Modelado procedural de terreno

Ignacio García Fernández

29 de noviembre de 2020

1. Introducción

Supondremos una malla con $2^N \times 2^N$ cuadros, asociada a una rejilla con $2^N + 1 \times 2^N + 1$ vértices. El objetivo de los algoritmos de modelado procedural de terreno es asignar un valor $h(v)$ a cada vértice v de la rejilla, que representará su altura.

Los algoritmos que veremos parten de este cuadrado inicial y realizan una serie de iteraciones, en las que en cada iteración se divide cada cuadrado de la iteración anterior en cuatro cuadrados iguales y se calculan todos los vértices nuevos de estos cuadrados. En la iteración $n = 0$, sólo los vértices de las esquinas están inicializados. Corresponden a los índices $(0, 0)$, $(0, 2^N)$, $(2^N, 0)$ y $(2^N, 2^N)$.

En los diferentes casos, el valor de la altura de un vértice se calculará promediando a partir de la altura de otros vértices cuya altura ya es conocida, y se le sumará una cantidad aleatoria z. Por ejemplo, si un vértice v se calcula a partir de dos vértices ya conocidos, v_1 y v_2, entonces la nueva altura vendrá dada por

$$h(v) = \frac{h(v_1) + h(v_2)}{2} + z. \tag{1}$$

La altura aleatoria z deberá ser un valor cada vez más pequeño a medida que el número de iteraciones, n, aumenta. Esto es debido a que la separación entre vértices se reduce exponencialmente a medida que n aumenta, por lo que un incremento aleatorio de tamaño constante produciría pendientes cada vez mayores. Por tanto, lo habitual es hacer que el tamaño máximo de z se reduzca de manera similar. Para el cálculo de z utilizaremos dos parámetros; la amplitud máxima inicial, $A > 0$, y la tasa de decaimiento de la amplitud, $H \in [0, 1]$. El valor de z se calculará como

$$z = A \cdot R \cdot 2^{-nH}, \tag{2}$$

donde R es un valor aleatorio $R \in [-1, 1]$, normalmente calculado con la distribución uniforme. Si $H = 1$ el valor máximo de z decaerá a la misma velocidad que la separación entre puntos de la malla, con lo que obtendremos relieves más suaves. Si por el contrario $H = 0$ no se produce ningún decaimiento de la magnitud de z, con lo que obtendremos relieves extremadamente rugosos. Al código mostrado en el Algoritmo 1 lleva a cabo el cálculo de R con propiedades indicadas anteriormente.

```
def get_rand():
    """ Devuelve un valor random entre -1 y 1
    """
    return 2.0*(np.random.rand()-0.5)
#
```

Algoritmo 1: Función que devuelve un valor aleatorio $z \in [-1, 1]$ utilizando la distribución uniforme.

1.1. Cálculo de las coordenadas de los vértices

Durante los diferentes desarrollos utilizaremos los siguientes cálculos. En primer lugar, utilizaremos que $2^n/2 = 2^{n-1}$. En la iteración $n = 0$ (estado inicial) se asigna valor a las esquinas. Tenemos un único cuadrado de lado 2^N.

Al completar el paso $n-1$, tenemos la rejilla dividida en $2^{n-1} \times 2^{n-1}$ cuadros. Cada cuadro tiene lado $d_{n-1} = 2^{N-(n-1)}$ y la esquina inferior izquierda del cuadro (i, j) tiene por coordenadas

$$(x, y) = (id_{n-1}, jd_{n-1}), \tag{3}$$

donde i y j toman valores entre 0 y $2^{n-1} - 1$. Es importante indicar que i aumenta cuando nos desplazamos hacia la derecha (cuando la coordenada x aumenta) y que j aumenta cuando nos desplazamos hacia arriba (cuando la coordenada y aumenta). Las cuatro esquinas del cuadro anterior tendrán por coordenadas:

$$\begin{aligned} v_1 &= (x, y), \\ v_2 &= (x, y) + (0, d_{n-1}), \\ v_3 &= (x, y) + (d_{n-1}, 0), \\ v_4 &= (x, y) + (d_{n-1}, d_{n-1}). \end{aligned} \tag{4}$$

Al iniciar el paso n, cada uno de los cuadros anteriores se divide en 4, quedando subcuadrados de lado $d_n = 2^{N-n}$. Para ello, asignaremos la altura del vértice en el centro de cada cuadro, y de los vértices de los cuatro lados. El vértice del medio del cuadrado (i, j) del paso anterior tiene por coordenadas

$$v_m = (x, y) + (d_n, d_n), \tag{5}$$

y los centros de los lados tienen por coordenadas

$$\begin{aligned} v_n &= (x, y) + (d_n, 2d_n), \\ v_s &= (x, y) + (d_n, 0), \\ v_e &= (x, y) + (2d_n, d_n), \\ v_w &= (x, y) + (0, d_n), \end{aligned} \tag{6}$$

donde los subíndices denotan los puntos cardinales para indicar cada uno de los laterales.

En estas notas veremos, en primer lugar, el algoritmo del punto medio y a continuación el algoritmo diamante-cuadrado. La diferencia entre ambos es el orden en que calculan los vértices del paso n.

2. El algoritmo del punto medio

El algoritmo del punto medio se basa en el algoritmo del punto medio para 2D en los laterales de los cuadrados nuevos, y luego promedia los centros usando esos cuatro laterales.

Iteración $n = 0$**.** En la iteración $n = 0$ se asigna valor a los vértices de las cuatro esquinas del cuadrado inicial.

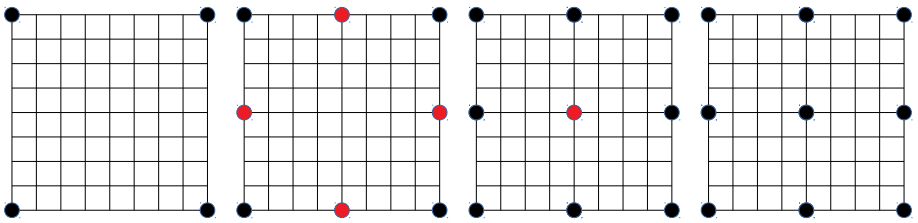
Iteración $n = 1$**.** En la iteración $n = 1$ partimos de un único cuadrado de lado 2^N en el que las 4 esquinas tienen ya un valor asignado. sus coordenadas son $v_1 = (0, 0)$, $v_2 = (0, 2^N)$, $v_3 = (2^N, 0)$ y $v_4(2^N, 2^N)$.

El tamaño de los nuevos cuadrados que van a crearse es $d_1 = 2^{N-1}$. Las coordenadas de los puntos de los lados del cuadrado vienen dadas por $v_s = (d_1, 0)$, $v_w = (0, d_1)$, $v_n = (d_1, 2d_1)$ y $v_e = (2d_1, d_1)$. Los valores para estos puntos se calculan a partir de los puntos de las esquinas v_1, v_2, v_3 y v_4, definidos anteriormente, de la siguiente forma

$$\begin{aligned} h(v_n) &= \frac{h(v_3) + h(v_4)}{2} + z, \\ h(v_s) &= \frac{h(v_1) + h(v_2)}{2} + z, \\ h(v_e) &= \frac{h(v_2) + h(v_4)}{2} + z, \\ h(v_o) &= \frac{h(v_1) + h(v_3)}{2} + z, \end{aligned} \tag{7}$$

donde z es el incremento aleatorio de la altura definido por (2). Una vez calculadas las alturas de los puntos laterales, la altura del punto central se calcula promediando las alturas de éstos

$$h(v_m) = \frac{v_n + v_s + v_e + v_w}{4} + z. \tag{8}$$



Iteración $n = 2$**.** Cada uno de los cuatro cuadros anteriores va a ser dividido en 4. Cada nuevo cuadrado tiene lado $d_2 = 2^{N-2} = 2^{N-n}$. Las coordenadas de su esquina inferior izquierda se obtienen con la ecuación (3), siendo $(x, y) =$

$(i2^{N-1}, j2^{N-1})$, con i y j tomando valores 1, 2, 3 y 4. Los cuatro vértices del cuadrado (i, j) serán

$$\begin{aligned} v_1 &= (x, y), \\ v_2 &= (x, y) + (0, d_1) = (x, y) + (0, 2d_2), \\ v_3 &= (x, y) + (d_1, 0) = (x, y) + (2d_2, 0), \\ v_4 &= (x, y) + (d_1, d_1) = (x, y) + (2d_2, 2d_2), \end{aligned}$$

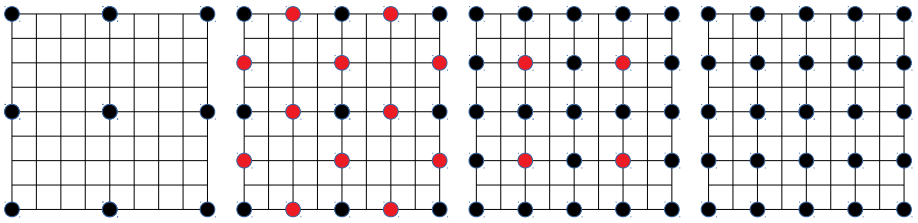
donde se ha utilizado que $d_1 = 2d_2$ para poder expresar todas las coordenadas en función de d_2.

En primer lugar se calculan los cuatro laterales, v_n, v_s, v_e, v_w. Sus coordenadas vienen dadas por las ecuaciones (6), tomando $n = 2$.

$$\begin{aligned} v_n &= (x, y) + (d_2, 2d_2), \\ v_s &= (x, y) + (d_2, 0), \\ v_e &= (x, y) + (2d_2, d_2), \\ v_w &= (x, y) + (0, d_2), \end{aligned}$$

Una vez calculados estos puntos, se calcula el punto central que tiene coordenadas dadas en la ecuación (5)

$$v_m = (x, y) + (d_2, d_2).$$



Una vez tenemos las coordenadas de los 5 puntos nuevos de cada cuadrado, se aplican las mismas fórmulas que en la iteración $n = 1$ para el cálculo de sus alturas. Es importante observar que tras calcular el punto v_e del cuadrado $(0, 0)$, el punto v_w del cuadrado $(1, 0)$ ya tiene altura. Si lo calculáramos de nuevo, el punto medio del cuadrado $(0, 0)$ se habría calculado con un valor distinto del valor definitivo de su vértice v_e, lo que provocaría escalones excesivamente elevados, especialmente en las regiones centrales de la cuadrícula.

Por tanto, debe evitarse calcular de nuevo la altura de los vértices. En general, esto ocurre siempre con dos cuadrados situados uno al lado del otro. Lo mismo ocurre con el punto v_n y el punto v_s de dos cuadrados situados uno sobre el otro.

Para evitar esta duplicidad en los cálculos, se pueden seguir dos estrategias. La primera opción es seguir una implementación puramente iterativa, en la que se calculan en primer lugar todos los vértices laterales, de todos los cuadrados, y a continuación se calculan los vértices v_m de todos los cuadrados.

La alternativa es calcular inicialmente los valores de v_w y v_s de todos los cuadrados con alguna de sus dos coordenadas 0 (calcular primer todos los laterales inferior e izquierdo). Una vez hecho este cálculo, se realiza el cálculo del resto de vértices iterando sobre los cuadrados, calculando los únicamente los vértices v_n y v_e. Si la iteración se realiza en el orden correcto, todos los cuadrados tendrán ya calculado sus vértices v_s y v_w. Esta segunda estrategia tiene la ventaja, como veremos, de admitir una implementación recursiva.

Iteración n. Cuando llegamos a la iteración n, dividimos la rejilla en $2^n \times 2^n$ cuadros. Cada cuadro tiene lado $d_n = 2^{N-n}$ y la esquina inferior izquierda del cuadrado (i, j) tiene por coordenadas $(x, y) = (i2^{N-n}, j2^{N-n}) = (id_n, jd_n)$.

Los vértices de las esquinas del cuadrado (k, l) tienen coordenadas

$$\begin{aligned} v_1 &= (x, y), \\ v_2 &= (x, y) + (0, 2d_n), \\ v_3 &= (x, y) + (2d_n, 0), \\ v_4 &= (x, y) + (2d_n, 2d_n), \end{aligned}$$

donde, de nuevo, hemos usado que $d_{n-1} = 2d_n$. Los centros de los lados serán

$$\begin{aligned} v_n &= (x, y) + (d_n, 2d_n), \\ v_s &= (x, y) + (d_n, 0), \\ v_e &= (x, y) + (2d_n, d_n), \\ v_w &= (x, y) + (0, d_n), \end{aligned}$$

y el vértice del medio

$$v_m = (x, y) + (d_n, d_n),$$

Una vez calculadas las coordenadas de los 5 vértices nuevos asociados al cuadrado (i, j) las alturas desconocidas, $h(v_n)$, $h(v_s)$, $h(v_e)$, $h(v_w)$ y $h(v_m)$, se calculan por medio de las ecuaciones (7) y (8). De nuevo, debemos tener en cuenta que los vértices laterales son compartidos por dos cuadrados, por lo que deberemos seguir alguna de las estrategias indicadas arriba.

2.1. Implementación

A continuación se presentan dos ejemplos de implementación del algoritmo del punto medio. En primer lugar se presenta un algoritmo iterativo en el que en cada iteración se calculan primero los puntos laterales, y posteriormente los puntos medios de todos los cuadrados. A continuación se muestra una implementación recursiva, en la que se realizan los cálculos para un cuadrado.

Implementación iterativa. En el Algoritmo 2 se muestra el código necesario para implementar la versión iterativa del algoritmo del punto medio. Se ha implementado en una única función, que recibe como argumento el valor de N.

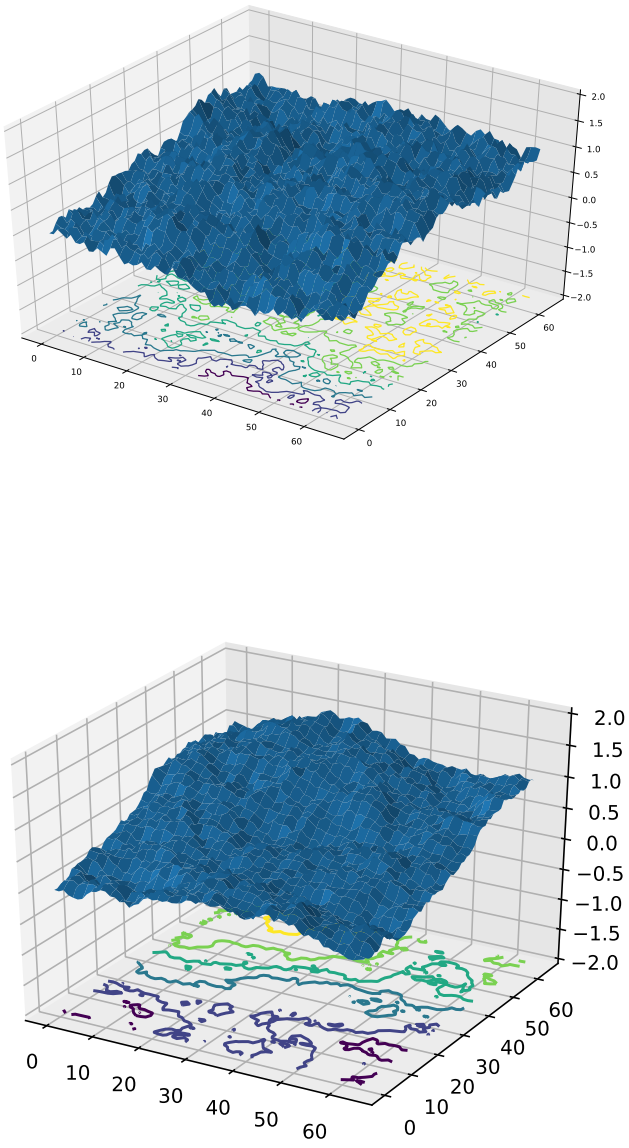


Figura 1: El resultado de la aplicación del algoritmo, con $H = 0{,}35$ (arriba) y con $H = 0{,}55$.

```
def terreno3D(N):
    h = np.zeros( (2**N+1,2**N+1 ) )

    h[0,0] = A*get_rand()
    h[0,-1] = A*get_rand()
    h[-1,0] = A*get_rand()
    h[-1,-1] = A*get_rand()

    for n in range(N):
        d = 2**(N-(n+1))
        n_sq = 2**n
        # Punto lateral norte y sur
        for i in range(n_sq):
            xv = d+2*d*i                # Coordenada x del vertice
            for j in range(n_sq+1):     # +1 para el lado norte del ultimo
                yv = 2*d*j              # Coordenada y del vertice

                R = get_rand()          # Random entre -1 y 1
                z = A*R*2**(-n*H)
                h[xv][yv] = (h[xv-d][yv]+h[xv+d][yv]) * 0.5 + z

        # Punto lateral este y oeste
        for i in range(n_sq+1):         # +1 para el lado este del ultimo
            xv = 2*d*i
            for j in range(n_sq):
                yv = d+2*d*j
                R = get_rand()          # Random entre -1 y 1
                z = A*R*2**(-n*H)
                h[xv][yv] = (h[xv][yv-d]+h[xv][yv+d]) * 0.5 + z

        # Punto central
        for i in range(n_sq):
            xv = d+2*d*i
            for j in range(n_sq):
                yv = d+2*d*j
                R = get_rand()          # Random entre -1 y 1
                z = A*R*2**(-n*H)
                h[xv][yv] = (h[xv][yv-d]+h[xv][yv+d] \
                            +h[xv-d][yv]+h[xv+d][yv] ) * 0.25 + z
    return h
#
```

Algoritmo 2: Versión iterativa del algoritmo del punto medio para generar terrenos 3D

La función tiene un primer bucle, que lleva a cabo las iteraciones que van construyendo los cuadrados cada vez de menor tamaño. En el paso n del bucle se calcula en primer lugar el tamaño de los nuevos cuadrados, `d = 2**(N-n)`, y el número de cuadrados por lado, `n_sq = 2**n`. Dentro de este bucle, encontramos tres bucles que se encargan de calcular todos los puntos: el primer bucle calcula los puntos laterales situados en los lados horizontales de los cuadrados, v_s y v_n; el segundo bucle calcula los puntos situados en los lados verticales de los cuadrados, v_e y v_w; por último, se realiza un tercer bucle que calcula los puntos medios v_m de todos los cuadrados de la iteración n.

Implementación recursiva En el Algoritmo 3 se muestra una implementación recursiva del algoritmo[1]. En este caso, la función principal `terreno3D()`

empieza calculando completos los dos laterales sur y oeste de la rejilla, utilizando la versión 2D del algoritmo del punto medio.

Con estos puntos calculados, el algoritmo recursivo se limita a calcular los puntos norte, este y central de un cuadrado. A continuación, divide el cuadrado que acaba de calcular en cuatro cuadros iguales, y los envía de forma recursiva a la misma función. Si la recursión se hace en el orden correcto, esto hace que al empezar el cálculo de cualquier cuadrado, éste ya tiene calculados sus lados sur y oeste. Esto soluciona además el problema del posible cálculo duplicado de los vértices comunes a dos cuadrados, discutido con anterioridad.

3. Algoritmo Diamante-Cuadrado

El algoritmo del punto medio tiene una limitación. Los vértices de los lados de los cuadrados se calculan utilizando sólamente dos vértices que están en una única dirección. El algoritmo diamante-cuadrado intenta solucionar este problema haciendo que todos los vértices de la malla se calculen siempre usando información de cuatro vértices previos, situados en dos direcciones.

El algoritmo diamante-cuadrado da valores a los vértices interiores de la malla utilizando la misma idea de promediar los valores de los vecinos, añadiendo un desplazamiento aleatorio, usada en el algoritmo del punto medio. Sin embargo, en este caso se calcula en primer lugar el vértice central, v_m de cada cuadrado del paso anterior (paso *diamante*), y a continuación se calculan los vértices laterales usando los vértices centrales de dos cuadros adyacentes (paso *cuadrado*). En la iteración $n = 0$ sinicializan los cuatro vértices del cuadrado principal. A continuación se desarrolla el resto de pasos. El valor de d_n sigue la misma definición que en el algoritmo del punto medio, $d_n = 2^{N-n}$.

Iteración $n = 1$. En el paso diamante, el centro de cada cuadrado se calcula usando los vértices del cuadrado. El paso diamante promedia el punto medio del cuadrado, con los valores de las esquinas. El punto medio, v_m, tiene coordenadas

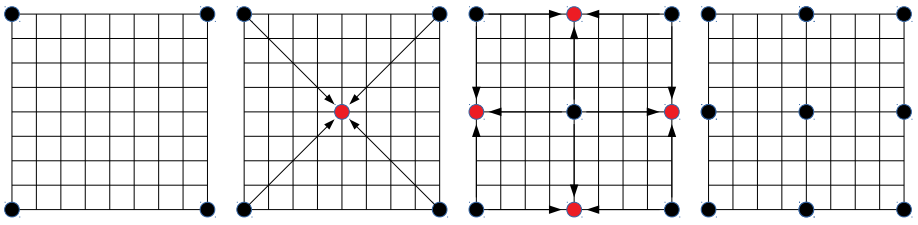
$$v_m = (2^{N-1}, 2^{N-1}) = (d_1, d_1).$$

En el paso cuadrado, los puntos de los lados del cuadrado se calculan a partir de las esquinas y del centro del cuadrado. Las coordenadas de los puntos de los lados del cuadrado vienen dadas por $v_s = (2^{N-1}, 0) = (d_1, 0)$, $v_w = (0, 2^{N-1}) = (0, d_1)$, $v_n = (2^{N-1}, 2^N) = (d_1, 2d_1)$ y $v_e = (2^N, 2^{N-1}) = (2d_1, d_1)$.

[1] La implementación de la versión recursiva es **opcional**. El algoritmo es exactamente el mismo en sus versiones iterativa y recursiva. Por tanto, en la entrega sólo debe presentarse uno de los dos. No se valorará mejor ninguna de las dos opciones ni el hecho de haber implementado ambas.

```
def terreno3D_rec(h,n):
    l = len(h)
    # Base de recursion. Si solo quedan 2, salimos
    if not l%2:
        return

    i_med = l//2

    # norte
    R = get_rand() # Random entre -1 y 1
    z = A*R*2**(-n*H)
    h[-1][i_med] = ( h[-1][0] + h[-1][-1] )*0.5 + z

    # este
    R = get_rand() # Random entre -1 y 1
    z = A*R*2**(-n*H)
    h[i_med][-1] = ( h[0][-1] + h[-1][-1] )*0.5 + z

    # centro
    R = get_rand() # Random entre -1 y 1
    z = A*R*2**(-n*H)
    h[i_med][i_med] = (h[i_med][0]+h[i_med][-1] + \
                                    h[0][i_med]+h[-1][i_med])*0.25 + z

    # Recursion. El orden importa!
    terreno3D_rec(h[0:i_med+1,0:i_med+1],n+1)
    terreno3D_rec(h[i_med:   ,0:i_med+1],n+1)
    terreno3D_rec(h[0:i_med+1,i_med:],n+1)
    terreno3D_rec(h[i_med:,i_med:],n+1)
#


def terreno3D(N):
    h = np.zeros( (2**N+1,2**N+1 ) )
    # sur y oeste. No se calculan en recursion
    h[0,:] = terreno2D(N)
    h[:,0] = terreno2D(N)

    terreno3D_rec(h,1)

    return h

#
```

Algoritmo 3: Versión recursiva del algoritmo del punto medio para generar terrenos 3D. Los laterales sur y oeste del cuadrado principal se calculan previamente, con la versión 2D del punto medio.



Iteración $n = 2$**.** Al empezar el nivel $n = 2$ cada lateral se ha dividido previamente en $2 = 2^{n-1}$. Por tanto, empezamos con $2^n \times 2^n = 4$ cuadrados

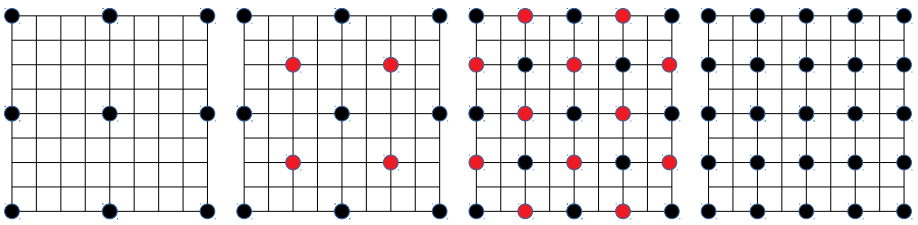
de lado d_1. Si tomamos el cuadrado (i, j), su esquina inferior izquierda tendrá coordenadas $(x, y) = (i2^{N-1}, j2^{N-1})$, y sus cuatro vértices serán $v_1 = (x, y)$, $v_2 = (x, y) + (0, d_1)$, $v_3 = (x, y) + (d_1, 0)$, $v_4 = (x, y) + (d_1, d_1)$.

El paso *diamante* utiliza estos cuatro puntos para calcular el valor del vértice medio, v_m, que tiene por coordenadas

$$v_m = (x, y) + (d_2, d_2) = (x, y) + (2^{N-2}, 2^{N-2}).$$

El paso *diamante* debe completarse para todos los vértices centrales antes de empezar el paso *cuadrado*. El paso *cuadrado* calculará los puntos de los lados de los cuatro cuadrados utilizando los centros y las esquinas de los cuadrados. Las coordenadas de los puntos de los lados que deben ser calculadas en esta iteración son

$$v_s = (x, y) + (d_2, 0),$$
$$v_w = (x, y) + (0, d_2),$$
$$v_n = (x, y) + (d_2, 2d_2),$$
$$v_e = (x, y) + (2d_2, 2d_2).$$



Iteración n. En el paso n, partimos de la división del paso anterior, que dejo la rejilla dividida en $2^{n-1} \times 2^{n-1}$ cuadros. Cada cuadro tiene lado $d_{n-1} = 2^{N-(n-1)}$ y la esquina inferior izquierda del cuadrado (i, j) tiene por coordenadas $(x, y) = (id_{n-1}, jd_{n-1})$, donde i y j van desde 0 hasta $2^{n-1} - 1$. El vértice del medio del cuadrado (i, j) tiene por coordenadas

$$v_m = (x, y) + (d_n, d_n),$$

y los centros de los lados tienen por coordenadas

$$v_s = (x, y) + (d_n, 0),$$
$$v_w = (x, y) + (0, d_n),$$
$$v_n = (x, y) + (d_n, 2d_n),$$
$$v_e = (x, y) + (2d_n, d_n),$$

y los vértices de las esquinas son $v_1, \ldots, v_4$, que vienen dados por las ecuaciones (4).

El paso *diamante* consiste en el cálculo del punto medio a partir de estos últimos puntos:

$$v_m = \frac{v_1 + v_2 + v_3 + v_4}{4} + z \tag{9}$$

donde z es el desplazamiento aleatorio definido por (2). Una vez se ha completado el cálculo de los centros para todos los cuadros se calcula el paso *cuadrado*. El paso *cuadrado* consiste en el cálculo de los 4 puntos laterales, a partir de los puntos de las esquinas y los puntos medios que los rodean.

Como ya ocurría en la iteración $n = 2$, el cálculo de los puntos laterales de cada cuadrado no dependen ahora únicamente de esquinas y centros de un único cuadrado. Por el contrario, en esta ocasión cada punto lateral se obtiene utilizando el punto medio de dos cuadrados vecinos. Esto hace que no sea factible, de una forma sencilla, implementar el cálculo de ambos pasos para un cuadrado cualquiera y realizar iteración por cuadrado, o proceder de forma recursiva. El motivo es que debemos tener completo el paso *diamante* de todos los cuadrados antes de realizar el paso *cuadrado* en cualquiera de ellos.