Perlin Noise I

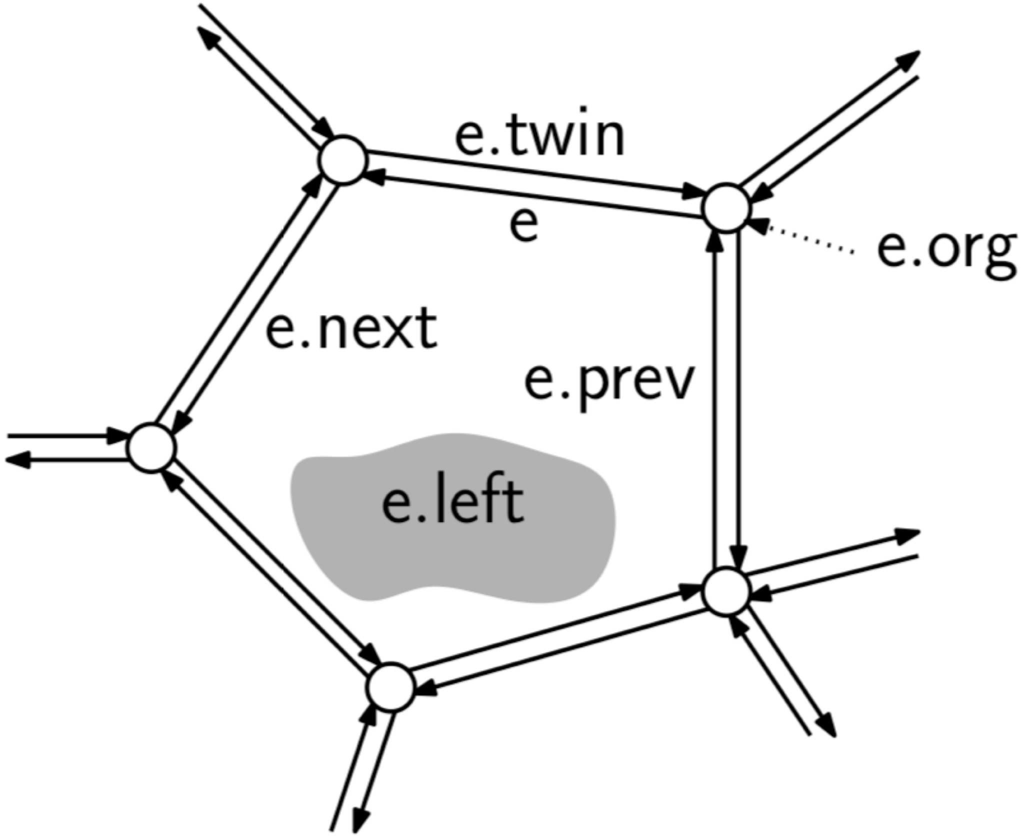
CMSC425.01 Spring 2019

Administrivia

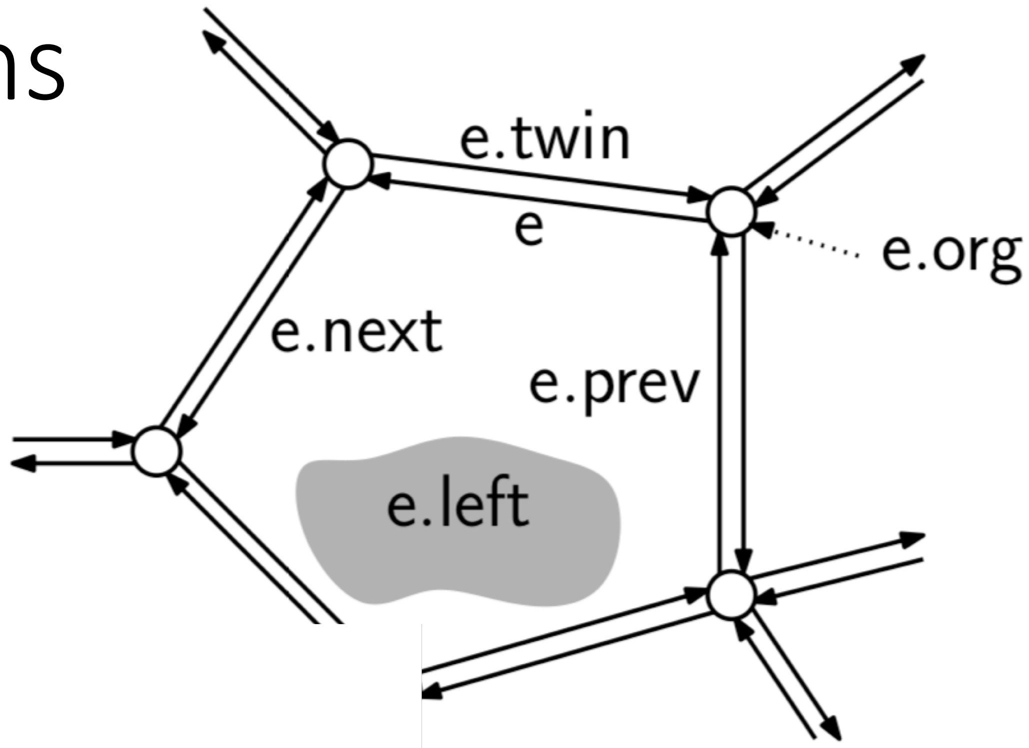
- Google form distributed for grading issues
- Final work outlined soon
  - Final homework
  - Final midterm
  - Final project grading standards

Winged edge representations

- Vertex v has coordinates plus one link to incident edge
- Face f has link to one half edge
- Edge (origin u, destination v) has
- *e.org*: e’s origin
- *e.twin*: e’s opposite twin half-edge
- *e.left*: the face on e’s left side
- *e.next*: the next half-edge after e in counterclockwise order about e’s left face
- *e.prev*: the previous half-edge to e in counterclockwise order about e’s left face (that is, the next edge in clockwise order).



Winged edge representations

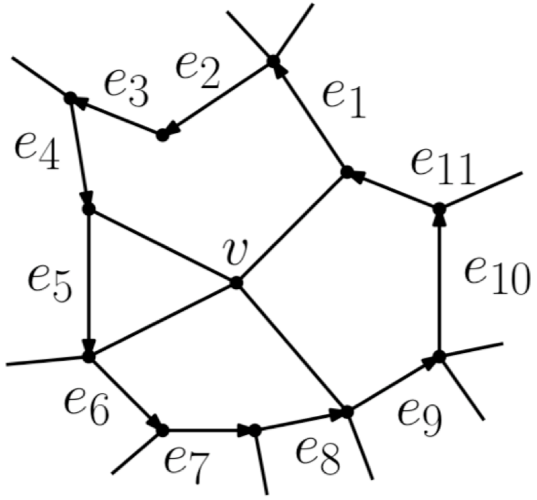


- Question: how traverse all vertices that are neighbors of v in cw order?

```
vertexNeighborsCW(Vertex v) {
    Edge start = v.incident;
    Edge e = start;
    do {
        output e.dest; // formally: output e.twin.org
        e = e.oprev; // formally: e = e.twin.next
    } while (e != start);
}
```

In class exercise

Given vertex v in a cell complex of a 2-manifold, the *link* of v is defined to be the edges that bound the faces that are incident to v, excluding the edges that are incident to v itself. Present a procedure (in pseudocode) that, given a vertex v of a DCEL, returns a list L consisting of the half edges of v's link ordered counterclockwise about v. For example, in the figure below, a possible output would be $\langle e_1, \ldots, e_{11} \rangle$. (Any cyclic permutation would be correct.)



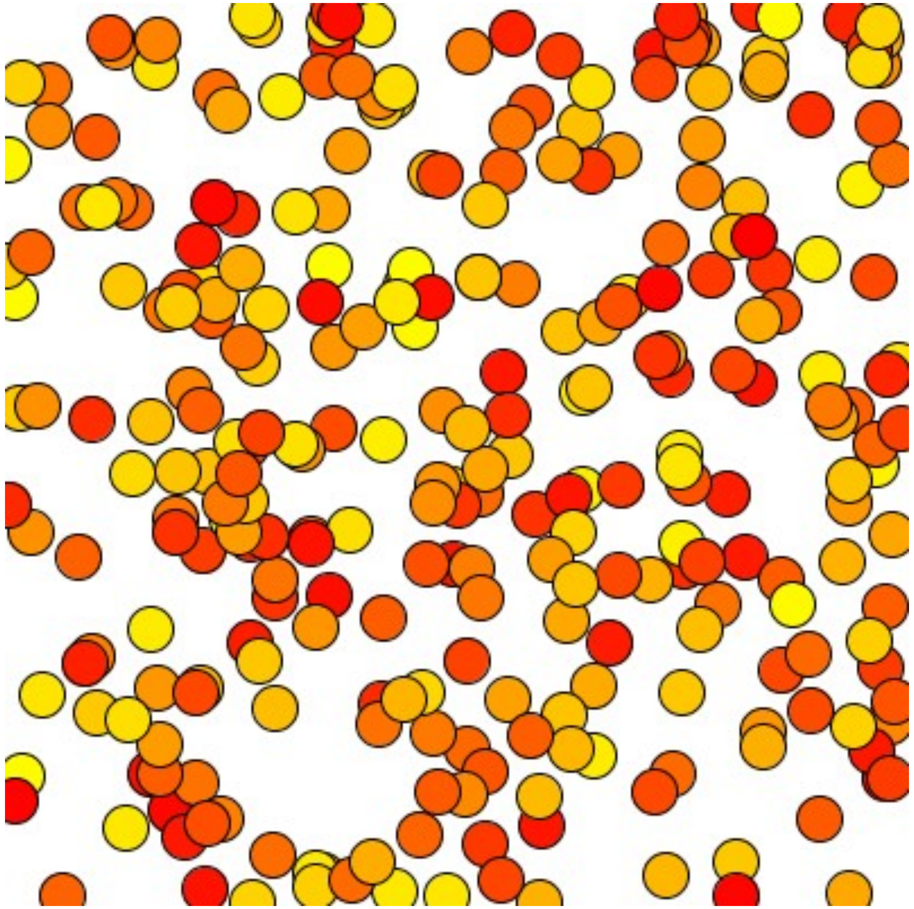
Today’s question

How do you convert the output of
a pseudo-random number generator into
a smooth, naturalistic function?

Randomness – useful tool

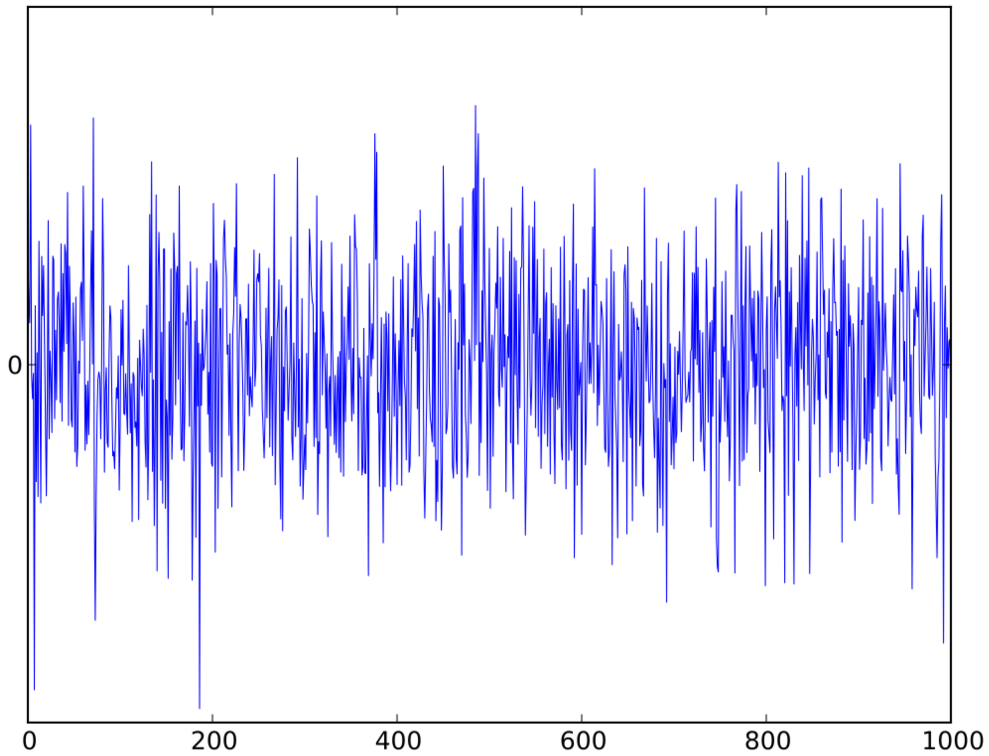
```
// RandomRain
void setup() {
  size(400,400);
  background(255);
  colorMode(HSB,360,100,100);
}

void draw() {
  float x = random(0,400);
  float y = random(0,400);
  float hue = random(0,60);
  fill(hue,100,100);
  ellipse(x,y,20,20);
}
```



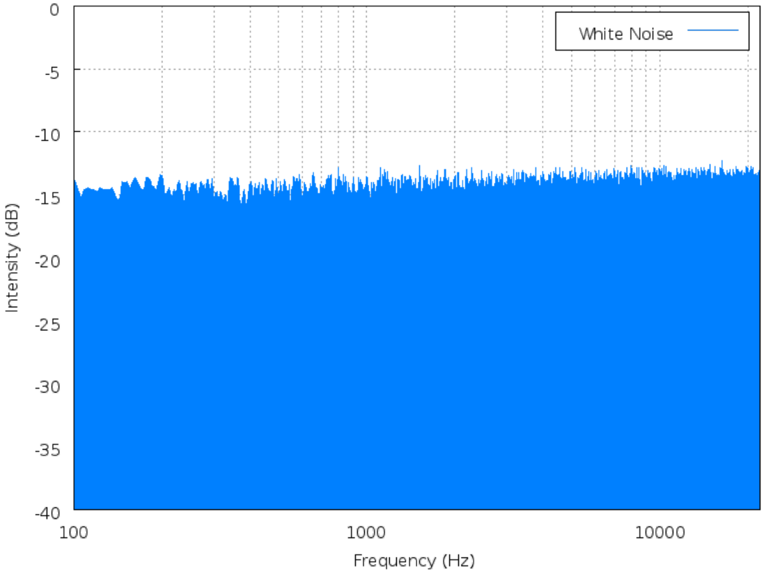
How make it natural and pleasing?

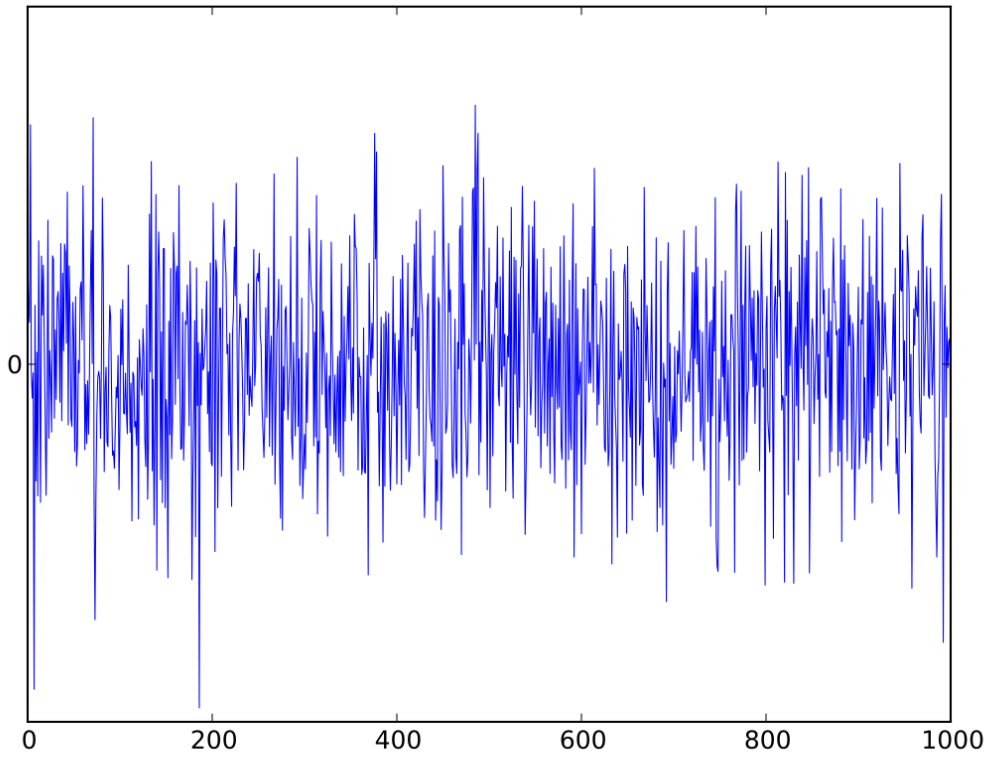
- Pure randomness – white noise
- Each data point independent of rest



White noise

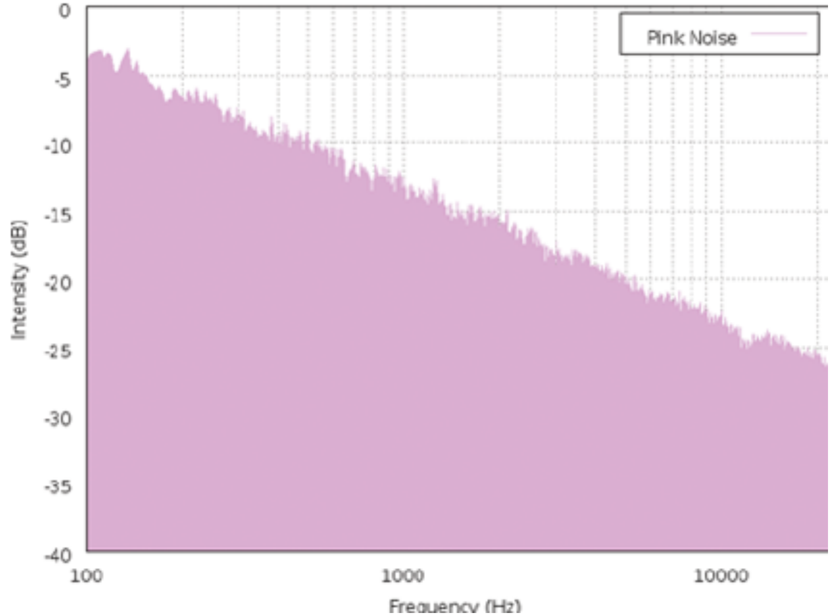
- Pure randomness – white noise
- Each data point independent of rest
- Frequency plot uniform

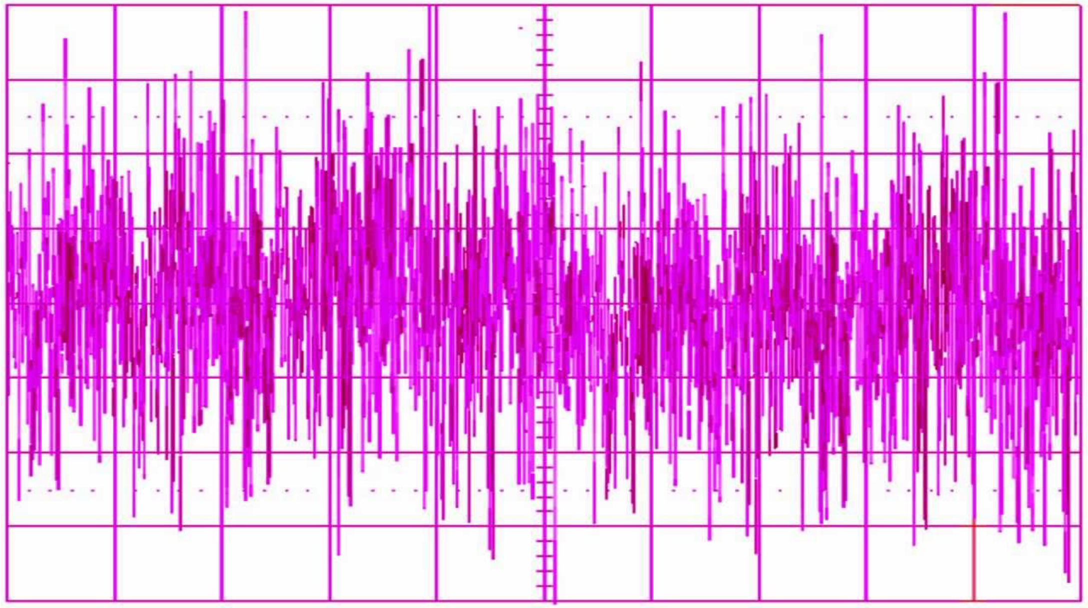




Pink noise

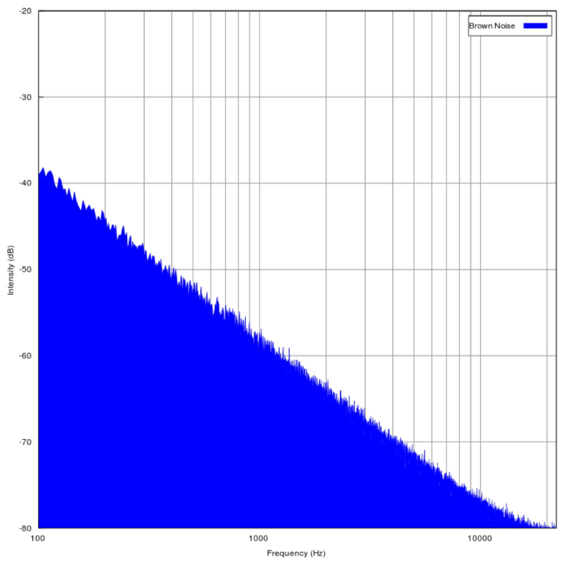
- Shaped randomness – pink noise
- Still independent
- Frequency plot 1/f

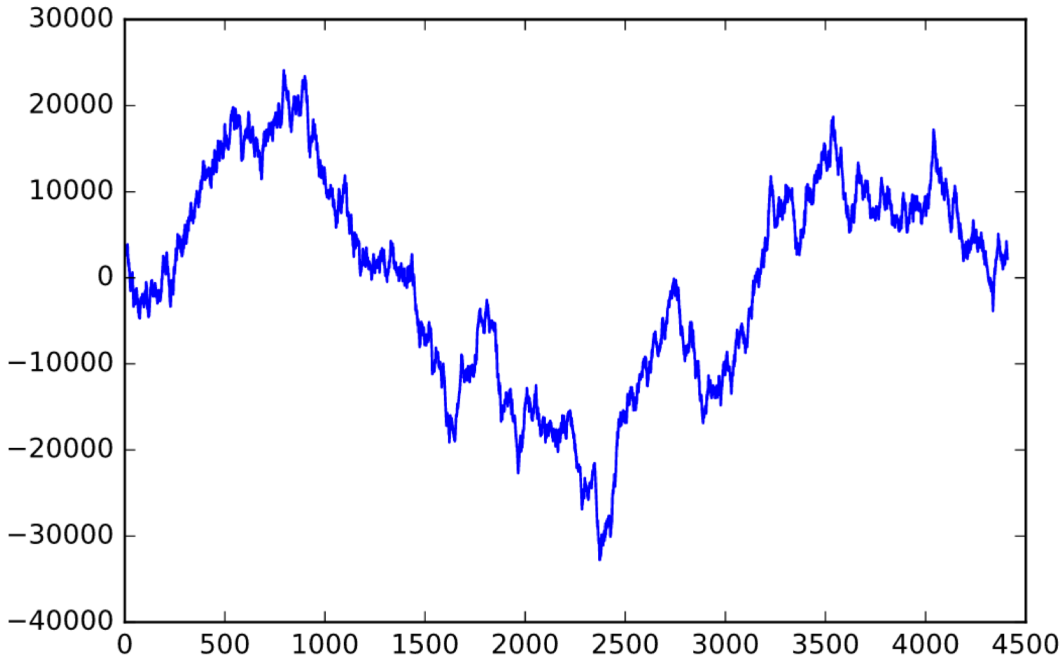




Brown noise

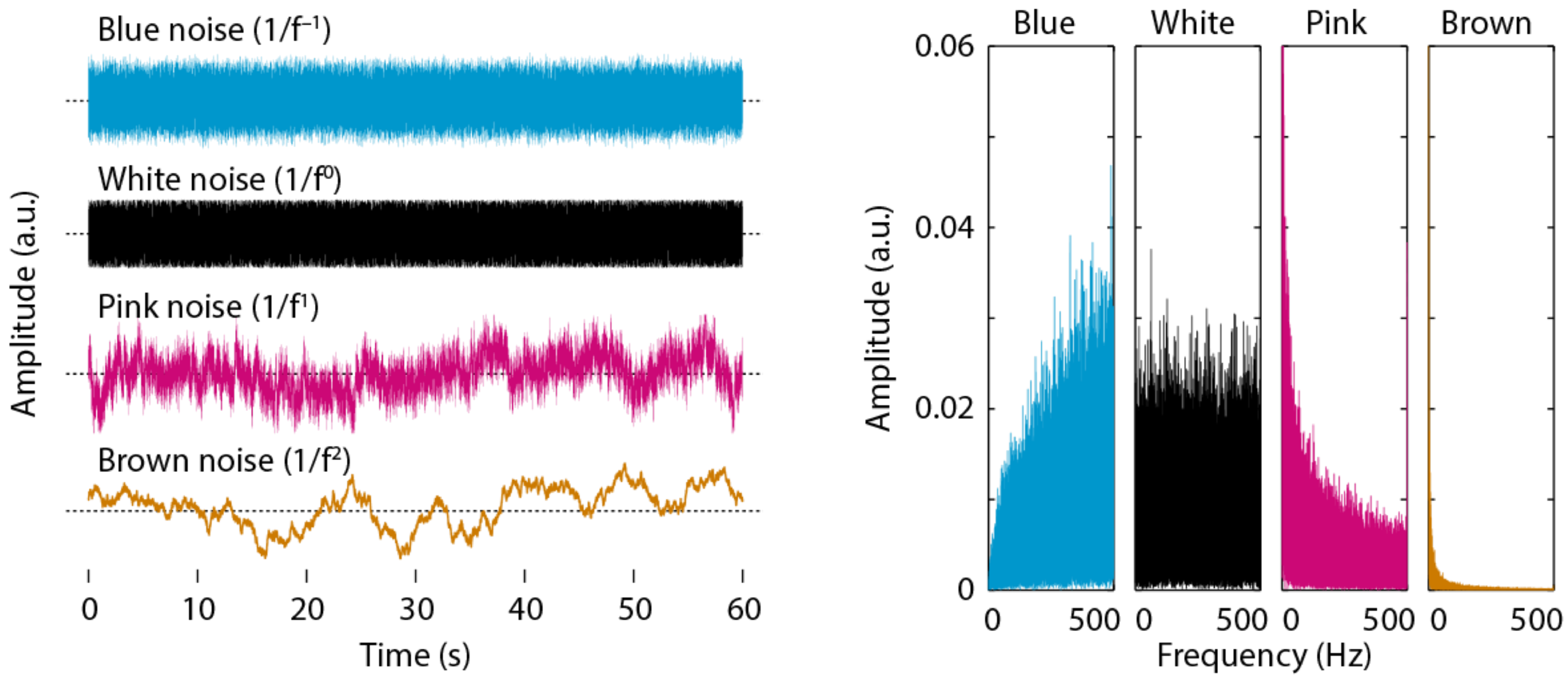
- Random walk – Brownian noise
- Each point random position from last (deltaY = random(-d,d))
- Frequency plot $1/f^2$





Colors of noise

- Music – close to pink noise 1/f
- Natural objects – close to brown $1/f^2$
- Some physical objects – close to white $1/f^0$
- Model object,

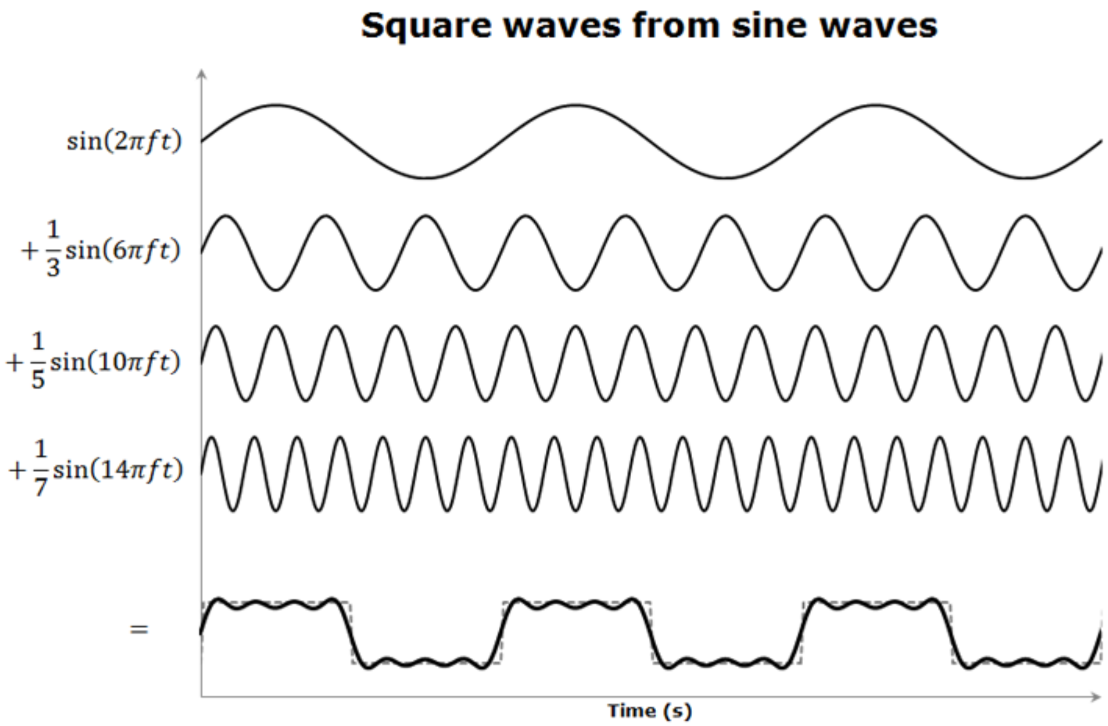


https://archive.org/details/TenMinutesOfWhiteNoisePinkNoiseAndBrownianNoise/BrownianNoise.flac

Generating 1/f^x noise

- Fourier Cosine (sine) Series
- Frequency set by *n*

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right)$$

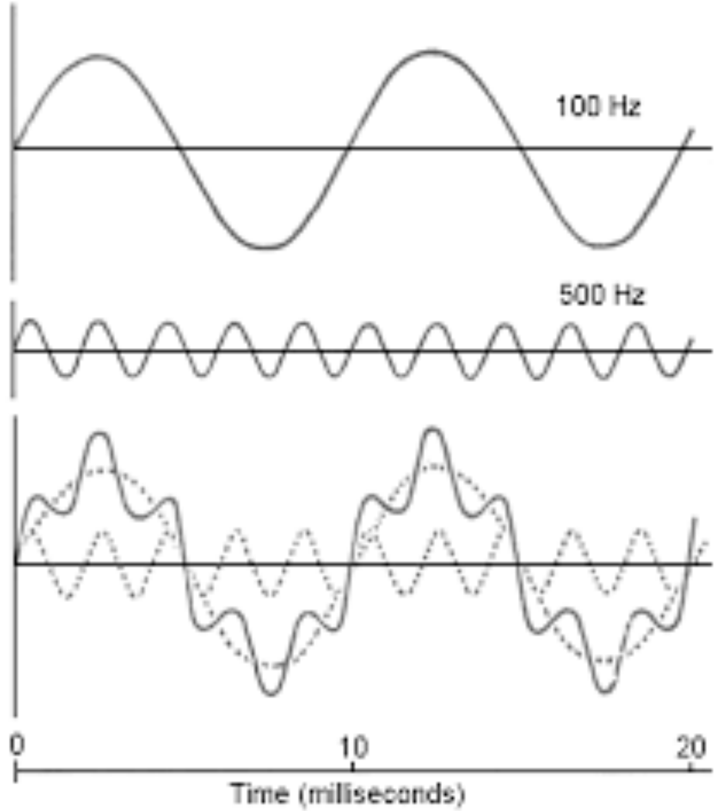


Generating $1/f^x$ noise

- Fourier Cosine (sine) Series
- Frequency set by *n*

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right)$$

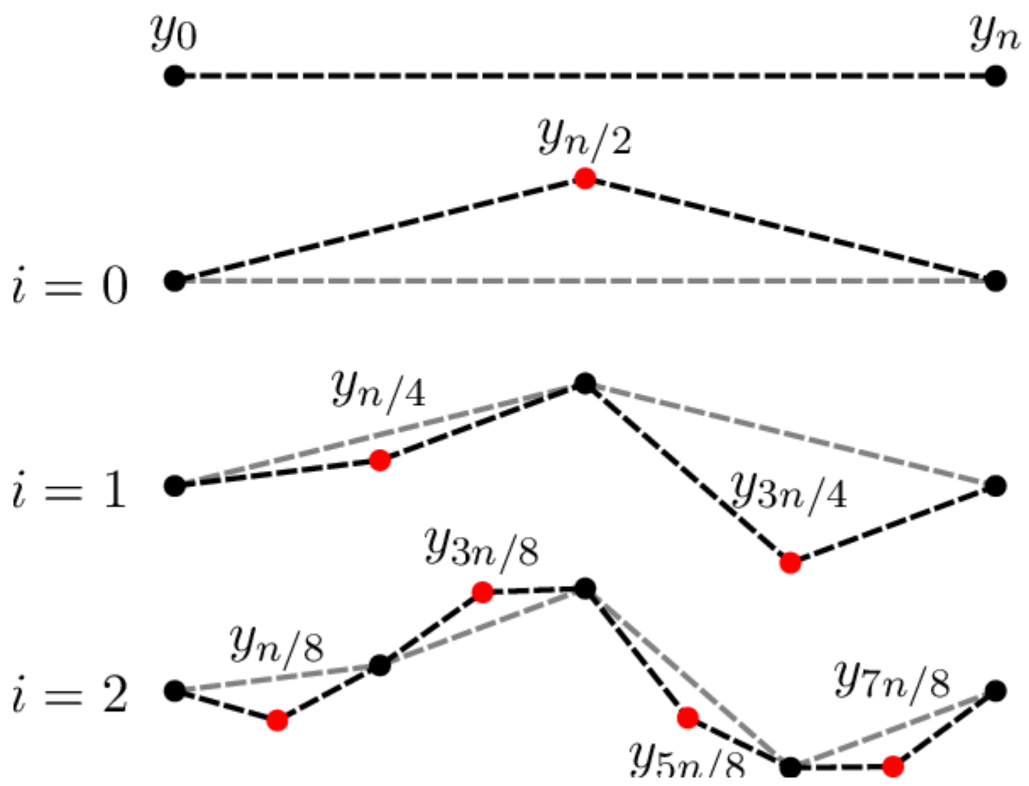
- Generate random terms of frequency, phase
- Decrease amplitude (height) as you increase frequency (n)



More energy higher frequencies => rugged

Application: midpoint displacement

- Recursive curve generation
- Given two points:
  - Create perp bisector
  - Randomly pick t in (-h,h), generate point
  - Repeat for two new line segments
- Works in 3D



Application: midpoint displacement

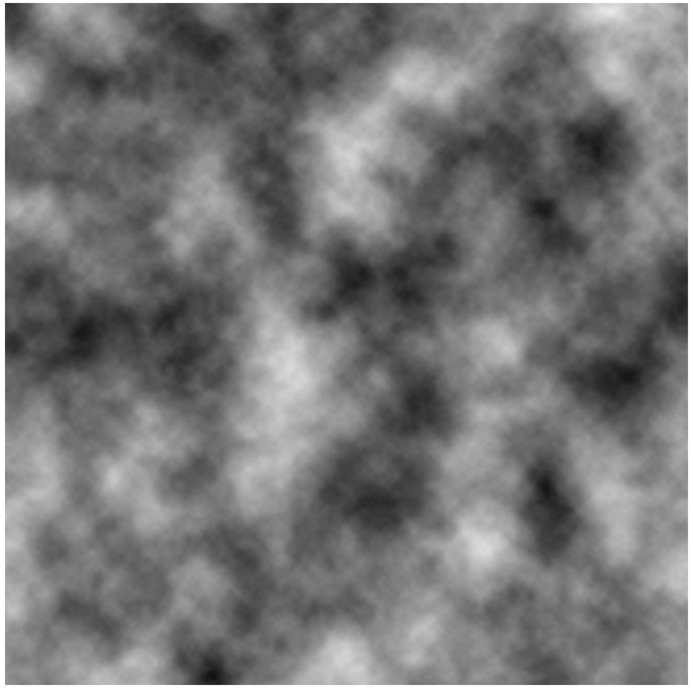
- Recursive curve generation
- Given two points:
  - Create perp bisector
  - Randomly pick t in (-h,h), generate point
  - Repeat for two new line segments
- Works in 3D

- Question

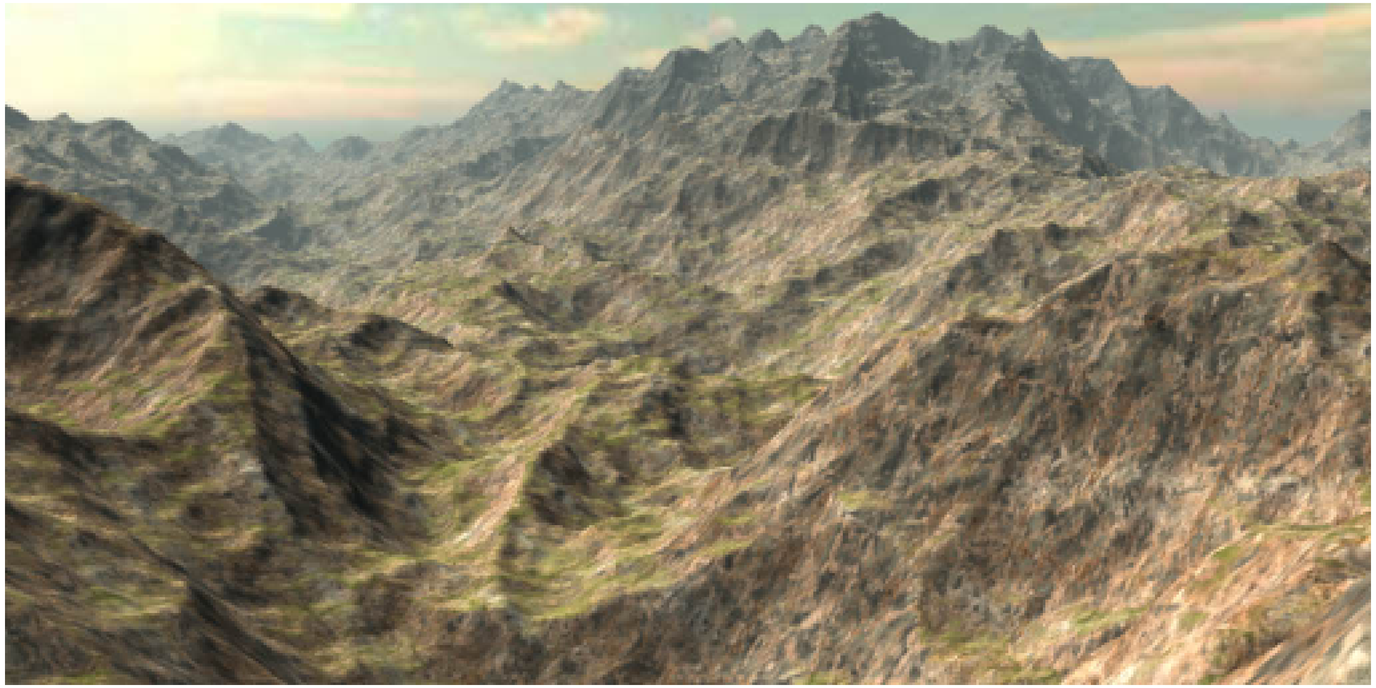
- How would you tune midpoint displacement to get more or less rugged landscapes?

Perlin noise

- Ken Perlin 1983
- (a) height map (b) resulting landscape



(a)



(b)

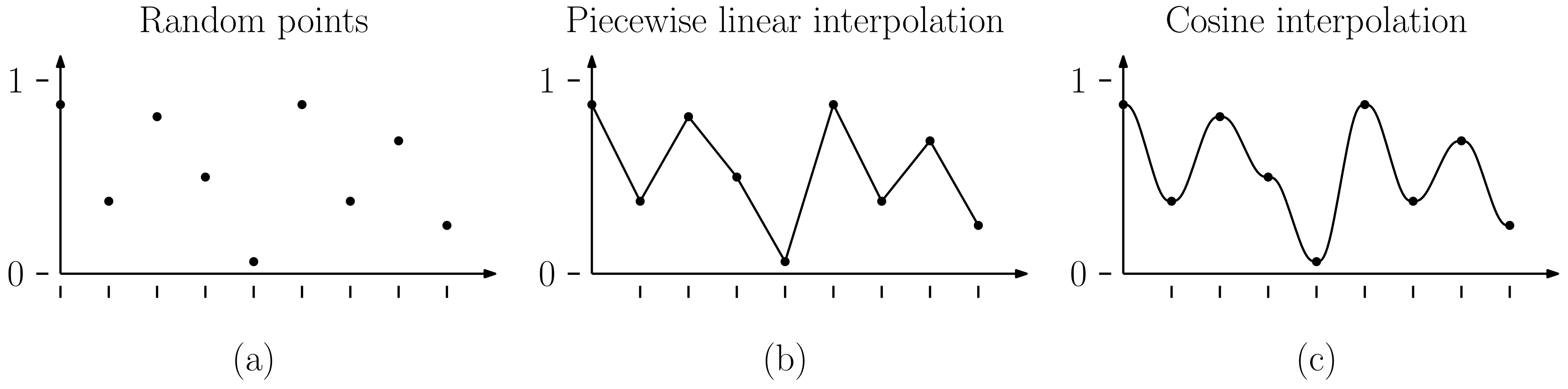
Perlin noise

- Ken Perlin 1983
- Vary frequency component => control ruggedness

Noise fcn f(x) - interpolating random points

- Generate series at uniformly placed $\quad Y = \langle y_0, y_1, y_2, \ldots, y_n \rangle$ $\quad X = \langle x_0, x_1, x_2, \ldots, x_n \rangle$

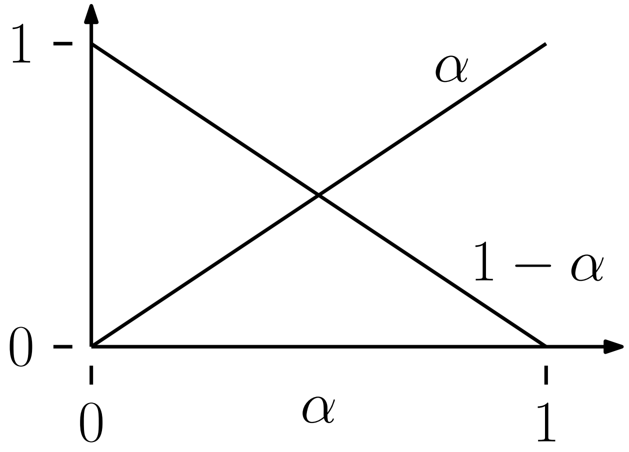
$$f_\ell(x) \;=\; \mathrm{lerp}(y_i, y_{i+1}, \alpha), \qquad \text{where } i = \lfloor x \rfloor \text{ and } \alpha = x \bmod 1$$



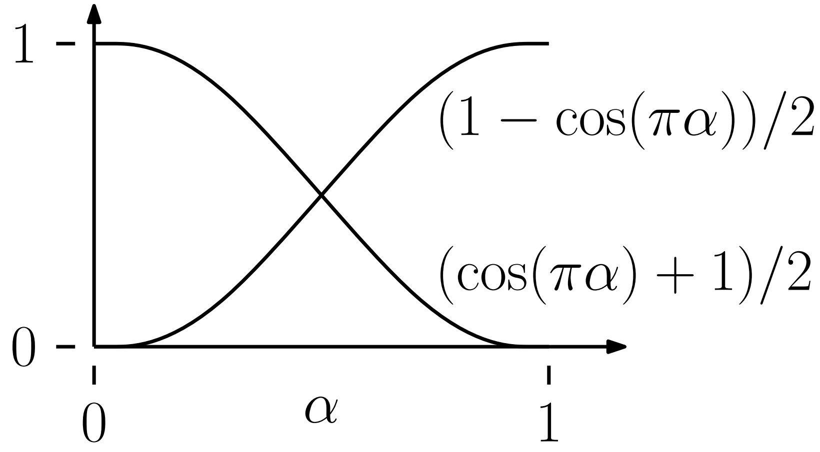
Interpolating weight functions

- Generate series at uniformly placed $Y = \langle y_0, y_1, y_2, \dots, y_n \rangle$ $X = \langle x_0, x_1, x_2, \dots, x_n \rangle$

$$f_\ell(x) \;=\; \mathrm{lerp}(y_i, y_{i+1}, \alpha), \qquad \text{where } i = \lfloor x \rfloor \text{ and } \alpha = x \bmod 1$$



(a)



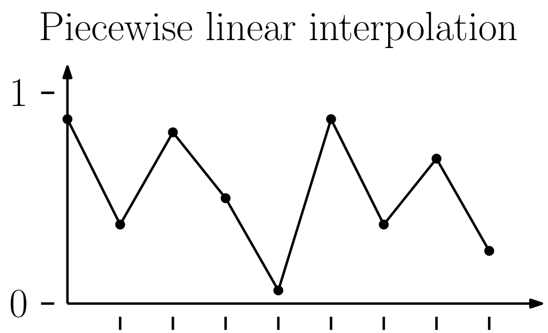
(b)

Interpolating weight functions

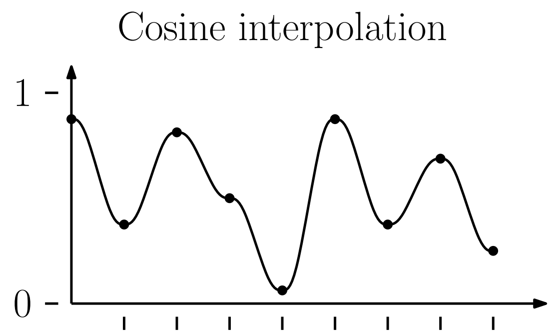
Cosine – smoother because

Slower to leave p0

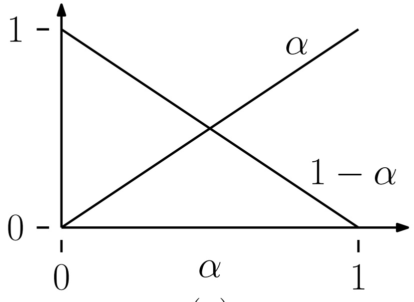
Faster to arrive at p1



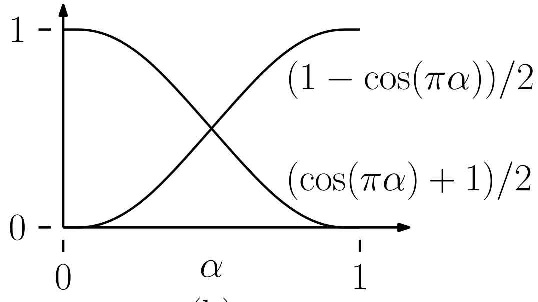
(b)



(c)



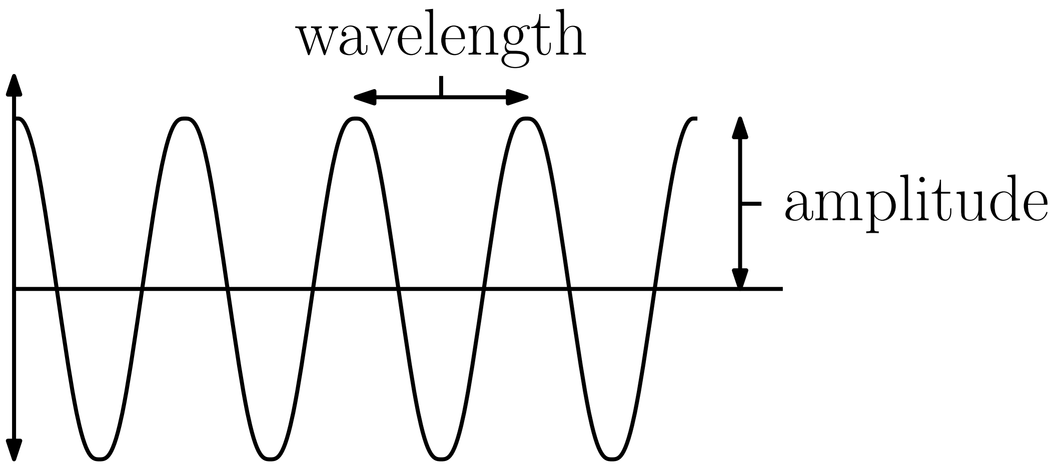
(a)



(b)

$\alpha \sin(\omega t)$

- ***Wavelength***: The distance between successive wave crests
- ***Frequency***: The number of crests per unit distance, that is, the reciprocal of the wavelength
- ***Amplitude***: The height of the crests



- α – amplitude
- ω – frequency
- $2\pi/\omega$ – wavelength

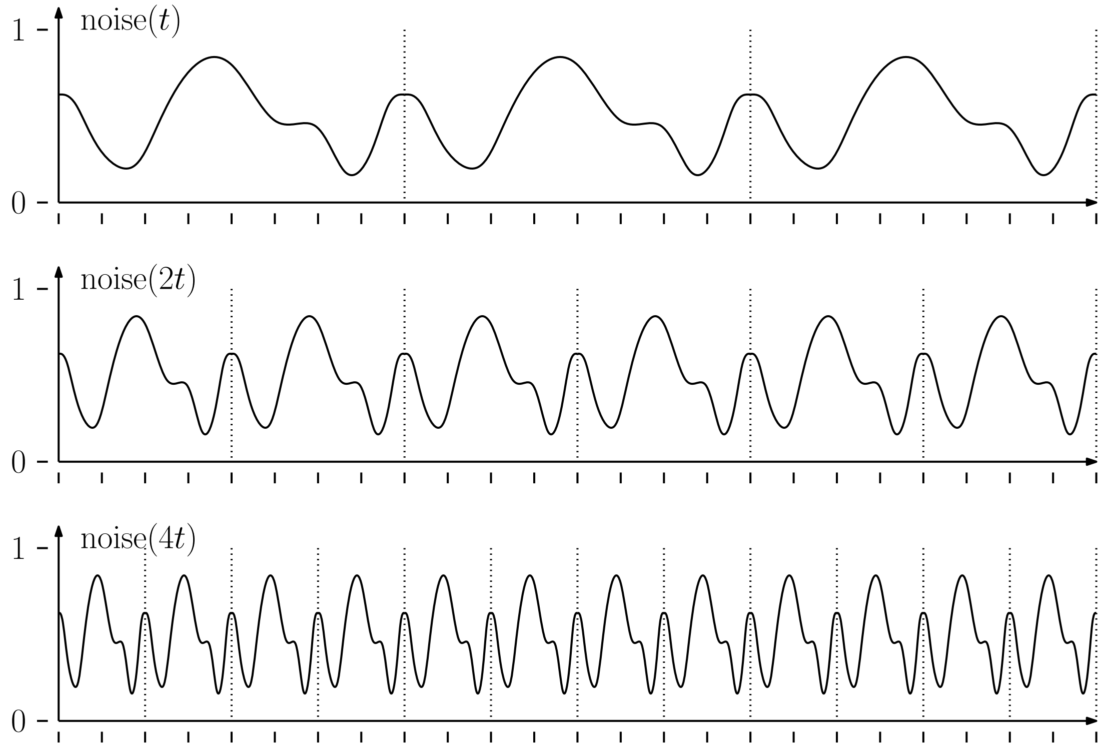
Periodic noise function

- $f(x)$ defined on range [0,n]
- With $f(0) = f(n)$

- Now define

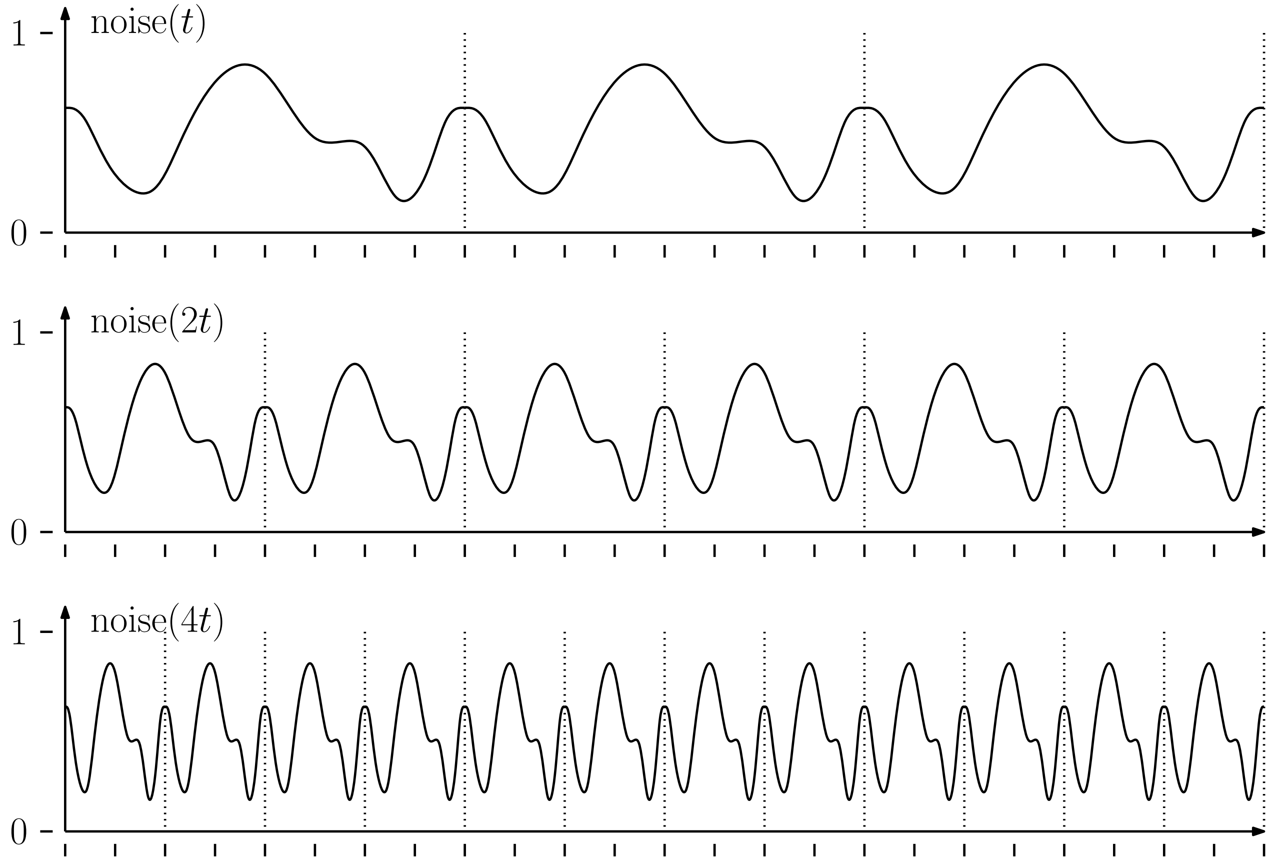
- $\text{noise}(t) = f(t\ mod\ n)$

- *Not sine* – randomly created
- Same curve – self-similar



Frequency octaves

- $\text{noise}(t)$
- $\text{noise}(2t)$
- $\text{noise}(4t)$
- …
- $\text{noise}(2^i t)$

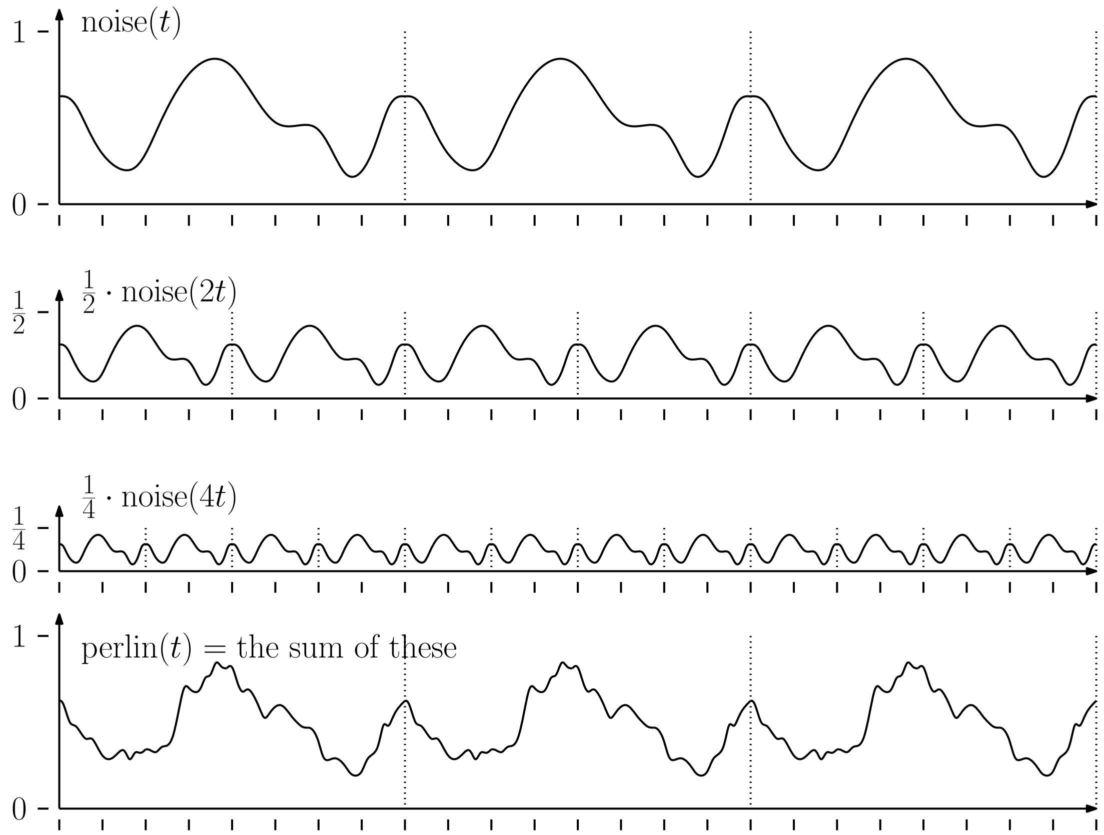


Persistence

- $p^0 \text{noise}(t)$
- $p^1 \text{noise}(2t)$
- $p^2 \text{noise}(4t)$
- …
- $p^i \text{noise}(2^i t)$

$$p = \frac{1}{2}$$

$$\text{perlin}(t) = \sum_{i=0}^{k} p^i \text{noise}(2^i t)$$



Perlin noise summary

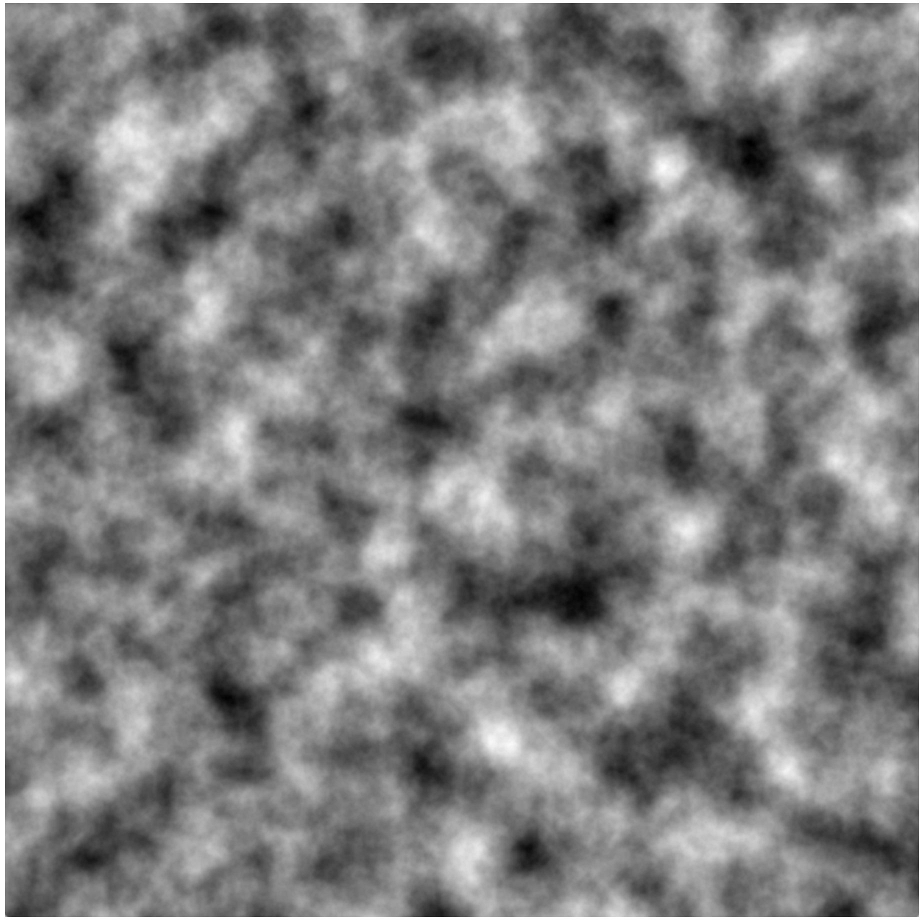
- Perlin noise is
  - Constant after generation
  - Periodic
  - Fractally self-similar

- Unity

  public static float **PerlinNoise**(float **x**, float **y**);

  returns value in [0,1.0]

  (Set y = constants to get 1D function)



https://cpetry.github.io/TextureGenerator-Online/

Unity: Scripting Perlin => Terrain

```
float[,] heights = new float[width, height];

for (int i = 0; i < width; i++) {
  for (int k = 0; k < height; k++) {
      heights [i,k] = baseHeight +  (float)hillHeight *
            (Mathf.PerlinNoise (
            ((float)i / (float)width) * tileSize,
            ((float)k / (float)height) * tileSize));
            }
       }


terrain.terrainData.SetHeights (0, 0, heights);
```

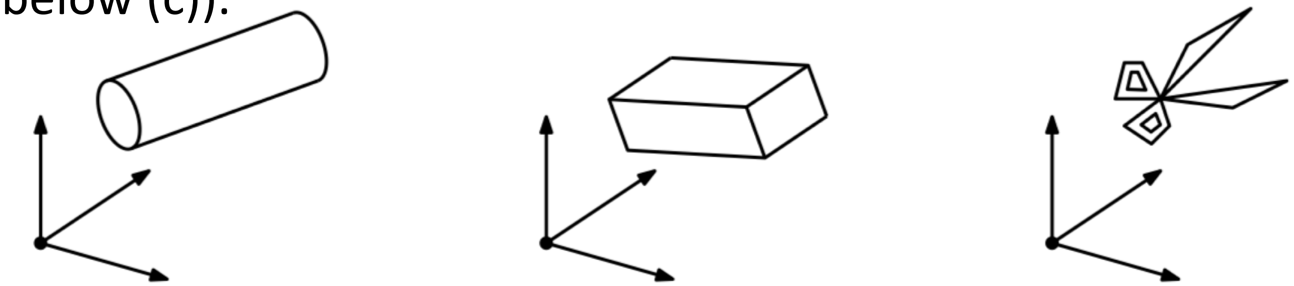
https://forum.unity.com/threads/perlin-noise-based-terrain-hill-generator-working-script.214701/

Question

- How would the idea of multiple scales apply to
- Generating plants for a game
- Generating cities/towns/etc for a game
- Creating plot variations/bosses

Problem – configuration spaces

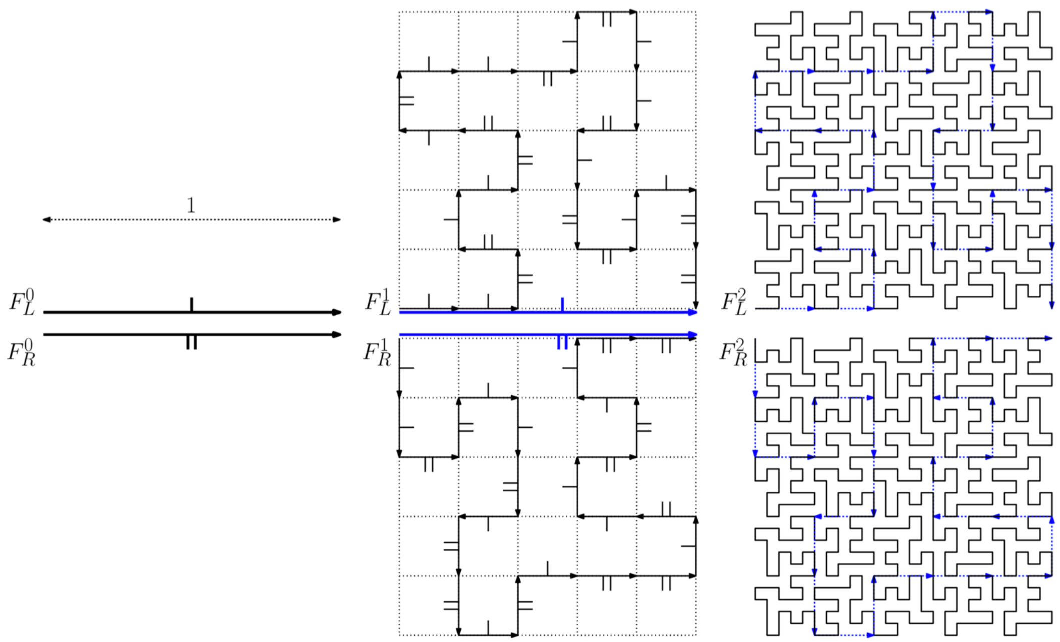
- How many dimensions are there in the configuration spaces for each of the following motion-planning problems. Justify your answer in each case by explaining what each coordinate of the space corresponds to.
- (i) Moving a cylindrical shape in 3-dimensional space, which may be translated and rotated (see the figure below (a)).
- (ii) Moving a brick in 3-dimensional space, which may be translated and rotated (see the figure below (b)).
- (iii) Moving a pair of scissors in 3-dimensional space, which may be translated, rotated, and swung open and closed (see the figure below (c)).



(a) (b) (c)

Problem – Fractal curve

- Derive an L-system that generates FL and FR. In particular, please provide the recursive rules for FL and FR.
- Consider the curve FL in the limit. Derive its fractal dimension.
- Each generation distances are scaled by σ = 1/5, and each individual segment of the basic length is replaced by 25 segments of the next smaller size.



Problem – DECL intersection

- Compute a list L = ⟨e1,e2,...,em⟩ of edges that intersect a line segment **ab**
- Given:
  - Faces fa and fb that contain a and b, respectively
  - Function e.cross(a,b) that returns true/false if edge e crosses **ab**