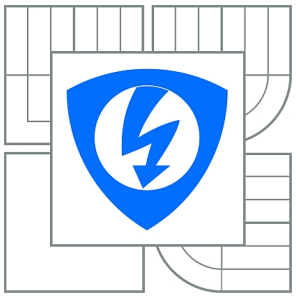
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

PERLINOVY ŠUMOVÉ FUNKCE PRO GENEROVÁNÍ TEXTUR

PERLIN NOISE FUNCTIONS FOR GENERATION OF TEXTURES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — RADKA JAKUBÍKOVÁ

AUTHOR

VEDOUCÍ PRÁCE — Mgr. PAVEL RAJMIC, Ph.D.

SUPERVISOR

BRNO 2012

ABSTRAKT

Tato práce se zabývá teoreticky i prakticky Perlinovou šumovou funkcí, pomocí níž lze generovat procedurální textury. Její algoritmus má hned několik výhod jako je přirozený vzhled, jednoduchost nanášení dvojrozměrných textur na trojrozměrné objekty nebo menší nároky na kapacitu paměti. Z těchto důvodů se dnes tato technika používá velice často v různých obměnách a vylepšeních v počítačové grafice. Nejčastěji se s takto vytvořenými texturami setkáme v počítačových hrách nebo animovaných filmech. Perlinovou šumovou funkcí lze vytvořit procedurální textury jenž působí jako mramor, dřevo nebo tráva. Příklady těchto textur generuje program, který je popsán v praktické části této práce. Tento program byl navrhnut pro jednoduchou prezentaci procedurálních textur vytvořené Perlinovým šumem. Program lze ovládat přes uživatelské rozhraní, v němž uživatel změnou některých parametrů mění vzhled generované textury. Tento program by měl sloužit k prezentaci Perlinova šumu studentům.

KLÍČOVÁ SLOVA

Perlinovy šumové funkce, textura, procedurální textura, struktura, program, šum, generovat, parametr, funkce, persistence, turbulence, vylepšení

ABSTRACT

This work deals with Perlin noise function in theoretical and practical way. Procedural textures can be generated by Perlin noise function. Perlin´s algorithm has many advantages, for example natural appearance, smaller demands on memory capacity and possibility of simple application on cubic objects. It is for these reasons that this technique is widely used nowadays in many varieties and improvements. We can meet these kind of textures mainly in computer games or animated movies. It is possible to create procedural textures by Perlin noise function so that they look like marble, wood or grass. Examples of these textures are generated by a program which is in the practical part of this work. The program was designed for simple presentation of procedural textures made by Perlin noise. Program can be operated via user interface, where the user can change some parameters which affect the look of the texture. The program can be used for presentation of Perlin noise to students.

KEYWORDS

Perlin noise function, texture, procedural texture, structure, programm, noise, generate, parameter, function, persistence, turbulence, improvements

JAKUBÍKOVÁ, Radka *Perlinovy šumové funkce pro generování textur*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 43 s. Vedoucí práce byl Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Perlinovy šumové funkce pro generování textur“ jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Mgr. Pavlu Rajmicovi, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne .....

.....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V počítačové grafice se dnes běžně používá Perlinova šumová funkce. Díky ní je jednoduché vytvářet přirozeně vypadající pozadí například v počítačových hrách anebo v animovaných filmech. Pomocí jednoduchého šumu je možné generovat různé textury připomínající například mraky, mramor nebo dřevo.

Ken Perlin vytvořil tuto procedurální techniku již v roce 1985. Od té doby se tento algoritmus vylepšoval a upravoval a dnes je možné používat různé postupy s různými implementacemi. Procedurálních technik je celá řada, Perlinova funkce je příkladem jedné z nich. Výhodou těchto technik je především jednoduchá aplikace textury na trojrozměrný objekt. Nezanedbatelnou výhodou je samozřejmě také malá náročnost na kapacitu paměti počítače.

Textury je možné generovat i za použití neprocedurálních technik. Jednou z nejznámějších je fraktálová technika. Na základě fraktální geometrie je možné vykreslit velice zajímavé obrazce. Nevýhodou této techniky je vyšší spotřeba paměti.

Textury je možné generovat pomocí různých programovacích jazyků. Původní verze Perlinovy funkce je známá v jazyku C. Cílem této práce bylo vytvořit program v jazyku MATLAB, jenž bude schopen generovat různé textury při změnách různých parametrů. V praktické části je popsán program, který vykresluje různé textury. Záleží na uživateli jak si zvolí velikosti parametrů a druhy textur v uživatelském rozhraní.

Text této práce je rozdělen na dvě kapitoly – teoretickou a praktickou. V teoretické části je popsána procedurální technika a její kategorie. Následuje část věnovaná fraktálům, které s procedurálními texturami úzce souvisí. Ještě více je s těmito texturami spjato jméno Ken Perlin, který Perlinovu šumovou funkci vymyslel. Popis původní verze jeho šumové funkce je v další podkapitole, která se zabývá i různými parametry, které mohou vzhled textury ovlivnit. Je uvedeno i možné vylepšení tohoto algoritmu a také použití Perlinovy funkce na vytvoření textury vypadající jako vodopád. Praktická část je vyplněna textem o programu, který se ovládá pomocí uživatelského rozhraní a generuje různé textury.

1 PROCEDURÁLNÍ TEXTURY

1.1 Pojem procedurální textura a její použití

Textura popisuje vlastnosti povrchu objektu, které jsou dány jeho velikostí, tvarem nebo hustotou. Textury mohou být rozděleny do dvou kategorií: hmatatelné neboli objemové a visuální. Textury jenž vnímáme hmatem popisují povrch daného objektu např. jeho hrubost či jemnost. To co člověk vnímá zrakem je tzv. visuální textura, jedná se především o barvu, intenzitu nebo orientaci obrazu sledovaného objektu [27].

Druhů textur je mnoho a tak byly rozděleny do skupin. První roztřídění provedl Heckbert [8] podle vlastností povrchů. Postupem času bylo třeba toto rozdělení rozšířit. Dnes tedy rozlišujeme textury podle toho, co popisují [28]:

- barvu povrchu,
- odraz světla,
- změnu normálového vektoru,
- průhlednost a
- hypertexturu.

Jiné možné rozdělení je podle jejich rozměru. Jednorozměrné textury jsou používány například pro generování opakujících se podélných vzorů např. přechod pro chodce. Dvourozměrné jsou mapovány na povrch tělesa, můžeme je i označit jako visuální, a trojrozměrné definují hodnotu textury v prostoru, jedná se tedy o textury objemové. Čtyřrozměrných textur lze použít například pro animaci textur trojrozměrných. Zmíněné textury se nejčastěji ukládají do jedno nebo dvou rozměrných tabulek například jako obrazový soubor. Nevýhodou tabulek je, že zabírají velké množství paměti a tak je lépe využívat procedurálních textur.

Procedurální technikou myslíme kód nebo algoritmus jenž specifikuje některé vlastnosti nebo efekty počítačově vygenerované textury. Například na definování barvy mramoru nepoužijeme digitální fotografii, ale nějakou matematickou funkci či algoritmus. Výhody používání procedurální techniky mohou být například [12]:

- kompaktnost, není potřeba megabytů na vytvoření a ukládání textury do paměti, stačí pouze pár kilobytů,
- funkce vzniklá na základě této techniky je spojitá, je možné vytvořit texturu jakéhokoli rozlišení,
- není periodická, můžeme pomocí ní pokrýt velké plochy bez obav z opakování vzorku.

Tyto a další výhody nemusí být bezpodmínečně splněny, ale měly by se zvážit při výběru techniky.

Příkladem procedurální techniky může být Perlinova šumová funkce. Lze ji taktéž zařadit do skupiny šumy gradientů v mřížce („lattice gradient noise“). Perlinova funkce generuje šum pomocí interpolace náhodných hodnot a gradientů definovaných v bodech mřížky. Výhodou a důvodem používání této funkce je, že je jednoduché texturu, vzniklou pomocí Perlinovy šumové funkce, namapovat na 3D (trojrozměrný – three dimensions) objekt (objekt musí být možné popsat matematickou funkcí) a nejdůležitější vlastnost je přirozený vzhled.

1.1.1 Procedurální šumové funkce

Procedurální techniky neboli procedurální šumové funkce, kterými vytváříme procedurální textury, můžeme rozdělit do tří kategorií [12]:

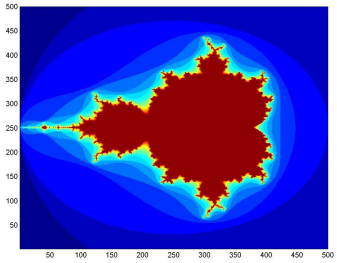
- šumy gradientů v mřížce („lattice gradient noises“) – generují šum pomocí interpolace nebo konvoluce náhodných hodnot a gradientů definovaných v bodech mřížky, příkladem je Perlinova šumová funkce a nebo její variace, vylepšení či rozšíření jako je např. „flow noise“ – funkce generující časově proměnnou plynoucí texturu založenou na Perlinově šumu připomínající proud lávy nebo vodopád,
- explicitní šumy („explicit noises“) – generují šum explicitně v předprocesoru a ukládají ho do paměti, příkladem těchto šumů může být vlnkový šum („wavelet noise“) [3],
- řídké konvoluční šumy („sparse convolution noises“) – příkladem může být bodový šum („spot noise“), který funguje na principu náhodného rozložení malých vzorků („spot“) po povrchu tak, že vytvoří texturu.

1.1.2 Fraktály

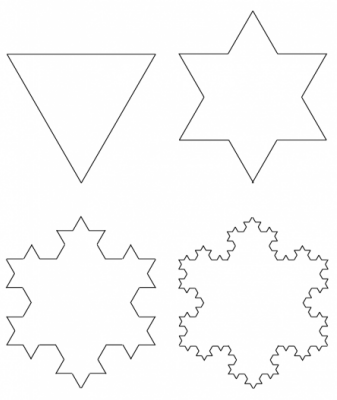
Fraktály se sice neoznačují jako procedurální textury, ale velice s procedurálními techniky souvisí, proto je uveden krátký popis.

Za objevitele fraktální geometrie je považován Benoit B. Mandelbrot [1]. Jeho nejznámější nelineární deterministický fraktál byl pojmenován podle něj a to Mandelbrotova množina viz obr. 1.1.

Fraktální geometrie využívá soběpodobnosti, jednodušeji řečeno – čtverec lze složit z libovolného počtu menších čtverců. Dalším velice známým fraktálem je Kochova sněhová vločka viz obr. 1.2.



Obr. 1.1: Obarvená Mandelbrotova množina [9]



Obr. 1.2: Postup vzniku Kochovy vločky [6]

Tato vločka vznikla z prostého rovnostranného trojúhelníku. Z každé strany trojúhelníku je vyjmuta prostřední třetina, nad tímto úsekem jsou vztyčeny dvě strany rovnostranného trojúhelníka, jehož velikosti stran se rovnají třetině jedné strany původního trojúhelníka viz obr. 1.2. Tímto způsobem by se dalo pokračovat, okraj vločky by se zjemnil a po nekonečně mnoha iteracích by se docílilo nekonečné délky okraje. Fraktály se mohou používat nejen pro generování zajímavých textur, ale také jako tzv. fraktálové kapacitory [15].

Pro vyobrazování přírodních jevů jako jsou mraky, hory nebo kameny se využívá náhodných fraktálů, ty mají základ v Brownově pohybu (chaotický pohyb částic v kapalině nebo plynu). Ke generování povrchů pomocí fraktálů lze použít dvou metod [28]:

- metodu náhodných chyb, např. k vykreslení modelů planet,
- metodu náhodného přesouvání středního bodu, nejčastěji k vytváření obrysu pobřeží.

Nadále se budeme zabývat již jen Perlinovými funkcemi, které do „počítačově chladných“ obrázků vnášejí náhodné prvky, takže výsledek působí jako realita. Nejčastější použití těchto funkcí je pro generování textur mramoru, dřeva, mraků a jiných. Můžeme se s nimi setkat v každém filmu, který používá renderovanou grafiku (generování obrazu na základě počítačového modelu) nebo v počítačových hrách.

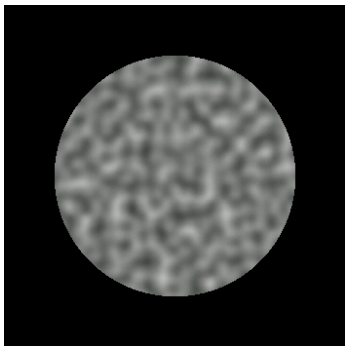
1.2 Profesor Perlin

S pojmem Perlinova funkce je úzce spjat jeho autor Ken Perlin.

Obr. 1.3: Ken Perlin [16]

Perlin je profesorem na Newyorské univerzitě v oblasti počítačové techniky a získal několik ocenění v oblasti počítačové grafiky. Za svou práci v rámci procedurálních textur získal roku 1997 Akademické ocenění za technické úspěchy od Akademie filmových umění a věd. Ovšem počítačovou grafikou se začal zabývat již roku 1981, kdy spolupracoval na filmovém projektu. Ve filmu se objevilo velké množství počítačem generovaných prvků, ale vše bylo „počítačově chladné“. Perlin se rozhodl vymanit se z tohoto nepřirozeného prostředí a začal pracovat na tom, jak by se daly uměle vykreslit povrchy tak, aby působily přirozeně. Prvním krokem bylo uvědomění si, že se dosud všude využívalo booleanovských kombinací matematických primitiv jako jsou elipsoidy, kužely nebo válce. Výsledky těchto matematických programů nepůsobily dobře a tak se Perlin rozhodl zabývat otázkou textury ne na povrchu ale v objemu. Tak vznikly „projection texture“ tedy projekční textury. Ty ale potřebovaly velkou kapacitu paměti RAM a to počítače, které byly tehdy dostupné, neměly. To pro Perlina a jeho tým znamenalo nutnost zkoušet dál. V následujícím roce vytvořil primitivní šumovou funkci, která naplňovala prostor a působila náhodně. Dalším požadavkem bylo získat proměnnou, která vypadá náhodně a musí být zároveň kontrolovatelná, takže se dají vytvořit různé textury. Důležité je, aby „náhodné proměnné“ byly stejné velikosti a zhruba izotropní (stejné vlastnosti ve všech směrech). Na základě těchto požadavků a poznatků vytvořil Perlin kód pro svou první verzi procedurální textury viz 1.3.2.

Dalším požadavkem bylo vytvoření zajímavějších textur jenž by působily nepravidelně. K tomu Perlin využil funkce šumu. Koncem roku 1983 napsal Perlin programovací jazyk, který pro každý pixel v obrázku vzal pozici na ploše a vyvolal pro něj program pro stínování, vytvoření textury a nebo barvy. Tuto práci Perlin prezentoval roku 1984 na SIGGRAPHu (jedna z nejvýznamnějších konferencí počítačové grafiky). Protože byla tato technika velice jednoduchá, začala se brzy používat firmami jako Pixar, Alias nebo SoftImage. V roce 1988 byla již tato metoda používána ve všech komerčních programech. Perlin si své metody nikdy nedal patentovat.



Obr. 1.4: Jeden z prvních Perlinových pokusů [13]

Koncem 80. let se Perlin přidal k výzkumům, které probíhaly na Newyorské univerzitě. Zabýval se zde otázkou, jestli je možné sjednotit vykreslování a tvarování procedurálních textur, a tak začal pro generování procedurálních textur využívat paprsky, které prochází objemem dané struktury a podle zjištěného gradientu hustoty vyvolávají správné nasvícení. Tuto techniku nazval *hypertexture*, protože se jedná o texturu ve vyšší dimenzi.

Perlin dostal spoustu dalších ocenění za svou práci, stále pracuje na zajímavých metodách v počítačové grafice, je zván na setkání se studenty a na konference po celém světě. Svou práci neustále rozvíjí a své denní poznatky a zážitky uveřejňuje na svůj blog [10]. Ukázky jeho práce jsou dostupné na jeho webu [16]. Je zde pár textur, ale také zajímavé hry a jiné interaktivní applety.

1.3 Perlinovy šumové funkce

1.3.1 Šum obecně

Šum je nechtěná elektromagnetická nebo elektrická energie, která snižuje kvalitu signálů a dat, jež jsou vysílána jakýmkoliv zdrojem. Šum se vyskytuje u digitálních i analogových systémů a může ovlivnit komunikaci různých typů například obsahující text, programy, obrázky anebo zvuk [26]. U fotografií se projevuje například tak, že je obraz tečkovaný a barevně nevzhledný. Naopak u generování textur vnáší do obrazu prvky přirozenosti. Toho Perlin využil a zkombinoval šum s matematickými funkcemi. Touto kombinací vznikají procedurální textury, jež lze generovat Perlinovou šumovou funkcí, která splňuje tyto požadavky [28]:

- funkce je statisticky invariantní vzhledem k otáčení,
- je statisticky invariantní vzhledem k posunutí,
- je spojitá,
- má omezené frekvenční spektrum,
- je opakovatelná, zavoláním funkce s parametrem x vrátí vždy stejnou hodnotu y.

Splnění dvou prvních požadavků zajistí statistickou neměnnost při otáčení a posouvání funkce. Nezáleží tedy na tom, kde začneme texturu mapovat. Splněním třetího bodu dosáhneme určité maximální frekvence u funkce a aby nevznikal alias, dá se v algoritmu navolit tato nejvyšší frekvence. Rychlý výpočet funkce je spojen s vlastností spojitosti funkce. Funkce je náhodná, ale tato „náhoda“ se dá řídit.

1.3.2 Generování jednoho pásma

Základem Perlinova šumu je generování šumu spojitého. Jedná se o výpočty náhodných hodnot a mezi nimi je prováděna interpolace, to má za výsledek vyhlazenou funkci. Na vyhlazení funkce se používá různých interpolací, jak jednoduché lineární tak kvadratické, kubické nebo funkce goniometrické a jejich vzájemných kombinací.

Perlinův šum může být generován v různé dimenzi. Uvedeme si trojrozměrný případ. Máme funkci $noise(x, y, z)$, která nám vytváří základ šumové funkce, jenž nám vrací pro hodnoty $[x, y, z]$ vždy stejné „náhodné“ číslo z intervalu $\langle -1; 1 \rangle$. Pokud funkci zavoláme s určitým parametrem, vždy bude výsledek stejný, to je důležitá vlastnost, která je i jedním z požadavků na tuto funkci, viz výše. Důsledkem je, že na stejné místo objektu bude vždy namapována stejná barva.

Výpočet Perlinovy šumové funkce si uvedeme v 3D prostoru, který je definován pravidelnou mřížkou. Postup výpočtu funkce pro bod s reálnými souřadnicemi $[x, y, z]$, který si pro přehlednost označíme jako A, je následující:

1. Najdeme si náš výchozí bod A se souřadnicemi $[x, y, z]$.
2. Poté hledáme osm okolních bodů v mřížce (počet okolních bodů vyplývá z počtu dimenzí, pro n dimenzí: 2^n okolních bodů, tedy pro dvojrozměrný prostor by bylo 2^2 okolních neboli sousedních bodů), které budou mít souřadnice: $[i, j, k], [i + 1, j, k], [i, j + 1, k], \ldots, [i + 1, j + 1, k + 1]$ a budeme je označovat jako body Q. Hodnota i se určí zaokrouhlením dolů souřadnice x bodu A. Stejně se určí velikost j a k, zaokrouhlením y, z.
3. Vygenerujeme pole G, obsahující 256 vektorů:
   - Vygenerujeme 3 pseudonáhodná čísla x, y, z z intervalu $\langle -1; 1 \rangle$, tím vznikne vektor (x, y, z).
   - Vektor znormalizujeme, to znamená, že absolutní hodnota vektoru bude rovna 1.
   - Dva předešlé body postupu provedeme 256krát a naplníme tak pole G 256 vektory.
4. Nyní zavedeme pole P, jenž slouží k přístupu do pole G. Pole P vznikne následujícím způsobem:

- Naplníme pole P čísly od 0 do 255: $\mathrm{P}[0] = 0, \mathrm{P}[1] = 1, \ldots, \mathrm{P}[255] = 255$.
- Pro i od 0 do 255 vygenerujeme celá pseudonáhodná čísla k z intervalu $\langle 0; 255 \rangle$.
- Zaměníme $P[i]$ a $P[k]$, např.: $P[i = 0] = 0$ zaměníme za $P[k = 3] = 3$, výsledkem bude $P[0] = 3$.
- Nyní máme pole P naplněné náhodně čísly od 0 do 255, přičemž každé číslo z intervalu $\langle 0; 255 \rangle$ se v tomto poli bude vyskytovat právě jednou.

5. V dalším bodu vybereme z pole G jen jeden vektor (x, y, z) pomocí tohoto vztahu:

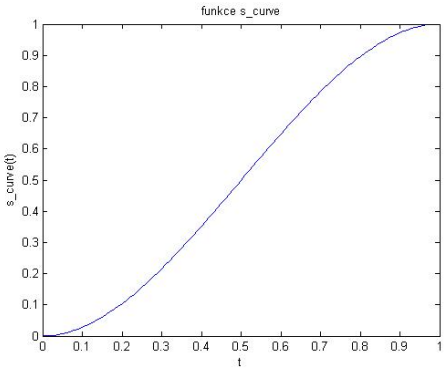
$$\mathrm{G} = \mathrm{G}[(i + \mathrm{P}[(j + \mathrm{P}[k])\mathrm{mod}n])\mathrm{mod}n], \tag{1.1}$$

kde $n = 256$ a hodnoty *i, j, k* jsou vždy hodnoty souřadnic okolních bodů Q. Znamená to tedy, že pro každý okolní bod Q, je vybrán jeden náhodný vektor (x, y, z) z pole G.

6. Dále vypočítáme hodnotu t, parametr pro lineární interpolaci, pomocí funkce, kterou pojmenujeme *s_ curve*:

$$s\_curve(t) = (t^2(3 - 2t)). \tag{1.2}$$

Průběh funkce je názorně vidět na obr. 1.5 . Za proměnou t dosazujeme postupně hodnoty souřadnic *x, y, z* bodu A. Tyto souřadnice ale nejdříve upravíme: $x^{'} = x - \lfloor x \rfloor$, $\lfloor x \rfloor$ je hodnota x zaokrouhlená dolů. Stejně postupujeme pro hodnoty y a z. Nyní máme tři nová čísla $x^{'}, y^{'}, z^{'}$, která jsme upravili funkcí *s_ curve*. Tato čísla teď použijeme jako parametr t v lineární interpolaci.



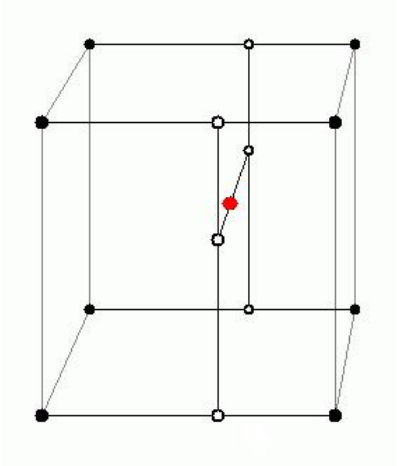
Obr. 1.5: Průběh funkce *s_ curve*

7. Posledním bodem je výpočet interpolace. Počítá se vždy $2^n - 1$ interpolací, v našem případě (ve 3D) to znamená 7 interpolací. Lineární interpolace označená jako *lerp* má tvar:

$$lerp(t, a, b) = (a + t(b - a)). \tag{1.3}$$

Parametr t jsme vypočítali v předchozím bodu. Hodnoty a a b budou odpovídat souřadnicím dvou bodů, mezi kterými budeme interpolaci počítat. Interpolace provedeme takto, názorně na obr. 1.6 :

- 4 interpolace ve směru x – použijeme pro výpočet parametru t hodnotu x' bodu A, hodnoty a, b budou rovny souřadnicím i ve dvou okolních bodech Q, pro které interpolaci počítáme. Výsledné body interpolace jsou v obrázku vyznačeny prázdnými tečkami.
- 2 interpolace ve směru y – použijeme parametry stejně jako v předchozím bodě, jen ve směru osy y.
- 1 interpolace ve směru z – parametry ve směru osy z, získáme hodnotu výsledného bodu, který jsme hledali, je vyznačen v krychli červenou tečkou.



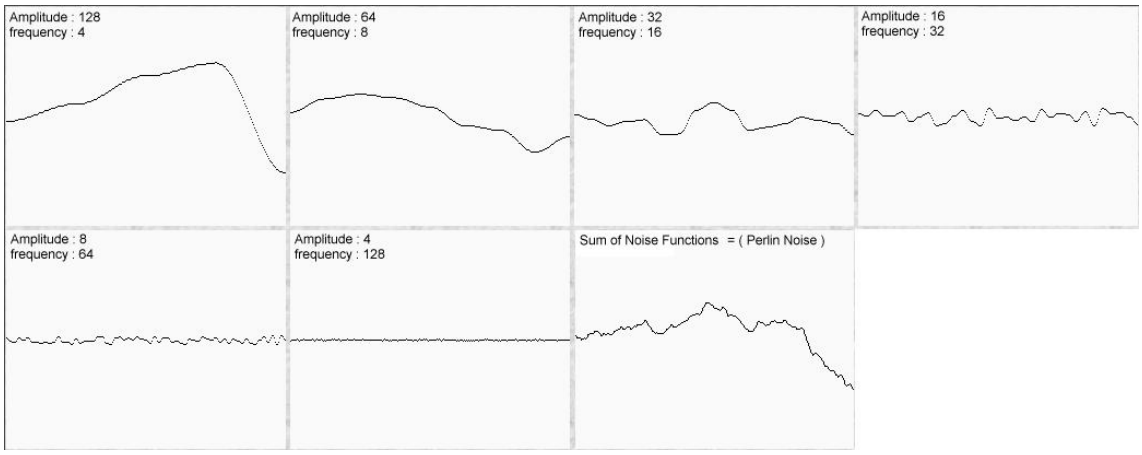
Obr. 1.6: Výsledky interpolací [14]

V tomto postupu jsme použili pole G o velikosti 256 a permutované pole P z důvodu podpoření náhodnosti a předcházení opakování vzorku. Šum se i přesto opakuje, ale po dlouhém intervalu, což znamená, že je opakovaný vzorek těžce postřehnutelný. Tyto funkce se používají především pro generování pozadí, takže nejsou často přibližovány a blíže zkoumány, proto nám pole o velikosti 256 stačí.

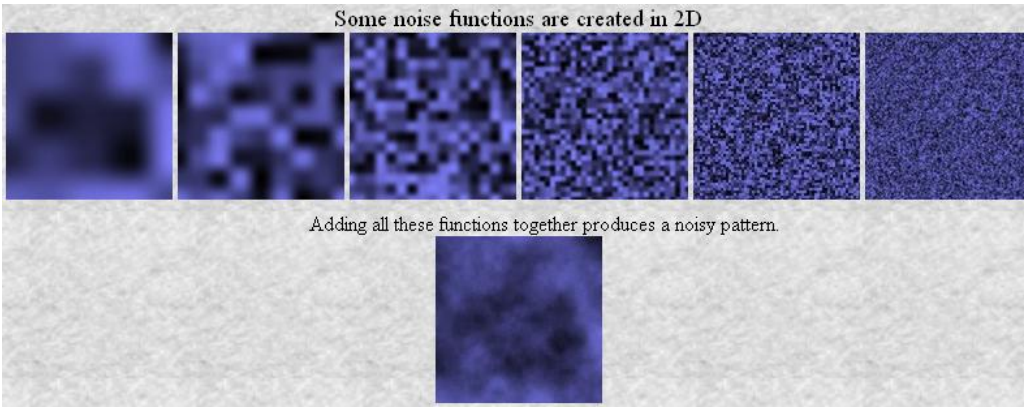
1.3.3 Generování ostatních pásem

Šum jednoho pásma, který byl rozebrán v předchozí kapitole, se nepoužívá přímo. Většinou je více takovýchto šumů s různou amplitudou a frekvencí sečteno.

Jestliže máme funkci $noise(x, y, z)$ z předchozího příkladu, a vytvoříme si další funkce jí podobné, pouze změníme amplitudu nebo frekvenci a z těchto funkcí uděláme sumu, dostaneme složenou šumovou funkci. Každé další nové funkci, kterou přičítáme, říkáme oktáva, neboli „šumová harmonická“. Takový součet může vypadat například jako na obr. 1.7.



Obr. 1.7: Generování ostatních pásem a jejich součet v 1D [21]



Obr. 1.8: Jednotlivá pásma a jejich součet ve 2D [21]

Ve funkcích se se zvyšující frekvencí snižuje amplituda a výsledná funkce tvarem připomíná pohoří. Již se nejedná o hladkou křivku a použitím takovéto funkce ve 2D (dvourozměrný – two dimensions) by již mohla vzniknout zajímavá textura. Rozdíl mezi dvourozměrným a jednorozměrným šumem je ten, že interpolaci provádíme ve dvou směrech a výsledek těchto interpolací se sečte. Výstupem zmíněné funkce je již obrázek nějakého šumu připomínající kouř či mrak viz obr. 1.8.

1.3.4 Skládání funkcí, pojem persistence

Jak bylo naznačeno, jednoduchá šumová funkce není až tolik zajímavá o samotě, jako když se sečte s dalšími oktávami. Nejčastěji se používají oktávy s dvojnásobnou frekvencí, ale pokud se oktávy od sebe liší jinak, budou výsledné textury zajímavější.

Výsledná funkce složená z několika jednoduchých šumů se nazývá *snoise*():

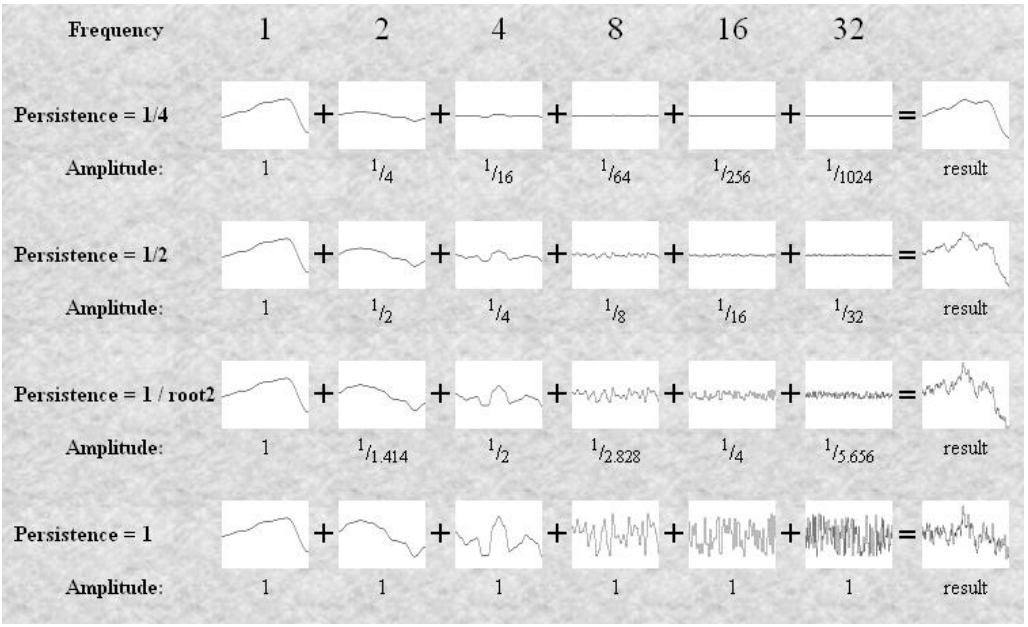
$$snoise(x, y, z, p, n) = \sum_{i=0}^{n-1} a_i \cdot noise(f_i\ x, f_i\ y, f_i\ z), \tag{1.4}$$

v tomto výrazu platí:

$$f_i = 2^i,$$

$$a_i = p^i,$$

kde f je frekvence, a je amplituda, i je i-tá oktáva, n značí počet oktáv a $p \in (0, 1)$ je persistence – je to pojem zavedený pro vyjádření amplitudy pro danou frekvenci. Tato hodnota byla poprvé použita Mandelbrotem [1], který stál u počátku fraktálů. Vliv persistence na výslednou funkci je nejlépe pozorovatelný na obr. 1.9.



Obr. 1.9: Ukázka vlivu persistence [21]

Z obrázku 1.9 je patrné, že při zvyšující se frekvenci, rychle se snižující amplitudě a persistenci 1/4 má výsledný šum tvar podobný šumu při nejnižší frekvenci a šumy s větší frekvencí se na výsledném tvaru nepodílí. Při persistenci 1 a stálé amplitudě 1 je výsledek nejvíce rozkmitaný, šum je zde největší. Čím vyšší je persistence, tím více se projevuje frekvence, tím zajímavější obrázky budou. Je tedy vidět, že nízké oktávy udávají tvar výsledné funkce a vysoké oktávy určují detaily.

1.3.5 Turbulence, rampa funkce

Pro vytváření různě zajímavých textur se na Perlinovu funkci aplikuje tzv. turbulence. Díky změně tohoto parametru se výsledná funkce stává více náhodnou. Pomocí turbulence se vytvářejí textury, které připomínají mramor nebo dřevo a další. Velikost turbulence ve funkci lze měnit podle potřeby, je to tedy parametr, jenž si uživatel zadává.

Na vykreslení jednoduché textury bez turbulence, viz obr. 1.10 první povrch vlevo nahoře, použijeme tzv. *rampa funkci* nebo také „marble_color“ tedy barevný filtr, který vytvoříme pomocí funkce sinus:

$$rampa(x) = \frac{1}{2}(1 + \sin x). \tag{1.5}$$

Objemová struktura mramoru se pak určí:

$$mramor(x, y, z) = rampa(x + c \cdot snoise(x, y, z, p, a, n)). \tag{1.6}$$

V tomto případě je parametr turbulence roven proměnné c, která může být zadána uživatelem, nejčastěji je v rozsahu od nuly do jedné. Pokud za *rampa funkci* dosadíme, dostaneme výraz:

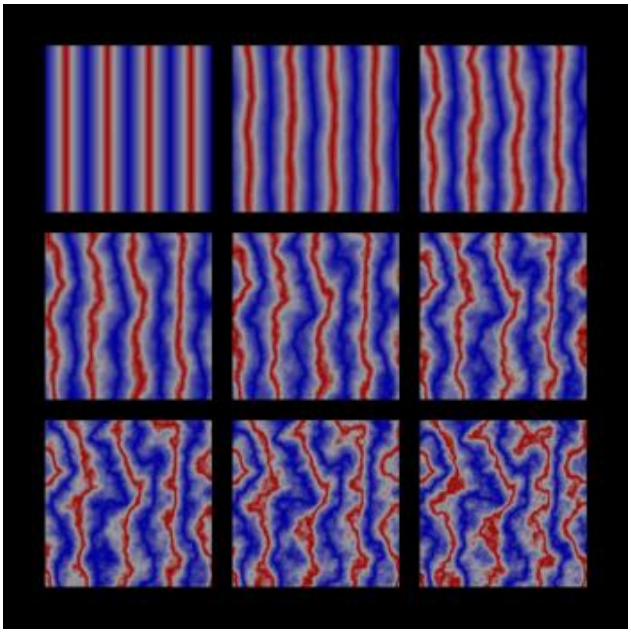
$$mramor(x, y, z) = \frac{1}{2}(1 + \sin(x + c \cdot snoise(x, y, z, p, a, n)). \tag{1.7}$$

Výsledek takové funkce ve 2D může vypadat jako na obr. 1.10. *Rampa funkce* určuje barvu v konkrétním bodě, zesilující turbulence pravidelné pruhy více „rozkmitá“ a tak výsledný obrázek připomíná strukturu mramoru. Na obr. 1.11 je výsledek funkce v 1D (jednorozměrný – one dimension) v případě $c = 0$ a $c = 0,8$. Je vidět, jak se výsledná funkce díky turbulenci změní.

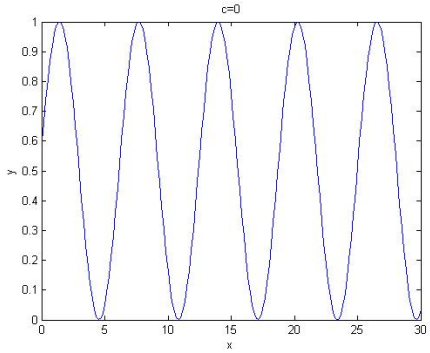
Další známou texturou, kde se využívají Perlinovy funkce, je dřevo. Pro tu se použije vztah:

$$wood(x, y, z) = perlin(x, y, z) - \lfloor perlin(x, y, z) \rfloor , \tag{1.8}$$

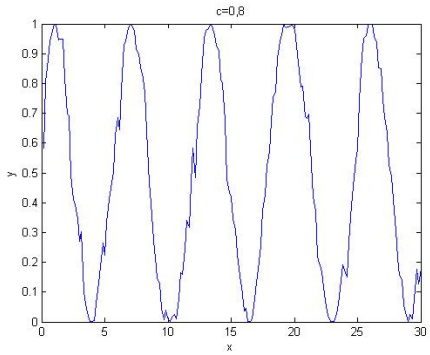
kde $\lfloor f(x) \rfloor$ vrací hodnotu této funkce zaokrouhlenou směrem dolů. Vzhled struktury v tomto případě lze ovlivnit volbou harmonických složek šumu.



Obr. 1.10: Vliv turbulence na Perlinovu funkci. U prvního povrchu je míra turbulence $c = 0$ a u posledního je $c = 0,8$. [23]



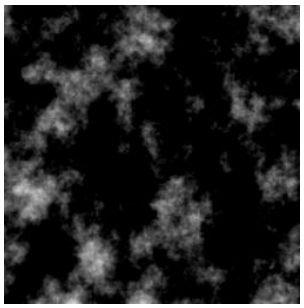
(a) $c = 0$



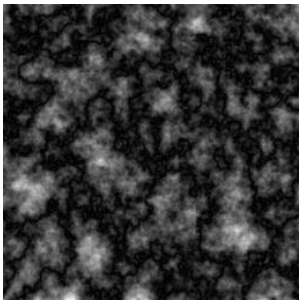
(b) $c = 0,8$

Obr. 1.11: Vliv turbulence v 1D

Pro obměnu vzhledu textury se též využívá absolutní hodnoty $|noise(x, y)|$, to způsobí vyobrazení i záporných hodnot, zviditelní se více frekvencí a např. obrázek oblohy poté působí jako více oblačný.



(a) použití šumové funkce bez absolutní hodnoty



(b) funkce s absolutní hodnotou

Obr. 1.12: Použití funkce $|noise(x, y)|$ [2]

1.4 Simplex noise

Perlin v roce 2001 navrhl zjednodušení („simplex noise“) své původní funkce [7], kterou zveřejnil roku 1985. Hned v dalším roce 2002 navázal a dále vylepšil svou funkci, jak je popsáno v další kapitole 1.5.

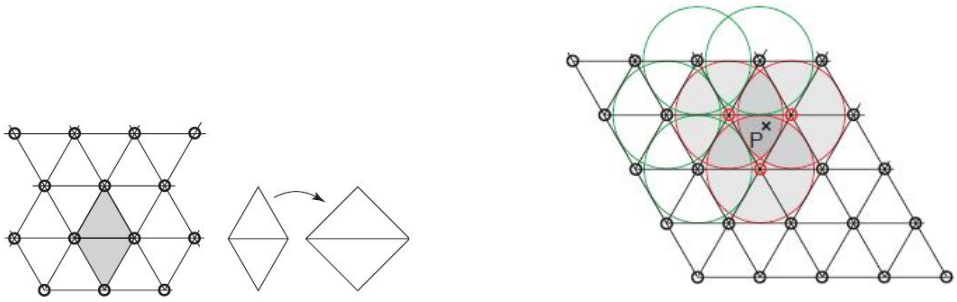
Tato kapitola se zabývá použitím tzv. „simplex noise“, který bychom mohli zařadit do skupiny „šumy gradientů v mřížce“ viz rozdělení v 1.1.1. Jedná se o funkci, která je srovnatelná s původní verzí Perlinova algoritmu, pouze má za cíl zjednodušit výpočty a to především pro N-dimenzionální šum.

Hlavním důvodem úpravy původního algoritmu je vyhnutí se některým nedostatkům. „Simplex noise“ má hned několik výhod [7]:

- výpočetně nenáročný a nevyžaduje spoustu násobení,
- nemá žádné patrné artefakty,
- má dobře definovaný a spojitý gradient, takže je jednoduchý na počítání,
- může být zobecněný pro vyšší dimenze a bude stále relativně málo výpočetně náročný.

„Simplex noise“ používá tzv. „simplex grid“ tedy jednoduchou souřadnicovou síť (mřížku). Hlavní myšlenkou je najít nejjednodušší a nejkompaktnější tvar, který můžeme opakovat a naplnit jím prostor s N dimenzemi. Pro 1D jsou to intervaly stejné délky, které jsou kladeny jeden za druhým. V dvojrozměrném prostoru se používá rovnostranných trojúhelníků, které jsou skládány vedle sebe a na sebe viz obr. 1.13a.

Ve 3D je použitým tvarem zkosený čtyřstěn. Tyto čtyřstěny se používají v šesti kusech a tvoří tak krychli, která je zkosená podle hlavní diagonály. Obecně jde vždy o takový tvar, který má $N + 1$ rohů, přičemž N je počet dimenzí, a $N!$ tvarů může vyplnit tzv. N-dimenzionální hyperkrychli, která je zkosená podle své hlavní diagonály. Z toho je viditelný rozdíl mezi původní Perlinovou šumovou funkcí, která používá hyperkrychli s 2^N rohy, a „simplex noise“, který používá pro N dimenzí $N + 1$ rohů.

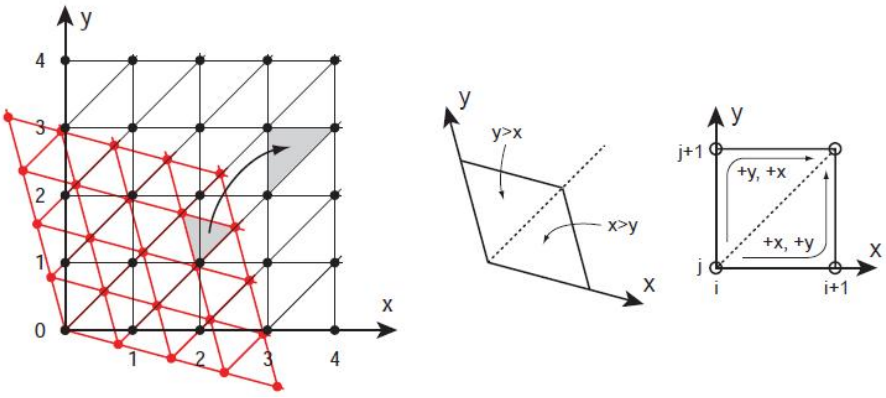


(a) tvar použitý v 2D

(b) body pro sčítání ve 2D – výsledek se určí sečtením pouze červeně zakroužkovancýh bodů

Obr. 1.13: Mřížka ve 2D [7]

Místo výpočtů interpolace v každém směru prostoru používá „simplex noise“ přímé sčítání příspěvků každého rohu tvaru, který je použit podle počtu dimenzí. Pokud bychom měli rovnostranný trojúhelník v 2D mřížce, bude bod, který je uvnitř jednoho trojúhelníku, počítat pouze s příspěvky okolních vrcholů trojúhelníku, ostatní vrcholy trojúhelníků, ve kterých bod neleží, nebere v úvahu a považuje je za nulové viz obr. 1.13b.



Obr. 1.14: Určení pozice bodu v 2D mřížce [7]

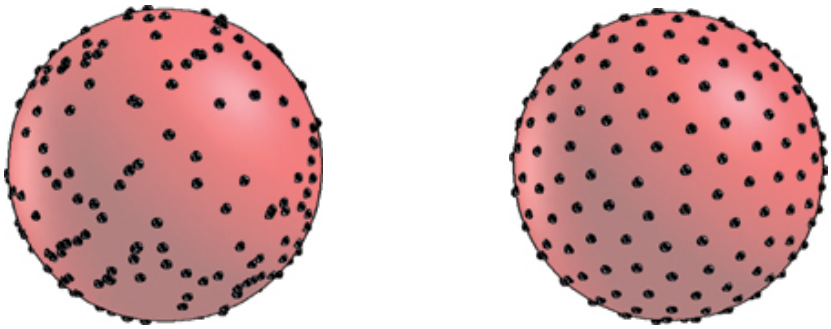
Posledním bodem algoritmu „simplex noise“ je rozhodnout ve které části mřížky se bod, který chceme vyčíslit, nachází. Zaprvé nejdříve musíme zkosit vstupní souřadnicový systém podle hlavní diagonály, takže dostaneme klasický souřadnicový systém viz obr. 1.14. Poté se jednoduše rozhodneme, ve které části prostoru se hledaný bod nachází, podle celočíselné části souřadnic hledaného bodu. Podle velikosti souřadnic lze také stanovit, zda bod leží výše nebo níže v mřížce, kterou prohledáváme. Podobný postup lze použít i ve 3D a vyšších dimenzích. [7]

1.5 Vylepšení Perlinovy šumové funkce

1.5.1 Nedostatky původní verze Perlinovy funkce

Původní verze Perlinovy funkce není úplně ideální. Nedostatky jsou [19]:

- Použití interpolační funkce $3t^2 - 2t^3$ v prvotní verzi Perlinovy funkce 1.3.2, jejíž druhá derivace je nenulová, může při generování šumu způsobit nechtěné visuální artefakty.
- Rozložení pseudonáhodných čísel ve vektoru G (1.3.2 třetí bod) je nepravidelné a zároveň jsou produkovány vysoké frekvence v šumu. Nerovnoměrné rozložení hodnot lze pozorovat na jednotkové kouli viz obr. 1.15.



(a) nerovnoměrné rozložení gradientů

(b) rovnoměrné rozložení gradientů

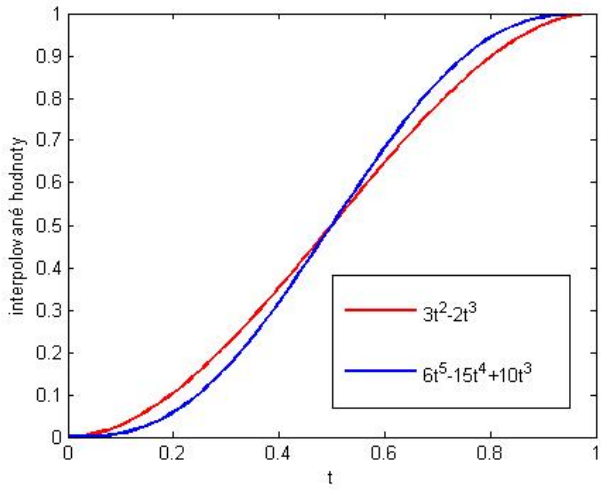
Obr. 1.15: Rozložení gradientů vektoru G [19]

1.5.2 Vylepšení šumové funkce

Perlin odstranil a vylepšil svou původní verzi následujícím způsobem [19]:

- Původní interpolace se prováděla pomocí funkce $3t^2 - 2t^3$ jejíž první derivace $6t - 6t^2$ je v bodech $t = 0$ a $t = 1$ rovna nule. Druhá derivace $6 - 12t$ je ale nespojitá, protože v bodech $t = 0$ a $t = 1$ je různá od nuly. Tento problém byl vyřešen změnou interpolační funkce na pátý řád interpolace: $6t^5 - 15t^4 + 10t^3$. Tato funkce má první derivaci $30t^4 - 60t^3 + 30t^2$ a druhou derivaci $120t^3 - 180t^2 + 60t$ rovnu nule v $t = 0$ i $t = 1$. Rozdíl mezi interpolacemi je zřejmý z obr. 1.16.
- Rovnoměrného rozložení gradientů vektoru G (1.3.2 třetí bod) je dosaženo nahrazením 256 gradientů pouze 12 gradienty ve vektoru G. Výsledkem této změny je méně „flekatá“ textura. Důvodem použití pouze 12 gradientů je, že žádný z těchto gradientů není příliš blízko jiného. „Flekatý“ vzhled způsobovalo nerovnoměrné rozložení.

Perlinovu funkci pro své algoritmy použili i jiní. Při aplikaci různých vylepšení na tuto funkci jsou schopni vygenerovat různé procedurální textury jenž můžeme řadit do skupiny šumy gradientů v mřížce 1.1.1.



Obr. 1.16: Průběhy interpolací

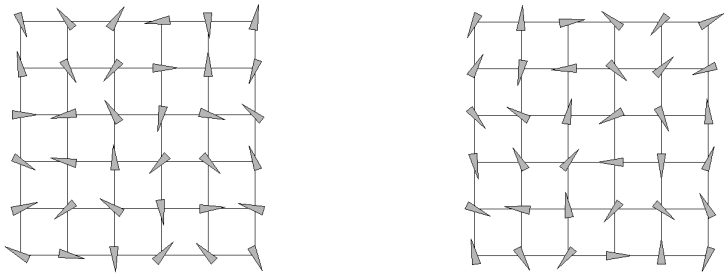
1.6 Flow noise – proudící šum

Mezi procedurální textury nepatří pouze Perlinova šumová funkce, je možné tyto textury vygenerovat i jinými technikami. Existuje několik procedurálních textur, které jsou založeny přímo na Perlinově šumu. Jednou z nich je i tzv. funkce „flow noise“ [18].

Flow noise vznikla na základě toho, že Perlinův šum nepůsobí jako opravdový proud např. vody nebo lávy. Aby mohly vznikat textury, na kterých je zřejmý tok a víření kapaliny nebo plynu, došlo k rozšíření Perlinovy šumové funkce.

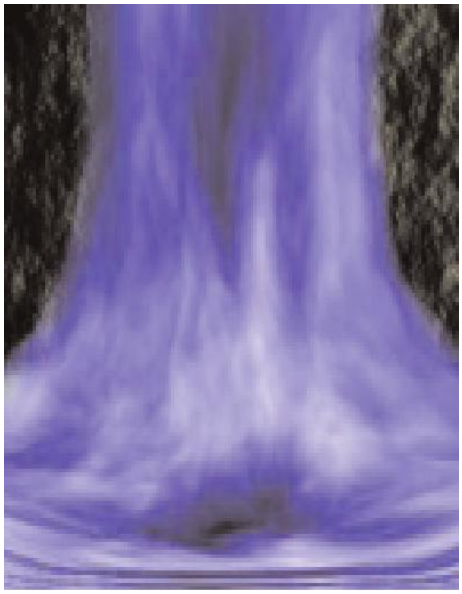
Modifikace je provedena na výsledném bodu (vektoru) po interpolaci, který vznikne výpočtem tzv. funkce lerp viz 1.3. Pro vytvoření textury je však nutné sečíst více takových výsledků. Abychom dosáhli „proudící“ textury, je třeba všechny jednotlivé výsledky po interpolaci rotovat podle času. To způsobí rotaci na místě každého výsledku, viz obr. 1.17. Všechny výsledky (vektory) na sobě nezávisejí ani před rotací ani po ní, takže výsledek vypadá stále jako Perlinova šumová funkce. Avšak čas způsobí, že textura bude působit jako „víření“, „proudění“ kapaliny nebo plynu. Jakmile sečteme všechny rotované vektory namodelujeme tím opravdový proud. [18]

Výsledek ve statické podobě může vypadat podobně jako je na obr. 1.18, ale je možné si prohlédnout i plynoucí vodopád v čase a to na: [4].



(a) vektory (b) vektory po rotaci

Obr. 1.17: Ukázka rotace vektorů [18]



Obr. 1.18: Ukázka „flow noise“ [18]

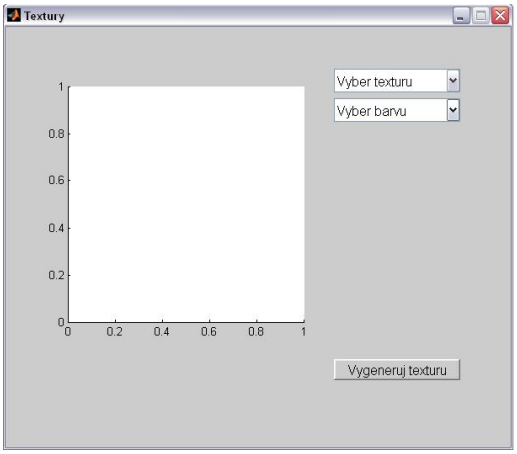
2 PRAKTICKÁ ČÁST – GENERÁTOR TEXTUR

2.1 Generátor textur

Cílem práce bylo vytvořit program, který bude generovat různé textury a bude měnit jejich vzhled při změnách některých parametrů. V této praktické části je popsán tento program, který byl vytvořen v programovacím jazyku MATLAB a byl odzkoušen ve verzi 7.11.0 (R2010b).

Na základě požadavku ukazovat příklady textur studentům, byl vytvořen program jenž se ovládá pomocí uživatelského rozhraní. Rozhraní se spustí po zadání příkazu „generace_textury()“ v příkazovém oknu MATLABu. Uživateli se otevře okno s názvem „Textury“ viz obr. 2.1, které obsahuje dvě jednoduché nabídky (nabídka textury a barvy), tlačítko pro generování textury a pole, do nějž se textury budou vykreslovat.

Při zvolení jakékoli textury, z pěti možností, v první rozbalovací nabídce, jejíž název je „Vyber texturu“, se objeví další nabídky, které slouží ke změnám parametrů vybrané textury. Parametry, které lze měnit, jsou persistence, vyhlazení, rozostření, počet oktáv nebo turbulence. Tyto hodnoty lze měnit pomocí posuvníku („slideru“), jehož velikost se vypisuje do vedlejšího políčka. Hodnotu posuvníku je zároveň možné měnit zápisem do textového pole.



Obr. 2.1: Okno *Textury*

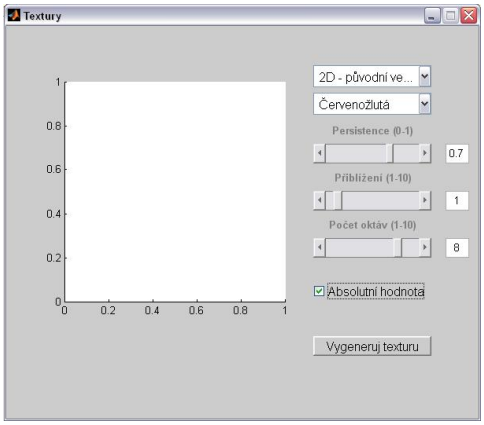
U některých textur se objevuje zaškrtávací políčko, které při aktivaci zobrazí texturu v absolutní hodnotě. Projeví se tak i záporné hodnoty viz 1.3.5.

Další obměnou textury může být změna barvy v rozbalovací nabídce „Vyber barvu“. U některých textur je velice vhodné použít jen jednu barvu, například pro texturu Dřevo je vhodná barva Hnědá, textura pak připomíná letokruhy v kmenu stromu. Jiným příkladem může být textura s názvem Mramor u které je pro dosažení odpovídajícího výsledku doporučené použít barvu Šedou. Pro některé barvy

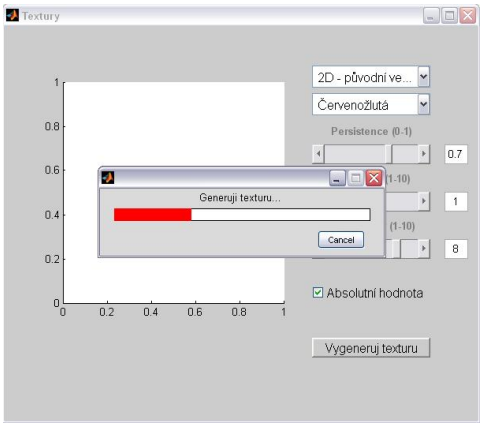
jsou použity barevné palety, které nabízí MATLAB. Patří mezi ně například barva Výchozí („default“) a Červenožlutá („hot“). Ostatní barvy jsou upravené barevné palety pomocí editoru barev („colormap editor“).

Poslední aktivní prvek, který je v okně „Textury“ je tlačítko s názvem „Vygeneruj texturu“. Po aktivaci tohoto tlačítka se vykreslí vybraná textura se zadanými parametry do bílého pole. Pokud změníme jakýkoliv parametr, neprojeví se tato změna okamžitě, musíme opět stisknout tlačítko a vygenerovat texturu znovu.

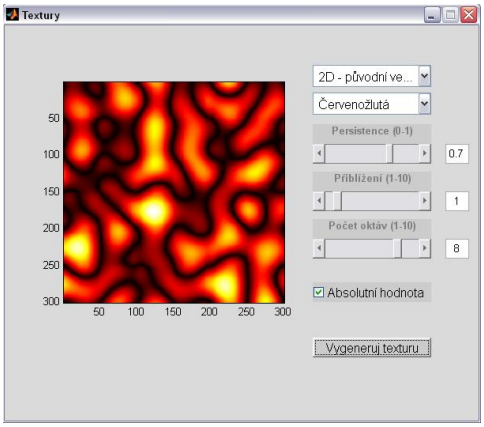
Při generování většiny textur musí program udělat množství výpočtů, které chvíli trvají. Z tohoto důvodu je použit tzv. ukazatel průběhu („waitbar“), který názorně ukazuje postup počítání (obr. 2.2b). Objeví se bezprostředně po stisknutí tlačítka „Vygeneruj texturu“. Pokud uživatel nechce čekat do konce výpočtů nebo chce ještě změnit parametry je možné generování textury přerušit za pomocí tlačítka „Cancel“. To generování textury zastaví a v poli se zobrazí jen ta část textury, která se stihla do doby aktivace tlačítka vypočítat. Postup volby parametrů a generování textury je patrný z obr. 2.2.



(a) ukázka možného nastavení



(b) generování textury



(c) vygenerovaná textura

Obr. 2.2: Okno Textury

2.2 Programy jednotlivých textur

V této části jsou popsány funkce, které jsou použity pro generování jednotlivých textur. Všechny algoritmy jsou napsány na základě původní Perlinovy šumové funkce. Jeho původní verze byla uvedena v programovacím jazyku C. Tento program byl upraven a přepsán v MATLABu. Všechny textury až na jednu, které si uživatel může zobrazit, volají stejné funkce. Vypadají různě, protože se různě vykresluje jejich výsledek.

2.2.1 Textura 1 – 1D

První textura, kterou si uživatel může zvolit v rozbalovací nabídce, představuje Perlinovu funkci v 1D. Na tomto průběhu je názorně vidět co se stane pokud se sčítá více oktáv a jak na funkci působí persistence (1.3.4).

Funkce pro tuto texturu je odvozena z původní verze Perlinovy funkce, jejíž postup byl uveden výše 1.3.2. Nejdůležitějšími výpočty v tomto programu je tzv. funkce *s_ curve* a funkce *lerp* neboli interpolace. V tomto případě se výpočty provádí pro jednorozměrné pole, tedy pro vektor. Výsledný vektor je poté zobrazen pomocí „matlabovské“ funkce *plot*.

Ukázka části kódu, kde probíhá výpočet funkce *s_ curve* a *lerp*, kde je interpolace výsledkem (*result*), který se pak již volá v dalších funkcích:

```
// výpočet funkce s_curve
sx = rx0 * rx0 * (3 - 2 * rx0);

// parametry pro výpočet interpolace
u = rx0 * g1(p(bx0));
v = rx1 * g1(p(bx1));

// výpočet interpolace
result = u + sx * (v - u);
```

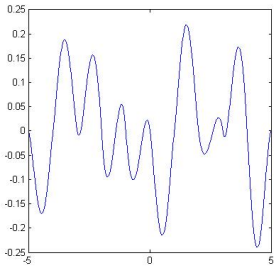
Výsledek je použit ve funkci nazvané *PerlinNoise1D*, kde je výsledek volán a upravován pomocí zadaných parametrů, kterými jsou *alpha*, jenž způsobuje změnu persistence, *beta*, která mění tzv. vyhlazení, a parametr *n*, který určuje počet oktáv. V kódu je výsledek proměnná *sum* a jsou zde i pomocné proměnné jako je *val* a *scale*:

```
for(i = 1:n)
  val = noise1(p);              //výpočet šumové funkce
  sum = sum + (val * scale);    //sčítání oktáv
  scale = scale * alpha;        //změna persistence
  p = p * beta;
end
```

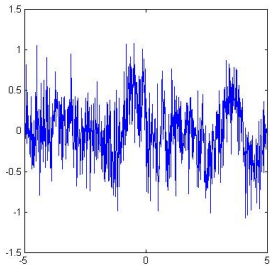
Poslední důležitou částí kódu je volání funkce *PerlinNoise1D*, uložení výsledku a jeho vykreslení:

```
//volání funkce PerlinNoise1D
y(i) = PerlinNoise1D(x(i), alpha, beta, n);
//vykreslení
plot(x, y);
```

Na obr. 2.4 je vidět postup výpočtů, vývojový diagram obsahuje nejdůležitější body v programu. Pokud u této funkce nastavíme počet oktáv na 1, persistenci a vyhlazení jakékoli, vždy bude výsledná funkce „hladká“, protože se výpočet provede pouze jednou a změna amplitudy, tedy persistence, se neprojeví. Pro názornou ukázku projevu persistence je nejlepší nastavit počet oktáv například na hodnotu 5 a pak persistenci měnit od minima do maxima a pozorovat změny viz obr. 2.3. U této textury nelze měnit barvy.



(a) persistence=0, vyhlazení=5, počet oktáv=5



(b) persistence=1, vyhlazení=5, počet oktáv=5

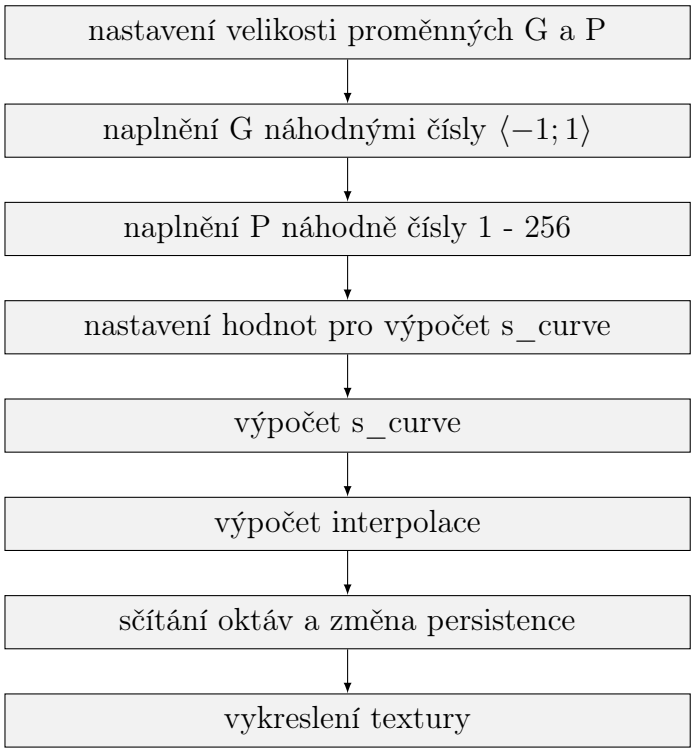
Obr. 2.3: Textura 1 – 1D

2.2.2 Textura 2 – 2D – Perlin

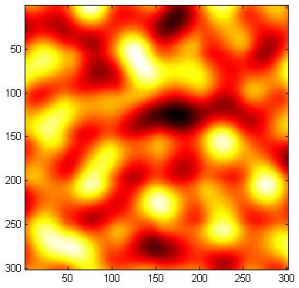
Dvojrozměrná textura je v podstatě stejná jako předchozí 1D, jen se vše počítá pro dva rozměry a výsledkem již není vektor, ale matice. Výsledná matice se vykreslí pomocí „matlabovské“ funkce *imagesc*.

Texturu můžeme opět upravovat pomocí změn různých parametrů. V tomto případě lze měnit persistenci, počet oktáv a rozostření. Při aplikaci různých barevných palet bude textura připomínat různé přírodní jevy např. oheň, pokud vybereme Červenožlutou barvu. Aby textura působila jako tráva, je třeba změnit barevnou paletu na Zelenou. Při všech těchto barvách je stále možné měnit i ostatní parametry a sledovat, kdy bude textura působit nejvíce přirozeně.

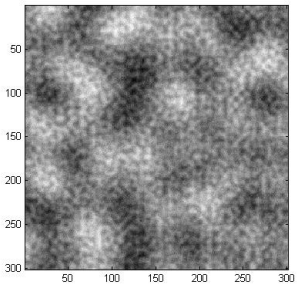
Tuto texturu můžeme také pozměnit pomocí absolutní hodnoty. Budou zobrazeny i záporné hodnoty a tím se vzhled textury opět pozmění.



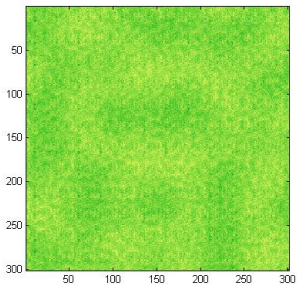
Obr. 2.4: Vývojový diagram textury 1D



(a) barva Červenožlutá, persistence=0



(b) barva Šedá, persistence=0.5

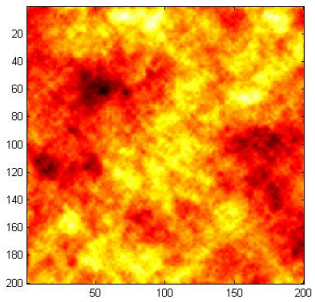


(c) barva Zelená, persistence=1

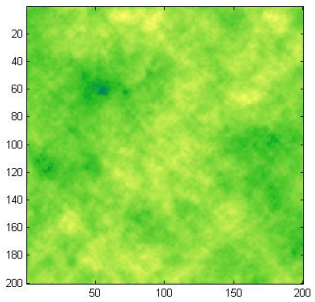
Obr. 2.5: Textura 2 – 2D – Perlin

2.2.3 Textura 3 – 2D – MATLAB

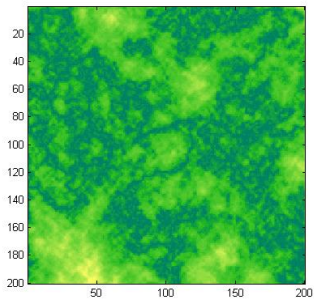
Tato textura má podobný název jako předchozí – 2D. Rozdíl mezi těmito texturami je samozřejmě ve vzhledu, ale hlavně v použitém kódu, z toho důvodu má textura přízvisko MATLAB. V tomto případě je na zjednodušení výpočtů použita „matlabovská“ funkce, která výpočet urychluje. Výsledek vypadá podobně jako předchozí, ale nedají se měnit některé parametry. Je možné si volit rozlišení obrazu a také aplikovat absolutní hodnotu.



(a) rozlišení=200, barva Červenožlutá



(b) rozlišení=200, barva Zelená



(c) rozlišení=200, barva Zelená, absolutní hodnota

Obr. 2.6: Textura 3 – 2D – MATLAB

Stejně jako v předchozím bodě vypadají textury jako oheň nebo tráva při vybrání vhodné barevné palety. Pokud zvolíme jiné barvy než Zelenou a Červenožlutou je na fantazii uživatele co v této textuře uvidí.

V použitém kódu ke generování této textury je stejně jako v předchozím počítána interpolace. V tomto případě se používá přímo „matlabovské“ funkce *interp2*. Tato funkce provede interpolaci mezi zadanými body vybranou metodou. Výchozí metodou je interpolace lineární, v tomto výpočtu se používá interpolace splajnem. A nakonec je opět aplikována funkce *imagesc* na vykreslení textury. Nejdůležitější část kódu této textury je následující:

```
while w > 3
    i = i + 1;
    //interpolace splajnem
    d = interp2(randn(w), i-1, 'spline');
    //ukládání výsledku do "s" a sčítání oktáv
    s = s + i * d(1:m, 1:m);
    //ceil - zaokrouhlení nahoru
    w = w - ceil(w/2 - 1);
end

//vykreslení textury
imagesc(s);
```

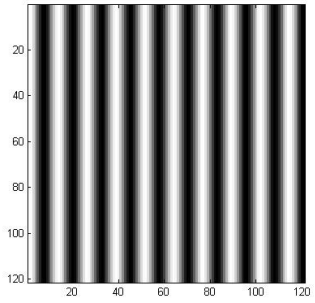
2.2.4 Textura 4 – Mramor

Již název napovídá, jak při vhodném výběru barvy bude tato textura vypadat. Opět jde o stejný algoritmus a je použit stejný kód jako u druhé textury, ale s tím rozdílem, že je aplikována funkce *mramor*, které se věnuje kapitola 1.3.5. Pro správné vyobrazení je však tento výraz upraven následujícím způsobem:

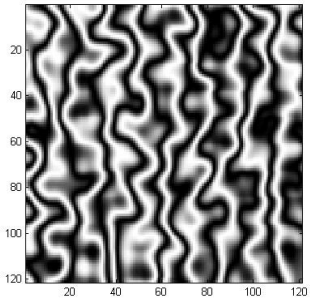
```
result(j, i) = 0.5*(1 + sin(10*x(i) + t * PerlinNoise2D(x(i)/300, y(j)
    /300, alpha, beta, n)));
```

V kódu je *result* výsledek funkce, který se pak vykreslí pomocí funkce *imagesc*. Ve vzorci 1.7 chybí násobení deseti proměnné x. Násobení proměnné ve funkci sinus znamená „zrychlení“ průběhu této funkce. V tomto případě to dopomůže k tomu, že se zobrazuje více pruhů a ne pouze jeden. Parametr turbulence je zde zastoupen proměnnou t, uživatel si tuto proměnnou volí. Funkce *PerlinNoise2D* je v tomto případě funkcí *snoise*. Proměnné *alpha*, *beta* a n mají následující hodnoty: alpha=2, beta=2, n=10. Proměnné x a y jsou upravovány hodnotou 300. Toto číslo bylo vybráno po několika pokusech se změnou čísla a sledováním výsledné textury tak, aby bylo možné na výsledku nejlépe pozorovat změnu turbulence.

Tato textura slouží především k názorné ukázce vlivu změny turbulence. Pokud také zvolíme vhodnou barvu, nejlépe Šedou nebo Černou, a zároveň bude zvolena turbulence větší jak 0, bude textura připomínat strukturu mramoru.



(a) turbulence=0, barva Šedá



(b) turbulence=15, barva Šedá

Obr. 2.7: Textura 4 – Mramor

2.2.5 Textura 5 – Dřevo

Poslední textura v rozbalovací nabídce „Vyber texturu“ má připomínat letokruhy stromu. Opět je základ algoritmu v původní Perlinově šumové funkci a pouhý rozdíl od předešlých textur je způsob vykreslení. Stejně jako čtvrtá textura má i tato prezentovat vliv turbulence. Je použito upraveného výrazu 1.7:

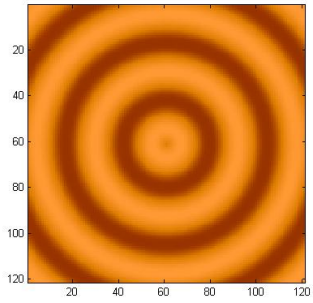
```
result(j, i) = 0.5*(1 + sin(5 * sqrt(x(i)^2+y(j)^2) + t * PerlinNoise2D
    (x(i)/divisor, y(j)/divisor, alpha, beta, n)));
```

Tentokrát se proměnná x násobí pouze pěti, aby nevznikalo tolik čar a obrázek byl přehledný. Jak je vidět nenásobí se pouze x, ale celý výraz, který je pod odmocninou. Výsledkem je velikost poloměru kružnice, ve kterých se čáry vykreslují. Vychází se ze základního matematického vzorce, pro poloměr kružnice r platí:

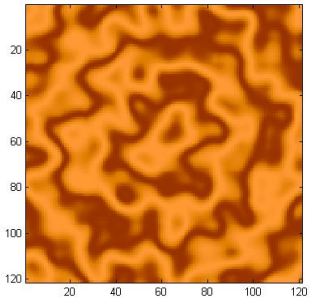
$$r^2 = x^2 + y^2. \tag{2.1}$$

Opět se využívá stejné funkce *PerlinNoise2D*, která se volá s parametry alpha=2, beta=2 a n=10, hodnotu *divisor* si uživatel zadává pod názvem rozostření pomocí posuvníku stejně jako hodnotu t tedy turbulence.

Opět je možné měnit barevné palety podle přání. Aby textura připomínala letokruhy stromu, je doporučeno tuto texturu generovat v barvě Hnědá. Pokud zvolíme jinou barvu je na uživateli, jak bude texturu vnímat.



(a) turbulence=0, rozostření=300, barva Hnědá



(b) turbulence=10, rozostření=300, barva Hnědá

Obr. 2.8: Textura 5 – Dřevo

2.2.6 Doplňující informace k použitým programům

V jednotlivých programech byly použity i konstanty, které byly empiricky zjištěny při vytváření programu na základě vzhledu výsledné textury. První takovou hodnotou je konstanta, která upravuje velikost parametru turbulence. Používá se v programu pro generování textury Mramor a Dřevo. Jedná se o číslo 1000, které slouží jako dělitel vstupního čísla od 0 do 20, které uživatel mění pomocí posuvníku. Další konstantou je číslo 10 a 5, které se používá opět v kódu pro Mramor a Dřevo. Tato hodnota slouží k určení počtu čar v textuře, byla vysvětlena již v jednotlivých popisech textur.

Ve funkcích Mramor a Dřevo se volá funkce *PerlinNoise2D* se vstupními parametry alpha=2, beta=2, n=10 z toho důvodu, že při tomto nastavení lze v textuře dobře pozorovat změnu turbulence a rozostření. Tyto velikosti hodnot byly určeny po několika pokusech a zhodnocení výsledného vzhledu textury.

Jako vstupní hodnoty funkce *PerlinNoise2D* jsou také x a y, které se při počítání každého bodu textury mění:

```
for(i = 1:length(x))
  for(j = 1:length(y))
    result(j, i) = PerlinNoise2D(x(i), y(j), alpha, beta, n);
  end
end
```

Proměnné x a y jsou vektory naplněné čísly od -3 do 3 po kroku 0,02. Velikostí vektoru určujeme velikost výsledné matice, která se vykresluje. Její rozměry tedy jsou 301×301 bodů. Tím nastavíme dostatečné rozlišení textury a dosáhneme přiměřeně rychlého výpočtu. Tyto rozměry se používají v texturách 2D – Perlin, Mramor a Dřevo. V textuře 1D se průběh funkce vykresluje do pole 1001×1001 bodů. Rozlišení textury 2D – MATLAB si může uživatel měnit přímo v uživatelském rozhraní od 100×100 bodů v obrazu po 200×200.

2.3 Porovnání procedurálních technik

2.3.1 Obecné porovnání

Pro generování procedurálních textur bylo v této práci použito Perlinovy šumové funkce. Ta není jediná ve skupině procedurálních technik a zároveň ve skupině šumu gradientů v mřížce viz 1.1.1. Do této skupiny patří jak šum gradientů („gradient noise“) tak šum hodnot („value noise“ – jedná se o funkci šumu, která vznikne výpočtem interpolace mezi pseudonáhodnými čísly [11]) a jejich kombinace – výsledkem je součet obou šumů.

Pokud tyto techniky navzájem porovnáme, zjistíme, že jsou každá jinak výpočetně náročná a zároveň se liší jejich visuální výsledek. Nejméně výpočtů se provádí při generování textury „value noise“ a proto bude nejrychlejší, pomalejší je Perlinův šum, tedy „gradient noise“, a z toho vyplývá, že součet obou, tedy jejich kombinace, bude nejpomalejší, ale dosáhneme nejlepších výsledků co se týče vzhledu textury. Porovnání rychlostí je možné vyčíst z práce [11].

2.3.2 Porovnání použitých technik

Praktická část práce se zabývala generováním textur pouze pomocí Perlinova šumu. Pro splnění zadání a vytvoření základních textur mramoru a dřeva byl pro názorné ukázání vlivu turbulence a persistence zvolen původní algoritmus Perlinovy funkce. Pro srovnání byla vytvořena i textura 2D – MATLAB, jejíž generování je rychlejší než 2D – Perlin. Rozdíl rychlosti je způsoben využitím „matlabovské“ funkce.

Pro ukázku bylo provedeno měření rychlostí generování textur 2D – Perlin a MATLAB. Toto měření proběhlo na notebooku, na kterém byl při měření spuštěn pouze program MATLAB a Microsoft Excel, do kterého se zapisovaly výsledky. Doba generování textur se měřila pomocí funkcí *tic* a *toc* v MATLABu. Bylo provedeno 20 měření a z nich je v tab. 2.1 uveden průměr. Nastavení textur při tomto měření bylo vybráno tak, aby byl vzhled textur podobný viz obr. 2.9:

- 2D – Perlin: persistence=$0,5$, rozostření=3, počet oktáv=5, Výchozí barva,
- 2D – MATLAB: rozlišení=150, Výchozí barva.

Z tab. 2.1 lze vyčíst, že průměry rychlostí se liší přibližně o $11,4136$ sekund. Znamená to, že použití „matlabovské“ funkce pro výpočet interpolace může generování textur výrazně urychlit.

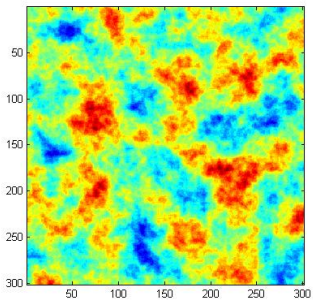
Tab. 2.1: Výsledky měření rychlostí

textura	průměr rychlostí [s]
2D – Perlin	11,5557
2D – MATLAB	0,1421
rozdíl rychlostí [s]	11,4136

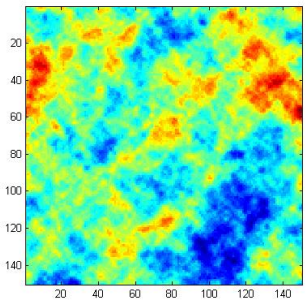
Pokud v kódu pro texturu 2D – MATLAB uděláme pouze malou úpravu, můžeme generovat i texturu mramoru či dřeva a rychlost výpočtu bude stále o něco rychlejší než při použití původního Perlinova algoritmu. Do kódu bychom mohli přidat například tuto část:

```
for (j = 1:m)
    for (i = 1:m)
        result(i,j)= 0.5*(1 + sin(30*x(i) + t *s(i,j)));
    end
end
```

Důvodem pro použití původního Perlinova šumu byla také možnost řádně nastudovat tento algoritmus. V dnešní době je ale tato verze vystřídána za implementaci z roku 2002 viz 1.5. Pro vylepšení uvedeného programu v této práci by bylo možné použít jinou interpolaci nebo omezit počet gradientů. K zrychlení by vedlo zavedení „matlabovských“ funkcí do kódu anebo využití „value noise“, ten by ale mohl způsobit zhoršení kvality textury. Vždy je tedy třeba se při výběru procedurální techniky rozhodovat podle požadavků – zda nám záleží na rychlosti anebo kvalitě.



(a) 2D – Perlin



(b) 2D – MATLAB

Obr. 2.9: Generované textury při měření rychlostí

3 ZÁVĚR

Tato práce obsahuje stručný úvod do procedurálních textur. Stěžejní byla procedurální technika Perlinova šumu. Jak již bylo uvedeno tato funkce vznikla z podnětu vytvořit přirozeně vypadající texturu pomocí počítače. V teoretické části je algoritmus Perlinovy šumové funkce popsán a vysvětleny jsou také nejdůležitější parametry – turbulence a persistence, které výslednou texturu výrazně ovlivňují. Dvě z podkapitol se také zabývají vylepšením Perlinovy funkce. Důvod vzniku těchto vylepšení byl především ve snaze zredukovat výpočty pro několik dimenzí a rovnoměrně rozložit gradienty.

V praktické části je uveden program, který je napsán v programovacím jazyku MATLAB. Tento program se ovládá přes uživatelské rozhraní, kde je možné měnit určité parametry a tak ovlivňovat vzhled generované textury. Tento program je vytvořen na základě předpokladu, že bude sloužit pro ukázku studentům. Je zde tedy použito algoritmu původní verze Perlinovy šumové funkce, u které je nejvíce názorná změna persistence a turbulence. Jedním z důvodů použití této funkce také je, že pouhou úpravou způsobu vykreslování dosáhneme jiného výsledku. Pomocí programu je možné vygenerovat dvě velice podobné textury jenž jsou pod názvem 2D – Perlin a 2D – MATLAB. Oba způsoby jsou uvedeny proto aby bylo jasné, že lze použít jak Perlinova algoritmu tak i kódu, který používá funkce MATLABu a tím se některé výpočty urychlují. Porovnání rychlostí je v poslední podkapitole, kde lze vyčíst i průměrné rychlosti generování těchto textur.

Použitý algoritmus pro generování textur je poměrně náročný na výpočet, proto je použit ukazatel průběhu generování textury. Možným řešením pro zrychlení by mohlo být spojení některých funkcí do jednoho m – souboru a pro výpočet interpolace použít funkci *interp2*, jako je tomu například v kódu textury 2D – MATLAB. Dalším krokem by mohla být implementace interpolace pátého řádu, jak je uvedeno v podkapitole o vylepšení Perlinovy funkce. Pro úpravy by bylo možné použít i zcela jiný algoritmus např. „value noise“ pro zrychlení anebo součet Perlinovy funkce s „value noise“ pro zkvalitnění . Záleží na tom, co uživatel požaduje. Rychlost výpočtů lze např. ovlivnit i změnou počtu oktáv a rozlišení. Jinou variantou techniky pro generování textur mohou být fraktály, kterým je věnována jedna celá podkapitola. Fraktálová technika má ale nevýhodu – probíhá zde spousta náročných výpočtů a tak není vhodná pokud nám záleží na rychlosti.

Program, jenž vznikl v rámci práce je vytvořen v MATLABu, ovšem není jediný, který je pro generování textur používaný. Mnoho grafiků dnes používá program POV-Ray, který umožňuje vykreslování trojrozměrných scén. Perlin často používá jazyk Java nebo C a někteří počítačoví grafici pracují s GNU Octave, který je velice podobný MATLABu.

Na práci by mohlo navazovat další vylepšování stávajícího programu. Dalo by se zde doplnit více textur, které by byly generovány pomocí jiných procedurálních technik např. „value noise“ nebo „simplex noise“. Do uživatelského rozhraní by bylo možné dodat další posuvníky např. pro změnu rozlišení. Praktickým doplňkem by byla možnost uložení výsledné textury do souboru. Pro porovnání by zde mohl být výběr použité interpolace – lineární nebo pomocí splajnu. Uživatelské prostředí by se spolu s těmito změnami mohlo také inovovat.

LITERATURA

[1] „Benoit B. Mandelbrot“ [online]. 16. 3. 2010 [cit. 2011-12-09]. Benoit B. Mandelbrot. Dostupné z URL: <http://users.math.yale.edu/∼bbm3/index.html>

[2] BYŠKA, Jan. *Perlinův šum a jeho aplikace* [online]. [s.l.], 2009. 38 s. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Dostupné z URL: <http://is.muni.cz/th/207879/fi_b/bakalarka-text.pdf>

[3] COOK, Robert L. a Tony DEROSE. *Wavelet Noise* [online]. 24. vyd. NY, USA: ACM New York, 2005[cit. 2012-04-10]. Dostupné z URL: <http://dl.acm.org/citation.cfm?id=1073264>

[4] Examples of Flow Noise. *NYU Media Research Lab* [online]. 2004 [cit. 2012-04-29]. Dostupné z URL: <http://mrl.nyu.edu/flownoise/>

[5] FlowNoise. *NYU Media Research Lab* [online]. 2001 [cit. 2012-04-29]. Dostupné z URL: <http://mrl.nyu.edu/∼perlin/flownoise-talk/#12>

[6] Fractals for Fun. *Galileo´s Pendulum: A Science Blog by Matthew Francis* [online]. 31.1.2012. [cit. 2012-05-25]. Dostupné z URL: <http://galileospendulum.org/2012/01/31/fractals-for-fun/>

[7] GUSTAVSON, Stefan. Simplex noise demystified. [online]. 2005-03-22 [cit. 2012-04-30]. Dostupné z URL: <http://webstaff.itn.liu.se/∼stegu/simplexnoise/simplexnoise.pdf>

[8] HECKBERT, Paul S. Survey of texture mapping. In „IEEE Computer Graphics and Applications“ [online]. [s.l.] : [s.n.], 1986 [cit. 2011-12-09]. Dostupné z URL: <http://www.cs.cmu.edu/∼ph/texsurv.pdf>

[9] HEJTMÁNEK, Martin. Teorie chaosu. *Martin Hejtmánek*[online]. 2009 [cit. 2012-05-25]. Dostupné z URL: <http://kmlinux.fjfi.cvut.cz/∼hejtmmar/chaos.php>

[10] *Ken´s Blog* [online]. 30. 10. 2011 [cit. 2011-10-30]. Ken´s Blog. Dostupné z URL: <http://blog.kenperlin.com/>

[11] KUČERA, Ondřej. *KNIHOVNA PRO VÝPOČET ŠUMŮ POUŽÍVANÝCH V PROCEDURÁLNÍM TEXTUROVÁNÍ* [online]. Brno, 2007 [cit. 2012-05-15]. Dostupné z URL: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=5625>. Diplomová práce. VUT v Brně. Vedoucí práce Ing. Adam HEROUT, Ph.D.

[12] LAGAE, A., S. LEFEBVRE, R. COOK, T. DEROSE, G. DRETTAKIS, D.S. EBERT, J.P. LEWIS, K. PERLIN a M. ZWICKER. A Survey of Procedural Noise Functions. *COMPUTER GRAPHICS forum* [online]. 2010, 1–20 [cit. 2012-04-10]. Dostupné z URL: <https://lirias.kuleuven.be/bitstream/123456789/278824/1/LLCDDELPZ10SPNF.pdf>

[13] *Making Noise* [online]. 19. 2. 2006 [cit. 2011-11-17]. Controlled random primitive. Dostupné z URL: <http://www.noisemachine.com/talk1/6.html>

[14] *Making Noise* [online]. 19. 2. 2006 [cit. 2011-12-14]. The 8 neighbors in three dimensions. Dostupné z URL: <http://www.noisemachine.com/talk1/16.html>

[15] MAŠEK, Martin. *Fraktálové kapacitory* [online]. [s.l.], 2008. 40 s. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická. Dostupné z URL: <http://www.ieee.cz/mtt/soutez08/Masek.pdf>

[16] *NYU Media Research Lab* [online]. 19. 10. 2011 [cit. 2011-10-30]. Ken Perlin. Dostupné z URL: <http://mrl.nyu.edu/∼perlin/>

[17] *NYU Media Research Lab* [online]. 26. 7. 2011 [cit. 2011-10-30]. Brief Biography, Ken Perlin. Dostupné z URL: <http://mrl.nyu.edu/∼perlin/doc/bio.html>

[18] PERLIN, Ken a Fabrice NEYRET. Flow Noise. In: *Siggraph Technical Sketches and Applications* [online]. 2001 [cit. 2012-04-29]. Dostupné z URL: <http://www-evasion.imag.fr/Publications/2001/PN01>

[19] PERLIN, Ken. Implementing Improved Perlin Noise. FERNANDO, Randima. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics* [online]. Addison-Wesley Professional, 2004, 16.6.2009 [cit. 2012-04-15]. ISBN 0321228324. Dostupné z URL: <http://developer.nvidia.com/node/106>

[20] PERLIN, Ken. *Making Noise* [online]. 19. 2. 2006 [cit. 2011-10-30]. Making Noise. Dostupné z URL: <http://www.noisemachine.com/talk1/index.html>

[21] *Perlin Noise* [online]. 7. 12. 1998 [cit. 2011-10-30]. Perlin Noise. Dostupné z URL: <http://freespace.virgin.net/hugo.elias/models/m_perlin.htm>

[22] TIŠNOVSKÝ, Pavel. Perlinova šumová funkce a její aplikace. *Fraktály v počítačové grafice* [online]. 20. 3. 2007, 72., [cit. 2011-10-30]. Dostupný z URL: <http://www.root.cz/clanky/perlinova-sumova-funkce-a-jeji-aplikace/>

[23] TIŠNOVSKÝ, Pavel. Procedurální textury a role turbulence. *Vykreslujeme 3D scény s POV-Ray* [online]. 24. 6. 2008, 17., [cit. 2011-10-30]. Dostupný z URL: <http://www.root.cz/clanky/proceduralni-textury-a-role-turbulence/>

[24] TIŠNOVSKÝ, Pavel. *Seriál Vykreslujeme 3D scény s POV-Ray* [online]. 2008 [cit. 2012-05-12]. Dostupné z URL: <http://www.root.cz/serialy/vykreslujeme-3d-sceny-s-pov-ray/>

[25] WELLONS, Christopher. *Null program* [online]. 20. 11. 2007 [cit. 2011-11-17]. Noise Fractals and Clouds. Dostupné z URL: <http://nullprogram.com/blog/2007/11/20/>

[26] „What is? com“ [online]. 5. 7. 2008 [cit. 2011-12-10]. What is noise?. Dostupné z URL: <http://whatis.techtarget.com/definition/0,,sid9_gci212667,00.html>

[27] ZHOU, Dongxiao. What is a Texture?. *Texture Analysis and Synthesis using a Generic Markov-Gibbs Image Model* [online]. 22 .2. 2006 [cit. 2012-04-10]. Dostupné z URL: <http://www.cs.auckland.ac.nz/~georgy/research/texture/thesis-html/node5.html>

[28] ŽÁRA, Jiří, et al. *Moderní počítačová grafika*. druhé, rozšířené a přepracované vydání. Brno: Computer Press, 2004. 609 s. ISBN 80-251-0454-0.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

1D jednorozměrný – one dimension

2D dvourozměrný – two dimensions

3D trojrozměrný – three dimensions

POV-Ray Persistence of Vision Ray Tracer

RAM Paměť s přímým přístupem – Random-access memory

A CD

A.1 Program pro generování textur

Všechny kódy a vše potřebné pro spuštění programu je ve složce s názvem Program. Při otevření této složky v MATLABu je pro spuštění samotného programu nutné v příkazovém oknu („Command Window“) zadat příkaz „generace_textury“. Program byl odzkoušen ve verzi 7.11.0 (R2010b).

A.2 Text bakalářské práce

Celý text bakalářské práce se nachází ve složce s názvem Text.