

Computación Gráfica

Texturas Procedurales

Grupo 2

Guillermo Campelo
Juan Ignacio Goñi
Juan Tenaillon
Santiago Vázquez

June 21, 2010

Abstract

Se agregó al motor de Ray Tracing, la posibilidad de utilizar texturas procedurales. Para lograr esto, se implementó Perlin Noise, y se implementaron texturas que se asemejan al fuego, agua, granito, madera, mármol y material de tipo orgánico para triángulos y esferas.

1 Introducción

A lo largo de este informe, detallaremos el diseño y desarrollo de la extensión al motor de ray tracing. Esta extensión consistió en agregar texturas procedurales al motor, mediante la implementación de Perlin Noise.

Para realizar la extensión, partimos desde la base de la entrega anterior, donde lo único que tuvimos que hacer fue implementar los nuevos *shaders*.

Para obtener los mismos resultados que obtiene Ken Perlin con su algoritmo de texturas procedurales, realizamos una investigación sobre los distintos tipos de ruidos que él utilizó, así como también con distintas fórmulas para aplicar a ese ruido y obtener así las distintas texturas.

En la sección 2 entraremos en más detalle con respecto a los posibles ruidos, en la sección 3 comenzaremos a explicar cada textura en particular, mostrando cómo fue lograda y mostrando ejemplos de la misma. En la sección 4 mostraremos cómo decidimos representar los distintos *shaders*, y daremos valores recomendados para su uso. Por último, comentaremos los resultados obtenidos así como nuestras conclusiones en la sección 5.

2 Ruido

Como todos sabemos, la idea general de Perlin Noise es la de realizar sumas sucesivas de ruidos a distintas amplitudes, para luego aplicarle a esa sumatoria una función que nos determinará el color en el punto.

Entonces, es necesario definir primero el ruido para después poder realizar las sumas correspondientes. Dentro de las posibilidades con las que nos encontramos, resaltaban dos: Ruido Aleatorio y Ruido Mejorado.

2.1 Ruido Aleatorio

La idea de ruido aleatorio es básicamente esa. Generamos una matriz de cierto tamaño con números aleatorios entre 0 y 255. De esta forma, se van a ir realizando las sucesivas sumas y su resultado será nuevamente un número aleatorio.

Esta fue la primera implementación de Ken Perlin, que luego fue suplantada por el Ruido Mejorado que comentamos a continuación en la sección 2.2.

2.2 Ruido Mejorado

El ruido mejorado parte de la base del ruido aleatorio, originándose con un vector de 256 valores aleatorios entre 0 y 255 inclusive. Luego se crea otro vector de permutación, que es luego utilizado para realizar las sumas de los ruidos.

La ventaja que presenta este tipo de ruido con respecto al aleatorio, es que este es más eficiente para los cálculos que realiza, obteniendo así una mejora en el desempeño del algoritmo para generar las texturas.

Es debido a estas mejoras, presentadas por Ken Perlin en su paper “Improving Noise”, es que decidimos realizar la segunda implementación de ruido para generar nuestras texturas.

3 Texturas

Las texturas que implementamos son las siguientes: fuego, agua, material orgánico, piedra tipo granito, madera y mármol. A continuación, mostraremos para cada una de las texturas, la fórmula utilizada y el resultado obtenido.

3.1 Fuego y Agua

Para realizar la textura que simule al fuego y al agua, utilizamos la misma fórmula pero variando los colores aplicados.

La fórmula utilizada fue:

$$\text{sum}(\frac{1}{f} * |\text{Noise}|) = |\text{Fnoise}(p)| + \frac{1}{2}|\text{Fnoise}(2p)| + \frac{1}{4}|\text{Fnoise}(4p)| + \dots \quad (1)$$

Donde, FNoise es la función que dado un punto p me devuelve el ruido en ese punto y donde f es la frecuencia. La idea sería realizar sumas sobre los puntos, donde a medida que se hace más *profunda* la suma van perdiendo fuerza los términos y siempre tomando el valor absoluto.

En las figuras 1 y 2 se observa la textura obtenida para simular fuego así como la textura obtenida para simular agua.

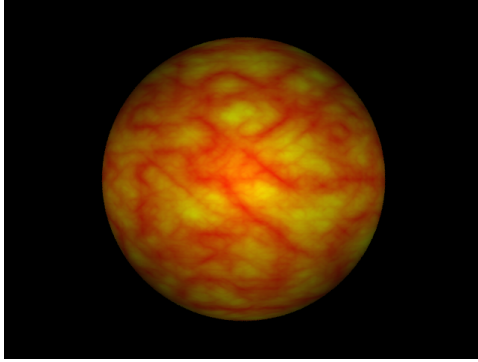


Figure 1: Fuego Procedural

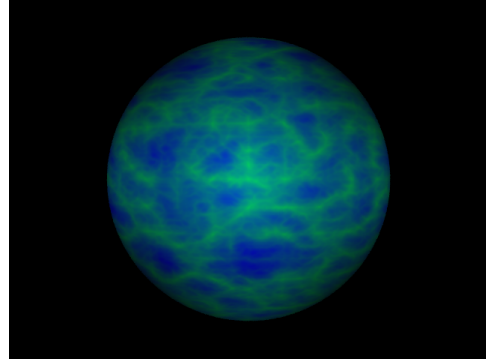


Figure 2: Agua Procedural

3.2 Material Orgánico

Para realizar una textura que simule a un material orgánico, tuvimos que determinar primero que era un material orgánico y luego realizar experimentaciones para obtener algo que lo simule. Luego de realizar distintas pruebas, y fallar en el intento, nos inclinamos por utilizar solamente la función de ruido de perlin.

$$Noise = Fnoise(p) + \frac{1}{2}Fnoise(2p) + ... \quad (2)$$

De esta forma, lo que simulamos es una superficie con moho, un tipo de hongos. En la figura 3 se puede observar el resultado obtenido.

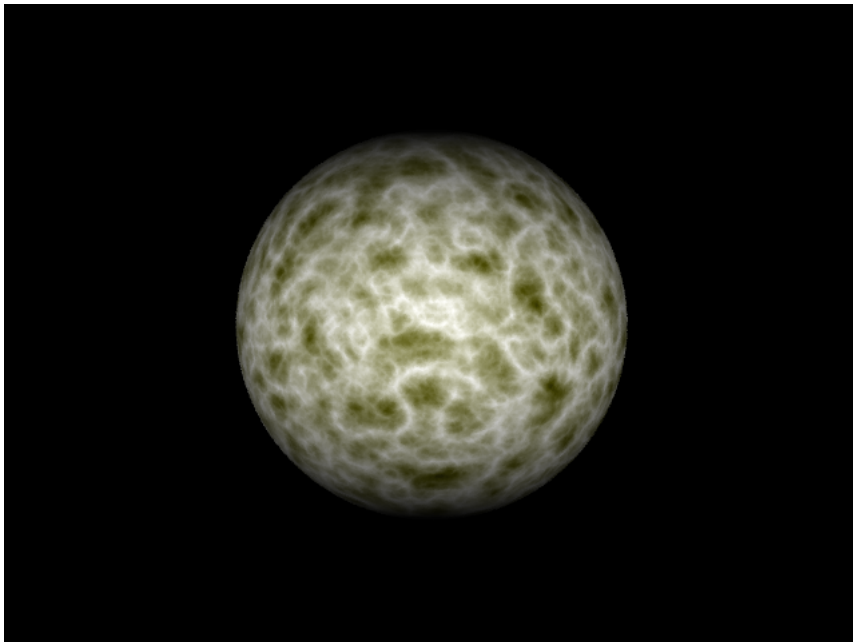


Figure 3: Material Orgánico Procedural

3.3 Piedra tipo Granito

Esta textura fue otra de las que nos trajo aparejados cierto problemas. Al investigar la textura del granito, no lograbamos encontrar la fórmula que nos permita simularlo.

Luego de intentar mediante prueba y error, advertimos que utilizando solamente la función de ruido, pero escalandolo a un objeto de mayor tamaño, podíamos obtener algo que simule granito. Es por eso que utilizamos la misma ecuación que para el Material Orgánico, pero viendolo desde muy lejos, donde obtuvimos lo que se puede observar en la figura 4.

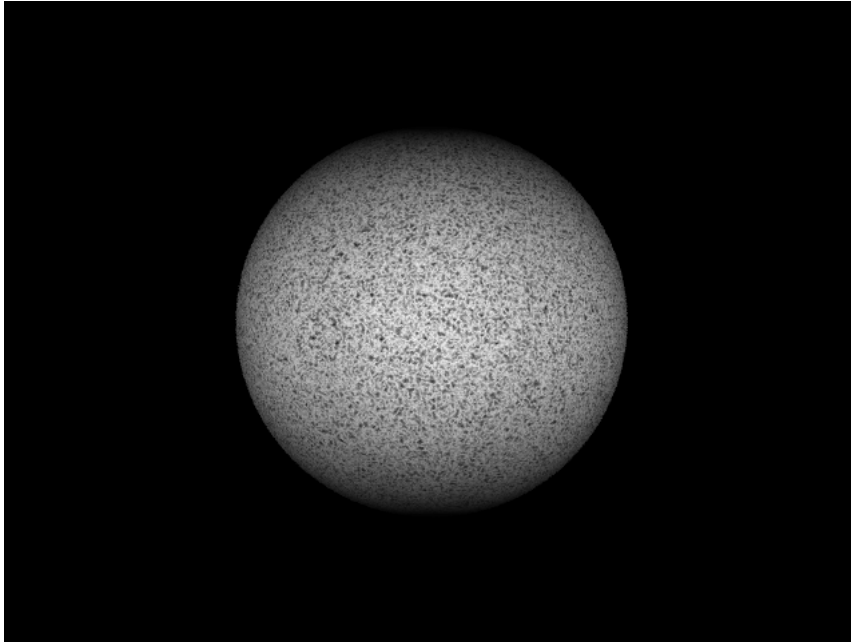


Figure 4: Piedra tipo Granito Procedural

3.4 Madera

Para obtener madera, utilizamos la función de ruido sin ningún agregado, pero al resultado obtenido le realizamos una corrección para obtener los anillos que nos dan la impresión de un árbol. Esta corrección es la resta de la parte entera del ruido que se puede observar en la ecuación 4.

$$Noise = Fnoise(p) + \frac{1}{2}Fnoise(2p) + ... \quad (3)$$

$$Noise = Noise - (int)Noise \quad (4)$$

De esta manera, obtuvimos como resultado lo que se puede observar en la figura 5.

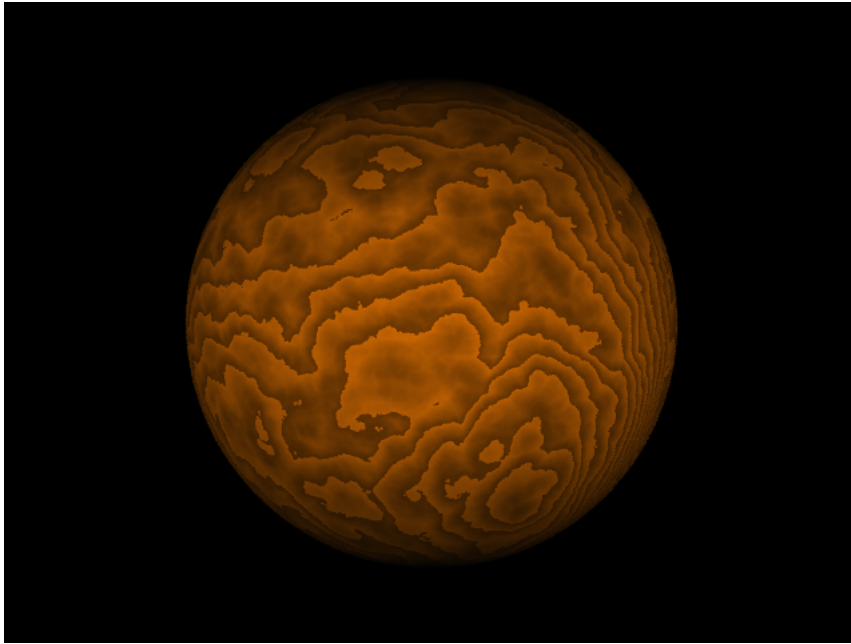


Figure 5: Madera Procedural

3.5 Mármol

Para obtener mármol, utilizamos la función propuesta por Ken Perlin. Este consiste en:

$$\sin(x + \sum(\frac{1}{f}|noise|)) = \sin(x + |noise(p)| + \frac{1}{2}|noise(2p)| + \dots) \quad (5)$$

Donde el resultado obtenido se observa en la figura 6.

4 Representación

5 Conclusión

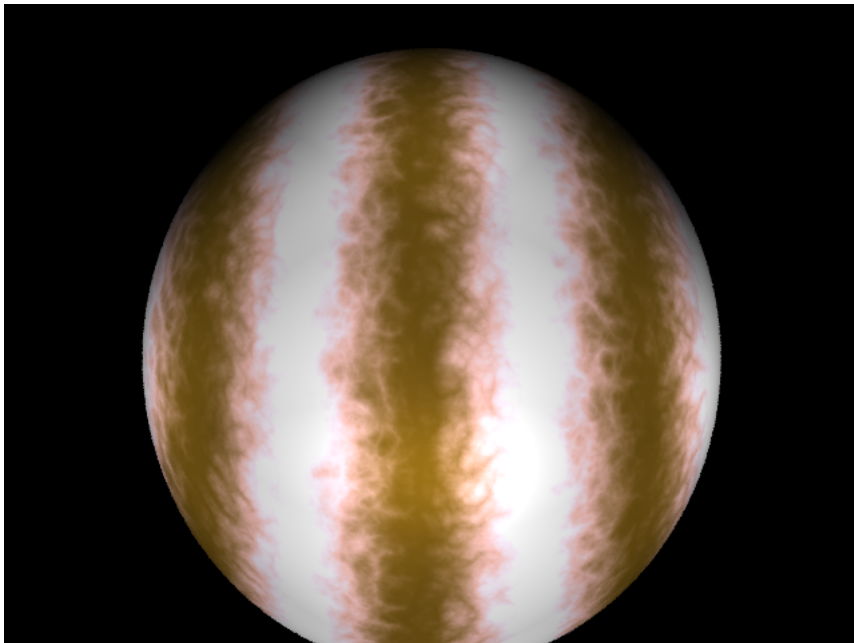


Figure 6: Mármol Procedural