Perlin-zaj

Pintér Bálint

2026. január 27.

Tartalomjegyzék

1. Perlin-zaj **3**

1.1. Működése összefoglalva . . . 3

2. Előkészítés **3**

2.1. Gradienstábla . . . 4

2.1.1. Vektorgenerálás . . . 4

2.2. Permutációs tábla . . . 5

3. Zajszámítás **6**

3.1. Rácspontok meghatározása . . . 6

3.2. Skaláris szorzat kiszámítása . . . 7

3.2.1. Gradiens vektorok kiválasztása . . . 7

3.2.2. A sarkokból a pontba mutató vektorok kiszámítása . . . 7

3.2.3. Skaláris szorzat kiszámítása . . . 8

3.3. Interpoláció . . . 8

3.3.1. Simítófüggvény . . . 8

3.3.2. Interpoláció . . . 9

4. Teljes zajfüggvény **10**

5. Fractal Brownian Motion (Fraktálzaj) **11**

5.1. Paraméterei . . . 11

5.2. FBM matematikai leírása és szemléltetése . . . 11

5.3. FBM pszeudókód . . . 12

Forrásjegyzék **13**

1. Perlin-zaj

A Perlin-zaj [1] egy gradiensalapú zajgenerálási algoritmus, amelynek a célja véletlenszerű, mégis összefüggő zaj létrehozása. Segítségével a természetben előforduló véletlenszerű jelenségeket lehet jól szimulálni, mint például domborzatokat, felhőket vagy a víz hullámzását. Tetszőleges n dimenzióra létrehozható, de jellemzően az elsőtől a negyedik dimenzióig alkalmazzák. A kódban egy kétdimenziós Perlin-zaj van implementálva.

1.1. Működése összefoglalva

1. **Rács meghatározása:** A zaj dimenziójában egy egész koordináták által kijelölt elméleti rácsot használunk, amelynek minden sarkához egy véletlenszerűen kiválasztott egységvektort rendelünk.
2. **Rácsnégyzeten belüli vektorok kiszámítása:** Meghatározzuk a sarkokból a belső pontba mutató vektorokat.
3. **Skaláris szorzás:** Az adott sarokból a pontba mutató vektornak és az adott sarokhoz rendelt vektornak vesszük a skaláris szorzatát.
4. **Interpoláció:** A kapott skaláris szorzatokat végül tengelyenként interpoláljuk. Az interpoláció súlyozásához egy simítófüggvényt használunk. Például a kétdimenziós zajnál először az x tengely mentén interpolálunk, majd a kapott részeredményeket az y tengely mentén interpoláljuk.

2. Előkészítés

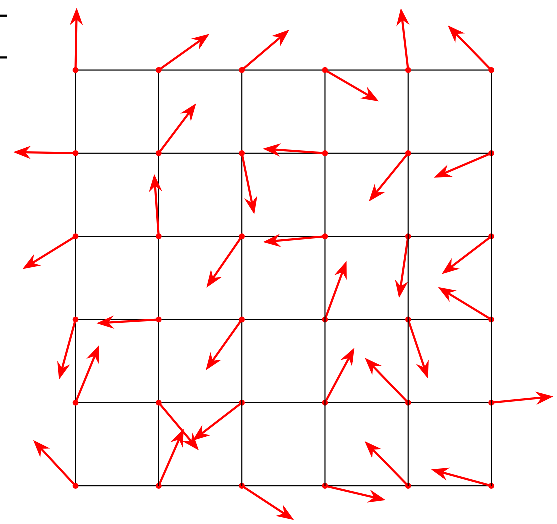
A Perlin-zaj hatékony generálásához két adat inicializálására van szükség: egy gradienstáblára és egy permutációs táblára.

2.1. Gradienstábla

A Perlin-zaj egy úgynevezett gradiensalapú zaj. Eszerint rácspontokat határozunk meg, amelyekhez véletlenszerű vektorokat rendelünk. A gradienstábla 256 darab ilyen vektort tárol el. A vektorok dimenziószáma megegyezik a zaj dimenziószámával. (Például: Kétdimenziós zaj esetén kétdimenziós vektorokat használunk.)

1. Algoritmus: Gradienstábla létrehozása

```
Konstans: MaxG = 256
Típus: VéletlenSzámGenerátor = Osztály (
    jelenlegiSzám: Egész
    Függvény Következő: Egész
    Függvény KövetkezőValós: Valós   [∈ [0; 1[]
 )
Típus: Vektor = Rekord (
    x: Valós
    y: Valós
)
Eljárás GradiensTablaGeneral (Változó:
 GradiensTabla: Tömb(1..MaxG: Vektor), Rand:
 VéletlenSzámGenerátor):
  | Változó: i: Egész
  | Ciklus i := 1-től 256-ig
  |   | GradiensTabla[i] :=
  |   |   VeletlenVektorGeneral(Rand)
  | Ciklus vége
Eljárás vége
```



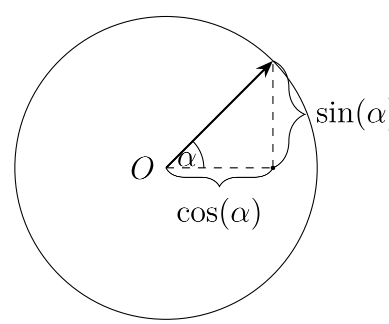
1. Ábra
A zaj rácsának szemléltetése.

2.1.1. Vektorgenerálás

Generálunk egy véletlenszerű számot $[0; 2\pi[$ intervallumban. Majd egyszerű trigonometriával a szöget egy vektorrá alakítjuk, ahol a vektor x komponense a véletlen szög koszinusza, és az y komponense a szög szinusza.

2. Algoritmus: Véletlenszerű vektor generálás

```
Típus: VéletlenSzámGenerátor = Osztály (
    jelenlegiSzám: Egész
    Függvény Következő: Egész
    Függvény KövetkezőValós: Valós          [∈ [0; 1[]
 )
Típus: Vektor = Rekord (
    x: Valós
    y: Valós
)
Függvény VeletlenVektorGeneral (Változó: Rand:
 VéletlenSzámGenerátor): Vektor
  | Változó: szog: Valós
  | Változó: vektor: Vektor
  | szog := Rand.KövetkezőValós() ∗2 ∗ π
  | vektor.x := cos(szog)
  | vektor.y := sin(szog)
  | VeletlenVektorGeneral := vektor
Függvény vége
```



2. Ábra
A vektorok előállításának szemléltetése.

2.2. Permutációs tábla

A permutációs tábla kezdetben 0-tól 255-ig tartalmazza a számokat növekvő sorrendben. Ezt a listát egy véletlenszám-generátor segítségével összekeverjük és önmaga után fűzzük (ezzel egy 512 elemű tömböt kapunk). Így a hashelésnél elkerülhető a túlindexelés, amely gyorsítja a zajgenerálást, mivel elhagyható a határellenőrzés.

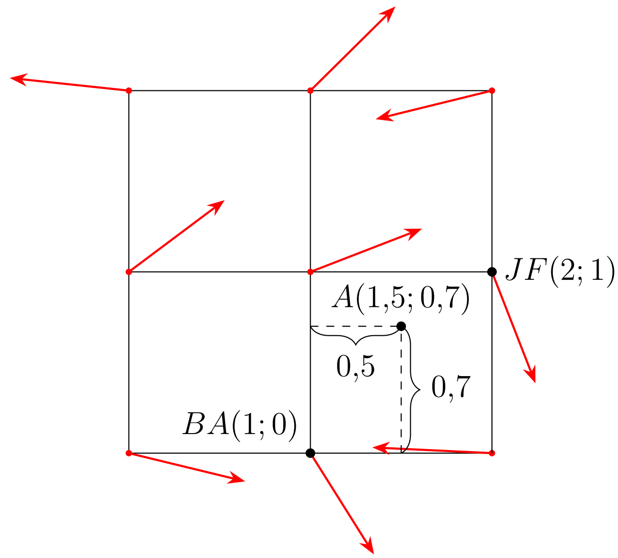
3. Algoritmus: Permutációs tábla létrehozása

```
Konstans: MaxP = 512
Típus: VéletlenSzámGenerátor = Osztály (
    jelenlegiSzám: Egész
    Függvény Következő: Egész
    Függvény KövetkezőValós: Valós  [∈ [0; 1[  ]
)
Eljárás PermutaciosTablaGeneral (Változó: PermutaciosTabla: Tömb(1..MaxP: Egész),
  Rand: VéletlenSzámGenerátor):
    Változó: i, j, temp: Egész
    Ciklus i := 1-től 256-ig
        PermutaciosTabla[i] := i − 1
    Ciklus vége

    Ciklus i := 256-tól 2-ig −1-esével
        j := (Rand.Következő() Mod i) + 1
        temp := PermutaciosTabla[i]
        PermutaciosTabla[i] := PermutaciosTabla[j]
        PermutaciosTabla[j] := temp
    Ciklus vége

    Ciklus i := 1-től 256-ig
        PermutaciosTabla[i + 256] := PermutaciosTabla[i]
    Ciklus vége
Eljárás vége
```

3. Zajszámítás

3.1. Rácspontok meghatározása

Először meghatározzuk, hogy az adott (x, y) pont melyik négyzetbe esik. Ehhez a koordinátákat lefelé kerekítjük, majd az eredményen elvégzünk egy 255-tel való bitenkénti ÉS (AND) műveletet. Ez a művelet az eredményt a $[0; 255]$ tartományba hozza: ha az érték nagyobb, mint 255, akkor visszafordul az intervallum elejére (pl. 256-ból 0 lesz, 257-ből 1 lesz). Ezt elvégezve az x-re és y-ra megkapjuk a bal alsó rácspont koordinátáit.



3. Ábra
A rácspont koordinátáinak szemléltetése.

A jobb felső rácspont koordinátáit úgy kapjuk meg, hogy a bal alsó pont értékeit 1-gyel megnöveljük, majd az eredményt 255-tel bitenkénti ÉS művelettel maszkoljuk. A bitenkénti ÉS műveletre a túlcsordulás kezelése miatt van szükség.
A négyzeten belüli pontot úgy kapjuk meg, hogy a számból kivonjuk annak lefelé kerekített értékét.

Pszeudókódban megvalósítva:

4. Algoritmus: Rácspontok és négyzeten belüli koordináták kiszámítása

Típus: Rácspont = Rekord (
    balAlsóPontX, balAlsóPontY: Egész
    jobbFelsőPontX, jobbFelsőPontY: Egész
    relatívX, relatívY: Valós
)
Függvény *RacspontKiszamolasa(**Konstans**: x, y: Valós) : Rácspont*
    Változó: jelenlegiRácspont: Rácspont
    jelenlegiRácspont.balAlsóPontX := floor(x) & 255
    jelenlegiRácspont.balAlsóPontY := floor(y) & 255
    jelenlegiRácspont.jobbFelsőPontX := (jelenlegiRácspont.balAlsóPontX $+1$) & 255
    jelenlegiRácspont.jobbFelsőPontY := (jelenlegiRácspont.balAlsóPontY $+1$) & 255

    jelenlegiRácspont.relatívX := $x-$ floor(x)
    jelenlegiRácspont.relatívY := $y-$ floor(y)
    RacspontKiszamolasa := jelenlegiRácspont
Függvény vége

3.2. Skaláris szorzat kiszámítása

3.2.1. Gradiens vektorok kiválasztása

A gradiens vektorokat a permutációs tábla és a *Hash* függvény segítségével választjuk ki. Vesszük a permutációs tábla x-edik elemét, hozzáadjuk az y értékét, majd ezt az összeget használjuk indexként a permutációs táblában. Az így kapott eredmény lesz az indexe a gradiens vektornak a gradienstáblából.

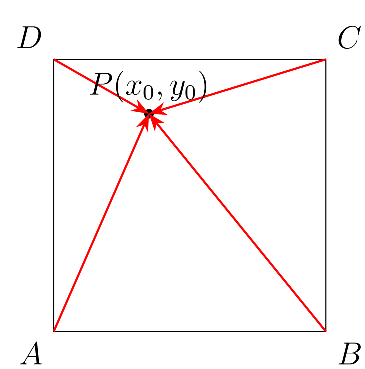
5. Algoritmus: Gradiens vektor kiválasztása

```
Típus: Vektor = Rekord (
    x: Valós
    y: Valós
)
Konstans: MaxP = 512, MaxG = 256
Változó: PermutaciosTabla: Tömb(1..MaxP: Egész)
Változó: GradiensTabla: Tömb(1..MaxG: Vektor)
Függvény Hash(Konstans: x, y: Egész) : Egész
    [A permutációs tábla 0-tól 255-ig tartalmazza a számokat ]
    [Azonban a pszeudókód 1-től kezdi az indexelést, ezért hozzáadunk
     egyet ]
    Hash := PermutaciosTabla[(PermutaciosTabla[x] + 1) + y] + 1
Függvény vége

Függvény GradiensVektorKivalaszt(Konstans: x, y: Egész) : Vektor
    [A rácspont koordináták [0; 255] intervallumba esnek ]
    [Azonban a pszeudókód 1-től kezdi az indexelést, ezért hozzáadunk
     egyet ]
    GradiensVektorKivalaszt := GradiensTabla[Hash(x + 1, y + 1)]
Függvény vége
```

3.2.2. A sarkokból a pontba mutató vektorok kiszámítása

Legyen a négyzeten belüli P pont relatív koordinátái (x_0, y_0), ahol $x_0, y_0 \in [0; 1]$. A rácsnégyzet sarkai legyen A, B, C, D.
Így a sarkokból a pontba mutató vektorok:

- **Bal alsó:** $\vec{v}_{AP}(x_0, y_0)$
- **Jobb alsó:** $\vec{v}_{BP}(x_0 - 1, y_0)$
- **Bal felső:** $\vec{v}_{DP}(x_0, y_0 - 1)$
- **Jobb felső:** $\vec{v}_{CP}(x_0 - 1, y_0 - 1)$



4. Ábra
Relatív vektorok.

3.2.3. Skaláris szorzat kiszámítása

A vektorok meghatározása után kiszámítjuk az adott sarokhoz tartozó gradiens- és relatív vektorok skaláris szorzatát. A skaláris szorzatot a matematikai tétel alapján végezzük: $\vec{a} \cdot \vec{b} = x_a x_b + y_a y_b$

6. Algoritmus: Skaláris szorzat

```
Típus: Vektor = Rekord (
        x: Valós
        y: Valós
)
Függvény SkalarisSzorzat(Konstans: v1, v2: Vektor) : Valós
    SkalarisSzorzat := v1.x * v2.x + v1.y * v2.y
Függvény vége
```

3.3. Interpoláció

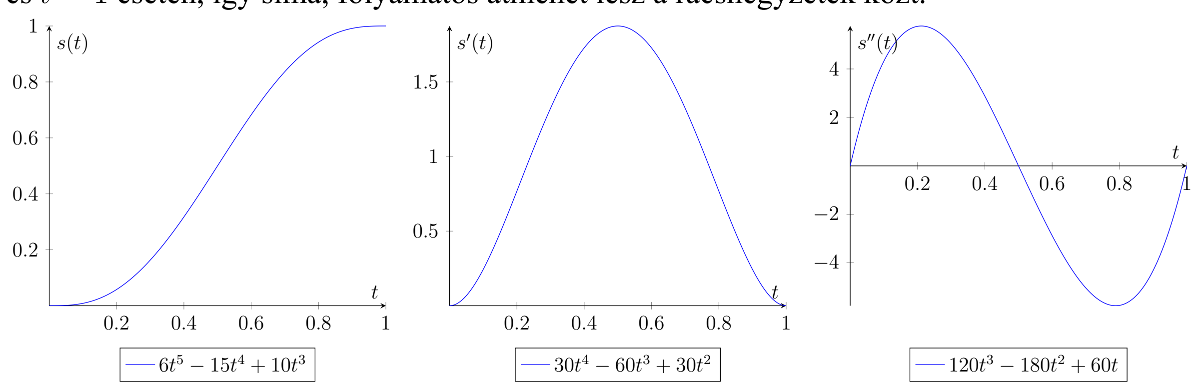
A kapott skalárszorzatokat végül a simítófüggvény által módosított súlytényezővel interpoláljuk a tengelyek mentén.

3.3.1. Simítófüggvény

Simítófüggvényként a Ken Perlin által 2002-ben, az 'Improved Noise'-ban [2] bevezetett függvényt használjuk.

$$s(t) = 6t^5 - 15t^4 + 10t^3$$

A függvény fontos jellemzője, hogy az első deriváltja és a második deriváltja is egyenlő 0-val $t = 0$ és $t = 1$ esetén, így sima, folyamatos átmenet lesz a rácsnégyzetek közt.



7. Algoritmus: Simítófüggvény

```
Függvény Simitofuggveny(Konstans: t: Valós) : Valós
    [Horner-elrendezésben (12 szorzás helyett csak 5 szorzás):   ]
    Simitofuggveny := t * t * t * (10 + t * ((-15) + t * 6))
Függvény vége
```

3.3.2. Interpoláció

A végeredményt a skaláris szorzatok interpolálásával kapjuk. Kétdimenziós esetben először kiszámítjuk a relatív x-koordináta simítófüggvénybeli értékét, majd eszerint interpoláljuk a skaláris szorzatokat az x tengely mentén, tehát a felső két sarokhoz, illetve az alsó két sarokhoz tartozó skaláris szorzatokat interpoláljuk egymással. Majd meghatározzuk a relatív y-koordináta simítófüggvénybeli értékét, és eszerint interpoláljuk az előző két interpolált részeredményt.

8. Algoritmus: Interpoláció

Függvény *Interpolacio(**Konstans**: a, b, t: Valós) : Valós*
    Interpolacio $:= a + t * (b - a)$
Függvény vége

4. Teljes zajfüggvény

Először meghatározzuk a vizsgált pontot tartalmazó rácsnégyzet koordinátáit és a ponton belüli relatív helyzetét a *RacspontKiszamolasa* függvénnyel. Ezt követően lekérjük a négy sarokhoz tartozó vektorokat a *GradiensVektorKivalaszt* függvénnyel, majd kiszámítjuk a sarkokból a pontba mutató távolságvektorokat. Végül a *SkalarisSzorzat* függvény segítségével kiszámítjuk az egyes sarkokhoz tartozó relatív és gradiens vektorok skaláris szorzatát. Legvégül interpoláljuk a skaláris szorzatokat az *Interpolacio* függvénnyel.

9. Algoritmus: Teljes zajfüggvény

```
Típus: Vektor = Rekord (x, y: Valós)
Típus: Rácspont = Rekord (
    balAlsóPontX, balAlsóPontY: Egész
    jobbFelsőPontX, jobbFelsőPontY: Egész
    relatívX, relatívY: Valós
)
Függvény Zaj(Konstans: x, y: Valós) : Valós
    Változó: rácspont: Rácspont
    Változó: g00, g10, g01, g11: Vektor
    Változó: v00, v10, v01, v11: Vektor
    Változó: tX, tY, a, b: Valós
    [1. Rácspont és relatív koordináták kiszámítása ]
    rácspont := RacspontKiszamolasa(x, y)
    [2. Gradiens vektorok lekérdezése ]
    g00 := GradiensVektorKivalaszt(rácspont.balAlsóPontX, rácspont.balAlsóPontY)
    g10 := GradiensVektorKivalaszt(rácspont.jobbFelsőPontX, rácspont.balAlsóPontY)
    g01 := GradiensVektorKivalaszt(rácspont.balAlsóPontX, rácspont.jobbFelsőPontY)
    g11 := GradiensVektorKivalaszt(rácspont.jobbFelsőPontX, rácspont.jobbFelsőPontY)
    [3. Relatív vektorok definiálása ]
    v00.x := rácspont.relatívX
    v00.y := rácspont.relatívY
    v10.x := rácspont.relatívX - 1.0
    v10.y := rácspont.relatívY
    v01.x := rácspont.relatívX
    v01.y := rácspont.relatívY - 1.0
    v11.x := rácspont.relatívX - 1.0
    v11.y := rácspont.relatívY - 1.0
    [4. Simítófüggvény alkalmazása ]
    tX := Simitofuggveny(rácspont.relatívX)
    tY := Simitofuggveny(rácspont.relatívY)
    [5. Skaláris szorzatok kiszámítása és interpolálásuk ]
    a := Interpolacio(SkalarisSzorzat(g00, v00), SkalarisSzorzat(g10, v10), tX)
    b := Interpolacio(SkalarisSzorzat(g01, v01), SkalarisSzorzat(g11, v11), tX)
    Zaj := Interpolacio(a, b, tY)
Függvény vége
```

5. Fractal Brownian Motion (Fraktálzaj)

A Perlin-zaj önmagában túl sima, és hiányoznak belőle az apró részletek. Ezt a **Fractal Brownian Motion (FBM)** segítségével oldhatjuk meg. A módszer lényege, hogy több réteg (úgynevezett *oktáv*) Perlin-zajt generálunk és adunk össze, ahol minden újabb réteg nagyobb frekvenciával és kisebb amplitúdóval rendelkezik.

5.1. Paraméterei

A fraktálzaj finomhangolható a következő paraméterekkel:

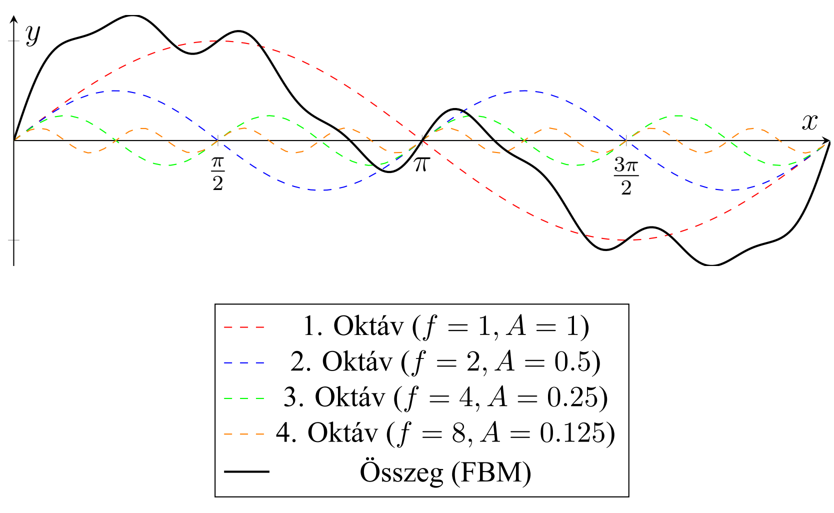
- **Oktávok:** Azt határozza meg, hány réteg zajt adunk össze. Minél magasabb ez a szám, annál részletesebb a végeredmény, de annál többször kell lefuttatni a zajgenerálást.
- **Amplitúdó (nagyság):** A zaj kezdeti magassága.
- **Frekvencia:** A zaj kezdeti sűrűsége.
- **Lacunarity:** Azt határozza meg, hogy az oktávok között hogyan változik a frekvencia. Az értéke általában $2{,}0$, tehát minden következő réteg kétszer olyan sűrű, mint az előző.
- **Persistence:** Azt határozza meg, hogy az oktávok között hogyan csökken az amplitúdó. Az értéke általában $0{,}5$, tehát minden következő réteg fele olyan magas, mint az előző.

5.2. FBM matematikai leírása és szemléltetése

A végső zajfüggvényt matematikailag így írhatjuk fel:

$$FBM(x,y) = \sum_{i=0}^{n-1} A \cdot P^i \cdot \text{zaj}(x \cdot F \cdot L^i, y \cdot F \cdot L^i)$$

Ahol az oktávok száma n, a kezdeti amplitúdó A, a persistence P, a lacunarity L, a frekvencia pedig F.



5. Ábra
Az oktávok összegzésének szemléltetése egy dimenzióban a szinuszfüggvénnyel.

5.3. FBM pszeudókód

A megvalósított kódban a fraktálzajt normalizáljuk a $[-1;1]$ intervallumra a maximális lehetséges értékkel való osztással. A normalizálás után két saját paramétert alkalmazunk.

- **Kontraszt:** A normalizált érték abszolútértékét erre a hatványra emeljük az eredeti előjel megtartásával. Így nagyobb lesz a kontraszt a nagyságok között.
- **Zajméret:** A kontraszt alkalmazása után ezzel az értékkel megszorozzuk a zaj értékét.

A fraktálzaj pszeudókódban megvalósítva:

10. Algoritmus: Fractal Brownian Motion (FBM)

```
[A zaj paraméterei:                                                  ]
Konstans: oktavok, kontraszt: Egész
Konstans: frekvencia, amplitudo, persistence, lacunarity, zajMeret: Valós

Függvény FBM(Konstans: x, y: Valós) : Valós
    Változó: zajErtek, maxErtek, jelenlegiAmplitudo, jelenlegiFrekvencia, zajElojel:
             Valós
    Változó: i: Egész
    zajErtek := 0
    maxErtek := 0
    jelenlegiAmplitudo := amplitudo
    jelenlegiFrekvencia := frekvencia

    Ciklus i := 1-től oktavok-ig
        [Zaj hozzáadása az aktuális frekvenciával és amplitúdóval      ]
        zajErtek := zajErtek + Zaj(x * jelenlegiFrekvencia, y * jelenlegiFrekvencia) *
         jelenlegiAmplitudo

        [Maximális lehetséges érték                                    ]
        maxErtek := maxErtek + jelenlegiAmplitudo

        [Paraméterek frissítése a következő oktávhoz                   ]
        jelenlegiAmplitudo := jelenlegiAmplitudo * persistence
        jelenlegiFrekvencia := jelenlegiFrekvencia * lacunarity
    Ciklus vége

    [Normalizálás, kontraszt alkalmazása és méretezés                  ]
    zajElojel := Elojel(zajErtek)
    FBM := (Abs(zajErtek/maxErtek))^kontraszt * zajElojel * zajMeret
Függvény vége
```

Forrásjegyzék

[1] Ken Perlin. “An image synthesizer”. *SIGGRAPH Comput. Graph.* 19.3 (1985. júl.), 287–296. old. ISSN: 0097-8930. DOI: 10.1145/325165.325247. URL: https://doi.org/10.1145/325165.325247.

[2] Ken Perlin. “Improving noise”. *ACM Trans. Graph.* 21.3 (2002. júl.), 681–682. old. ISSN: 0730-0301. DOI: 10.1145/566654.566636. URL: https://doi.org/10.1145/566654.566636.