Diseño 3D de gabinete utilizando OpenScad

Rodriguez Guido

***Resumen*—El presente informe se corresponde con el trabajo propuesto por la cátedra de introducción al diseño asistido por computador (86.70) donde se pretende realizar un diseño 3D sobre un gabinete en el programa de diseño paramétrico Openscad.**

I. Introducción y objetivos

Se desea hacer un diseño 3D de un gabinete para un amplificador de audio Stereo con dos parlantes. A continuación se muestra un dibujo bosquejo del gabinete, el cual debe contener agujeros para la salida del sonido de cada parlante. La aplicación a utilizar para hacer el diseño debe ser OpensCad.

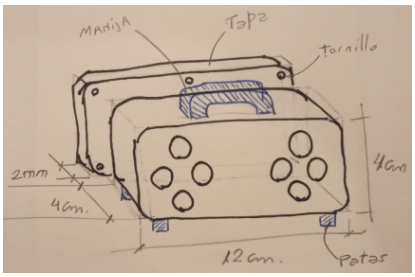


Figura 1: Bosquejo del gabinete

II. Desarrollo

En el proceso de creación del gabinete, se optó por un enfoque de diseño modular para construir cada componente del conjunto. A continuación, se presenta un desglose detallado del desarrollo del gabinete, destacando la contribución de cada módulo al producto final. Es importante mencionar que tanto los materiales como las funciones físicas mencionadas se presentan a modo ilustrativo.

II-A. Rubber feet - "Patas de goma"

En la parte inferior del gabinete se han agregado "patas de goma". Estas patas proporcionan estabilidad al gabinete y evitan que se deslice o se mueva de una forma indeseada. También actúan como amortiguadores, reduciendo la transmisión de vibraciones al entorno, lo que podría afectar la calidad de sonido.

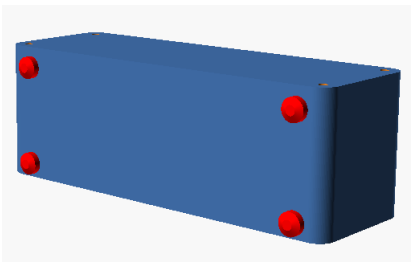


Figura 2: Patas de goma resaltadas en color rojo

Se presenta un *snippet* del módulo en cuestión, luego se detalla el funcionamiento interno del mismo.

```
module rubber_feet() {
    for (i = [0:1], j = [0:1]) {
        rotate([ 90, 0, 0 ])
            translate([ i * (box_l - box_thickness - 3.5 * box_rounding) +
                            box_min_thickness + screw_hole_support_r + box_rounding,
                        j * (box_w - box_thickness - 2 * screw_hole_support_r) +
                            screw_hole_support_r + box_min_thickness,
                        -box_w ]) {
            hull() {
                translate([ 0, 0, -box_thickness ])
                    cylinder(r = screw_hole_support_r,
                             h = box_thickness);
                translate([ 0, 0, -box_thickness - 1 ])
                    cylinder(r = screw_hole_r,
                             h = box_thickness + box_min_thickness);
            }
        }
    }
}
```

Figura 3: Código del módulo *rubber_feet*

- El bucle for se ejecuta dos veces, una para i en [0,1] y otra para j en [0,1]. Esto crea cuatro pies de goma en las cuatro esquinas del gabinete.
- Dentro del bucle for, se rota cada pie de goma 90 grados en el eje X para que se coloquen en la parte inferior del gabinete.
- Se utiliza la función `translate()` para calcular la posición precisa de cada pie de goma en función de las dimensiones del gabinete (*box_l*, *box_w*, *box_thickness*) y otros parámetros.
- Luego, dentro de la función `hull()`, se crean dos cilindros: uno para la base del pie de goma y otro para la parte superior. El cilindro de la base tiene un radio de *screw_hole_support_r* y una altura de *box_thickness*, mientras que el cilindro superior tiene un radio de *screw_hole_r* y una altura de *box_thickness* + *box_min_thickness*. Estos dos cilindros se combinan para formar cada pie de goma.

II-B. Handle - Manija

Se ha diseñado un asa funcional y ergonómica para el gabinete utilizando el módulo llamado *handle*. El diseño se logra mediante la extrusión de un polígono con puntos definidos, se complementa con dos cubos en los extremos y un cilindro en el centro y luego se redondean sus bordes mediante la función `minkowski`.

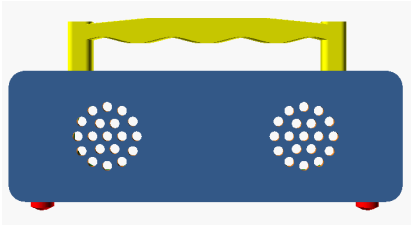


Figura 4: Asa resaltada en color amarillo

- El módulo handle() utiliza el operador minkowski() para “redondear” o suavizar el asa del gabinete.
- La forma del asa se crea en 3D y se rota 180 grados alrededor del eje X mediante rotate([180, 0, 0]). Esto coloca el asa alineada con la parte superior del gabinete.
- Se utiliza translate(tr) para definir la posición del asa en función de las coordenadas (x, y, z) especificadas de forma externa al módulo.
- El asa se crea principalmente como un polígono extruido en altura utilizando linear_extrude(). El polígono se genera mediante una serie de puntos dados por una función seno.
- Se añaden dos cubos utilizando translate() para formar los extremos del asa, uno en la posición (-4, -10, 0) y otro en la posición (78.75, -10, 0).

```
module handle(tr = [ 0, 0, 0 ]) {

    minkowski() {
        rotate([ 180, 0, 0 ])
            translate(tr) {
            linear_extrude(height = 3)
                polygon(points = [ for (i = [0:5:315])
                                    [i / 4, sin(4 * i) * 1],
                                    [ 78.75, 0 ], [ 78.75, 4 ], [ 58.75, 6 ],
                                    [ 20, 6 ], [ 0, 4 ], [ 0, 0 ] ]);
            translate([ -4, -10, 0 ]) cube(size = [ 4, 15, 3 ]);
            translate([ 78.75, -10, 0 ]) cube(size = [ 4, 15, 3 ]);
        }
        rotate([ 90, 0, 0 ]) cylinder(r = 2, h = 1);
    }
}

```

Figura 5: Código del módulo *handle*

II-C. *Screw holes - Agujeros de tornillos*

En el cuerpo del gabinete se colocan los agujeros de tornillos necesarios para fijar la tapa.

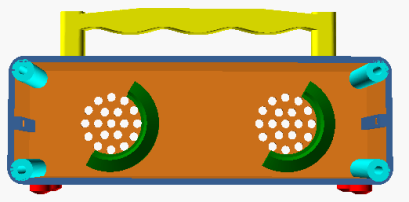


Figura 6: Agujeros de tornillos resaltados en color cian

- Se utiliza un bucle for anidado para iterar dos veces, una para la variable i y otra para la variable j. Esto crea cuatro ubicaciones para los agujeros de los tornillos en las esquinas del gabinete.
- El módulo translate() se utiliza para colocar cada agujero de tornillo en función de las dimensiones del gabinete (box_l, box_w) y otros parámetros.
- Dentro de la función difference(), se crean dos cilindros. El primer cilindro tiene un radio de screw_hole_support_r y una altura que abarca desde la parte superior del gabinete (box_thickness) hasta la parte inferior (box_h - box_thickness). Este cilindro representa el soporte del agujero del tornillo.
- El segundo cilindro tiene un radio de screw_hole_r - 0.15 y una altura de box_h + 0.1. Este cilindro representa el agujero del tornillo en sí. El valor -0.15 y 0.1 se utilizan para evitar problemas visuales como el ghosting.
- La función difference() se utiliza para restar el cilindro del agujero del tornillo del cilindro del soporte, creando así el agujero del tornillo en la posición deseada.
- El uso del booleano only_screws, por defecto con un valor false, permite reutilizar el módulo screw_holes() de manera versátil. Cuando only_screws se encuentra en su valor predeterminado (false), el módulo crea tanto el agujero del tornillo como el soporte del agujero en las ubicaciones especificadas. Esto es útil cuando se desea obtener la representación completa de los agujeros para tornillos en el diseño del gabinete. Sin embargo, si se establece only_screws en true, el módulo generará solo los tornillos, omitiendo la creación de los soportes. Esto tiene una gran utilidad al momento de generar los agujeros de la tapa realizando la diferencia en el correspondiente módulo.

```
module screw_holes(only_screws = false) {
    for (i = [0:1], j = [0:1]) {
        translate([
            i * (box_l - box_thickness - screw_hole_support_r * 2) +
                screw_hole_support_r + box_min_thickness,
            j * (box_w - box_thickness - screw_hole_support_r * 2) +
                screw_hole_support_r + box_min_thickness,
            0
        ]) {
            difference() {
                if (!only_screws) translate([ 0, 0, box_thickness ])
                    cylinder(r = screw_hole_support_r, h = box_h - box_thickness);

                // NOTE  -0.15 is a hack to make the screw holes fit
                // and 0.1 is also a hack to avoid ghost faces
                cylinder(r = screw_hole_r - 0.15, h = box_h + 0.1);
            }
        }
    }
}
```

Figura 7: Código del módulo *screw_holes*

II-D. *Speaker support - Soporte de parlante*

El módulo speaker_support() tiene como objetivo crear un soporte con una forma específica para el altavoz en el diseño del gabinete. A continuación, se explica su funcionamiento de manera resumida:

La pieza generada se corresponde con la resaltada en color verde en la figura 6.

- El módulo comienza con una transformación de translate(tr), que posiciona el soporte en el lugar deseado en función de las coordenadas especificadas en el argumento tr.
- Luego, dentro de la función difference(), se llevan a cabo varias operaciones para definir la forma del soporte del altavoz.
- Se utiliza rotate([0, 0, 60]) para rotar la figura en 60 grados alrededor del eje Z. Esto facilita la colocación del parlante al momento de ensamblar el producto.
- La función rotate_extrude() se utiliza para girar y extruir el polígono que define la forma del soporte. Se establece una convexity de 10 para controlar la suavidad de la extrusión y se limita el ángulo de extrusión a 180 grados.

  El polígono está definido mediante una serie de puntos que forman una figura que diseñé para soportar un parlante de una pulgada. El resultado es una forma tridimensional que actúa como soporte para el altavoz.

```
module speaker_support(tr = [ 0, 0, 0 ]) {

    translate(tr)
        difference() {
        rotate([ 0, 0, 60 ])
            rotate_extrude(convexity = 10, angle = 180)
                polygon(points = [
                    [ grill_r - 1, 5 ],[ grill_r + 1, 4 ],
                    [ grill_r + 1, 2 ],[ grill_r - 2, 2 ],
                    [ grill_r - 1, 0 ],[ grill_r + 4, 0 ],
                    [ grill_r + 4, 4 ],[ grill_r, 6 ],
                ]);
    }
}
```

Figura 8: Código del módulo *speaker_support*

II-E. Cantilever snap-fit - "Encastre de ganchos"

Los módulos canteliver_snap_fit_w_release_male() y canteliver_snap_fit_w_release_female() están diseñados para crear lo que se denomina en diseño industrial, conexión tipo "snap-fit" entre una parte macho y otra hembra, permitiendo que se unan y se liberen de forma manual sin la necesidad e utilizar herramientas. Para un mayor detalle técnico observar la bibliografía adjunta.

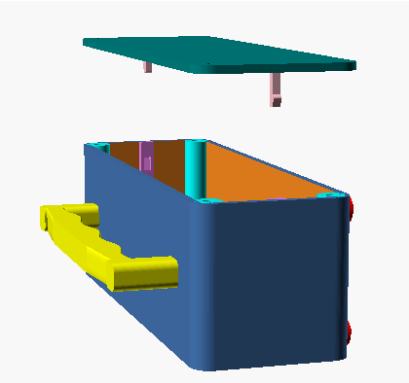


Figura 9: Encastre macho y hembra resaltados en color rosa y rosa salmón

A continuación se explica cómo funcionan de manera resumida:

- Con la ayuda de funciones de translación y rotación aplicadas sobre cubos se crea la componente macho. Esto incluye la creación de dos cubos que representan partes del componente macho en diferentes posiciones y rotaciones. Estos cubos se combinan para formar una estructura que encaja en la parte hembra del sistema de conexión.
- Nuevamente mediante la diferencia de un prisma generado a partir de la función cube() y la pieza macho se genera el cuerpo de la pieza hembra.

```
module canteliver_snap_fit_w_release_male(fillet_offset = 0) {
    minkowski() {
        union() {
            translate([ 2 * box_thickness + canteliver_l / 2,
                        box_w / 2 - canteliver_l + canteliver_fillet_h, box_h -
                        canteliver_h ])
                rotate([ 0, 2, 0 ]) cube(size = [ canteliver_l, canteliver_w, canteliver_h ]);
            translate([ box_thickness + canteliver_pivot_offset + canteliver_l / 2,
                        box_w / 2 - canteliver_l + canteliver_fillet_h,
                        box_h -
                        canteliver_h ])
                rotate([ 0, -30, 0 ]) cube(size = [ canteliver_l,
                                                    canteliver_w, canteliver_pivot_h +
                                                    fillet_offset ]);
        }
        rotate([ 90, 0, 0 ]) cylinder(r = canteliver_fillet, h = canteliver_fillet_h);
    }
}

module canteliver_snap_fit_w_release_female() {
    difference() {
        translate([ box_thickness,
                    box_w / 2 - 2 * canteliver_l,
                    box_thickness ])
            cube(size = [ box_thickness, 4 * canteliver_w, box_h - box_thickness ]);

        translate([ box_thickness + canteliver_fillet,
                    box_w / 2 - canteliver_l,
                    box_h - (canteliver_h / 2 + canteliver_fillet_h) ])
            rotate([ 0, 75, 0 ])
                cube(size = [ canteliver_l + 3,
                              canteliver_fillet_h + canteliver_w, canteliver_pivot_h +
                              canteliver_h ]);
    }
}
```

Figura 10: Código del módulo *canteliver_snap_fit_w_release_male* y *canteliver_snap_fit_w_release_female*

II-F. Box - Caja

El módulo box() desempeña un papel crucial en la creación del gabinete, ya que establece su estructura fundamental. Comienza por verificar que el radio de las esquinas redondeadas (box_rounding) sea adecuado para evitar problemas en el diseño. Luego, utilizando operaciones de minkowski(), se combina un cubo que representa las dimensiones generales del gabinete con un cilindro de radio box_rounding. Esto crea las esquinas redondeadas características del gabinete. Además, se añade otro cubo con dimensiones ajustadas para definir el grosor de las paredes del gabinete (box_thickness).

Para completar el diseño, se incorpora una "reja" para los parlantes en la parte frontal del gabinete, con la opción de elegir un patrón de enrejado, a lo que denomino 'exótico' pues se basa en las curvas de Lissajous o uno regular.

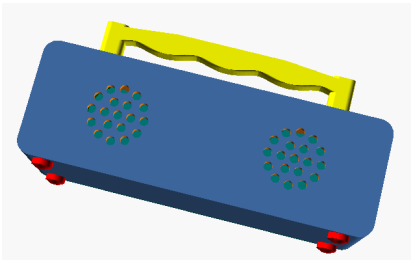


Figura 11: Caja resaltada en azul

- Primero, se realiza una verificación para asegurarse de que el valor de box_rounding (radio de las esquinas redondeadas) no sea excesivo, ya que podría causar problemas en el diseño.
- Luego, se utiliza minkowski() para redondear el gabinete.
- A continuación, se crea otro cubo, pero con dimensiones ligeramente reducidas para acomodar el grosor de las paredes del gabinete (box_thickness). Esto asegura que las paredes del gabinete sean del grosor deseado.

- Posteriormente, se itera a lo largo del gabinete, creando una serie de elementos que formarán la reja del altavoz en la parte frontal del gabinete. Dependiendo de si se ha seleccionado un patrón de parrilla ”exótico” o no, se invoca uno de los dos módulos correspondientes: speaker_grill_exotic() o speaker_grill_holes().

```
module box() {

    difference() {

        if (box_rounding >= box_w / 2)
            echo("box_rounding: The radius is too big | El radio es demasiado grande");

        minkowski() {
            translate([ box_rounding, box_rounding, 0 ]) cube(size = [
                box_l - 2 * box_rounding, box_w - 2 * box_rounding,
                box_h -
                box_min_thickness
            ]);
            cylinder(r = box_rounding);
        }

        translate([ box_thickness, box_thickness, box_thickness ]) cube(size = [
            box_l - 2 * box_thickness, box_w - 2 * box_thickness, box_h - box_thickness + 1
        ]);

        for (x_off = [box_l / 4:box_l / 2:box_l - box_l / 4]) {
            if (grill_exotic)
                speaker_grill_exotic(grill_r         = grill_r,
                                     grill_thickness = box_thickness,
                                     tr              = [ x_off, box_w / 2, 0 ]);
            else
                speaker_grill_holes(grill_r, box_thickness, tr = [ x_off, box_w / 2, 0 ]);
        }
    }
}
```

Figura 12: Código del módulo *box*

II-G. *Box cover - Tapa de la caja*

El módulo en cuestión es la tapa de la caja. El proceso de construcción es análogo al del módulo *box*. La representación en 3D se puede ver en la figura 9

```
module box_cover() {
    difference() {

        minkowski() {

            if (box_rounding >= box_w / 2)
                echo("box_rounding: The radius is too big| El radio es demasiado grande");

            box_thickness = box_thickness - box_min_thickness < box_min_thickness
                                ? box_min_thickness
                                : box_thickness - box_min_thickness;

            translate([ box_rounding, box_rounding, box_h ]) cube(size = [
                box_l - 2 * box_rounding, box_w - 2 * box_rounding, box_thickness
            ]);

            cylinder(r = box_rounding);
        }

        // Screw holes
        translate([ 0, 0, box_thickness ]) screw_holes(only_screws = true);
    }
}

```

Figura 13: Código del módulo *box*

II-H. *Speaker grill - “Rejilla” del parlante*

Para el diseño de la rejilla que cubre el parlante se proponen dos diseños distintos, el primero de ellos centrado en la funcionalidad y el otro en la estética del producto. Para este último propósito se utilizaron las curvas de Lissajous permitiéndonos esto crear una cantidad infinita de diseños posibles.

En lo que respecta al diseño funcional podemos observarlo en la imagen 4 donde se puede ver un patrón de rejilla circular generado por el módulo speaker_grill_holes().

- Se aplica una traslación (translate(tr)) para mover el sistema de coordenadas a la posición especificada por tr.
- Luego, se utiliza un bucle for para iterar a través de i en el rango [0:2 * grill_hole_r + grill_hole_r:r]. Esto permite variar la distancia radial desde el centro de la rejilla donde se crearán los agujeros.
- Dentro de este bucle, otro bucle for se utiliza para iterar a través de j en el rango [0:180 / floor(i / grill_hole_r):360]. Esto controla la disposición angular de los agujeros.
- Para cada combinación de i y j, se utiliza la función translate para desplazar un cilindro (cylinder) en el espacio tridimensional. Este cilindro representa un agujero en la parrilla de altavoz. Los parámetros h y r especifican la altura / espesor y el radio del agujero, respectivamente.

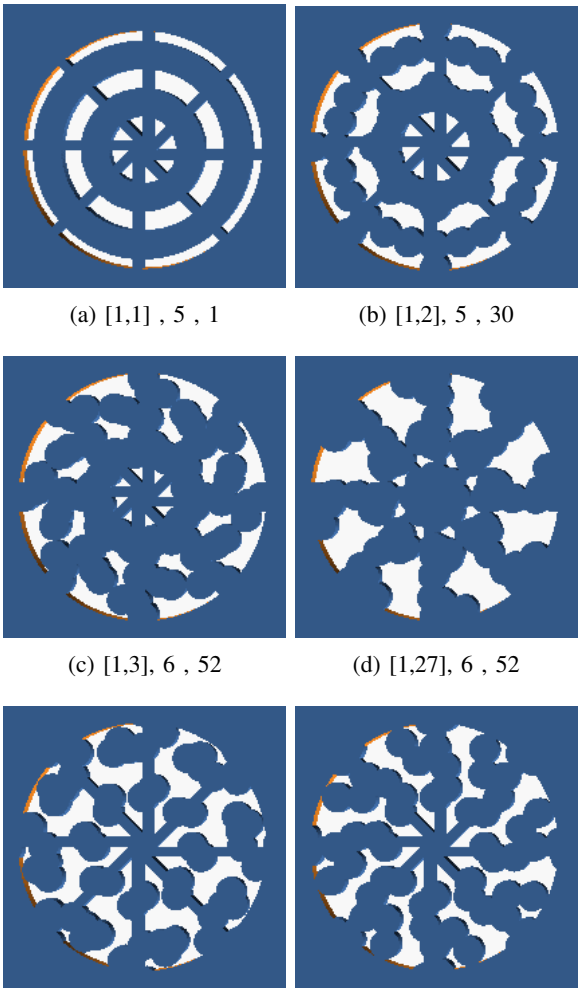
```
module speaker_grill_holes(r, thickness, tr = [ 0, 0, 0 ]) {
    translate(tr) {
        for (i = [0:2 * grill_hole_r + grill_hole_r:r])    //
            for (j = [0:180 / floor(i / grill_hole_r):360])
                translate([ i * cos(j), i * sin(j), -1 ])
                    cylinder(h = thickness + 2, r = grill_hole_r);
    }
}

```

Figura 14: Código del módulo *speaker_grill_holes*

De forma alternativa, el módulo speaker_grill_exotic funciona de la siguiente forma.

- Primero, se crea un cilindro que representa los límites del altavoz con un radio dado (grill_r) y un grosor (grill_thickness).
- Luego, se itera a través de un bucle for (rev = [0:grill_rev_step:360]), que rota el patrón Lissajous en incrementos angulares. En cada rotación, se crean estructuras de soporte en la rejilla.
- Luego, se itera a través de dos bucles for (i = [grill_exotic_spacing:grill_r + grill_support_w]) y for (j = [0:grill_exotic_figure_step:360]) para generar un patrón Lissajous en el plano XY. El patrón puede ser controlado mediante los parámetros definidos al inicio del archivo **grill_exotic_coef, grill_exotic_spacing, grill_exotic_figure_step**.
- La función lissajous_3D se llama dentro de este bucle para mapear este patrón 2D a uno 3D generando una terna ordenada, donde i y j controlan las coordenadas en el plano XY, y se utiliza un valor de 0 en el eje Z.
- Para cada punto calculado, se crea un cilindro (cylinder) con un radio de grill_hole_r y una altura de grill_thickness, que representa un “punto” en la rejilla siguiendo el patrón.

Aquí unos ejemplos de posibles patrones para diferentes configuraciones de **grill_exotic_coef, grill_exotic_spacing, grill_exotic_figure_step**.



(a) [1,1] , 5 , 1 (b) [1,2], 5 , 30

(c) [1,3], 6 , 52 (d) [1,27], 6 , 52

(e) [1,9], 10 , 48 (f) [1,21], 10 , 50

Figura 15: Diferentes configuraciones de **grill_exotic_coef, grill_exotic_spacing, grill_exotic_figure_step**

III. Conclusiones

Las principales conclusiones a las que abordo son las siguientes

- OpenSCAD es un programa de diseño 3D útil para la creación de piezas complejas, no así proyectos más complejos pues carece de herramientas propias de un programa de diseño 3D como podría ser FreeCAD, un árbol de relaciones, etc.
- Al momento de diseñar un producto que involucra la interacción con humanos se debe tener en cuenta la ergonomía del diseño.
- Se optó por un enfoque de diseño modular para construir cada componente del gabinete por separado. Esto permite una mayor flexibilidad en la creación y modificación de piezas individuales y facilita la reutilización en otros proyectos similares.
- El diseño de la rejilla del altavoz destaca por su flexibilidad estética. La capacidad para generar patrones de rejilla basados en curvas de Lissajous permite una variedad infinita de diseños posibles, lo que agrega un elemento estético distintivo al producto final.
- El diseño modular y la capacidad de personalizar parámetros como el grosor de las paredes del gabinete o los patrones de la rejilla hacen que este diseño sea versátil y adaptable a diferentes necesidades y preferencias.

Referencias

[1] Christoph Klahn, (2016, August 9). Design Guidelines for Additive Manufactured Snap-Fit Joints. Procedia CIRP. https://www.sciencedirect.com/science/article/pii/S2212827116303213

[2] Bayer. (n.d.). Snap-Fit Book Final 11-05 - Massachusetts Institute of Technology. https://fab.cba.mit.edu/classes/S62.12/people/vernelle.noel/Plastic_Snap_fit_design.pdf

APÉNDICE A
REPOSITORIO CON EL CÓDIGO DESARROLLADO

Figura S.1: Repositorio

APÉNDICE B
IMAGEN FINAL DE LA PIEZA

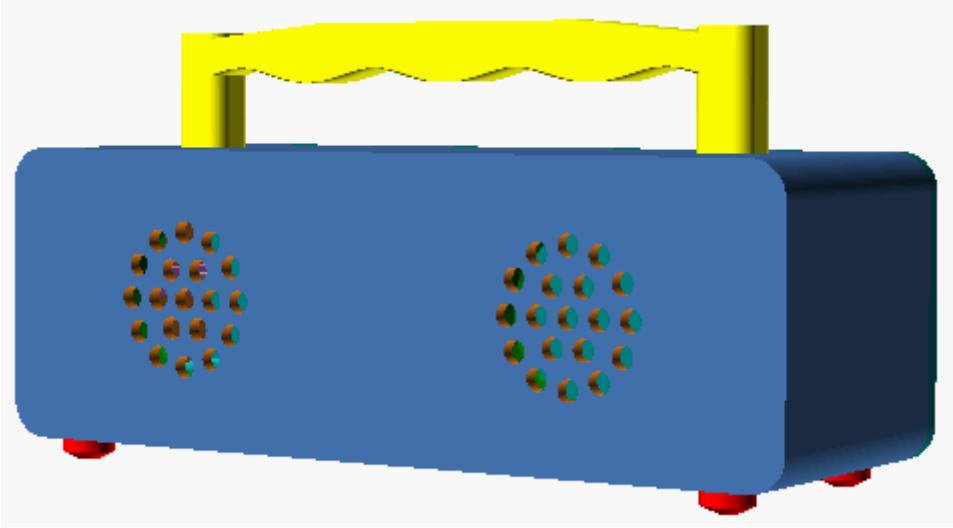


Figura S.2: Imagen de la pieza