UNIVERSIDAD
DE GRANADA

Informática Gráfica (curso 2025-26)

Guiones de Prácticas

Dpt. Lenguajes y Sistemas Informáticos.
ETSI Informática y de Telecomunicación.

Práctica 2.
Carga de modelos externos y normales.

2.1. Objetivos

El objetivo de esta práctica es:

- Comprender la representación de mallas triangulares.
- Aprender a cargar y visualizar modelos 3D externos en Godot.
- Entender la diferencia entre los tipos de sombreado de Godot (por vértice y por pixel).
- Aprender a generar normales mediante scripts cuando el modelo no las incluye.
- Crear una malla mediante revolución de un perfil 2D.

2.2. Requisitos previos

- Haber realizado la práctica 1.
- Crear un proyecto nuevo, creando el nodo raiz, una fuente de luz y la cámara orbital creada en la práctica anterior.
- Disponer de los mismos archivos `.gd` descritos en los requisitos de la práctica 1.
- Disponer de lps archivos `script_raiz.gd`, `utilidades.gd` y `donut.gd` que se necesitan para la práctica y se mencionan en las actividades.

2.3. Actividades

En las siguientes subsecciones se detallan las actividades a realizar, son las siguientes:

1. Añadir modo de visualización en alambre (*wireframe*)
2. Cargar modelos 3D en formato *glb*.
3. Cargar modelos 3D en formato *obj*.
4. Cálculo de normales de objetos *suaves* y tipos de sombreado de Godot.
5. Normales de objetos con aristas reales (no *suaves*).
6. Creación de mallas por revolución de un perfil.

2.3.1. Añadir modo de visualización en alambre (*wireframe*)

Cuando se trabaja con algoritmos de generación de mallas, es muy útil poder ver las aristas de los triángulos que forman la malla para depurar los algoritmos. Godot permite activar un modo de visualización en alambre (*wireframe*) que muestra las aristas de todos los objetos 3D de la escena, en lugar de los triángulos rellenos. En esta práctica se activará o desactivará al pulsar la tecla W.

A modo de ejemplo, en la figura 16 se muestra un modelo 3D (el donut que se menciona más adelante en este guión) en modo normal (izquierda) y en modo *wireframe* (derecha).

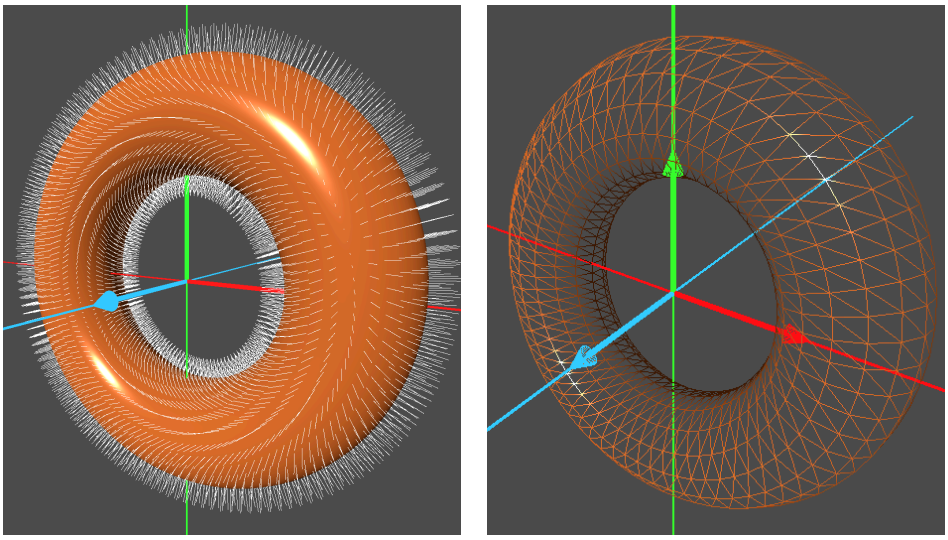

Figura 16: Donut en modo normal (izquierda) y en modo wireframe *(derecha).*

Da estos pasos para incorporar esta funcionalidad al proyecto:

1. Añade un script al nodo raiz de la escena, al añadirlo usa el archivo `script_raiz.gd` que tienes en los materiales de prácticas. Ese script ya tiene el código necesario para activar o desactivar el modo *wireframe* al pulsar la tecla W. El código se ve en la figura 17.

```
extends Node3D

var dibujar_aristas : bool = false

func _init():
  RenderingServer.set_debug_generate_wireframes(true)

func _unhandled_key_input( key_event ):
  if key_event.keycode == KEY_W and not key_event.pressed :
    dibujar_aristas = not dibujar_aristas
    var viewport = get_viewport()
    if dibujar_aristas:
      viewport.debug_draw = Viewport.DEBUG_DRAW_WIREFRAME
      print("Dibujar en modo aristas: activado")
    else:
      viewport.debug_draw = Viewport.DEBUG_DRAW_DISABLED
      print("Dibujar en modo aristas: desactivado")
```

Figura 17: Código para poder activar el modo wireframe *con una tecla*

2.3.2. Cargar modelos 3D en formato *glb*

El formato **glb** permite almacenar escenas completas con diferentes componentes, incluyendo sus texturas. Este formsto es la versión en binario del formato 3D **glTF** (cuyos archivos son JSON). Godot permite importar archivos *glb* de forma simple, y permite editar individualmente sus partes tras usar la opción para *convertir a escena editable* .

1. Descargar al menos un modelo en formato *glb*. Puedes hacerlo desde cualquier repositorio abierto como:

    - *Sketchfab*: https://sketchfab.com,
    - *Kenney.nl*: https://kenney.nl/assets o
    - *Poly Pizza*: https://poly.pizza.

2. Para importar el modelo en Godot, cópialo al directorio del proyecto. El modelo aparecerá en el panel *Sistema de archivos* (abajo a la derecha). Para que la carpeta del proyecto no aparezca con muchos archivos de modelos diferentes, crea una subcarpeta de ella llamada `modelos_3D`, dentro de esa carpeta, crea una subcarpeta distinta para cada modelo que descargues y ponle un nombre descriptivo del mismo. En cada subcarpeta de cada modelo aparecerán varios archivos relacionados con dicho modelo, los que descargues y los que crea Godot al importarlo.

3. Para organizar mejor el árbol de escena, crea un nodo de tipo `Node3D` como hijo del nodo raiz, y renómbralo como `ObjetosP2`. Este nodo servirá para tener como hijos todos los objetos de esta práctica y así distinguirlos fácilmente de los ejes, la cámara y la fuente de luz. A medida que vayas añadiendo objetos debajo de ese nodo, los verás todos ellos juntos en la escena. Para poder apreciar los objetos por separado, puedes poner como invisibles todos ellos menos uno, que será el que quieras ver en cada momento. Para cambiar la visibilidad de un nodo, haz click en el icono con forma de ojo que hay a la derecha del nombre del nodo en el panel del árbol de escena.

4. Para incorporar el modelo *glb* arrástralo desde el panel *Sistema de archivos* hasta el nodo `ObjetosP2`. Después verifica que la escala del modelo es apropiada (no aparece muy grande o muy pequeño en relación a los ejes), y si es necesario, edita su transformación y añaádele un escalado adecuado.

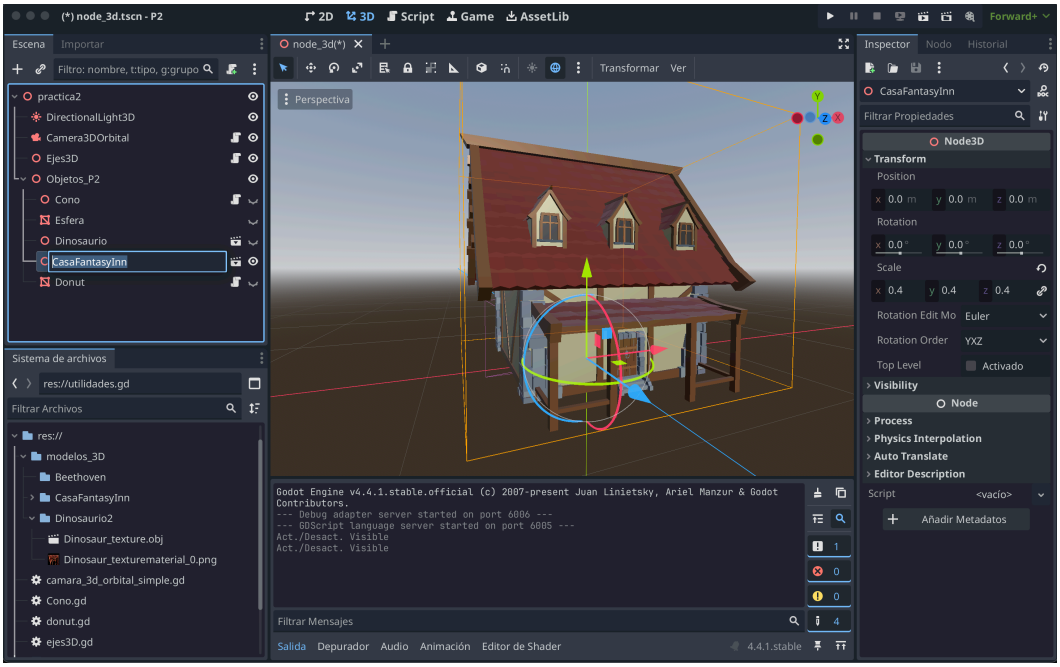

Figura 18: Ejemplo de carga de modelo con formato `glb`

2.3.3. Cargar modelos 3D en formato *obj*

En esta actividad podemos probar a añadir a la escena uno o varios modelos en formato *obj*, otro formato para modelos 3D muy extendido.

1. Descargar al menos un modelo en formato *obj*.
2. Descomprímelo y muevelo a su propia subcarpeta dentro de la carpeta `modelos_3D` (ponle a la carpeta un nombre descriptivo). Debe haber un archivo `.obj`, que contiene la geometría, un archivo `.mtl`, de definición de materiales, y una o varias texturas.
3. Crea un nodo `MeshInstance3D` en la escena, como nodo hijo de `ObjetosP2` y ponle un nombre descriptivo del modelo que has descargado.
4. Arrastra el archivo `.obj` desde el panel *Sistema de archivos* sobre el nuevo nodo.

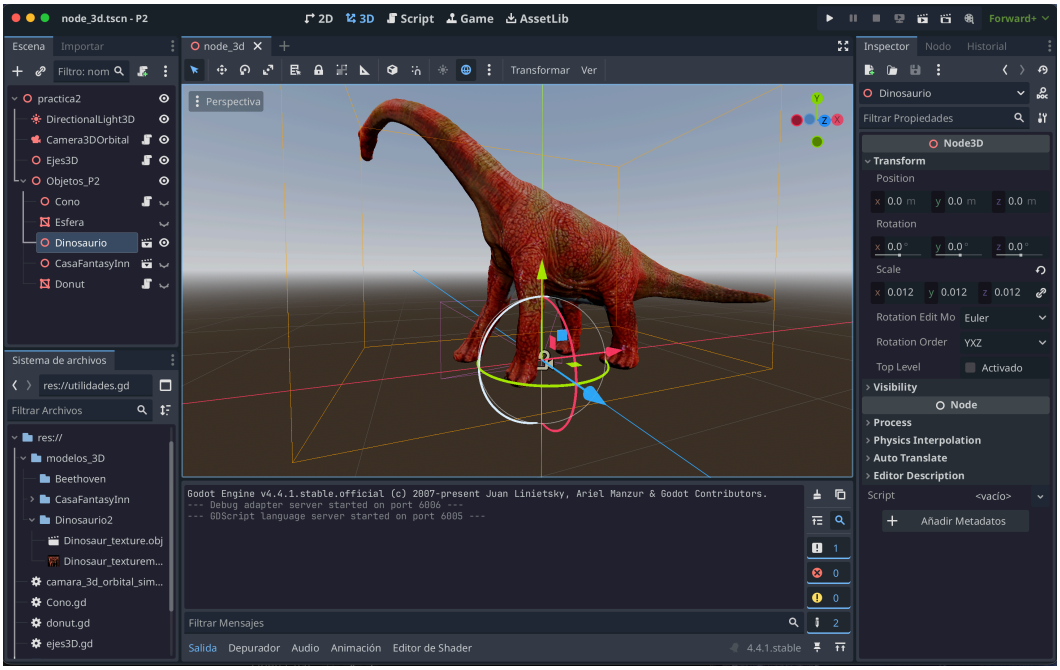

Figura 19: Ejemplo de carga de modelo con formato obj

```gdscript
extends MeshInstance3D

func _ready() -> void:

  ## Crear las tablas de vértices y triángulos de un Donut
  var vertices   := PackedVector3Array([])
  var triangulos := PackedInt32Array([])
  Utilidades.generarDonut( vertices, triangulos )

  var normales := Utilidades.calcNormales( vertices, triangulos )

  ## inicializar el array con las tablas
  var tablas : Array = []   ## tabla vacía incialmente
  tablas.resize( Mesh.ARRAY_MAX ) ## redimensionar al tamaño adecuado
  tablas[ Mesh.ARRAY_VERTEX ] = vertices
  tablas[ Mesh.ARRAY_INDEX  ] = triangulos
  tablas[ Mesh.ARRAY_NORMAL ] = normales

  mesh = ArrayMesh.new() ## crea malla en modo diferido, vacía
  mesh.add_surface_from_arrays( Mesh.PRIMITIVE_TRIANGLES, tablas )

  ## crear un material
  var mat := StandardMaterial3D.new()
  mat.albedo_color = Color( 1.0, 0.5, 0.2 )
  mat.metallic = 0.3
  mat.roughness = 0.2
  mat.shading_mode = BaseMaterial3D.SHADING_MODE_PER_PIXEL

  material_override = mat
```

Figura 20: Código del script *para el donut, en el archivo* `donut.gd`

```
func calcNormales( verts : PackedVector3Array,
                   tris  : PackedInt32Array ) -> PackedVector3Array :

    var nv : int = verts.size()  ## número de vértices
    var nt : int = tris.size()/3 ## número de triángulos

    ## Inicializa normales a cero
    var normales := PackedVector3Array([])
    for iv in nv:
      normales.append( Vector3.ZERO )

    ## Para cada triángulo, calcular su normal y sumarla
    ## a las normales de sus tres vértices.
    for it in nt :
      var t := Vector3i( tris[3*it+0], tris[3*it+1], tris[3*it+2] )
      var a := verts[t[0]] ;
      var b := verts[t[1]] ;
      var c := verts[t[2]] ;
      var normal_tri := (c-a).cross(b-a).normalized()
      for ivt in 3 :
        normales[t[ivt]] += normal_tri

    # Normalizar todos los vectores normales
    for iv in nv:
      normales[iv] = normales[iv].normalized()

    # Hecho
    return normales
```

Figura 21: Función que calcula las normales (en `utilidades.gd`)

2.3.4. Cálculo de normales de objetos *suaves* y tipos de sombreado de Godot.

En esta actividad veremos el uso de un algoritmo para calcular las normales de los vértices de una malla indexada de triángulos que aproxima una superficie suave (es decir, sin discontinuidades en la normal). En este tipo de mallas muchas veces se dispone de las coordenadas de los vértices, pero no de las normales, que son necesarias la iluminación.

Para calcular las normales, y puesto que sabemos que la malla aproxima una superficie suave, podemos asignar a cada vértice la normal promedio (normalizada) de las caras adyacentes a dicho vértice. El algoritmo que hace eso se encuentra implementado en la función `calcNormales` que tienes disponible en el archivo `utilidades.gd` (puedes ver el código en la figura 21). A modo de ejemplo, cuando se ejecuta el algoritmo para un objeto con forma de donut, las normales que produce se pueden observar en la figura 22.

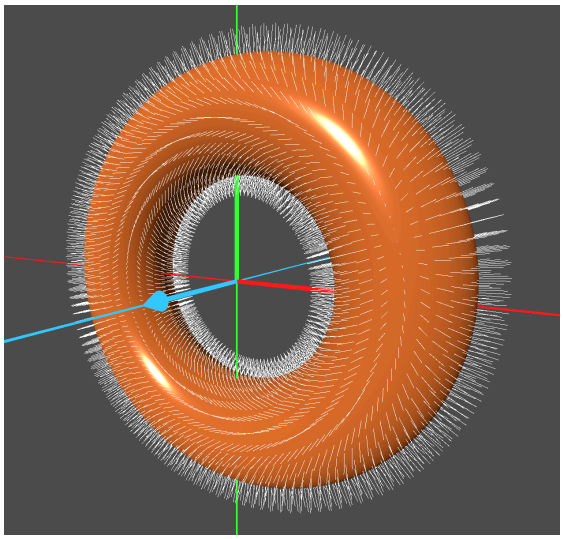

Figura 22: Donut con las normales calculadas por el algoritmo.

Además de los anterior, también veremos el efecto que tiene el **tipo de sombreado** en gráficos, término que se refiere a la forma de hacer el cálculo de la iluminación de una escena. Godot incorpora dos posibilidades:

- Calcular la iluminación en cada pixel donde se proyecta cada triángulo (usando la normal interpolada de los vértices). Se denomina **sombreado por pixel**.
- Calcular la iluminación en cada vértice de la malla, y luego interpolar el color resultante en cada pixel de cada triángulo. Se denomina **sombreado por vértice**. Es más rápido en comparación con el anterior (para mallas que no tengan muchísimos vértices), pero produce resultados menos realistas.

Para ilustrar este algoritmo, usaremos un objeto con forma de toroide (un donut), el cual se puede generar con un script sin las normales. Una vez generado probaremos a calcular las normales con el citado algoritmo, y luego probaremos también a cambiar el tipo de sombreado.

Las actividades a realizar son las siguientes:

1. Crear un archivo global de scripts con funciones que podrás llamar desde cualquier otro script. Lo puedes hacer desde el menú de Godot, en *Proyecto* → *Configuración del proyecto* → *Globales*. Ahí añade un nuevo objeto *autoload*, con nombre `Utilidades` y con el archivo asociado `utilidades.gd` (que tienes disponible en el material de la asignatura). Ese archivo tiene, entre otras cosas, el código de la función `calcNormales`, que implementa el citado algoritmo de calcular normales, y que puedes ver en la figura 21.
2. Crear un nodo hijo de tipo `MeshInstance3D`. Renómbralo como `Donut`.
3. Asóciale un script con las declaraciones y la función `_ready` que vemos en la figura 20 (está disponible en el archivo `donut.gd` que hay en los materiales de la asignatura). Compara este código con el de la práctica 1: ahora estamos definiendo un array de vértices (`vertices`) y un array de caras (`triangulos`), para poder calcular las normales en los vértices (vector `normales`). Ahí se fija el tipo de sombreado a *sombreado por pixel*.

4. Ejecuta. El donut aparece como se observa en la figura 23 (izquierda). Puedes observar que el material produce unos brillos especulares, que se ven correctos.
5. Cambia el modo de sombreado en el script del donut, usa sombreado por vértice en lugar de por pixel. Para ello modifica en el script el correspondiente atributo del material. El donut tendrá ahora el aspecto de la figura 23 (derecha). Puedes ver como ahora no se reproducen bien esos brillos. Razona a qué se debe esto.

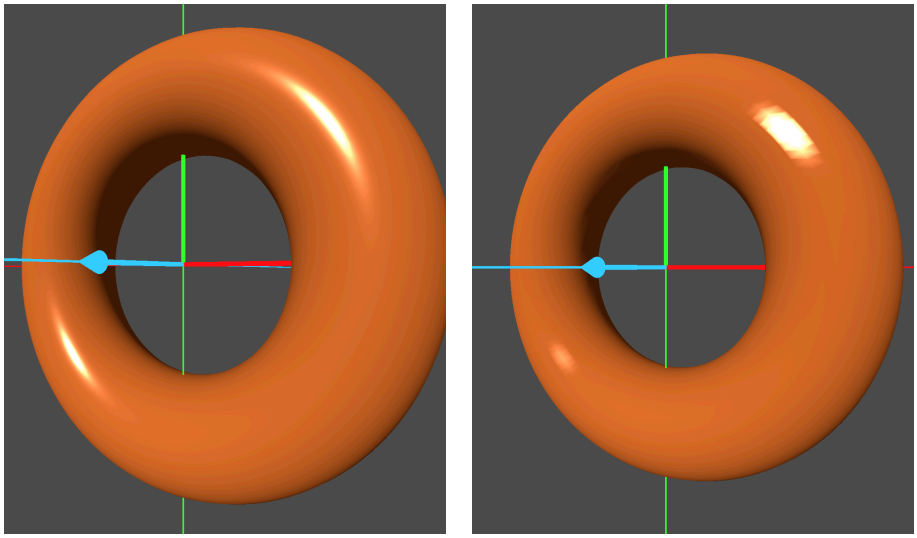

Figura 23: Donut con sombreado por pixel (izquierda) y por vértice (derecha).

2.3.5. Normales de objetos con aristas reales (no *suaves*).

En algunos casos queremos asignar normales a objetos que no son *suaves*, en el sentido de que tienen aristas reales, un ejemplo simple es un cubo real, o cualquier paralelepípedo real. Si usamos para estos objetos el algoritmo de cálculo de normales, obtendremos resultados inesperados, ya que el algoritmo asume que las aristas son suaves y no tiene en cuenta las discontinuidades de la normal en las aristas.

Para usar el algoritmo en estos objetos, será necesario modelas las aristas reales con aristas del modelo, pero en estos casos se hace necesario replicar los vértices, de forma que tengamos más de un vértice en una misma posición (en los extremos de algunas aristas), pero cada réplica del vértice tiene una normal distinta y es adyacente únicamente a triángulos con dicha normal.

Para el ejemplo del cubo, en principio podemos pensar que se puede modelar con un modelo de 8 vértices (correspondientes a las 8 *esquinas* reales de un cubo de 6 caras) y 12 triángulos (2 por cara), pero entonces el algoritmo de cálculo de normales no funcionará bien, como se ha descrito. Para que **el mismo algoritmo de generación de normales** funcione bien, necesitaremos un modelo con 12 triángulos (igual que antes) pero con 24 vértices, en concreto con 3 vértices en cada esquina real del cubo. Cada uno de esos tres vértices en la misma esquina (con la misma coordenada) tendrá asignada una normal perpendicular a una de las 3 caras del cubo que confluyen en dicha esquina (y será adyacente uno o dos triángulos coplanares en dicha cara).

Lo anterior se ilustra en la figura 24, donde se muestra un cubo con sus normales y sombreado. A la izquierda vemos el cubo de 8 vértices y a la derecha el de 24 vértices. En el de 8 la iluminación es incorrecta, ya que las normales se han generado en las direcciones de las 8 diagonales, lo cual no representa bien al objeto real. En el cubo de 24 vértices, sin embargo, vemos que en cada esquina del cubo hay 3 normales, cada una perpendicular a una de las 6 caras de dicho cubo.

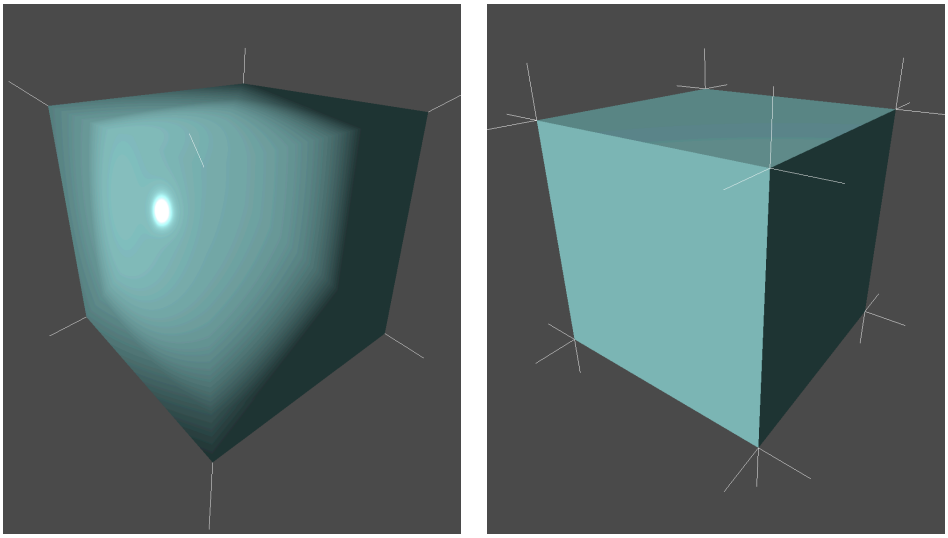

Figura 24: Cubo de 8 vértices (izquierda) y de 24 vértices (derecha).

Las actividades a realizar son las siguientes:

1. Crea un nodo hijo de `ObjetosP2`, de tipo `MeshInstance3D`. Renómbralo como `Cubo8vertices`.
2. Añade un script de forma que en el método `_ready` se definan los arrays de vértices y triángulos de un cubo de 8 vértices, se calculen las normales con la función que ya dispones, y se cree la malla y su material. Usa sombreado por pixel. Puedes basarte en el script del donut, pero definiendo los arrays de vértices y triángulos del cubo.
3. Crea un segundo nodo hijo de `ObjetosP2`, de tipo `MeshInstance3D`. Renómbralo como `Cubo24vertices`.
4. Añade un script de forma que en el método `_ready` se definan los arrays de vértices y triángulos de un cubo de 24 vértices, e igualmente se calculen las normales con la función que ya dispones, y se cree la malla y el material.
5. Compara los resultados de ambos objetos.

2.3.6. Creación de mallas por revolución de un perfil.

El objetivo de esta actividad es ejercitar las posibilidades de generación procedural de geometría 3D. Para ello se pide implementar un algoritmo el cual, a partir de un perfil 2D (es un array de vectores de 2 componentes, tipo `Vector2`, que representa una secuencia de puntos en el plano Z=0).

En la figura 25 se muestra un ejemplo de perfil que, al girar alrededor del eje Y, genera un modelo 3D con forma de peón de ajedrez. La coordenada X de los vértices (eje rojo)

debe ser siempre positiva. Suponemos que un vértice del perfil tiene como coordenadas (x, y), esaas cordenadas se interpretan como un punto del plano $z = 0$, con coordenadas $(x, y, 0)$. Después se rotarán un ángulo θ alrededor del eje Y, produciendo unas coordenadas (x', y', z), donde ya $z \neq 0$.

El perfil se puede crear explícitamente dando la secuencia de puntos del mismo, o bien proceduralmente, mediante algoritmos que generan el array con dicha secuencia.

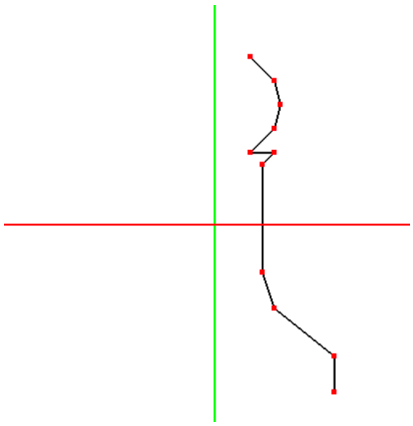

Figura 25: Ejemplo de una perfil que por revolución produce un modelo de un peón de ajedrez.

En la figura 26 se muestran (a la izquierda) las aristas de las dos primeras copias del perfil ya en 3D, y (a la derecha) todas las aristas del modelo 3D.

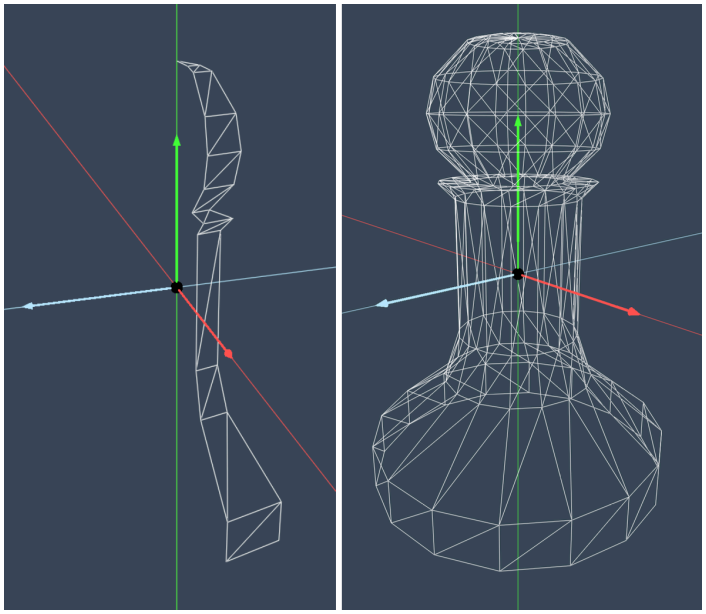

Figura 26: Aristas del objeto de revolución de un perfil.

Las actividades a realizar son las siguientes:

1. Implementa una función global que genere la malla por revolución alrededor del eje Y. La puedes situar en un archivo con definiciones globales que añades al proyecto, igual que has hecho con `utilidades.gd`, pero con otro script. La función aceptará como parámetros:

- El perfil 2D de entrada (un array de `Vector2`, de tipo `Array[Vector2]` o bien `PackedVector2Array`.
- El número número de copias del perfil que se generan al dar una vuelta completa.
- Dos arrays de salida (vacíos al llamar) que se rellenarán con los vértices y triángulos resultantes. Son de tipos `PackedVector3Array` y `PackedInt32Array`, respectivamente.

2. Crea un nodo hijo de `ObjetosP2`, de tipo `MeshInstance3D`. Renómbralo con un nombre descriptivo.
3. Añade un script a ese objeto. En el método `_ready`:
    1. Define un perfil 2D como lista de puntos en el plano X-Y.
    2. Invoca la función que has implementado para generar la malla por revolución.
    3. Invoca la función de cálculo de normales para obtener las normales de los vértices. Alternativamente, puedes pensar como calcular las normales de los vértices del perfil original y luego como *rotarlas* para el resto de los vértices en las copias del perfil.
    4. Crea el objeto `mesh` y su material, de forma semejante al script del donut.
4. Ejecuta y observa el resultado.

Experimenta con diferentes perfiles: un cilindro, un cono, una esfera, un vaso, una botella, etc… Compara el aspecto usando sombreado por pixel y por vértice.

2.4. Entrega de la práctica

- Subir la carpeta del proyecto `godot` con todos los archivos necesarios comprimido en un zip. El proyecto debe incluir todos los pasos realizados en la práctica.
- Incluir en la entrega un breve documento explicando la generación de objetos de revolución y como se le han asignado las normales.