Noise

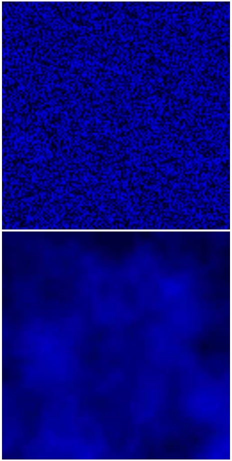
Geoffrey Matthews

Department of Computer Science
Western Washington University

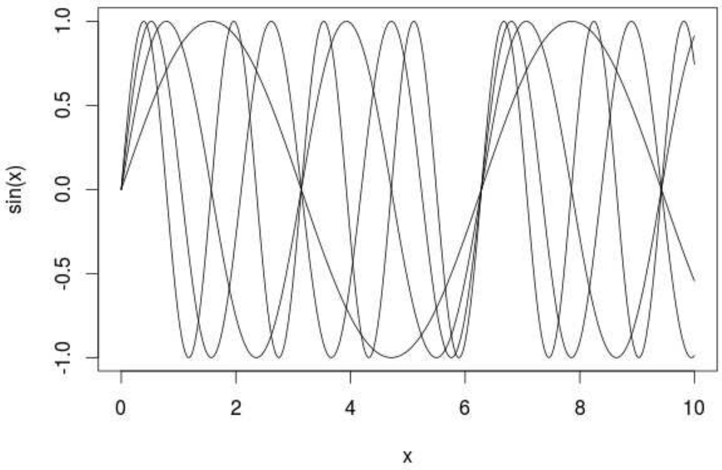
Fall 2012

Noise

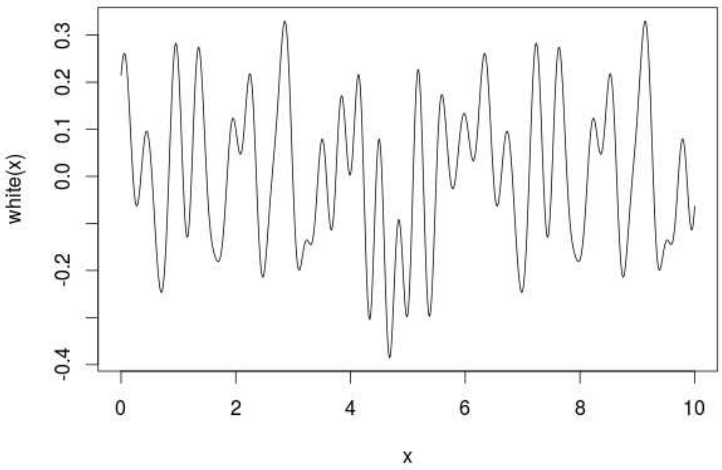
- Idea is to define a function that looks like the randomness we find in nature.
- Dirt, clouds, waves on the ocean, skin, all exhibit randomness.
- However, it is not **white noise**, seen at upper right. It is much smoother, as seen at lower right.
- Real world noise exhibits smoothness, going from dark to light gradually, with dark and light patches randomly distributed.
- How can we simulate that? We will add up smoothed white noise at different wavelengths, with the *amplitude* of the noise proportional to its *wavelength*.
- This is called **pink noise**.



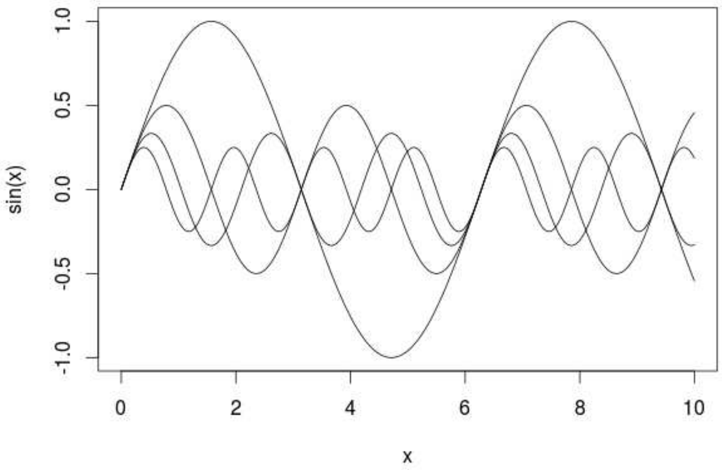
Adding up sin curves



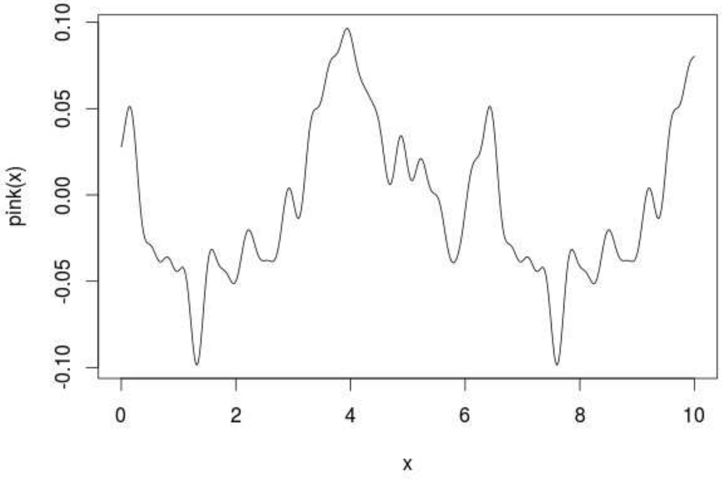
Adding up sin curves



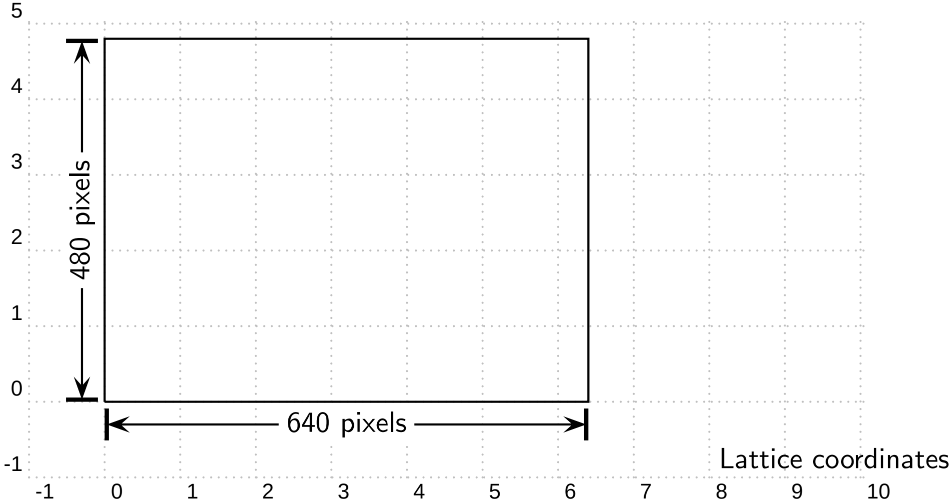
Adding up scaled sin curves



Adding up scaled sin curves

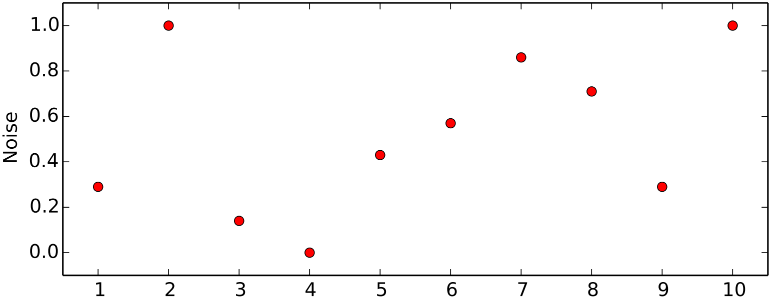


Defining a lattice over screen space.



- Divide by the *wavelength*, the distance between lattice lines.
- Optionally, flip the origin to lower left.

latticeNoise: A white noise value for lattice points



- We start with just a single dimension, x (scaled to lattice space).
- `latticeNoise` only defined at integers.
- White noise values between 0 and 1
- Our space is generally too big for an array for all those numbers, so we will find a function `latticeNoise(x)` that uses less memory but is still very fast.

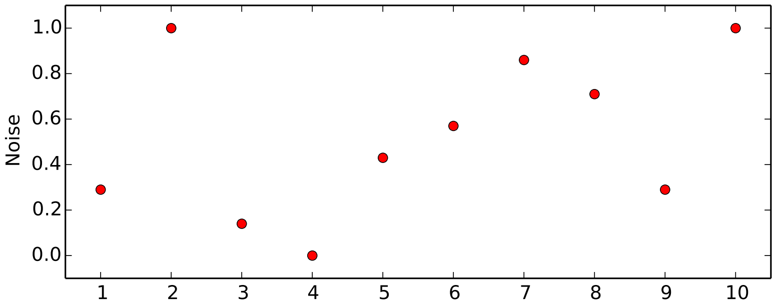
latticeNoise: a Simple Implementation

- Pick a modest array size, normally $n = 256$
  ... but for illustration we use $n = 8$
- Create an array of n floats evenly spaced between 0 and 1, *e.g.*,
  `noiseTable = (0.00,0.14,0.29,0.43,0.57,0.71,0.86,1.00)`
- We could randomize this table, but instead we randomize the index into this table. We use a permutation of the first n integers, *e.g.*,
  `hashTable = (5,2,7,1,0,3,4,6)`
- Our function becomes
  `latticeNoise(x): return noiseTable[hashTable[x%n]],`
- This will be very fast.
- *Note: The sequence repeats every n integers, but that won't be as important when we move up to 2 and 3 dimensions.*

latticeNoise: a Simple Implementation

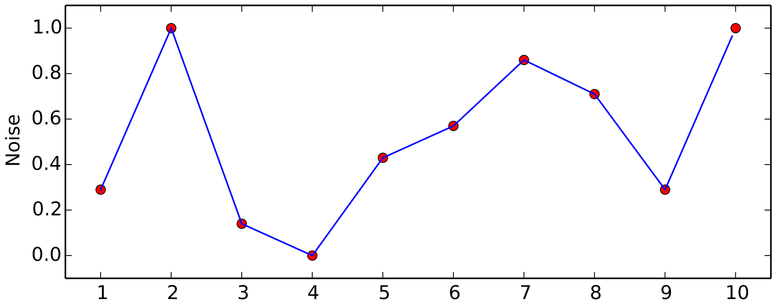
- Pick a modest array size, normally $n = 256$
  ... but for illustration we use $n = 8$
- Create an array of n floats evenly spaced between 0 and 1, *e.g.*,
  `noiseTable = (0.00,0.14,0.29,0.43,0.57,0.71,0.86,1.00)`
- We could randomize this table, but instead we randomize the index into this table. We use a permutation of the first n integers, *e.g.*,
  `hashTable = (5,2,7,1,0,3,4,6)`
- Our function becomes
  `latticeNoise(x): return noiseTable[hashTable[x%n]]`,
- This will be very fast.
- *Note: The sequence repeats every n integers, but that won't be as important when we move up to 2 and 3 dimensions.*
- Do `noiseTable` and `hashTable` have to be the same size?

latticeNoise: A white noise value for lattice points



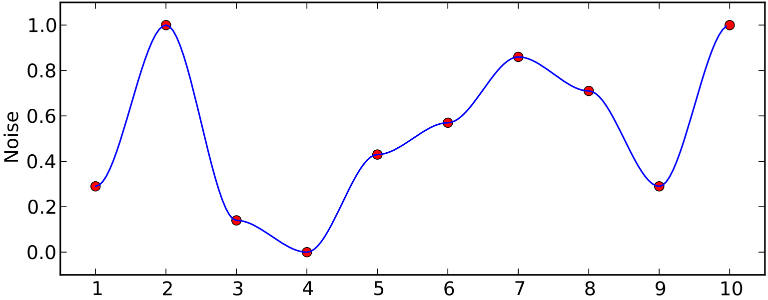
- `hashTable = (5,2,7,1,0,3,4,6)`
- `noiseTable = (0.00,0.14,0.29,0.43,0.57,0.71,0.86,1.00)`
- `latticeNoise(x): return noiseTable[hashTable[x%n]]`

lerpNoise: filling in between the lattice points



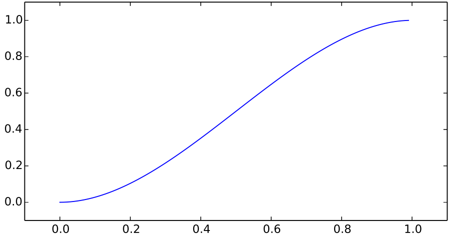
- One option is to use linear interpolation.
- Better than white noise.
- Easy to compute: `lerp(pct, a, b): a + pct*(b-a)`
- However, makes for a spikey curve, not like the noise we find in nature.

smerpNoise: Smoothly Interpolate Between the Integers



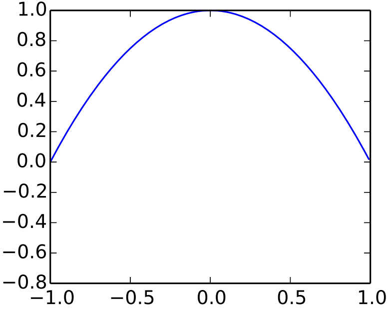
- This looks more like what we want.
- There are many ways to compute smooth curves through a set of points. Look up **B-splines**, **Hermite curves**, and **Bezier curves**.
- Here we want a simpler approach: an easily computable **S-curve** between each pair of points.

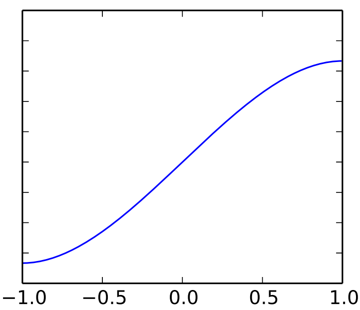
S-curve



- There are many S-curves, such as the **logistic function**, or even the **cosine** function between 0 and π, but we can come up with our own easily enough. It would help if we could avoid transcendental functions, too. (Why?)
- The S-curve we need smoothly maps the 0-1 interval to itself, and is horizontal at both ends.
- What might be a good approach?

S-curve

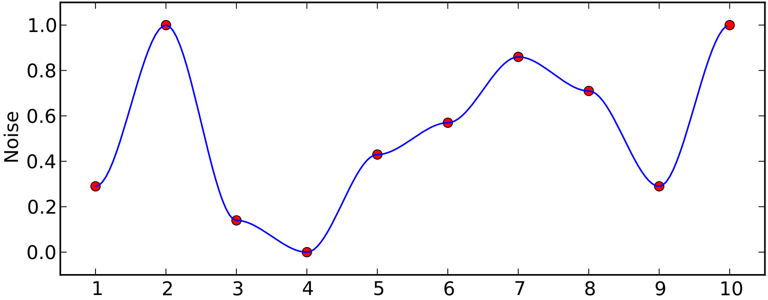




- On the left we plot $1 - x^2$. This is zero at ± 1, which means any curve that has this as its derivative will be horizontal at ± 1
- On the right we plot $x - \frac{x^3}{3}$. Shifting and scaling leads to:

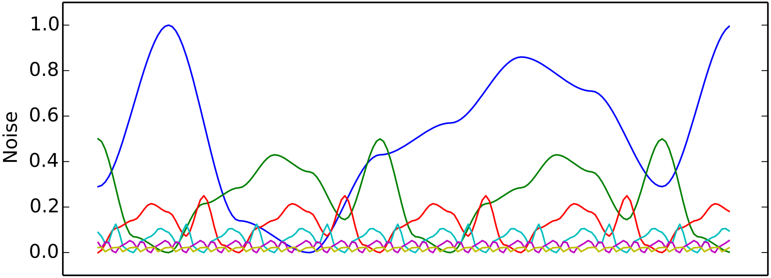
```
smerp(pct, a, b):
    x = ???
    return a + (??? + ???*x - ???*x*x*x)*(b-a)
```

smerpNoise: Smoothly Interpolate Between the Integers



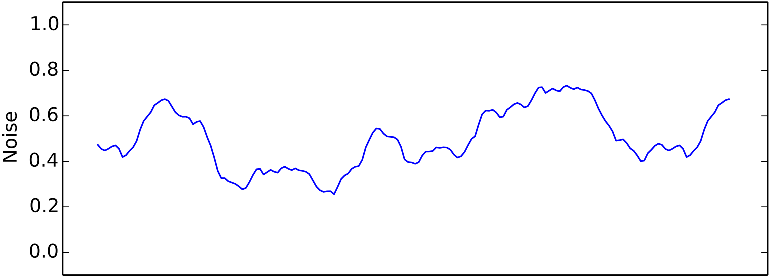
- The curve height is always between 0 and 1.
- The derivative of the curve is zero at lattice points.
- Highest and lowest points will always be at lattice points.
- Repeats after *n* lattice points.
- We would like the curve to have more detail at smaller scales, so we move on to *pink noise*.

Pink Noise



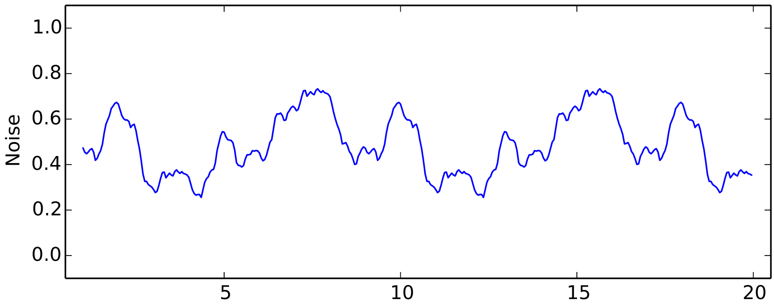
- Lattice distance is arbitrary, so we can scale the lattice frequency.
- We can also scale the maximum amplitude.
- Here we show curves for i from 0 to 5:
  - `smerpNoise(x*2**i)/2**i`
- Amplitude has a $1/f$ relationship to frequency.

Pink Noise



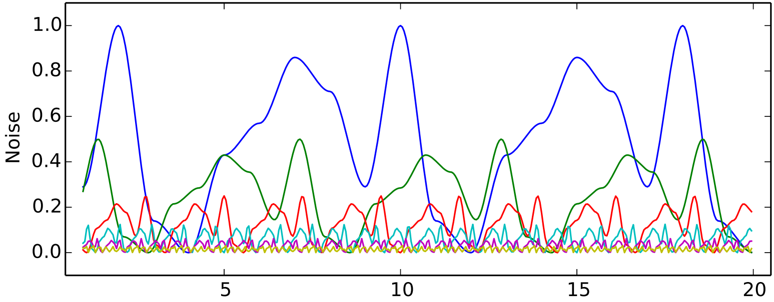
- Here we show the sum for *i* from 0 to 5 of: `smerpNoise(x*2**i)/2**i`
- Sum is divided by 2. Why?
- This is also an example of **fractal Brownian motion** (fBm).

Pink Noise



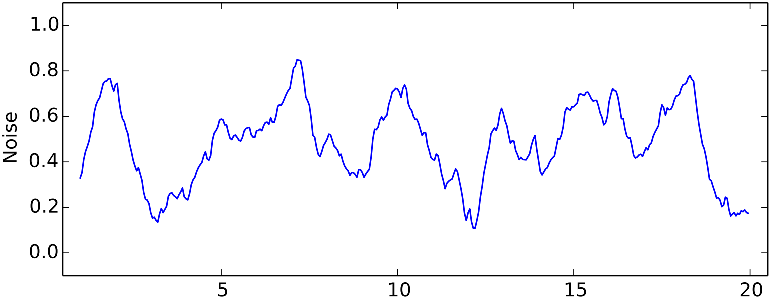
- If we look at a larger range, we can see that it still repeats after *n* integers. Why?
- Could we fix that? How?

Pink Noise without repeats



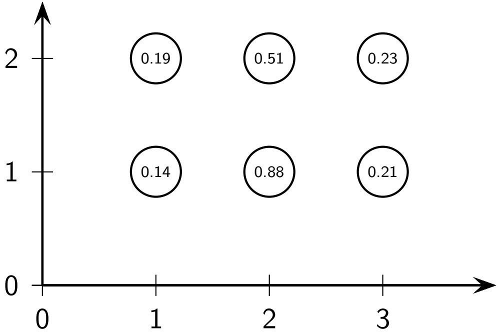
- We rescale each wavelength by a small amount, so they don't line up.

Pink Noise without repeats



- When we add them up they don't repeat for a long time.
- This would be even more effective with a larger *n*, *e.g.* 256.
- However, we won't do this because the problem will be solve itself in higher dimensions.

Lattice Noise in 2D

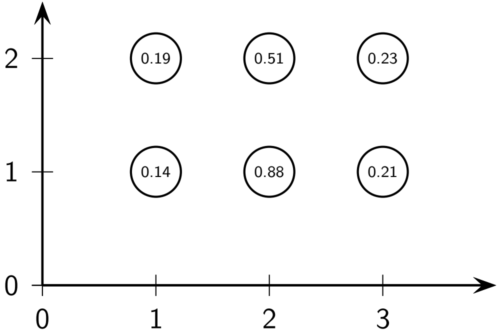


- For lattice noise in 2D we could use a two dimensional array of noise values, and a hashed lookup into that table:

```
latticeNoise(x,y):
    noiseTable[hashTable[x%n], hashTable[y%n]]
```

- ... but there's a better way.

Lattice Noise in 2D



- We hash both x and y to get a single lookup into the same one-dimensional noise table:

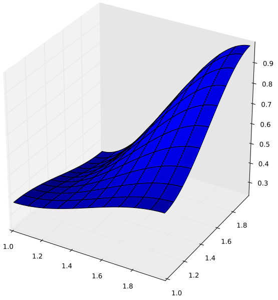
```
latticeNoise(x,y):
    noiseTable[hashTable[(x + hashTable[y%n])%n]]
```

- This solves the repetition problem, too. Why?
- What would we do in 3D? 4D?

Smooth Noise in 2D

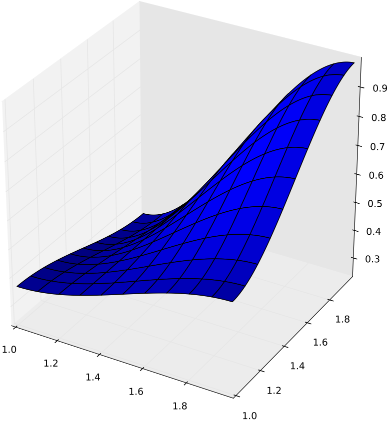
- Now that we have our random values at the corners, we need to smoothly interpolate the ones in between:

```
smerpNoise2(x, y):
    intx = floor(x)
    inty = floor(y)
    pctx = x - intx
    pcty = y - inty
    aa = latticeNoise2(intx, inty)
    ab = latticeNoise2(intx, inty+1)
    ba = latticeNoise2(intx+1, inty)
    bb = latticeNoise2(intx+1, inty+1)
    xa = smerp(pctx, aa, ba)
    xb = smerp(pctx, ab, bb)
    return smerp(pcty, xa, xb)
```

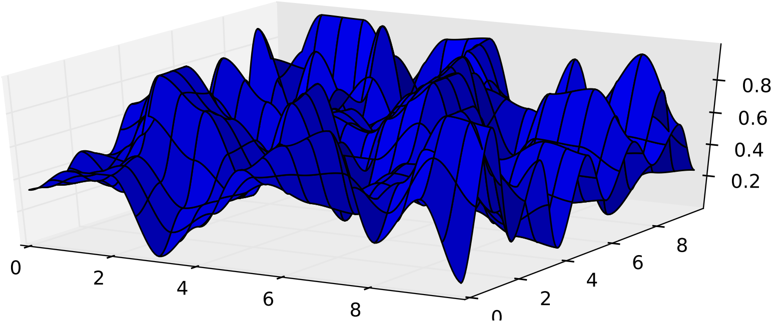


Smooth Noise in 2D

- What does the interpolation function look like in 3D?
- 4D?
- Do you see a problem?
- Look up **simplex noise**.

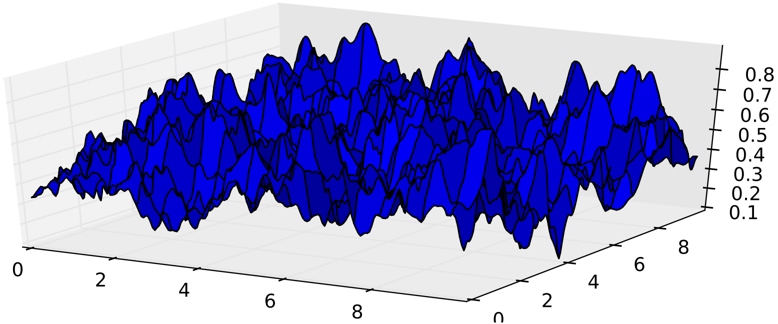


Smooth Noise in 2D



- Doing this over the whole lattice gives us smooth noise in 2D.

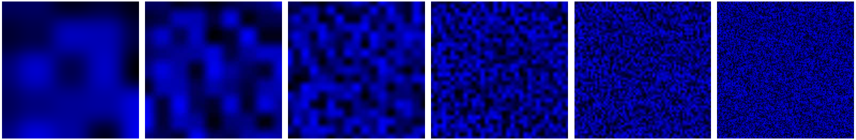
Pink Noise in 2D



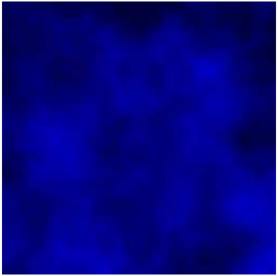
- If we add up many smooth noises of shorter wavelength and smaller amplitude we end up with 2D pink noise.

Pink Noise in 2D

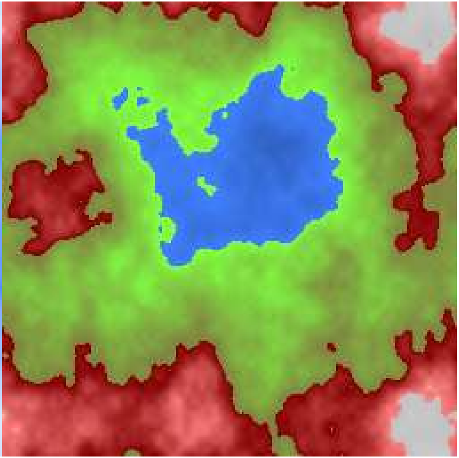
- Smooth noises at diminishing wavelengths:



- The pink noise sum of the above, amplitude diminishing with wavelength:



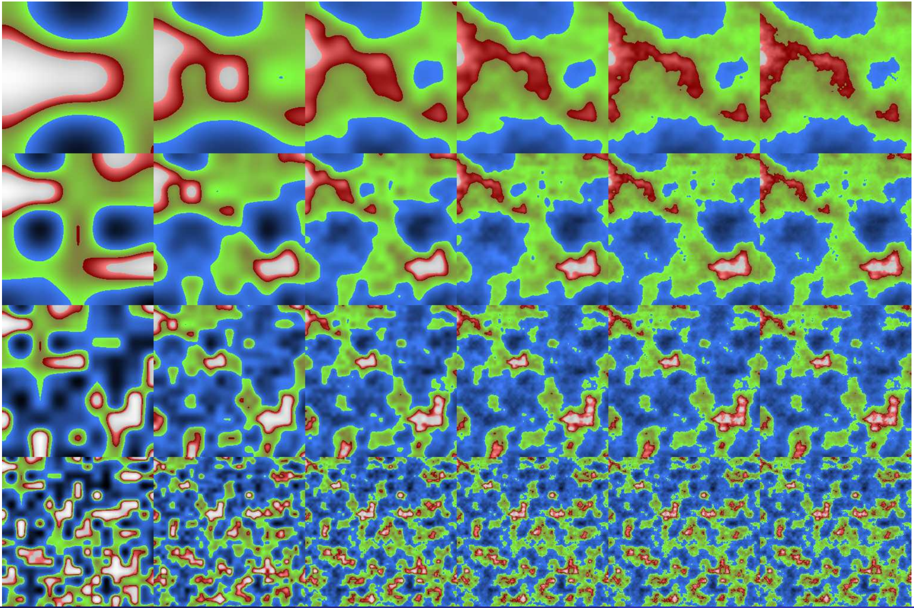
Colorizing noise for different effects



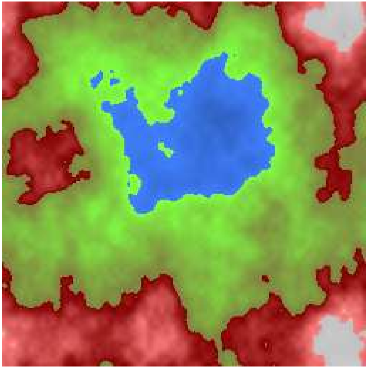
Sky

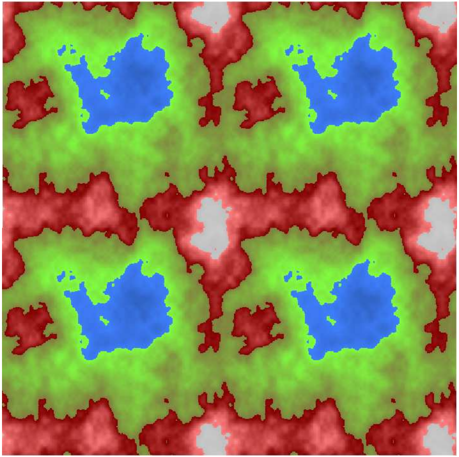
Terrain

Effects of starting wavelength and number of octaves



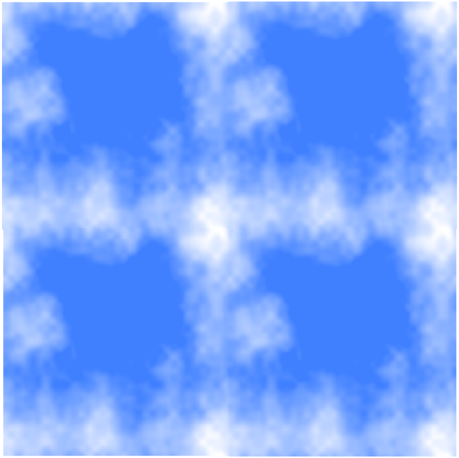
Tilable noise





- How can we do this?

Tilable noise



3D noise: solid textures and animations



Online Resources

Readings

- http://www.noisemachine.com/talk1/
- http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
- http://en.wikipedia.org/wiki/White_noise
- http://en.wikipedia.org/wiki/Pink_noise
- http://mrl.nyu.edu/~perlin/doc/oscar.html
- http://legakis.net/justin/MarbleApplet/
- http://www.planetside.co.uk/

Grad students:

- http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf