Table of Contents

..... 1
Set LIDAR's parameters 1
Generate synthetic sphere 2
Supress "redundant" data 3
Triangulate comun 4
Llenado de espacios 5
Verificamos 16


```
clear;clc;close all
```

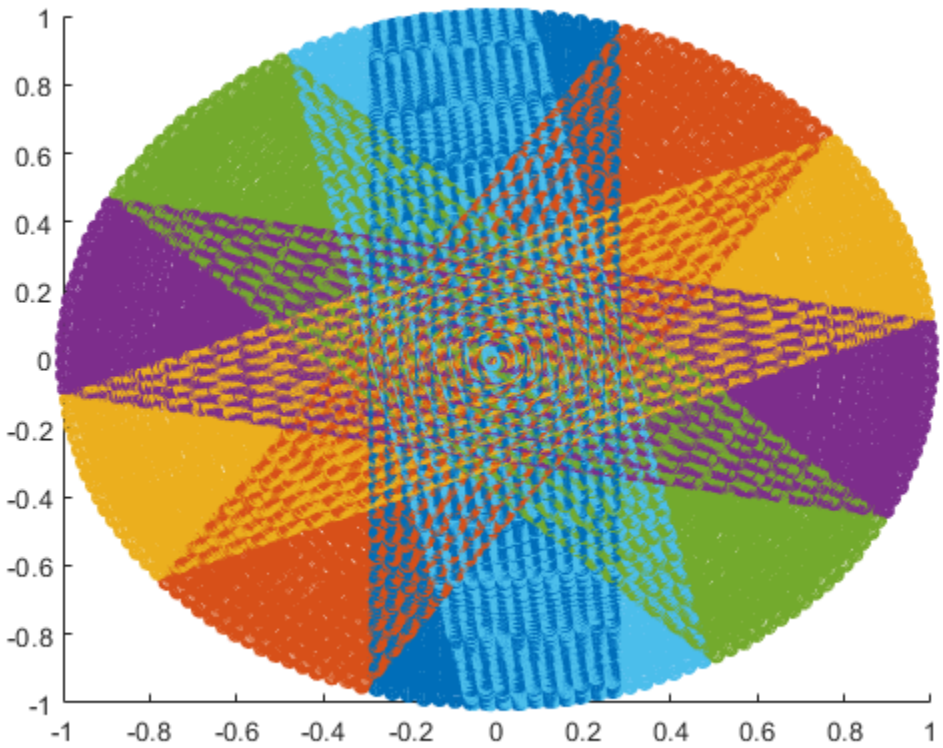
Set LIDAR's parameters

```
tic
%Angle elevation range;
    AZBLK_angle=[15.379 13.236 11.128 9.03 6.941 4.878 2.788 0.705
 -1.454 -3.448 -5.518 -7.601 -9.697 -11.789 -13.914 -16.062]*pi/180;
    n_rays=length(AZBLK_angle);
%Offset angle of AZBLK's points
    %AZBLK_offset=regexprep(num2str(linspace(-1.24,-0.857,16)),'\s
+',',')

 AZBLK_offset=[-1.24,-1.2145,-1.1889,-1.1634,-1.1379,-1.1123,-1.0868,-1.0613,-1.03
%Encoder range: [0,90111] ticks
    total_ticks=90112;
%Mode 1023 azimuth. Every azimuth increments in 88 ticks
    %1 tick = 360/90112= 0.003995°.     Aprox -> 0.004°
    %88 tick = 360*88/90112 = 0.3516°
    n_AZBLK=1024;
    %It should be noted that, we obtain 1024*16 = 16384 points in
 every LIDAR scan
    n_points=n_AZBLK*n_rays;
    ticks_between_azimuths=total_ticks/n_AZBLK;
    angle_between_azimuths=-2*pi*(ticks_between_azimuths/total_ticks);
                          %=-2*pi/n_AZBLK
%Angle rotation,n° donuts,angular velocity,time per scan.
    n_donuts=6;
    rot_angle=-33.53706667*pi/180;
    rot_matrix=[cos(rot_angle) -sin(rot_angle)  0;
                sin(rot_angle) cos(rot_angle)   0;
                0               0               1];
%pre-allocate point clouds
    temp=zeros(n_AZBLK*n_rays,3);
    point_cloud=cell(1,n_donuts);
    for i=1:n_donuts
        point_cloud{i}=temp;
    end
```

Generate synthetic sphere

```
R=1;%radius sphere
for i=1:n_donuts
    for j=1:n_AZBLK
        for k=1:n_rays
            x=R*sin(AZBLK_angle(k));
            z=-
R*cos(AZBLK_angle(k))*cos(angle_between_azimuths*(j-1)+AZBLK_offset(k));

 y=R*cos(AZBLK_angle(k))*sin(angle_between_azimuths*(j-1)+AZBLK_offset(k));
            temp((j-1)*n_rays+k,:)=(rot_matrix^(i-1)*[x;y;z])';
        end
    end
    point_cloud{i}=temp;
end
figure (1)
hold
for i=1:n_donuts
    temp=point_cloud{i};
    scatter3(temp(:,1),temp(:,2),temp(:,3))
end
hold off
```

Current plot held

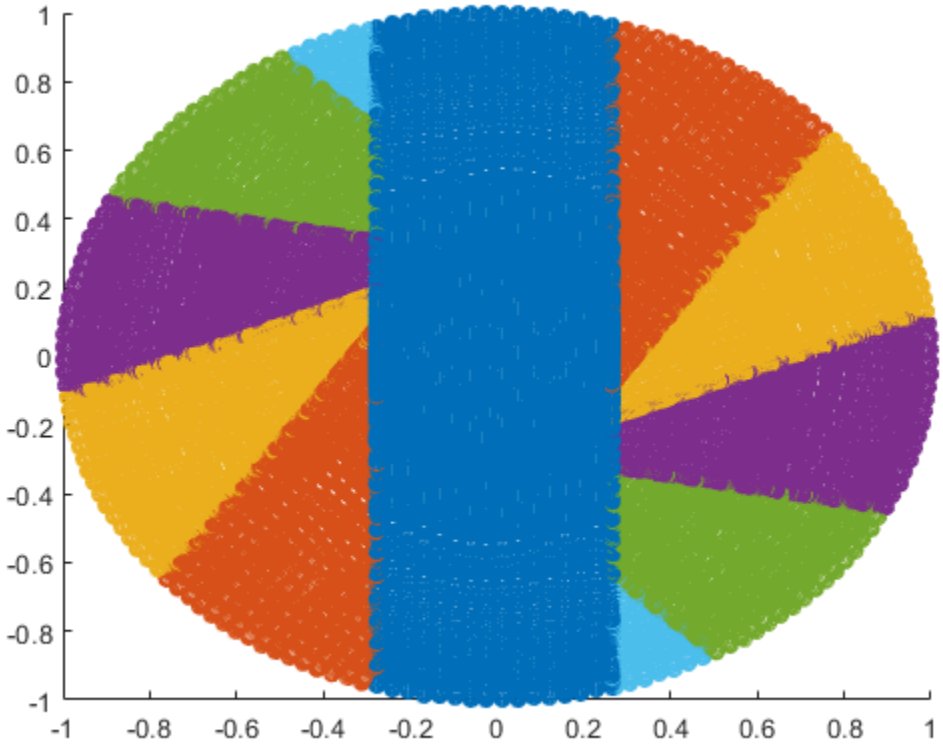


Supress "redundant" data

```
%set parameters

alfa1=AZBLK_angle(1);
alfa2=AZBLK_angle(end);

p1=zeros(n_donuts,3);
p2=p1;
l1_param=[0,0,0];
l2_param=l1_param;
offset=-pi/2;
%Set boundary points of the donuts and every pair of equation lines
for i=1:n_donuts
    angle_temp=rot_angle*(i-1)+alfa1+offset;
    p1(i,:)=[cos(angle_temp),sin(angle_temp),0];
    angle_temp=rot_angle*(i-1)+alfa2+offset;
    p2(i,:)=[cos(angle_temp),sin(angle_temp),0];
    if i~=1
        l1_param(i,:)=[tan(rot_angle*(i-1)+offset),p1(i,1),p1(i,2)];
        l2_param(i,:)=[tan(rot_angle*(i-1)+offset),p2(i,1),p2(i,2)];
    end
end
%supress data
new_point_cloud=point_cloud;

%Donut2
i=2;
for j=1:length(new_point_cloud{i})
    x_data=new_point_cloud{i}(j,1);
    y_data=new_point_cloud{i}(j,2);
    z_data=new_point_cloud{i}(j,3);
    if x_data<=p1(1,1) && x_data>=p2(1,1)
        new_point_cloud{i}(j,:)=[0,0,0];
    end
end
%Donut 3 to 6
for i=3:n_donuts
    for j=1:length(new_point_cloud{i})
        x_data=new_point_cloud{i}(j,1);
        y_data=new_point_cloud{i}(j,2);
        z_data=new_point_cloud{i}(j,3);
        if x_data>p1(1,1)
            y_temp=line_equation(x_data,l1_param(i-1,:));
            if y_data>=y_temp
                new_point_cloud{i}(j,:)=[0,0,0];
            end
        elseif x_data<p2(1,1)
            y_temp=line_equation(x_data,l2_param(i-1,:));
            if y_data<=y_temp
                new_point_cloud{i}(j,:)=[0,0,0];
            end
        else
```

```
            new_point_cloud{i}(j,:)=[0,0,0];
        end
    end
end
%Data a Carlos:
point_cloud_without_overlapped=zeros(n_points*n_donuts,3);

figure(2)
hold
for i=1:n_donuts
    temp=[new_point_cloud{i}];
    scatter3(temp(:,1),temp(:,2),temp(:,3));
    point_cloud_without_overlapped((i-1)*n_points
+1:i*n_points,:)=temp;
end
hold off
```

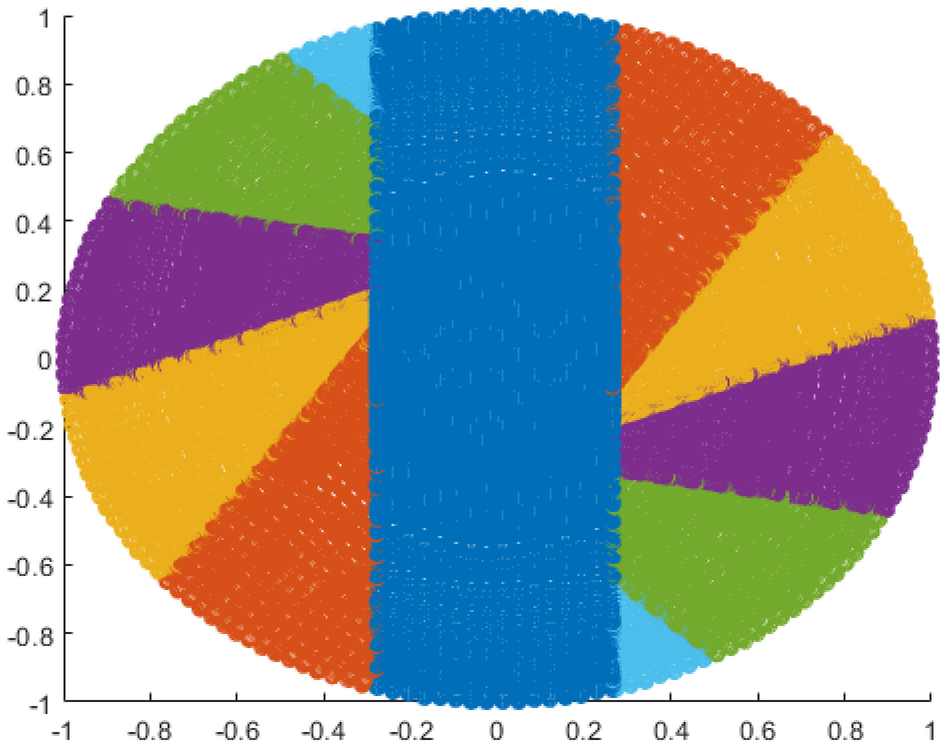
Current plot held



Triangulate comun

```
Triangle_mesh=[];
for i=1:n_donuts
    temp=new_point_cloud{i};
    T=triangulate_v1(temp,n_rays,n_AZBLK);
    %trimesh(T,temp(:,1),temp(:,2),temp(:,3))
```

```
    %Data a Carlos
    Triangle_mesh=[Triangle_mesh;T+(i-1)*n_points];
end
```



Llenado de espacios

%%%%%%%%para jugar%%%%%%%%%%%%%% figure (3) temp=point_cloud_without_overlapped; trimesh(Triangle_mesh,temp(:,1),temp(:,2),temp(:,3)) hold %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%

```
T_mesh_holes=[];
for i=2:n_donuts
    %Deseo conocer, de esta Donut, si existen filas completamente
 eliminadas y
    %los puntos en donde están los limites de cada fila

 boundary_points=get_boundary_points_v2(new_point_cloud{i},n_rays,n_AZBLK);
    Triangle_midle_boundary=zeros(2,3);
    for slot=1:4 %%llenamos los huecos
        %Tenemos dos slot de triangulos, los superiores e inferiores.
 Aunque
        %tambien hay los del lado opuesto (Son, en realidad, 4 slots.
 el
        %tercero será el superior-opuesto y el 4to el inferior-
opuesto)
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%            CREACIÓN DE LOS TRI-PIVOT              %%%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        T=[];
        if slot<=2
            %Al pertenecer a los slot 1 y 2, estarán en lado x<0
            k_ray=16;
            k_offset_azimuth=floor(AZBLK_offset(k_ray)/
angle_between_azimuths);

 minimal_offset=mod(AZBLK_offset(k_ray),angle_between_azimuths);
            for p=1:n_rays
                %obtenemos el v3 que nos ayudará a obtener los v1 y v2
 de cada
                %triángulo; en conjunto con la coordenada. Tener en
 cuenta
                %que es necesario aplicar un offset
                v3_point=boundary_points{p}(slot,:);
                %tenemos que verificar que el triángulo existe (v3~=0)
                if v3_point(1)~=0
                    v3=v3_point(1)+(i-1)*n_points;
                    %hallamos las coordenadas del vertice
                    point=v3_point(2:4);
                    %obtenemos los vértices

 [v1,v2]=get_azimuth_vex_v2(point,i,R,angle_between_azimuths,n_AZBLK,n_rays,rot_ma
                    T=[T;v1,v2,v3];
                end
            end
        else
            k_ray=1;
            k_offset_azimuth=floor(AZBLK_offset(k_ray)/
angle_between_azimuths);

 minimal_offset=mod(AZBLK_offset(k_ray),angle_between_azimuths);
            for p=1:n_rays
                %obtenemos el v3 que nos ayudará a obtener los v1 y v2
 de cada
                %triángulo; en conjunto con la coordenada. Tener en
 cuenta
                %que es necesario aplicar un offset
                v3_point=boundary_points{p}(slot,:);
                %tenemos que verificar que el triángulo existe (v3~=0)
                if v3_point(1)~=0
                    v3=v3_point(1)+(i-1)*n_points;
                    %hallamos las coordenadas del vertice
                    point=v3_point(2:4);
```

```
                    %obtenemos los vértices y definimos el sentido
horario
                    %es por eso que invertimos las salidas

[v1,v2]=get_azimuth_vex_v2(point,i,R,angle_between_azimuths,n_AZBLK,n_rays,rot_ma
                    T=[T;v1,v2,v3];
                end
            end
        end
        %La siguiente variable, permitirán conocer los vértices de los
        %triángulos límites para hacer la unión del medio
        if slot<=2%%%%%%%Debido a la parte opuesta%%%%%
            Triangle_midle_boundary(slot,:)=T(1,:);
            if i==6
                Triangle_midle_boundary(slot+4,:)=T(end,:);
            end
        else
                Triangle_midle_boundary(slot,:)=T(end,:);
            if i==6
                Triangle_midle_boundary(slot+4,:)=T(1,:);
            end
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%     LLENADO DE LAS ZONAS DE LOS TRI-PIVOT     %%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %a continuación, la creación de las mallas triangulares
        for j=1:length(T)-1
            %establecemos variables para identificar los vertices de
 los dos
            %triangulos que contienen el espacio a llenar
            v3_act=T(j,3);
            v2_act=T(j,2);
            v1_act=T(j,1);
            v3_next=T(j+1,3);
            v2_next=T(j+1,2);
            v1_next=T(j+1,1);
            %definimos los parámetros según el modo y tipo
            if isequal(point_cloud_without_overlapped(v3_act-
n_rays,:),[0 0 0])
                %Escalera_tipo_1 o Rampa_tipo_2
                if slot<=2%%%%%%%Debido a la parte opuesta%%%%%
                    v_init=v2_next;
                    v_fin=v1_act;
                else
                    v_fin=v2_next;
                    v_init=v1_act;
                end
```

```
                    if v3_act+1>=v3_next
                        %MODO ESCALERA TIPO 1
                        %Obtenemos los puntos libres de L1
                        common_vex=v3_act+1;
                        free_points_L1=(common_vex-v3_next)/16;
                        %creamos el triangulo con mismo vertice
                        v1=v1_act;
                        v2=v3_act;
                        v3=common_vex;
                        %esta variable servirá para cambiar el sentido del
llenado
                        %ya que el v_init sería el v_fin, coon respecto al
common_vex
                        escalera=boolean(1);
                        tipo=1;
                    else
                        %MODO RAMPA TIPO 2
                        %Obtenemos los puntos libres de L1
                        common_vex=v3_next-17;
                        free_points_L1=(common_vex-v3_act)/16;
                        %creamos el triangulo con mismo vertice
                        v1=v3_next;
                        v2=v2_next;
                        v3=common_vex;
                        escalera=boolean(0);
                        tipo=2;
                    end
                else
                    %Escalera_tipo_2 o Rampa_tipo_1
                    if slot<=2%%%%%%%Debido a la parte opuesta%%%%%
                        v_init=v2_act;
                        v_fin=v1_next;
                    else
                        v_fin=v2_act;
                        v_init=v1_next;
                    end
                    if v3_act>=v3_next-1
                        %MODO ESCALERA TIPO 2
                        %Obtenemos los puntos libres de L1
                        common_vex=v3_next-1;
                        free_points_L1=(v3_act-common_vex)/16;
                        %creamos el triangulo con mismo vertice
                        v1=v1_next;
                        v2=v3_next;
                        v3=common_vex;
                        escalera=boolean(1);
                        tipo=2;
                    else
                        %MODO RAMPA TIPO 1
                        %Obtenemos los puntos libres de L1
                        common_vex=v3_act+17;
                        free_points_L1=(v3_next-common_vex)/16;
                        %creamos el triangulo con mismo vertice
                        v1=v3_act;
```

```
                    v2=v2_act;
                    v3=common_vex;
                    escalera=boolean(0);
                    tipo=1;
                end
            end
            %La variable escalera nos ayudará a definir si es escalera
 o rampa
            %Luego de tener el modo y tipo, debemos ver si los
 vertices
            %correspondientes a sus triangulos están en distintas
 Donuts, sino, se
            %debe hacer un distinto llenado
            if (v_init-n_points)*(v_fin-n_points)>=0 %pertenecen a la
 misma Donut
                %hallamos el numero de la Donut en la cual están los
 puntos
                Donut_L2=ceil(v_init/n_points);
                if v_init>v_fin
                    %si la Donut está concatenada, le restamos el
 offset
                    free_points_L2=(v_fin-(v_init-n_points))/16;
                else
                    free_points_L2=(v_fin-v_init)/16;
                end
                %Tambien debemos verificar que free_point_L2 sea mayor
 que los de L1
                if free_points_L1 <= free_points_L2
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %%%%%%%     L1 <= L2    %%%%%%%
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %Ahora hacemos el llenado que más se repite
                    %El triangulo con el common_vex ya fue creado al
 definir el modo y tipo
                    T=[T;v1,v2,v3];
                    aristas_mismo_vertice=free_points_L2-
free_points_L1;
                    if escalera
                        %si es ESCALERA el llenado es restando
                        %A continuación, el llenado de los triangulos
 con misma arista
                        for k=1:aristas_mismo_vertice
                            v2=v1;
                            if slot<=2%%%%%%%Debido a la parte opuesta
%%%%%
                                v1=v1-n_rays;
                                %notar que v1 puede ser negativo, o
 estar
                                %fuera de los puntos de la Donut
                                if v1<=(Donut_L2-1)*n_points
                                    v1=v1+n_points;
                                end
                            else
                                v1=v1+n_rays;
```

```
                                %notar que v1 puede estar fuera de los
                                %puntos de la Donut
                                if v1>Donut_L2*n_points
                                    v1=v1-n_points;
                                end
                            end
                            T=[T;v1,v2,v3];
                        end
                        %Ahora, el llenado que une los puntos libres
 de L1 y L2
                        for k=1:free_points_L1
                            %Primer triangulo
                            v2=v1;
                            if slot<=2%%%%%%%Debido a la parte opuesta
%%%%%
                                v1=v1-n_rays;
                                %notar que v1 puede ser negativo, o
 estar
                                %fuera de los puntos de la Donut
                                if v1<=(Donut_L2-1)*n_points
                                    v1=v1+n_points;
                                end
                            else
                                v1=v1+n_rays;
                                %notar que v1 puede estar fuera de los
                                %puntos de la Donut
                                if v1>Donut_L2*n_points
                                    v1=v1-n_points;
                                end
                            end
                            T=[T;v1,v2,v3];
                            %Segundo triangulo y dependiendo del tipo,
 se suma o resta
                            v2=v3;
                            if tipo==1
                                v3=v3-n_rays;
                                %para este caso no analizamos si se
 excede
                                %ya que las Donut no estarán
 concantenadas
                                %nunca
                            else
                                v3=v3+n_rays;
                                %para este caso no analizamos si se
 excede
                                %ya que las Donut no estarán
 concantenadas
                                %nunca
                            end
                            T=[T;v1,v2,v3];
                        end
                    else
                        %El llenado del modo RAMPA
                        %A continuación, el llenado con el common_vex
```

```
                        for k=1:aristas_mismo_vertice
                            v1=v2;
                            if slot<=2%%%%%%%Debido a la parte opuesta
    %%%%%
                                v2=v1+n_rays;
                                %notar que v2 puede estar fuera de los
                                %puntos de la Donut
                                if v2>(Donut_L2)*n_points
                                    v2=v2-n_points;
                                end
                            else
                                v2=v1-n_rays;
                                %notar que v2 puede ser negativo, o
     estar
                                %fuera de los puntos de la Donut
                                if v2<=(Donut_L2-1)*n_points
                                    v2=v2+n_points;
                                end
                            end
                            T=[T;v1,v2,v3];
                        end
                        %Procedemos, al llenado de los triángulo
     rectángulos
                        for k=1:free_points_L1
                            %Primer triangulo
                            v1=v2;
                            if tipo==1
                                v2=v3+n_rays;
                            else
                                v2=v3-n_rays;
                            end
                            T=[T;v1,v2,v3];
                            %Segundo triangulo
                            v3=v2;
                            if slot <=2%%%%%%%Debido a la parte
     opuesta%%%%%
                                v2=v1+n_rays;
                                %notar que v2 puede estar fuera de los
                                %puntos de la Donut
                                if v2>(Donut_L2)*n_points
                                    v2=v2-n_points;
                                end
                            else
                                v2=v1-n_rays;
                                %notar que v2 puede ser negativo, o
     estar
                                %fuera de los puntos de la Donut
                                if v2<=(Donut_L2-1)*n_points
                                    v2=v2+n_points;
                                end
                            end
                            T=[T;v1,v2,v3];
                        end
                    end
```

```
                else
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %%%%%%%     L1 > L2    %%%%%%%
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %En caso los puntos de L1 son mayores a los de L2.
                    %El llenado es análogo al otro caso, solo que con
 otro
                    %vértice en común
                    error("hacer codigo para L1<=L2")
                end
            else
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%  TRI-DONUT FILL %%%%%%%
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%Primer Triangulo->T(j,:)%%%%
                T1=T(j,:);
%               v3_act=T(j,3);
%               v2_act=T(j,2);
%               v1_act=T(j,1);
                %%%%Tercer Triángulo->T(j+1,:)%%%%
                T3=T(j+1,:);
%                     v3_next=T(j+1,3);
%                     v2_next=T(j+1,2);
%                     v1_next=T(j+1,1);
                %%%%Segundo Triángulo%%%%
                slot_bits=de2bi(slot-1);
                slot_bits(3)=0;
                v3_mid=T(j
+bitshift(slot-1,-1),xor(slot_bits(1),slot_bits(2))+1);
                v3_mid=v3_mid+(-1)^slot*n_rays;
                free_points_L2=1;
                point=point_cloud_without_overlapped(v3_mid,:);
                while(~isequal(point,[0 0 0]))
                    %Hallamos el vertice v3 del triángulo del medio
                    v3_mid=v3_mid+(-1)^slot*n_rays;
                    free_points_L2=free_points_L2+1;
                    point=point_cloud_without_overlapped(v3_mid,:);
                    %No consideramos el caso que se tenga que contenar
 la
                    %Donut
                end
                v3_mid=v3_mid-(-1)^slot*n_rays;%tenemos que volver al
 vértice
                                                %anterior, el que es
 distinto de cero
                free_points_L2=free_points_L2-1;%Restamos debido al
 exceso
                point=point_cloud_without_overlapped(v3_mid,:);
                %obtenemos los vértices, notar que este triángulo
 pivot
                %pertenece a la Donut anterior!!!!

[v1_mid,v2_mid]=get_azimuth_vex_v2(point,i-1,R,angle_between_azimuths,n_AZBLK,n_r
                T2=[v1_mid,v2_mid,v3_mid];
```

```
                %free_points_L2 fue definido al hallar el 2doTriangulo
                %free_points_L1 ya fue calculado al definir el modo y
 tipo
                switch slot
                    case 1 %slot 1
                        free_points_mid=(v1_mid-v2_next)/n_rays;
                    case 2 %slot 2
                        free_points_mid=(v1_next-v2_mid)/n_rays;
                    case 3 %slot 3
                        free_points_mid=(v2_mid-v1_act)/n_rays;
                    otherwise %slot 4
                        free_points_mid=(v2_act-v1_mid)/n_rays;
                end
                if free_points_mid<0
                    free_points_mid=free_points_mid+n_AZBLK;
                end
                %fprintf('\n')
                [free_points_L2 free_points_mid free_points_L1]
                %display(escalera)
                %display(tipo)
                %El triangulo con el common_vex ya fue creado al
 definir el modo y tipo
                T=[T;v1,v2,v3];
                Desplazador_L1=(-1)^slot*n_rays;

 Desplazador_Laux=(-1)^(bitshift(slot-1,-1)+escalera)*n_rays;
                %considerando que no importa el orden de los vertices,
 se
                %tiene que:
                %va:es el vértice de la Donut actual
                va=v3;
                %vb: es el vértice que da inicio a la Línea Auxiliar
                vb=T(j+1-xor(escalera,slot_bits(1)),2-escalera);
                point_L1=point_cloud_without_overlapped(va
+Desplazador_L1,:);
                point_Laux=point_cloud_without_overlapped(vb
+Desplazador_Laux,:);
                while (~isequal(point_L1,[0 0 0])&&
 ~isequal(point_Laux,[0 0 0]))
                    %primer Triangulo
                    v_temp=vb+Desplazador_Laux;
                    %verificamos que v_temp esté dentro del rango
                    T=[T;va vb v_temp];
                    %segundo Triángulo
                    vb=v_temp;
                    v_temp=va+Desplazador_L1;
                    %verificamos que v_temp esté dentro del rango
                    T=[T;va vb v_temp];
                    va=v_temp;
                    point_L1=point_cloud_without_overlapped(va
+Desplazador_L1,:);
                    point_Laux=point_cloud_without_overlapped(vb
+Desplazador_Laux,:);
                end
```

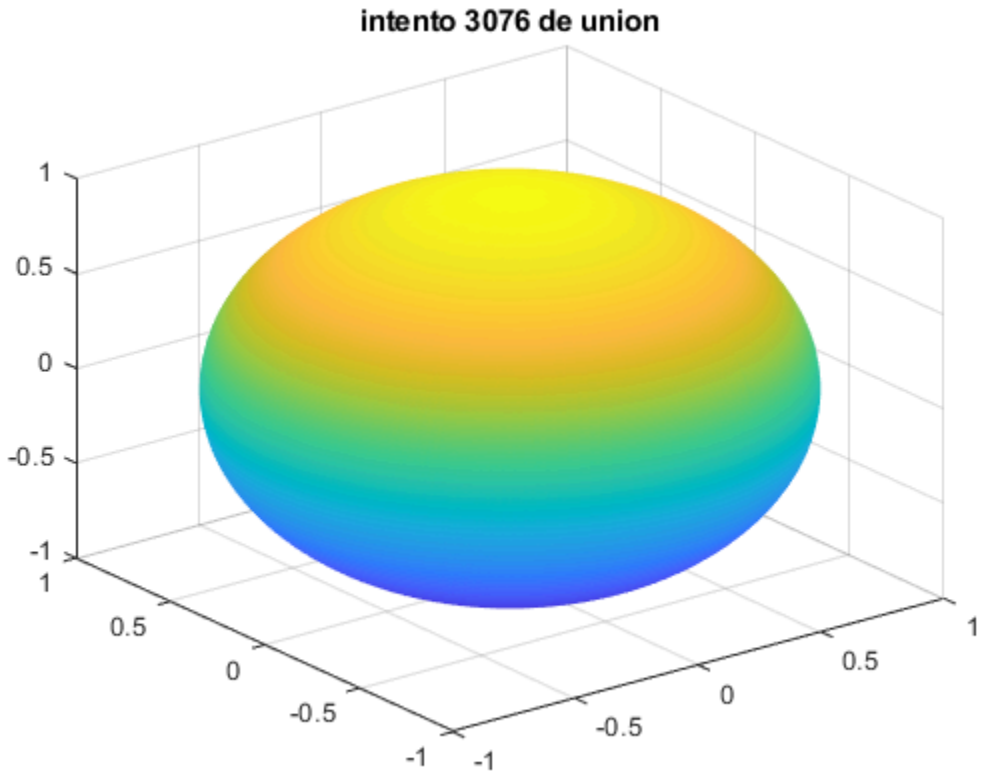
```
%CONTINUAR AQUI %


            end
        end
%         %%%%%%%%%% para jugar %%%%%%%%%%
%         pause(0.5)
%         temp=point_cloud_without_overlapped;
%         trimesh(T,temp(:,1),temp(:,2),temp(:,3))
%         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        T_mesh_holes=[T_mesh_holes;T];
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
    %%%%%%%%%%             LLENADO DE LA MIDDLE ZONE           %%%%%%%%
%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
    T=[];
    for j=1:2
        free_points_L2=(Triangle_midle_boundary(j*2,1)-
Triangle_midle_boundary(j*2-1,2))/16;
        free_points_L1=(Triangle_midle_boundary(j*2,3)-
Triangle_midle_boundary(j*2-1,3))/16;
        %[freepoints_L1 freepoints_L2]
        if free_points_L1 ~=free_points_L2
            error("falta hacer codigo")
        end
        v2=Triangle_midle_boundary(j*2-1,2);
        v3=Triangle_midle_boundary(j*2-1,3);
        for k=1:free_points_L1
            %Primer triángulo
            v1=v2;
            v2=v3+n_rays;
            %No consideraremos que existe la necesidad de concatenar
 la Donut
            T=[T; v1 v2 v3];
            %Segundo triángulo
            v3=v2;
            v2=v1+n_rays;
            T=[T; v1 v2 v3];
        end
    end
    T_mesh_holes=[T_mesh_holes;T];
    T=[];
    if i==6
        for j=3:4
            free_points_L2=(Triangle_midle_boundary(j*2-1,1)-
Triangle_midle_boundary(j*2,2))/16;
```

```
            free_points_L1=(Triangle_midle_boundary(j*2,3)-
Triangle_midle_boundary(j*2-1,3))/16;
            %[freepoints_L1 freepoints_L2]
            if free_points_L1 <=free_points_L2
                %vértice en común
                aristas_mismo_vertice=free_points_L2-free_points_L1;
                v3=Triangle_midle_boundary(j*2,3);%[slot2,v3]
                v2=Triangle_midle_boundary(j*2,2);
                %triángulos mismo vértices
                for k=1:aristas_mismo_vertice
                    v1=v2;
                    v2=v2+n_rays;
                    T=[T; v1 v2 v3];
                end
                %trángulos rectangulares
                for k=1:free_points_L1
                    %Primer triángulo
                    v1=v2;
                    v2=v3-n_rays;
                    %No consideraremos que existe la necesidad de
 concatenar la Donut
                    T=[T; v1 v2 v3];
                    %Segundo triángulo
                    v3=v2;
                    v2=v1+n_rays;
                    T=[T; v1 v2 v3];
                end
            else
                error('Hacer código :(');
            end
        end
    end
    T_mesh_holes=[T_mesh_holes;T];
end

ans =
     5     3     5
ans =
     4     3     4
ans =
     6     1     5
ans =
     5     2     6
ans =
    10     0     8
ans =
    10     0     8
ans =
     0     5     0
ans =
     0     5     0
ans =
     6     3     3
ans =
```

```
     6     2     3
ans =
    10     1     7
ans =
    10     1     8
ans =
    11    10     0
ans =
    11    10     1
ans =
     3    33    30
ans =
     2    33    29
```

Verificamos

```
figure (3)
temp=point_cloud_without_overlapped;
trimesh([Triangle_mesh;T_mesh_holes;T],temp(:,1),temp(:,2),temp(:,3))
title('intento 3076 de union')
```



Published with MATLAB® R2021a