UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Química Computacional

Tarea: División de caras de icosaedro

René Rogelio Corrales Millán, 1941587

Profesor: Dr. Alfredo Tlahuice

28 de febrero de 2021

1.Enunciado: Escribir un código que genere divida las aristas de un icosaedro en N partes de tal manera que haya N menos un punto en medio de cada dos vértices que conforman una arista y posteriormente unir cada uno de los puntos con otro en la arista vecina de manera tal que se genere un mallado en cada una de las caras.

Solución

Este problema se puede solucionar básicamente con dos pasos, el primero consiste en identificar las aristas que componen al icosaedro y dividirlas en N partes, guardando para cada arista no solo los vértices que la forman sino también los puntos que forman los segmentos que la dividen.

El segundo paso consiste en comparar las distancias de los nuevos puntos dentro de las aristas con los puntos de las aristas vecinas, y ver si la distancia entre los puntos es equivalente a un múltiplo de la distancia mínima entre cada punto de la arista, de esta manera sabremos que esos dos puntos tienen una relación que debemos tomar en cuenta para crear el mallado dentro de las caras.

Codigo

Primero nos encargamos de leer las coordenadas de nuestro icosaedro y las guardamos en un arreglo llamado “ico” . Después comparamos las distancias entre cada uno de los puntos que están guardados en “ico” para encontrar la distancia mínima y definir cuál será la medida de nuestras aristas. Después definimos N es decir el número de partes en el que vamos a dividir nuestra arista.

```
//Scenario definition
camera {location <12,12,12> look_at <0,0,0>}
background{color White}
light_source{<1000,1000,-1000> color White}
//--------abrimos archivo de entrada---------------
#fopen Myfile_entrada "4-icosaedro.xyz" read

//--------creamos archivo de salida----------------
#fopen Myfile_salida "Coordenadas_dodecaedro.txt" write
//--------------minimo--------------------------
#fopen Myfile_minimo "Minimo_monto.txt" write

//----leemos coordeandas de icosaedro y las guardamos en un arreglo-----
#declare ico=array[12];
//------------bloque para leer coordenadas de icosaedro----------------
#for (i,0,11,1)
  #read (Myfile_entrada, x0,y0,z0)
  #declare ico[i]=<x0,y0,z0>;
  #end
//-----------vamos a calcular la distancia minima entre cada atomo del icosaedro-
#declare minimo=1000;
#for (i,0,11,1)
  #if (VDist(ico[0],ico[i])<minimo)
    #declare minimo=VDist(ico[0],ico[i]);
    #end
  #end
#write(Myfile_minimo, minimo)
```

Fig 1.Codigo para leer, y comparar las distancias de las coordenadas del icosaedro.

Una vez teniendo la distancia minima podemos identificar que puntos están a dicha distancia y guardarlos en un arreglo que nos permita identificar que ese par de puntos generan una arista y al mismo tiempo creamos los puntos que van dentro de cada arista toda esta información la guardamos en un arreglo llamado “Arista” en el cual también incluimos el punto medio de los vértices de cada arista ya que nos ayudará a reducir el número de comparaciones en el paso 3.

```
#while (n<30)
  #declare Arista[n][0]=<1000,10000,1000>;
  #declare Arista[n][1]=<1000,10000,1000>;
  #declare Arista[n][2]=<1000,10000,1000>;

  #declare n=n+1;
  #end
#declare j=0;
#declare s=0;
  #while(j<11)
    #declare i=j+1;
    #while(i<12)
      #if ((VDist(ico[j],ico[i])<(minimo+0.1))&(VDist(ico[j],ico[i])>(minimo-0.1)))
    //----------------------------asegurarnos de que no se repita--------------------
        #declare r=0;
        #declare Pi=0;
        #while (r<30)
          #if(VDist((ico[i]+ico[j])/2,Arista[r][2])<0.1)
            #declare Pi=1;
            #end
          #declare r=r+1;
          #end
        #if (Pi=1)
          #write(Myfile_minimo,"si")
   //-------------------------------------------------------------------------------
        #else
          #declare Arista[s][0]=ico[j];
          #declare Arista[s][1]=ico[i];
          #declare Arista[s][1+N]=(ico[i]+ico[j])/2;
          #declare h=2;
          #while (h<N+1)
            #declare Arista[s][h]=Arista[s][1]+(h-1)*(Arista[s][0]-Arista[s][1])/N;
            #declare h=h+1;
            #end
          #declare s=s+1;
          #end
        #end
        #declare i=i+1;
      #end
      #declare j=j+1;
    #end
```

Fig 2.Codigo para identificar las aristas del icosaedro y dividirlas.

Después de esto generamos un ciclo en donde se evalúa si la distancia entre los puntos de dos aristas es múltiplo de la distancia mínima entre cada una de ellas, por ejemplo si la distancia entre dos puntos de dos aristas es S*(distancia mínima), quiere decir que en el segmento de línea que une a esos dos puntos se habrán de encontrar S-1 puntos los cuales se encuentran en una de las caras del icosaedro, guardamos los puntos en el arreglo “caras”.Para poder llegar a comparar los puntos de dos aristas primero se tiene que cumplir con un condicional que condiciona a que para poder comparar las dos aristas las distancias entre sus puntos medios deben de ser de (medida de una arista)/2 para garantizar que estas sean vecinas y ahorrar cálculos innecesarios.

```
#declare caras=array[10000];
#declare caras2=array[10000];
#declare f=0;
#declare delta=0.001;
#for(i,0,29,1)
  #for(s,i+1,29,1)
    #if (VDist(Arista[i][1+N],Arista[s][1+N])<minimo/2+0.02)
      #for(j,2,N,1)
        #for(h,2,N,1)
          #for(mil,2,N-1,1)
            #if ((VDist(Arista[i][j],Arista[s][h])>mil*minimo/N-delta)&(VDist(Arista[i][j],Arista[s][h])<mil*minimo/N+delta))
              #declare beta=1;
              #while (beta<mil)
                #declare caras[f]=Arista[i][j]+beta*(Arista[s][h]-Arista[i][j])/mil;
                #declare f=f+1;
                #declare beta=beta+1;
                #end
              #end
            #end
          #end
        #end
      #end
    #end
  #end
```

Fig 3.Generamos los puntos de las caras.

A pesar del condicional relacionado con los puntos medios, las coordenadas guardadas salen repetidas, por lo que las guardamos en otro arreglo pero sin que se repitan para después imprimirlas en un archivo .xyz junto con las posiciones de las aristas

```
#declare caras2[0]=caras[0];
#declare contador2=1;
#declare zeta=0;
#for (i,1,f-1,1)
  #declare ata=1;
  #for(s,0,contador2-1,1)
    #if (VDist(caras2[s],caras[i])<delta)
      #declare ata=0;
      #end
    #end
  #if (ata=1)
    #declare caras2[contador2]=caras[i];
    #declare contador2=contador2+1;
    #end
  #end
//-------------------------------------------------------
```

Fig 4.Codigo para guardar solo coordenadas no repetidas.

Las coordenadas obtenidas fueron graficadas en Avogadro, sin embargo, al aumentar el numero de particiones de las aristas a 6 particiones, Avogadro no funciona de manera eficiente en mi equipo por lo que tuve que graficar en un programa llamado VMD.

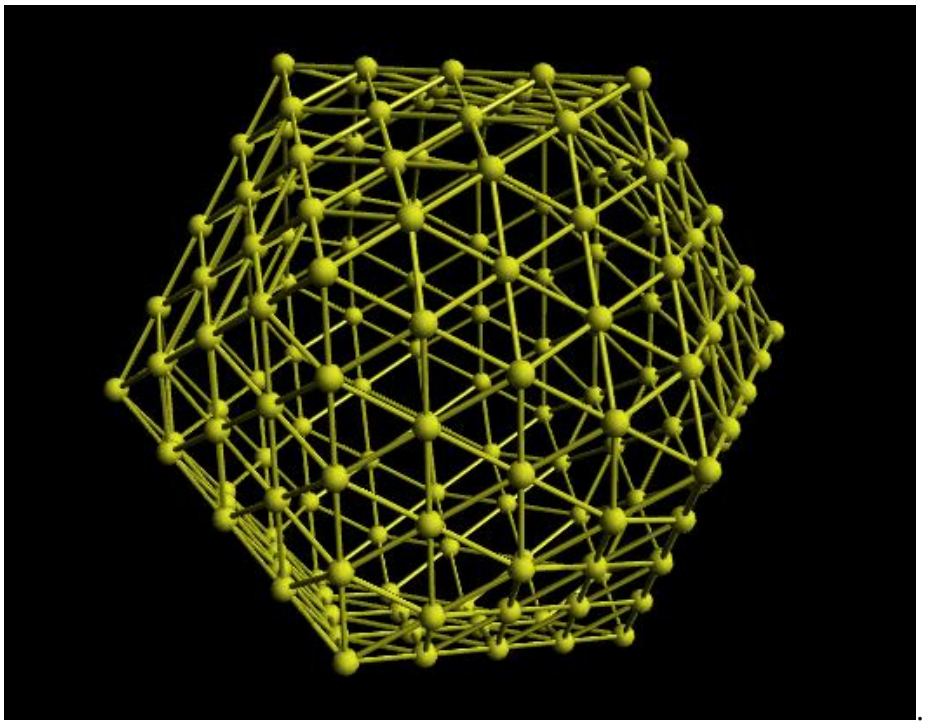

Fig 5.Grafica en Avogadro de una corrida de N=4.

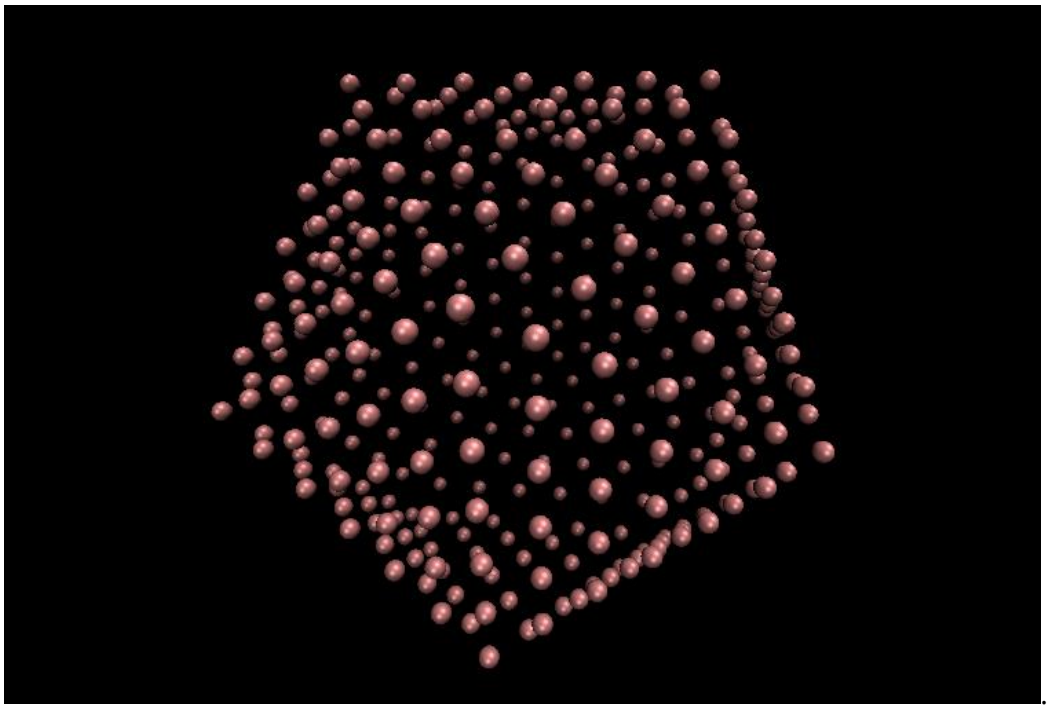

Fig 6.Grafica en VMD de una corrida de N=6.

Conclusiones

Se cumplio el objetivo de la actividad, el código generado calcula las coordenadas para N divisiones de las caras, sin embargo el algoritmo al momento de calcular las coordenadas y guardarlas en un arreglo, estas aparecen repetidas en varias ocasiones, para poder escribir esas coordenadas sin repetirse en un archivo externo primero las debemos eliminar. Esa parte el código cumple con su objetivo, es decir nos quedamos con coordenadas que no se repiten, pero el código es bastante lento en este paso.