This paper has been published in : *Graphics Gems, vol 5, p367-374, Eds Alan W. Paeth, Academic Press, 1995*

Wave Generators for Computer Graphics

Christophe Schlick

LaBRI,[1] 351 cours de la libération, 33405 Talence (France)
schlick@labri.u-bordeaux.fr

Introduction

The work of Perlin [1] [3] shows that combining simple generic operators (e.g. noise, turbulence, bias, gain) allows the creation of complex specific visual effects such as marble, fire or water. Though first presented as a procedural approach to texturing, the ideas have been generalized to many computer graphics applications [4], including modeling, deformation, rendering and animation. This gem provides several wave generators which represent some basic tools for creating many regular or random visual effects in computer imagery.

Noise

The most ubiquitous operator is the `noise` function. It generates a distribution having a user-controlled randomness while maintaining valuable statistical properties of invariance under translation or rotation and limited spectrum bandpass. Several implementations of the `noise` function have been proposed since its original development; a remarkably elegant recursive implementation appears in a previous gem [6]. Because the original `noise` routine was intended for solid texturing, it is often implemented as a 3D function which maps a point on the Euclidian space $\mathbb{R}^3$ onto a scalar value on the unipolar or bipolar interval[2] $[0,1]$ or $[-1,1]$.

Despite all its useful properties, the `noise` function also includes a few weaknesses. In many situations (for instance, in rendering or animation) there is a need only for 1D randomness; in such cases, using a 3D noise generator involves wasteful calculations. Another drawback of `noise` (at least with the usual implementations) is that it provides random signals having always C^1 continuity. Therefore, many natural phenomena which involve C^0 and C^{-1} randomness [2] cannot be simulated by `noise` without specific post-treatment of the function.

Description

This gem corrects the previous weaknesses by introducing *wave generators* as an addendum to the `noise` function. Three monodimensional operators are presented here: `Rwave` (rectangular wave), `Twave` (triangular wave), `Swave` (sine wave). Each maps $\mathbb{R}$ onto $[0,1]$ providing C^{-1}, C^0 and C^1 continuity respectively. Each function `*wave` takes four parameters :

t	:	wave parameter	$(t \in \mathbb{R})$
s	:	shape factor	$(-1 \le s \le 1)$
f	:	frequency variance	$(0 \le f \le 1)$
a	:	amplitude variance	$(0 \le a \le 1)$

The default waves ($s = f = a = 0$) are shown in Section (a) of Figures 1-3. Following the idea of integer lattice [1], the routines are implemented so as the extrema of the function are located at integer values for t. In other words, the period of the function is 2 :

$$\forall p \in \mathbb{N} \quad \texttt{Xwave}\,(2p,0,0,0) = 1 \quad \text{and} \quad \texttt{Xwave}\,(2p+1,0,0,0) = 0$$

[1] Laboratoire Bordelais de Recherche en Informatique (*Université Bordeaux I* and *Centre National de la Recherche Scientifique*). The present work is also granted by the *Conseil Régional d'Aquitaine*.
[2] The first form is recommended as transformation to the second is straightforward: "`t += t-1;`".

The shape factor s allows a stretch either towards low values ($s < 0$) or towards high values ($s > 0$). The resulting waves with three different values for the shape factor are shown respectively in Sections (b), (c) and (d) of Figures 1-3. The stretching effect is obtained by using the rational bias function described elsewhere [5].

The frequency variance f and the amplitude variance a are used to introduce some noise to the waves, either in the frequency domain ($a = 0, f \neq 0$), in the amplitude domain ($a \neq 0, f = 0$), or in both ($a \neq 0, f \neq 0$). These *AM*, *FM* and hybred *AM+FM* waves appear respectively in Sections (b), (c) and (d) of Figures 4-6.

The amount of noise that may be introduced is strictly bounded to preserve the properties defined by Perlin. For instance, even with maximum frequency domain variance ($f = 1$), the pseudo-period of the wave stays within the range $[1, 4]$. Similarly, maximum amplitude domain variance ($a = 1$), keeps the wave's local maximum (minimum) within $[\frac{1}{2}, 1]$ ($[0, \frac{1}{2}]$). Therefore parameters f and a provide a double continuum between four kinds of waves :

$$
\begin{array}{lcll}
f = 0 & \text{and} & a = 0 & : \text{constant-frequency/constant-amplitude} \\
f = 0 & \text{and} & a = 1 & : \text{constant-frequency/varying-amplitude} \\
f = 1 & \text{and} & a = 0 & : \text{varying-frequency/constant-amplitude} \\
f = 1 & \text{and} & a = 1 & : \text{varying-frequency/varying-amplitude}
\end{array}
$$

Extensions

In the above description, no precision has been given about the relationship between the wave parameter t and the point (x, y, z) for which the wave function is computed. At least three different schemes may be used, giving three different visual effects:

- $t = ax + by + cz$ which provides *planar waves* (every point at a given distance from the plane $ax + by + cz = 0$ has the same wave value).
- $t = \sqrt{(ay - bx)^2 + (bz - cy)^2 + (cx - az)^2}$ which provides *cylindrical waves* (every point at a given distance from the axis directed by (a, b, c) has the same wave value).
- $t = \sqrt{x^2 + y^2 + z^2}$ which provides *spherical waves* (every point at a given distance from the origin has the same wave value).

Although using only monodimensional waves is sufficient in many cases, wave generators in higher dimensions (with similar shape and variance factors) would be a valuable extension. A naïve solution could be to create such waves by multiplying or averaging several orthogonal monodimensional waves. Unfortunately such a process alters the statistical properties of the resulting waves. The only solution is to devise a specific scheme for each new dimension. The 2D case is already tricky but can be implemented with a reasonable effort (a future gem?). The 3D case appears to be much more difficult; volunteers are welcome.

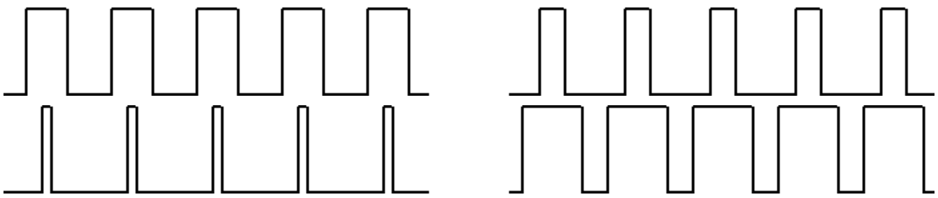

Figure 1: *Rectangular-like waves for different shape factors*
(a) $s = 0.0$ $f = 0.0$ $a = 0.0$ *(b)* $s = 0.6$ $f = 0.0$ $a = 0.0$
(c) $s = 0.9$ $f = 0.0$ $a = 0.0$ *(d)* $s = -0.6$ $f = 0.0$ $a = 0.0$

Figure 2: *Triangular-like waves for different shape factors*
(a) $s = 0.0 \quad f = 0.0 \quad a = 0.0$ *(b)* $s = 0.6 \quad f = 0.0 \quad a = 0.0$
(c) $s = 0.9 \quad f = 0.0 \quad a = 0.0$ *(d)* $s = -0.6 \quad f = 0.0 \quad a = 0.0$

Figure 3: *Rectangular-like waves for different frequency and amplitude variances*
(a) $s = 0.0 \quad f = 0.0 \quad a = 0.0$ *(b)* $s = 0.6 \quad f = 0.0 \quad a = 0.0$
(c) $s = 0.9 \quad f = 0.0 \quad a = 0.0$ *(d)* $s = -0.6 \quad f = 0.0 \quad a = 0.0$

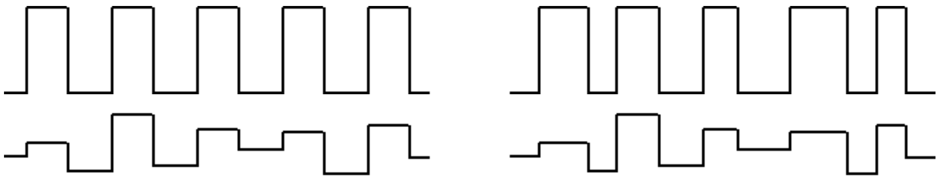

Figure 4: *Rectangular-like waves for different frequency and amplitude variances*
(a) $s = 0.0 \quad f = 0.0 \quad a = 0.0$ *(b)* $s = 0.0 \quad f = 1.0 \quad a = 0.0$
(c) $s = 0.0 \quad f = 0.0 \quad a = 1.0$ *(d)* $s = 0.0 \quad f = 1.0 \quad a = 1.0$

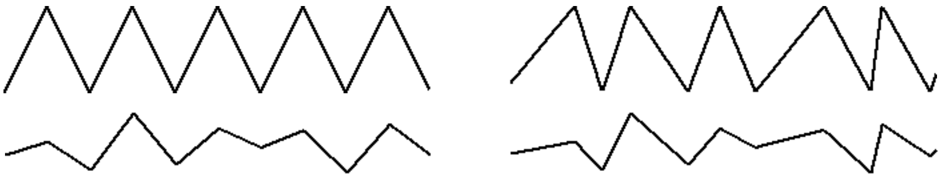

Figure 5: *Triangular-like waves for different frequency and amplitude variances*
(a) $s = 0.0 \quad f = 0.0 \quad a = 0.0$ *(b)* $s = 0.0 \quad f = 1.0 \quad a = 0.0$
(c) $s = 0.0 \quad f = 0.0 \quad a = 1.0$ *(d)* $s = 0.0 \quad f = 1.0 \quad a = 1.0$

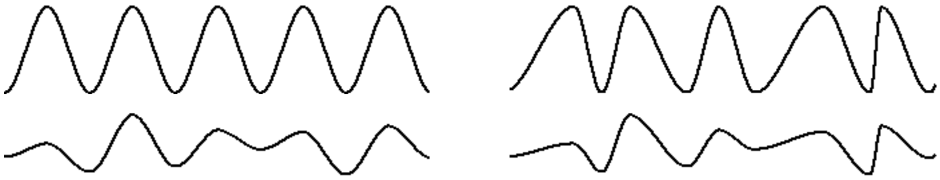

Figure 6: *Sinusoidal-like waves for different frequency and amplitude variances*
(a) $s = 0.0 \quad f = 0.0 \quad a = 0.0$ *(b)* $s = 0.0 \quad f = 1.0 \quad a = 0.0$
(c) $s = 0.0 \quad f = 0.0 \quad a = 1.0$ *(d)* $s = 0.0 \quad f = 1.0 \quad a = 1.0$

Source Files

```
/* ------------------------------------------------------------------------- *\
   WAVE.H : Christophe Schlick (10 September 1993)

   This package provides 3 routines for generating rectangular-like,
   triangular-like and sine-like waves including specific features.

   "Wave Generators for Computer Graphics"
   in Graphics Gems V (edited by A. Paeth), Academic Press
\* ------------------------------------------------------------------------- */

#ifndef _WAVE_
#define  _WAVE_

extern double Rwave (register double t, double s, double Fvar, double Avar);
extern double Twave (register double t, double s, double Fvar, double Avar);
extern double Swave (register double t, double s, double Fvar, double Avar);

#endif

/* ------------------------------------------------------------------------- *\
   WAVE.C : Christophe Schlick (10 September 1993)

   This package provides 3 routines for generating rectangular-like,
   triangular-like and sine-like waves including specific features.

   "Wave Generators for Computer Graphics"
   in Graphics Gems V (edited by A. Paeth), Academic Press
\* ------------------------------------------------------------------------- */

#include <math.h>
#include "wave.h"

/*
** Macro functions
*/

#define ABS(a)          ((a) < 0 ? -(a) : (a))
#define FLOOR(a)        ((a) < 0 ? (int) ((a)-1.0) : (int) (a))
#define MAX(a,b)        ((a) > (b) ? (a) : (b))
#define MIN(a,b)        ((a) < (b) ? (a) : (b))

/*
** rnd : Random function (adapted from Greg Ward in Graphics Gems II)
*/

static double rnd (register long s)
{
  s = s << 13 ^ s;
  return (((s*(s*s*15731+789221)+1376312589) & 0X7FFFFFFF) / 2147483648.0);
}

#define FRND(a)         rnd (17*(a))
#define ARND(a)         rnd (97*(a))

/*
```

```
** Rwave : Rectangular-like monodimensional wave
**
** Input : t = Wave parameter
**         s = Shape factor (in [-1,1])
**         f = Frequency variance (in [0,1])
**         a = Amplitude variance (in [0,1])
*/

double Rwave (register double t, double s, double f, double a)
{
  register int    i, j;
  register double a, b;

  i = j = FLOOR (t); t -= i; j++;

  if (f) {
    a = (FRND (i) - 0.5) * f;
    b = (FRND (j) - 0.5) * f + 1.0;
    t = (t-a) / (b-a);
  }

  if (i & 1) {i++; j--; t = 1.0-t;}
  t = (s < 0.0) ? (t+s*t) / (1.0+s*t) : (s > 0.0) ? t / (1.0-s+s*t) : t;
  t = t < 0.5 ? 0.0 : 1.0;

  if (a) {
    a = ARND (i) * a * 0.5;
    b = ARND (j) * a * 0.5;
    t = a + t * (1.0-a-b);
  }
  return (t);
}

/*
** Twave : Triangular-like monodimensional wave
**
** Input : t = Wave parameter
**         s = Shape factor (in [-1,1])
**         f = Frequency variance (in [0,1])
**         a = Amplitude variance (in [0,1])
*/

double Twave (register double t, double s, double f, double a) {
  register int    i, j;
  register double a, b;

  i = j = FLOOR (t); t -= i; j++;

  if (f) {
    a = (FRND (i) - 0.5) * f;
    b = (FRND (j) - 0.5) * f + 1.0;
    if (t < a) {
      i--; j--; t++; a++;
      b = a; a = (FRND (i) - 0.5) * f;
    } else if (t > b) {
      i++; j++; t--; b--;
      a = b; b = (FRND (j) - 0.5) * f + 1.0;
```

```
    }
    t = (t-a) / (b-a);
  }

  if (i & 1) {i++; j--; t = 1.0-t;}
  t = (s < 0.0) ? (t+s*t) / (1.0+s*t) : (s > 0.0) ? t / (1.0-s+s*t) : t;

  if (a) {
    a = ARND(i) * a * 0.5;
    b = ARND(j) * a * 0.5;
    t = a + t * (1.0-a-b);
  }
  return (t);
}

/*
** Swave : sinusoidal-like monodimensional wave
**
** Input : t = Wave parameter
**         s = Shape factor (in [-1,1])
**         f = Frequency variance (in [0,1])
**         a = Amplitude variance (in [0,1])
*/

double Swave (register double t, double s, double f, double a)
{
  register int    i, j;
  register double a, b;

  i = j = FLOOR (t); t -= i; j++;

  if (f) {
    a = (FRND (i) - 0.5) * f;
    b = (FRND (j) - 0.5) * f + 1.0;
    if (t < a) {
      i--; j--; t++; a++;
      b = a; a = (FRND (i) - 0.5) * f;
    } else if (t > b) {
      i++; j++; t--; b--;
      a = b; b = (FRND (j) - 0.5) * f + 1.0;
    }
    t = (t-a) / (b-a);
  }

  if (i & 1) {i++; j--; t = 1.0-t;}
  t = (s < 0.0) ? (t+s*t) / (1.0+s*t) : (s > 0.0) ? t / (1.0-s+s*t) : t;
  t *= t * (3.0-t-t);

  if (a) {
    a = ARND(i) * a * 0.5;
    b = ARND(j) * a * 0.5;
    t = a + t * (1.0-a-b);
  }
  return (t);
}
```

Bibliography

[1] Perlin, K., *An Image Synthesizer*, Computer Graphics, v19, n3, p287-296, 1985.

[2] Peitgen, H. and Saupe, D., *The science of fractal images*, Springer Verlag, 1988.

[3] Perlin, K., *Hypertexture*, Computer Graphics, v23, n3, p253-262, 1989.

[4] Upstill, S., *The Renderman Companion*, Addison-Wesley, 1989.

[5] Schlick, C., *Fast Alternatives to Perlin's Bias and Gain Functions*, Graphics Gems, vol4, p401-404, Eds Paul Heckbert, Academic Press, 1994.

[6] Ward, G., *A Recursive Implementation of the Perlin Noise Function*, Graphics Gems vol2, p253-259, Eds James Arvo, Academic Press, 1991.