



**UNIVERSIDAD
DE GRANADA**

Informática Gráfica (curso 2025-26)

Ejercicios adicionales para prácticas

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

Índice

1.	Ejercicios adicionales de la práctica 1.....	3
1.1.	Pirámide con base en L.....	3
1.2.	Polígono regular.....	3
1.3.	Estrella perpendicular al eje Z	4
2.	Ejercicios adicionales de la práctica 2.....	5
2.1.	Casa en el eje X.....	5
2.2.	Pirámide con base en forma de estrella.....	6
2.3.	Rejilla perpendicular al eje Y.....	6
2.4.	Torre de planta cuadrada.....	7
3.	Ejercicios adicionales de la práctica 3.....	8
3.1.	Grafo jerárquico en forma de estrella.....	8
3.2.	Grafo jerárquico de cubos.	9
4.	Ejercicios adicionales de la práctica 4.....	10
4.1.	Asignación de texturas al cubo de 24 vértices.....	10
4.2.	Pirámide con texturas	11
4.3.	Coordenadas de textura basadas en coordenadas polares o cartesianas	12
4.4.	Coordenadas de textura para objetos de revolución.....	13
5.	Ejercicios adicionales de la práctica 5.....	14
5.1.	Escena con esferas arrastrables.....	14

1. Ejercicios adicionales de la práctica 1.

Para estos ejercicios es necesario que añadas al proyecto de la práctica 1 la posibilidad de visualizar las aristas de las mallas, para eso usa lo que se dice al respecto en el guión de la práctica 2 (añadir un script al nodo raíz).

Igualmente ten en cuenta que al añadir diversos objetos, todos ellos serán, en principio, simultáneamente visibles en la escena 3D, así que debes de hacer visible en cada momento únicamente el que quieras visualizar.

1.1. Pirámide con base en L.

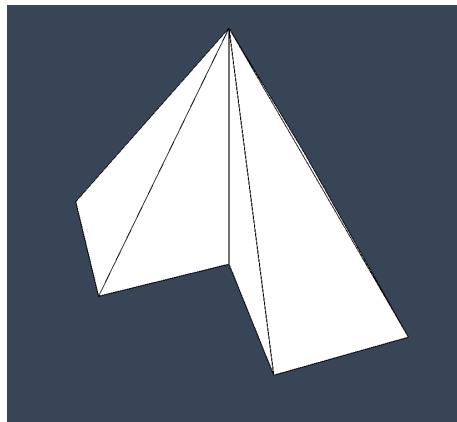


Figura 1 : Pirámide con base en L.

Crea un nodo de tipo `MeshInstance3D` y ponle nombre `MallaPiramideL`, es una malla indexada con forma de pirámide cuya base tiene forma de L. Está formado por 7 vértices e incluye los triángulos de la base y los 6 triángulos adyacentes al ápice. Usa un número mínimo de triángulos para formar la base.

En la figura 1 se observa como queda el objeto de la clase `MallaPiramideL` al visualizarse.

1.2. Polígono regular.

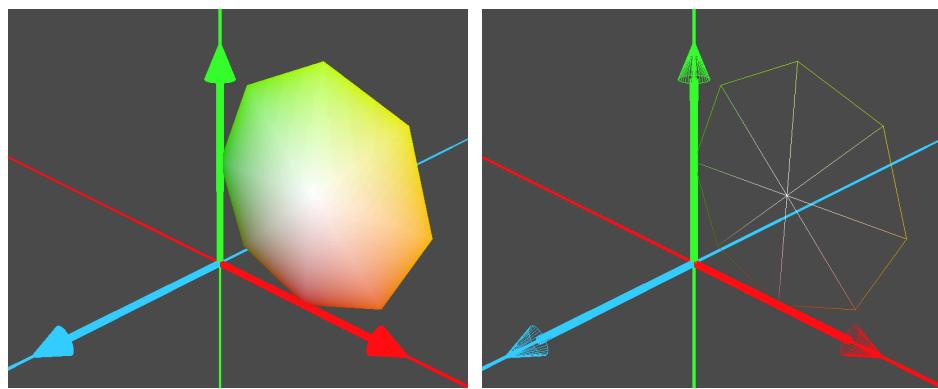


Figura 2 : Polígono regular (para $n = 8$). Vista en sólido y en wireframe

Usando el proyecto de la práctica 1, crea un nodo en Godot llamado `PolygonRegular`, de tipo `MeshInstance3D`.

Añádele un script. En el script, añade un método con esta declaración:

```
func ArrayMeshPoligonoRegular( n: int ) -> ArrayMesh :
```

Este método crea y devuelve un objeto `ArrayMesh` con las tablas de vértices, triángulos (índices) y colores de una malla indexada de triángulos con n triángulos y $n + 1$ vértices, en forma de polígono regular de n lados (en el plano perpendicular al eje Z). Los vértices tienen coordenadas entre 0 y 1 en X y en Y (y todos tienen Z igual a cero). El centro del polígono está en (0.5, 0.5) en X e Y, y los radios son de longitud 0.5. El vértice central tiene color blanco. El resto de vértices tienen colores cuyas componentes R, G y B coinciden con sus coordenadas X, Y y Z, respectivamente.

Añádele un método `_ready` al script en el cual se inicialize el objeto `MeshInstance3D`, para ello se le asigna a `mesh` el objeto obtenido llamado a la función indicada arriba. Usa un material sin iluminación.

En la figura 2 se observa como debe quedar este objeto una vez que se visualiza. Como variantes, se propone hacer la estrella en un plano perpendicular al eje X o al eje Y.

1.3. Estrella perpendicular al eje Z

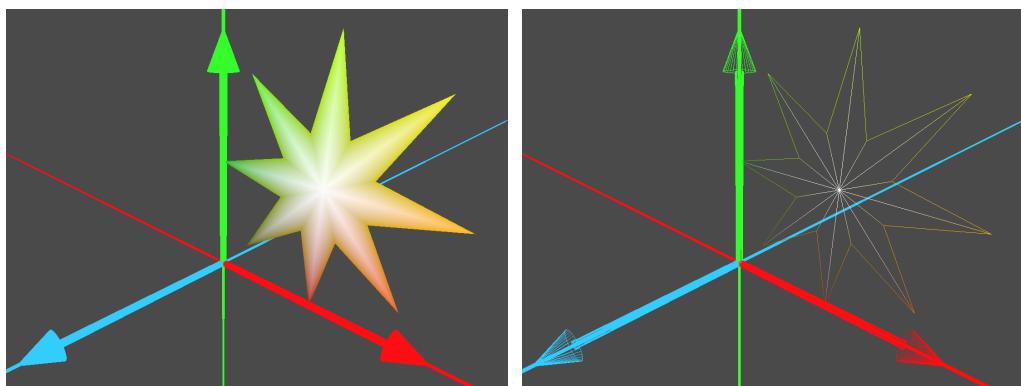


Figura 3 : Estrella perpendicular al eje Z (para $n = 5$). Vista en sólido y en wireframe

Usando el proyecto de la práctica 1, crea un nodo en Godot llamado `EstrellaZ`, de tipo `MeshInstance3D`.

Añádele un script. En el script, añade un método con esta declaración:

```
func ArrayMeshEstrellaZ( n: int ) -> ArrayMesh :
```

Este método crea y devuelve un objeto `ArrayMesh` con las tablas de vértices, triángulos (índices) y colores de una malla indexada de triángulos con $2n$ triángulos y $2n + 1$ vértices, en forma de estrella, plana (en el plano perpendicular al eje Z) y con n puntas. Los vértices tienen coordenadas entre 0 y 1 en X y en Y (y todos tienen Z igual a cero). El centro de la estrella está en (0.5, 0.5) en X e Y, y los radios hasta las puntas son de longitud 0.5. El vértice central tiene color blanco. El resto de vértices tienen colores cuyas componentes R, G y B coinciden con sus coordenadas X, Y y Z, respectivamente.

Añádele un método `_ready` al script en el cual se inicialize el objeto `MeshInstance3D`, para ello se le asigna a `mesh` el objeto obtenido llamado a la función indicada arriba. Usa un material sin iluminación.

En la figura 3 se observa como debe quedar este objeto una vez que se visualiza. Como variantes, se propone hacer la estrella en un plano perpendicular al eje X o al eje Y. Las aristas se han dibujado para mayor claridad, pero no forman parte del objeto visualizado.

2. Ejercicios adicionales de la práctica 2.

2.1. Casa en el eje X.

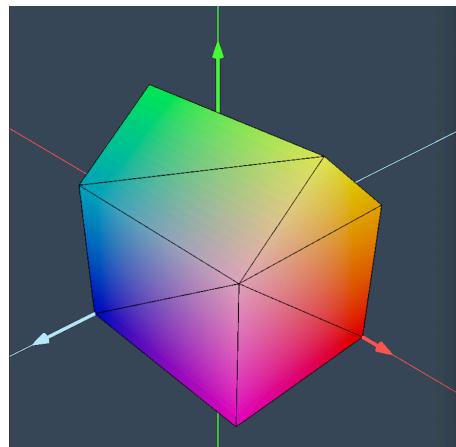


Figura 4 : Casa en el eje X.

Crea un nodo en el proyecto Godot de la práctica 2, de tipo `MeshInstance3D` llamado `CasaX`. En el método `_ready` incluye el código inicialmente de la pirámide una copia del cubo de 8 vértices pero luego adaptalo para (a) quitarle los dos triángulos de la base y los dos de la tapa (son los 4 triángulos perpendiculares al eje Y) y (b) añadirle en la parte superior 6 triángulos que forman una tejado a dos aguas, cuya arista superior es paralela al eje X. La casa es más alargada en el eje X que en el eje Z, pero tiene todas las coordenadas de todos los vértices entre 0 y 1.

Asígnale al nodo un material sin iluminación.

Cada vértice tiene un color RGB cuyas componentes son iguales a sus coordenadas XYZ. En la figura 4 se observa como queda el objeto de la clase `CasaX` al visualizarse.

2.2. Pirámide con base en forma de estrella.

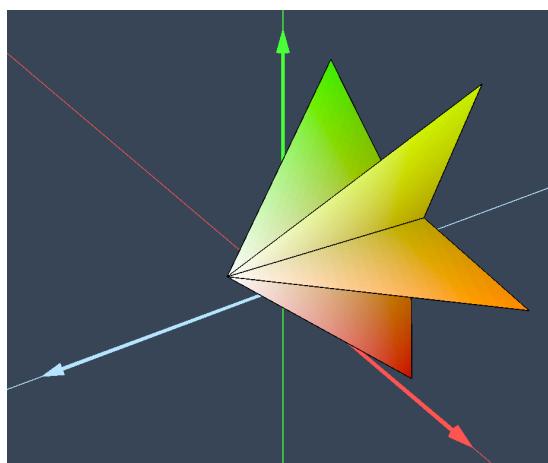


Figura 5 : Pirámide con base en forma de estrella.

Usando el proyecto de la práctica 1, crea un nodo en Godot llamado `PiramideEstrellaZ`, de tipo `MeshInstance3D`.

En el método `_ready` declara una constante entera $n (> 1)$. En esta función se inicializan las tablas de posiciones de vértices, triángulos y colores de una malla indexada con $4n$ triángulos y $2n + 2$ vértices, en forma de pirámide, con eje el eje Z, y cuya base es idéntica a la estrella descrita en el ejercicio adicional 1 de la práctica 1. El ápice de la pirámide es el único vértice adicional, tiene coordenadas $(0.5, 0.5, 0.5)$ y color blanco. En la figura 5 se observa el objeto.

Usa un material sin iluminación.

2.3. Rejilla perpendicular al eje Y.

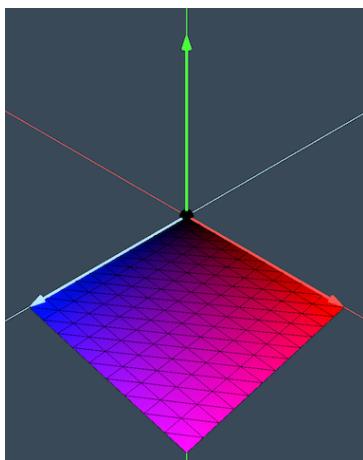


Figura 6 : Rejilla perpendicular al eje Y.

Crea un nodo en Godot llamado `RejillaY`, de tipo `MeshInstance3D`. Añadele un script.

En el script, implementa una función con esta cabecera:

```
func ArrayMeshRejilla( m: int, n: int ) -> ArrayMesh:
```

En esa función se debe crear un objeto `ArrayMesh` con malla indexada compuesta de una rejilla, donde cada celda es un rectángulo formado por dos triángulos adyacentes. Todos los vértices tienen coordenada Y igual a 0, y las coordenadas X y Z están entre 0 y 1. La figura completa es cuadrada con lado unidad. Hay $m - 1$ celdas en una dirección y $n - 1$ en la otra. Por tanto, en total tiene nm vértices y $2(n - 1)(m - 1)$ triángulos. Cada vértice tiene un color RGB cuyas componentes son iguales a sus coordenadas XYZ.

En la función `_ready` se debe inicializar el objeto `MeshInstance3D` con un material y con una malla (objeto `mesh`) obtenida llamando a la función `ArrayMeshRejilla`, con parámetros $m = 10$ y $n = 10$.

En la figura 6 se observa el objeto.

2.4. Torre de planta cuadrada.

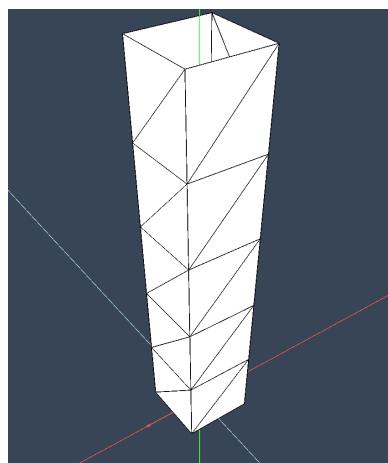


Figura 7 : Torre de planta cuadrada.

Crea un nodo en Godot llamado `Torre`, de tipo `MeshInstance3D`. En el método `_ready` declara una constante entera n (≥ 1) y crea una malla indexada con $4(n + 1)$ vértices y $8n$ triángulos, con estos requerimientos:

- La malla está formada por n secciones o plantas puestas una encima de la otra.
- Cada sección esta formada por cuatro caras cuadradas de lado unidad (como las caras laterales de un cubo). Cada una de esas caras son dos triángulos (en total 8 triángulos por planta).
- La torre no tiene los triángulos de la base ni el techo.
- No se puede usar el código para generar objetos por revolución.

En la figura 7 (derecha) se observa una vista del objeto, para $n = 5$.

3. Ejercicios adicionales de la práctica 3.

3.1. Grafo jerárquico en forma de estrella.

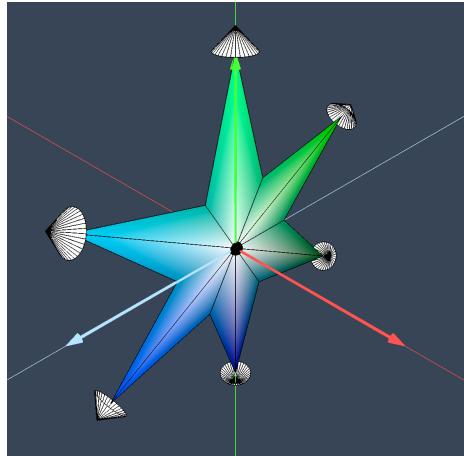


Figura 8 : Ejercicio adicional 1 de la práctica 3.

Copia tu proyecto de la práctica 3 en otro nuevo, en el nuevo crea un nodo de tipo `Node3D` y ponle de nombre `GrafoEstrellaX`, con estas especificaciones:

1. Añádele un `script`, en dicho script añade una copia de la función para generar un objeto de tipo `ArrayMesh` con **forma de cono** a partir de un perfil, por revolución entorno al eje Y (usa tu código de la práctica 2). Al código, añádele la generación de normales.
2. En el método `_ready` declara una constante n (entera, > 1), y úsala para crear una malla (un objeto `ArrayMesh`) en forma de estrella de n puntas, usa para ello la misma función del correspondiente ejercicio adicional de la práctica 1. A partir del `ArrayMesh`, crea un objeto `MeshInstance3D` que contiene dicha malla en forma de estrella, y añádelo como hijo del nodo `GrafoEstrellaX`.
3. En el método `_ready` crea n objetos de tipo `MeshInstance3D`, todos ellos comparten **una única instancia del `ArrayMesh` del cono**, usando la función que la genera (la debes de llamar una vez y obtener un único objeto `ArrayMesh`, es objeto debe ser compartido por todos los objetos `MeshInstance3D`). También deben compartir un **único objeto de tipo material**. Modifica la transformación de cada objeto `MeshInstance3D` de forma que sitúes un cono en cada punta de la estrella, y de forma que cada cono tenga radio de la base 0.14 y altura 0.15, y cuyo eje es el segmento que hace de radio desde el centro de la estrella a esa punta. Añade estos n objetos como hijos del nodo `GrafoEstrellaX`.
4. El objeto tiene un parámetro o grado de libertad, de forma que se puede rotar entorno al eje X (entorno al centro de la estrella). Si se activan las animaciones, gira a una velocidad de 2.5 vueltas por segundo.

En la figura 8 se observa el objeto jerárquico (sin iluminación).

3.2. Grafo jerárquico de cubos.

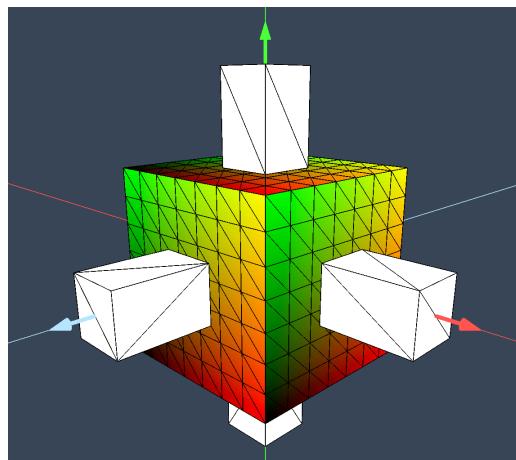


Figura 9 : Ejercicio adicional 2 de la práctica 3 (sin iluminación)

Usando el mismo proyecto del ejercicio adicional, crea un nodo nuevo de tipo `Node3D`, ponle de nombre `GrafoCubos`, y sigue estas instrucciones:

1. Añádele un *script*, en dicho script añade una copia de la función `ArrayMeshRejilla` para generar un objeto de tipo `ArrayMesh` con **forma de rejilla** que ya has escrito para el correspondiente ejercicio adicional de la práctica 2. A esa función añádele la generación de normales (todas son iguales, pero debe haber una por vértice).
2. En el mismo script, añade una función llamada `ArrayMeshCubo24` que genere un objeto `ArrayMesh` con la malla del cubo de 24 vértices con normales (copia el código del cubo de 24 vértices de la práctica 2).
3. En el método `_ready` de `GrafoEstrellaX`, crea un objeto `Node3D` que será el cubo central del objeto jerárquico con centro en el origen (ver figura 9). Este objeto está formado por 6 nodos hijos, de tipo `MeshInstance3D` (uno por cada una de las 6 caras del cubo), todos ellos compartiendo **un único objeto `ArrayMesh`** (obtenido con la función `ArrayMeshRejilla` descrita en el punto 1), y **un único objeto de tipo material**, pero cada uno con una transformación distinta para situarlo en la posición adecuada.
4. En el centro de cada una de esas seis caras hay un cubo más pequeño. Esos cubos se construyen como 6 nodos de tipo `MeshInstance3D`, pero todos ellos compartiendo **una única malla** obtenida con una llamada a `ArrayMeshCubo24` y **un mismo objeto material**. Cada uno de esos cubos está alargado en la dirección de la línea que va desde su centro al origen (son paralelepípedos).
5. El grafo tiene un único grado de libertad o parámetro. Cuando se anima, cada cubo pequeño rota entorno al eje que pasa por su centro y el origen (todos a la vez, ya que todos rotan con el mismo ángulo, al haber únicamente un grado de libertad).

Intenta crear un grafo con el mínimo número de nodos. Ten en cuenta que los cubos o paralelepípedos pequeños no entran dentro del grande.

En la figura 9 se observa el objeto jerárquico (los colores de los vértices o de los objetos en tu solución no tienen porque coincidir con los de la figura).

4. Ejercicios adicionales de la práctica 4.

Recuerda que para asignar un material (o un material con textura) a un nodo del árbol de escena puedes usar:

- **El editor** : seleccionando el nodo, y en el panel de sus propiedades, crear un nuevo *Standard Material 3D*. Si quieras asignarle una imagen de textura, copia la imagen a la carpeta del proyecto, verifica que aparece en el panel del sistema de archivos y que se ha importado sin problemas. Después, edita la propiedad *Albedo* ⇒ *Texture* del nodo y selecciona la opción *Carga rápida*, con la cual podrás elegir la imagen previamente cargada. Ver figura 10.
- **En código** : en los apuntes de teoría (en el archivo PDF de la sesión 2) se describe como asignar un material a un nodo, y como asociarle una textura a dicho nodo (usando la propiedad `albedo_texture`). El código para cargarlas se ha copiado en la figura 11. Con ese código, la asignación de la textura sería:

```
material.albedo_texture = CargarTexture("nombre_archivo_imagen.jpg")
```

4.1. Asignación de texturas al cubo de 24 vértices.

Añade al proyecto una función llamada `ArrayMeshCubo24_CCT`, inicialmente será una copia de la función `ArrayMeshCubo24` que genera los vértices, normales e índices del cubo de 24 vértices (es la función que se menciona en el ejercicio adicional del grafo de los cubos de la práctica 3). Extiende `ArrayMesh_CCT` para que además del resto de tablas, genere la

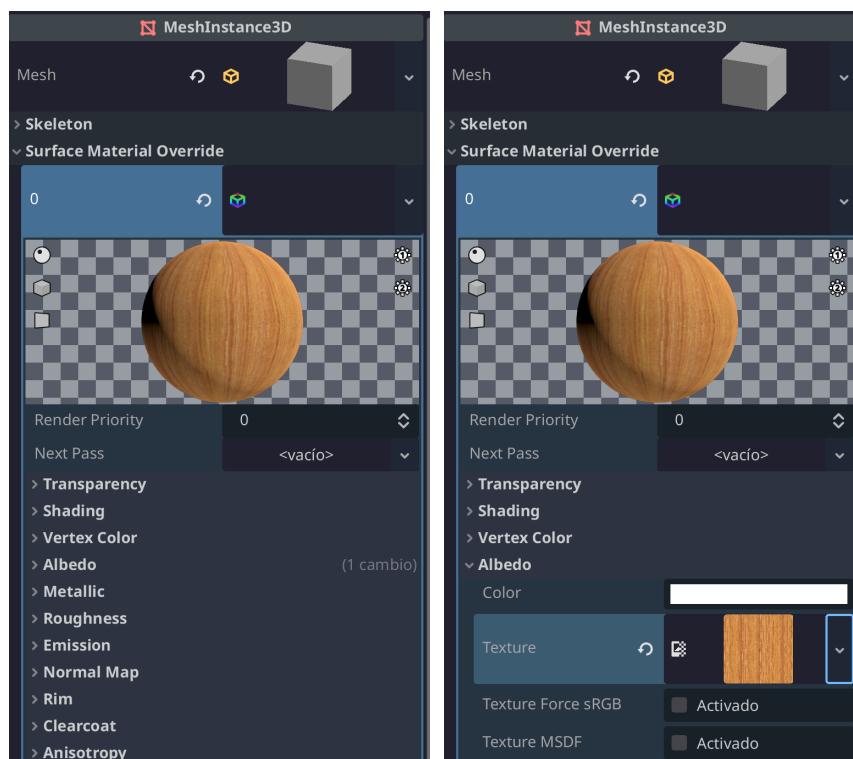


Figura 10 : Capturas del panel de propiedades de un nodo, mostrando las propiedades del material (izquierda), y la sección *Albedo* (derecha).

```

func CargarTextura( arch : String ) -> ImageTexture :

    ## crear un objeto 'Image' con la imagen
    var imagen := Image.new()
    assert( imagen.load(arch) == OK, "Error cargando '"+arch+"'." )

    ## crear un objeto 'ImageTexture' a partir del objeto 'Image'
    var textura := ImageTexture.create_from_image( imagen )
    print("Textura cargada desde archivo: '",arch,"'")

    ## devolver la textura
    return textura

```

Figura 11 : función para cargar una textura

tabla de coordenadas de textura, de forma que si se aplica una textura al objeto, la imagen de textura completa aparezca en cada una de las 6 caras del cubo. Usa la función para en el script de un nodo **MeshInstance3D** con un cubo con una textura.

A modo de ejemplo, puedes usar una textura con el logotipo de la Universidad de Granada. En la figura 12 se ve la textura y la apariencia del cubo con dicha textura.

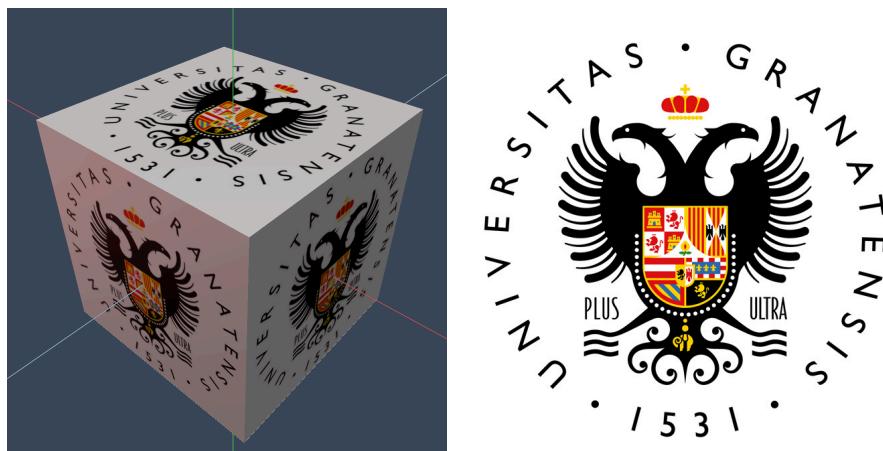


Figura 12 : La lata de coca-cola (izquierda) y su textura (derecha).

4.2. Pirámide con texturas

Crea un nodo **MeshInstance3D** con una pirámide de base cuadrada y 5 caras triangulares, usando un **ArrayMesh**. Crea la tabla de coordenadas de textura para que al aplicar una textura, la imagen de textura completa aparezca en la base de la pirámide, y una parte (triangular) de la misma aparezca en las caras laterales.

En la figura 13 se aprecia como debe quedar la textura del logotipo de la Universidad aplicada a este objeto.

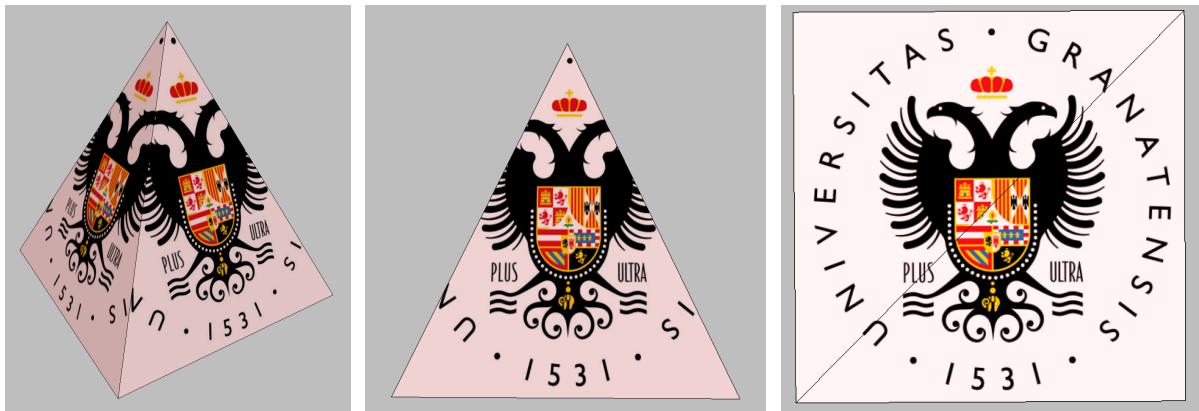


Figura 13 : Pirámide con textura: perspectiva (izquierda), alzado (centro), y base (derecha)

4.3. Coordenadas de textura basadas en coordenadas polares o cartesianas

Añade al proyecto una nodo de tipo `MeshInstance3D` con una malla indexada de triángulos en forma de disco plano, con un total de $n \times m$ pares de triángulos, y las aristas en los radios o en circunferencias concéntricas. Asignale coordenadas de textura a cada vértice de forma que una coordenada sea proporcional al ángulo y la otra al radio de ese vértice. En la figura 14 (a la izquierda) se aprecia una textura de cuadricula, y en el centro como queda aplicada al disco.

Haz una copia del nodo y crea otro igual al anterior, con el mismo script, después adapta el código que genera la tabla de coordenadas de textura, ahora debe calcular dichas coordenadas de forma que el disco se vea como aparece en la figura 14 a la derecha, en este caso las coordenadas (u, v) de textura son funciones lineales de las coordenadas (x, y) de cada vértice.

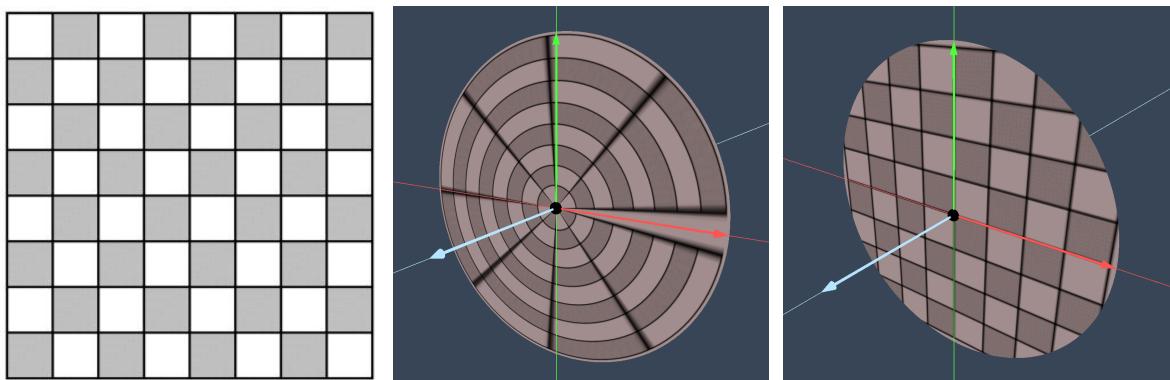


Figura 14 : Textura de disco (izquierda), aplicada al disco de dos formas distintas (centro y derecha).

Crea un nodo de tipo `MeshInstance3D` y copia el código que genera el Donut de la práctica 2. Usa la textura de rejilla o la de madera y aplícasela al Donut. Genera la tabla de coordenadas de textura en base a las coordenadas (x, y) o (x, z) de los vértices y observa el resultado. En la figura 15 puedes observar el Donut con las texturas.

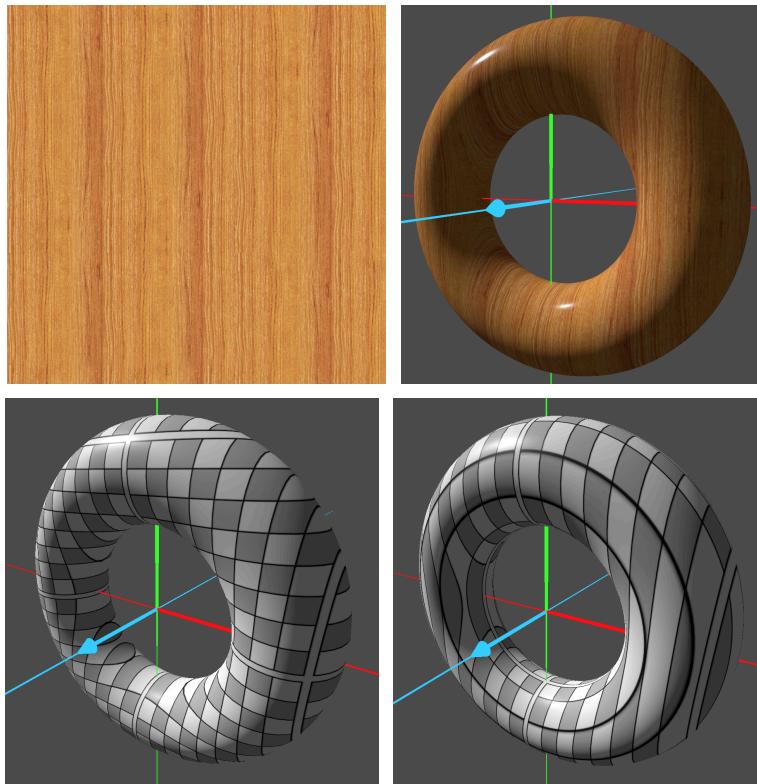


Figura 15 : Textura de madera (arriba izquierda), Donut con textura de madera (arriba derecha), Donut con textura de rejilla en XY (abajo izquierda), y en XZ (abajo derecho)

4.4. Coordenadas de textura para objetos de revolución.

El algoritmo de asignación de coordenadas de textura que se ha usado para la práctica 4 requiere que la textura tenga los mismos colores en las dos aristas verticales del rectángulo de la textura, de forma que al “envolver” la textura alrededor del objeto de revolución no se note la discontinuidad de las coordenadas de textura, de otra forma la textura no quedará bien en la última tira de triángulos.

Para evitar este problema, crea una copia del algoritmo de generación de objetos de revolución, de forma que tenga estos dos cambios respecto del implementado en las prácticas:

- Asegurate (si no es así ya) de que algoritmo genera una última réplica adicional del perfil, con un ángulo de 360° , que no está conectada a la primera réplica, con ángulo 0° . Esta última réplica adicional tiene los vértices en la misma posición que la primera, pero distintos índices. De esta forma, la malla tendrá una *topología* como la de la rejilla del ejercicio adicional 3 de la práctica 2 (figura 6), en lugar de una *topología cilíndrica*.
- La generación de la tabla de coordenadas de textura se hace a la vez que se generan los vértices. Supongamos que se genera el vértice v_{ij} , que es el j -ésimo vértice la i -ésima copia del perfil. A dicho vértice se le asignarán las coordenadas de textura (u, v) . El valor u es proporcional a i , es decir, es proporcional al ángulo de rotación del perfil que contiene a v_{ij} (v va de 0 a 1 cuando dicho ángulo va de 0 radianes a 2π radianes). El valor v es proporcional a la distancia desde el primer vértice del perfil (vértice v_{i0}) hasta el vértice v_{ij} (medida a lo largo del perfil), y va desde 0 (para el primer vértice del perfil) hasta 1 (para el último vértice del perfil).

A modo de ejemplo, intenta crear un objeto de revolución con la forma y textura de una lata de CocaCola, como la que se aprecia en la figura 16 (izquierda), para ello puedes usar una imagen como la que se ve en esa figura a la derecha.



Figura 16 : La lata de coca-cola (izquierda) y su textura (derecha).

5. Ejercicios adicionales de la práctica 5.

5.1. Escena con esferas arrastrables.

Crea un nodo de tipo `Node3D` llamado `EsferasArrastre`. Añádele un script y en el método `_ready` crea nodos hijos con forma de esferas, distribuidas en el espacio 3D en forma de rejilla o en forma radial (ver figura 17). Usa un par de constantes enteras n y m para definir el número de esferas, donde n puede ser el número de filas y m el número de columnas en la distribución en rejilla, o el número de radios y el número de esferas por radio en la distribución radial. Todas las esferas tienen el mismo radio r , y su centro está en el plano $y = 0$.

Configura el proyecto para que al hacer click con el ratón en el viewport (en concreto al bajar el botón), si una esfera se proyecta en el pixel donde se ha hecho click, dicha esfera pase estar seleccionada y cambie de color (ponla de color rojo). Si una esfera está seleccionada,

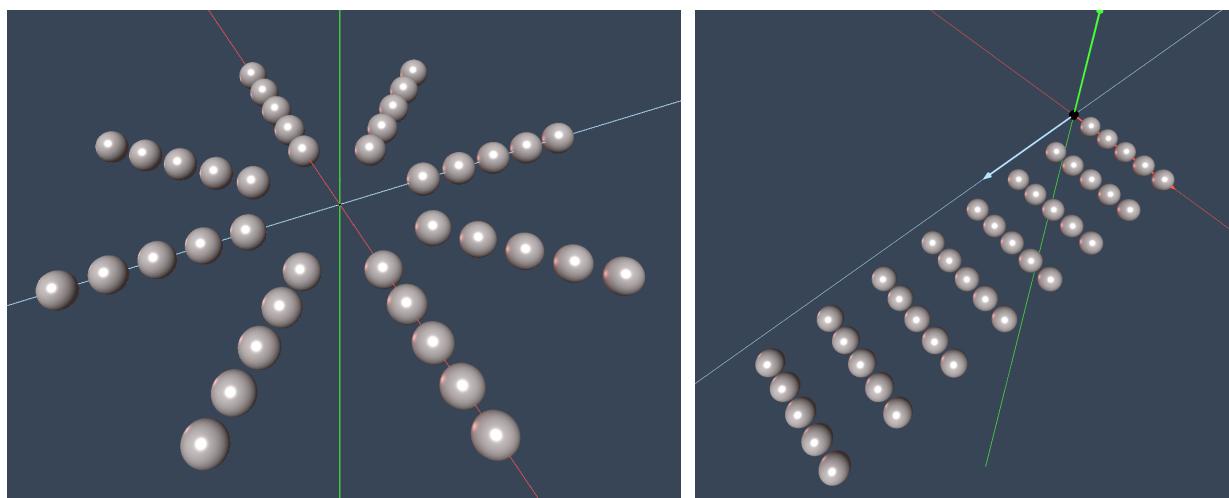


Figura 17 : Posibles disposiciones de las esferas arrastrables.

permanecerá seleccionada (de color rojo) hasta que se levante el botón del ratón (la misma esfera permanece seleccionada independientemente de si el ratón se mueve a otro pixel antes de levantar el botón). Una vez se levanta el botón del ratón, si había una esfera seleccionada, esta vuelve a su color original y deja de estar seleccionada.

Mientras una esfera permanece seleccionada, el usuario puede moverla en vertical (en el eje Y) moviendo el ratón hacia arriba y hacia abajo mientras mantiene el botón pulsado, es decir, la componente vertical de los desplazamientos del ratón se interpretan como movimientos de la esfera seleccionada, con lo cual el usuario puede arrastrarla para situarla a la altura que quiera.