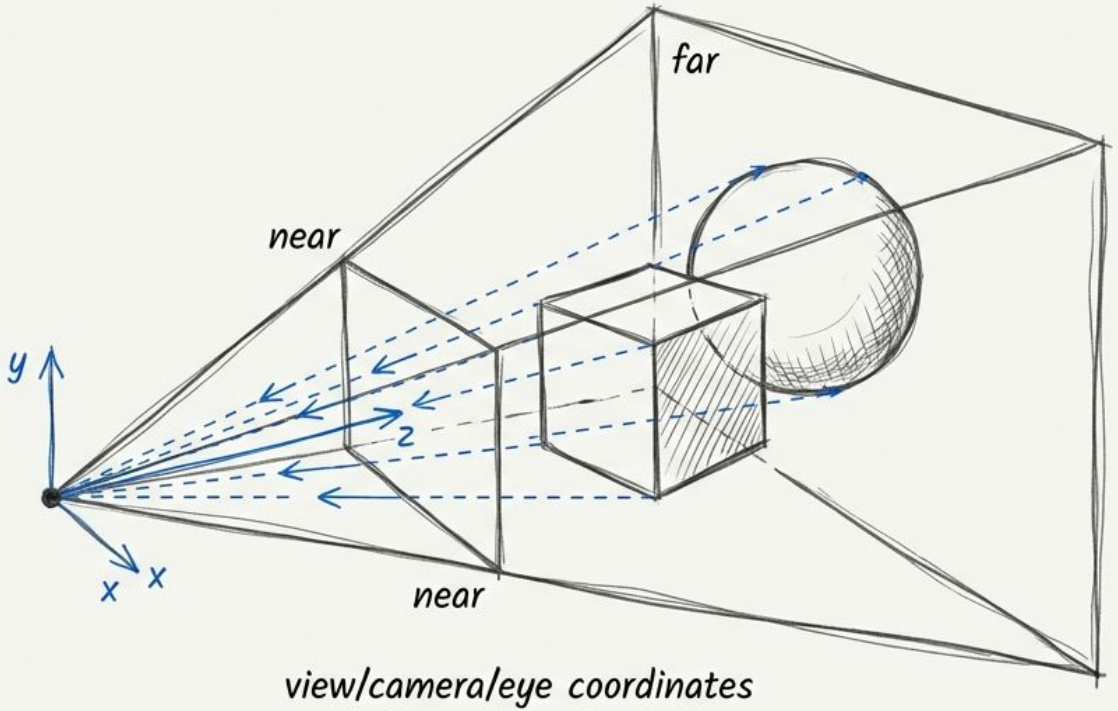
6. Depth Buffer y Visibilidad

Resolución de conflictos de profundidad en el pipeline de renderizado.



Agenda

1. Concepto y Necesidad
2. El Pipeline Gráfico
3. Matemáticas y Precisión
4. Artefactos y Motores Modernos

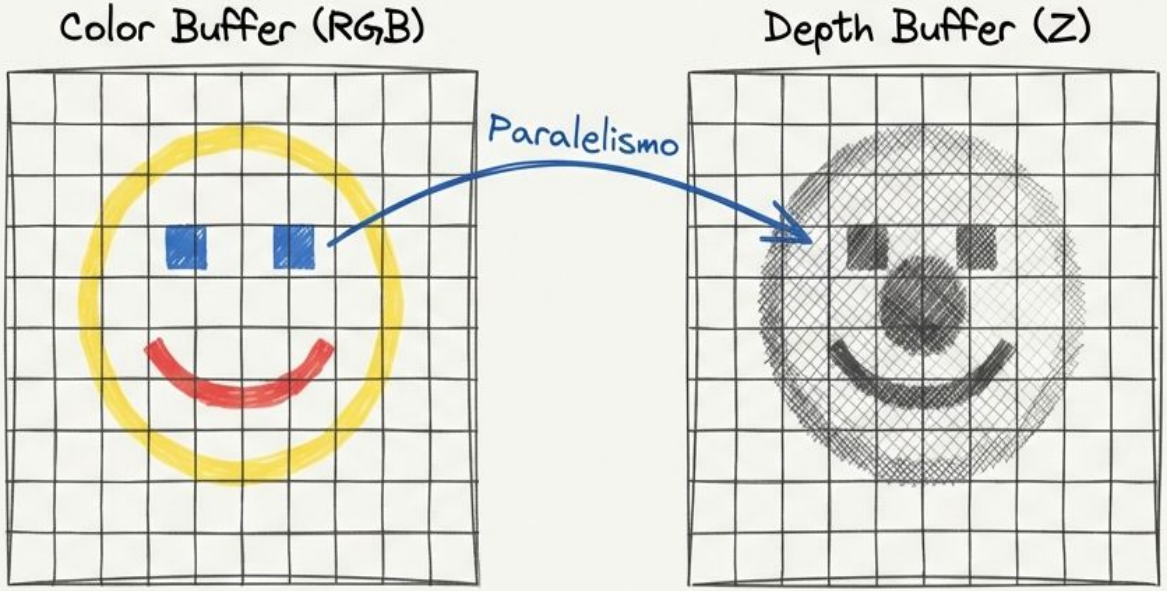
Equipo Completo: Brayan Muñoz, Álvaro Romero, Juan C. Lopez, Oscar Martinez

01

El Problema de la Visibilidad

- ☐ - Oclusión en la rasterización
- ☐ - Por qué falla el ordenamiento manual (Painter’s Algorithm)
- ☐ - Introducción al Z-Buffer

¿Qué es el Z-Buffer?



- Definición: Búfer de pantalla que almacena la profundidad (z) de cada fragmento.
- Resolución: Generalmente 24 bits (o 32 bits float).
- Propósito: Resolver visibilidad por píxel, no por objeto.
- Inicialización: Se limpia cada frame al valor 'lejos' (1.0).

El fallo del "Painter's Algorithm"

Superposición Cíclica

Painter's Algorithm: Dibujar de atrás hacia adelante (Back-to-Front).

La Falla Fundamental:

1. Intersecciones de geometría.
2. Ciclos de superposición (**A > B > C > A**).

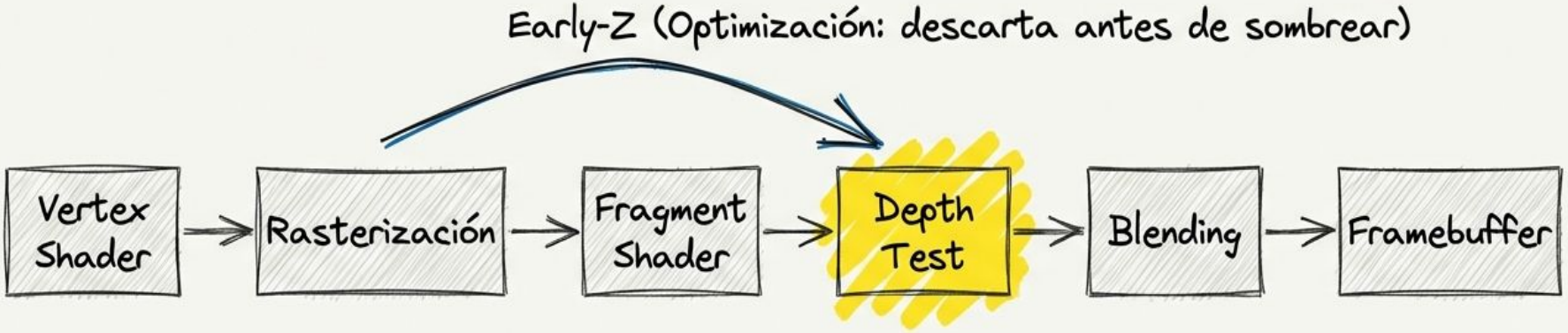
Solución: El Z-buffer resuelve esto píxel a píxel.

02

Implementación en el Pipeline

- [ ] - Ubicación en el Pipeline Gráfico
- [ ] - El proceso de Depth Test
- [ ] - Funciones de comparación

El Pipeline Gráfico (Depth Test)



Input: Fragmento con coordenada (x, y) y profundidad (z).

Proceso: Comparar z_fragmento vs z_buffer.

Resultado: Si falla el test, el fragmento se descarta (discard).

Lógica y Funciones de Comparación

```
if (newZ < storedZ) { // GL_LESS
   storedZ = newZ;
   storedZ = newZ;
   writeColor();
}
```

Función (glDepthFunc)	Descripción
GL_LESS	Estándar (Más cerca = Menor valor)
GL_GREATER	Usado en Reverse-Z
GL_LEQUAL	Multipass rendering (Sombras)
GL_ALWAYS	Siempre dibuja (ignora profundidad)

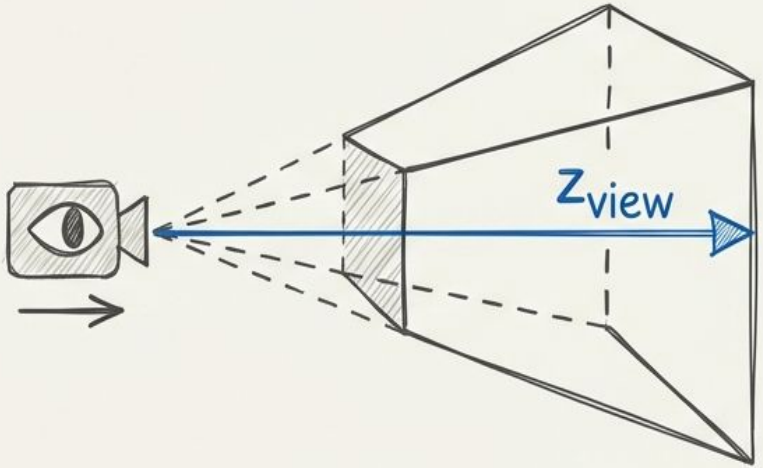
☆ **Depth Mask (glDepthMask):** Permite desactivar la ESCRITURA (solo lectura). Vital para partículas transparentes.

Matemáticas y Precisión

- ❑ Proyección en perspectiva (1/z)
- ❑ Distribución de bits
- ❑ La curva de precisión no lineal

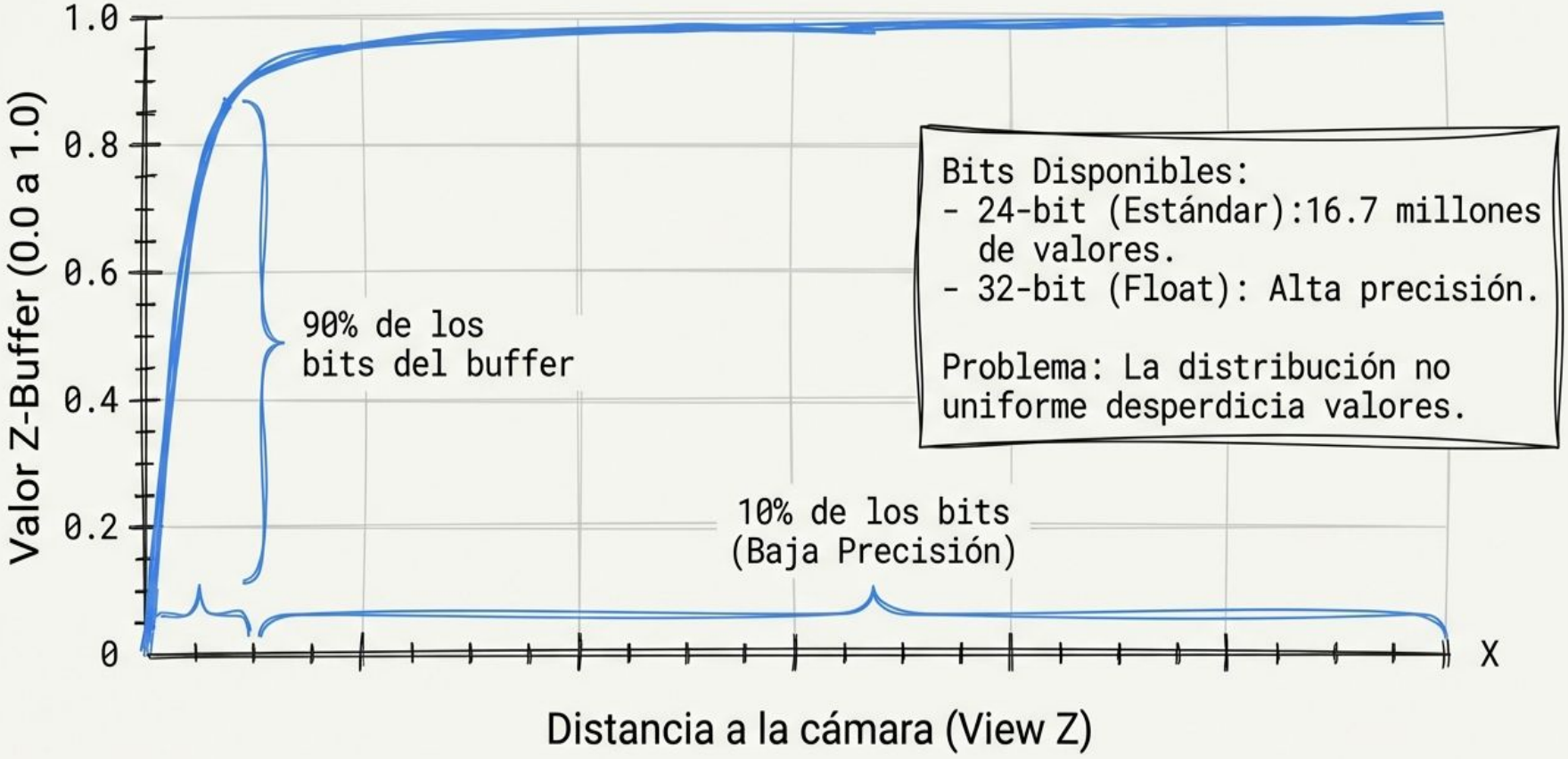
¿Por qué Z no es lineal?

$$z_{NDC} = A + \frac{B}{z_{view}}$$



- Para lograr perspectiva, dividimos por W (que es proporcional a Z).
- Consecuencia: La relación distancia vs buffer es una HIPÉRBOLA.
- El Costo: Gastamos muchísima precisión cerca de la cámara y muy poca lejos.

La Curva de Precisión



Bits Disponibles:
- 24-bit (Estándar):16.7 millones de valores.
- 32-bit (Float): Alta precisión.

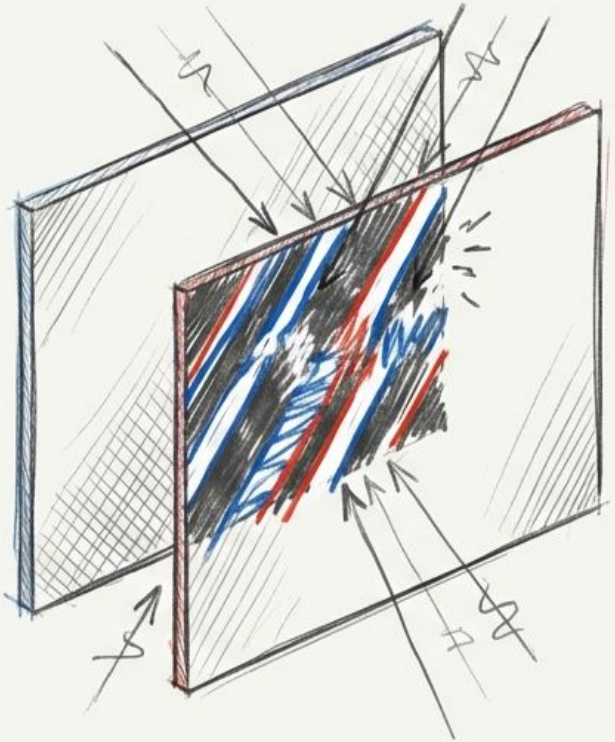
Problema: La distribución no uniforme desperdicia valores.

04

En el Mundo Real

- ❑ - Z-Fighting: Causas y Soluciones
- ❑ - El impacto crítico del Near Plane
- ❑ - Reverse-Z (Unreal / Unity)
- ❑ - Reverse-Z (Unreal / Unity)

Z-Fighting: El enemigo visible



¿Qué es?
Artefacto visual (parpadeo) cuando dos fragmentos tienen profundidad casi idéntica.

Causa:
La diferencia de profundidad es menor a la precisión del buffer (delta z < epsilon).

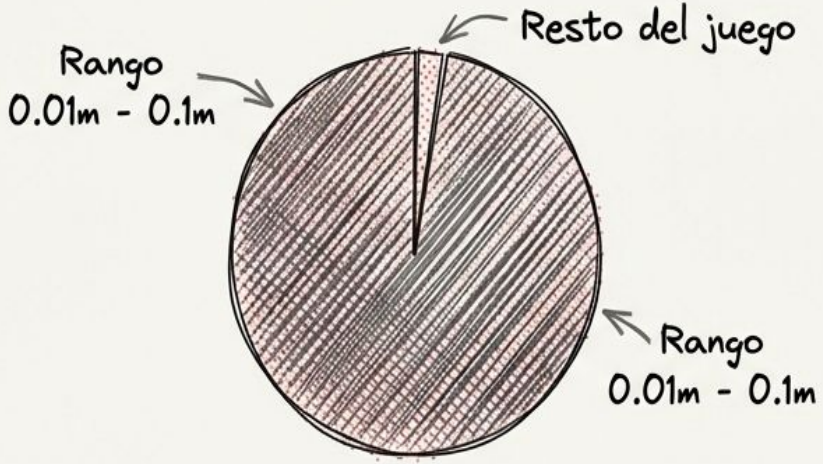
Escenarios Típicos:
- Geometría coplanar (ej. cuadros en pared, decals).
- Objetos muy lejanos.

Soluciones:
- Depth Bias / Polygon Offset.
- Alejar el Near Plane.

El Impacto Crítico del "Near Plane"

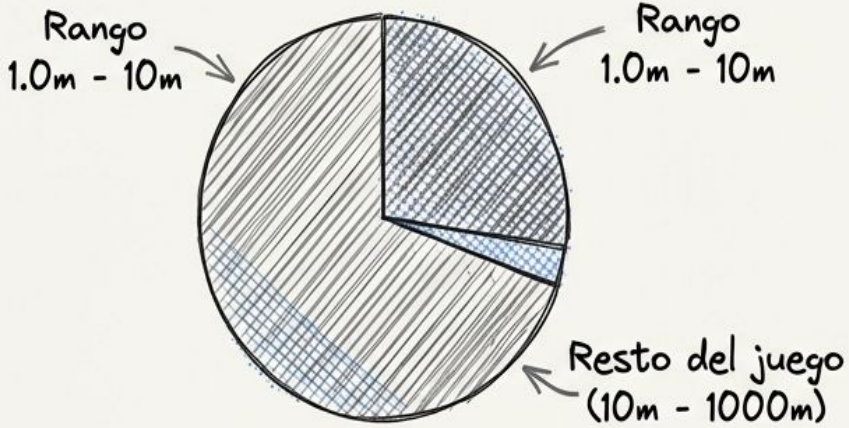
Escenario: Far Plane = 1000m

Caso A: Near = 0.01m



Z-fighting garantizado a media distancia.

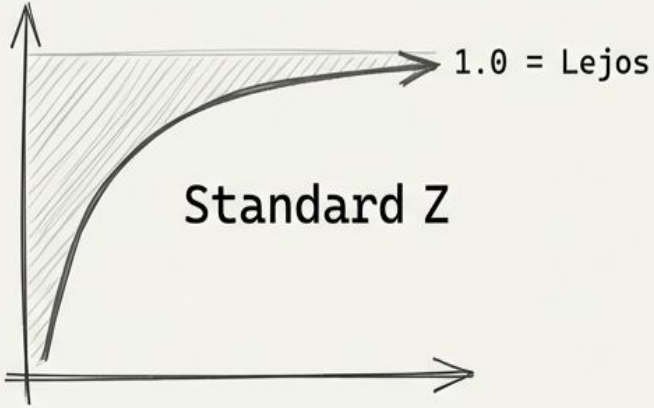
Caso B: Near = 1.0m



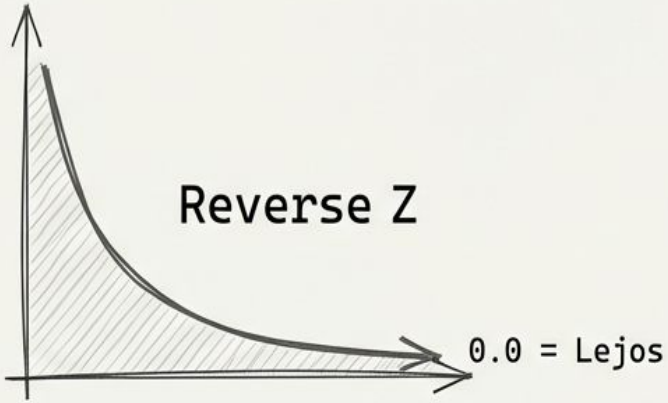
Recuperamos miles de veces más precisión.

Regla de Oro: Aleja el Near Plane tanto como sea posible.

La Solución Moderna: Reverse-Z



Float tiene baja precisión aquí.

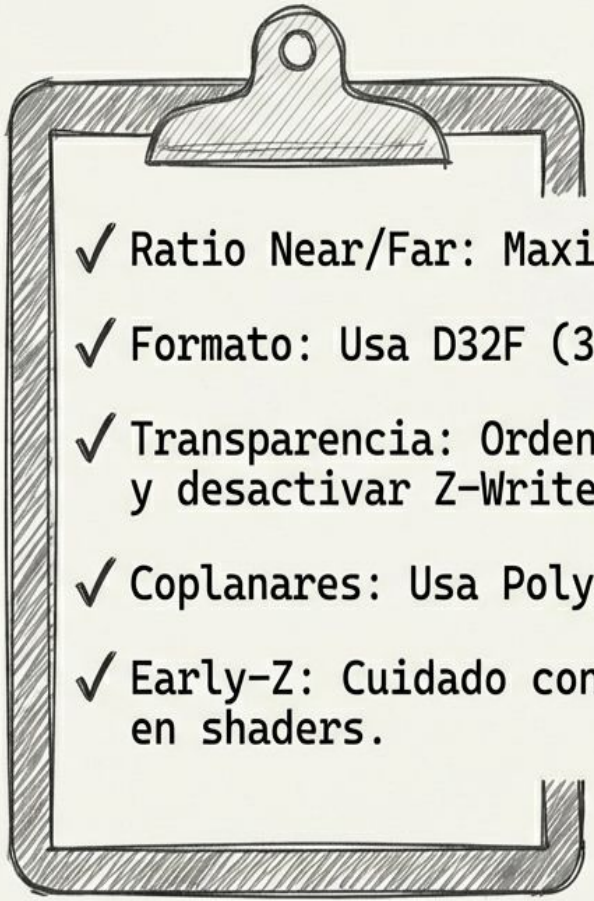


Float tiene ALTA precisión aquí.

- El Truco: Mapear Near → 1.0 y Far → 0.0.
- Sinergia: La precisión natural del Float (cerca de 0) compensa la pérdida de la perspectiva.
- Resultado: Distribución lineal y uniforme.

- Estándar en: Unreal Engine 5, Unity (HDRP), Godot 4.

Checklist de Buenas Prácticas



- ✓ Ratio Near/Far: Maximiza el Near, minimiza el Far.
- ✓ Formato: Usa D32F (32-bit Float) + Reverse-Z.
- ✓ Transparencia: Ordenamiento manual (Back-to-Front) y desactivar Z-Write.
- ✓ Coplanares: Usa Polygon Offset (Depth Bias).
- ✓ Early-Z: Cuidado con modificar profundidad en shaders.