Ročníkový projekt

# Procedurální generátor textur

Nikol Škvařilová

IT3B

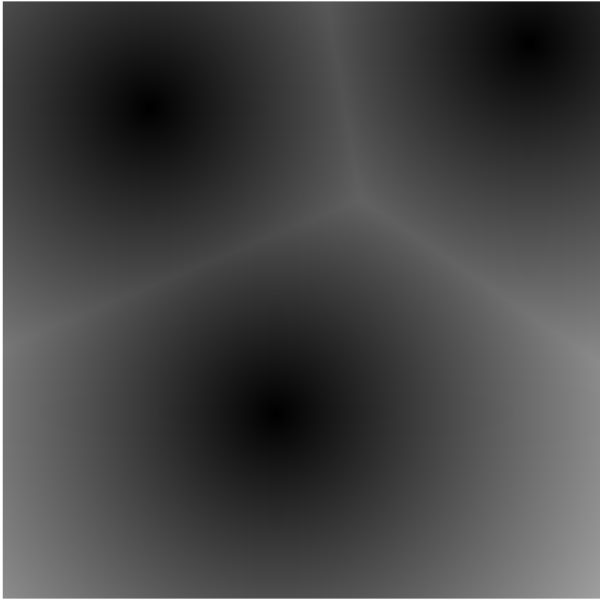
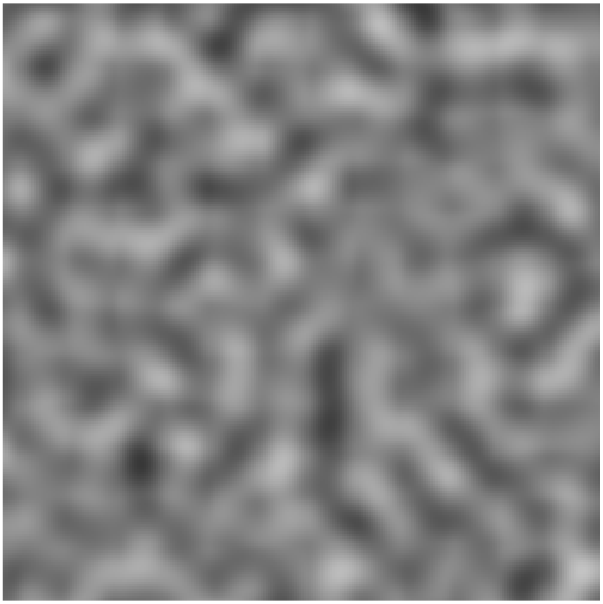
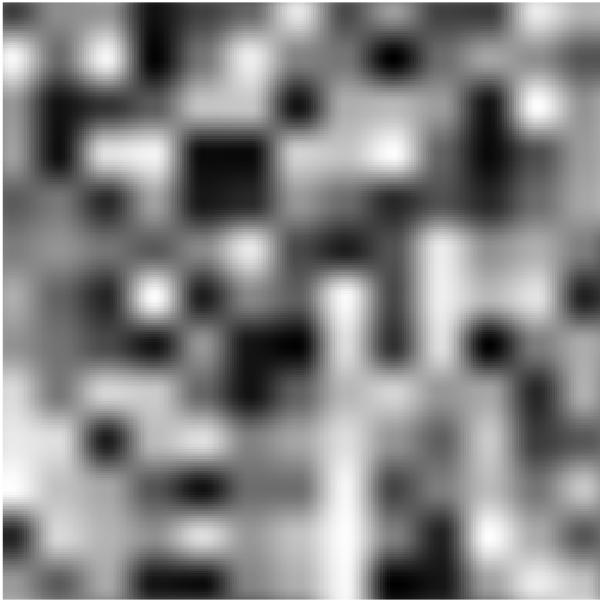
# O projektu

Procedurální generování představuje sadu algoritmů, které se využívají pro vytváření přirozeně vypadajících objektů/ploch/textur.Umožňují nám generovat textury které vypadají například jako kamení nebo dřevo bez toho, aniž bychom použili již předvytvořené/namalované.

Uplatnění nalezneme například v herním a filmovém průmyslu. Využívá se procedurálně generovaný terén – tedy takový, který byl vygenerován algoritmem a ne vytvořen artistou. Dále se s nimi můžeme setkat v programech pro tvorbu grafiky – např. Blender.

Mým cílem bylo vizualizovat 3 hlavní algoritmy, které se na tomto poli používají. Seznámit se s tím, jak fungují, jak se s nimi pracuje. Vytvořit přívětivé uživatelské prostředí, aby si i uživatel mohl představit, jak takové věci fungují a jak velké je jejich využití.

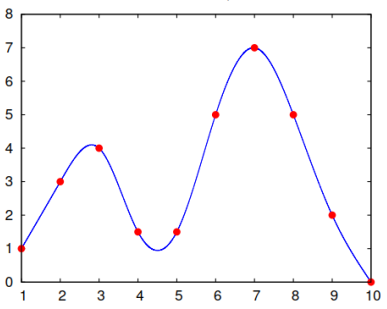
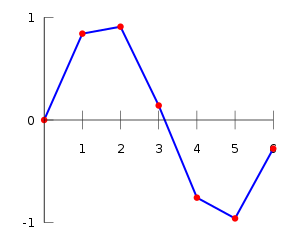
Kromě toho je možné vygenerované textury vyexportovat a použít jako obrázkovou texturu.



# Algoritmy

Abychom vytvořili náhodně vypadávající textury, potřebujeme do toho nějakým způsobem náhodu zakomponovat. 100% náhoda ale není vhodná, neb výsledek by vypadal jako šum z televize. My potřebujeme, aby tyto náhodné hodnoty byly provázané a vytvářely jakýsi přechod.

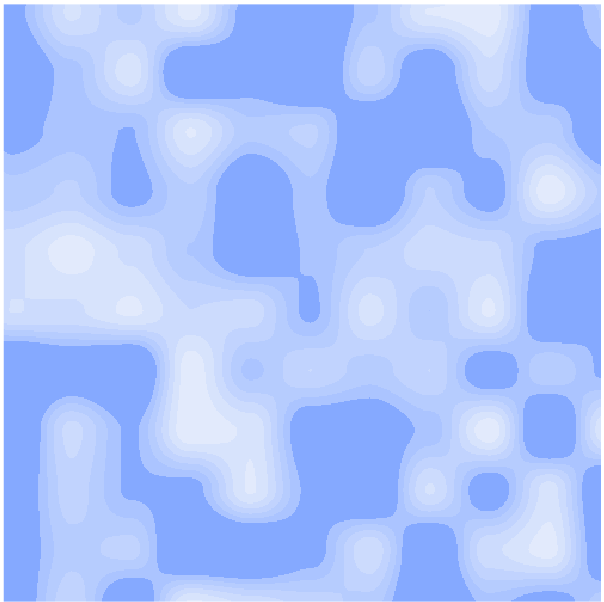
K tomu se využívá pseudonáhodný generátor čísel a princip interpolace mezi body. Pseudonáhodný generátor využijeme pro generování čísel odpovídajících různým pozicím v obrázků (pozice x, y). Interpolace nám umožňuje odhadnout hodnotu v intervalu, pro jehož krajní body známe hodnoty funkce.



*Lineární a kubická interpolace*

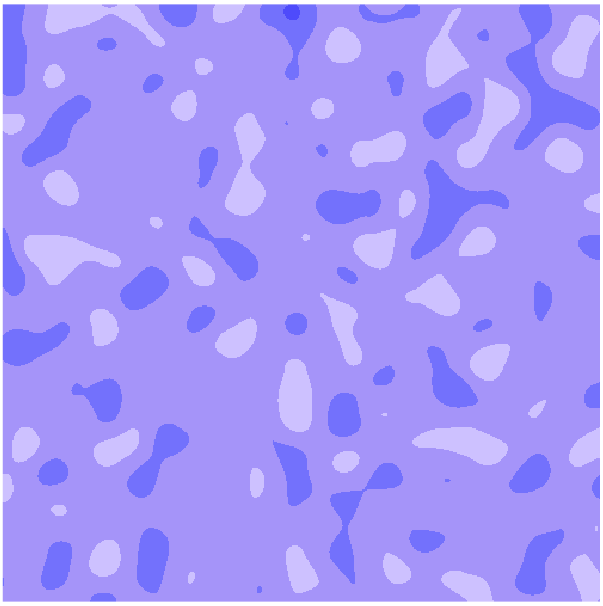
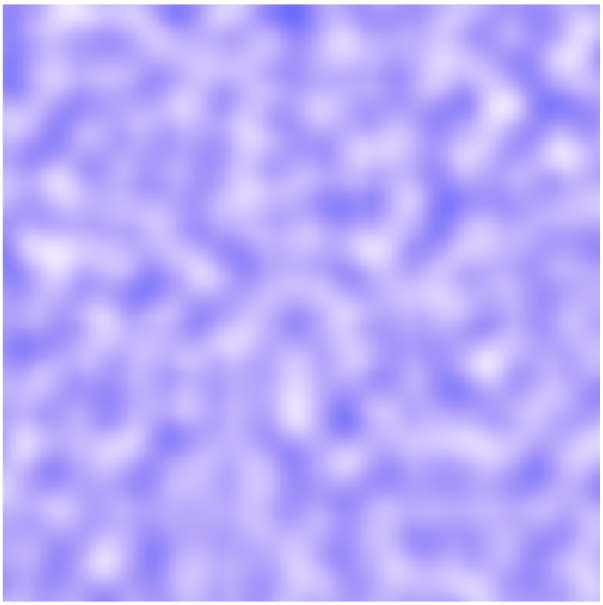
## Hodnotový šum (value noise)

Hodnotový šum byl prvním svého druhu. Při bližším pohledu si všimneme, že se zdá poněkud kostičkovaný, vyplněný obdelníky.



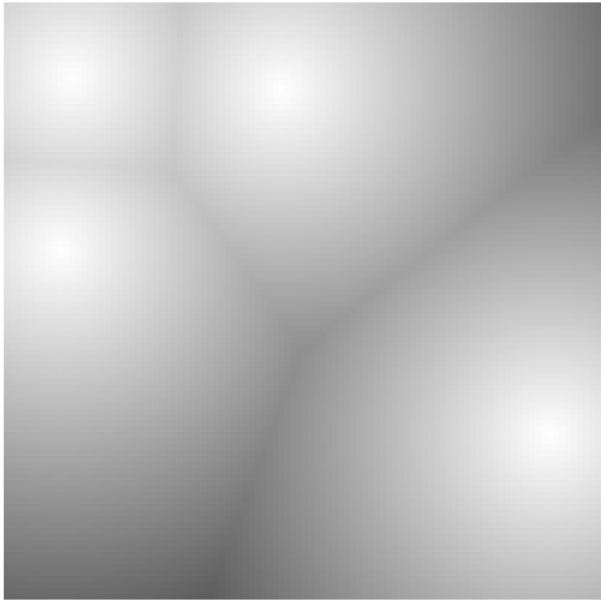
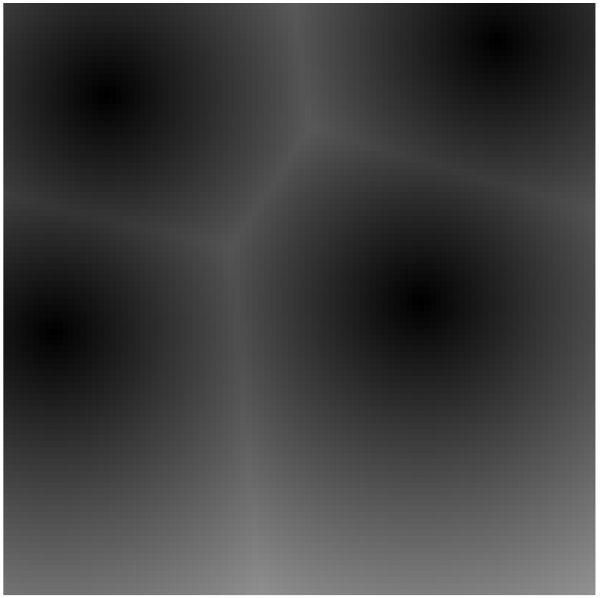
## Přechodový šum (value noise)

Vylepšená verze hodnotového šumu. Jeho hodnoty se soustředí více v intervalu 0.2 - 0.8 (nezachází tak do příliš malých/velkých hodnot).



## Buněčný šum (cellular noise)

Vytvořen tak, aby připomínal vzhled buněk.



### Princip

V prostoru obrázku je vybráno n bodů. Poté, pro každý pixel se změří nejmenší vzdálenost ke každému z nich a vybere se ta nejmenší. Tato hodnota je pak barvou daného pixelu.

Body se vybírají náhodně a lze změnit jejich počet. Čím více bodů, tím tmavší obrázek bude, protože vzdálenosti pixelů k bodům budou menší → menší hodnota → tmavší barva.

* Je také možné udělat přesný opak, aby pixely co jsou blíže byly nejsvětlejší.

# Aplikace

Aplikace se sestává z ploch, kde se ukazuje vygenerovaný šum, nastavení šumu, nastavení barev.



Po nastavení všech parametrů a stisku tlačítka generovat se nám vygeneruje náhled, jak šum vypadá.

## Parametry šumu

* **Velikost obrázku** – Ovlivňuje kolik pixelů se bude zpracovávat + jak velký bude vyexportovaný obrázek.
* **Přiblížení (noise scale)** – Určuje, jak moc máme šum přiblížit. Děje se tak pomocí zmenšování kroku mezi jednotlivými pixely. Tedy pokud je se naše hodnota přiblížení jmenuje **scale**, místo toho aby funkce šumu dostala hodnoty [x, y], [x + 1, y], apd., dostane [x \* scale, y \* scale, [(x + 1) \* scale, y \* Scale] … Krok mezi jednotlivými pixely je tak menší.
* **Nastavení seedu (seed settings)** – Když je náhodný, vygenerovaný šum vypadá pokaždé jinak. Funguje to tak, že místo toho aby funkce šumu dostávala pokaždé stejné hodnoty ([0, 0], [0, 1], …), začne s počátkem jiným, než [0, 0]. Tedy pokud bychom si seed nastavili na 100, počátkem bude [100, 100], a budeme se dále pohybovat následovně [100, 101], [100, 102]. A tak se šum počítá pro jinou pozici. U buněčného šumu tento princip využít nemůžeme – proto tedy jen náhodně generujeme body v prostoru.
* **Typ šumu (type of noise)** – Umožňuje nám vybrat si, jaký typ šumu chceme použít. Jak vypadají a jaké jsou mezi nimi rozdíly, je ukázáno výše.
* **Barevný mód (color mode)** – Můžeme si zvolit, zda-li chceme, aby výsledek byl jen černobílý, nebo i barevný (a tedy použijí se barvy co jsou v listu na pravé straně).
* **Použití přechodu (gradients between colors)** – Princip přechodů je popsán níže.
* **Počet bodů (number of cells)** – Pouze pro buněčný šum; udává, kolik náhodných bodů se má vygenerovat.

## Barvy

Barvy zadáváme v **HEX kódu**, a společně s nimi přidáváme **hranici**,která je v rozmezí <0; 1>. Tato hranice slouží k tomu, aby když nám ta funkce co počítá šum vrátí hodnotu ve stejném rozmezí, abychom mohli odvodit, jaká barva to má být.

Tedy ze screenshotu můžeme říct, že pokud je vygenerovaná hodnota

* < 0, barva bude #CB997E (barva s touto hranicí bude sloužit pouze pro vytvoření gradientu)
* < 0.25, bude barva #DDBEA9
* < 0.5, bude barva #FFE8D6
* apd.

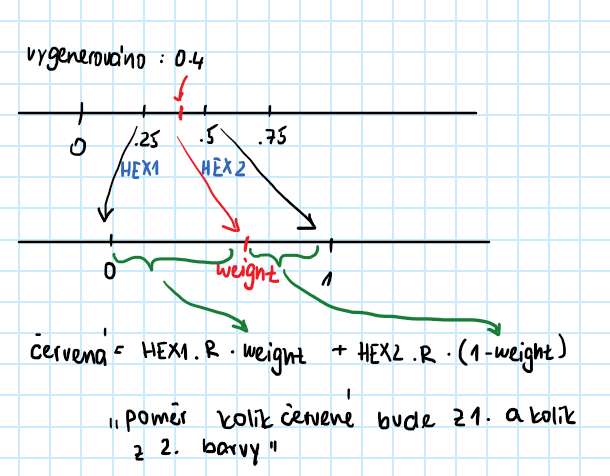
### Přechody

Můžeme si zaškrtnout, že chceme, aby mezi našimi barvami byl přechod. Přechod pro každou barvu se vypočítá následovně

1. Vypočítáme **hodnotu šumu** (ta je v rozmezí <0; 1>)
2. Zjistíme, která **barva + hranice je před a po této hodnotě** (příklad: kdyby nám šum vypočítal hodnotu 0.4, uložíme si, že první barva je ta pro hranici 0.25 (#DDBEA9), a další barva je ta pro hranici 0.5 (#FFE8D6)). Hranice si uložíme také. Poté tyto hranice společně s hodnotou šumu namapujeme do rozmezí <0; 1>. Tuto hodnotu si pojmenujme **weight**.
3. Vypočítáme výslednou hodnotu:

R1 \* weight + R2 \* (1 - weight) // “Dej tam weight \* červené první barvy, a zbytek dej červenou druhé barvu

// Postup zopakujeme pro zbylé barvy (zelená a modrá)



# Závěr

Projekt se povedl tak, jak jsem chtěla. Seznámila s algoritmy, které se používají na procedurální generování a mám v plánu s nimi i další projekty. I samotné plánování projektu bylo velmi dobrou zkušeností.

Kromě toho jsem si vyzkoušela vytváření, upravování i ukládání bitmapových obrázků v C# a obecně grafické zobrazování algoritmů.