

Sobre Tron y las nubes. Un acercamiento a las texturas procedurales

María Sebastiá Fortes y Antonio Pariente Granero

December 2020



Índice

1 Abstract	3
2 Antecedentes	4
3 Generación de Ruido Perlin	7
3.1 Paso 1	8
3.2 Paso 2	9
3.3 Paso 3	9
3.4 Paso 4	10
4 Ejemplos y Aplicaciones	10
4.1 Ejemplo 1: Aplicaciones en 2-D	10
4.2 Ejemplo 2: Aplicaciones 3-D	14
4.3 Ejemplo 3: Aplicaciones en animación	14
5 Conclusión	15
6 Bibliografía	16

1 Abstract

Las texturas procedurales, son ampliamente utilizadas en la industria audiovisual. El Ruido Perlin, interpreta pues, un papel esencial como precursor de las mismas, con aplicaciones aún vigentes hoy en día.

El objetivo es ilustrar una manera de crear texturas que imiten mejor objetos más "naturales", esto es, una señal pseudo-aleatoria que logre valores más coherentes con elementos de la naturaleza o el entorno. Para esto, se explicará el uso del *Ruido Perlin* o *Perlin Noise*, además de varios ejemplos actuales.

Tanto para ilustrar la generación de ruido en varias dimensiones, como para sus ejemplos, se ha hecho uso de la última versión del algoritmo de generación publicado por Ken Perlin, así como Microsoft Visual Code y Python 3.9 para la generación de ejemplos, de una manera más cercana, al alcance del lector.

Los resultados han sido pues satisfactorios, haciendo uso del Ruido Perlin, se puede lograr una generación visiblemente más "natural", con un aspecto que se aleja más de unas texturas hechas por ordenador. Combinando este ruido con diversas fórmulas matemáticas, logramos diferentes efectos que guardan gran similitud con elementos presentes en la naturaleza.

Todo esto, nos proporciona una ventana de visión a un mundo de posibilidades, con grandes aplicaciones aún hoy en la actualidad.

2 Antecedentes

Un suceso es *aleatorio*, cuando no es posible *prever* su resultado de antemano, sin intervención del azar. Este resultado desconocido, puede ser un valor, en cuyo caso recibe el nombre de *valor aleatorio*.

¿Cómo lo elegirías?. La respuesta no es tan sencilla como a priori pudiera parecer, pues en el caso del ser humano, no es posible escoger un valor al azar, ya que somos fuertemente sugestionables, podría suceder que este valor pensado o escogido, estuviera condicionado por factores que desconocemos. En la naturaleza, los valores puramente aleatorios también son escasos, pues detrás de un suceso y conociendo las condiciones en las que se produce, gracias a las leyes físicas, podemos prever su resultado.

En los ordenadores sucedería de un modo análogo, pues éstos obeceden a las intrucciones que los seres humanos les proporcionan, y con mayor o menor precisión las cumplen de una manera determinada dando siempre el mismo resultado. Es así que se debe hablar de valores *pseudo-aleatorios*.

Éstos valores son el resultado de funciones que devuelven siempre el mismo resultado $f(x)$ para el mismo valor de entrada x . La diferencia radica en que pequeños cambios en la x , producen grandes cambios en la imagen $f(x)$. Si por ejemplo esta x fuera el tiempo que ha transcurrido en milisegundos desde que se encendió el ordenador, resultaría *muy difícil* prever el resultado o repetirlo a conciencia.

Cerca de 1981, Ken Perlin (el apellido nos es familiar), estaba trabajando en lo que fue la película TRON (Figura 1), más concretamente, en la empresa MAGI (*Mathematical Association Group .Inc*), encargada de los efectos especiales y las texturas hechas por ordenador.



Figure 1: Portada de TRON (1983)

La *textura* de un objeto hecho por ordenador, suele ser una imagen que se estira y deforma para envolver al mismo y dar la sensación de realidad, como en la Figura 2. Se trata del aspecto de los objetos: sombras, colores, etc. Éstas texturas pueden estar diseñadas por ordenador, por lo que se conoce como *diseñadores gráficos*.



Figure 2: Dinosaurio con y sin texturas

Muchas de las texturas y figuras de TRON, estaban construidas a parte de booleanos de superficies en \mathbb{R}^3 (Figura 3), intersecciones donde se añaden o se quitan diferentes objetos tales como esferas, paraboloides, etc.

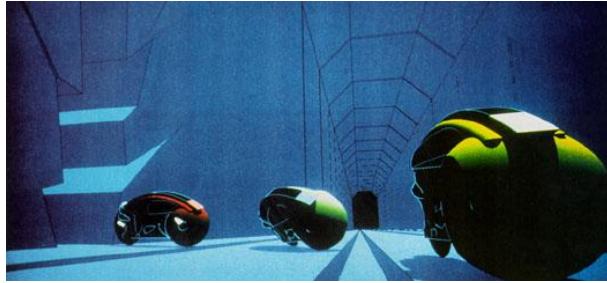


Figure 3: Motos de luz hechas por ordenador

Pero si quisieramos generar grandes texturas variadas, como suelos o paredes con grietas, que pareciesen naturales y diferentes, deberíamos recurrir a los valores aleatorios. Para Ken Perlin, las texturas generadas por ordenador, resultaban muy antinaturales.

Si se generaran valores aleatorios, digamos, para imitar el perfil de unas montañas en un paisaje, dado que estos valores no guardan ningún tipo de relación, obtendríamos un visible mal resultado. Valores muy diferentes estarían generados unos consecutivos a otros y no se podría "andar" o "caminar" por éstas montañas, ésto es lo que sucede en la Figura 4.

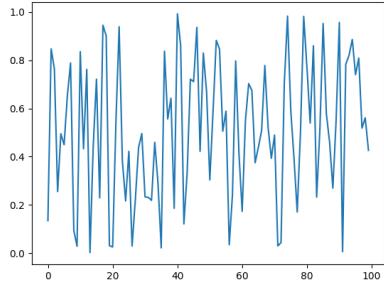


Figure 4: Valores aleatorios generados por ordenador

Fué entonces cuando Perlin dió con una buena solución a este problema con el algoritmo de generación de valores *pseudo-aleatorios* que lleva su nombre.

En términos de señales, el ruido de una imagen o señal de audio, es el conjunto de valores no deseados, aleatorios, que perturban los datos originales. Pueden producirse por ejemplo, como consecuencia de errores en los sensores, como en un micrófono o el objetivo de una cámara, como se puede observar en la Figura 5.

El ruido que trata este trabajo, no es tanto un conjunto de valores aleatorios, sino valores pseudo-aleatorios *controlados* para *crear* imágenes o señales. Así pues tratan de llenar el espacio disponible con información que no existía.



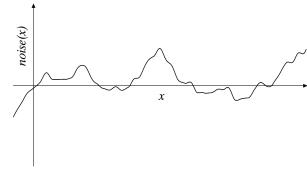
Figure 5: Ruido en una imagen con poca iluminación

Éstos valores pseudo-aleatorios de *ruido*, son después transformados por fórmulas matemáticas para lograr diferentes resultados, que reciben el nombre de texturas procedurales.

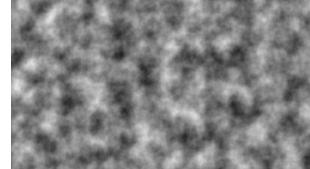
Suponen algo realmente importante en la tecnología y el mundo audiovisual, pero también en un nivel antropológico. Es la capacidad que el ser humano ha desarrollado para *imitar* a la naturaleza, de recrearla a su antojo. Este hecho es, a ojos de los autores de este documento, un paso evolutivo, pues ya no se trata de tener poder y conocimiento para dominar y moldear el entorno que nos rodea, como han hecho hasta ahora la ingeniería o la medicina. El objetivo es ahora crear un mundo a voluntad, que irónicamente, y para quién no haya tenido el placer de ver la película, resulta el hilo argumental de TRON. Una de esas casualidades que ligan el arte y el futuro de la raza humana.

3 Generación de Ruido Perlin

El Ruido Perlin no deja de ser una función tal $f : \mathbb{R}^n \rightarrow \mathbb{R}$, vemos pues que puede aplicarse a varias dimensiones y la imagen siempre será un valor en \mathbb{R} , con diferentes significados:



(a) $f : \mathbb{R} \rightarrow \mathbb{R}$, el valor generado se toma como la altura.



(b) $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, el valor generado se toma como la intensidad del color en escala de grises.

(c) $f : \mathbb{R}^4 \rightarrow \mathbb{R}$, el valor generado es la altura, a partir de \mathbb{R}^3 y el tiempo.

Figure 6: Ruido Perlin varias dimensiones

Se procede a explicar la generación de Ruido Perlin para dos dimensiones, es decir, $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, pues se considera el ejemplo visualmente más claro. Para el resto de dimensiones, cabe escalar el Paso 1, pero no se producen cambios significativos en la estructura general del algoritmo.

3.1 Paso 1

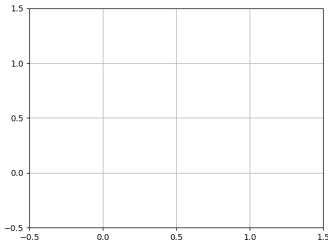
El primer paso consta en dividir el espacio, en este caso \mathbb{R}^2 , como se puede observar en la malla de la Figura 7,a. Los vértices de esta malla, internos y externos, reciben el nombre de nodos.

En cada uno de los nodos, se genera un vector gradiente pseudo-aleatorio de módulo 1, de tal manera que un nodo tiene siempre el mismo vector gradiente asociado, como muestra la Figura 7,b. Al generar éstos vectores, cabe eludir valores parecidos entre nodos cercanos, pues esto podría producir patrones que reducirían el factor de aleatoriedad. No obstante, no es importante si dichos patrones se producen a grandes distancias, es decir, cuando el observador se encuentra lejos de la textura o ésta abarca grandes objetos.

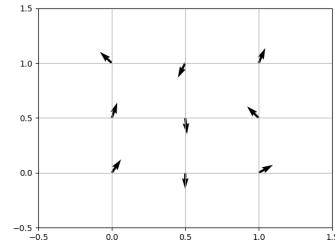
Para computar dichos vectores se usa un algoritmo que tiene su base en el método de Montecarlo, se generan puntos en un marco con una circunferencia de radio unidad, y se escogen los que se encuentran dentro de la misma, que más tarde se normalizan para que el módulo sea unitario. Seguidamente, se genera una tabla de permutaciones del tamaño de la malla y se aplica una función Hash, la cual mapea los vectores aleatorios generados a los nodos de la malla. Dicha función Hash tiene el siguiente aspecto:

$$G = G[(i + P[j + P[k]_{modn}])_{modn}]$$

Donde n es la dimensión de la malla.



(a) Malla en \mathbb{R}^2



(b) Gradientes pseudo-aleatorios

Figure 7: Paso 1

3.2 Paso 2

En cada celda se genera un único punto aleatorio que se encuentra dentro de los límites de la misma, como se observa en la Figura 8. Para cada nodo de la celda, se genera el vector del nodo a dicho punto. Se computa entonces, el producto escalar del vector gradiente en cada nodo y el vector nodo-punto.

Si ahora visualizamos la generación como altura en un tablero, y a nosotros mismos caminando por él, los vectores gradientes de cada nodo, supondrían la dirección en la que el terreno o su *pendiente*, tira o empuja de nosotros en lo que caminamos en dirección al punto generado siguiendo el vector nodo-punto. El producto escalar nos da la información de cómo afecta el gradiente al "camino" hacia el punto, si es negativo, la pendiente *tirará* de nosotros, pues el vector gradiente se encuentra en dirección contraria a la del camino nodo-punto, en el caso positivo, nos *empujará* para llegar antes al punto generado correspondiente a la celda. Por supuesto dicha relación se encuentra en $[-1, 1]$, pues es el resultado de un producto escalar de vectores unitarios o con módulo menor que 1.

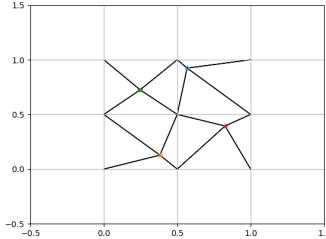


Figure 8: Punto aleatorio y vectores nodo-punto

Es éste producto escalar, el valor del Ruido Perlin en cada nodo, pero ¿Cómo se extendería de manera "natural" a toda la celda?.

3.3 Paso 3

La solución para extender el valor del ruido en cada nodo a toda la celda, es **interpolar** entre dichos valores en cada dimensión del espacio de partida. La clase de función de interpolación escogida es la llamada **Smoothstep**, y engloba funciones de tipo **sigmoide**. Generalmente son Splines de orden n que pasan por los valores generados en los nodos.

En su documento, Ken Perlin, recomienda el uso del polinomio $6t^6 - 15t^4 + 10t^3$, dicho polinomio se anula en la primera y segunda derivada en los correspondientes nodos, aportando así la suavidad necesaria para que la transición

de valores sea de aspecto *natural*. La Figura 9 proporciona un ejemplo muy visual de cómo actúa este polinomio.

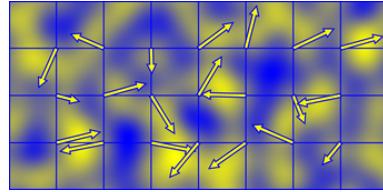


Figure 9: Interpolación de los valores de ruido

3.4 Paso 4

En este punto, se tienen pues valores de aspecto **natural** en los nodos de la malla, extendidos de manera suave sobre las diferentes celdas. El algoritmo pues, depende de dos valores esencialmente:

- **Amplitud:** supone el rango del ruido. En el algoritmo se ha tratado el caso en el que éste rango está entre $[-1, 1]$, fruto del producto escalar. No obstante, dicho intervalo puede extenderse a un genérico $[a, b]$ escalando el algoritmo.
- **Frecuencia:** supone el tamaño de la red o malla, que en el caso de \mathbb{R}^2 tratado hasta el momento, eran cuadrados unidad, aunque fácilmente puede variarse ésta medida en cualquier dimensión.

Combinando diferentes Ruidos Perlin generados con diversas características de Amplitud y Frecuencia, logramos resultados muy variados, que a un nivel de textura proporcionan mayor sensación de diversidad, haciendo los resultados mucho más interesantes para el propósito inicial, como se refleja en la figura 10.

También pueden combinarse dichos valores generados de ruido con diversas fórmulas matemáticas como las de naturaleza fractal de los ejemplos que siguen.

4 Ejemplos y Aplicaciones

4.1 Ejemplo 1: Aplicaciones en 2-D

En este ejemplo, se tomara el Ruido Perlin como una función $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, donde el valor generado se usará como intensidad de color. Se hará uso de diferentes circunferencias en el espacio sobre las cuales se aplicarán las texturas.

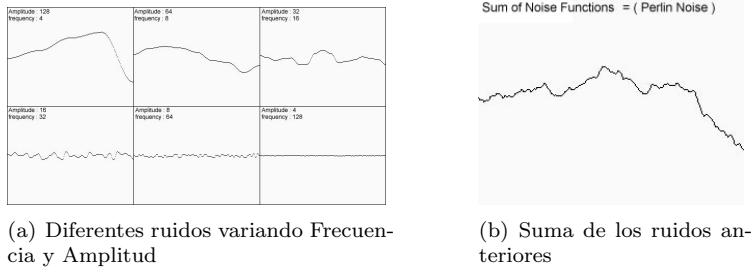


Figure 10: Paso 3

Just Noise: Para esta textura, se ha usado el Ruido Perlin sin alterar, como su propio nombre indica. Se puede reconocer fácilmente el aspecto "natural" que hemos perseguido a lo largo del trabajo.

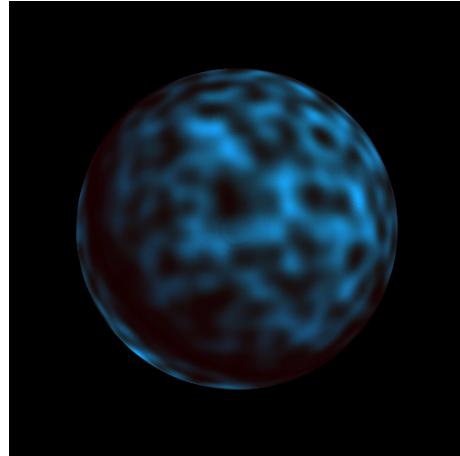


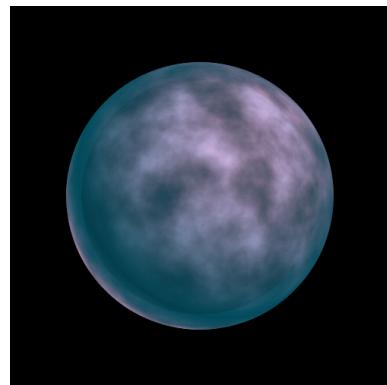
Figure 11: Noise Ball

Para las siguientes texturas de este ejemplo, se han aplicado a los valores generados de Ruido Perlin, diversas fórmulas matemáticas de naturaleza fractal.

Cloudy Noise: Para esta textura, se ha usado la fórmula fractal $F(x) = \text{noise}(x) + \frac{1}{2}\text{noise}(2x) + \frac{1}{4}\text{noise}(4x)\dots$. El caso presente, se trata de 8 iteraciones de profundidad, pues a partir de ahí los cambios no son visibles.

Con la fórmula fractal, lo que conseguimos es mayor **fluctuación** de los valores aleatorios entre píxeles, es decir, mayor cambio de color. La textura obtenida es muy similar a la de una nube natural, de ahí el nombre.

Flame Noise: De nuevo, se le aplica la fórmula fractal anterior, pero ésta vez con valor absoluto: $F(x) = |\text{noise}(x)| + \frac{1}{2}|\text{noise}(2x)| + \frac{1}{4}|\text{noise}(4x)|\dots$. Con esto, logramos que los cambios entre valores sean más bruscos y por lo tanto, que se produzcan discontinuidades en el gradiente. El efecto es parecido al de unas llamas y por tanto recibe el nombre de Flame Ball.



(a) Cloudy Ball



(b) Flame Ball

Figure 12: Ejemplo 1

-Marble Noise: Para el último ejemplo, se aplica una fórmula de naturaleza periódica, a la ya usada en la Flame Ball: $F(x) = \sin(|\text{noise}(x)| + \frac{1}{2}|\text{noise}(2x)| + \frac{1}{4}|\text{noise}(4x)|\dots)$. Con esto se consiguen texturas casi periódicas como la del ejemplo que sigue, que logra imitar la textura del mármol natural.

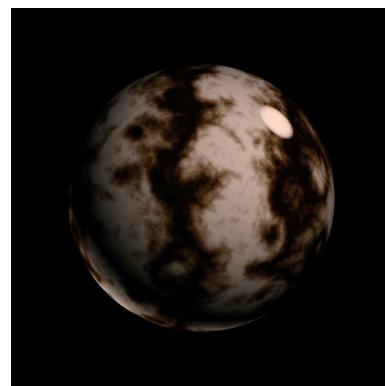


Figure 13: Marble Ball

Si en vez de considerar los valores de ruido como intensidad de color, se escogiese la altura, podríamos estar hablando de la generación de terreno usada en el videojuego Minecraft. Cabe destacar que las normas de Minecraft restringen la altura máxima generada y varían según el bioma, pero el principio es el mismo.

Este ruido, proporciona al jugador, la sensación de que el mundo es natural ya que no habrá dos mapas exactamente iguales, y dentro de un mapa, no habrá zonas repetidas. Nos encontramos pues en un mundo que da la sensación de ser vasto y real.



Figure 14: Generación de terreno en la superficie de Minecraft

4.2 Ejemplo 2: Aplicaciones 3-D

Para este ejemplo, estaremos tratando el Ruido Perlin, como la función $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. En este caso particular, consideramos como dominio, un cubo en el espacio, dividido en una malla de subcubos. Si valor de ruido generado es superior a un valor concreto, ese cubo se elimina del espacio, en caso contrario, se deja. Así, somos capaces de generar cuevas y cavernas que dan la sensación de haber sido erosionadas por factores reales.

De nuevo, aunque más decorado y florecido (no mucho más), este es el principio que rige la generación de cuevas en el videojuego Minecraft.

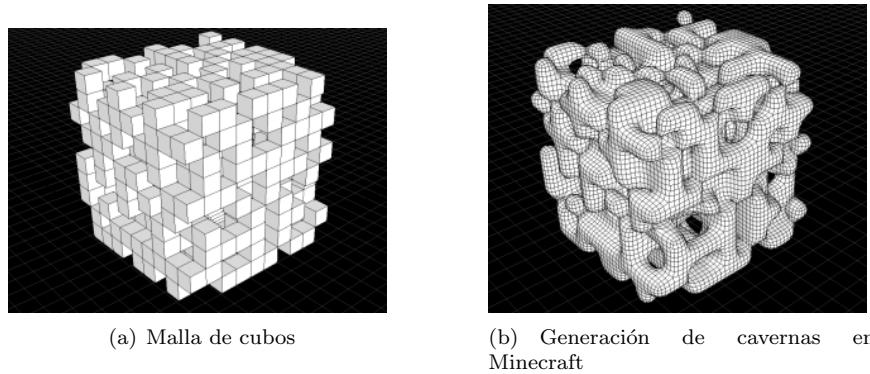


Figure 15: Ejemplo 2

4.3 Ejemplo 3: Aplicaciones en animación

En esete caso, estaremos hablando de una cunción $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ muy interesante. Este espacio \mathbb{R}^3 está compuesto por el plano y el tiempo. El valor de ruido generado será el ángulo de un vector con respecto al eje horizontal entre 0 y 2π . Así, estaremos generando a cada t , una serie de vectores orientados en el espacio. Claramente eliminaremos del algoritmo de la interpolación. pues estos vectores tan sólo nos interesan en los bordos de la malla.

El resultado que logramos es el de un campo vectorial de flujo que cambia con el tiempo. Si decidimos dibujar partículas y soltarlas en este campo, cada vector generado aceleraría cada partícula que pasara por su celda correspondiente, creando así una animación. Dicha animación simula sorprendentemente bien el humo.



Figure 16: Animación P5

5 Conclusión

Gracias un algoritmo sencillo pero eficaz como el Ruido Perlin, se pueden lograr resultados altamente satisfactorios y con un número de aplicaciones sorprendente. El Ruido Perlin se lleva usando en los gráficos por ordenador numerosos años y le valió a su autor, el primer (y seguramente último) óscar ganado por un matemático, demostrando así que existe también arte en una de las ciencias más antiguas de la humanidad. Ilustrando que un trozo de las mismas subyace en cada cosa que construimos.

6 Bibliografía

Los siguientes enlaces y referencias son tanto de información como recursos:

-*Making Noise*, Author: Ken Perlin (Dec,9,1999), Url: [WaybackMachine](#)

-*Simplex noise demystified*, Author: Stefan Gustavson (1005-03-22), Publisher: Linköping University, Sweden, Url: [Simplex noise](#)

-*Understanding Perlin Noise*, Author: Flafla2 (09, August 2014), Url: [Technical Writeup](#)

-*Perlin Noise: What is it, how to use it, and why it's better than value noise*, Author: Hirnschall, Url:[Blog](#)

- *Wikipedia*, Url: [Perlin Noise](#)

- *Youtube-CodingTrain*, Url: [Website](#), [Smoke Sim in JS](#), [Perlin Noise](#)

- *Microsoft Visual Code* como editor de código y *Python 3.9.0* como compilador.