Lluvia/Nieve con iluminación simple en tiempo real

Alumno: Meza Bravo Ivan Marcelino
Maestra: Rosaura Palma Orozco
E-mail: imezab1900@alumno.ipn.mx

Introducción y Alcance del Proyecto

El presente proyecto tiene como objetivo el desarrollo e implementación de una simulación gráfica interactiva en tiempo real que represente fenómenos meteorológicos complejos, específicamente lluvia y nieve, dentro de un entorno urbano tridimensional. La motivación principal reside en la necesidad de explorar técnicas de optimización gráfica en entornos web, superando las limitaciones tradicionales del renderizado de objetos individuales mediante el uso de geometrías instanciadas. El alcance del sistema abarca desde la generación procedural de la arquitectura urbana y el mobiliario público hasta la simulación de ciclos temporales día/noche y el comportamiento de agentes autónomos, todo ello validado mediante herramientas de auditoría técnica integradas.

Estado del arte

El Problema del Mallado en Computación Gráfica

El mallado (*mesh generation*) constituye el proceso fundamental mediante el cual las descripciones geométricas abstractas —tales como primitivas matemáticas, nubes de puntos, volúmenes CAD o superficies implícitas— se transforman en una estructura poligonal discreta apta para el renderizado en tiempo real y el cálculo numérico. En el contexto de la visualización interactiva moderna utilizando tecnologías web como WebGL y librerías de alto nivel como Three.js, la prioridad recae en la obtención de mallas "manifold". Esto implica generar superficies topológicamente cerradas y continuas que posean normales coherentes,

un rigor topológico crucial para garantizar una iluminación correcta y evitar artefactos visuales durante el sombreado. Asimismo, el control estricto sobre la densidad de la malla y la calidad de los elementos geométricos resulta indispensable para optimizar el rendimiento, buscando maximizar la tasa de cuadros por segundo (FPS) y minimizar el consumo de memoria gráfica.

Algoritmos de Triangulación y Reconstrucción

Para fundamentar técnicamente la generación de geometría, es necesario referir a los algoritmos clásicos de triangulación. La Triangulación de Delaunay y su estructura dual, el diagrama de Voronoi, son esenciales para garantizar mallas bien condicionadas donde se maximizan los ángulos mínimos de los triángulos, evitando geometrías alargadas que podrían causar errores de interpolación lumínica. En el ámbito de la reconstrucción de superficies a partir de datos dispersos, métodos como la Reconstrucción de Poisson y el algoritmo de *Ball-Pivoting* permiten inferir topología a partir de nubes de puntos, mientras que técnicas como *Marching Cubes* son el estándar para visualizar isosuperficies. Complementariamente, las técnicas de remallado y simplificación, basadas en métricas de error cuadrático (QEM), permiten gestionar la complejidad geométrica mediante la generación de Niveles de Detalle (LOD), asegurando que la carga poligonal se adapte dinámicamente a la distancia de visualización sin sacrificar la fidelidad de la silueta.

Implementación de la representación en mallas (objetos de interés)

Arquitectura de Generación Procedural

Para la materialización del entorno virtual propuesto, se descartó el uso de modelos estáticos importados en favor de una estrategia de modelado procedural utilizando las primitivas nativas de la librería gráfica. El plano del suelo se construyó mediante una geometría plana (PlaneGeometry) de alta resolución configurada explícitamente para recibir sombras, utilizando materiales estándar basados en física (PBR) para simular la rugosidad y respuesta lumínica del asfalto urbano. La representación de las edificaciones evolucionó desde volúmenes cúbicos simples hacia una generación algorítmica compleja gestionada por la clase CityBuilder. Esta implementación construye arquitecturas aleatorias en tiempo de ejecución, ensamblando primitivas geométricas básicas como prismas rectangulares (BoxGeometry) para los cuerpos principales y conos o pirámides (ConeGeometry) para las techumbres. Este enfoque permite crear variaciones arquitectónicas, como techos a dos aguas o fachadas escalonadas inspiradas en estilos europeos, sin la necesidad de modelar manualmente cada edificio.

Mobiliario Urbano y Entornos Naturales

La complejidad de la escena se incrementó mediante la clase SceneBuilder, encargada de poblar el entorno con elementos secundarios esenciales para el realismo. El mobiliario urbano, representado por las farolas, se generó combinando jerárquicamente cilindros para los postes y esferas para las luminarias, integrando en ellas lógica de emisión de luz que responde dinámicamente al ciclo día/noche. Paralelamente, se diseñó un parque urbano que incluye un lago circular y vegetación generada proceduralmente; los árboles se construyen instanciando troncos cilíndricos y copas dodecaédricas, una decisión de

diseño que mantiene una estética coherente "low-poly" mientras optimiza la carga de vértices en la escena.

Mallado Dinámico para Fenómenos Atmosféricos

El entorno se enriqueció con la implementación de un sistema de partículas meteorológicas avanzado (WeatherParticleSystem) que utiliza la técnica de InstancedMesh. En lugar de gestionar miles de objetos independientes, el sistema renderiza hasta diez mil partículas de lluvia o nieve utilizando una única geometría base (un plano simple) y variando sus matrices de transformación directamente en la GPU. Esto se complementa con texturas generadas proceduralmente en tiempo real mediante el API Canvas, lo que permite diferenciar visualmente la lluvia (trazos translúcidos) de la nieve (partículas difusas) y aplicar una mezcla aditiva para garantizar su visibilidad en condiciones nocturnas. Además, se integró un volumen emisor lógico (Box3), visualizado mediante un asistente gráfico (Box3Helper), que define espacialmente el área donde se originan las precipitaciones.

Documentación del proceso de mallado

Metodología de Instanciación (*Instanced Rendering*)

El proceso de mallado implementado en este proyecto se distingue por una metodología constructiva directa diseñada para maximizar la eficiencia en entornos web. A diferencia de los flujos de trabajo tradicionales que dependen de la importación de modelos externos, se optó por instanciar mallas paramétricas nativas. Esta decisión técnica garantiza matemáticamente que cada objeto generado posea una topología limpia y cerrada (*watertight*), con normales calculadas automáticamente y coordenadas de textura (UVs) optimizadas desde su creación. La técnica de *Instanced Rendering* permite que una sola llamada de dibujo (*draw call*) renderice miles de copias de la misma

geometría, reduciendo drásticamente la sobrecarga de la CPU y permitiendo que la simulación climática mantenga 60 cuadros por segundo estables.

Gestión de Memoria y Patrón *Object Pooling*

Para gestionar la vida útil de las partículas meteorológicas, se implementó el patrón de diseño *Object Pooling* o reciclaje de objetos. En lugar de instanciar y destruir objetos constantemente —lo cual provocaría fragmentación de memoria y pausas perceptibles causadas por el recolector de basura (*Garbage Collector*)—, el sistema inicializa un conjunto fijo de geometrías al inicio de la aplicación. Cuando una partícula de lluvia o nieve completa su trayectoria al colisionar con el plano del suelo, no se elimina de la memoria; en su lugar, el algoritmo simplemente reinicia sus coordenadas de posición hacia el volumen emisor superior y le asigna nuevos vectores de velocidad. Esta estrategia garantiza que la carga de memoria geométrica permanezca constante y predecible durante toda la ejecución.

Validación Técnica y Auditoría

Como mecanismo de control de calidad, se integró una herramienta de auditoría en tiempo real accesible mediante la tecla 'I'. Esta funcionalidad realiza una introspección profunda en el motor de renderizado, extrayendo métricas críticas directamente de la memoria de la GPU. Al activarse, reporta en la consola del navegador datos exactos sobre el número de geometrías y texturas residentes, la cantidad de llamadas de dibujo por cuadro y el recuento total de triángulos procesados. Esta instrumentación permite verificar empíricamente que, a pesar de la complejidad visual añadida por los sistemas de partículas y la ciudad procedural, la carga geométrica se mantiene dentro de los límites de eficiencia establecidos.

Sistemas Dinámicos y Comportamiento Autónomo

Ciclo Día/Noche e Iluminación Global

La percepción volumétrica de las mallas generadas depende intrínsecamente de su interacción con el sistema de iluminación. Se implementó un ciclo día/noche continuo que actúa como reloj maestro de la simulación, interpolando los valores ambientales para transicionar desde una iluminación diurna cenital hasta una configuración nocturna contrastada. La luz direccional principal orbita la escena modificando el ángulo de incidencia sobre las normales de las mallas, lo que resalta la tridimensionalidad de los edificios. Este sistema gestiona también la respuesta de los materiales emisivos: al detectar el umbral de oscuridad, modifica en tiempo real la propiedad emissiveIntensity de los materiales de las ventanas y farolas, simulando el encendido del alumbrado público.

Inteligencia Artificial de Agentes (Tráfico y Población)

Finalmente, la simulación cobra vida mediante la inclusión de agentes autónomos. Se implementó un sistema de tráfico vehicular donde los coches circulan respetando carriles predefinidos y direcciones de flujo, evitando colisiones mediante una lógica de posicionamiento restringido. Paralelamente, un sistema de población gestiona a los peatones, quienes deambulan aleatoriamente dentro de las zonas seguras designadas. Estos agentes poseen una lógica reactiva simple pero efectiva: monitorean el estado global del sistema meteorológico y, al detectar que la variable de precipitación es activa, modifican la jerarquía de su modelo 3D para hacer visible el objeto "paraguas", demostrando una integración directa entre la representación geométrica y la lógica de la simulación.

Tecnologías y Librerías Utilizadas

El desarrollo del proyecto se sustentó en un stack tecnológico moderno orientado a la web:

- **Three.js:** Motor gráfico principal utilizado para la gestión del grafo de escena, cámaras, luces y renderizado WebGL. Se utilizaron extensivamente sus clases de geometría primitiva y materiales estándar.
- **TypeScript:** Lenguaje de programación empleado para estructurar el código mediante tipado estático, interfaces y clases, fundamental para gestionar la complejidad de los múltiples sistemas interactivos de manera mantenible.
- **lil-gui:** Librería de interfaz de usuario utilizada para la manipulación paramétrica en tiempo real, permitiendo ajustar variables como la densidad de la lluvia, la velocidad del viento y la hora del día para pruebas de estrés y demostración.

Resultados y Evidencias Visuales

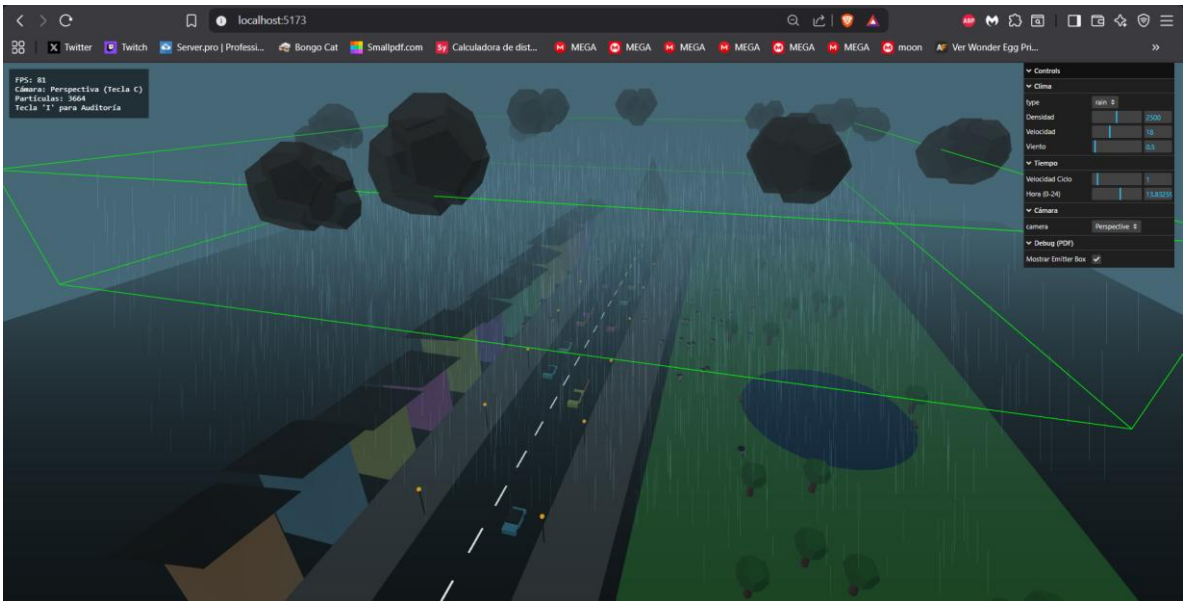

Figura 1. Validación del Volumen Emisor.

Vista en perspectiva que demuestra la delimitación espacial del sistema de partículas mediante Box3Helper (líneas verdes) y la correcta proyección de sombras dinámicas sobre el plano del suelo.

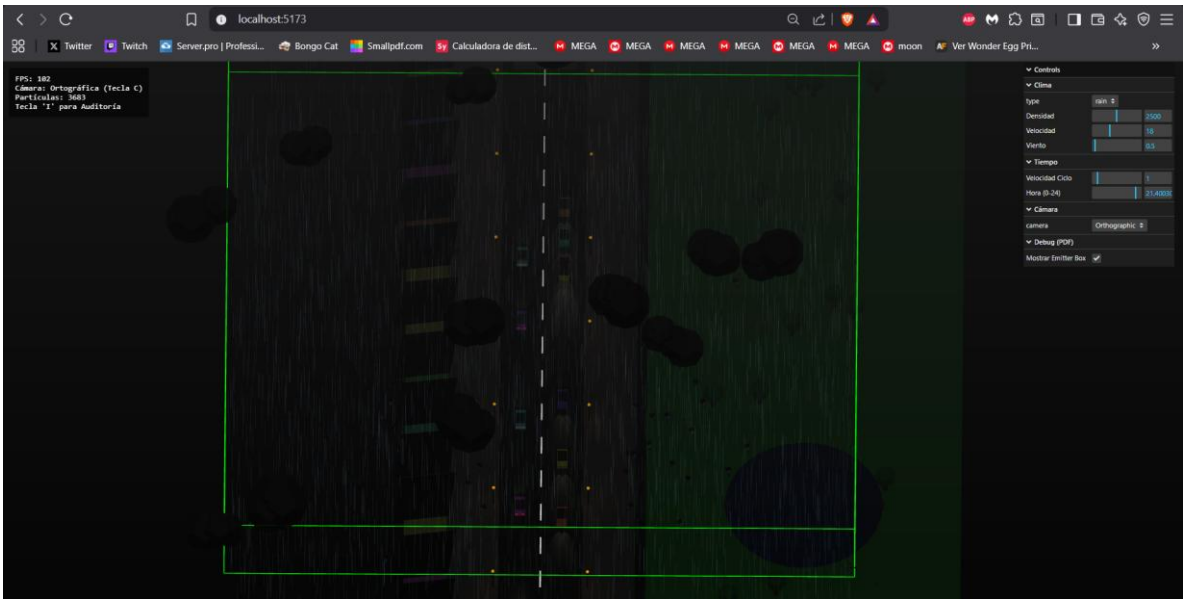

Figura 2. Inspección en Proyección Ortográfica.

Visualización técnica libre de distorsión de perspectiva, utilizada para validar la proporción y distribución equidistante de los elementos arquitectónicos generados proceduralmente.

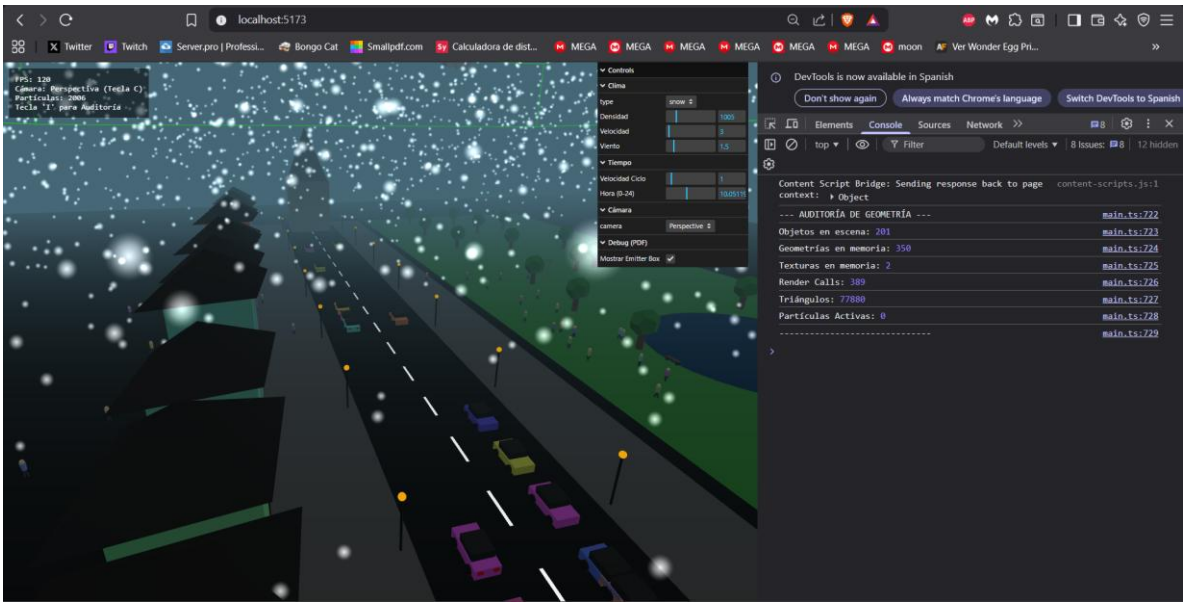

Figura 3. Auditoría de Rendimiento en Tiempo Real.

Reporte de introspección del motor gráfico (obtenido mediante el comando 'I') que detalla el uso de memoria de geometrías y el conteo de *draw calls*, validando la eficiencia del algoritmo de *InstancedMesh*.

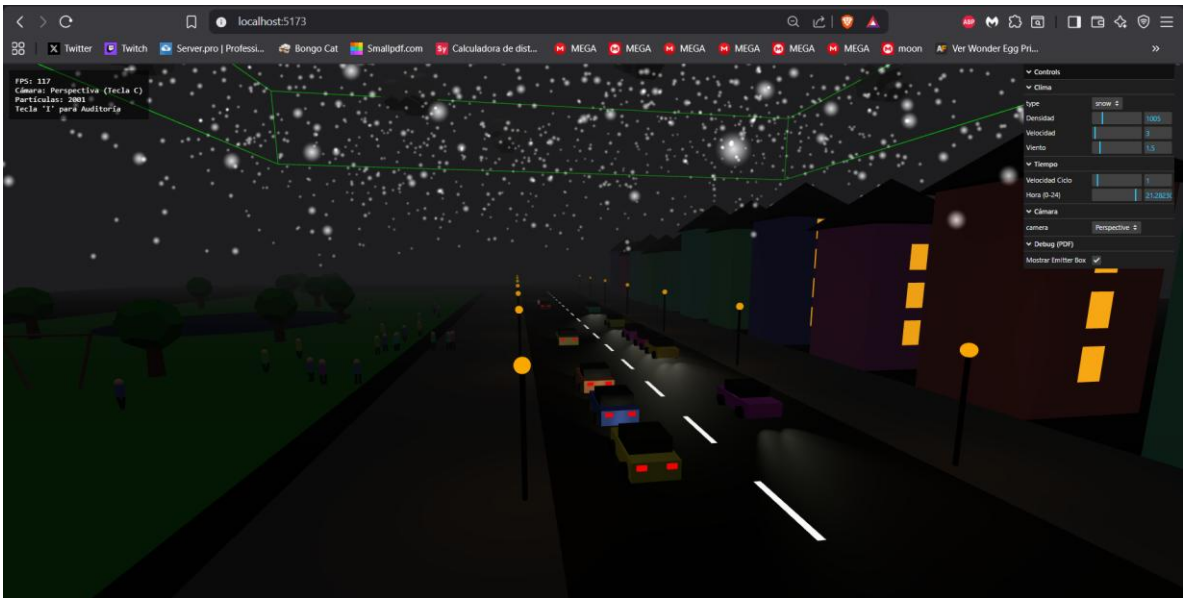

Figura 4. Comportamiento del Ciclo Circadiano.

Demostración de la simulación nocturna donde se aprecia la activación automática de las luces puntuales (SpotLights) en el sistema de tráfico y la respuesta de los materiales emisivos en la infraestructura urbana.

Guía de Despliegue y Requisitos de Ejecución

Requisitos del Entorno de Desarrollo (Software y Hardware)

Para garantizar la correcta compilación y ejecución de la simulación, el entorno huésped debe cumplir con una serie de especificaciones técnicas. A nivel de hardware, dado que la carga de renderizado recae principalmente en la Unidad de Procesamiento Gráfico (GPU), es indispensable contar con un equipo que soporte aceleración por hardware y sea compatible con el estándar **WebGL 2.0**. Aunque el sistema de partículas optimizado permite su ejecución en gráficos integrados modernos, se recomienda una tarjeta gráfica dedicada para mantener la estabilidad en 60 FPS con todas las características visuales activas (sombras dinámicas de alta resolución y suavizado de bordes).

En cuanto al software base, el proyecto se construyó sobre el ecosistema de JavaScript moderno. Es imperativo tener instalado el entorno de ejecución **Node.js** (preferiblemente la versión LTS v18.0.0 o superior). Node.js no se utiliza para ejecutar el proyecto en el servidor, sino para gestionar las herramientas de construcción y transpilación. Junto con Node, se requiere el gestor de paquetes **npm** (Node Package Manager) para resolver el árbol de dependencias del proyecto. Finalmente, la visualización debe realizarse en un navegador web moderno (Google Chrome, Mozilla Firefox, Microsoft Edge) actualizado a su última versión estable para asegurar la compatibilidad con las últimas especificaciones de ECMAScript y las APIs gráficas del navegador.

Gestión de Dependencias y Arquitectura del Proyecto

El proyecto no es un script monolítico, sino una aplicación modular estructurada. La gestión de librerías externas se centraliza en el archivo de configuración package.json. Las dependencias críticas incluyen la librería **Three.js** (el motor gráfico), **@types/three** (las definiciones de tipos para el desarrollo seguro en TypeScript) y **lil-gui** (para la interfaz de control).

El proceso de instalación comienza con la clonación o descarga del código fuente desde el repositorio de control de versiones. Una vez localizados los archivos en el sistema local, se debe ejecutar el comando de instalación de dependencias (npm install) en la terminal del sistema. Este proceso descarga todos los módulos necesarios desde el registro npm y los aloja en el directorio local node_modules, asegurando que el entorno de desarrollo cuente con todas las herramientas de renderizado y matemáticas requeridas sin necesidad de configuraciones globales en el sistema operativo.

Proceso de Transpilación y Servidor de Desarrollo

Dado que el código fuente está escrito en **TypeScript** (.ts) para garantizar la robustez del tipado y la orientación a objetos, este no puede ser interpretado directamente por los navegadores web. Por ello, se utiliza **Vite** como herramienta de construcción (*build tool*). Vite actúa como un servidor de desarrollo local extremadamente rápido que realiza la transpilación de TypeScript a JavaScript nativo (ES Modules) en tiempo real (*Hot Module Replacement*).

Para iniciar la simulación en modo de desarrollo, se ejecuta el comando npm run dev en la consola. Este comando levanta un servidor local (típicamente en el puerto 5173, accesible mediante http://localhost:5173) que sirve la aplicación y vigila los cambios en el código fuente, recargando el módulo gráfico instantáneamente ante cualquier modificación. Para un despliegue en producción o entrega final estática, se utiliza el comando npm run build, el cual invoca al compilador para generar una versión optimizada, minificada y empaquetada de la aplicación en la carpeta dist, lista para ser alojada en cualquier servidor web estático sin dependencias de Node.js activas.

Conclusiones

La implementación de técnicas de generación procedural y renderizado instanciado ha permitido crear una simulación atmosférica y urbana compleja que cumple con los rigurosos estándares de rendimiento de la web moderna. La validación mediante herramientas de auditoría confirma que es posible mantener una alta fidelidad visual y una topología de malla limpia sin recurrir a activos externos pesados. La integración de sistemas dinámicos, como el ciclo día/noche y la inteligencia artificial reactiva, demuestra la capacidad del sistema para gestionar múltiples capas de lógica simultáneas, resultando en una herramienta robusta tanto para la visualización técnica como para la simulación interactiva.