Perlin Noise

CS535 Fall 2016

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 $\mathbb{R}^n$ to $\mathbb{R}$ - 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 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

- Fundamental
- 2nd Partial
- 3rd Partial
- 4th Partial
- 5th Partial
- Composite Waveform
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

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

\[
\text{frequency} = 2^i \\
\text{amplitude} = \text{persistence}^i
\]
Interpolation

- **Linear Interpolation**

\[
\text{Linear\_Interpolate}(a, b, x) = a \cdot (1 - x) + b \cdot x
\]

- **Cosine Interpolation**

\[
\text{Cosine\_Interpolate}(a, b, x) = a \cdot (1 - f) + b \cdot f
\]

where

\[
f = (1 - \cos(\pi x)) \cdot 0.5
\]

\[
ft = x \cdot 3.1415927
\]
Interpolation

- Cubic Interpolation
Smoothed Noise

function Noise(x)
    ..
end function

function SmoothNoise_1D(x)
    return Noise(x)/2 + Noise(x-1)/4 + Noise(x+1)/4
end function

function SmoothNoise_2D(x, y)
    corners = ( Noise(x-1, y-1)+Noise(x+1, y-1)+Noise(x-1, y+1)+Noise(x+1, y+1) ) / 16
    sides = ( Noise(x-1, y) +Noise(x+1, y) +Noise(x, y-1) +Noise(x, y+1) ) / 8
    center = Noise(x, y) / 4
    return corners + sides + center
end function
Example Code

```c
/* (copyright Ken Perlin) */
#include <stdio.h>
#include <math.h>
define 50x100
define BM40
#define N 2000
#define NP 12   /* 2^N */
#define NM 0xfff
static p[BM + BM + 2];
static float g3[BM + BM + 2][3];
static float g2[BM + BM + 2][2];
static float g1[BM + BM + 2];
static start = 1;
define s_curve(t) t * t * (3.0 - 2.0 * 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.;
#define at2(rx,ry) (rx* q[0] + ry* q[1])
q = g2[b00]; u = at2(rx,ry);
q = g2[b10]; v = at2(rx,ry);
a = lerp(t, u, v);
b = lerp(t, u, v);
return lerp(sx, a, b);
#define at3(rx,ry,rz) (rx* q[0] + ry* q[1] + rz* q[2])
q = g3[b00+bz0]; u = at3(rx,ry,rz);
q = g3[b10+bz0]; v = at3(rx,ry,rz);
a = lerp(t, u, v);
b = lerp(t, u, v);
c = lerp(sy, a, b);
q = g3[b01+bz0]; u = at3(rx,ry,rz);
q = g3[b11+bz0]; v = at3(rx,ry,rz);
d = lerp(t, u, v);
return lerp(sz, c, d);

float noise1(float arg) {
int bx0, bx1, by0, by1, bz0, bz1;
float n0, n1, sx, t, u, v, vec[1];
vec[0] = arg;
if (start) {
start = 0;
init();}
setup(0, bx0,bx1, n0, n1, sy);
setup(1, by0,by1, n0, n1, sz);
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 (~0) {
k = p[i];
p[i] = p[j = random() % B];
p[j] = k;
for (i = 0 ; i < B + 2 ; i++) {
for (j = 0 ; j < 2 ; j++)
g1[i] = (float)((random() % (B + B)) - B) / B;
g3[i][0] = (float)((random() % (B + B)) - B) / B;
normalize2(g3[i]);
for (i = 0 ; i < B + 2 ; i++)
return lerp(sx, a, b);
}
```
http://mrl.nyu.edu/~perlin/