UNIVERSIDAD CATÓLICA DE CÓRDOBA
JESUITAS

COMPUTACIÓN GRÁFICA

ANIMACIÓN

Profesora: Débora Theaux

Integrantes:
Nicolás Tobías Valentín
Rojas Nicolás

Resumen Ejecutivo

Este informe describe el desarrollo y la implementación de una aplicación gráfica interactiva en 3D utilizando la biblioteca **Three.js**. El proyecto se centra en la creación de una geometría (toroide) cuya forma y apariencia no están definidas por materiales estándar, sino por **shaders personalizados** escritos en **GLSL** (OpenGL Shading Language).

El proyecto demuestra la manipulación de vértices en tiempo real para la **deformación geométrica procedural** (Vertex Shader) y la generación de **texturas y colores dinámicos** (Fragment Shader). Adicionalmente, se implementa una cadena de **post-procesado** para aplicar efectos visuales avanzados, como UnrealBloomPass (resplandor), y se integra una interfaz de usuario (lil-gui) para la manipulación interactiva de los parámetros del shader.

Objetivos del Proyecto

Los objetivos principales establecidos para este proyecto fueron:

- **Aplicar Shaders Personalizados:** Demostrar la capacidad de anular el *pipeline* de renderizado estándar de Three.js mediante el uso de ShaderMaterial para controlar la GPU.
- **Deformación de Geometría:** Utilizar un **Vertex Shader** para modificar la posición de los vértices de un objeto basándose en algoritmos de ruido (Ruido Perlin) y variables de tiempo (uTime), logrando una animación orgánica.
- **Coloración Procedural:** Implementar un **Fragment Shader** para calcular el color de cada píxel de forma algorítmica, basándose en la posición y el ruido, en lugar de utilizar texturas de imagen estáticas.
- **Integración de Post-Procesado:** Configurar un EffectComposer para aplicar efectos visuales a la escena renderizada, específicamente el efecto *Bloom* para simular emisión de luz.
- **Interactividad:** Vincular una interfaz gráfica (GUI) a las variables uniform del shader, permitiendo al usuario controlar la animación y los efectos en tiempo real.

Arquitectura Tecnológica

El proyecto se sustenta en el siguiente *stack* tecnológico:

- **Motor 3D: Three.js** (Core).
- **Lenguaje de Shaders: GLSL (ES 3.0)**, implementado a través de los archivos vertex.js y fragment.js.
- **Post-Procesado:** Módulos JSM de Three.js, incluyendo EffectComposer, RenderPass y UnrealBloomPass.
- **Interfaz de Usuario: lil-gui** para la creación de controles deslizantes (sliders) interactivos.
- **Entorno de Desarrollo:** Un entorno basado en módulos (ESM) que requiere un empaquetador (*bundler*) como Webpack o Vite para la resolución de dependencias y la importación de *shaders* como cadenas de texto.

Desarrollo e Implementación

El núcleo del proyecto se divide en tres componentes principales:

4.1. Vertex Shader (Control de Forma)

El Vertex Shader es responsable de calcular la posición final de cada vértice de la geometría.

1. **Función de Ruido:** Se implementó una función pnoise (Ruido Perlin 3D) directamente en GLSL.
2. **Deformación:** Se calcula un patrón de ruido (pat) basado en las coordenadas UV del vértice (vUv), el tiempo (uTime) y un parámetro de ruido (uNoise).
3. **Animación de "Pulso":** Se calcula una máscara de proximidad (proximity) que se anima con el tiempo (uTime) y se controla con uSpread.
4. **Desplazamiento:** El resultado final (full) se utiliza para desplazar el vértice a lo largo de su vector normal (vNormal).
5. **Resultado:** La geometría del toroide se deforma y ondula en tiempo real, creando un efecto de "pulso" orgánico y animado.

4.2. Fragment Shader (Control de Color)

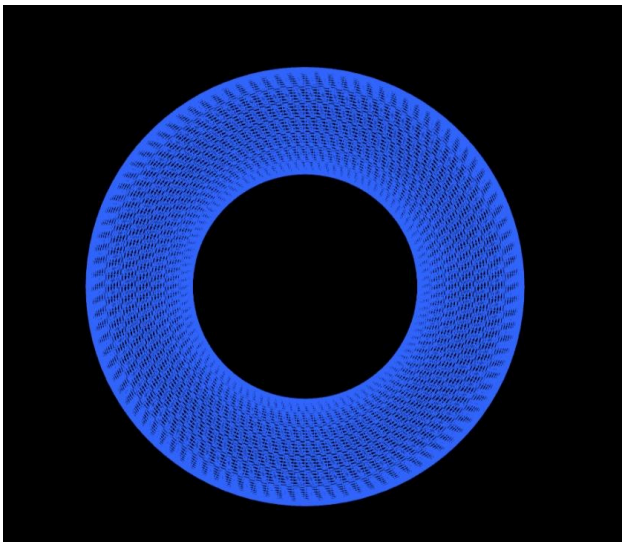
El Fragment Shader determina el color final de cada píxel en la superficie del objeto.

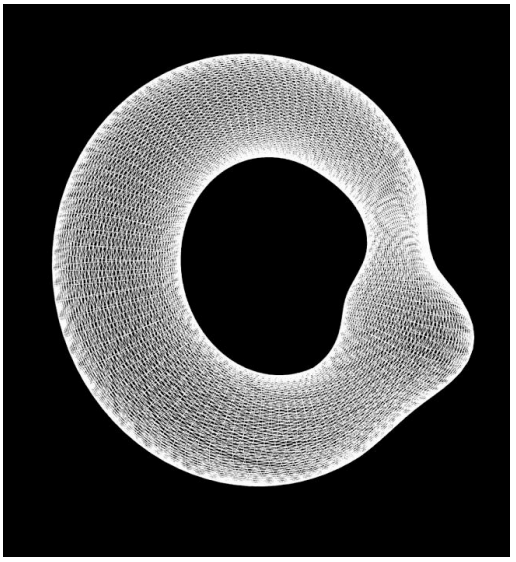
1. **Textura Procedural:** Se utiliza la misma función pnoise para generar un patrón de color, esta vez basándose en la posición del vértice (vPosition.z).
2. **Definición de Color:** Se define un color base (púrpura) y se multiplica por el valor del ruido.
3. **Intensidad para Bloom:** El color se calcula con una alta intensidad (mediante una división estratégica de vectores RGB). Esto es intencional para "alimentar" el pase de *Bloom* en la siguiente etapa, asegurando que estas áreas sean identificadas como emisoras de luz.

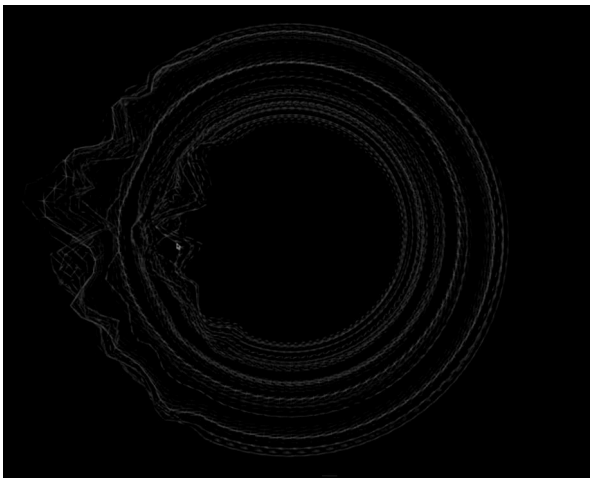
4.3. Post-Procesado y Renderizado (Efecto *Bloom*)

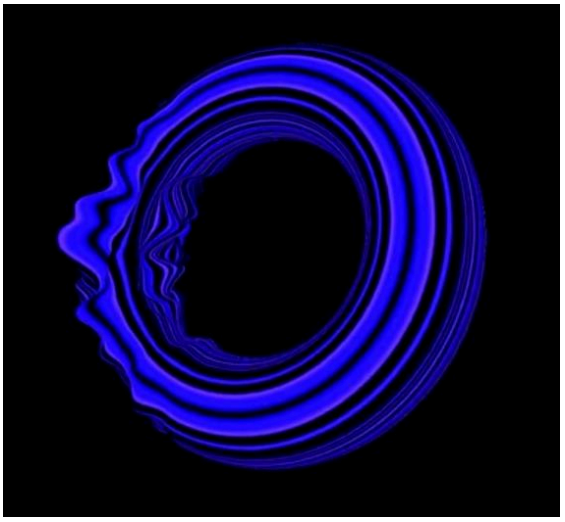
En lugar de renderizar la escena directamente, se utiliza un EffectComposer.

1. **RenderPass:** Se añade un pase inicial que renderiza la escena 3D (el toroide deformado).
2. **UnrealBloomPass:** Se añade un segundo pase. Este efecto toma la imagen renderizada, aísla los píxeles que superan un cierto umbral de brillo (nuestro color púrpura intenso), y difumina ese brillo para crear un efecto de resplandor o *glow*.
3. **composer.render():** En el bucle de animación, se llama a composer.render() en lugar del renderizador estándar, aplicando así la cadena de efectos.









Resultados y Conclusión

El proyecto culminó exitosamente en una demostración técnica que cumple con todos los objetivos propuestos. El resultado es un objeto 3D visualmente complejo y dinámico, cuya apariencia y comportamiento están completamente definidos por código algorítmico ejecutado en la GPU.

Aprendizajes Clave:

- El uso de **shaders personalizados** ofrece un control granular sobre el *pipeline* de renderizado, permitiendo efectos visuales que son imposibles de lograr con materiales estándar.
- La **separación de lógica** es fundamental: el Vertex Shader maneja la "forma" y el Fragment Shader maneja el "color", aunque ambos pueden usar las mismas técnicas (como el Ruido Perlin).
- El **post-procesado** no es un efecto secundario, sino una parte integral del diseño visual; el Fragment Shader fue diseñado específicamente para interactuar con el UnrealBloomPass.
- La **interactividad** mediante una GUI conectada a los uniforms del shader demuestra la naturaleza dinámica de los gráficos en tiempo real y es una herramienta de depuración invaluable.