Universidad de Granada

Doble Grado en Ingeniería Informática y Matemáticas

Informática Gráfica

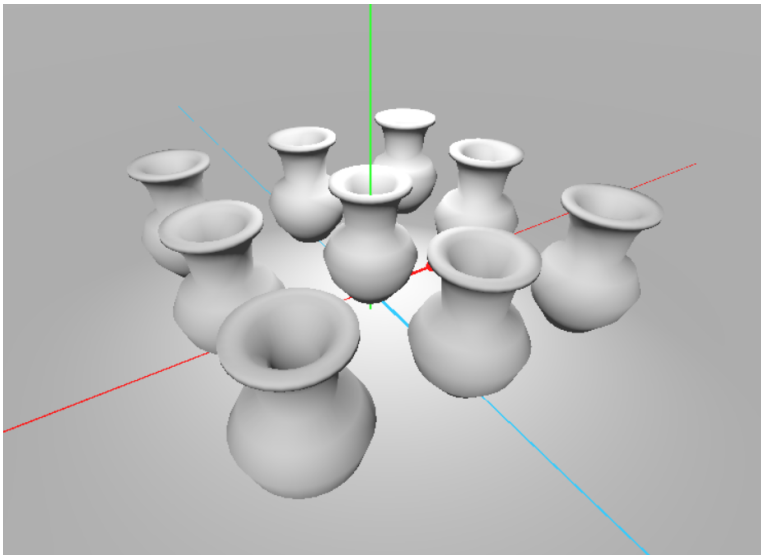
Informe Práctica 4

Autor:
Jesús Muñoz Velasco

Curso 2025-2026

1. Escena inicial con iluminación

Tras seguir los pasos establecidos en la práctica se ha llegado a la siguiente disposición:

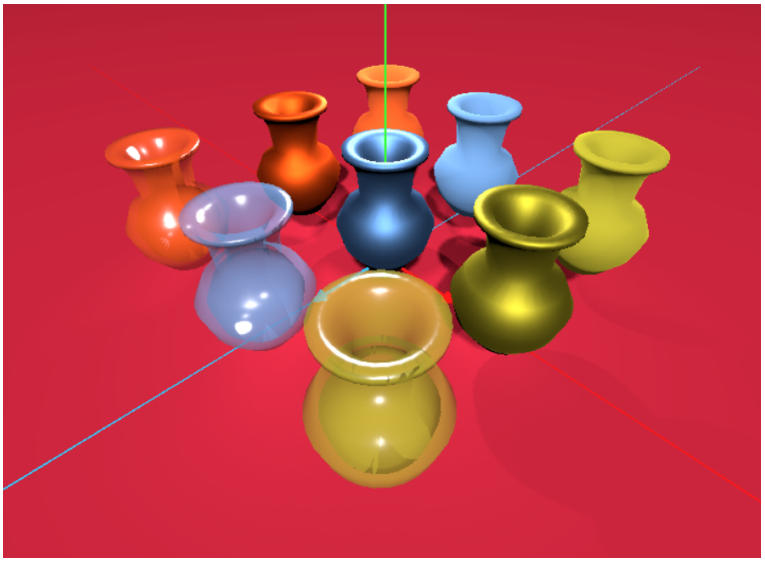


En ella he creado 9 instancias de `MeshInstance3D`, las he posicionado desde el editor y les he asignado a todas el mismo script `objeto-revolucion.gd` copiado de la práctica 3 y quitándole todo lo relativo al material (ya que así lo establecía el guión).

Después he creado las 3 luces que menciona el guión y ha quedado finalmente como se ve en la imagen (se pueden ver los parámetros consultando la práctica y se han ajustado de forma experimental).

2. Materiales

De nuevo tras seguir el guión se ha obtenido el siguiente resultado:



Para el suelo se ha establecido un color rojizo (en concreto el color `#e32b45`) y el resto de parámetros (Metallic, Specular y Roughness) a 0.

Para los colores del resto de filas se han elegido los siguientes colores:

Color	**Valor Hexadecimal**
Amarillo	`#c4b936`
Azul	`#6d9ed8`
Naranja	`#ff5f1a`

Ahora para cada columna se han intentado emular ciertas texturas jugando con los parámetros del material. Se detallan a continuación:

2.1. Cristal/Vidrio

Con el objetivo de jugar con la transparencia se ha intentado conseguir esta textura con la siguiente configuración:

Parámetro	**Valor**
Transparency	`Alpha`
A (en Albedo)	`165`
Metallic	`0.0`
Specular	`1.0`
Roughness	`0.26`

2.2. Metal

Esta vez se quiería probar la textura más metalizada para la fila intermedia y se ha aproximado con los siguientes parámetros:

Parámetro	**Valor**
Transparency	`Alpha`
A (en Albedo)	`165`
Metallic	`0.0`
Specular	`1.0`
Roughness	`0.26`

2.3. Arcilla

La última configuración buscaba simular la arcilla, un material sin reflejos y sencillo:

Parámetro	**Valor**
Metallic	`0.0`
Specular	`0.5`
Roughness	`19`

3. Texturas

Tras aplicar las texturas requeridas se ha llegado al siguiente resultado:

Donde lo único fuera del guión es el código para calcular las coordenadas de textura. Se ha optado por la siguiente solución:

```
vertices = tabla de posiciones de vertices
##
static func calcUV(vertices: PackedVector3Array) -> PackedVector2Array:
        var uvs := PackedVector2Array()
        var max_u = 1.0
        var max_v = 1.0

        # Calcular extremos de y
        var min_y : float = vertices[0].y
        var max_y : float = vertices[0].y
        for vert in vertices:
                if vert.y < min_y:
                        min_y = vert.y
                if vert.y > max_y:
                        max_y = vert.y

        var altura := max_y - min_y

        for v in vertices:
                #Calcular el valor del parámetro u
                var phi = atan2(v.z, v.x)
                var u = max_u*((phi / (2*PI)+0.5))

                #  Calcular el valor del parámetro v de forma aproximada en
                función de y
                var v_coord = max_v * ((min_y - v.y) / altura)
                var uv_coords = Vector2(1-u,1-v_coord)

                uvs.append(uv_coords)
        return uvs
```

En este caso se ha hecho en función de `y` (ya que por la forma de mi objeto de revolución tenía más sentido). De esta forma se han buscado los vértices que ocupan la posición más alta y la más baja (en el eje Y). Una vez hecho esto para asignar la altura `v` de la coordenada de textura se ha hecho una interpolación donde se le ha asignado a la coordenada más alta del perfil el valor mínimo de la coordenada de textura, es decir, 0 y a la coordenada más baja del perfil el valor máximo de la coordenada de textura, es decir, `max_v`. El resto de coordenadas se establecen linealmente entre estos 2 parámetros.

De esta forma consigo que el jarrón se vea por fuera con la textura completa (y el interior también). La otra posible solución (calculando el desplazamiento en la superficie) se ha implementado en los ejercicios adicionales. Extrayendo el fragmento relativo a esto sería el siguiente:

```
## Calcular distancias
var d_ac : Array =([0.0]) # distancia acumulada a lo largo del perfil
var d_total : float = 0

for j in range(n_points-1):
    d_total+= profile[j].distance_to(profile[j+1])
    d_ac.append(d_total)

## Genera vértices rotando el profile alrededor del eje y (duplicando los
primeros)
for i in range(n_copies+1):
    var angulo = (i * 2 * PI) / n_copies
    var rot = Transform3D(Basis(Vector3.UP, angulo), Vector3.ZERO)

    for j in range(n_points):
        var v := profile[j]
        var v_3D = rot * Vector3(v.x, v.y, 0)
        vertices.append(v_3D)
        uvs.append(Vector2((float(i)/n_copies), 1-(d_ac[j]/d_total)))
```

Ignorando todo lo relativo al cálculo de vértices (que es parte del ejercicio de revolucionar el perfil).