```
# -*- coding: utf-8 -*-
# FreeCAD Macro: Nave DFD XL adaptada a perihelio extremo
# Autor: Víctor Alonso García + Copilot
# Descripción: Variante XL del DFD con escudos TPS multilayer, blindajes, radiadores en
sombra,
# tanques, sensores y tren, fusionada en un único sólido manifold para 3D printing.
# Unidades: mm

import FreeCAD as App, Part, math

doc = App.newDocument("Nave_DFD_XL_Solar")

# ========================
# Parámetros
# ========================
P = {
    'scale': 2.0,  # nave más grande

    # Fuselaje (igual base, estilo DFD)
    'nose_len': 1500.0, 'nose_base_d': 1100.0,
    'mid_len': 3000.0, 'mid_d': 1800.0,
    'rear_len': 1500.0, 'rear_d': 2200.0,
    'hull_t': 30.0,

    # Escudo térmico frontal multilayer (TPS)
    'shield_d': 2600.0,      # diámetro del escudo principal
    'shield_flecha': 80.0,   # flecha/abombamiento frontal
    't_ceramic': 4.0, 't_foam': 120.0, 't_cc': 12.0,
    'rim_w': 60.0, 'rim_h': 80.0,

    # Mangas/blindajes TPS alrededor del fuselaje y reactor
    'hull_shield_t': 80.0, 'hull_shield_l': 2800.0,
    'reactor_shield_t': 120.0, 'reactor_shield_l': 2200.0,

    # Reactor (como base DFD)
    'reactor_d': 1500.0, 'reactor_l': 1800.0,

    # Módulo hábitat
    'hab_d': 1400.0, 'hab_l': 2500.0,

    # Cabina
    'cockpit_d': 900.0, 'cockpit_l': 800.0, 'window_r': 150.0,

    # Tanques laterales
```

```
    'tank_r': 400.0, 'tank_l': 2000.0, 'tank_off': 1200.0,

    # Tanques esféricos
    'sphere_r': 450.0, 'sphere_off': 1600.0,

    # Radiadores (reformulados a sombra trasera)
    'wing_span': 2500.0, 'wing_th': 60.0, 'wing_l': 2200.0, 'wing_back_offset': 1200.0,

    # Collar térmico y paravientos (deflectores)
    'collar_d_delta': 300.0, 'collar_h': 120.0, 'collar_t': 40.0,
    'def_count': 8, 'def_l': 800.0, 'def_w': 160.0, 'def_t': 30.0,

    # Antenas
    'mast_l': 1000.0, 'mast_r': 40.0, 'dish_r': 400.0,

    # Tren de aterrizaje
    'leg_r': 100.0, 'leg_l': 800.0, 'foot_r': 250.0, 'foot_t': 50.0,

    # Escotillas y acoplamiento
    'dock_r': 400.0, 'dock_l': 300.0, 'dock_off': 800.0,

    # Sensores
    'sensor_r': 50.0, 'sensor_l': 200.0,

    # Refuerzos internos
    'beam_r': 50.0, 'beam_l': 3000.0,

    # Tolerancias de solape para fusión robusta
    'overlap': 2.0
}

# ========================
# Función auxiliar
# ========================
def add_obj(shape, name):
    obj = doc.addObject("Part::Feature", name)
    obj.Shape = shape
    return obj

# ========================
# Fuselaje principal (base DFD)
# ========================
nose = Part.makeCone(0, P['nose_base_d']/2, P['nose_len'])
mid = Part.makeCylinder(P['mid_d']/2, P['mid_len'])
```

```
mid.translate(App.Vector(0,0,P['nose_len']))
rear = Part.makeCone(P['rear_d']/2, P['mid_d']/2, P['rear_len'])
rear.translate(App.Vector(0,0,P['nose_len']+P['mid_len']))
hull = nose.fuse(mid).fuse(rear)

# ========================
# Escudo térmico frontal multilayer (TPS)
# ========================
shield_R = P['shield_d']/2.0
# Cara cerámica (disco + flecha mediante cono corto)
cer = Part.makeCylinder(shield_R, P['t_ceramic'])
cone = Part.makeCone(shield_R, shield_R - 40.0, P['shield_flecha'])
cone.translate(App.Vector(0,0,-P['shield_flecha']))
cer = cer.fuse(cone)

# Núcleo foam
foam = Part.makeCylinder(shield_R - P['overlap'], P['t_foam'])
foam.translate(App.Vector(0,0,P['t_ceramic'] - P['overlap']))

# Capa trasera C/C
back = Part.makeCylinder(shield_R - 2*P['overlap'], P['t_cc'])
back.translate(App.Vector(0,0,P['t_ceramic'] + P['t_foam'] - 2*P['overlap']))

# Rim perimetral
rimOD = shield_R; rimID = shield_R - P['rim_w']
rim = Part.makeCylinder(rimOD, P['rim_h']).cut(Part.makeCylinder(rimID, P['rim_h']))
rim.translate(App.Vector(0,0,P['t_ceramic'] + P['t_foam'] + P['t_cc'] - P['rim_h']))

# Ensamble del escudo como sólido único
shield = cer.fuse(foam).fuse(back).fuse(rim)
# Posicionamiento delante del fuselaje
shield.translate(App.Vector(0,0,-(P['t_ceramic'] + P['t_foam'] + P['t_cc'])))

# ========================
# Blindajes TPS alrededor del fuselaje y reactor (mangas)
# ========================
# Manga cilindrica sobre sección media
hull_shield = Part.makeCylinder(P['mid_d']/2 + P['hull_shield_t'], P['hull_shield_l'])
hull_shield.translate(App.Vector(0,0,P['nose_len'] + (P['mid_len'] - P['hull_shield_l'])/2.0))

# Manga reactor
reactor_shield = Part.makeCylinder(P['reactor_d']/2 + P['reactor_shield_t'], P['reactor_shield_l'])
reactor_shield.translate(App.Vector(0,0,P['nose_len'] + P['mid_len'] - 200.0))
```

```
# ========================
# Reactor + boquilla (base DFD)
# ========================
reactor = Part.makeCylinder(P['reactor_d']/2, P['reactor_l'])
reactor.translate(App.Vector(0,0,P['nose_len']+1200))
nozzle = Part.makeCone(P['rear_d']/2, P['rear_d'], 1000)
nozzle.translate(App.Vector(0,0,P['nose_len']+P['mid_len']+P['rear_len']))
reactor_full = reactor.fuse(nozzle)

# ========================
# Módulo hábitat
# ========================
hab = Part.makeCylinder(P['hab_d']/2, P['hab_l'])
hab.translate(App.Vector(0,0,P['nose_len']+P['mid_len']+500))

# ========================
# Cabina de mando
# ========================
cockpit = Part.makeCylinder(P['cockpit_d']/2, P['cockpit_l'])
cockpit.translate(App.Vector(0,0,50))
window = Part.makeSphere(P['window_r'])
window.translate(App.Vector(P['cockpit_d']/3,0,P['cockpit_l']/2))
cockpit_cut = cockpit.cut(window)

# ========================
# Tanques laterales y esféricos
# ========================
tankL = Part.makeCylinder(P['tank_r'], P['tank_l'])
tankL.translate(App.Vector(P['tank_off'],0,P['nose_len']+1000))
tankR = Part.makeCylinder(P['tank_r'], P['tank_l'])
tankR.translate(App.Vector(-P['tank_off'],0,P['nose_len']+1000))
sphereL = Part.makeSphere(P['sphere_r'])
sphereL.translate(App.Vector(P['sphere_off'],0,P['nose_len']+2500))
sphereR = Part.makeSphere(P['sphere_r'])
sphereR.translate(App.Vector(-P['sphere_off'],0,P['nose_len']+2500))
tanks = tankL.fuse(tankR).fuse(sphereL).fuse(sphereR)

# ========================
# Radiadores en sombra (reubicados hacia atrás)
# ========================
wingL = Part.makeBox(P['wing_span'], P['wing_th'], P['wing_l'])
wingL.translate(App.Vector(-P['wing_span']/2, -P['mid_d']/2-150,
P['nose_len']+P['mid_len']+P['wing_back_offset']))
wingR = Part.makeBox(P['wing_span'], P['wing_th'], P['wing_l'])
```

```
wingR.translate(App.Vector(-P['wing_span']/2, P['mid_d']/2+150,
P['nose_len']+P['mid_len']+P['wing_back_offset']))
wings = wingL.fuse(wingR)

# ========================
# Collar térmico y paravientos (deflectores)
# ========================
collarOD = P['mid_d'] + P['collar_d_delta']
collar = Part.makeCylinder(collarOD/2.0, P['collar_h']).cut(Part.makeCylinder((collarOD/2.0 -
P['collar_t']), P['collar_h']))
# Centrado en mitad del tramo medio
collar.translate(App.Vector(0,0,P['nose_len'] + P['mid_len']/2.0 - P['collar_h']/2.0))

# Deflectores longitudinales (placas) alrededor del mid
defs = []
for i in range(P['def_count']):
    ang = i * (360.0 / P['def_count'])
    d = Part.makeBox(P['def_l'], P['def_w'], P['def_t'])
    d.translate(App.Vector(-P['def_l']/2.0, -P['def_w']/2.0, P['nose_len'] + P['mid_len']/2.0 -
P['def_t']/2.0))
    # Rotamos alrededor del eje Z para distribuir como pétalos alrededor del collar
    # Luego desplazamos radialmente hasta el radio del collar
    baseR = collarOD/2.0 + P['overlap']
    d.Placement = App.Placement(App.Vector(baseR,0,0), App.Rotation(App.Vector(0,0,1), ang))
    defs.append(d)
deflectores = defs[0]
for d in defs[1:]:
    deflectores = deflectores.fuse(d)

# ========================
# Escotillas y acoplamientos
# ========================
dockL = Part.makeCylinder(P['dock_r'], P['dock_l'])
dockL.translate(App.Vector(P['dock_off'],0,P['nose_len']+1800))
dockR = Part.makeCylinder(P['dock_r'], P['dock_l'])
dockR.translate(App.Vector(-P['dock_off'],0,P['nose_len']+1800))
docking = dockL.fuse(dockR)

# ========================
# Sensores y cámaras externas
# ========================
sensor1 = Part.makeSphere(P['sensor_r'])
sensor1.translate(App.Vector(P['mid_d']/2+100,0,P['nose_len']+2000))
sensor2 = Part.makeSphere(P['sensor_r'])
```

```
sensor2.translate(App.Vector(-P['mid_d']/2-100,0,P['nose_len']+2000))
sensors = sensor1.fuse(sensor2)

# ========================
# Refuerzos internos
# ========================
beam1 = Part.makeCylinder(P['beam_r'], P['beam_l'])
beam1.translate(App.Vector(0,0,P['nose_len']))
beam2 = Part.makeCylinder(P['beam_r'], P['beam_l'])
beam2.translate(App.Vector(0,0,P['nose_len']+P['mid_len']))
beams = beam1.fuse(beam2)

# ========================
# Antena + parabólica
# ========================
mast = Part.makeCylinder(P['mast_r'], P['mast_l'])
mast.translate(App.Vector(P['mid_d']/2+100,0,P['nose_len']+P['mid_len']))
dish = Part.makeSphere(P['dish_r'])
# Simular plato comprimido (paraboloide aproximado): escalado no está directamente en Part,
# así que modelamos plato por corte simple
dish_flat = Part.makeCone(P['dish_r'], P['dish_r']-200.0, 180.0)
dish_flat.translate(App.Vector(P['mid_d']/2+100,0,P['nose_len']+P['mid_len']+P['mast_l']))
antenna = mast.fuse(dish_flat)

# ========================
# Tren de aterrizaje 4 patas
# ========================
legs = []
for angle in [0,90,180,270]:
    leg = Part.makeCylinder(P['leg_r'], P['leg_l'])
    leg.translate(App.Vector(P['mid_d']/2*math.cos(math.radians(angle)),
                             P['mid_d']/2*math.sin(math.radians(angle)),0))
    foot = Part.makeCylinder(P['foot_r'], P['foot_t'])
    foot.translate(App.Vector(P['mid_d']/2*math.cos(math.radians(angle)),
                              P['mid_d']/2*math.sin(math.radians(angle)),-P['foot_t']))
    legs.append(leg.fuse(foot))
landing_full = legs[0].fuse(legs[1]).fuse(legs[2]).fuse(legs[3])

# ========================
# Ensamblaje final con fusión robusta
# ========================
nave = hull
for part in [shield, hull_shield, reactor_shield, cockpit_cut, reactor_full, hab, tanks,
             wings, collar, deflectores, docking, sensors, beams, antenna, landing_full]:
```

```
    try:
        nave = nave.fuse(part)
    except Exception:
        # micro-solape para evitar coplanaridades estrictas
        part.translate(App.Vector(0,0,0.2))
        nave = nave.fuse(part)

nave_obj = add_obj(nave, "Nave_DFD_XL_Solar")
doc.recompute()
```