Perlin-zaj

Pintér Bálint

2026. január 25.

Tartalomjegyzék

1. Perlin-zaj **3**
1.1. Működése összefoglalva 3

2. Előkészítés **3**
2.1. Gradiens tábla 3
2.1.1. Vektor generálás 4
2.2. Permutációs tábla 4

3. Zajszámítás **5**
3.1. Rácspontok meghatározása 5
3.2. Skaláris szorzat kiszámítása 6
3.2.1. Gradiens vektorok kiválasztása 6
3.2.2. A sarkokból a pontba mutató vektorok kiszámítása 6
3.2.3. Skaláris szorzat kiszámítása 7
3.3. Interpoláció 7
3.3.1. Simítófüggvény 7

Forrásjegyzék **8**

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

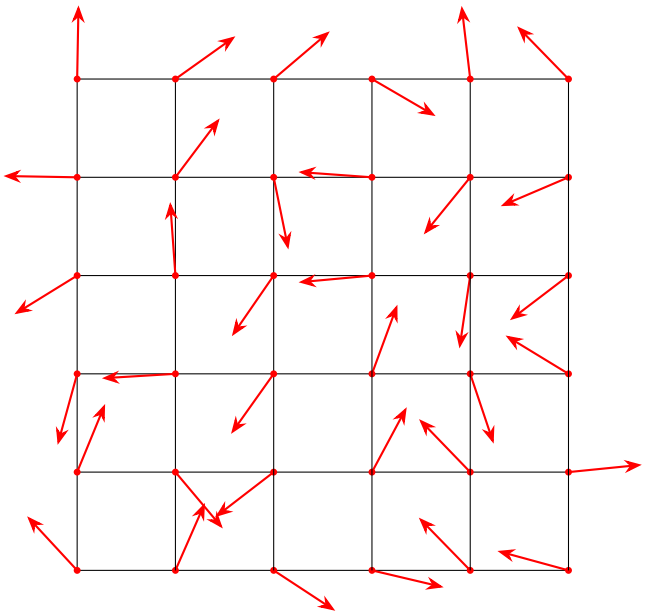
1. **Rács meghatározása:** A zaj dimenziójában egy szabályos rácsot határozunk meg, amelynek minden sarkához egy véletlenszerűen generált egységvektort rendelünk. A rácsvonalak jellemzően az egész koordinátáknál helyezkednek el.
2. **Rácsnégyzeten belüli vektorok kiszámítása:** Kiszámoljuk a rácsnégyzetén 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

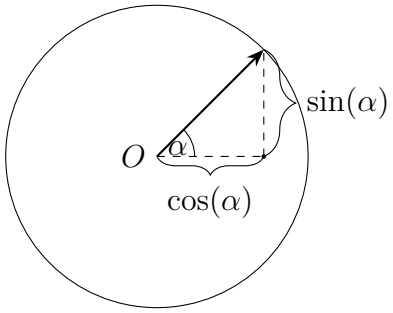
A Perlin-zaj egy úgynevezett gradiens-zaj. Eszerint rácspontokat határozunk meg, amikhez egy véletlenszerű vektort rendelünk. A gradiens tábla ezeket a véletlenszerű vektorokat tárolja. A vektorok dimenziószáma megegyezik a zaj dimenziószámával. (Kétdimenziós zaj $\rightarrow$ kétdimenziós vektor)



1. Ábra
A záj rácsainák szemléltetése.

2.1.1. Vektor generálás

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.

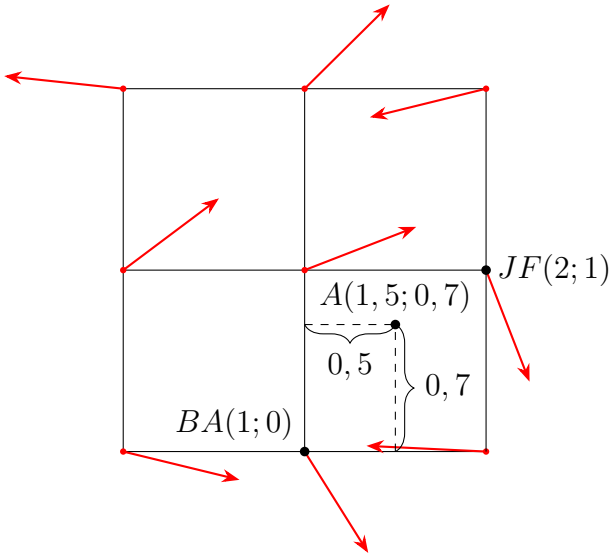
1. Algoritmus: Permutációs tábla létrehozása

```
   Konstans: MaxP=512
   Típus: VéletlenSzámGenerátor=Osztály (
          jelenlegiSzám:Egész
          Függvény Következő:Egész
   )
 1 Eljárás PermutaciosTablaGeneral (Változó: PermutaciosTabla:Tömb(1..MaxP:Egész),
 2 Rand:VéletlenSzámGenerátor ):
       Változó: i, j, temp:Egész
 3
 4     Ciklus i := 1-től 256-ig
 5         PermutaciosTabla[i] := i
 6     Ciklus vége
 7
 8     Ciklus i := 256-tól 2-ig −1-esével
 9         j := Rand.Következő() Mod (i + 1)
10         temp := PermutaciosTabla[i]
11         PermutaciosTabla[i] := PermutaciosTabla[j]
12         PermutaciosTabla[j] := temp
13     Ciklus vége
14
15     Ciklus i := 1-től 256-ig
16         PermutaciosTabla[i + 256] := PermutaciosTabla[i]
17     Ciklus vége
18 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 255-től, 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 koordinátáainak szemléltetése.

Pszeudókódban megvalósítva:

2. Algoritmus: Rácspontok és négyzeten belüli koordináták kiszámolása

```
Típus: Rácspont=Rekord (
        balAlsóPontX, balAlsóPontY:Egész
        jobbFelsőPontX, jobbFelsőPontY:Egész
        relatívX, relatívY :Valós
)
1  Függvény RacspontKiszamolasa(Konstans: x, y: Valós) : Rácspont:
       Változó: jelenlegiRácspont: Rácspont
2      jelenlegiRácspont.balAlsóPontX := (Egész)floor(x) & 256 + 1
3      jelenlegiRácspont.balAlsóPontY := (Egész)floor(y) & 256 + 1
4
5      jelenlegiRácspont.jobbAlsóPontX := (jelenlegiRácspont.balFelsőPontX + 1) & 256
6      jelenlegiRácspont.jobbAlsóPontY := (jelenlegiRácspont.balFelsőPontY + 1) & 256
7
8      jelenlegiRácspont.relatívX := x− floor(x)
9      jelenlegiRácspont.relatívY := y− floor(y)
10     RacspontKiszamolasa := jelenlegiRácspont
11 Függvény vége
```

3.2. Skaláris szorzat kiszámítása

3.2.1. Gradiens vektorok kiválasztása

A gradiens vektorokat a permutációs tábla segítségével válasszuk 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

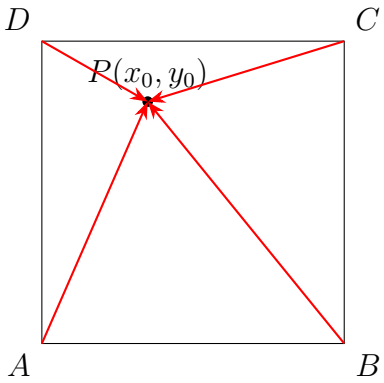
```
Típus: Vektor=Rekord (
        x:Egész
        y:Egész
)
Konstans: MaxP=512, MaxG=256
Változó: PermutaciosTabla:Tömb(1..MaxP:Egész)
Változó: GradiensTabla:Tömb(1..MaxG:Egész)

1 Függvény Hash(Konstans: x, y: Egész) : Egész:
2 |   Hash := PermutaciosTabla[PermutaciosTabla[x] + y]
3 Függvény vége
4
5 Függvény GradiensVektorKivalaszt(Konstans: x, y: Egész) : Vektor:
6 |   GradiensVektorKivalaszt := GradiensTabla[hash(x, y)]
7 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: $\vec{a} \cdot \vec{b} = x_a \times x_b + y_a \times y_b$"fade" vagy "quintic"

4. Algoritmus: Skaláris szorzat

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

3.3. Interpoláció

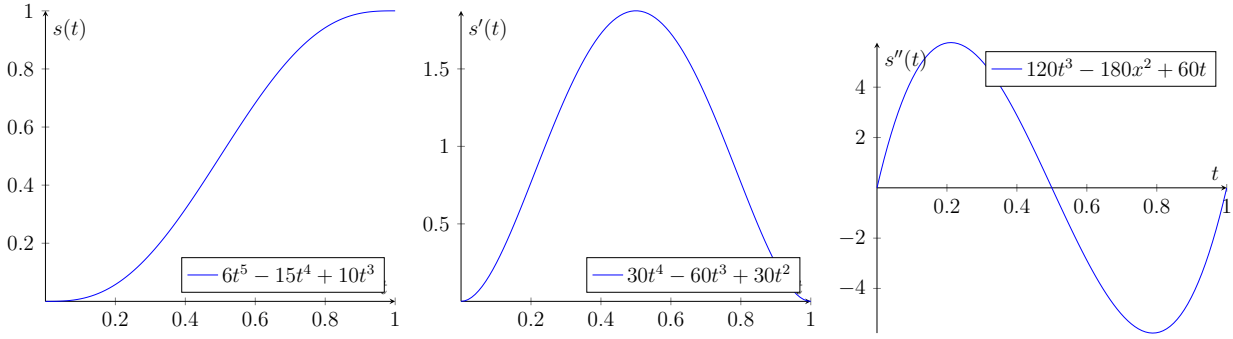
A kapott skalárszorzatokat végül egy simítófüggvény segítségével interpoláljuk a tengelyek mentén.

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 van a rácsnégyzetei közt.



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.