Simplex Noise

Simplex noise is an improved noise algorithm presented by Ken Perlin in 2001. This algorithm works by dividing a plane into *simplices*, the most compact possible shape for that plane, and assigns a pseudo-random value to each corner. For a two-dimensional plane the shape is a triangle. For three-dimensional planes, the shape is a tetrahedron. It then sums the values of the corners, where the value for each corner is reduced the further it is from the point.

Contents

Simplex Noise Algorithm

Algorithm

```
public class SimplexNoise {

    private static int grad3[][] = {{1,1,0},
{-1,1,0},{1,-1,0},{-1,-1,0},
  {1,0,1},{-1,0,1},{1,0,-1},{-1,0,-1},
  {0,1,1},{0,-1,1},{0,1,-1},{0,-1,-1}};

     private static int p[] =
{151,160,137,91,90,15,

131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,1
 190,
6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,
 88,237,149,56,87,174,20,125,136,171,168,

133,230,220,105

,169,200,196,
98,173,186,
```

```
 223,183,170,213,119,248,152, 2,44,154,163,
70,221,153,101,155,167, 43,172,9,
 129,22,39,253,
19,98,108,110,79,113,224,232,178,185,
112,104,218,246,97,228,
 251,34,242,193,238,210,144,12,191,179,162,241,
81,51,145,235,249,14,239,107,
 49,192,214, 31,181,199,106,157,184,
84,204,176,115,121,50,45,127, 4,150,254,

138,236,205,93,222,114,67,29,24,72,243,141,128,195,7
    private static int perm[] = new int[512];

    static { for(int i=0; i<512; i++) perm[i]=p[i
& 255]; }

    private static int fastfloor(double x) {
        return x>0 ? (int)x : (int)x-1;
    }

    private static double dot(int g[], double x,
double y) {
        return g[0]*x + g[1]*y;
    }

    public static double noise(double xin, double
yin) {
        double n0, n1, n2; // Noise contributions
from the three corners

        // Skew the input space to determine which
simplex cell we're in
        final double F2 = 0.5*
(Math.sqrt(3.0)-1.0);
        double s = (xin+yin)*F2; // Hairy factor
for 2D
        int i = fastfloor(xin+s);
        int j = fastfloor(yin+s);
        final double G2 = (3.0-
Math.sqrt(3.0))/6.0;
        double t = (i+j)*G2;
        double X0 = i-t; // Unskew the cell origin
back to (x,y) space
        double Y0 = j-t;
        double x0 = xin-X0; // The x,y distances
from the cell origin
        double y0 = yin-Y0;

        // For the 2D case, the simplex shape is
an equilateral triangle.
        // Determine which simplex we are in.
        int i1, j1; // Offsets for second (middle)
corner of simplex in (i,j) coords
        if(x0>y0) {i1=1; j1=0;} // lower triangle,
XY order: (0,0)->(1,0)->(1,1)
        else {i1=0; j1=1;} // upper triangle, YX
order: (0,0)->(0,1)->(1,1)

        // A step of (1,0) in (i,j) means a step
of (1-c,-c) in (x,y), and
        // a step of (0,1) in (i,j) means a step
of (-c,1-c) in (x,y), where
        // c = (3-sqrt(3))/6
        double x1 = x0 - i1 + G2; // Offsets for
middle corner in (x,y) unskewed coords
        double y1 = y0 - j1 + G2;
        double x2 = x0 - 1.0 + 2.0 * G2; //
Offsets for last corner in (x,y) unskewed coords
        double y2 = y0 - 1.0 + 2.0 * G2;

        // Work out the hashed gradient indices of
the three simplex corners
        int ii = i & 255;
        int jj = j & 255;
        int gi0 = perm[ii+perm[jj]] % 12;
        int gi1 = perm[ii+i1+perm[jj+j1]] % 12;
        int gi2 = perm[ii+1+perm[jj+1]] % 12;

        // Calculate the contribution from the
three corners
        double t0 = 0.5 - x0*x0-y0*y0;
        if(t0<0) n0 = 0.0;
        else {
            t0 *= t0;
            n0 = t0 * t0 * dot(grad3[gi0], x0,
y0); // (x,y) of grad3 used for 2D gradient
        }

        double t1 = 0.5 - x1*x1-y1*y1;
        if(t1<0) n1 = 0.0;
        else {
            t1 *= t1;
            n1 = t1 * t1 * dot(grad3[gi1], x1,
y1);
        }

        double t2 = 0.5 - x2*x2-y2*y2;
        if(t2<0) n2 = 0.0;
        else {
            t2 *= t2;
            n2 = t2 * t2 * dot(grad3[gi2], x2,
y2);
        }
h corner to
turn values
```

Algorithm Explained

Simplex noise starts similarly to Perlin noise. It populates an array of 512 elements with 256 numbers ordered in a psuedo-random fashion. This is the heart of what gives Simplex its random appearance. Instead of determining which cube unit the point is in, the algorithm first looks at what *simplex* the point is in.

For this two-dimensional algorithm, a simplex is an equilateral triangle. Thus the point lies in a grid of triangles instead of cubes, so some math must be done to find the coordinate of the triangle itself. Then the algorithm finds the relative location of the point inside its parent simplex.

Here is where Simplex noise most strongly diverges from Perlin noise, and where much of its benefits come from. For each of the three corners, find its value. Then reduce the value according to the distance of the corner from the point. Finally, sum the three values and scale them so the returned value is between -1 and 1.

Advantages over Perlin Noise

- Far fewer calculations needed.
- Operates faster.
- As the algorithm moves from two dimension to three, to four, etc., the amount of calculations doesn't increase at as much a rate as Perlin noise does.

Code Examples

The demo program is available in the simplex (https://github.com/shawnco/procedural-content-generation-wiki/tree/master/simplex) folder.

Simplex Biomes

This example is nearly identical to the Perlin biomes example. The Simplex example uses noise in a slightly different way, resulting in a different type of output. Use WASD to scroll throughout the map. Yellow indicates desert, green indicates grasslands, and white indicates snow.

Citations

- Gustavon, Stefan. "Simplex noise demystified." *Linkoping University*. Linkoping University. n.d. 24 Nov., 2016. http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf

Categories