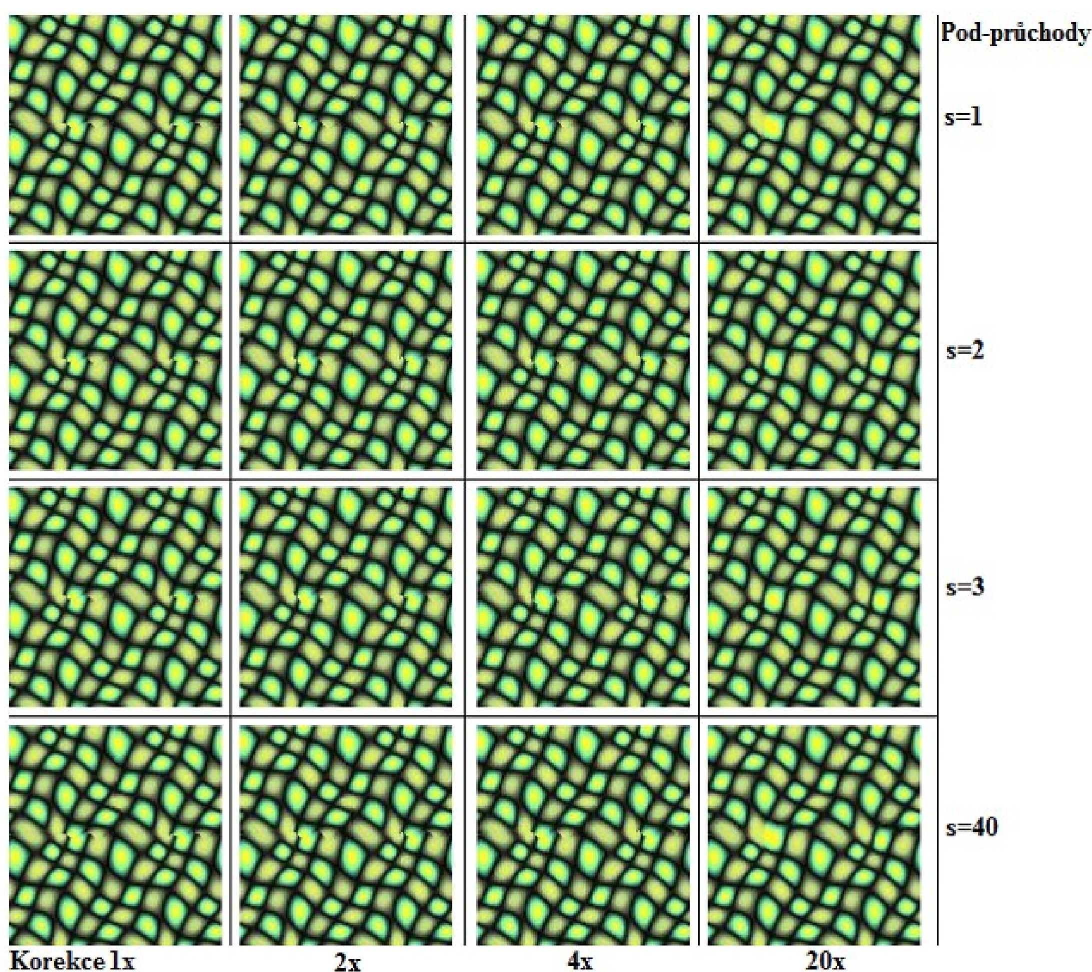


Obrázek 4.1: *Na obrázku lze vidět kolikrát je potřeba aplikovat korekční krok než zmizí všechny viditelné hrany. Jakmile je textura v opraveném stavu tak už korekční krok texturu téměř nemění. Tudíž je dobré nastavit správný počet korekčních kroků. Obvykle stačí korekce aplikovat 2-5krát. Pět krát aplikovaný korekční krok už je více než dostačující*

4.2 Počet pod-průchodů

Počet pod-průchodů určuje, do kolika kratších pod-průchodů se rozdělí jeden sekvenční korekční průchod každým pixelem v opravované textuře. Umocněním vstupní hodnoty na druhou získáme počet pod-průchodů. Tedy pokud je hodnota $s = 1$ jde o sekvenční přístup zobrazený na obrázku číslo 3.9. U sekvenčního přístupu nastává problém, že opravujeme pixely podle okolí, které se mění. Proto je doporučeno hodnotu $s = 1$ nepoužívat. Pro hodnotu $s = 2$ je počet pod-průchodů 4. Pomocí rozdělení na pod-průchody už budeme opravovat pixely dle okolí, které se nemění. Tudíž získáme lepší kvalitu korekce za stejnou dobu opravovaní. Doporučená hodnota na použití je $s = 2$ nebo $s = 3$. Hodnoty výrazně větší než 3 nedoporučuji používat. Tradiční sekvenční algoritmus je podobný velkému množství pod-průchodů aplikovaných v řadě. Zajímavé je, že velké množství pod-průchodů dosahuje horších výsledků. Vysvětlení je, že program se již nedostane tolikrát zpět, aby opravil předchozí chyby.

Doporučená hodnota: $s = 2$ nebo $s = 3$

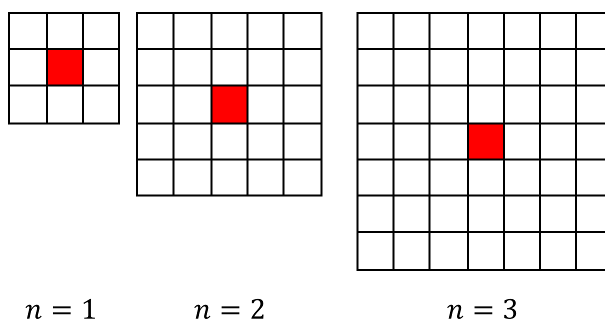


Obrázek 4.2: *Zde na obrázku lze vidět, že při sekvenčním přístupu $s = 1$ se může stát, že korekce nedokáže texturu kvalitně opravit, protože se textura opravuje na bázi okolí, které se mění. Pro $s = 2$ a $s = 3$ korekce kvalitně opravila texturu. Zato když je hodnota $s > 3$ nebo zásadně vyšší, kvalita korekce se spíše snižuje*

4.3 Velikost porovnávaného okolí

Tato hodnota je celé kladné číslo určující velikost okolí, podle kterého se hledá nejlepší kandidát na opravovaný pixel. Toto porovnávání dvou okolí se provádí pro každý pixel v nově vygenerované textuře pro jeden korekční krok. Aby nově vygenerovaná textura vypadala k obrazu vstupní textury. Protože se porovnání dvou okolí provádí pro každý pixel textury, velikost okolí zásadně ovlivňuje čas strávený na korekci. Ale i kvalitu vygenerované textury. Pro $n = 1$ výsledky nedosahují dostatečné kvality, zato $n = 2$ je už dostačující k porovnávání a tedy i nejlepší hodnota pro kvalitu / čas zpracování. Hodnoty $n > 2$ dosahují už jen velmi malého rozdílu vylepšení kvality po opravení textury. Tudíž doporučuji používat hodnotu $n = 2$ pro velikost porovnávaní okolí.

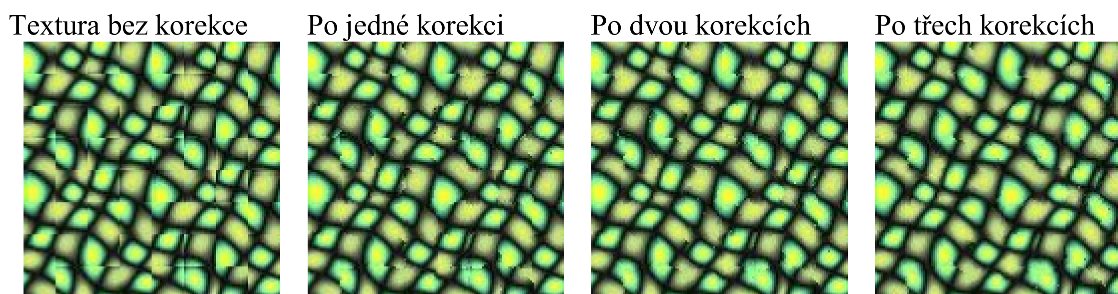
Doporučená hodnota: $n = 2$



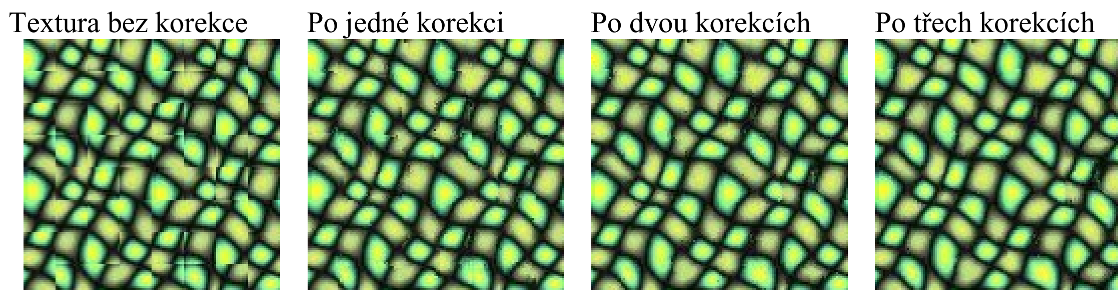
$n = 1$ $n = 2$ $n = 3$

Obrázek 4.3: *Zde vidíme, jak hodnota n ovlivňuje velikost porovnávaného okolí. Velikost tohoto okolí zásadně ovlivňuje dobu zpracování a kvalitu syntézy. Červený pixel uprostřed vyznačuje pixel, který bude nahrazen*

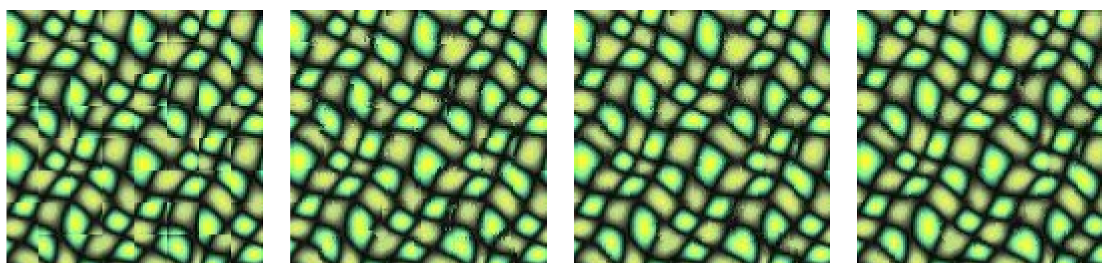
Textura bez korekce | Po jedné korekci | Po dvou korekcích | Po třech korekcích



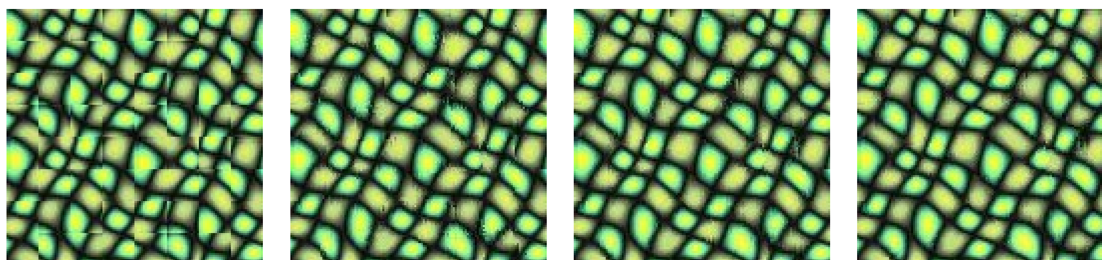
Obrázek 4.4: *Na obrázku lze vidět postupná korekce textury, kde se porovnávají okolí o velikosti 3x3, neboli $n = 1$. Zpracování všech korekčních kroků trvalo 424ms. Z první textury zprava můžeme vidět, když použijeme okolí $n = 1$ tak korekce nestihne ani po třech aplikovaných korekcích za sebou opravit texturu do kvalitního stavu bez hran*



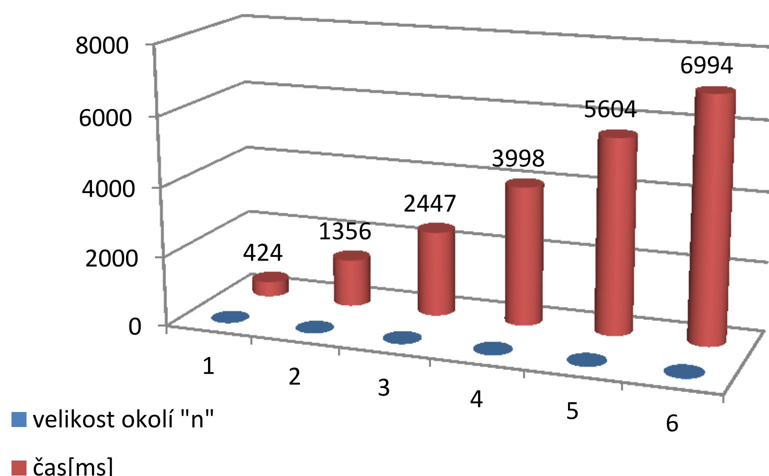
Obrázek 4.5: *Na obrázku lze vidět postupná korekce textury, kde se porovnávají okolí o velikosti 5x5, neboli* $n = 2$. *Zpracování všech korekčních kroků trvalo 1356ms. Z vygenerovaných textur můžeme usoudit, že již po druhém korekčním kroku je textura ve stavu bez viditelných hran. Zde vidíme obrovský rozdíl v počtu potřebných korekčních kroků oproti korekci s nastaveným okolím* $n = 1$, *zobrazené na obrázku 4.4*



Obrázek 4.6: *Na obrázku lze vidět postupná korekce textury, kde se porovnávají okolí o velikosti 7x7, neboli* $n = 3$. *Zpracování všech korekčních kroků trvalo 2447ms. Zde můžeme vidět, že okolí* $n = 3$ *již nezpůsobuje velký rozdíl oproti okolí* $n = 2$ *zobrazeno na obrázku 4.5, ale čas zpracování se zvýšil 2 krát, ale kvalita zůstává skoro stejná jak pro okolí* $n = 2$



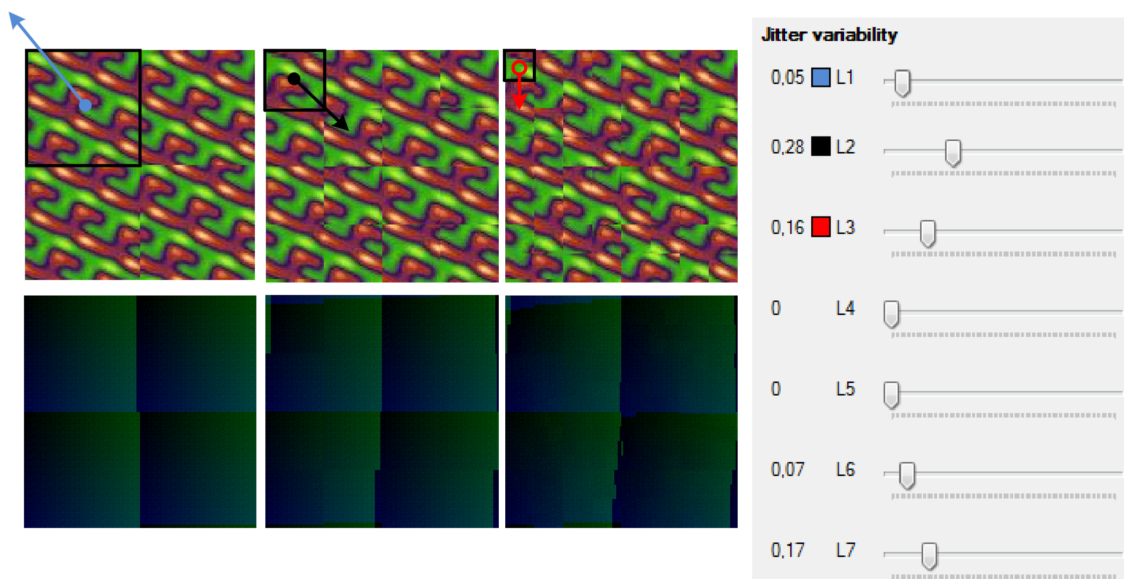
Obrázek 4.7: *Na obrázku lze vidět postupná korekce textury, kde se porovnávají okolí o velikosti 9x9, neboli* $n = 4$. *Zpracování všech korekčních kroků trvalo 3998ms. Oproti ostatním okolím vidíme, že konečná textura po třech opravách vypadá nejlépe, ale časová náročnost je až moc velká na použití takto obrovských okolí k porovnávání pixelů*



Obrázek 4.8: *Z grafu lze vidět poměr časové náročnosti na nastavení velikosti porovnávaného okolí n pro naleznutí náhrady za opravovaný pixel. Čas zobrazuje dobu v ms potřebnou k provedení tří korekčních kroků za sebou*

4.4 Hodnoty pro změnu náhodnosti textury

Náhodnost získáváme pomocí funkce chvění popsané v kapitole 3.3.2. Jednoduše řečeno funguje na posouvání bloků textury. Velikost bloků se mění dle úrovně, ve kterém se zrovna algoritmus nachází. Každý pixel v bloku je posunut o stejně velký náhodně vygenerovaný vektor, kde velikost tohoto vektoru může být ovlivněna uživatelem. Vstup uživatele má největší váhu na určení velikosti tohoto vektoru, protože vstupní hodnota uživatele je 0.0 až 1.0 a touto hodnotou vynásobíme náhodný vektor. Tím můžeme tento náhodný vektor kompletně vyrušit nebo ponechat bez dotyku. Pro každou úroveň je velikost náhodně vygenerovaného vektoru jiná, protože čím menší bloky posouváme, tím menší musí být vektor posunu. Jinak dosáhneme až moc velké změny a korekce již nedokáže tak velké změny ve struktuře opravit. Obecně řečeno na nízké úrovni posouváme velké bloky a směrem k vyšší úrovni posouváme čím dále menší a menší bloky. Posun bloků názorně zobrazuje obrázek 4.9. Například pokud chceme, zachovat určitou strukturu velkou jak vstupní textura, nastavíme náhodnost pro úroveň 1 a úrovní 5,6,7 změníme jen malé detaily výstupní textury. Doporučená hodnota by například mohla být 0.50, 0.0, 0.0, 0.1, 0.3, 0.5, 0.5.



Obrázek 4.9: *Na obrázku si lze všimnout posunu bloků pro různé úrovně. První obrázek vlevo posouvá velké bloky a na texturách směrem doprava se velikost bloků zmenšuje. Pomocí šipek je zobrazeno jakým směrem náhodné vektory posouvají vyznačené bloky. Vpravo lze vidět část uživatelského rozhraní, kde uživatel pomocí posuvníků vybírá velikost náhodných vektorů pro určitou úroveň*

4.5 Paměťová a časová náročnost

Všechny uvedené časové hodnoty byly naměřeny na notebooku s procesorem AMD Phenom II X3 N830 2.1GHz s pamětí 4GB na operačním systému windows7. Čas strávený na výpočet nové textury je velice ovlivněn nastavením jako například počtem korekčních kroků, velikostí porovnávaného okolí a velikostí výstupní textury. Tyto naměřené hodnoty byly získány pro velikost okolí rovno 2, počet korekčních kroků rovno 3.

Tabulka 4.1: *Tabulka ukazuje přehled časové a paměťové náročnosti*

Rozlišení	256x256	512x512	1000x1000	2000x2000	4000x4000
Paměť[MB]	3-6	11-15	30-40	93-105	340-380
Čas[min]	2-3	6-10	30-40	60-100	140-180

4.6 Vstupní a výstupní textury

Pomocí mé implementace vytvořené podle publikovaného algoritmu Parallel Controllable Texture Synthesis [1] jsem vygeneroval velké množství nových uměle vytvořených textur. Spustitelná aplikace je k naleznutí na DVD ve složce Spustitelný program s knihovny. Pro testovací účely bylo vybráno kvantum různorodých textur od pravidelných až po náhodné. Uloženy na dvd ve složce Vstupní textury. Většina nově vygenerovaných textur byla kvalitně zpracována bez viditelných hran, ale narazil jsem na dvě vstupní textury, které nebyly dostatečně kvalitně opraveny. Příklad nedokonale opravené textury je zobrazen na obrázku 4.11. Tento algoritmus nerozpoznal strukturu kvůli malému porovnávanému okolí, a proto textura nebyla kvalitně opravena. Tato metoda funguje pro většinu použitých vstupní textur od pravidelných až po náhodné s kvalitními výstupy. Kvůli velkému rozlišení a množství výstupních textur jsou všechny výstupy k naleznutí jen na DVD ve složce Výstupní textury. Některé uměle vygenerované textury jsem nanesl na 3D objekty a vyrenderoval pomocí programu Blender. Opět všechny vyrenderované obrázky jsou k naleznutí na DVD ve složce Renderované obrázky.

Obrázek 4.10: *Úspěšně uměle vygenerovaná textura*

Obrázek 4.11: *Neúspěšně vygenerovaná textura. Tento algoritmus syntézy textur nedokázal opravit tuto strukturu*

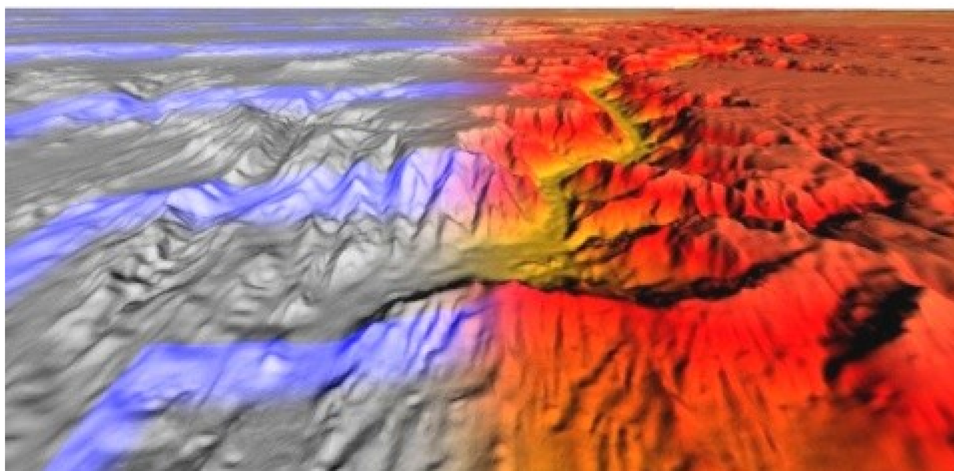
5 Budoucí práce

5.1 Vylepšení kvality

Do budoucna by vývojáři chtěli vylepšit kvalitu syntézy textur, protože na některých texturách nedosahuje takové kvality, jaká by byla potřeba.

5.2 Syntéza terénu

V dnešní době se pracuje na syntéze 3D terénu, některé algoritmy jako Distributed Texture-based Terrain Synthesis [7] už jsou vytvořeny, ale stále se vyvíjí další.



Obrázek 5.1: *Obrázek poskytnut z diplomové práce algoritmu Distributed Texture-based Terrain Synthesis[7]*

5.3 Menší nároky na paměť:

Jedním z důležitých parametrů je paměťová náročnost algoritmu. Proto vývojáři usilují o co nejnižší náročnost na paměť a chtějí zkusit naimplementovat komprimaci dat do algoritmu syntézy textur k co největší redukci paměťových nároků.

A další.

6 Závěr

Z testování a výsledků generovaných mou implementací již vytvořeného algoritmu Parallel Controllable Texture Synthesis jsem zjistil, že tento algoritmus přináší uživateli obrovskou kontrolu nad uměle generovanými texturami pomocí funkce chvění. Tento algoritmus syntézy textur zvládne generovat textury od pravidelných až po náhodné do určité míry s kvalitními výstupy. Ale jak víme, nic není dokonalé tudíž i tato metoda má své limity a nefunguje na všechny textury stoprocentně. Hlavní slabinou je známý nedostatek pixelově založených algoritmů, který je způsoben korekcí podle malého porovnávaného okolí. Použitím malého okolí pro korekci algoritmus nedokáže rozpoznat určité struktury, které tudíž nedokáže opravit. Hlavním důvodem použití malého okolí je získání rychlosti algoritmu. Protože jak je znázorněno v podkapitole 4.3 velikost porovnávaného okolí zásadně ovlivňuje rychlost korekčního kroku, který je nejvíce náročný na výpočet. Obrovským přínosem tohoto algoritmu je opravování pixelů nezávisle na sobě, neboli můžeme opravovat pixely v libovolném pořadí a pomocí více vláken. Dalším vylepšením, které nelze opomenout je použití textury souřadnic, díky které využíváme méně paměti a zvyšujeme rychlost funkce chvění.

Má implementace využívá pro všechny výpočty jen více jádrové procesory. Při generování normálních velikostí jako například 512×512 je nová textura vygenerována přibližně za 5 minut při použití 3 korekcí pro každou úroveň a velikost okolí je rovna 2. Ale při generování obrovských textur o velikosti například 4000×4000 pixelů s korekcí 7 krát a okolí o velikosti 2(5x5 pixelů) generování trvalo už více než dvě hodiny. Proto generování obrovských textur s využitím jen procesoru je nedostačující aspoň při využití mého procesoru z notebooku - AMD Phenom II X3 N830 2.1GHz.

Po detailním prozkoumání a testováním této metody bych do budoucna rád předělal výpočet korekce na GPU a chtěl vyzkoušet zkombinovat tuto metodu ještě více s metodou zaplatově orientovanou. Funkce zvětšování by mohla zůstat úplně stejná. Funkci chvění bych jen trošku předělal, aby ukládala větší okolí bloků, než jsou bloky posouvání. Na tyto přečnívající okolí bloků by se aplikoval algoritmus na naleznutí cesty nejmenšího rozdílu mezi dvěma překrývajícími se bloky. A až poté by se aplikovala korekce. S tím, že počet korekčních cyklů by se dramaticky zredukoval, protože obě okolí už by byli mnohem blíže a tudíž by nastavení korekce mohlo být hodně založené spíše na rychlost nebo na zvětšené okolí, čím bychom mohli vyrušit známý nedostatek pixelově založených algoritmů, který nastává při porovnávání dle malého okolí.

Použitá literatura

[1] LEFEBVRE, Sylvain a Hugues HOPPE. Parallel controllable texture synthesis [online]. ACM Trans. on Graphics 24, 3 (2005)[cit. 2014-04-10]. 777–786. Dostupné z: http://research.microsoft.com/en-us/um/people/hoppe/paratexsyn.pdf

[2] Parallel Controllable Texture Synthesis: Results data. LEFEBVRE, Sylvain a Hugues HOPPE.Microsoft Research: Parallel Controllable Texture Synthesi [online]. Proc. SIGGRAPH, 2005 [cit. 2014-04-10]. Dostupné z: http://research.microsoft.com/en-us/um/redmond/projects/paratexsyn/

[3] TONG, Xin, Jingdan ZHANGZ, Ligang LIU, Xi WANGZ, Baining GUO a Heung-Yeung SHUM.Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces [online]. Tsinghua University: ACM SIGGRAPH, 665-672., 2002[cit. 2014-04-10]. Dostupné z: http://research.microsoft.com/en-us/people/bainguo/p665-tong.pdf

[4] XU, Ying-Qing, Baining GUO a Harry SHUM. Chaos Mosaic: Fast and Memory Efficient Texture Synthesis [online]. April 20, 2000. Redmond, WA 98052[cit. 2014-04-10]. Dostupné z:http://www.visgraf.impa.br/Courses/ip00/papers/tr-2000-32.pdf

[5] Texture_synthesis. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-04-10]. Dostupné z:http://en.wikipedia.org/wiki/Texture_synthesis

[6] LASRAM, Anass a Sylvain LEFEBVRE. Parallel patch–based texture synthesis [online]. 2012[cit. 2014-04-23]. Dostupné z:http://alice.loria.fr/publications/papers/2012/ParallelPatch/ParallelPatchBasedTextureSynthesis.pdf

[7] TASSE, Flora Ponjou. Distributed Texture-based Terrain Synthesis [online]. University of Cape Town Cape Town, South Africa, 2011[cit. 2014-04-23]. Dostupné z: https://www.cl.cam.ac.uk/~fp289/pdfs/MScThesis_lowres.pdf

[8] ZHOU, Dongxiao. Texture analysis and synthesis using a generic Markov-Gibbs image model: What is a Texture [online]. The university of Auckland, 2006[cit. 2014-04-27]. Dostupné z: https://www.cs.auckland.ac.nz/~georgy/research/texture/thesis-html/node5.html

Seznam příloh

Součástí BP je DVD.

Adresářová struktura přiloženého DVD:

- Impelemntace – solution
- Obrázky z dokumentu
- Renderované obrázky
- Spustitelný program s knihovny
    - Data a knihovny
- Vstupní textury
- Výstupní textury