Perlin-zaj

Pintér Bálint

2026. január 25.

Tartalomjegyzék

1. Perlin-zaj

A Perlin-zaj egy gradiens alapú zajgenerálási algoritmus, amelynek a célja a véletlenszerű, de összefüggő zaj létrehozása. [1] Segítségével a természetben előforduló véletlenszerű jelenségeket jól lehet szimulálni, mint például domborzatok, felhők vagy a víz hullámzása. Tetszőleges n dimenzióra létrehozható, de jellemzően az 1-től a 4. dimenzióig alkalmazzák. A kódban egy kétdimenziós Perlin-zaj van implementálva.

1.1. Működése összefoglalva

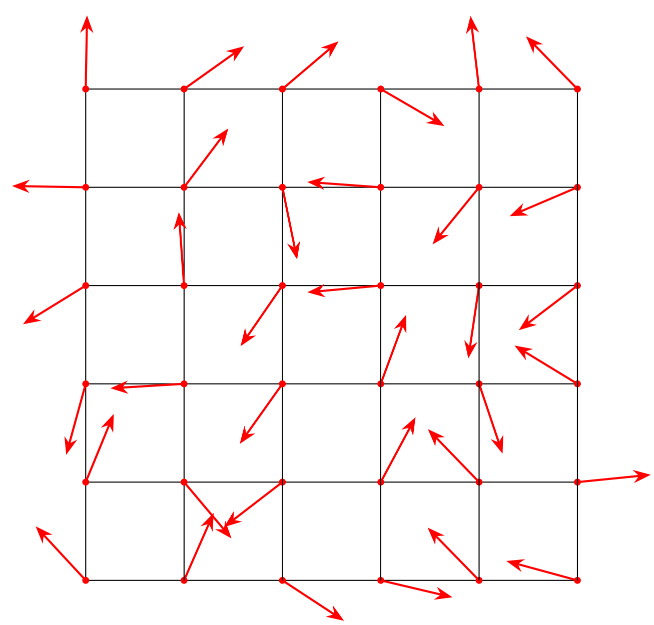
1. **Rács meghatározása:** A zaj dimenziójában egy szabályos rácsot határozunk meg, amelynek minden sarkához egy véletlenszerűen generált egységvektort rendelünk. A rácsvonalak jellemzően az egész koordinátáknál helyezkednek el.
2. **Rácsnégyzeten belüli vektorok kiszámítása:** Kiszámoljuk a rácsnégyzeten belüli pontba a rács sarkaiból 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 tengelyekként interpoláljuk egy simítófüggvénnyel. 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 gradiens táblára és egy permutációs táblára.

2.1. Gradiens tábla

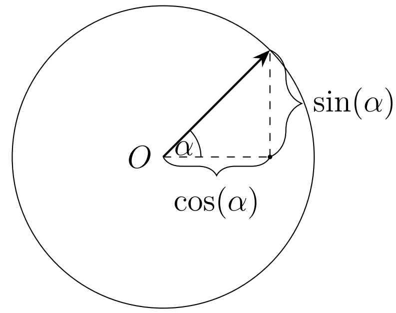
A Perlin-zaj egy úgynevezett gradiens-zaj. Eszerint rácspontokat határozunk meg, amikhez egy véletlenszerű vektort rendelünk. A gradiens tábla ezeket a véletlenszerű vektorokat tárolja. A vektorok dimenziószáma megegyezik a zaj dimenziószámával. (Kétdimenziós zaj $\rightarrow$ kétdimenziós vektor)



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. 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, ami gyorsítja a zajgenerálást, mivel elhagyható a túlindexelésre való ellenőrzés.

1. Algoritmus: Permutációs tábla létrehozása

Konstans: MaxP=512
Típus: VéletlenSzámGenerátor=Osztály (
  jelenlegiSzám:Egész
  Függvény Következő:Egész
)
Eljárás *PermutaciosTablaGeneral (**Változó**: PermutaciosTabla:Tömb(*1*..MaxP:Egész), Rand:VéletlenSzámGenerátor)***:**
  Változó: i, j, temp:Egész

  Ciklus $i := 1$-*től* 256-*ig*
    PermutaciosTabla[i] := i
  Ciklus vége

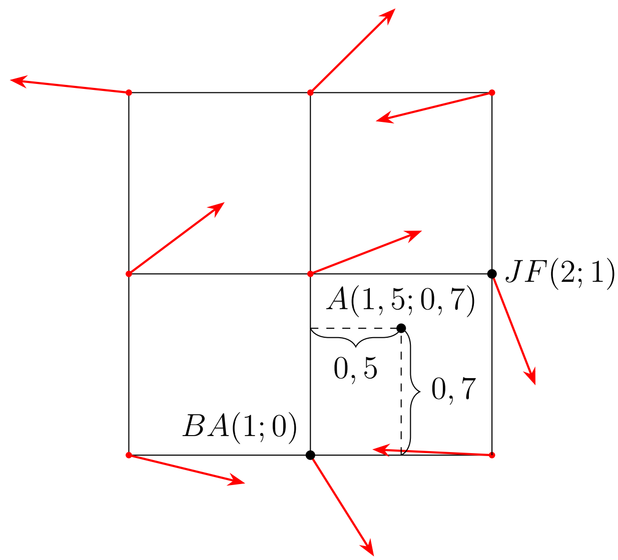
  Ciklus $i := 256$-*tól* 2-*ig* -1-*esével*
    j := Rand.Következő() Mod $(i + 1)$
    temp := PermutaciosTabla[i]
    PermutaciosTabla[i] := PermutaciosTabla[j]
    PermutaciosTabla[j] := temp
  Ciklus vége

  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 tartozik ezt a bitenkénti ÉS 255 művelettel tesszük, így az eredmény a $[0; 255]$ tartományba fog esni: ha az érték nagyobb, mint 255, akkor visszafordul az intervallum elejére (pl. 256-ból 0 lesz). Ezt elvégezve az x-re és y-ra megkapjuk a bal alsó rácspont koordinátáit. A bal alsó rácspont koordinátáihoz hozzáadva 1-et majd egy bitenkénti ÉS 255 művelettel megkapjuk a jobb felső rácspont koordinátáit. A négyzeten belüli pontot úgy kapjuk meg, hogy a szám egész részét elhagyjuk.



3. Ábra
A rácspont koordinátáinak szemléltetése.

Pszeudókódban megvalósítva:

2. Algoritmus: Rácspontok és négyzeten belüli koordináták kiszámolása

```
Típus: Rácspont=Rekord (
        balAlsóPontX, balAlsóPontY:Egész
        jobbFelsőPontX, jobbFelsőPontY:Egész
        relatívX, relatívY :Valós
)
Függvény RacspontKiszamolasa(Konstans: x, y: Valós) : Rácspont:
    Változó: jelenlegiRácspont: Rácspont
    jelenlegiRácspont.balAlsóPontX := ((Egész)floor(x)) & 256 + 1
    jelenlegiRácspont.balAlsóPontY := ((Egész)floor(y)) & 256 + 1

    jelenlegiRácspont.jobbFelsőPontX := (jelenlegiRácspont.balAlsóPontX +1) & 256
    jelenlegiRácspont.jobbFelsőPontY := (jelenlegiRácspont.balAlsóPontY +1) & 256

    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 segítségével választjuk ki. Vesszük a permutációs tábla x-edik elemét, hozzáadjuk az y értékét, majd az így kapott összeget használjuk indexként a permutációs táblában. Az így kapott eredmény lesz az indexe a gradiens vektornak a gradiens táblából.

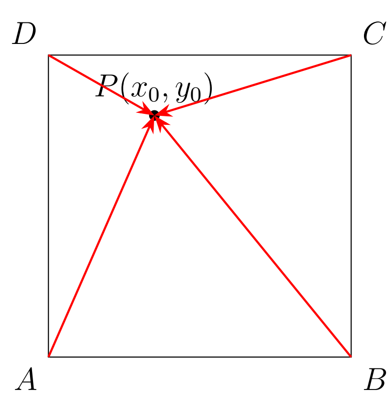
3. Algoritmus: Gradiens vektor kiválasztása

```
Típus: Vektor=Rekord (
    x:Valós
    y:Valós
)
Konstans: MaxP=512, MaxG=256
Változó: PermutaciosTabla:Tömb(1..MaxP:Egész)
Változó: GradiensTabla:Tömb(1..MaxG:Egész)

Függvény Hash(Konstans: x, y: Egész) : Egész:
    Hash := PermutaciosTabla[PermutaciosTabla[x] + y]
Függvény vége

Függvény GradiensVektorKivalaszt(Konstans: x, y: Egész) : Vektor:
    GradiensVektorKivalaszt := GradiensTabla[hash(x, y)]
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:** $\vec{v}_{CP}(x_0, y_0 - 1)$
- **Jobb felső:** $\vec{v}_{DP}(x_0 - 1, y_0 - 1)$



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 definíció alapján végezzük: $\vec{a} \cdot \vec{b} = x_a x_b + y_a y_b$

4. 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

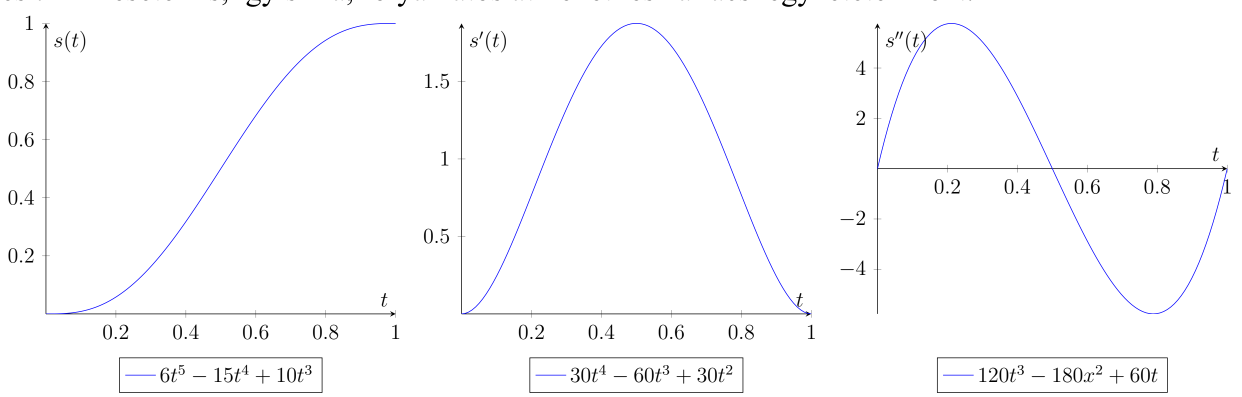
A kapott skalárszorzatokat végül egy simítófüggvény segítségével interpoláljuk a tengelyek mentén.

3.3.1. Simítófüggvény

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

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

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



5. Algoritmus: Simítófüggvény

Függvény *Simitofuggveny(**Konstans**: t: Valós) : Valós*:
    Simitofuggveny := $6t^5 - 15t^4 + 10t^3$
Függvény vége

3.3.2. Interpoláció

A végeredményt a skaláris szorzatok interpolálásával kapjuk. Kétdimenzió esetén először kiszámoljuk a relatív x-koordináta simítófüggvénybeli értékét, majd eszerint interpoláljuk a skaláris szorzatokat az x tengely mentén, tehát a felső skaláris szorzatokat és az alsó skaláris szorzatokat. Majd kiszámoljuk a relatív y-koordináta simítófüggvénybeli értékét, és eszerint interpoláljuk az előző kettő interpolált részeredményt.

6. Algoritmus: Interpoláció

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

4. Teljes zajfüggvény

Először kiszámoljuk 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 kiszámoljuk a skaláris szorzatukat az adott sarkokhoz való vektoroknak a *SkalarisSzorzat* segítségével. Végül interpoláljuk a skaláris szorzatokat az *Interpolacio* függvénnyel.

7. 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: p00, p10, p01, p11: Vektor
    Változó: u, v, 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 ]
    p00.x := rácspont.relatívX
    p00.y := rácspont.relatívY
    p10.x := rácspont.relatívX - 1.0
    p10.y := rácspont.relatívY
    p01.x := rácspont.relatívX
    p01.y := rácspont.relatívY - 1.0
    p11.x := rácspont.relatívX - 1.0
    p11.y := rácspont.relatívY - 1.0
    [4. Simítófüggvény alkalmazása ]
    u := Simitofuggveny(rácspont.relatívX)
    v := Simitofuggveny(rácspont.relatívY)
    [5. Skaláris szorzatok kiszámítása és interpolálásuk ]
    a := Interpolacio(SkalarisSzorzat(g00, p00), SkalarisSzorzat(g10, p10), u)
    b := Interpolacio(SkalarisSzorzat(g01, p01), SkalarisSzorzat(g11, p11), u)
    Zaj := Interpolacio(a, b, v)
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 amplitudóval rendelkezik.

5.1. Paraméterei

A fraktál zaj finomhangolható a következő paraméterekkel:

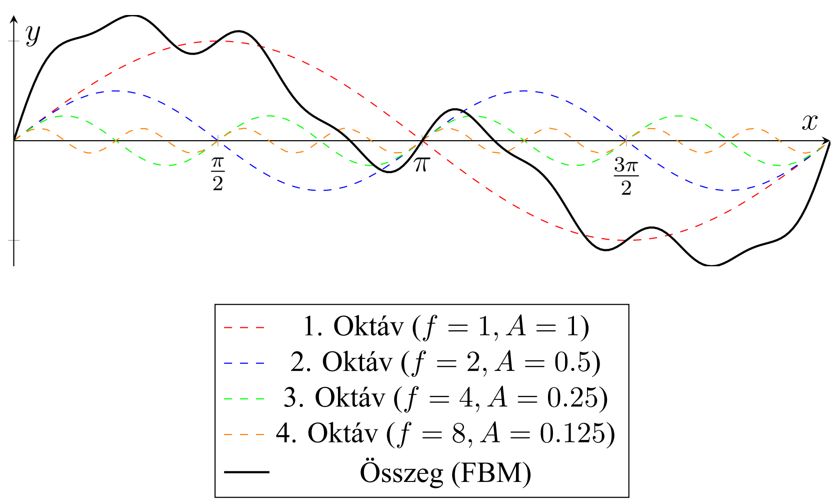
- **Oktávok:** Azt határozza meg, hány réteg zajt adunk össze. Minél magasabb ez a szám, annál részletesebb a végeredmény, de annál többször kell lefutattni a zajgenerálást.
- **Amplitudó (nagyság):** A zaj kezdeti magassága.
- **Frekvencia:** A zaj kezdeti sűrűsége.
- **Lacunarity:** Azt határozza meg, 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 a amplitudó. Az értéke általában 0.5, tehát minden következő réteg fele olyan magas, mint az előző.

5.2. FBM matematikailag 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 amplitudó A, a persistence P, a lacunarity L, a frekvencia pedig F.



5. Ábra
Az oktávok összegzésének szemléltetése 1 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 kettő saját paramétert alkalmazunk.

- **Kontraszt:** A normalizált értéket erre az értékre emeli az eredeti előjel megtartásával. Így nagyobb lesz a kontraszt a nagyságok között.
- **Zajméret:** A kontraszt alkalmazása után egy adott értékkel megszorozza a zajt.

A fraktálzaj pszeudókódban megvalósítva:

8. 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-tól 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 := (abszolutErtek(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.