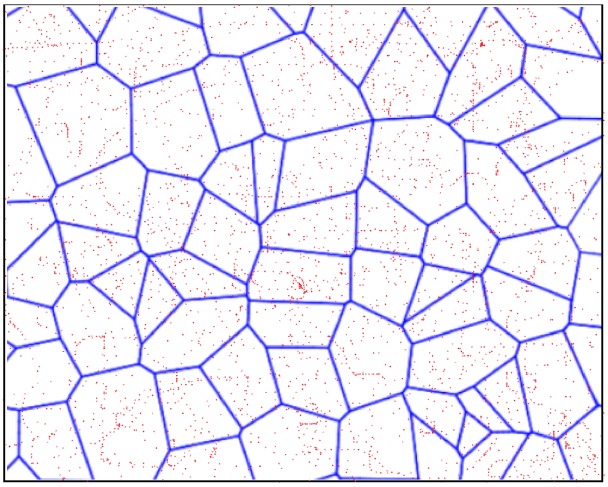

Figure 1: *Representación visual de una malla con nodos de geometría variada.*

Consideremos una caja de de lado L. El algoritmo de maya, primero divide el espacio en $n << N$ regiones que son conocidas como nodos y que pueden tener geometrías variadas como se muestra en la figura (1). Una vez hecho esto, calcula las distancia máxima entre cada una de las regiones empleando el algoritmo de fuerza bruta. Si la distancia entre dos nodos $\mathcal{N}_i$ y $\mathcal{N}_j$, $||\mathcal{N}_i - \mathcal{N}_j||$, $i \neq j$, es mayor que la distancia máxima de interés, las distancias $d(\mathbf{x}, \mathbf{y})$ entre cualesquiera dos puntos $\mathbf{x} \in \mathcal{N}_i$, $\mathbf{y} \in \mathcal{N}_j$ no se encuentra en el rango de interés, por lo que su cálculo resulta innecesario. Nos enfocamos entonces el el cálculo de las distancias entre puntos que pertenezcan a nodos que satisfagan la condición $||\mathcal{N}_i - \mathcal{N}_j|| < d_{max}$. Es necesario destacar que la eficiencia de este algoritmo yace en encontrar un equilibrio entre el número de nodos y el rango de distancias que de nuestro interés. Si son pocos nodos, existirán pocas duplas de nodos que podamos despreciar. También, dado que para calcular la distancia entre nodos empleamos el algoritmo de fuerza bruta, si es del orden del número de puntos y el número de nodos es el mismo, el algoritmo de maya pudiese ser incluso de complejidad mayor al algoritmo de fuerza bruta. En el algoritmo Alg.1, mostramos el algoritmo de maya para n nodos.[1]

En pocas palabras, este es el conocido algoritmo de malla. Dado que aun usamos fuerza bruta para correlacionar los nodos, este algoritmo puede ser aun mejorado utilizando diagramas de árbol o determinando los vecinos a piori.

Nuestra labor se enfocará en lo siguiente.

- Implementar el algoritmo de Maya para una caja bidimensional
- Implementación en Python.
- Implementación en C o C++
- Generalizar el resultado al caso tridimensional.

Podemos pensar en una caja de lado $L = 10$. En esta caja, consideremos particiones uniformes de lado $L/10 = 1$. Entonces resultan 10×10 cajas, las cuales pueden ser etiquetadas por un número del 1 al 100 (o del 0 al 99).$\{\mathcal{N}_1,\ \mathcal{N}_2, \ldots, \mathcal{N}_{100}\}$ tal como vemos en la figura. 2

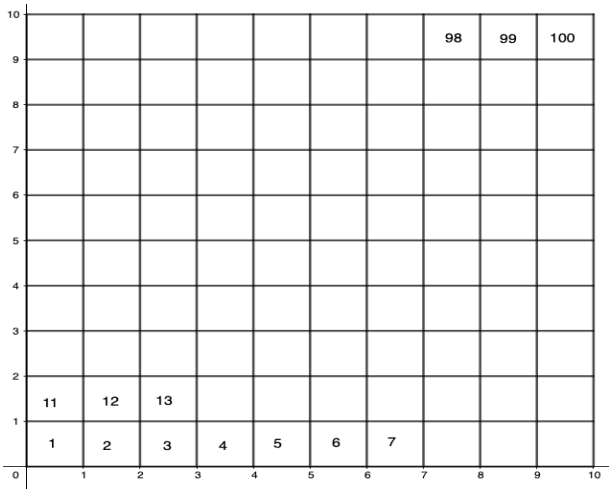

Figure 2: *Etiquetado de nodos.*

Si tenemos en cuenta que la caja puede estar definida por la unión de estos 100 pequeños nodos (los cuales por construcción son disjuntos), podemos asociar cada punto a un (y solo un) nodo.

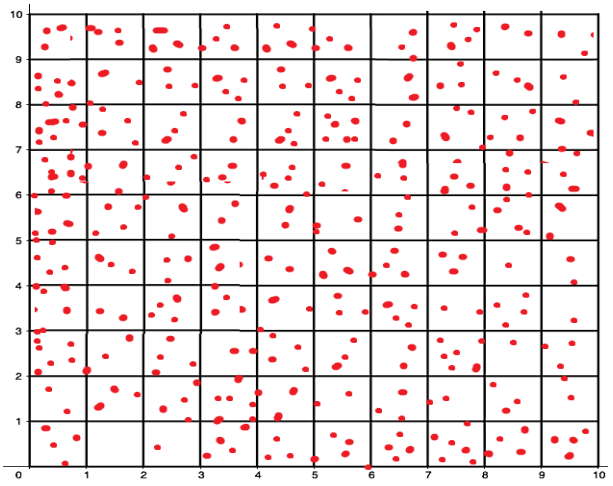

Figure 3: *Cada punto se encuentra asociuado a un único nodo.*

Consideremos un par de nodos como en la figura 4. Si la distancia mínima entre estos dos nodos es mayor que la distancia máxima de interés, entonces podemos asegurar que la distancia entre cada par de puntos (uno de cada nodo) no será de nuestro interés, por lo que su cálculo representa un consumo no deseado de recursos de cómputo. Si estas cajas tienen n_1, n_2 puntos, entonces con el cálculo de una sola distancia (la distancia entre los dos nodos) es posible despreciar el cálculo de $n_1 \times n_2$ distancias. Si por el contrario, la distancia mínima entre los dos nodos es menor que la distancia máxima de interés, estonces en este caso particular requeriremos el cálculo de las distancias entre todos los pares correspondientes.

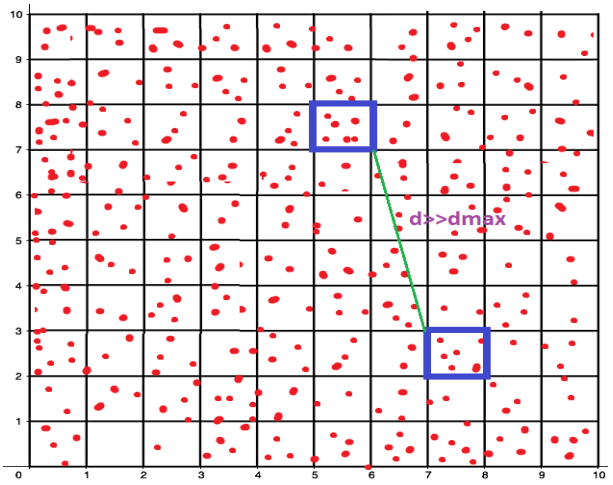

Figure 4: *Cada punto se encuentra asociado a un único nodo.*

De ahora en adelante, diremos que dos nodos $\mathcal{N}_i$, $\mathcal{N}_j$ se encuentran correlacionados si la distancia mínima entre estos $d_{min}(\mathcal{N}_i,\ \mathcal{N}_j) < d_{max}$ es menor a la distancia máxima de interés.
En la figura. 5, ilustramos algunos de los nodos correlacionados con cierto nodo resaltado en color azul.

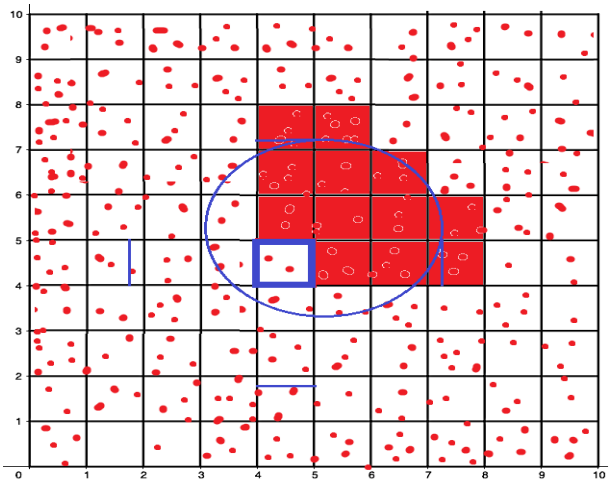

Figure 5: *Cada punto se encuentra asociado a un único nodo.*

A continuación, mostramos un pseudocódigo para determinar la 2PCF isotrópica haciendo uso del algoritmo de mallas

Algorithm 1 Algoritmo de Malla para la 2PCF isotrópica

Require: $[d_{min}, d_{max}]$: rango de distancias requerido.
dl={}: lista vacía para guardar estas distancias.
d: variable para la distancia entre puntos.
D: variable para la distancia máxima entre nodos.
Gl=$\{\mathcal{N}_1, \mathcal{N}_2, \cdots, \mathcal{N}_n\}$: lista de n nodos.

for $1 \leq i \leq n$ **do**
  Aplica algoritmo BFA sobre todos los puntos del nodo ($\mathcal{N}_i$)
end for
for $1 \leq i < n$ **do**
  for $i < j \leq n$ **do**
    $D = \|\mathcal{N}_i - \mathcal{N}_j\|_{min}$
    if $D < d_{max}$ **then**
      for $1 \leq a \leq \text{length}(\mathcal{N}_i)$ **do**
        for $1 \leq b \leq \text{length}(\mathcal{N}_j)$ **do**
          $d = \|x_a^{(i)} - x_b^{(j)}\|$
          if $d < d_{max}$ **then**
            dl.**apend**(d)
          end if
        end for
      end for
    end if
  end for
end for
return dl