COMPUTACIÓN GRÁFICA

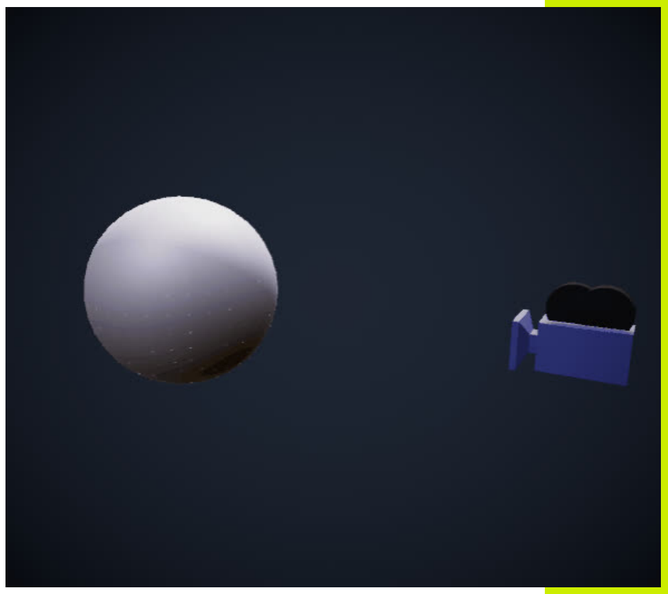
DETECCIÓN DE BORDES

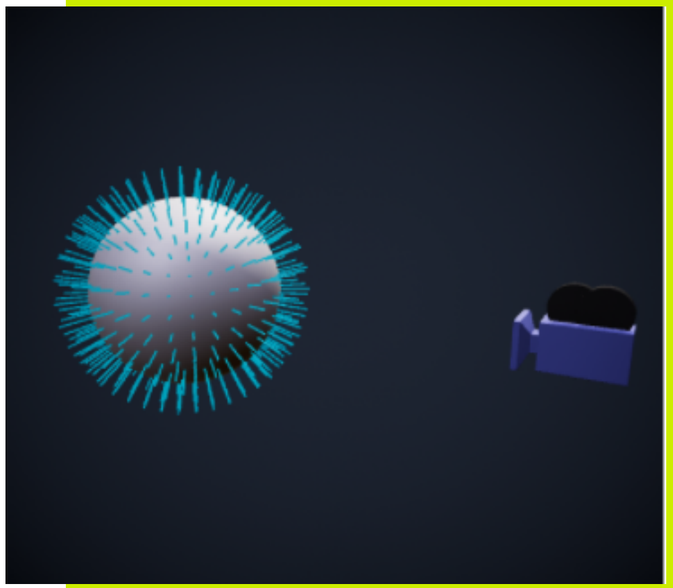
SERGIO PEÑALOZA HERRERA

Técnicas de superficie

Técnicas de imagen

Dirección de la cámara





Normales

Superficie

Se compara la dirección normal con la dirección de la cámara para hallar qué tan perpendiculares son estos vectores entre sí. entre más perpendicularidad haya, más posibilidad existe de que se trate de un borde.

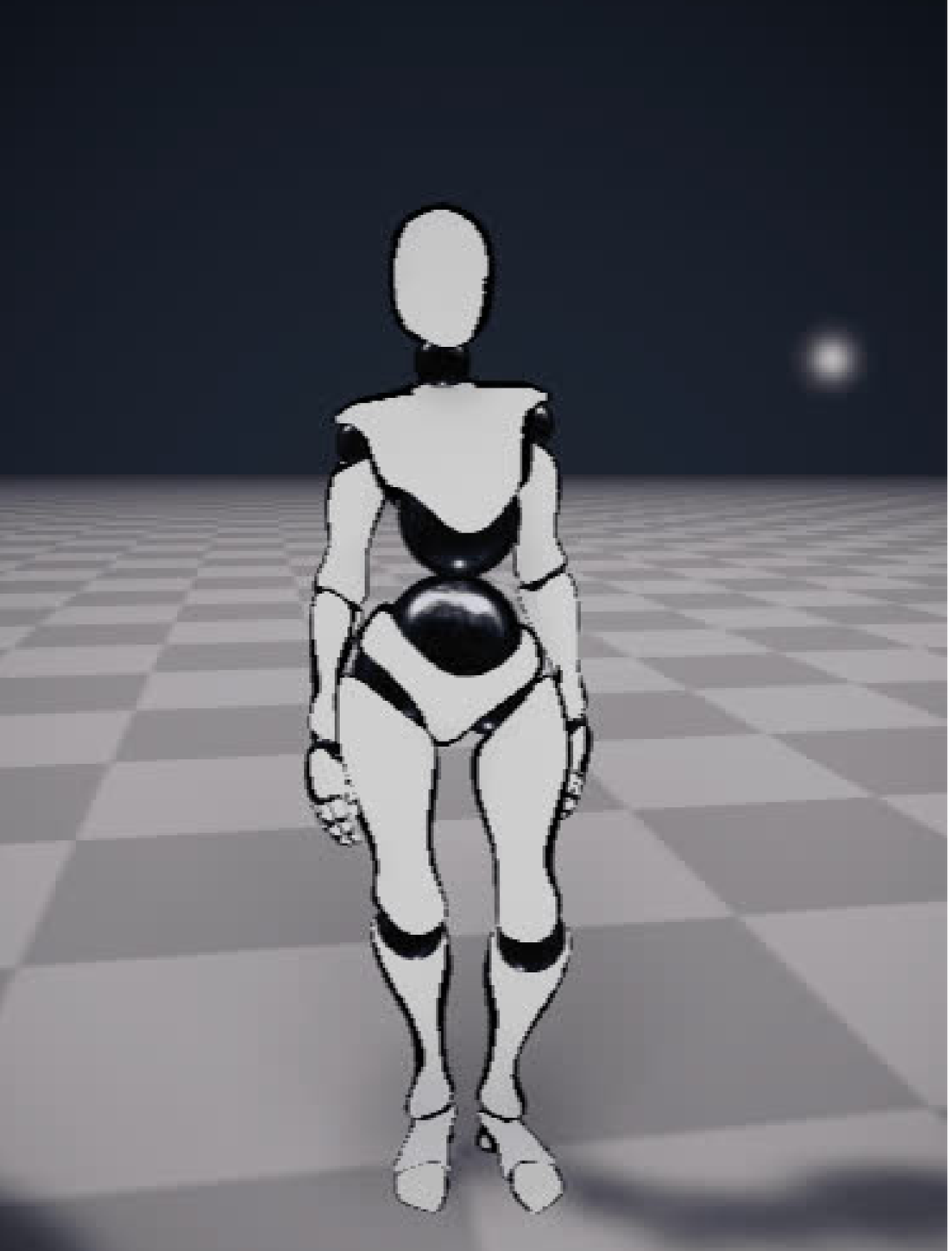
Obteniendo los datos necesarios

En surface shaders, se utilizan las palabras reservadas `viewDir` y `worldNormal` dentro de la estructura `Input` para guardar la dirección de la cámara y la dirección normal respectivamente. Una vez hecho esto podemos utilizar estos valores en la función `surf` .

```
struct Varyings
{
    float4 positionHCS  : SV_POSITION;
    float3 normalWS : NORMAL;
    float3 viewDirWS : TEXCOORD0;
};
```

Cálculo de Perpendicularidad

- La manera más intuitiva de encontrar la perpendicularidad entre 2 vectores es el producto punto, escrito en CG/HLSL como `dot(worldNormal , viewDir)` , de esta forma podemos obtener la máscara que la describe (Tambien conocida como Rim).
  Adicionalmente, se recomienda evitar valores negativos con `saturate(dot(worldNormal , viewDir));`



Punto de corte

- Para convertir el **Rim** en un borde duro podemos hacer uso de la función step(0.4, rim) cuyo primer parámetro es el punto de corte en el cual la función evaluará:
  si rim < 0.4, retorna 0
  Si rim > 0.4, retorna 1

Implementación

```
half4 _Color;
half4 _BorderColor;
half _BorderSize;

struct Varyings
{
    float4 positionHCS  : SV_POSITION;
    float3 normalWS : NORMAL;
    float3 viewDirWS : TEXCOORD2;
};

Varyings vert(Attributes IN)
{
    Varyings OUT;
    OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
    OUT.normalWS = TransformObjectToWorldNormal(IN.normalOS);
    OUT.viewDirWS = GetCameraPositionWS() - TransformObjectToWorld(IN.positionOS.xyz);
    return OUT;
}

half3 frag(Varyings IN) : SV_Target
{
    const float3 normal = normalize(IN.normalWS);
    const float3 viewDir = normalize(IN.viewDirWS);
    //Not actually an accurate fresnel reflection model
    const float fresnel = saturate(dot(normal, viewDir));
    const float border = step(_BorderSize, fresnel);
    return lerp(_BorderColor,_Color, border);
}
```

Pros y Contras

01

- La técnica no requiere de buffers extra.
- No se necesitan muchos recursos extra para realizar los cálculos necesarios.
- Simple de implementar y entender.

02

- Muy propensa a artefactos visuales.
- Irregularidades debido a que los valores dependen del modelo en lugar de ser constantes.