Improving Noise

Ken Perlin
Media Research Laboratory, Dept. of Computer Science, New York University
perlin@cat.nyu.edu

ABSTRACT

Two deficiencies in the original Noise algorithm are corrected: second order interpolation discontinuity and unoptimal gradient computation. With these defects corrected, Noise both looks better and runs faster. The latter change also makes it easier to define a uniform mathematical reference standard.

Keywords

procedural texture

1 INTRODUCTION

Since its introduction 17 years ago [Perlin 1984; Perlin 1985; Perlin and Hoffert 1989], Noise has found wide use in graphics [Foley et al. 1996; Upstill 1990]. The original algorithm, although efficient, suffered from two defects: second order discontinuity across coordinate-aligned integer boundaries, and a needlessly expensive and somewhat problematic method of computing the gradient. We (belatedly) remove these defects.

2 DEFICIENCIES IN ORIGINAL ALGORITHM

As detailed in [Ebert et al 1998], Noise is determined at point **(x,y,z)** by computing a pseudo-random gradient at each of the eight nearest vertices on the integer cubic lattice and then doing splined interpolation. Let **(i,j,k)** denote the eight points on this cube, where **i** is the set of lower and upper bounding integers on **x**: **{⌊x⌋,⌊x⌋+1}**, and similarly **j** = **{ ⌊y⌋,⌊y⌋+1}** and **k** = **{ ⌊z⌋,⌊z⌋+1}**. The eight gradients are given by $g_{i,j,k}$ = ***G*[P[P[P[i]+j]+k]]** where precomputed arrays **P** and ***G*** contain, respectively, a pseudo-random permutation, and pseudo-random unit-length gradient vectors. The successive application of **P** hashes each lattice point to de-correlate the indices into ***G***. The eight linear functions $g_{i,j,k}$ **· (x-i,y-j,z-k)** are then trilinearly interpolated by **s(x-⌊x⌋)**, **s(y-⌊y⌋)** and **s(z-⌊z⌋)**, where $\mathbf{s(t) = 3t^2-2t^3}$.

The above algorithm is very efficient but contains some deficiencies. One is in the cubic interpolant function's second derivative **6-12t**, which is not zero at either **t=0** or **t=1**. This non-zero value creates second order discontinuities across the coordinate-aligned faces of adjoining cubic cells. These discontinuities become noticeable when a Noise-displaced surface is shaded; then the surface normal (which is itself a derivative operator) has a visibly discontinuous derivative (Figure 1a).

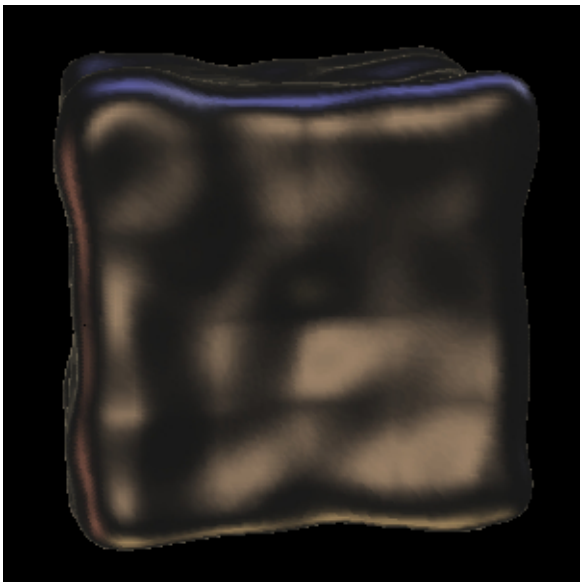

Figure 1a: Noise-displaced superquadric with old interpolants

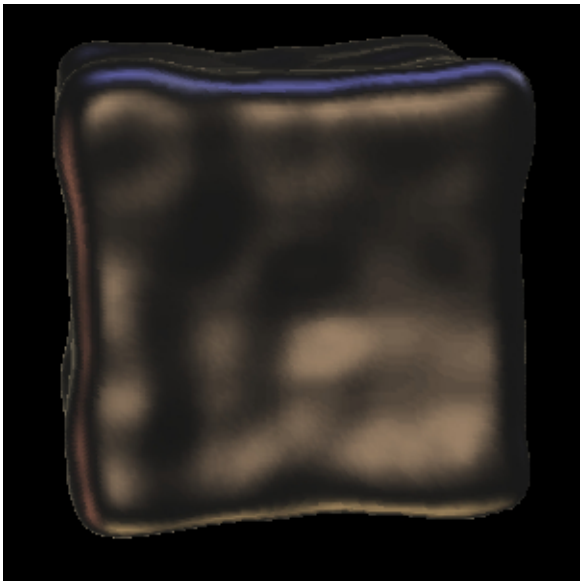

Figure 1b: Noise-displaced superquadric with new interpolants

The second deficiency is that whereas the gradients in ***G*** are distributed uniformly over a sphere, the cubic grid itself has directional biases, being shortened along the axes and elongated on the diagonals between opposite cube vertices. This directional asymmetry tends to cause a sporadic clumping effect, where nearby gradients that are almost axis-aligned, and therefore close together, happen to align with each other, causing anomalously high values in those regions (Figure 2a).

Copyright © 2002 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1-212-869-0481 or e-mail permissions@acm.org.
© 2002 ACM 1-58113-521-1/02/0007 $5.00

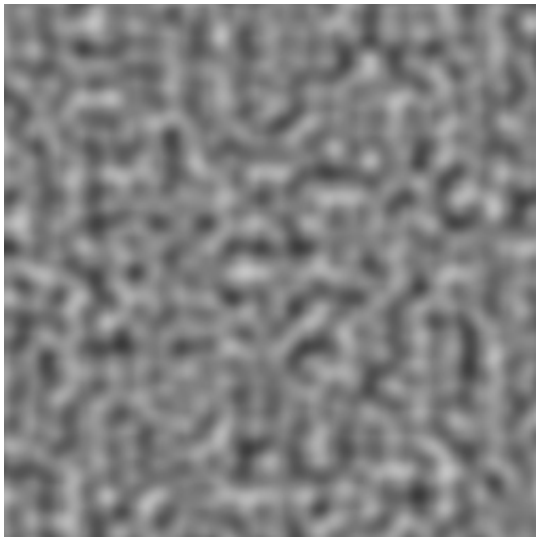

Figure 2a: High-frequency Noise, with old gradient distributions

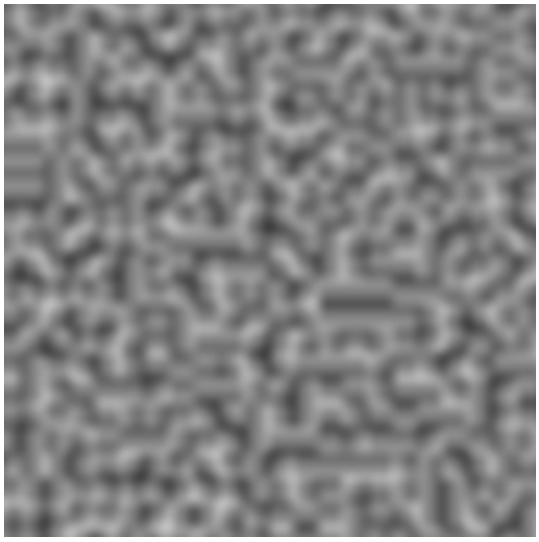

Figure 2b: High-frequency Noise, with new gradient distributions

3 MODIFICATIONS

The above deficiencies are addressed as follows. $3t^2-2t^3$ is replaced by $6t^5-15t^4+10t^3$, which has zero first and second derivatives at both **t=0** and **t=1**. The absence of artifacts can be seen in Figure 1b.

The key to removing directional bias in the gradients is to skew the set of gradient directions away from the coordinate axes and long diagonals. In fact, it is not necessary for ***G*** to be random at all, since **P** provides plenty of randomness. The corrected version replaces ***G*** with the 12 vectors defined by the directions from the center of a cube to its edges:

(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)

Gradients from this set are chosen by using the result of **P**, modulo 12. This set of gradient directions was chosen for two reasons: *(i)* it avoids the main axis and long diagonal directions, thereby avoiding the possibility of axis-aligned clumping, and *(ii)* it allows the eight inner products to be effected without requiring any multiplies, thereby removing 24 multiplies from the computation.

To avoid the cost of dividing by 12, we pad to 16 gradient directions, adding an extra **(1,1,0),(-1,1,0),(0,-1,1)** and **(0,-1,-1)**. These form a regular tetrahedron, so adding them redundantly introduces no visual bias in the texture. The final result has the same non-directional appearance as the original distribution but less clumping, as can be seen in Figure 2b.

4 PERFORMANCE

In a timing comparison (C implementations on the Intel optimizing compiler running on a Pentium 3), the new algorithm runs approximately ten percent faster than the original. The cost of the extra multiplies required to compute the three corrected interpolants is apparently outweighed by the savings from the multiplies no longer required to compute the eight inner products. Examination of the assembly code indicates that the Intel processor optimizes by pipelining the successive multiplies of the three interpolant calculations since no memory fetches are required within this block of computations.

Rather than use a 12-entry table to avoid inner product multiples, the ***G*** table can also be expanded and used to replace the last lookup into **P**. Whether this method is more efficient is processor dependent. For example, 3D inner products are single operations on both **nVidia** and **ATI** pixel processors.

5 CONCLUSIONS

The described changes result in an implementation of Noise which is both visually improved and computationally more efficient. Also, with the pseudo-random gradient table removed, the only pseudo-random component left is the ordering of the permutation table **P**. Once a standard permutation order is determined, it will at last be possible to give a uniform mathematical definition for the Noise function, identical across all software and hardware environments.

ACKNOWLEDGEMENTS

Thanks to the reviewers for constructive criticisms that improved the paper, to Denis Zorin for his invaluable suggestions, and to Nathan Wardrip-Fruin and Chris Poultney, who helped greatly in the rush of production.

References

EBERT, D. ET AL. 1998. *Texturing and Modeling; A Procedural Approach,* Second Edition. AP Professional, Cambridge.

FOLEY, J. ET AL. 1996. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading.

PERLIN, K., ACM SIGGRAPH 84 conference, course in "Advanced Image Synthesis."

PERLIN, K. 1985. An Image Synthesizer. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85), 24*. 3.

PERLIN, K. AND HOFFERT, E. 1989. Hypertexture. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89), 23*, 3.

UPSTILL, S. 1990. *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley.