



Perlin Noise

CS334 Fall 2023

Daniel G. Aliaga
Department of Computer Science
Purdue University

(slides with help from **Jyun-Ming Chen**, homepage.ttu.edu.tw/jmchen
and
Harriet Fell, <http://www.ccs.neu.edu/home/fell>)



The Oscar™

To Ken Perlin for the development of Perlin Noise, a technique used to produce natural appearing textures on computer generated surfaces for motion picture visual effects.





The Movies

- James Cameron Movies (Abyss, Titanic, ...)
- Animated Movies (Lion King, Moses, ...)
- Arnold Movies (T2, True Lies, ...)
- Star Wars Episode I
- Star Trek Movies
- Batman Movies
- *and lots of others*

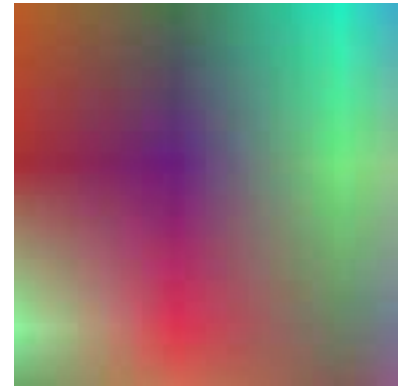
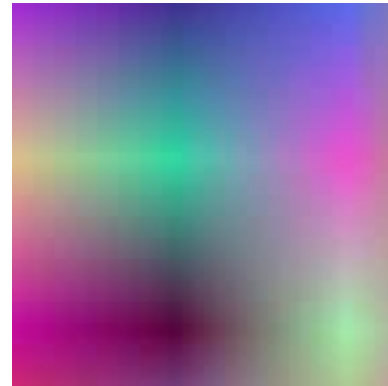
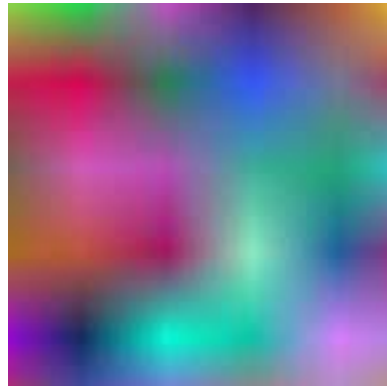
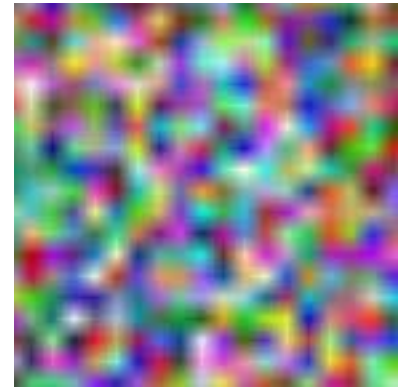
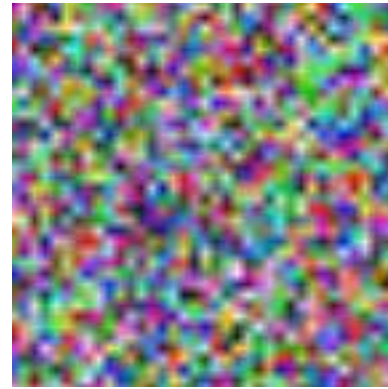
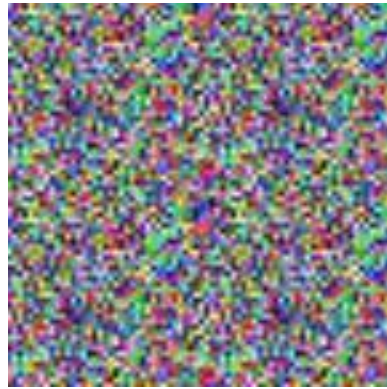
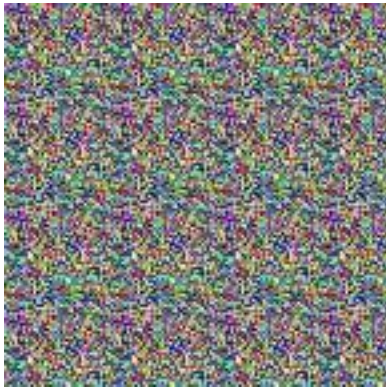
In fact, after around 1990 or so, *every* Hollywood effects film has used it.



What is Noise?

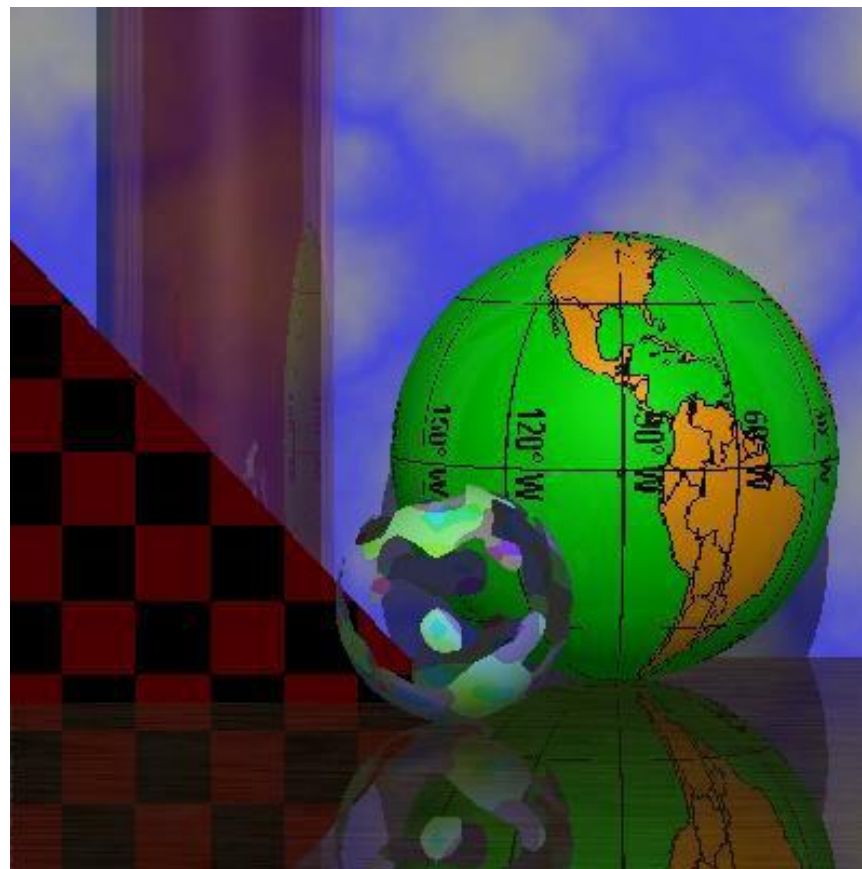
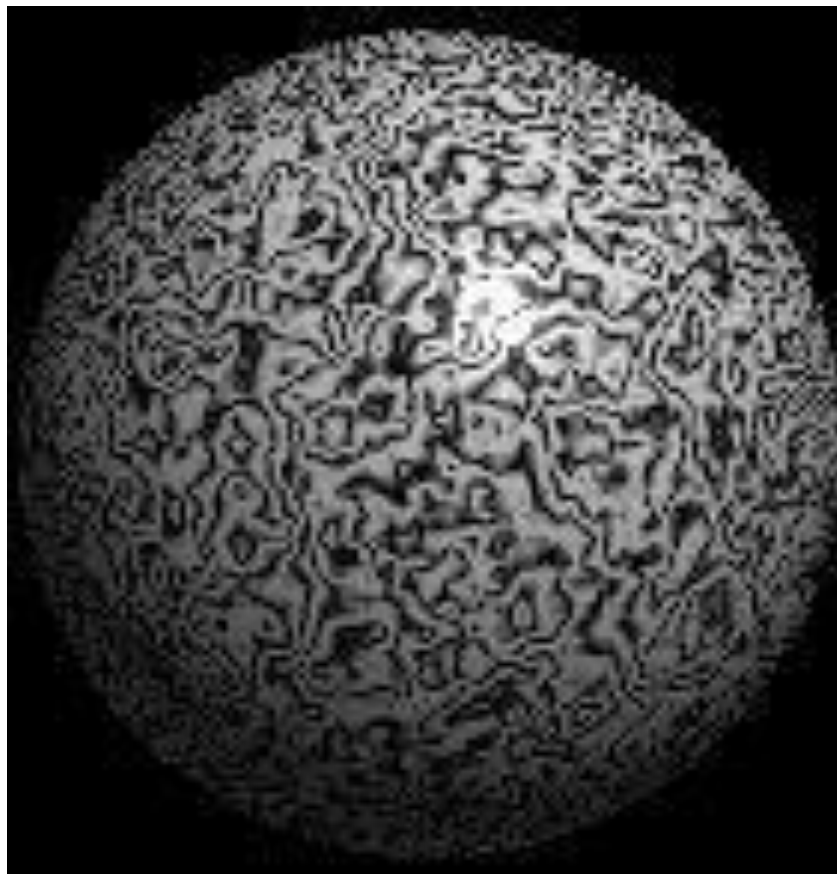
- Noise is a mapping from R^n to R - you input an n-dimensional point with real coordinates, and it returns a real value.
- n=1 for animation
- n=2 cheap texture hacks
- n=3 less-cheap texture hacks
- n=4 time-varying solid textures

Noise is Smooth Randomness



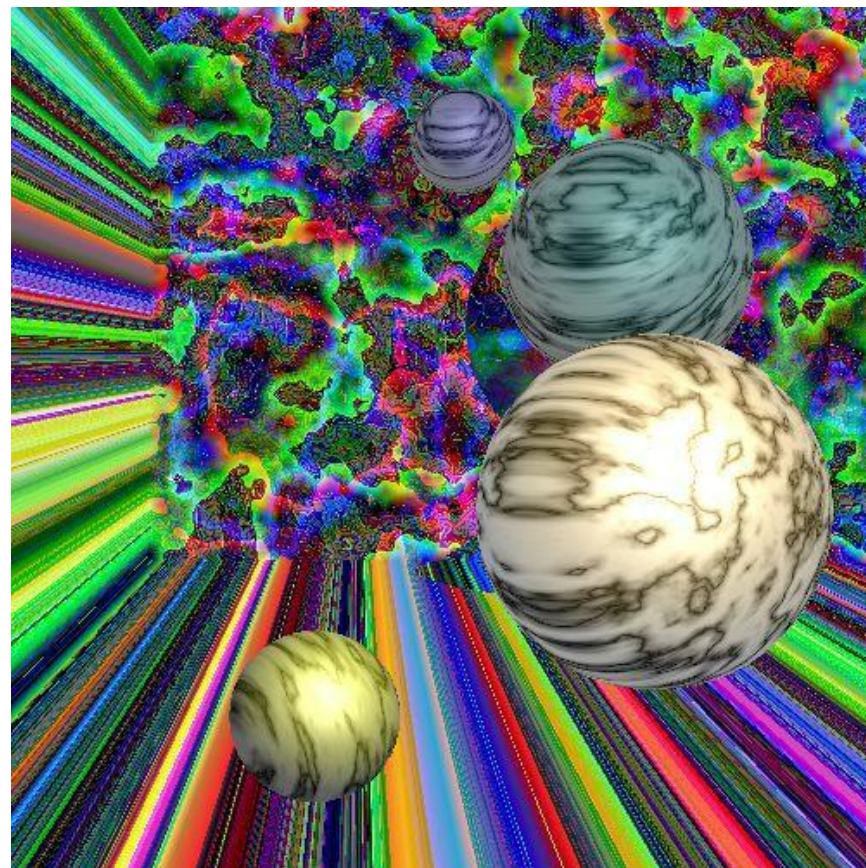
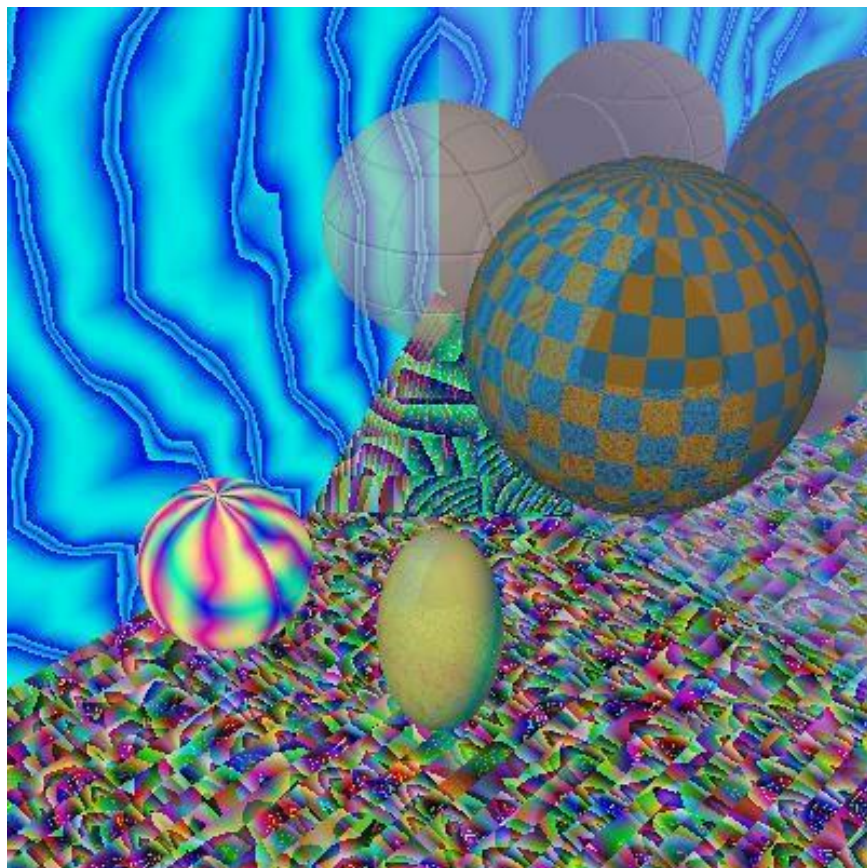


Example Images



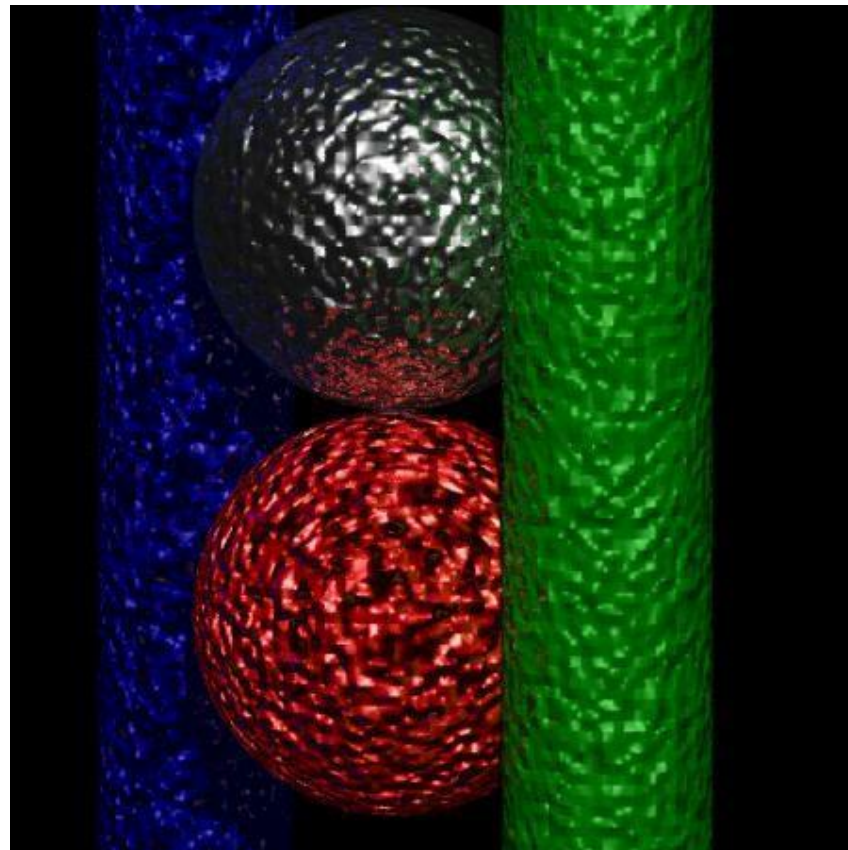


Example Images

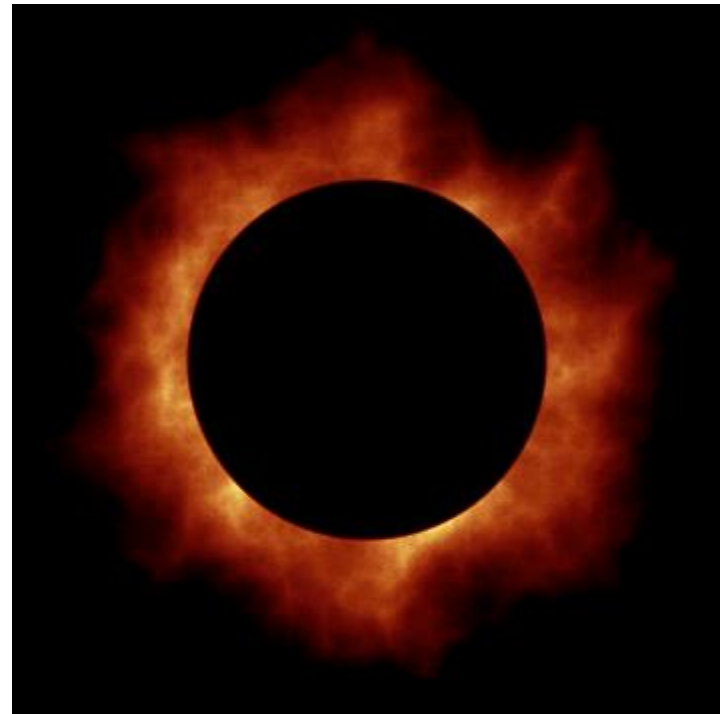
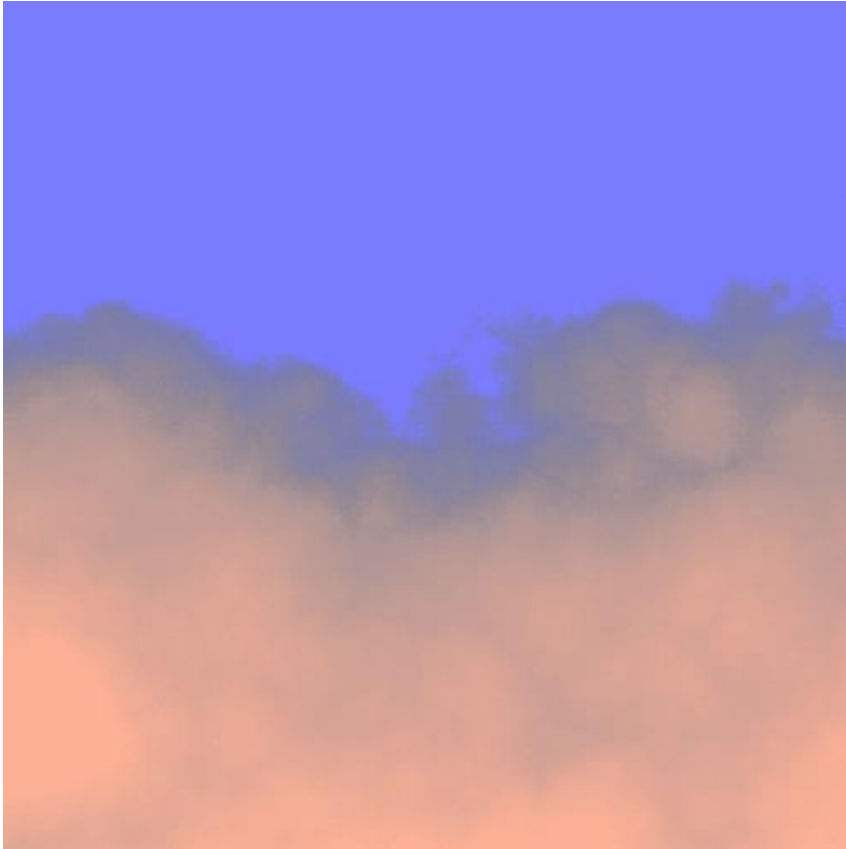




Example Images



Perlin's Clouds and Corona



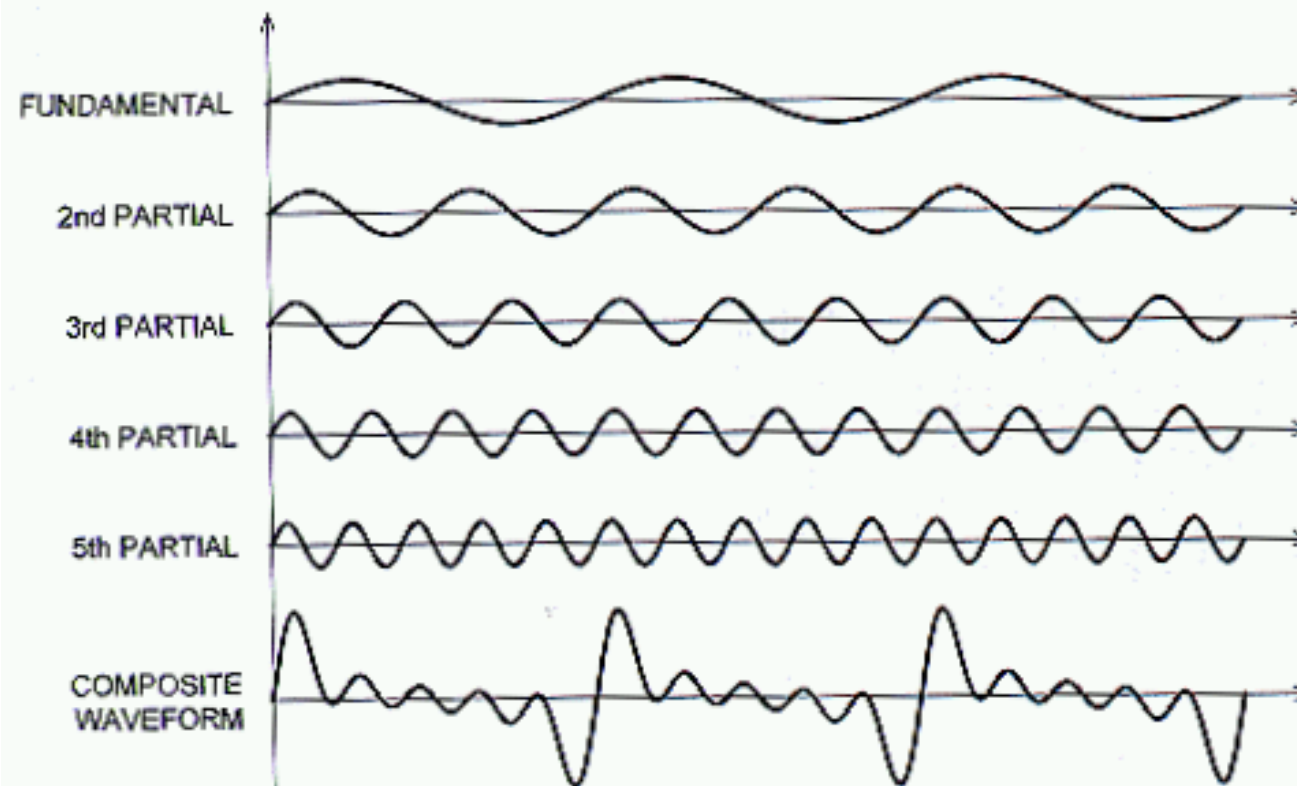


Perlin Noise Function

- Take lots of such smooth functions, with various frequencies and amplitudes
 - Idea similar to fractal, Fourier series, ...
- Add them all together to create a nice noisy function.



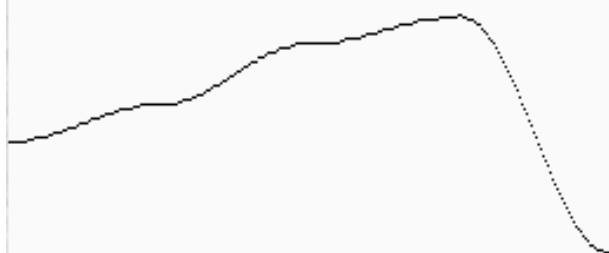
Fourier Analysis



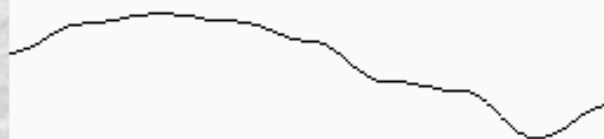


Example

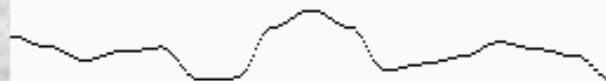
Amplitude : 128
frequency : 4



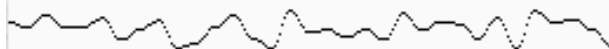
Amplitude : 64
frequency : 8



Amplitude : 32
frequency : 16



Amplitude : 16
frequency : 32



Amplitude : 8
frequency : 64



Amplitude : 4
frequency : 128

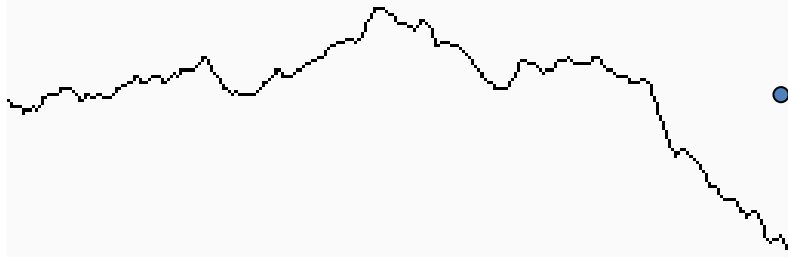




Example (cont)

- Function has large, medium and small variations.

Sum of Noise Functions = (Perlin Noise)

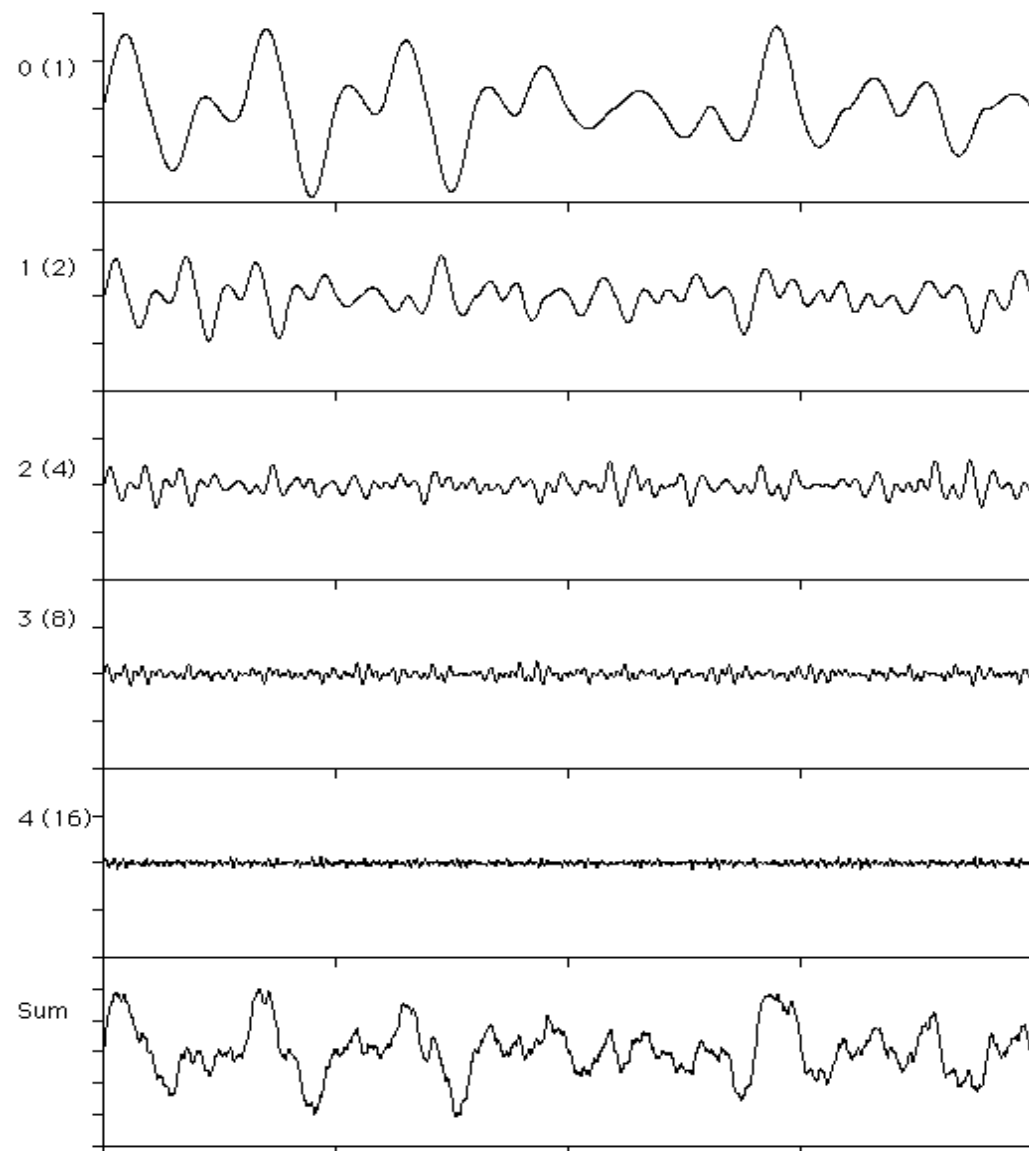


Similar to the ideas of fractal



Alternate formula

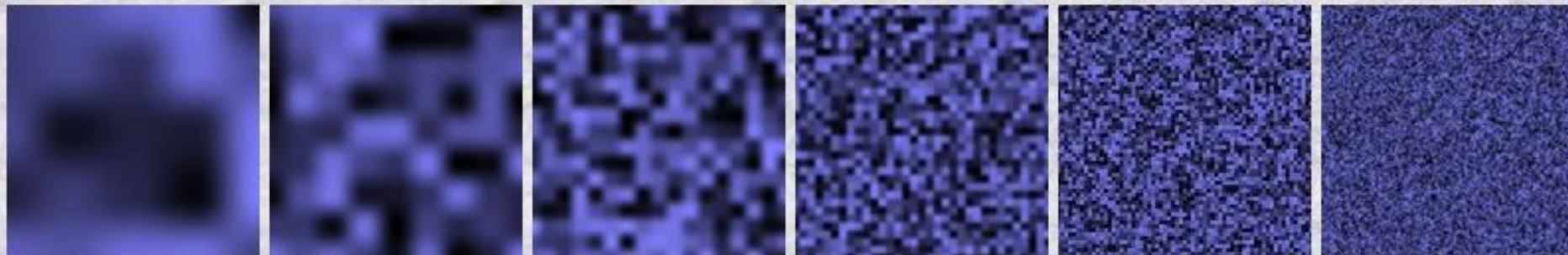
$$\text{NOISE}(\mathbf{x}) = \sum_{i=0}^{N-1} \frac{\text{Noise}(b^i \mathbf{x})}{a^i}$$



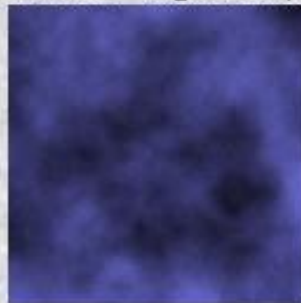


2 dimensions

Some noise functions are created in 2D



Adding all these functions together produces a noisy pattern.





Persistence

- You can create Perlin noise functions with different characteristics by using other frequencies and amplitudes at each step
 - Is a multiplier that determines how quickly the amplitudes diminish for each successive octave in a Perlin-noise function.

```
frequency = 2i  
amplitude = persistencei
```




Frequency

1

2

4

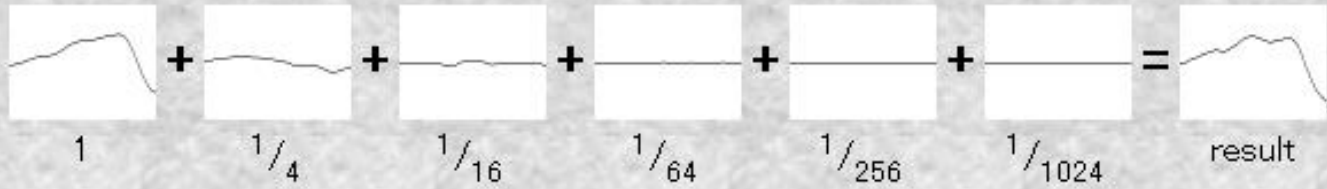
8

16

32

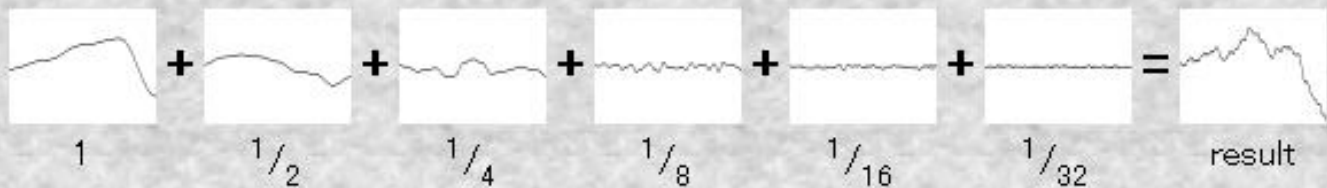
Persistence = 1/4

Amplitude:



Persistence = 1/2

Amplitude:



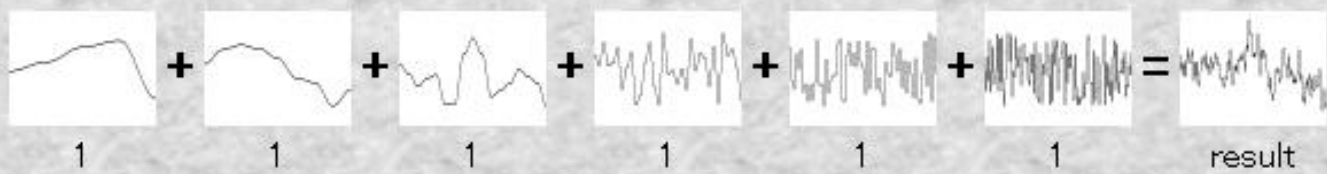
Persistence = 1 / root2

Amplitude:



Persistence = 1

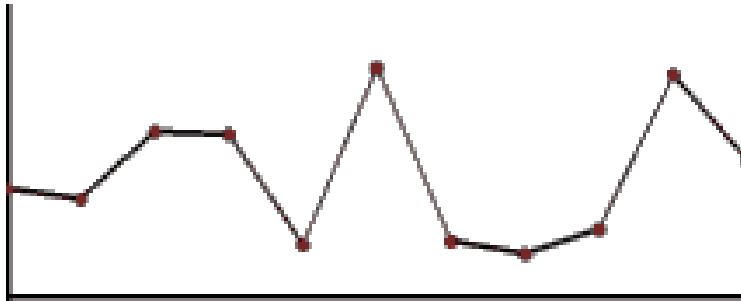
Amplitude:



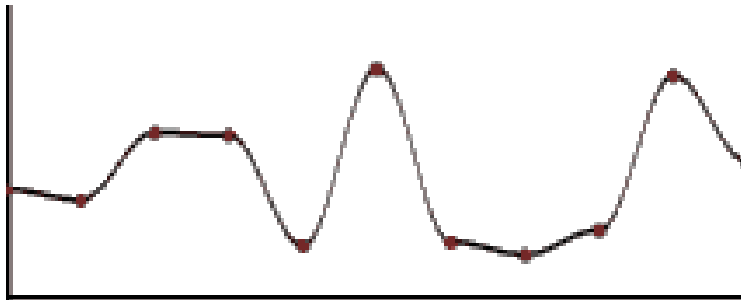


Interpolation

- Linear Interpolation



- Cosine Interpolation



function

Linear_Interpolate(a, b, x)

return $a*(1-x) + b*x$

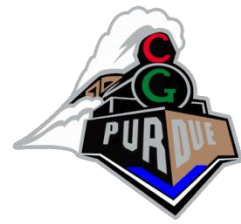
function

Cosine_Interpolate(a, b, x)

$ft = x * 3.1415927$

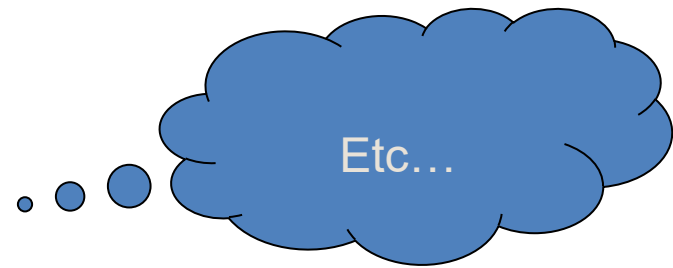
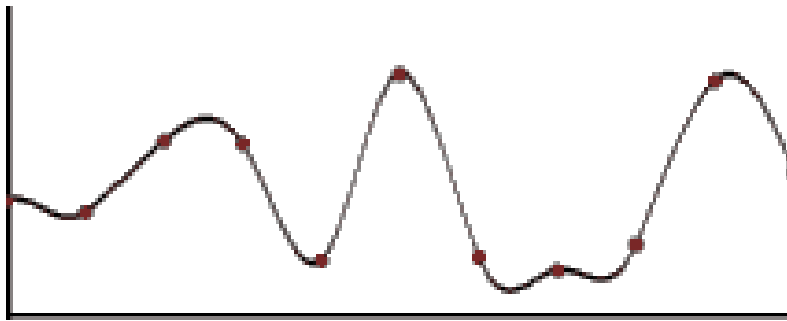
$f = (1 - \cos(ft)) * .5$

return $a*(1-f) + b*f$



Interpolation

- Cubic Interpolation





Example Code

```

/* (copyright Ken Perlin) */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define B 0x100
#define BM 0xff
#define N 0x1000
#define NP 12 /* 2^N */
#define NM 0xffff
static p[B + B + 2];
static float g3[B + B + 2][3];
static float g2[B + B + 2][2];
static float g1[B + B + 2];
static start = 1;
static void init(void);
#define s_curve(t) ( t * t * (3. - 2. * t) )
#define lerp(t, a, b) ( a + t * (b - a) )
#define setup(i,b0,b1,r0,r1)
    t = vec[i] + N; b0 = ((int)t) & BM; b1 = (b0+1) & BM; r0 = t - (int)t; r1 = r0 - 1;
double noise1(double arg)
{
    int bx0, bx1;
    float rx0, rx1, sx, t, u, v, vec[1];
    vec[0] = arg;
    if (start) {
        start = 0;
        init();
    }
    setup(0, bx0, bx1, rx0, rx1);
    sx = s_curve(rx0);
    u = rx0 * g1[ p[ bx0 ] ];
    v = rx1 * g1[ p[ bx1 ] ];
    return lerp(sx, u, v);
}

float noise2(float vec[2])
{
    int bx0, bx1, by0, by1, b00, b10, b01, b11;
    float rx0, rx1, ry0, ry1, *q, sx, sy, a, b, t, u, v;
    register i, j;
    if (start) {
        start = 0;
        init();
    }
    setup(0, bx0, bx1, rx0, rx1);
    setup(1, by0, by1, ry0, ry1);
    i = p[ bx0 ];
    j = p[ bx1 ];
    b00 = p[ i + by0 ];
    b10 = p[ j + by0 ];
    b01 = p[ i + by1 ];
    b11 = p[ j + by1 ];
    sx = s_curve(rx0);
    sy = s_curve(ry0);
}

```

```

#define at2(rx,ry) ( rx * q[0] + ry * q[1] )
    q = g2[ b00 ]; u = at2(rx0,ry0);
    q = g2[ b10 ]; v = at2(rx1,ry0);
    a = lerp(sx, u, v);
    q = g2[ b01 ]; u = at2(rx0,ry1);
    q = g2[ b11 ]; v = at2(rx1,ry1);
    b = lerp(sy, u, v);
    return lerp(sy, a, b);
}

float noise3(float vec[3])
{
    int bx0, bx1, by0, by1, bz0, bz1, b00, b10, b01, b11;
    float rx0, rx1, ry0, ry1, rz0, rz1, *q, sy, sz, a, b, c, d, t, u, v;
    register i, j;
    if (start) {
        start = 0;
        init();
    }
    setup(0, bx0, bx1, rx0, rx1);
    setup(1, by0, by1, ry0, ry1);
    setup(2, bz0, bz1, rz0, rz1);
    i = p[ bx0 ];
    j = p[ bx1 ];
    b00 = p[ i + by0 ];
    b10 = p[ j + by0 ];
    b01 = p[ i + by1 ];
    b11 = p[ j + by1 ];
    t = s_curve(rx0);
    sy = s_curve(ry0);
    sz = s_curve(rz0);
}

#define at3(rx,ry,rz) ( rx * q[0] + ry * q[1] + rz * q[2] )
    q = g3[ b00 + bz0 ]; u = at3(rx0,ry0,rz0);
    q = g3[ b10 + bz0 ]; v = at3(rx1,ry0,rz0);
    a = lerp(t, u, v);
    q = g3[ b01 + bz0 ]; u = at3(rx0,ry1,rz0);
    q = g3[ b11 + bz0 ]; v = at3(rx1,ry1,rz0);
    b = lerp(t, u, v);
    c = lerp(sy, a, b);
    q = g3[ b00 + bz1 ]; u = at3(rx0,ry0,rz1);
    q = g3[ b10 + bz1 ]; v = at3(rx1,ry0,rz1);
    a = lerp(t, u, v);
    q = g3[ b01 + bz1 ]; u = at3(rx0,ry1,rz1);
    q = g3[ b11 + bz1 ]; v = at3(rx1,ry1,rz1);
    b = lerp(t, u, v);
    d = lerp(sy, a, b);
    return lerp(sz, c, d);
}

```

```

static void normalize2(float v[2])
{
    float s;
    s = sqrt(v[0] * v[0] + v[1] * v[1]);
    v[0] = v[0] / s;
    v[1] = v[1] / s;
}

static void normalize3(float v[3])
{
    float s;
    s = sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
    v[0] = v[0] / s;
    v[1] = v[1] / s;
    v[2] = v[2] / s;
}

static void init(void)
{
    int i, j, k;
    for (i = 0; i < B; i++) {
        p[i] = i;
        g1[i] = (float)((random() % (B + B)) - B) / B;
        for (j = 0; j < 2; j++)
            g2[i][j] = (float)((random() % (B + B)) - B) / B;
        normalize2(g2[i]);
        for (j = 0; j < 3; j++)
            g3[i][j] = (float)((random() % (B + B)) - B) / B;
        normalize3(g3[i]);
    }
    while (--i) {
        k = p[i];
        p[i] = p[k] = random() % B;
        p[j] = k;
    }
    for (i = 0; i < B + 2; i++) {
        p[B + i] = p[i];
        g1[B + i] = g1[i];
        for (j = 0; j < 2; j++)
            g2[B + i][j] = g2[i][j];
        for (j = 0; j < 3; j++)
            g3[B + i][j] = g3[i][j];
    }
}

```

<http://mrl.nyu.edu/~perlin/>

