



FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z KIV/PRO
**Porovnání algoritmů Value Noise,
Perlin Noise a Worley Noise**

Obsah

1	Zadání	2
2	Existující metody	3
2.1	Value Noise	3
2.2	Perlin Noise	3
2.3	Worley Noise	3
3	Zvolené řešení	4
3.1	Sjednocené rozhraní	4
3.2	Pseudokód algoritmů	4
4	Experimenty a výsledky	7
4.1	Výsledné obrázky	7
4.2	Porovnání	9
4.2.1	Value Noise	9
4.2.2	Perlin Noise	9
4.3	Worley Noise	9
4.4	Tabulka výsledků	9
5	Závěr	10

1 Zadání

Cílem práce je implementace algoritmů pro procedurální generování dvourozměrného šumu a jejich následné porovnání. Hlavní metodou je algoritmus **Perlin noise**, ke kterému jsou pro účely porovnání implementovány algoritmy **Value noise** a **Worley (cellular) noise**.

Všechny algoritmy budou implementovány v jazyce Python a porovnávány při stejných parametrech (seed, měřítko, rozlišení). Výstupem budou grayscale obrázky, analýza vlastností jednotlivých noise metod a jejich vzájemné porovnání.

2 Existující metody

2.1 Value Noise

Value noise je jednoduchý procedurální šum, ve kterém je každému rohu mřížky přiřazena pseudonáhodná skalární hodnota. Hodnota šumu uvnitř buňky se pak získává interpolací těchto čtyř hodnot, obvykle pomocí lineární nebo vyhlazené fade funkce. Výsledkem je hladký, vizuálně jednoduchý "kostkovitý" šum, který zachovává základní mřížkovou strukturu.

Výhodou value noise je jeho rychlá a snadná implementace a nízká výpočetní náročnost. Nevýhodou je omezená vizuální komplexnost a to, že výsledný šum může působit uměle nebo příliš pravidelně. Value noise se používá spíše jako úvodní metoda nebo základ pro složitější algoritmy, které z něj vycházejí a její základní podoba se v praxi moc nevyužívá.

Za pomoci fBm (fractional Brownian motion) lze základní value noise dále rozšířit a výrazně zvýšit jeho vizuální komplexitu. fBm je metoda, která kombinuje více „oktáv“ šumu — tedy více vrstev value noise — kde každá oktáva má vyšší frekvenci a nižší amplitudu než předchozí. [2]

2.2 Perlin Noise

Perlin noise, který navrhl Ken Perlin v roce 1985, je gradientový šum. Místo náhodných hodnot přiřazuje každému rohu buňky náhodný gradientový vektor. Hodnota šumu se vypočítá jako skalární součin tohoto vektoru a směru k vyhodnocovanému bodu. Výsledný šum je následně plynule interpolován pomocí speciální vyhlazovací funkce.

Hlavní výhodou Perlinova šumu je jeho přirozený a spojitý vzhled, díky kterému se používá pro generování terénů, oblaků, ohně nebo organických textur. Nevýhodou je složitější implementace a možnost vzniku jemných mřížkových artefaktů. I přesto je Perlin noise dlouhodobě standardem v počítačové grafice a základní metodou mnoha dalších šumových algoritmů. [1]

2.3 Worley Noise

Worley noise, představený Stevenem Worleym v roce 1996, je šum založený na vzdálenosti. Každá buňka mřížky obsahuje náhodně umístěný feature point a hodnota šumu je určena vzdáleností k nejbližšímu z těchto bodů. Výsledné vzory připomínají Voroného diagramy nebo buněčné struktury.

Výhodou Worley noise je charakteristický, výrazně odlišný vzhled, vhodný pro textury jako kámen, pěna, popraskané povrchy nebo organické materiály. Nevýhodou je vyšší výpočetní náročnost a méně spojitý charakter výsledného šumu. Oproti Perlinovu šumu vytváří ostřejší a výraznější obrazce. [3]

3 Zvolené řešení

3.1 Sjedené rozhraní

Pro všechny tři algoritmy byla zvolena metoda:

```
sample(x : float, y : float) -> float in [0,1]
```

3.2 Pseudokód algoritmů

Value Noise – pseudokód

Algorithm 1 ValueNoise.sample(x, y)

```
1: Spočítej celočíselnou souřadnici buňky:
2:    $x_0 \leftarrow \lfloor x \rfloor, y_0 \leftarrow \lfloor y \rfloor$ 
3:    $x_1 \leftarrow x_0 + 1, y_1 \leftarrow y_0 + 1$ 
4:
5: Spočítej lokální souřadnice bodu v buňce:
6:    $u \leftarrow x - x_0, v \leftarrow y - y_0$   $\triangleright u, v \in [0, 1)$ 
7:
8: Získej náhodné hodnoty v rozích buňky:
9:    $v_{00} \leftarrow \text{randomValue}(x_0, y_0)$ 
10:   $v_{10} \leftarrow \text{randomValue}(x_1, y_0)$ 
11:   $v_{01} \leftarrow \text{randomValue}(x_0, y_1)$ 
12:   $v_{11} \leftarrow \text{randomValue}(x_1, y_1)$ 
13:
14: Spočítej vyhlazené váhy v osách:
15:    $s_x \leftarrow \text{fade}(u)$ 
16:    $s_y \leftarrow \text{fade}(v)$ 
17:
18: Interpoluj v ose  $x$ :
19:    $a \leftarrow \text{lerp}(v_{00}, v_{10}, s_x)$ 
20:    $b \leftarrow \text{lerp}(v_{01}, v_{11}, s_x)$ 
21:
22: Výsledná hodnota je interpolace v ose  $y$ :
23:   return  $\text{lerp}(a, b, s_y)$ 
```

Perlin Noise – pseudokód

Algorithm 2 PerlinNoise.sample(x, y)

```
1: Spočítej celočíselnou souřadnici buňky:
2:    $x_0 \leftarrow \lfloor x \rfloor, y_0 \leftarrow \lfloor y \rfloor$ 
3:    $x_1 \leftarrow x_0 + 1, y_1 \leftarrow y_0 + 1$ 
4:
5: Spočítej lokální souřadnice bodu v buňce:
6:    $u \leftarrow x - x_0, v \leftarrow y - y_0$ 
7:
8: Získej gradientové vektory v rozích buňky:
9:    $g_{00} \leftarrow \text{randomGradient}(x_0, y_0)$ 
10:   $g_{10} \leftarrow \text{randomGradient}(x_1, y_0)$ 
11:   $g_{01} \leftarrow \text{randomGradient}(x_0, y_1)$ 
12:   $g_{11} \leftarrow \text{randomGradient}(x_1, y_1)$ 
13:
14: Spočítej vektory od rohu k bodu:
15:   $d_{00} \leftarrow (u, v)$ 
16:   $d_{10} \leftarrow (u - 1, v)$ 
17:   $d_{01} \leftarrow (u, v - 1)$ 
18:   $d_{11} \leftarrow (u - 1, v - 1)$ 
19:
20: Spočítej skalární součiny (příspěvky z rohů):
21:   $n_{00} \leftarrow \text{dot}(g_{00}, d_{00})$ 
22:   $n_{10} \leftarrow \text{dot}(g_{10}, d_{10})$ 
23:   $n_{01} \leftarrow \text{dot}(g_{01}, d_{01})$ 
24:   $n_{11} \leftarrow \text{dot}(g_{11}, d_{11})$ 
25:
26: Spočítej vyhlazené váhy v osách:
27:   $s_x \leftarrow \text{fade}(u)$ 
28:   $s_y \leftarrow \text{fade}(v)$ 
29:
30: Interpoluj příspěvky v ose  $x$ :
31:   $a \leftarrow \text{lerp}(n_{00}, n_{10}, s_x)$ 
32:   $b \leftarrow \text{lerp}(n_{01}, n_{11}, s_x)$ 
33:
34: Výsledná hodnota je interpolace v ose  $y$ :
35:   $h \leftarrow \text{lerp}(a, b, s_y)$ 
36:
37: Normalizuj do intervalu  $[0, 1]$ :
38:  return  $0.5 \cdot h + 0.5$ 
```

Worley Noise – pseudokód

Algorithm 3 WorleyNoise.sample(x, y)

```
1: Najdi index buňky, která obsahuje bod  $(x, y)$ :
2:    $i_0 \leftarrow \lfloor x \rfloor, j_0 \leftarrow \lfloor y \rfloor$ 
3:
4: Inicializuj minimální vzdálenost:
5:    $minDist \leftarrow +\infty$ 
6:
7: Pro všechny sousední buňky v okolí  $3 \times 3$ :
8:   for  $\Delta j \leftarrow -1$  to  $1$  do
9:     for  $\Delta i \leftarrow -1$  to  $1$  do
10:       $i \leftarrow i_0 + \Delta i$ 
11:       $j \leftarrow j_0 + \Delta j$ 
12:      Získej pozici náhodného feature-pointu v buňce  $(i, j)$ :
13:       $(p_x, p_y) \leftarrow \text{randomPointInCell}(i, j)$ 
14:      Spočítej eukleidovskou vzdálenost od  $(x, y)$ :
15:       $d \leftarrow \sqrt{(p_x - x)^2 + (p_y - y)^2}$ 
16:      Aktualizuj minimální vzdálenost:
17:       $minDist \leftarrow \min(minDist, d)$ 
18:     end for
19:   end for
20:
21: Normalizuj vzdálenost do intervalu  $[0, 1]$ :
22:   return  $\text{normalize}(minDist)$ 
```

4 Experimenty a výsledky

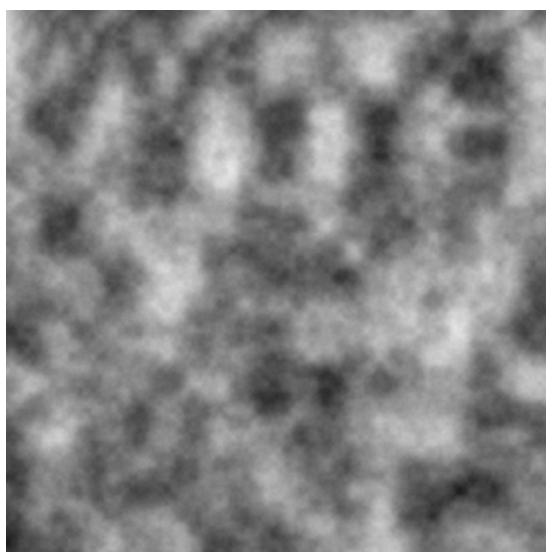
Experimenty byly provedeny v jazyce Python. Všechny tři noise algoritmy byly testovány se stejnými parametry:

- seed = 1234
- rozlišení 512×512 pixelů
- scale = 0,02

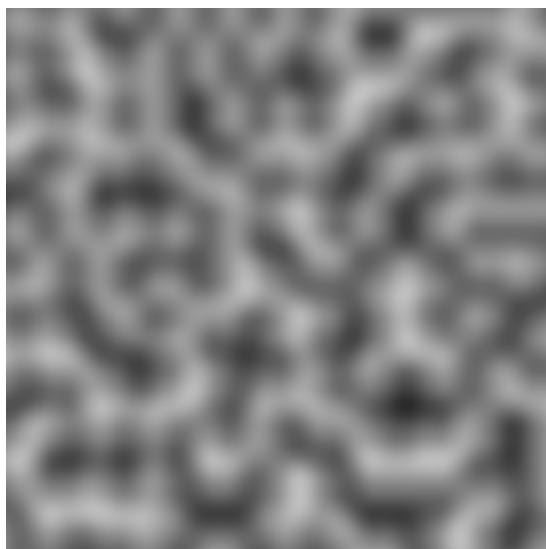
4.1 Výsledné obrázky



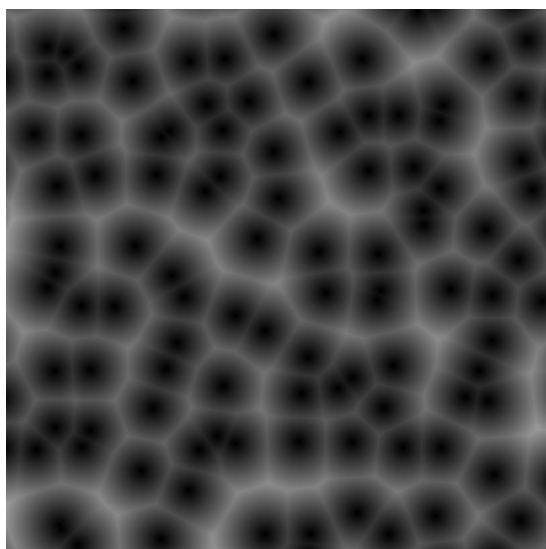
Obrázek 1: Value Noise (bez fBm)



Obrázek 2: Value Noise (s využitím fBm)



Obrázek 3: Perlin Noise



Obrázek 4: Worley Noise

4.2 Porovnání

4.2.1 Value Noise

Value Noise je založen na interpolaci náhodných hodnot v rozích mřížkových buněk. Základní varianta (jedna oktáva) vykazuje výrazně blocky strukturu s viditelnými přechody mezi buňkami, což omezuje jeho praktické využití. Value Noise kombinující více oktáv dosahuje výrazně hladšího výsledku s plynulejšími přechody, ale stále postrádá detail a přirozenost ve srovnání metodami využívající gradienty. Výsledný šum je poměrně "měkký" a jednoduchý. Hodí se pro základní generování terénů nebo mraků, kde není vyžadován vysoký realismus.

4.2.2 Perlin Noise

Perlinův šum je z testovaných metod nejvyváženější. Vytváří vizuálně spojitější a hladší struktury než Value noise a zároveň nabízí mnohem přirozenější variabilitu. Díky gradientové konstrukci je šum plynulý ve všech směrech a přechody mezi oblastmi jsou jemné. Perlin noise také nevykazuje výrazné blokové artefakty, které jsou typické pro Value noise.

Jedním z mála možných artefaktů je jemná mřížková stopa při extrémním přiblížení, ale u standardního použití je prakticky neviditelná. Perlinův šum se velmi dobře hodí pro generování terénů, oblaků, organických vzorů nebo přírodních textur. Je to univerzální metoda, která poskytuje realistický, hladký charakter a lze ji snadno kombinovat do více oktáv (fBm), čímž vznikají bohaté a detailní struktury.

4.3 Worley Noise

Worley noise se od předchozích metod zásadně liší. Nevytváří hladké nebo spojité struktury, ale výrazné buněčné vzory připomínající Voroného diagram. Ty jsou charakteristické tmavými oblastmi v místech nejbližších feature pointů a světlejšími přechody směrem k hranicím buněk. Šum nepůsobí hladce, ale spíše ostře a segmentovaně.

Tento typ šumu je velmi vhodný pro efekty, které mají přirozeně buněčný charakter — například textury kamene, popraskaných povrchů, lávy, pěny nebo biomorfních struktur. Naopak se nehodí pro generování terénu nebo měkkých přírodních tvarů, kde je žádoucí spojitost a hladkost.

4.4 Tabulka výsledků

Metoda	Hladkost	Charakter	Typické použití	Poznámka
Value Noise (1 oktáva)	nízká	blocky, umělé	prototypy	mřížková struktura
Value Noise (fBm)	střední	měkký, jednoduchý	oblaka, terény	vyžaduje více oktáv
Perlin Noise	vysoká	organický, přírodní	terény, textury	standard pro CGI
Worley Noise	nízká	buněčný, segmentovaný	kámen, praskliny	vzdálenostní funkce

Tabulka 1: Porovnání vlastností jednotlivých noise algoritmů.

5 Závěr

V této práci byly implementovány a porovnány tři algoritmy pro procedurální generování dvourozměrného šumu: Value Noise, Perlin Noise a Worley Noise. Každý z nich představuje odlišný přístup k tvorbě pseudonáhodných struktur a vykazuje specifické vizuální i technické vlastnosti. Implementace byla sjednocena pod jednotné rozhraní a jednotlivé metody byly testovány za stejných podmínek, což umožnilo jejich objektivní porovnání.

Z experimentů vyplynulo, že základní Value Noise (jedna oktáva) trpí výraznou kostovou strukturou a viditelnými mřížkovými artefakty, které jej činí nevhodným pro většinu praktických aplikací. Fraktální varianta Value Noise (fBm) kombinující více oktáv dosahuje výrazně hladšího výsledku, avšak stále nedosahuje kvality metod za využití gradientu. Perlin Noise se ukázal jako nejvyváženější metoda — vytváří spojitě, organické struktury s minimem artefaktů již v základní variantě a je vhodný pro širokou škálu aplikací v oblasti počítačové grafiky a procedurálního generování terénů. Naproti tomu Worley Noise produkuje výrazně odlišné, buněčné a segmentované vzory založené na výpočtu vzdáleností. Není určen pro generování plynulých povrchů, ale vyniká při tvorbě specifických textur jako je kámen, žíraní kůže, praskliny nebo jiné přírodní struktury s buněčným charakterem.

Celkově lze říci, že každá z metod má své specifické využití. Perlin Noise je univerzální volbou pro přirozeně působící šum a standard v odvětví. Value Noise ve fraktální variantě se hodí pro základní textury a atmosférické efekty, kde není vyžadován vysoký realismus. Worley Noise dominuje při generování buněčných nebo voronoi-podobných struktur. V rámci práce byla také implementována podpora fraktálního šumu (fractal Brownian motion) pro Value Noise, což demonstruje možnost zvýšení kvality výsledků kombinací více oktáv.

Reference

- [1] Wikipedia contributors. Perlin noise. Wikipedia, The Free Encyclopedia, 2024. Accessed: 2025-11-28.
- [2] Wikipedia contributors. Value noise. Wikipedia, The Free Encyclopedia, 2024. Accessed: 2025-11-28.
- [3] Wikipedia contributors. Worley noise. Wikipedia, The Free Encyclopedia, 2024. Accessed: 2025-11-28.