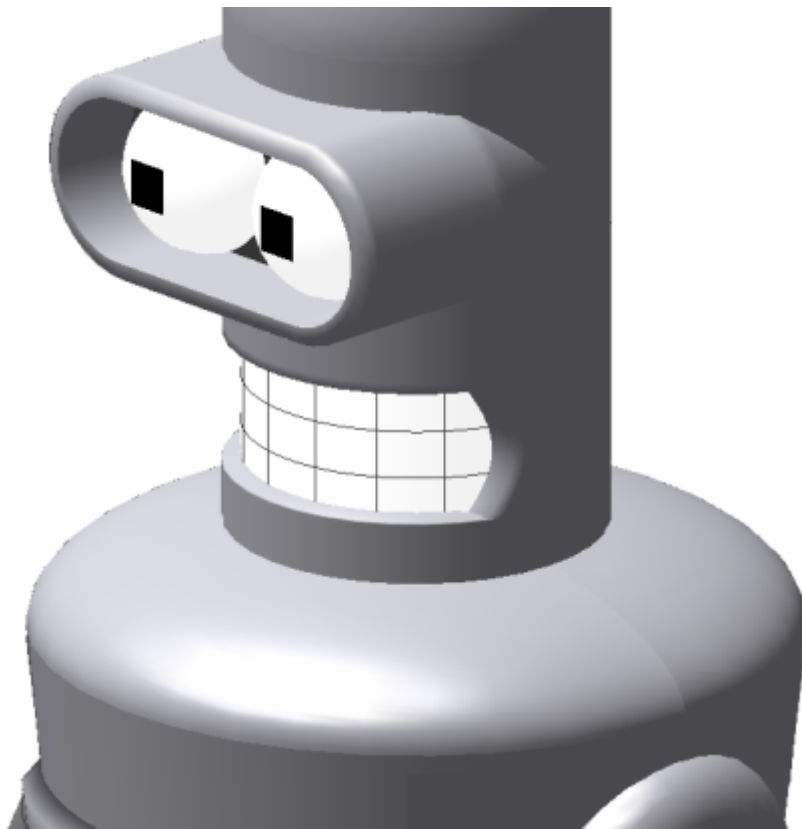


# Geometría Computacional

# Práctica Final



José Barranquero Tolosa  
72047582-J  
Junio 2006

# TABLA DE CONTENIDOS

TABLA DE CONTENIDOS .....	1
INTRODUCCION .....	3
1. MODELADO DE LAS EXTREMIDADES INFERIORES .....	4
1.1. Pie izquierdo .....	4
1.2. Pierna izquierda .....	5
1.3. Simetrías ortogonales .....	6
2. MODELADO DEL CUERPO .....	7
2.1. Carcasa inferior .....	7
2.2. Cuerpo Inferior .....	8
2.3. Cuerpo Superior .....	9
2.4. Puerta .....	10
3. MODELADO DE LAS EXTREMIDADES SUPERIORES .....	11
3.1. Hombro izquierdo .....	11
3.2. Brazo izquierdo .....	12
3.3. Mano izquierda .....	13
3.4. Dedos de la mano izquierda .....	14
3.5. Simetrías ortogonales .....	15
4. MODELADO DE LA CABEZA .....	16
4.1. Cuello .....	16
4.2. Nuca .....	17
4.3. Dientes .....	18
4.4. Cabeza .....	19
4.5. Antena .....	20
4.6. Visera .....	21
4.7. Fondo de la visera .....	22
4.8. Ojo izquierdo .....	23
4.9. Simetrías ortogonales .....	24
5. MODELO 3D COMPLETO .....	25
5.1. Galería de imágenes .....	25
5.2. Fichero principal .....	26

ANEXO .....	27
aplica2.m .....	27
aplicap.m .....	27
dibujar.m.....	27
dibujarDientes.m .....	28
dibujarExtremidad.m .....	28
dibujarLinea.m.....	29
dibujarMano.m .....	29
dibujarMaterialPalido.m .....	30
escalado3D.m .....	30
getPuntosAntena.m.....	31
getPuntosBrazo.m.....	32
getPuntosCabeza.m .....	32
getPuntosCilindro.m.....	33
getPuntosCilindroRebajado.m.....	33
getPuntosConoCortado.m.....	34
getPuntosConoRebajado.m .....	34
getPuntosCuerpoInferior.m .....	35
getPuntosCuerpoSuperior.m.....	36
getPuntosDedo.m.....	37
getPuntosDientes.m .....	37
getPuntosEsfera.m .....	38
getPuntosHombro.m .....	38
getPuntosMandibula.m .....	39
getPuntosNuca.m.....	40
getPuntosOjo.m .....	41
getPuntosPuerta.m .....	42
getPuntosPupila.m .....	43
getPuntosSemiEsfera.m.....	43
getPuntosSemiEsferaAchatada.m.....	44
getPuntosSemiEsferaCerrada.m .....	44
getPuntosVisera.m.....	45
simetriaOtogonal3D.m .....	46
traslacion3D.m.....	46
unirSuperficies.m.....	47
xRotacion3D.m.....	48
yRotacion3D.m.....	48
zRotacion3D.m.....	48
 BIBLIOGRAFIA .....	 49

## INTRODUCCION

Esta memoria tratará de comentar brevemente las técnicas empleadas en el modelado de una figura en 3D mediante el programa de tratamiento de matrices MATLAB.

Con el fin de no alargar demasiado la extensión de la misma y que resulte más llevadera su lectura, el autor parte del supuesto de que el lector tiene unos conocimientos mínimos sobre el tema y por tanto sólo se explicarán aquellos detalles más relevantes del diseño sin profundizar excesivamente en las bases geométricas de cada una de la piezas que forman el objeto final.

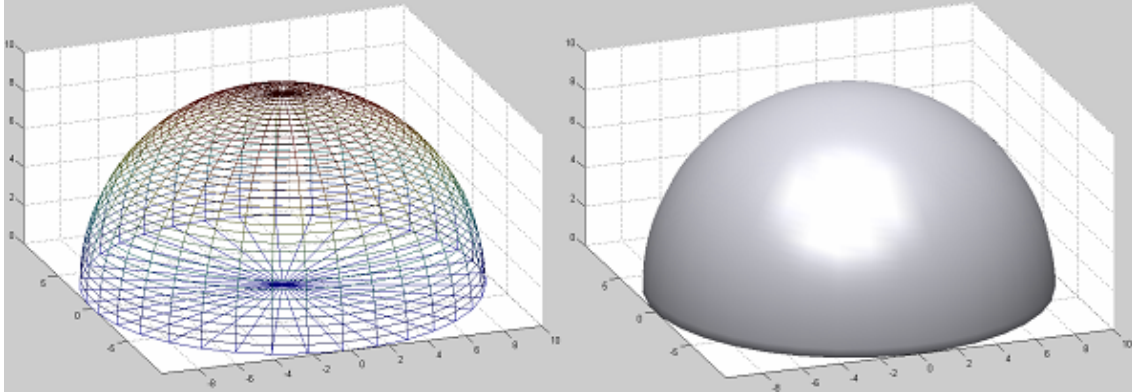
En el caso de que se haga necesaria una visión más técnica también se aporta un pequeño código fuente donde se muestra qué funciones son empleadas en para cada una de las piezas y de este modo se podría acceder directamente a los ficheros fuente donde se explican todos estos aspectos con mayor detalle.

El código fuente que se muestra en este documento está extraído del fichero 'bender.m', el cual se emplea para generar el modelo 3D completo desde el intérprete de órdenes de MATLAB.

# 1. MODELADO DE LAS EXTREMIDADES INFERIORES

## 1.1. Pie izquierdo

### 1.1.1. Imágenes



### 1.1.2. Código fuente

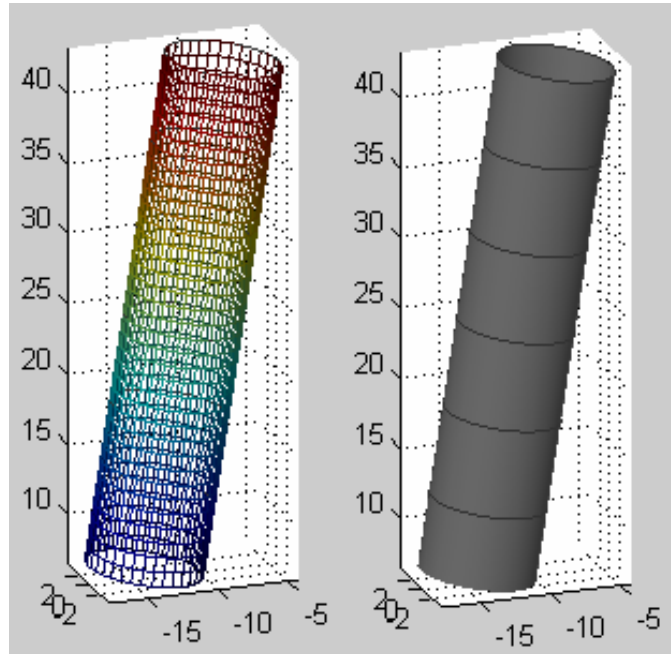
```
% posicion relativa del pie izquierdo respecto al origen
tPieIzq = traslacion3D(-15, 0, 0);
[xPieIzq yPieIzq zPieIzq] = getPuntosSemiEsferaCerrada(10, n);
[xPieIzq yPieIzq zPieIzq] = aplica2(tPieIzq, xPieIzq, yPieIzq, zPieIzq);
dibujar(xPieIzq, yPieIzq, zPieIzq, colorCarcasa);
```

### 1.1.3. Comentarios

Se genera el pie a partir de la función que calcula los puntos de una semiesfera cerrada y se traslada 15 puntos a la izquierda desde el origen.

## 1.2. Pierna izquierda

### 1.2.1. Imágenes



### 1.2.2. Código fuente

```
% posicion relativa de la pierna izquierda respecto al origen
tPiernaIzq = tPieIzq * yRotacion3D(pi/20);
[xPiernaIzq yPiernaIzq zPiernaIzq] = getPuntosCilindro(4, 7, 43, n);
[xPiernaIzq yPiernaIzq zPiernaIzq] = aplica2(tPiernaIzq,
                                             xPiernaIzq,
                                             yPiernaIzq,
                                             zPiernaIzq);
dibujarExtremidad(xPiernaIzq, yPiernaIzq, zPiernaIzq,
                  colorExtremidades, colorLineas);
```

### 1.2.3. Comentarios

La pierna izquierda se genera a partir de un cilindro de radio 4, altura inicial 7 y altura final 43, centrado en el origen. Luego se le gira 'pi/20' radianes respecto al eje Y, desplazándolo la misma distancia hacia la izquierda que al pie correspondiente.

Las secciones se simulan pintando las líneas que recorren las circunferencias que generan el cilindro en intervalos preestablecidos.

## 1.3. Simetrías ortogonales

### 1.3.1. Código fuente

```
% Pie Derecho
[xPieDch yPieDch zPieDch] = aplica2(s, xPieIzq, yPieIzq, zPieIzq);
dibujar(xPieDch, yPieDch, zPieDch, colorCarcasa);

% Pierna Derecha
[xPiernaDch yPiernaDch zPiernaDch] = aplica2(s, xPiernaIzq,
                                              yPiernaIzq, zPiernaIzq);
dibujarExtremidad(xPiernaDch, yPiernaDch, zPiernaDch,
                  colorExtremidades, colorLineas);
```

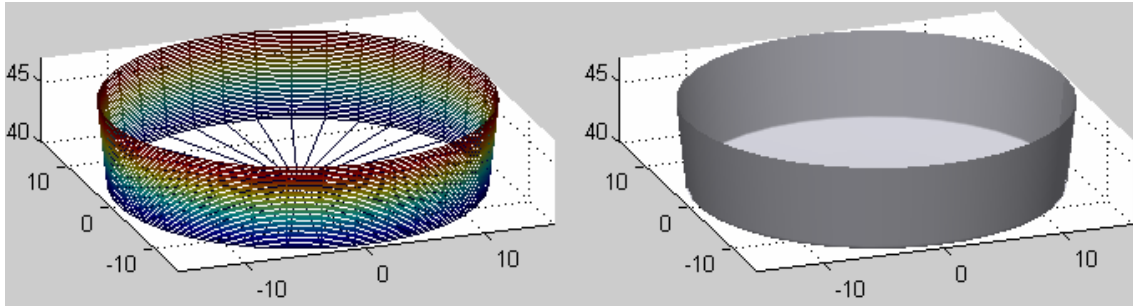
### 1.3.2. Comentarios

Para modelar el pie y la pierna derecha recurrimos a una simetría ortogonal respecto al plano OX, ya que la figura está premeditadamente centrada en el origen de coordenadas.

## 2. MODELADO DEL CUERPO

### 2.1. Carcasa inferior

#### 2.1.1. Imágenes



#### 2.1.2. Código fuente

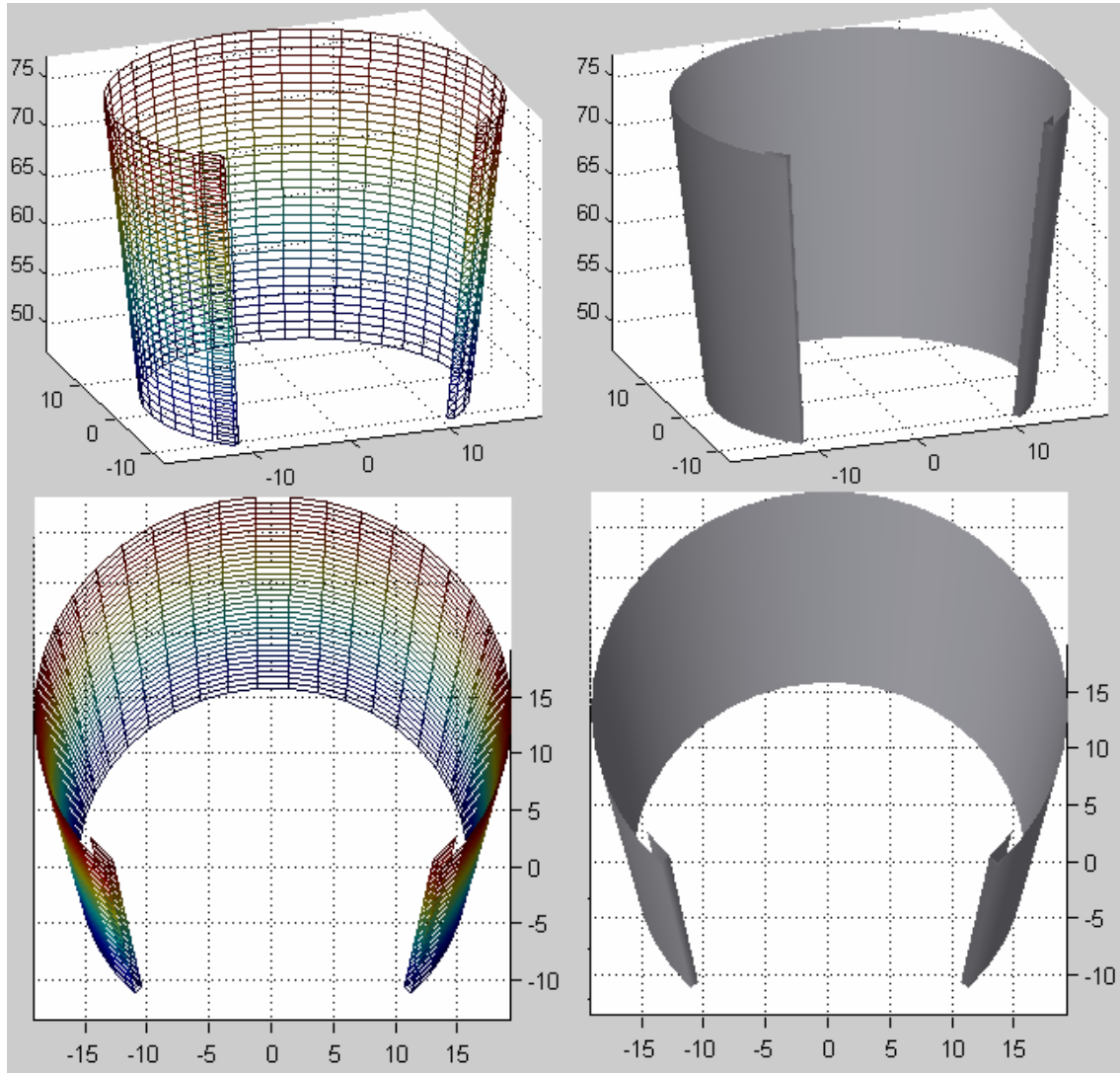
```
[xTrasero yTrasero zTrasero] =  
    getPuntosConoRebajado(15, 15 + 5*(7/45), 0, 40, 47, n);  
dibujar(xTrasero, yTrasero, zTrasero, colorCarcasa);
```

#### 2.1.3. Comentarios

Para dibujar la carcasa inferior recurrimos a la función que genera los puntos de un cono rebajado en su parte inicial, en este caso el rebaje tiene un factor '0' y por tanto el cono aparece cerrado en su base. El radio superior del cono se obtiene a partir de la diferencia de los radios (5) y la altura relativa de esta pieza (7/45) respecto al cuerpo entero.

## 2.2. Cuerpo Inferior

### 2.2.1. Imágenes



### 2.2.2. Código fuente

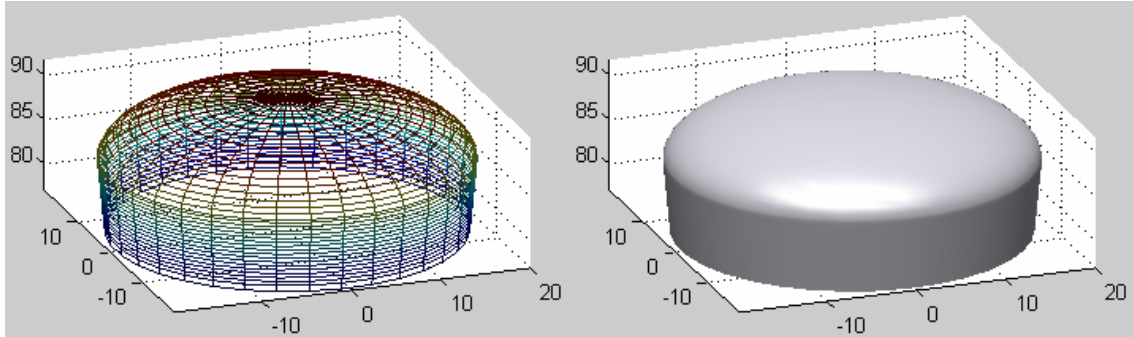
```
[xCuerpoInf yCuerpoInf zCuerpoInf] =  
    getPuntosCuerpoInferior(15 + 5*(7/45), 20 - 5*(7/45), 46.9, 77.1, n);  
dibujar(xCuerpoInf, yCuerpoInf, zCuerpoInf, colorCarcasa);
```

### 2.2.3. Comentarios

El cuerpo inferior se obtiene a partir de un cono que se genera a partir de circunferencias incompletas y además se le han doblado los bordes hacia dentro para darle más sensación de volumen al abrir la puerta.

## 2.3. Cuerpo Superior

### 2.3.1. Imágenes



### 2.3.2. Código fuente

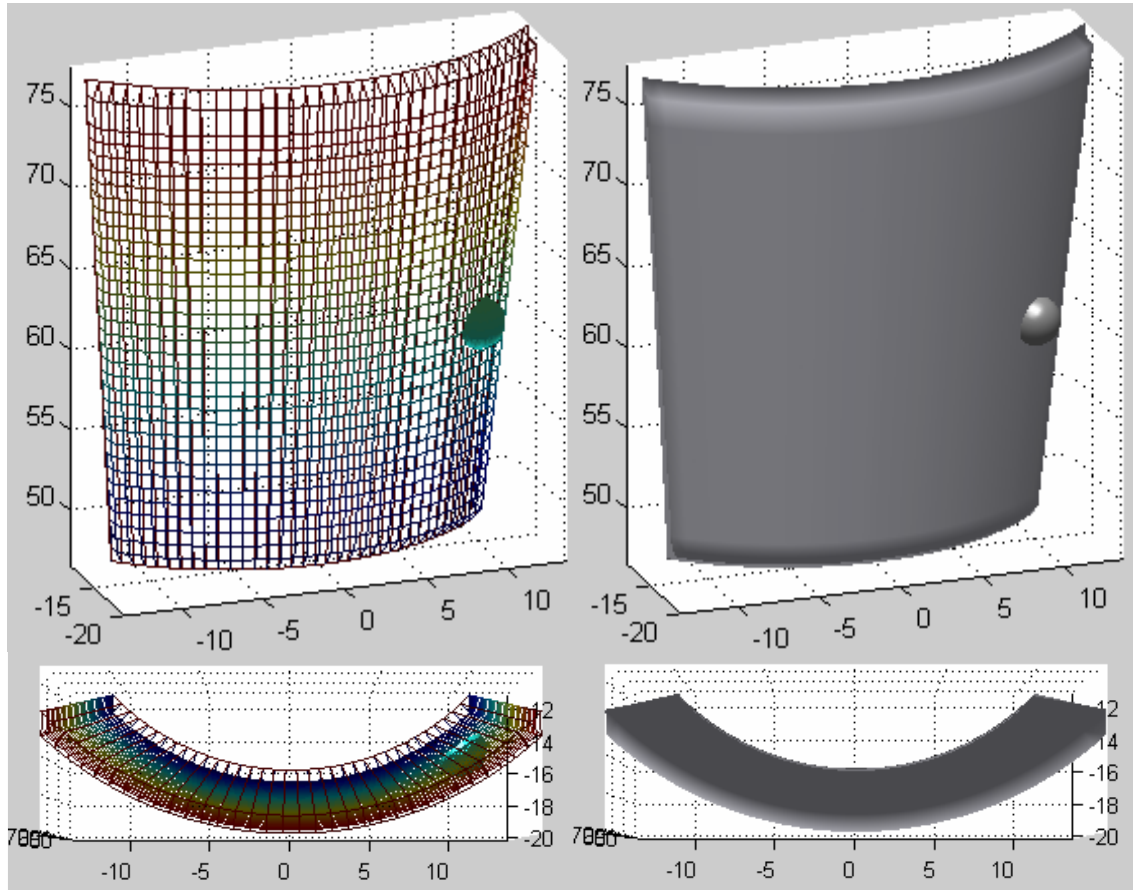
```
[xCuerpoSup yCuerpoSup zCuerpoSup] =  
    getPuntosCuerpoSuperior(20, 20 - 5*(7/45), 1/3, 85, 77, n);  
dibujar(xCuerpoSup, yCuerpoSup, zCuerpoSup, colorCarcasa);
```

### 2.3.3. Comentarios

El cuerpo superior está formado por un cono y una semiesfera achatada unidas mediante una función que toma la mitad de los puntos de cada figura y los fusiona en una sola para evitar que se generen demasiados puntos a la hora de pintar la superficie.

## 2.4. Puerta

### 2.4.1. Imágenes



### 2.4.2. Código fuente

```
tAbrirPuerta = traslacion3D(0, -14.5, -1)
                * xRotacion3D(pi/40)
                * zRotacion3D(-pi/2)
                * yRotacion3D(pi/40);
[xPuerta yPuerta zPuerta] =
    getPuntosPuerta(15 + 5*(7/45), 20 - 5*(7/45), 46.5, 77.5, n);
[xPuerta yPuerta zPuerta] = aplica2(tAbrirPuerta, xPuerta, yPuerta, zPuerta);
dibujar(xPuerta, yPuerta, zPuerta, colorCarcasa);

tPomo = traslacion3D(10, -15, 60) * zRotacion3D(pi/5) * xRotacion3D(9*pi/16);
tPomo = tAbrirPuerta * tPomo;
[xPomo yPomo zPomo] = getPuntosSemiEsfera(1.5, n);
[xPomo yPomo zPomo] = aplica2(tPomo, xPomo, yPomo, zPomo);
dibujar(xPomo, yPomo, zPomo, colorExtremidades);
```

### 2.4.3. Comentarios

La puerta también es una porción de cono al que se le han doblado todos los bordes y se ha unido el principio con el final por detrás para darle volumen.

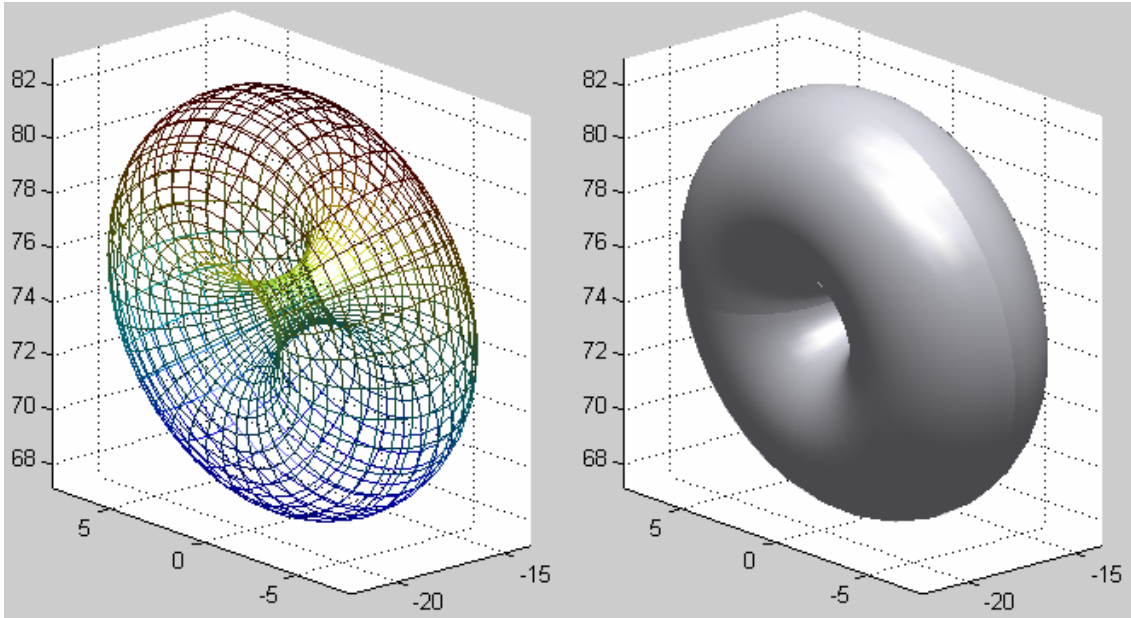
El pomo de la puerta es una semiesfera incrustada en la superficie de la misma.

El movimiento 'tAbrirPuerta' deja la puerta abierta y nos permite ver el interior.

## 3. MODELADO DE LAS EXTREMIDADES SUPERIORES

### 3.1. Hombro izquierdo

#### 3.1.1. Imágenes



#### 3.1.2. Código fuente

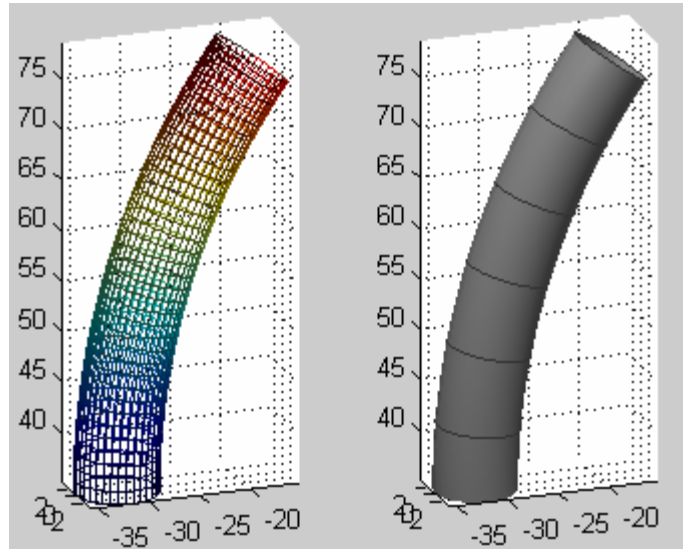
```
% posicion relativa del hombro izquierdo respecto al origen
tHombroIzq = traslacion3D(-18, 0, 75) * yRotacion3D(-pi/20);
[xHombroIzq yHombroIzq zHombroIzq] = getPuntosHombro(1, 8, n);
[xHombroIzq yHombroIzq zHombroIzq] =
    aplica2(tHombroIzq, xHombroIzq, yHombroIzq, zHombroIzq);
dibujar(xHombroIzq, yHombroIzq, zHombroIzq, colorCarcasa);
```

#### 3.1.3. Comentarios

El hombro es un toro de radio interior 1 y radio exterior 8. Una vez obtenidos los puntos se giran  $-\pi/20$  radianes sobre el eje Y y se desplazan la distancia adecuada para situarlo en el lateral izquierdo del cuerpo.

## 3.2. Brazo izquierdo

### 3.2.1. Imágenes



### 3.2.2. Código fuente

```
% posicion relativa del brazo izquierdo respecto al origen
tBrazoIzq = traslacion3D(38, 0, 35);
[xBrazoIzq yBrazoIzq zBrazoIzq] = getPuntosBrazo(4, 70, n);
[xBrazoIzq yBrazoIzq zBrazoIzq] =
    aplica2(tBrazoIzq, xBrazoIzq, yBrazoIzq, zBrazoIzq);
dibujarExtremidad(xBrazoIzq, yBrazoIzq, zBrazoIzq,
    colorExtremidades, colorLineas);
```

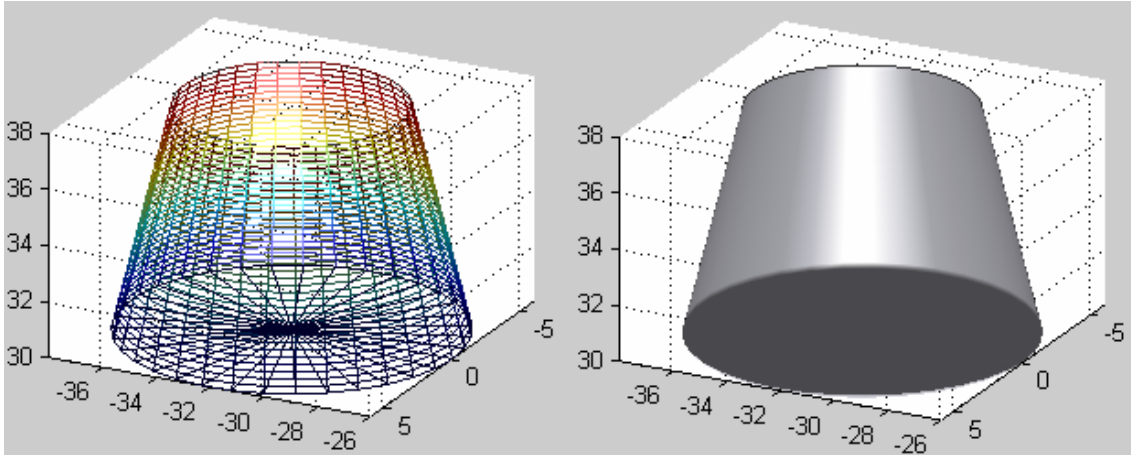
### 3.2.3. Comentarios

El brazo es una porción de toro de radio 70 y formado por circunferencias de radio 4. Una vez obtenidos los puntos se desplazan la distancia adecuada para situarlo en el lateral izquierdo del cuerpo, incrustado en el hombro.

Las secciones se simulan pintando las líneas que recorren las circunferencias que generan el cilindro en intervalos preestablecidos.

### 3.3. Mano izquierda

#### 3.3.1. Imágenes



#### 3.3.2. Código fuente

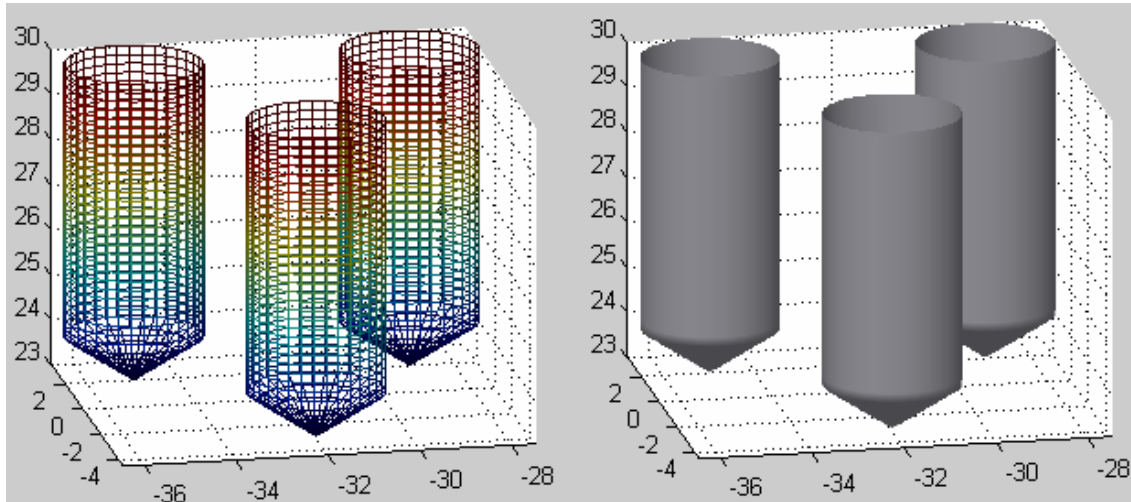
```
% posicion relativa del centro de la mano izquierda respecto al origen
tManoIzq = traslacion3D(-32, 0, 30);
[xManoIzq yManoIzq zManoIzq] = getPuntosConoRebajado(6, 4, 0, 0, 8, n);
[xManoIzq yManoIzq zManoIzq] =
    aplica2(tManoIzq, xManoIzq, yManoIzq, zManoIzq);
dibujarMano(xManoIzq, yManoIzq, zManoIzq, colorCarcasa, colorLineas);
```

#### 3.3.3. Comentarios

Al igual que con la carcasa inferior recurrimos a la función que genera los puntos de un cono rebajado en su parte inicial, en este caso el rebaje también tiene un factor '0' y por tanto el cono aparece cerrado en su base.

## 3.4. Dedos de la mano izquierda

### 3.4.1. Imágenes



### 3.4.2. Código fuente

```
% posicion relativa del primer dedo de la mano izquierda respecto al origen
tDedoIzq1 = traslacion3D(-35, 2, 23);
[xDedoIzq1 yDedoIzq1 zDedoIzq1] = getPuntosDedo(1.5, 0, 7, n);
[xDedoIzq1 yDedoIzq1 zDedoIzq1] =
    aplica2(tDedoIzq1, xDedoIzq1, yDedoIzq1, zDedoIzq1);
dibujar(xDedoIzq1, yDedoIzq1, zDedoIzq1, colorCarcasa);

% posicion relativa del segundo dedo de la mano izquierda respecto al primero
tDedoIzq2 = traslacion3D(6, 0, 0);
[xDedoIzq2 yDedoIzq2 zDedoIzq2] =
    aplica2(tDedoIzq2, xDedoIzq1, yDedoIzq1, zDedoIzq1);
dibujar(xDedoIzq2, yDedoIzq2, zDedoIzq2, colorCarcasa);

% posicion relativa del tercer dedo de la mano izquierda respecto al segundo
tDedoIzq3 = traslacion3D(-3, -5, 0);
[xDedoIzq3 yDedoIzq3 zDedoIzq3] =
    aplica2(tDedoIzq3, xDedoIzq2, yDedoIzq2, zDedoIzq2);
dibujar(xDedoIzq3, yDedoIzq3, zDedoIzq3, colorCarcasa);
```

### 3.4.3. Comentarios

Los dedos se generan a partir de la ecuación de un cilindro al que se le reduce linealmente el radio en su parte inferior para que termine en un cono afilado.

La posición del primer dedo se obtiene a partir del origen de coordenadas, la del segundo a partir de la del primero y la del tercero a partir de la del segundo.

## 3.5. Simetrías ortogonales

### 3.5.1. Código fuente

```
% Hombro Derecho
[xHombroDch yHombroDch zHombroDch] =
    aplica2(s, xHombroIzq, yHombroIzq, zHombroIzq);
dibujar(xHombroDch, yHombroDch, zHombroDch, colorCarcasa);

% Brazo Derecho
[xBrazoDch yBrazoDch zBrazoDch] = aplica2(s, xBrazoIzq, yBrazoIzq, zBrazoIzq);
dibujarExtremidad(xBrazoDch, yBrazoDch, zBrazoDch,
    colorExtremidades, colorLineas);

% Mano Derecha
[xManoDch yManoDch zManoDch] = aplica2(s, xManoIzq, yManoIzq, zManoIzq);
dibujarMano(xManoDch, yManoDch, zManoDch, colorCarcasa, colorLineas);

% Dedos de la Mano Derecha
[xDedoDch1 yDedoDch1 zDedoDch1] = aplica2(s, xDedoIzq1, yDedoIzq1, zDedoIzq1);
dibujar(xDedoDch1, yDedoDch1, zDedoDch1, colorCarcasa);
[xDedoDch2 yDedoDch2 zDedoDch2] = aplica2(s, xDedoIzq2, yDedoIzq2, zDedoIzq2);
dibujar(xDedoDch2, yDedoDch2, zDedoDch2, colorCarcasa);
[xDedoDch3 yDedoDch3 zDedoDch3] = aplica2(s, xDedoIzq3, yDedoIzq3, zDedoIzq3);
dibujar(xDedoDch3, yDedoDch3, zDedoDch3, colorCarcasa);
```

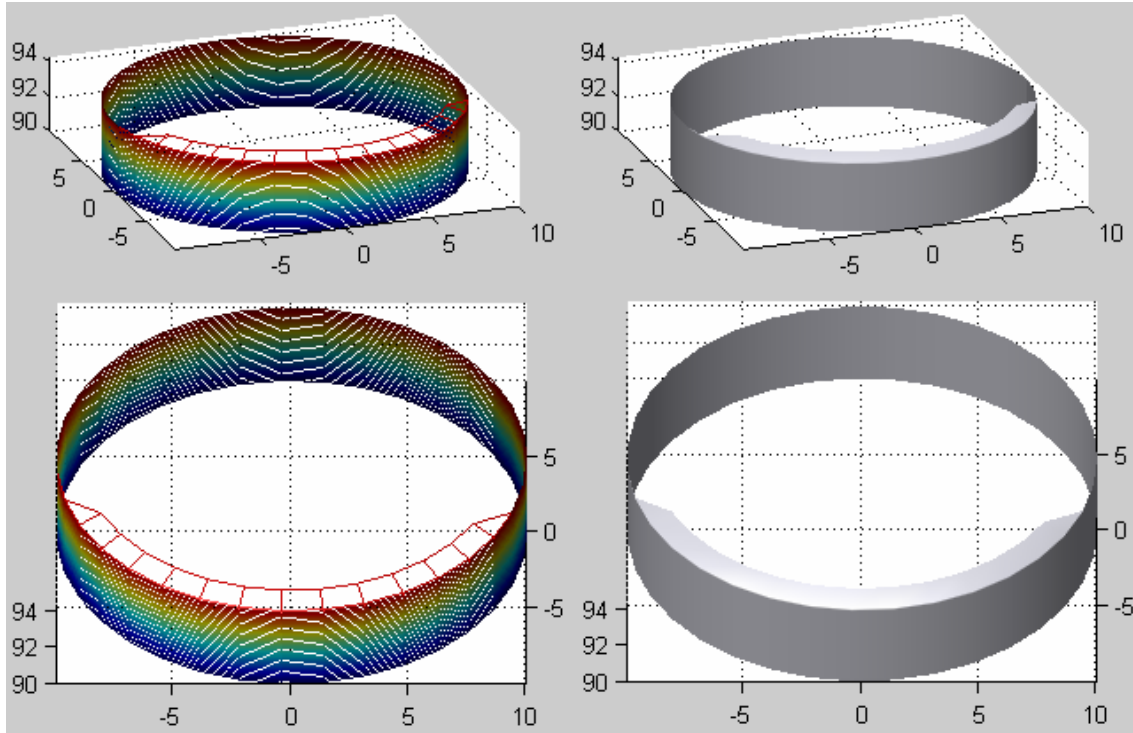
### 3.5.2. Comentarios

Al igual que con las extremidades inferiores recurrimos a una simetría ortogonal respecto al plano OX, ya que la figura está premeditadamente centrada en el origen de coordenadas.

## 4. MODELADO DE LA CABEZA

### 4.1. Cuello

#### 4.1.1. Imágenes



#### 4.1.2. Código fuente

```
[xCuello yCuello zCuello] = getPuntosMandibula(10, 90, 94, n);  
dibujar(xCuello, yCuello, zCuello, colorCarcasa);
```

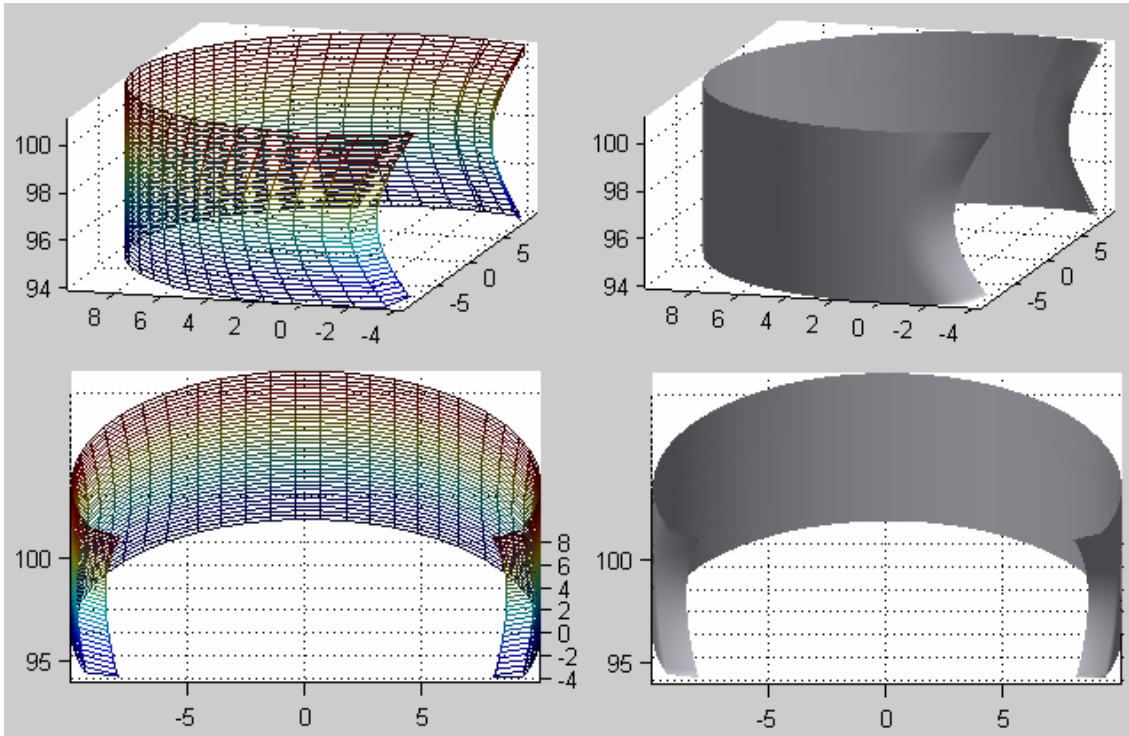
#### 4.1.3. Comentarios

El cuello es un cilindro rebajado en la zona de los dientes para aportar un mayor volumen a la mandíbula inferior.

Como la figura está centrada en el origen, evitamos trasladar la superficie de esta pieza en el eje Z asignado unas alturas adecuadas en el momento de generar los puntos.

## 4.2. Nuca

### 4.2.1. Imágenes



### 4.2.2. Código fuente

```
[xNuca yNuca zNuca] = getPuntosNuca(10, 93.9, 101.1, n);  
dibujar(xNuca, yNuca, zNuca, colorCarcasa);
```

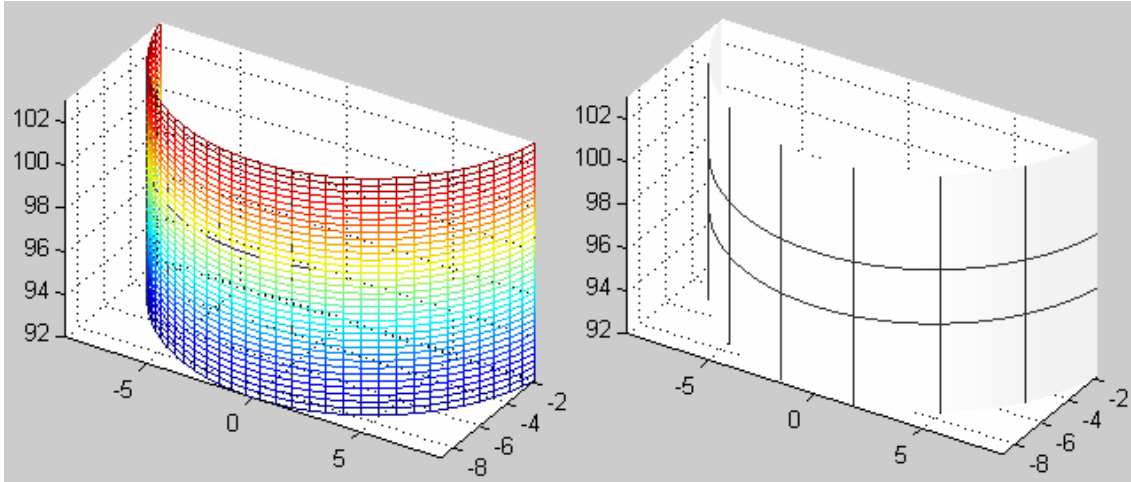
### 4.2.3. Comentarios

La zona de la nuca es quizá la parte más complicada de todo el modelo. De manera simplificada se puede decir que la nuca es una porción de cilindro al que se le hace variar el radio en los extremos para darle el aspecto redondeado a los bordes laterales de la boca. También se le aplica un cierto rebaje para conseguir nuevamente una mejor sensación de volumen al modelo.

Como la figura está centrada en el origen, evitamos trasladar la superficie de esta pieza en el eje Z asignado unas alturas adecuadas en el momento de generar los puntos.

## 4.3. Dientes

### 4.3.1. Imágenes



### 4.3.2. Código fuente

```
[xDientes yDientes zDientes] = getPuntosDientes(9, 92, 103, n);  
dibujarDientes(xDientes, yDientes, zDientes, colorDientes, colorLineas);
```

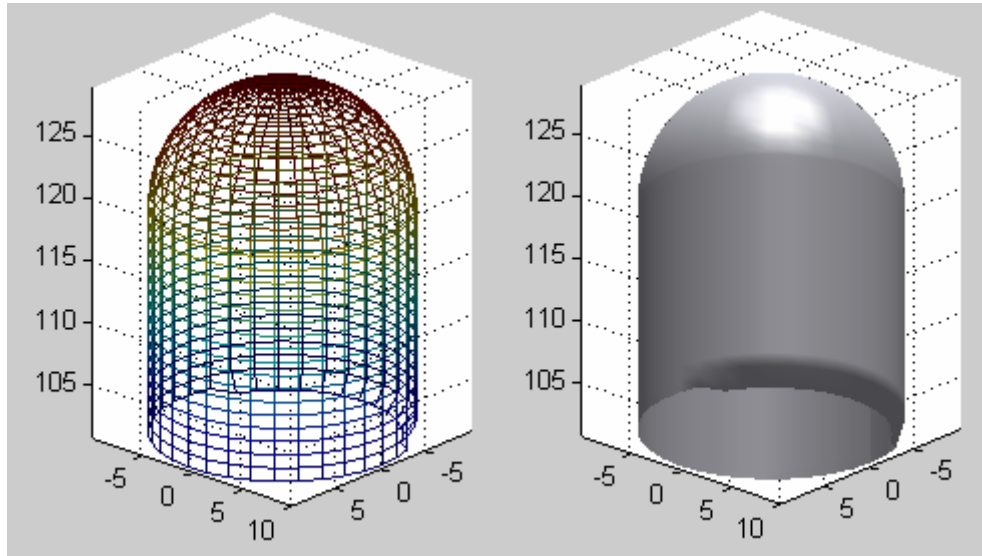
### 4.3.3. Comentarios

Para representar los dientes se emplea nuevamente una porción de cilindro con un radio un poco menor que el resto de la cabeza y se dibujan las líneas de puntos adecuadas para simular los dientes.

Como la figura está centrada en el origen, evitamos trasladar la superficie de esta pieza en el eje Z asignado unas alturas adecuadas en el momento de generar los puntos.

## 4.4. Cabeza

### 4.4.1. Imágenes



### 4.4.2. Código fuente

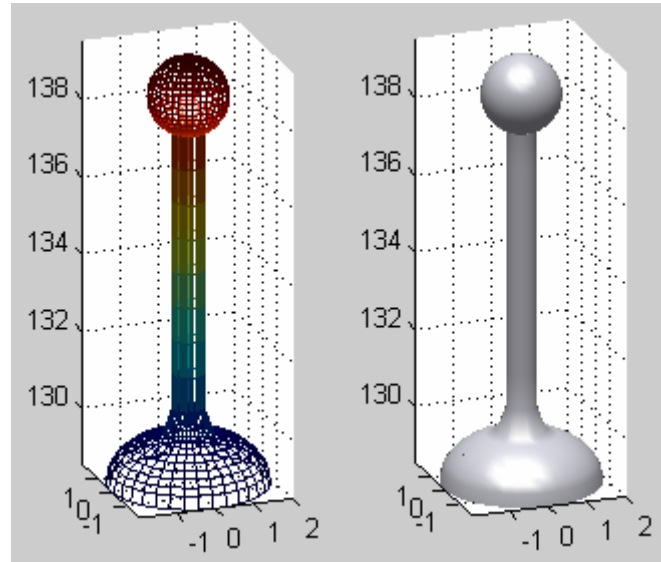
```
% posicion relativa de la cabeza respecto al origen
tCabeza = traslacion3D(0, 0, 101);
[xCabeza yCabeza zCabeza] = getPuntosCabeza(10, 18, n);
[xCabeza yCabeza zCabeza] = aplica2(tCabeza, xCabeza, yCabeza, zCabeza);
dibujar(xCabeza, yCabeza, zCabeza, colorCarcasa);
```

### 4.4.3. Comentarios

La cabeza está formada por una pieza idéntica al cuello volteada respecto al plano OZ y una semiesfera fusionadas en una sola pieza mediante la función definida en 'unirSuperficies.m'

## 4.5. Antena

### 4.5.1. Imágenes



### 4.5.2. Código fuente

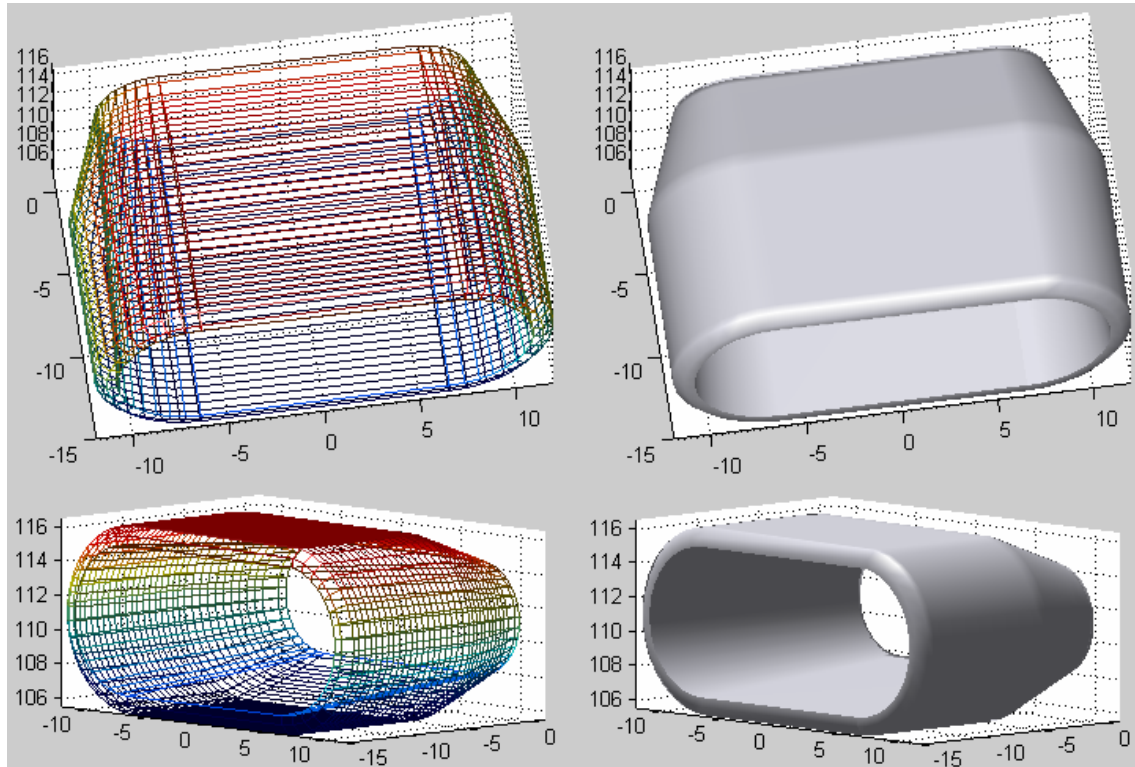
```
% posicion relativa de la antena respecto al origen
tAntena = traslacion3D(0, 0, 128.5);
[xAntena yAntena zAntena] = getPuntosAntena(1, 2, 10, n);
[xAntena yAntena zAntena] = aplica2(tAntena, xAntena, yAntena, zAntena);
dibujar(xAntena, yAntena, zAntena, colorCarcasa);
```

### 4.5.3. Comentarios

La antena está formada por una esfera, un cilindro y una semiesfera achatada en la base. El número de puntos empleados en la esfera es el doble que en las otras superficies.

## 4.6. Visera

### 4.6.1. Imágenes



### 4.6.2. Código fuente

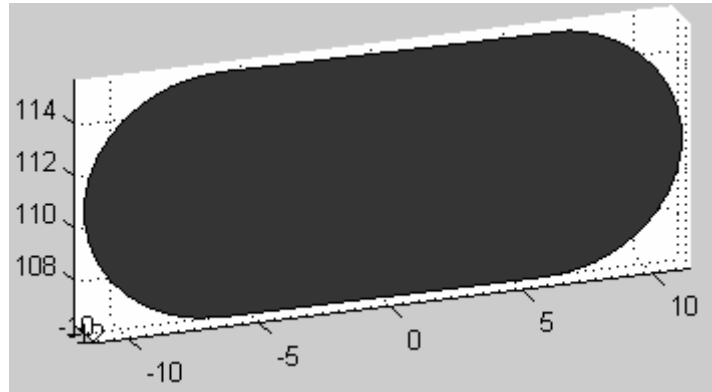
```
% posicion relativa de la visera respecto al origen
tVisera = traslacion3D(-6.5, -15, 111) * xRotacion3D(-pi/2);
[xVisera yVisera zVisera] = getPuntosVisera(5.5, 13, 0, 16, n);
[xVisera yVisera zVisera] = aplica2(tVisera, xVisera, yVisera, zVisera);
dibujar(xVisera, yVisera, zVisera, colorCarcasa);
```

### 4.6.3. Comentarios

La antena está formada por una esfera, un cilindro y una semiesfera achatada en la base. El número de puntos empleados en la esfera es el doble que en las otras superficies.

## 4.7. Fondo de la visera

### 4.7.1. Imágenes



### 4.7.2. Código fuente

```
tFondoVisera = traslacion3D(0, 0, 111)
                * escalado3D(0.95, 0.95, 0.85)
                * traslacion3D(0, 0, -111);
[xFondoVisera yFondoVisera zFondoVisera] = aplica2(tFondoVisera,
                                                    xVisera(11,:),

                                                    yVisera(11,:),
                                                    zVisera(11,:));
fill13(xFondoVisera, yFondoVisera, zFondoVisera, colorSombra);
```

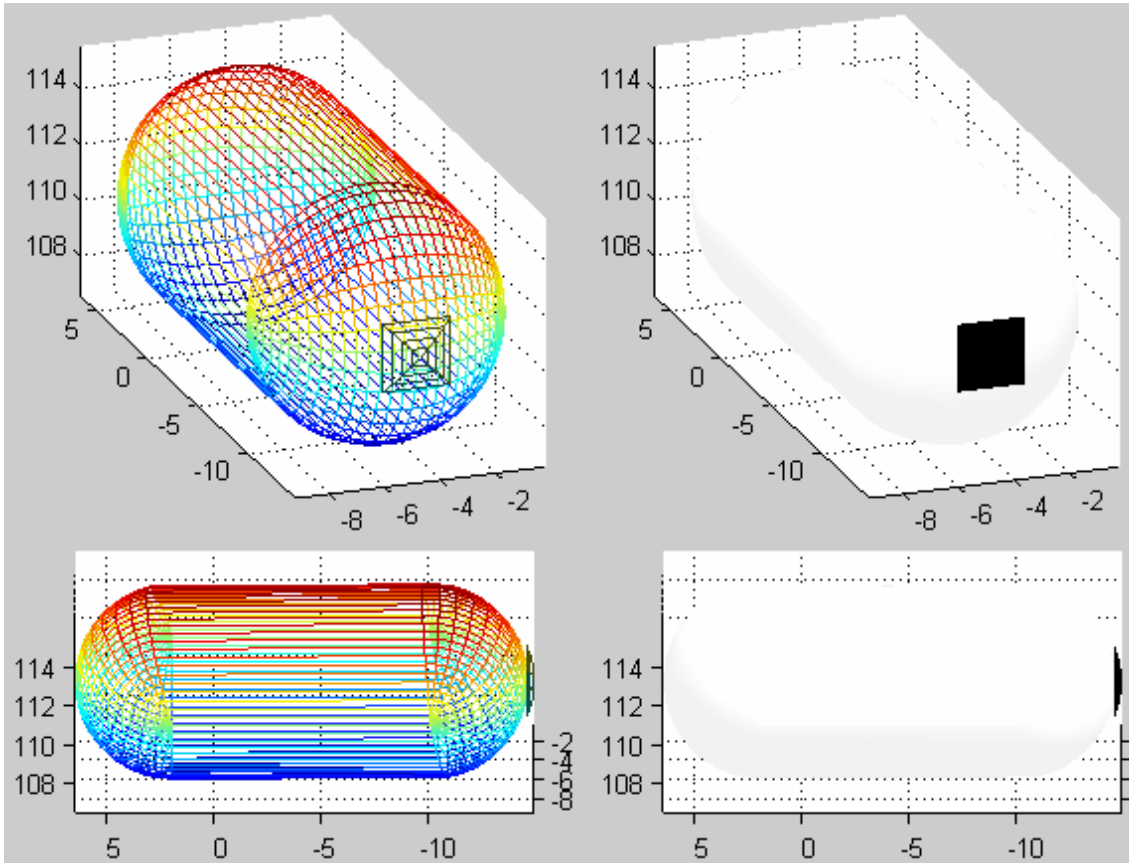
### 4.7.3. Comentarios

Esta pieza tiene como misión oscurecer la parte interior de la visera para aportar mayor realismo y expresividad a la zona de los ojos.

Los puntos se obtienen a partir de los de la visera explicada en el apartado anterior.

## 4.8. Ojo izquierdo

### 4.8.1. Imágenes



### 4.8.2. Código fuente

```
% posicion relativa del ojo izquierdo respecto al origen tOjoIzq =  
traslacion3D(-4.8, -10, 111) * xRotacion3D(-pi/2);  
[xOjoIzq yOjoIzq zOjoIzq] = getPuntosOjo(4.5, 12, n);  
[xOjoIzq yOjoIzq zOjoIzq] = aplica2(tOjoIzq, xOjoIzq, yOjoIzq, zOjoIzq);  
dibujarMaterialPalido(xOjoIzq, yOjoIzq, zOjoIzq, colorOjos);  
  
% posicion relativa de la pupila izquierda respecto al origen  
tPupilaIzq = translacion3D(-4.8, -10.3, 111) * xRotacion3D(-pi/2);  
[xPupilaIzq yPupilaIzq zPupilaIzq] = getPuntosPupila(4.5);  
[xPupilaIzq yPupilaIzq zPupilaIzq] =  
    aplica2(tPupilaIzq, xPupilaIzq, yPupilaIzq, zPupilaIzq);  
dibujar(xPupilaIzq, yPupilaIzq, zPupilaIzq, colorPupilas);
```

### 4.8.3. Comentarios

El ojo es una esfera con dos porciones separadas y la pupila es otra porción de esfera más pequeña generada a partir de una circunferencia de 5 puntos (cuadrado).

## **4.9. Simetrías ortogonales**

### **4.9.1. Código fuente**

```
% Ojo Derecho
[xOjoDch yOjoDch zOjoDch] = aplica2(s, xOjoIzq, yOjoIzq, zOjoIzq);
dibujarMaterialPalido(xOjoDch, yOjoDch, zOjoDch, colorOjos);

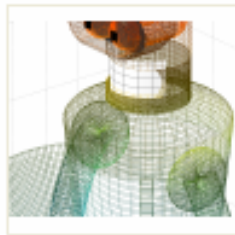
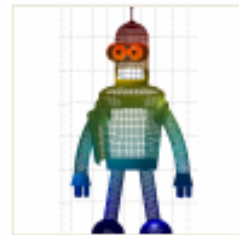
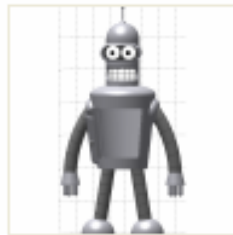
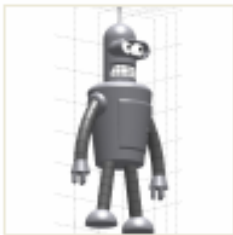
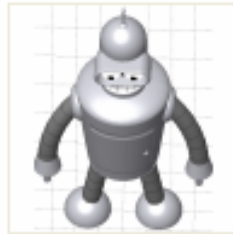
% Pupila Derecha
[xPupilaDch yPupilaDch zPupilaDch] = aplica2(s, xPupilaIzq, yPupilaIzq,
zPupilaIzq);
dibujar(xPupilaDch, yPupilaDch, zPupilaDch, colorPupilas);
```

### **4.9.2. Comentarios**

Al igual que con las extremidades para modelar el ojo y la pupila derecha recurrimos a una simetría ortogonal respecto al plano OX, ya que la figura está centrada en el origen de coordenadas.

## 5. MODELO 3D COMPLETO

### 5.1. Galería de imágenes



## 5.2. *Fichero principal*

La función que genera y dibuja el modelo 3D completo está definida en el fichero 'bender.m'.

### 5.2.1. Parámetros de entrada

Recibe un único parámetro que sirve para indicar si la puerta hay que representarla pintarla abierta o cerrada, para lo cual especificamos un valor distinto de cero o cero respectivamente.

### 5.2.2. Valor de retorno

Esta función no devuelve ningún valor.

### 5.2.3. Constantes

```
% Dimension de las matrices de puntos
n = 36;

% Color de la carcasa general
colorCarcasa = [244/255 244/255 255/255];

% Color de las extremidades
colorExtremidades = [175/255 175/255 175/255];

% Color de las lineas
colorLineas = [50/255 50/255 50/255];

% Color de los ojos
colorOjos = [1 1 1];

% Color de las pupilas
colorPupilas = [0 0 0];

% Color de las sombra
colorSombra = [100/255 100/255 100/255];

% Color de los dientes
colorDientes = [1 1 1];

% Simetria ortogonal respecto al eje central de la figura (plano OX)
s = simetriaOrtogonal3D([0 0 0; 0 1 0; 0 0 1]);
```

## ANEXO

### ***aplica2.m***

```
function [X,Y,Z] = aplica2(t, x, y, z)
%Forma de uso: [X,Y,Z] = aplica2(t, x, y, z)
%Aplica transformacion 't' a x,y,z en formato patch

for i=1:size(x, 1),
    for j=1:size(x,2),
        [X(i,j),Y(i,j),Z(i,j)] = aplicap(t,x(i,j),y(i,j),z(i,j));
    end
end
end
```

### ***aplicap.m***

```
function [X,Y,Z] = aplicap(t,x,y,z)
%Forma de uso: [X,Y,Z] = aplicap(t, x, y, z)
%Aplica transformacion 't' al punto (x,y,z)

p = t * [x;y;z;1];
X = p(1);
Y = p(2);
Z = p(3);
```

### ***dibujar.m***

```
function h = dibujar(X, Y, Z, color)
% Forma de uso: h = dibujar(X, Y, Z, color)
%
% Entrada:
% X      -> Matriz de valores de 'x'
% Y      -> Matriz de valores de 'y'
% Z      -> Matriz de valores de 'z'
% color  -> color de la superficie
%
% Salida:
% h -> manejador de la superficie representada

h = surf(X,Y,Z);
set(h, 'FaceColor', color);
set(h, 'LineStyle', 'none');
set(h, 'FaceLighting', 'phong');
```

## **dibujarDientes.m**

```
function dibujarDientes(X, Y, Z, colorDientes, colorLineas)
% Forma de uso: h = dibujarDientes (X, Y, Z, colorDientes, colorLineas)
%
% Entrada:
% X          -> Matriz de valores de 'x'
% Y          -> Matriz de valores de 'y'
% Z          -> Matriz de valores de 'z'
% colorDientes -> color de los dientes
% colorLineas -> color de las lineas
%
% Salida:
% h -> manejador de la superficie representada

h = dibujarMaterialPalido(X, Y, Z, colorDientes);

hold on

% Dibujamos las dos lineas horizontales de los dientes
c = round(length(X) / 3);
dibujarLinea(X(c+2,:), Y(c+2,:), Z(c+2,:), colorLineas);
dibujarLinea(X(2*c-2,:), Y(2*c-2,:), Z(2*c-2,:), colorLineas);

% Dibujamos las lineas verticales de los dientes
c = round(length(X) / 8);
for i=1:6
    dibujarLinea(X(:,i*c+1), Y(:,i*c+1), Z(:,i*c+1), colorLineas);
end
```

## **dibujarExtremidad.m**

```
function h = dibujarExtremidad(X, Y, Z, colorCarcasa, colorLineas)
% Forma de uso: h = dibujarExtremidad(X, Y, Z, colorCarcasa, colorLineas)
%
% Entrada:
% X          -> Matriz de valores de 'x'
% Y          -> Matriz de valores de 'y'
% Z          -> Matriz de valores de 'z'
% colorCarcasa -> color de la carcasa
% colorLineas -> color de las lineas
%
% Salida:
% h -> manejador de la superficie representada

h = dibujar(X, Y, Z, colorCarcasa);

% Dibujamos las circunferencias que separan
% porciones de 1/5 de la superficie para
% simular las secciones de la extremidad
c = round(length(X) / 5) - 1;
for i=1:6
    dibujarLinea(X(i*c,:), Y(i*c,:), Z(i*c,:), colorLineas);
end
```

## **dibujarLinea.m**

```
function h = dibujarLinea(X, Y, Z, colorLinea)
% Forma de uso: h = dibujarLinea(X, Y, Z, colorLinea)
%
% Entrada:
% X          -> Matriz de valores de 'x'
% Y          -> Matriz de valores de 'y'
% Z          -> Matriz de valores de 'z'
% colorLinea -> color de la linea
%
% Salida:
% h -> manejador de la superficie representada

h = plot3(X, Y, Z, 'k');
set(h, 'Color', colorLinea);
set(h, 'LineWidth', 1);
```

## **dibujarMano.m**

```
function h = dibujarMano(X, Y, Z, colorCarcasa, colorLineas)
% Forma de uso: h = dibujarMano(X, Y, Z, colorCarcasa, colorLineas)
%
% Entrada:
% X          -> Matriz de valores de 'x'
% Y          -> Matriz de valores de 'y'
% Z          -> Matriz de valores de 'z'
% colorCarcasa -> color de la carcasa
% colorLineas  -> color de las lineas
%
% Salida:
% h -> manejador de la superficie representada

h = dibujar(X, Y, Z, colorCarcasa);

hold on

% Dibujamos tambien la linea de union con el brazo
lg = length(X);
dibujarLinea(X(lg,:), Y(lg,:), Z(lg,:), colorLineas);
```

## **dibujarMaterialPalido.m**

```
function h = dibujarMaterialPalido(X, Y, Z, color)
% Forma de uso: h = dibujarMaterialPalido(X, Y, Z, color)
%
% Entrada:
% X      -> Matriz de valores de 'x'
% Y      -> Matriz de valores de 'y'
% Z      -> Matriz de valores de 'z'
% color  -> color de la superficie
%
% Salida:
% h -> manejador de la superficie representada

h = dibujar(X, Y, Z, color);

% Estas lineas modifican el material
% de la superficie dibujada dejando
% el resto de las superficies igual
set(h, 'AmbientStrength', 0.95);
set(h, 'DiffuseStrength', 0.2);
set(h, 'SpecularStrength', 0.5);
set(h, 'SpecularExponent', 10);
set(h, 'SpecularColorReflectance', 1);
```

## **escalado3D.m**

```
function t = escalado3D(ex, ey, ez)
% Forma de uso: t = escalado3D(ex, ey, ez)
%
% Transformacion escalado 3D:
% (y1)   ( ex  ___  ___  b1)   (x1)
% (y2)   = ( ___  ey  ___  b2) * (x2)
% (y3)   ( ___  ___  ez  b3)   (x3)
% ( 1)   (  0  0  0  1)   (x4)

t = eye(4);
t(1,1) = ex;
t(2,2) = ey;
t(3,3) = ez;
```

## getPuntosAntena.m

```
function [X, Y, Z] = getPuntosAntena(rEsfera, rBase, h, n)
% Forma de uso: [X, Y, Z] = getPuntosAntena(rEsfera, rBase, h, n)
%
% Entrada:
% rEsfera -> radio de la esfera superior
% rBase -> radio de la base de la antena
% h -> altura de la antena
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Esfera
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preparamos la malla para la esfera dejando un
% hueco abajo para unirla con la base (7*pi/8)
u = linspace(0, 2*pi, n);
v = linspace(0, 7*pi/8, n);
[U V] = meshgrid(u, v);

% Ecuacion de la esfera en coordenadas esfericas
xEsfera = rEsfera * sin(V) .* cos(U);
yEsfera = rEsfera * sin(V) .* sin(U);
zEsfera = rEsfera * cos(V);

% Elevamos la esfera la altura deseada
t = traslacion3D(0, 0, h);
[xEsfera yEsfera zEsfera] = aplica2(t, xEsfera, yEsfera, zEsfera);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Base
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preparamos la malla para la base dejando un
% hueco arriba para unirla con la esfera (pi/10)
u = linspace(0, 2*pi, n);
v = linspace(pi/10, pi/2, n);
[U V] = meshgrid(u, v);

% Ecuacion de la esfera en coordenadas esfericas
% modificada para que este achatada en el eje z
xBase = rBase * sin(V) .* cos(U);
yBase = rBase * sin(V) .* sin(U);
zBase = (rBase * 2/3) * cos(V);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Cono
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Unimos la esfera y la semiesfera con un cono
% recortado por la punta
radio = xEsfera(n,1); % punto mas a la derecha del nivel inferior de la esfera
hIni = zEsfera(n,1); % aqui vale cualquier valor de 'z' del nivel inferior de
la esfera
rFin = xBase(2,1); % punto mas a la derecha del nivel superior de la base
hFin = zBase(2,1); % aqui vale cualquier valor de 'z' del nivel superior de la
base
[xCono yCono zCono] = getPuntosCilindro(radio, hIni, hFin, n);

[X, Y, Z] = unirSuperficies(xCono, yCono, zCono, xBase, yBase, zBase, n);
[X, Y, Z] = unirSuperficies(xEsfera, yEsfera, zEsfera, X, Y, Z, n);
```

## **getPuntosBrazo.m**

```
function [X, Y, Z] = getPuntosBrazo(radio, radioToro, n)
% Forma de uso: [X, Y, Z] = getPuntosBrazo(radio, radioToro, n)
%
% Entrada:
% radio      -> radio de las circunferencias del brazo
% radioToro  -> radio hasta el centro de las circunferencias
% n          -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = linspace(pi, 4*pi/5, n);
[U, V] = meshgrid(u, v);

% Ecuacion del toro
X = (radioToro + radio * cos(U)) .* cos(V);
Y = radio * sin(U);
Z = (radioToro + radio * cos(U)) .* sin(V);
```

## **getPuntosCabeza.m**

```
function [X, Y, Z] = getPuntosCabeza(radio, largo, n)
% Forma de uso: [X, Y, Z] = getPuntosCabeza(radio, largo, n)
%
% Entrada:
% radio -> radio del cilindro inferior y la semiesfera
% largo -> distancia desde la parte inferior del cilindro hasta el centro de
la semiesfera
% n      -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% SemiEsfera Superior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[xSup ySup zSup] = getPuntosSemiEsfera(radio, n);

% Elevamos la semiesfera la altura indicada
t = traslacion3D(0, 0, largo - 0.1);
[xSup ySup zSup] = aplica2(t, xSup, ySup, zSup);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Cilindro Inferior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reutilizamos la funcion de la mandibula
[xInf yInf zInf] = getPuntosMandibula(radio, -largo, 0, n);

% Volteamos el cilindro verticalmente
s = simetriaOrtogonal3D([1 0 0; 0 1 0; 0 0 0]);
[xInf yInf zInf] = aplica2(s, xInf, yInf, zInf);

[X, Y, Z] = unirSuperficies(xSup, ySup, zSup, xInf, yInf, zInf, n);
```

## **getPuntosCilindro.m**

```
function [X, Y, Z] = getPuntosCilindro(radio, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosCilindro(radio, hIni, hFin, n)
%
% Entrada:
% radio -> radio de las circunferencias del cilindro
% hIni -> altura inicial del cilindro
% hFin -> altura final del cilindro
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla y calculamos las alturas (Z)
u = linspace(0, 2*pi, n);
h = linspace(hIni, hFin, n)';
[U Z] = meshgrid(u, h);

% Ecuacion de la circunferencia
X = radio * cos(U);
Y = radio * sin(U);
```

## **getPuntosCilindroRebajado.m**

```
function [X, Y, Z] = getPuntosCilindroRebajado(radio, factor, hIni, hFin, n)
% Forma de uso:
% [X, Y, Z] = getPuntosCilindroRebajado(radio, factor, hIni, hFin, n)
%
% Entrada:
% radio -> radio de las circunferencias del cilindro
% factor -> factor de rebaje: 0 (cerrado)
% < 0 (rebaje interior)
% 1 (sin rebaje)
% > 0 (rebaje exterior)
% hIni -> altura inicial del cilindro
% hFin -> altura final del cilindro
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = [(ones(1, (n - 1))*radio), (radio * factor)];
[U V] = meshgrid(u, v);

% Calculamos las alturas (Z)
h = [linspace(hIni, hFin, (n - 1)), hFin];
[H Z] = meshgrid(h);

% Ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);
```

## **getPuntosConoCortado.m**

```
function [X, Y, Z] = getPuntosConoCortado(rIni, rFin, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosConoCortado(rIni, rFin, hIni, hFin, n)
%
% Entrada:
% rIni -> radio de la circunferencia inicial
% rFin -> radio de la circunferencia final
% hIni -> altura inicial del cilindro
% hFin -> altura final del cilindro
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = linspace(rIni, rFin, n);
[U V] = meshgrid(u, v);

% Calculamos las alturas (Z)
h = linspace(hIni, hFin, n);
[H Z] = meshgrid(h);

% Ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);
```

## **getPuntosConoRebajado.m**

```
function [X, Y, Z] = getPuntosConoRebajado(rIni, rFin, factor, hIni, hFin, n)
% Forma de uso:
% [X, Y, Z] = getPuntosConoRebajado(rIni, rFin, factor, hIni, hFin, n)
%
% Entrada:
% rIni -> radio de la circunferencia inicial
% rFin -> radio de la circunferencia final
% factor -> factor de rebaje: 0 (cerrado)
% < 0 (rebaje interior)
% 1 (sin rebaje)
% > 0 (rebaje exterior)
% hIni -> altura inicial del cilindro
% hFin -> altura final del cilindro
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = [(rIni * factor), linspace(rIni, rFin, n-1)];
[U V] = meshgrid(u, v);

% Calculamos las alturas (Z)
h = [hIni, linspace(hIni, hFin, n-1)];
[H Z] = meshgrid(h);

% Ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);
```

## **getPuntosCuerpoInferior.m**

```
function [X, Y, Z] = getPuntosCuerpoInferior(rIni, rFin, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosCuerpoInferior(rIni, rFin, hIni, hFin, n)
%
% Entrada:
% rIni    -> radio de la circunferencia inicial
% rFin    -> radio de la circunferencia final
% hIni    -> altura inicial
% hFin    -> altura final
% n       -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos los valores para la malla de puntos
u = [0, -pi/4, linspace(-pi/4, 5*pi/4, n-4), 5*pi/4, 0];
v = linspace(rIni, rFin, n);

% Doblamos hacia dentro los bordes
% de la abertura de la puerta
u(1,1) = u(1,4);
u(1,n) = u(1,n-3);

% El tipico meshgrid
[U V] = meshgrid(u, v);

% Reducimos el radio en los extremos
% de la figura para doblar los bordes
% laterales de la abertura de la puerta
V(:,1) = V(:,1) * 0.95;
V(:,2) = V(:,2) * 0.95;
V(:,n-1) = V(:,n-1) * 0.95;
V(:,n) = V(:,n) * 0.95;

% Ecuacion del cono a partir de la
% ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);

% Elevamos (Z) cada porcion la altura
% correspondiente
h = linspace(hIni, hFin, n)';
[H Z] = meshgrid(h, h);
```

## **getPuntosCuerpoSuperior.m**

```
function [X, Y, Z]=getPuntosCuerpoSuperior(rIni, rFin, factor, hIni, hFin, n)
% Forma de uso: [X, Y, Z] =
%   getPuntosCuerpoSuperior(rIni, rFin, factor, hIni, hFin, n)
%
% Entrada:
% rIni   -> radio de semiesfera superior
% rFin   -> radio pequeño del cono inferior
% factor -> factor de achatamiento de la semiesfera superior
% hIni   -> altura del centro de la semiesfera superior
% hFin   -> altura de la parte inferior del cono
% n      -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% SemiEsfera Achatada Superior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[xSup ySup zSup] = getPuntosSemiEsferaAchatada(rIni, factor, n);

% Elevamos la semiesfera la altura correspondiente
t = translacion3D(0, 0, hIni);
[xSup ySup zSup] = aplica2(t, xSup, ySup, zSup);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Cono Inferior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[xInf yInf zInf] = getPuntosConoCortado(rIni, rFin, hIni, hFin, n);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Resultado final
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[X, Y, Z] = unirSuperficies(xSup, ySup, zSup, xInf, yInf, zInf, n);
```

## **getPuntosDedo.m**

```
function [X, Y, Z] = getPuntosDedo(radio, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosDedo(radio, hIni, hFin, n)
%
% Entrada:
% radio -> radio de las circunferencias
% hIni -> altura inicial
% hFin -> altura final
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Dividimos la superficie en 6 porciones
m = 5 * round(n/6);

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = [linspace(0, radio, (n - m)) (ones(1,m) * radio)];
[U V] = meshgrid(u, v);

% Ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);

% Calculamos las alturas (Z)
h = linspace(hIni, hFin, n)';
[H Z] = meshgrid(h, h);
```

## **getPuntosDientes.m**

```
function [X, Y, Z] = getPuntosDientes(radio, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosDientes(radio, hIni, hFin, n)
%
% Entrada:
% radio -> radio de la boca
% hIni -> altura inicial
% hFin -> altura final
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(-pi/15, -14*pi/15, n);
h = linspace(hIni, hFin, n);
[U Z] = meshgrid(u, h);

% Ecuacion de la circunferencia
X = radio * cos(U);
Y = radio * sin(U);
```

## **getPuntosEsfera.m**

```
function [X, Y, Z] = getPuntosEsfera(radio, n)
% Forma de uso: [X, Y, Z] = getPuntosEsfera(radio, n)
%
% Entrada:
% radio -> radio de la esfera
% n      -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = linspace(0, pi, n);
[U V] = meshgrid(u, v);

% Ecuacion de la esfera
X = radio * sin(V) .* cos(U);
Y = radio * sin(V) .* sin(U);
Z = radio * cos(V);
```

## **getPuntosHombro.m**

```
function [X, Y, Z] = getPuntosHombro(rInt, rExt, n)
% Forma de uso: [X, Y, Z] = getPuntosHombro(rInt, rExt, n)
%
% Entrada:
% rInt -> radio interior del toro
% rExt -> radio exterior del toro
% n     -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Calculamos los radio necesarios
r = (rExt + rInt) / 2;
a = (rExt - rInt) / 2;

% Preparamos la malla
u = linspace(0, 2*pi, n);
[U, V] = meshgrid(u, u);

% Ecuacion del toro
X = a * sin(U);
Y = (r + a * cos(U)) .* cos(V);
Z = (r + a * cos(U)) .* sin(V);
```

## **getPuntosMandibula.m**

```
function [X, Y, Z] = getPuntosMandibula(radio, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosDientes(radio, hIni, hFin, n)
%
% Entrada:
% radio -> radio de la cabeza
% hIni -> altura inicial
% hFin -> altura final
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Generamos un cilindro con un factor de rebaje de 0.9
[X Y Z] = getPuntosCilindroRebajado(radio, 0.9, hIni, hFin, n);

% Retocamos el rebaje para que no aparezcan
% lineas de corte con la superficie de la nuca
for i=1:n
    if (i < 7*n/12 | i > n-n/12)
        % Quitamos el rebaje en toda
        % la zona interior de la cabeza
        X(n,i) = X(n-1,i);
        Y(n,i) = Y(n-1,i);
    else
        % Elevamos un poco la zona del rebaje
        Z(n,i) = Z(n,i) + radio * 0.03;
    end
end
end
```

## **getPuntosNuca.m**

```
function [X, Y, Z] = getPuntosNuca(radio, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosNuca(radio, hIni, hFin, n)
%
% Entrada:
% radio -> radio de la cabeza
% hIni -> altura inicial
% hFin -> altura final
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Dividimos la superficie en 10 porciones
m = round(n/10);

% Preparamos los vectores para la malla
u = [linspace(-pi/16, 0, m), linspace(0, 17*pi/16, n-m)];
a = [linspace(-pi/200, -pi/1000, 3*m), (zeros(1, (n - 6*m))), linspace(pi/1000,
pi/200, 3*m)];
alfa = linspace(pi/2, pi, round(n/2));
beta = linspace(pi, pi/2, (n - round(n/2) + 1));

% Generamos parte de la malla para
% que la superficie sea redondeada
% en los laterales
U = zeros(n);
U(1,:) = u + round(n/2) * a;
a = a;
for i=2:round(n/2)
    U(i,:) = U(i-1,:) - a * sin(alfa(1,i));
end
for i=round(n/2):n
    U(i,:) = U(i-1,:) + a * sin(beta(1, (i - round(n/2) + 1)));
end

% Generamos otra malla para los radios (R)
% y para las alturas (Z)
r = [(radio * 0.9), (ones(1, (n - 2))*radio), (radio * 0.9)];
h = linspace(hIni, hFin, n)';
[R Z] = meshgrid(r, h);

% Ecuacion de la circunferencia
X = R .* cos(U);
Y = R .* sin(U);

% Retocamos 'manualmente' las alturas
% de las esquinas para recortarlas
Z(1,1) = Z(2,1);
Z(1,n) = Z(2,n);
Z(n,1) = Z(n-1,1);
Z(n,n) = Z(n-1,n);
```

## **getPuntosOjo.m**

```
function [X, Y, Z] = getPuntosOjo(radio, largo, n)
% Forma de uso: [X, Y, Z] = getPuntosOjo(radio, largo, n)
%
% Entrada:
% radio -> radio de los semiesferas laterales
% largo -> distancia de centro a centro de las 2 semiesferas
% n      -> dimension de la matrices de puntos generadas
%
% Salida:
% X -> Matriz de valores de 'x'
% Y -> Matriz de valores de 'y'
% Z -> Matriz de valores de 'z'

% dividimos los 'n' puntos en 2 porciones de 'm' puntos
m = round(n/2);

% matrices auxiliares
u = linspace(0, 2*pi, n);
v = [linspace(0, pi, (n - 1)) -pi];
[U V] = meshgrid(u, v);

% desplazamiento sobre el eje Z de cada una de las piezas
dz = [(ones(1, (n - m - 1))*largo) zeros(1, m) largo];

% esfera en coordenadas esfericas (modificadas)
X = radio * cos(V);
Y = radio * sin(V) .* cos(U);
Z = radio * sin(V) .* sin(U);

for i=1:n
    Z(i,:) = Z(i,:) + dz; % desplazamos las piezas sobre el eje Z
end
```

## **getPuntosPuerta.m**

```
function [X, Y, Z] = getPuntosPuerta(rIni, rFin, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosPuerta(rIni, rFin, hIni, hFin, n)
%
% Entrada:
% rIni -> radio inicial de la puerta
% rFin -> radio final de la puerta
% hIni -> altura inicial
% hFin -> altura final
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos los valores para la malla de puntos
u = linspace(-pi/4, -3*pi/4, n);
v = [rIni, linspace(rIni, rFin, n-2) * 1.05, rFin];
[U V] = meshgrid(u, v);

% Estas modificaciones del radio
% hacen que los bordes laterales
% se doblen hacia dentro
V(:,1) = V(:,1) * 0.95;
V(:,n) = V(:,n) * 0.95;

% Finalmente retocamos las esquinas
% de la puerta para que no queden
% demasiado afiladas
V(1,1) = V(1,2);
V(n,1) = V(n,2);
V(1,n) = V(1,n-1);
V(n,n) = V(n,n-1);

% Ecuacion del cilindro a partir de la
% ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);

% Generamos las alturas (Z) de cada porcion
% de circunferencia
h = linspace(hIni, hFin, n);
[H Z] = meshgrid(h, h);

% Sutituimos la porcion de circunferencia
% de arriba por la de abajo para cerrar
% la puerta por detras
X(n-2,:) = X(n-1,:); X(n-1,:) = X(n,:); X(n,:) = X(1,:);
Y(n-2,:) = Y(n-1,:); Y(n-1,:) = Y(n,:); Y(n,:) = Y(1,:);
Z(n-2,:) = Z(n-1,:); Z(n-1,:) = Z(n,:); Z(n,:) = Z(1,:);
```

## **getPuntosPupila.m**

```
function [X, Y, Z] = getPuntosPupila(radio)
% Forma de uso: [X, Y, Z] = getPuntosPupila(radio)
%
% Entrada:
% radio -> radio de la esfera de la pupila
%
% Salida:
% X -> Matriz de valores de 'x' de dimension 5
% Y -> Matriz de valores de 'y' de dimension 5
% Z -> Matriz de valores de 'z' de dimension 5

% dimension de la matrices de puntos generadas
n = 5;

% matrices auxiliares
u = linspace(pi/4, 9*pi/4, n);
v = linspace(7*pi/8, pi, n);
[U V] = meshgrid(u, v);

% esfera en coordenadas esfericas
X = radio * sin(V) .* cos(U);
Y = radio * sin(V) .* sin(U);
Z = radio * cos(V);
```

## **getPuntosSemiEsfera.m**

```
function [X, Y, Z] = getPuntosSemiEsfera(radio, n)
% Forma de uso: [X, Y, Z] = getPuntosSemiEsfera(radio, n)
%
% Entrada:
% radio -> radio de la esfera de la pupila
% n      -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = linspace(0, pi/2, n);
[U V] = meshgrid(u, v);

% Ecuacion de la esfera
X = radio * sin(V) .* cos(U);
Y = radio * sin(V) .* sin(U);
Z = radio * cos(V);
```

## **getPuntosSemiEsferaAchatada.m**

```
function [X, Y, Z] = getPuntosSemiEsferaAchatada(radio, factor, n)
% Forma de uso: [X, Y, Z] = getPuntosSemiEsferaAchatada(radio, factor, n)
%
% Entrada:
% radio -> radio de la esfera de la pupila
% factor -> factor de rebaje: 0 (plano)
%                               < 0 (achatada hacia dentro)
%                               1 (sin achatar)
%                               > 0 (achatada hacia fuera)
% n      -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = linspace(0, pi/2, n);
[U V] = meshgrid(u, v);

% Ecuacion de la esfera
% achatada en el eje Z
X = radio * sin(V) .* cos(U);
Y = radio * sin(V) .* sin(U);
Z = (radio * factor) * cos(V);
```

## **getPuntosSemiEsferaCerrada.m**

```
function [X, Y, Z] = getPuntosSemiEsferaCerrada(radio, n)
% Forma de uso: [X, Y, Z] = getPuntosSemiEsfera(radio, n)
%
% Entrada:
% radio -> radio de la esfera de la pupila
% n      -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Preparamos la malla
u = linspace(0, 2*pi, n);
v = [linspace(0, pi/2, n-1), 0];
[U V] = meshgrid(u, v);

% Ecuacion de la esfera
X = radio * sin(V) .* cos(U);
Y = radio * sin(V) .* sin(U);
Z = radio * cos(V);

% Retocamos la altura de la tapa inferior
Z(n,:) = zeros(1,n);
```

## **getPuntosVisera.m**

```
function [X, Y, Z] = getPuntosVisera(radio, largo, hIni, hFin, n)
% Forma de uso: [X, Y, Z] = getPuntosVisera(radio, hIni, hFin, n)
%
% Entrada:
% radio -> radio de los semicilindros laterales
% largo -> distancia de centro a centro de los 2 semicilindros
% hIni -> altura inicial de la visera
% hFin -> altura final de la visera
% n -> dimension de la matrices cuadradas de puntos generadas
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Dividimos los 'n' puntos en 2 porciones de 'm' puntos
m = round(n/2);

% Preparamos los valores para la malla de puntos
u = [pi/2, linspace(pi/2, 5*pi/2, n - 1)]; %
v = [(radio * 0.8), (radio * 0.9), (ones(1, (n - 16)) * radio), linspace(radio,
(radio * 0.6), 14)];
[U V] = meshgrid(u, v);

% Cilindro a partir de la ecuacion de la circunferencia
X = V .* cos(U);
Y = V .* sin(U);
Z = zeros(n);

% Desplazamiento sobre el eje X del semicilindro de la derecha
dx = [largo, zeros(1, m), (ones(1, (n - m - 1))*largo)];

% Alturas (eje Z)
h = [hIni; linspace(hIni, hFin, n - 1)'];

for i=1:n
    X(i,:) = X(i,:) + dx; % desplazamos las piezas sobre el eje X
    Z(:,i) = h; % asignamos las alturas
end

% Retocamos los puntos de la superficie para que tome volumen,
% repitiendo la primera fila al final de la malla de puntos
X(n-2,:) = X(n-1,:); X(n-1,:) = X(n,:); X(n,:) = X(1,:);
Y(n-2,:) = Y(n-1,:); Y(n-1,:) = Y(n,:); Y(n,:) = Y(1,:);
Z(n-2,:) = Z(n-1,:); Z(n-1,:) = Z(n,:); Z(n,:) = Z(1,:);
```

## **simetriaOtogonal3D.m**

```
function t = simetriaOtogonal3D(plano)
% Forma de uso: t = simetriaOtogonal3D(plano)
% plano -> tres puntos del plano de simetria (tres filas)
%
% Transformacion simetria ortogonal 3D:
% (y1)      ( _ _ _ b1)   (x1)
% (y2) =    ( _ _ _ b2) * (x2)
% (y3)      ( _ _ _ b3)   (x3)
% ( 1)      ( 0 0 0  1)   (x4)

A2 = [1 0 0; 0 1 0; 0 0 -1]; % matriz de la simetria antes del cambio de base
r = plano(1,:); % cogemos el primer punto del plano (1ªfila)
q = plano(2,:); % cogemos el segundo punto del plano (2ªfila)
s = plano(3,:); % cogemos el tercer punto del plano (3ªfila)
v1 = q - r; % 1º vector director del plano
v2 = s - r; % 2º vector director del plano
vn = cross(v1, v2); % vector ortogonal

if rank([v1;v2;vn]) ~= 3
    disp(' ERROR: Los tres puntos no forman un plano.')
else
    P = [v1(1) v2(1) vn(1); v1(2) v2(2) vn(2); v1(3) v2(3) vn(3)]; % matriz de
cambio de base
    A = P * A2 * inv(P); % matriz de la simetria en la base canonica
    B = (eye(3) - A) * r'; % B = (I - A) * r

    t = [A B; 0 0 0 1];
end
```

## **traslacion3D.m**

```
function t = traslacion3D(vx,vy,vz)
% Forma de uso: t = traslacion3D(vx, vy, vz)
%
% Transformacion traslacion 3D:
% (y1)      ( _ _ _ b1)   (x1)
% (y2) =    ( _ _ _ b2) * (x2)
% (y3)      ( _ _ _ b3)   (x3)
% ( 1)      ( 0 0 0  1)   (x4)

t = eye(4);
t(1,4) = vx;
t(2,4) = vy;
t(3,4) = vz;
```

## ***unirSuperficies.m***

```
function [X, Y, Z] = unirSuperficies(xSup, ySup, zSup, xInf, yInf, zInf, n)
% Forma de uso: [X, Y, Z] = unirSuperficies(xSup, ySup, zSup, xInf, yInf,
zInf, n)
%
% Entrada:
% xSup -> Matriz de valores de 'x' de la superficie inferior
% ySup -> Matriz de valores de 'y' de la superficie inferior
% zSup -> Matriz de valores de 'z' de la superficie inferior
% xInf -> Matriz de valores de 'x' de la superficie inferior
% yInf -> Matriz de valores de 'y' de la superficie inferior
% zInf -> Matriz de valores de 'z' de la superficie inferior
% n    -> Tamaño de las matrices cuadradas de puntos
%
% Salida:
% X -> Matriz cuadrada de valores de 'x'
% Y -> Matriz cuadrada de valores de 'y'
% Z -> Matriz cuadrada de valores de 'z'

% Creamos tres matrices cuadradas vacias
% para juntar las dos superficies en una
X = zeros(n);
Y = zeros(n);
Z = zeros(n);

% Cogemos la mitad de los puntos
% de cada superficie
m = round(n/2);

% Primero los puntos de la superior
for i=1:m
    X(i,:) = xSup(2*i-1,:);
    Y(i,:) = ySup(2*i-1,:);
    Z(i,:) = zSup(2*i-1,:);
end

% Luego los de la inferior
for i=1:n-m-1
    X(i+m,:) = xInf(2*i-1,:);
    Y(i+m,:) = yInf(2*i-1,:);
    Z(i+m,:) = zInf(2*i-1,:);
end

% La ultima porcion es la ultima
% de la superficie inferior
X(n,:) = xInf(n,:);
Y(n,:) = yInf(n,:);
Z(n,:) = zInf(n,:);
```

### ***xRotacion3D.m***

```
function t = xRotacion3D(rad)
% Forma de uso: t = xRotacion3D(rad)
% Rotacion 3D sobre el eje x, con angulo 'rad'

t = eye(4);
t(2,2) = cos(rad); t(2,3) = -sin(rad);
t(3,2) = sin(rad); t(3,3) = cos(rad);
```

### ***yRotacion3D.m***

```
function t = yRotacion3D(rad)
% Forma de uso: t = yRotacion3D(rad)
% Rotacion 3D sobre el eje y, con angulo 'rad'

t = eye(4);
t(1,1) = cos(rad); t(1,3) = sin(rad);
t(3,1) = -sin(rad); t(3,3) = cos(rad);
```

### ***zRotacion3D.m***

```
function t = zRotacion3D(rad)
% Forma de uso: t = zRotacion3D(rad)
% Rotacion 3D sobre el eje z, con angulo 'rad'

t = eye(4);
t(1,1) = cos(rad); t(1,2) = -sin(rad);
t(2,1) = sin(rad); t(2,2) = cos(rad);
```

## BIBLIOGRAFIA

Apuntes de la asignatura 'Geometría Computacional' de la EUIT Informática de Oviedo  
Profesor responsable: Nieto Fernández, María Covadonga

Notas sobre MATLAB  
Tomás Aranda, J.Gabriel García  
Servicio de Publicaciones  
Universidad de Oviedo

Página con modelos 3D similares al diseñado:  
<http://tfp.killbots.com/?p=list&pname=3d>

Otras imágenes obtenidas a través del buscador 'Google':  
<http://images.google.es/images?hl=es&q=bender+3D>