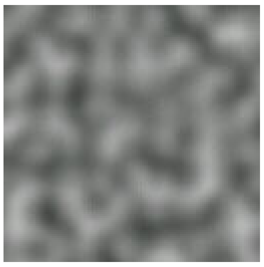
WIKIPEDIA

# Perlin noise

**Perlin noise** is a type of gradient noise developed by Ken Perlin in 1983 as a result of his frustration with the "machine-like" look of computer graphics at the time.[1] He formally described his findings in a SIGGRAPH paper in 1985 called *An image Synthesizer*.[2] In 1997, Perlin was awarded an Academy Award for Technical Achievement for creating the algorithm.[3]


Two-dimensional slice through 3D Perlin noise at z=0

> 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 development of Perlin Noise has allowed computer graphics artists to better represent the complexity of natural phenomena in visual effects for the motion picture industry.

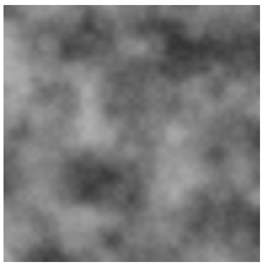
Perlin did not apply for any patents on the algorithm, but in 2001 he was granted a patent for the use of 3D+ implementations of simplex noise for texture synthesis. Simplex noise has the same purpose, but uses a simpler space-filling grid. Simplex noise alleviates some of the problems with Perlin's "classic noise", among them computational complexity and visually-significant directional artifacts.[4]

## Contents

## Uses

Perlin noise is a procedural texture primitive, a type of gradient noise used by visual effects artists to increase the appearance of realism in computer graphics. The function has a pseudo-random appearance, yet all of its visual details are the same size. This property allows it to be readily controllable; multiple scaled copies of Perlin noise can be inserted into mathematical expressions to create a great variety of procedural textures. Synthetic textures using Perlin noise are often used in CGI to make computer-generated visual elements – such as object surfaces, fire, smoke, or clouds – appear more natural, by imitating the controlled random appearance of textures in nature.

It is also frequently used to generate textures when memory is extremely limited, such as in demos, and is increasingly finding use in graphics processing units for real-time graphics in computer games.


Perlin noise rescaled and added into itself to create fractal noise.

## Development

Perlin noise resulted from the work of Ken Perlin, who developed it at Mathematical Applications Group, Inc. (MAGI) for Disney's computer animated sci-fi motion picture *Tron* (1982). In 1997, he won an Academy Award for Technical Achievement from the Academy of Motion Picture Arts and Sciences for this contribution to CGI.[5]

## Algorithm detail

Perlin noise is most commonly implemented as a two-, three- or four-dimensional function, but can be defined for any number of dimensions. An implementation typically involves three steps: grid definition with random gradient vectors, computation of the dot product between the distance-gradient vectors and interpolation between these values.

### Grid definition

Define an *n*-dimensional grid where each point has a random *n*-dimensional unit-length gradient vector, except in one dimension where the gradients are random scalars between -1 and 1.

Assigning the random gradients in one and two dimensions is trivial using a random number generator. For higher dimensions a Monte Carlo approach can be used[1] where random Cartesian coordinates are chosen in a unit cube, points falling outside the unit ball are discarded, and the remaining points are normalized to lie on the unit sphere. The process is continued until the required number of random gradients are obtained.

In order to negate the expensive process of computing new gradients for each grid node, some implementations use a hash and lookup table for a finite number of precomputed gradient vectors.[6] The use of a hash also permits the inclusion of a random seed where multiple instances of Perlin noise are required.

### Dot product

Given an *n*-dimensional argument for the noise function, the next step in the algorithm is to determine into which grid cell the given point falls. For each corner node of that cell, the distance vector between the point and the node is determined. The dot product between the gradient vector at the node and the distance vector is then computed.

For a point in a two-dimensional grid, this will require the computation of 4 distance vectors and dot products, while in three dimensions 8 distance vectors and 8 dot products are needed. This leads to the $\mathcal{O}(2^n)$ complexity scaling.

### Interpolation

The final step is interpolation between the $2^n$ dot products computed at the nodes of the cell containing the argument point. This has the consequence that the noise function returns 0 when evaluated at the grid nodes themselves.

Interpolation is performed using a function that has zero first derivative (and possibly also second derivative) at the $2^n$ grid nodes. This has the effect that the gradient of the resulting noise function at each grid node coincides with the precomputed random gradient vector there. If $n = 1$, an example of a function that interpolates between value $a_0$ at grid node 0 and value $a_1$ at grid node 1 is

$$f(x) = a_0 + \operatorname{smoothstep}(x) \cdot (a_1 - a_0) \quad \text{for } 0 \le x \le 1$$

where the smoothstep function was used.

Noise functions for use in computer graphics typically produce values in the range [-1.0,1.0]. In order to produce Perlin noise in this range, the interpolated value may need to be scaled by some scaling factor.[6]

## Implementation

The following is a two-dimensional implementation of Classical Perlin Noise, written in C++.

```
// Function to linearly interpolate between a0 and a1
// Weight w should be in the range [0.0, 1.0]
float lerp(float a0, float a1, float w) {
    return (1.0 - w)*a0 + w*a1;
}

// Computes the dot product of the distance and gradient vectors.
float dotGridGradient(int ix, int iy, float x, float y) {

    // Precomputed (or otherwise) gradient vectors at each grid node
    extern float Gradient[IYMAX][IXMAX][2];

    // Compute the distance vector
    float dx = x - (float)ix;
    float dy = y - (float)iy;

    // Compute the dot-product
    return (dx*Gradient[iy][ix][0] + dy*Gradient[iy][ix][1]);
}

// Compute Perlin noise at coordinates x, y
float perlin(float x, float y) {

    // Determine grid cell coordinates
    int x0 = int(x);
    int x1 = x0 + 1;
    int y0 = int(y);
    int y1 = y0 + 1;

    // Determine interpolation weights
    // Could also use higher order polynomial/s-curve here
    float sx = x - (float)x0;
    float sy = y - (float)y0;

    // Interpolate between grid point gradients
    float n0, n1, ix0, ix1, value;
    n0 = dotGridGradient(x0, y0, x, y);
    n1 = dotGridGradient(x1, y0, x, y);
    ix0 = lerp(n0, n1, sx);
    n0 = dotGridGradient(x0, y1, x, y);
    n1 = dotGridGradient(x1, y1, x, y);
    ix1 = lerp(n0, n1, sx);
    value = lerp(ix0, ix1, sy);

    return value;
}
```

## Complexity

For each evaluation of the noise function, the dot product of the position and gradient vectors must be evaluated at each node of the containing grid cell. Perlin noise therefore scales with complexity $O(2^n)$ for $n$ dimensions. Alternatives to Perlin noise producing similar results with improved complexity scaling include simplex noise and OpenSimplex noise.

## See also

- Value noise
- Simulation noise

## References

1. Perlin, Ken. "Making Noise" (https://web.archive.org/web/20160303232627/http://www.noisemachine.com/talk1/). *noisemachine.com*. Ken Perlin. Archived from the original (http://www.noisemachine.com/talk1/) on March 3, 2016.

2. Perlin, Ken (July 1985). "An Image Synthesizer" (http://doi.acm.org/10.1145/325165.325247). *SIGGRAPH Comput. Graph.* **19** (0097-8930): 287–296. doi:10.1145/325165.325247 (https://doi.org/10.1145/325165.325247). Retrieved 9 February 2016.
3. Original source code (http://mrl.nyu.edu/~perlin/doc/oscar.html#noise) of Ken Perlin's 'coherent noise function'
4. US patent 6867776 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US6867776), Kenneth Perlin, "Standard for perlin noise", issued 2005-03-15, assigned to Kenneth Perlin and Wsou Investments LLC
5. Kerman, Phillip. *Macromedia Flash 8 @work: Projects and Techniques to Get the Job Done*. Sams Publishing. 2006. ISBN 9780672328282.
6. libnoise (http://libnoise.sourceforge.net/index.html)

## External links

- Matt Zucker's Perlin noise math FAQ
- Rob Farber's tutorial demonstrating Perlin noise generation and visualization on CUDA-enabled graphics processors
- Jason Bevins's extensive C++ library for generating complex, coherent noise values
- Realization on PHP (github)
- Perlin Noise Explained in Depth (with C++ source code)
- The Book of Shaders by Patricio Gonzalez Vivo & Jen Lowe

Retrieved from "https://en.wikipedia.org/w/index.php?title=Perlin_noise&oldid=841806985"