Diseño y Desarrollo de un Prototipo Web 3D Interactivo para la Visualización Técnica y Análisis Estructural de Hardware de Alto Rendimiento mediante Pipelines de Optimización Gráfica y WebAssembly

Estudiante

Alexander Woodcock Salomón

Asesor

Deivid Enrique Triviño Lozada

Universidad Nacional Abierta y a Distancia - UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería

Ingeniería Multimedia

2025

Resumen

El diseño de hardware de alto rendimiento (drones, sistemas robóticos) enfrenta un desafío crítico en la comunicación técnica: los medios digitales tradicionales (imágenes 2D, PDFs estáticos) imponen alta carga cognitiva para comprender estructuras tridimensionales complejas. Este proyecto aborda esta brecha mediante el diseño, desarrollo y evaluación de un prototipo web 3D interactivo basado en Unity WebGL que optimiza la visualización técnica de hardware mediante pipelines de optimización gráfica avanzados y WebAssembly. El marco teórico integra teoría de carga cognitiva (Mayer, Paivio), interacción humano-computador en 3D (Bowman, Norman), y renderizado físicamente basado (Cook-Torrance, Burley). La metodología sigue Design Science Research con desarrollo iterativo en Sprints de 4 semanas, implementando técnicas de retopología, baking de normales, y Asset Bundles para cumplir con KPIs cuantitativos (¡50,000 polígonos, ¿30 FPS en móvil mid-range, ¡5s tiempo de carga). La validación incluye profiling de rendimiento (Unity Profiler) y pruebas de usabilidad (SUS, N=8-12 ingenieros/técnicos). El producto final es un MVP científico que valida la viabilidad técnica de WebGL para visualización de ingeniería, estableciendo un pipeline replicable documentado según estándares académicos.

Palabras clave: WebGL, Unity, WebAssembly, Renderizado Físicamente Basado (PBR), Visualización 3D Interactiva, Optimización de Assets 3D, Carga Cognitiva, Interacción Humano-Computador, Technical Art, Hardware de Alto Rendimiento.

Abstract

High-performance hardware design (drones, robotic systems) faces a critical challenge in technical communication: traditional digital media (2D images, static PDFs) impose high cognitive load for understanding complex three-dimensional structures. This project addresses this gap through the design, development, and evaluation of an interactive 3D web prototype based on Unity WebGL that optimizes technical hardware visualization through advanced graphics optimization pipelines and WebAssembly. The theoretical framework integrates cognitive load theory (Mayer, Paivio), 3D human-computer interaction (Bowman, Norman), and physically-based rendering (Cook-Torrance, Burley). The methodology follows Design Science Research with iterative development in 4-week Sprints, implementing retopology, normal baking, and Asset Bundles to meet quantitative KPIs (¡50,000 polygons, ¿30 FPS on mid-range mobile, ¡5s load time). Validation includes performance profiling (Unity Profiler) and usability testing (SUS, N=8-12 engineers/technicians). The final product is a scientific MVP that validates the technical feasibility of WebGL for engineering visualization, establishing a replicable, academically documented pipeline.

Keywords: WebGL, Unity, WebAssembly, Physically-Based Rendering (PBR), Interactive 3D Visualization, 3D Asset Optimization, Cognitive Load, Human-Computer Interaction, Technical Art, High-Performance Hardware.

Tabla de Contenido

Lista de Tablas

Información General de la Propuesta de Trabajo de Grado Proyecto Aplicado

Integrantes de la Propuesta de Investigación

Nombre del estudiante: Alexander Woodcock Salomón

Identificación C.C.: 1018474351

Programa Académico: Ingeniería Multimedia

No. de Créditos Aprobados: 152

% de créditos aprobados: 92.11

Correo electrónico: awoodcocks@unadvirtual.edu.co

Teléfono / Celular: 3054396581

Dirección residencia: Finca Armenia casa 2

Municipio / Departamento: Pasto / Nariño

CENTRO: ZCSUR

ZONA: PASTO

Datos Específicos del Proyecto

Duración del proyecto (meses): Seis (6)

Línea de Investigación: Ingeniería Multimedia, Visualización Técnica 3D Web

Escuela: Escuela de Ciencias Básicas, Tecnología e Ingeniería (ECBTI)

Descriptores palabras clave: WebGL, Unity, WebAssembly, PBR, Visualización 3D, Optimización, Carga Cognitiva, HCI, Technical Art.

Planteamiento del Problema

Los modelos CAD de ingeniería para hardware de alto rendimiento (drones, sistemas robóticos, componentes electrónicos) suelen superar los 2GB de datos y millones de polígonos, cifras incompatibles con el contexto de ejecución de un navegador web móvil, limitado frecuentemente a $\sim$2GB de memoria compartida y sin acceso directo a la API gráfica nativa. La infraestructura web actual depende mayoritariamente de medios pasivos (imágenes 2D estáticas, PDFs), los cuales generan una alta carga cognitiva intrínseca al obligar al usuario a reconstruir mentalmente la espacialidad tridimensional del objeto.

Desde la Teoría de la Carga Cognitiva (Sweller, 1988), la visualización estática impone una carga innecesaria en la memoria de trabajo del usuario, quien debe realizar rotaciones mentales complejas. Hegarty (2004) demostró que las tareas de rotación mental consumen recursos cognitivos visuoespaciales, dificultando la comprensión de estructuras complejas.

El problema técnico radica en la "Brecha de Fidelidad vs. Rendimiento":

- Tamaño de archivo. Modelos CAD originales (STEP, IGES) con millones de superficies NURBS son incompatibles con limitaciones de bandwidth web (target: $<$ 50MB build inicial).
- Rendimiento en tiempo real. Mantener $>$ 30 FPS (33.33ms frame time) en dispositivos móviles mid-range (e.g., Snapdragon 7 series) requiere optimización extrema.
- Latencia de interacción. Soluciones de *Pixel Streaming* introducen latencia de red inaceptable ($>$ 100ms) para interacción precisa en hotspots técnicos.

Existe, por tanto, la necesidad de un pipeline de Technical Art que permita desplegar estos modelos con alta fidelidad visual y bajo costo computacional, validando la viabilidad de

WebGL/WebAssembly para aplicaciones de ingeniería.

Justificación

La elección de Unity WebGL sobre librerías nativas de JavaScript (Three.js, Babylon.js) o soluciones de streaming remoto (Unreal Pixel Streaming) no es arbitraria, sino que responde a requisitos cuantitativos de *Computación de Alto Rendimiento (HPC)* en la web y necesidades del pipeline de arte técnico:

Criterios Técnicos de Selección

Rendimiento de CPU (WASM vs JavaScript)

Unity utiliza el backend IL2CPP para transpilar código C# a C++ y posteriormente a WebAssembly (WASM). Esto permite que la lógica de interacción y los cálculos físicos se ejecuten a velocidad casi nativa, evitando la sobrecarga del *Garbage Collector* de JavaScript que causa micro-congelamientos (frame drops) en aplicaciones complejas. Según documentación oficial de Unity (2024), IL2CPP elimina pauses de GC que en JavaScript pueden exceder 16ms, causando caída por debajo de 60 FPS.

Pipeline de Assets Robusto

A diferencia de Three.js que requiere construcción manual de cargadores y gestores de memoria, Unity ofrece un sistema integrado de *Asset Bundles* y compresión de texturas (ASTC/ETC2) esencial para garantizar que el prototipo se mantenga dentro del presupuesto de memoria de 50MB iniciales, permitiendo carga progresiva (Progressive Loading) de assets de alta resolución en segundo plano.

Optimizaciones Gráficas Avanzadas

El proyecto requiere técnicas de *Occlusion Culling* (no renderizar geometría bloqueada) y *LOD* (Level of Detail) automático. Implementar esto desde cero en una librería ligera es propenso

a errores y consume tiempo de desarrollo que debería dedicarse a la lógica de la aplicación. Unity URP ofrece estas técnicas integradas con profiling visual.

Herramientas Visuales para Artistas Técnicos

El flujo de trabajo requiere colaboración entre programadores y artistas técnicos. Unity Shader Graph permite a no-programadores crear materiales PBR complejos, mientras que Timeline facilita la animación del despiece (exploded view) sin código.

Escalabilidad Industrial

El uso de un motor comercial estandariza el flujo de trabajo, permitiendo que el prototipo (MVP) sea mantenible y escalable, alineándose con las prácticas de la Industria 4.0. El proyecto puede evolucionar a aplicación de producción sin cambiar stack tecnológico.

Aporte a la Ingeniería Multimedia

Este proyecto aporta al campo de la Ingeniería Multimedia al:

- documentar un pipeline técnico replicable para la optimización de activos CAD a WebGL
- validar el rol del ingeniero multimedia como puente entre la complejidad técnica y la experiencia de usuario final
- generar conocimiento sobre métricas cuantitativas (KPIs) para rendimiento en Web 3D
- aplicar teoría cognitiva (Mayer, Paivio) a casos de uso reales de ingeniería

Objetivos

Objetivo General

Desarrollar un prototipo de visualización Web 3D interactiva basado en Unity WebGL, enfocado en la exploración técnica y despiece funcional de hardware de alto rendimiento, optimizado mediante técnicas de Technical Art para su despliegue eficiente en navegadores web estándar y dispositivos móviles.

Objetivos Específicos

Diseñar un pipeline de optimización de activos 3D que reduzca la carga poligonal de modelos CAD originales en un 90 % (KPI: $P_{total} < 50,000$ triángulos) manteniendo la fidelidad visual mediante técnicas de *baking* de mapas normales y retopología.

Implementar shaders PBR personalizados en Unity URP para la simulación realista de materiales técnicos (fibra de carbono, metales anodizados, plásticos ABS) asegurando un *Frame Time* inferior a 33.33ms ($>$ 30 FPS) en dispositivos móviles de gama media (Snapdragon 7 series).

Programar un sistema de interacción en C# compilado a WebAssembly que permita manipulación orbital de cámara y ejecución paramétrica de "Vista Explosiva" (exploded view), facilitando la comprensión espacial de ensamblajes complejos.

Validar el rendimiento y la usabilidad del prototipo mediante herramientas de perfilado (Unity Profiler, Chrome DevTools) y pruebas de usabilidad (System Usability Scale - SUS) con ingenieros/técnicos (N=8-12), asegurando Time to Interactive (TTI) $<$ 5 segundos en redes 4G.

Marco de Referencia

Estado del Arte

El estado del arte analiza las tecnologías actuales de visualización 3D en la web, evaluando sus capacidades, limitaciones y pertinencia para la visualización técnica de hardware de alto rendimiento. Se examinan bibliotecas nativas, motores de juego y soluciones de streaming para identificar la herramienta óptima que equilibre fidelidad visual y rendimiento en navegadores.

Benchmarking de Soluciones Web 3D Actuales

La visualización 3D en la web ha evolucionado significativamente desde la introducción de WebGL en 2011 (Khronos Group). Yu et al. (2023) identifican WebGL como la tecnología dominante para renderizado 3D en navegadores, con creciente competencia de WebGPU. A continuación se presentan las principales tecnologías evaluadas:

Three.js. Three.js es una biblioteca de renderizado 3D ligera construida sobre WebGL. Según el repositorio oficial de Cabello (2024), cuenta con más de 110,000 estrellas en GitHub, indicando amplia adopción en la comunidad de desarrolladores web.

Ventajas. Curva de aprendizaje accesible, tamaño de bundle reducido ($\sim$600KB minified+gzip), documentación exhaustiva, ecosistema extenso de plugins.

Desventajas. Requiere implementación manual de pipeline de assets (cargadores GLTF/OBJ personalizados), gestión de memoria explícita, ausencia de editor visual para artistas no-programadores.

Babylon.js. Babylon.js es un motor WebGL/WebGPU completo diseñado con enfoque enterprise, ofreciendo abstracciones de alto nivel para física, GUI y animación.

Ventajas. Soporte nativo de TypeScript, editor visual integrado (Babylon.js Playground), sistema de física robusto (Cannon.js/Ammo.js), migración temprana a WebGPU implementa-

da (soporte desde v5.0). Denes et al. (2024) en la IEEE Conference on Games demostraron que WebGPU como backend en motores de juego muestra tiempos de cuadro consistentemente mejores que WebGL, validando la estrategia de migración temprana de Baby lon.js.

Desventajas. Tamaño de bundle mayor ($\sim$2MB minified), percepción de menor rendimiento en escenas ligeras comparado con Three.js, aunque recientes optimizaciones en v6.0 (2023) han reducido esta brecha.

Unity WebGL. Unity Technologies ofrece compilación a WebGL desde Unity 5.0 (2015), con mejoras significativas en Unity 2020+ mediante IL2CPP y WebAssembly.

Ventajas. Ecosistema completo (Asset Store con $>$500K assets, Profiler integrado, Shader Graph visual), compilación IL2CPP a WebAssembly para lógica compleja (elimina pauses de GC de JavaScript), Asset Bundles para gestión granular de memoria, herramientas visuales (Timeline, Cinemachine) para artistas no-programadores.

Desventajas. Tamaño de build inicial grande (5-15MB según configuración de compresión Brotli/Gzip), tiempo de carga mayor en primera ejecución (5-7s en 4G), licencia comercial para proyectos de ingresos $>$ \$100K/año.

Unreal Engine Pixel Streaming. Epic Games introdujo Pixel Streaming en Unreal Engine 4.21 (2018) para transmitir contenido de alta fidelidad desde servidores GPU a clientes web ligeros.

Ventajas. Calidad gráfica máxima (Ray Tracing en tiempo real, Lumen Global Illumination), hardware client mínimo (solo decodificación de video H.264), permite escenas extremadamente complejas ($>$10M polígonos) inviables en WebGL nativo.

Desventajas. Latencia de red crítica ($<$60ms recomendado para interactividad fluida, $>$100ms degrada UX significativamente), costo de infraestructura cloud escalable (\$0.50-

$2.00/hora por sesión GPU según AWS/Azure pricing), requiere servidor dedicado por usuario concurrente, escalamiento horizontal costoso para audiencias masivas.

Spline. Spline es un editor 3D web-first lanzado en 2021, enfocado en diseño iterativo y colaboración en tiempo real.

Ventajas. Colaboración multiplayer en tiempo real (similar a Figma para 3D), UI amigable para diseñadores sin experiencia en programación, export directo a frameworks web (React, Vue, Svelte, Vanilla HTML).

Desventajas. Limitaciones en escenas complejas (<100K polígonos para rendimiento óptimo), orientado a motion graphics y diseño de producto más que ingeniería técnica, falta de soporte para lógica custom en C#/JavaScript.

Marmoset Viewer. Marmoset Viewer (parte del ecosistema Marmoset Toolbag) es un visor WebGL optimizado para presentación de assets 3D con PBR de alta calidad.

Ventajas. Renderizado PBR de calidad excepcional (basado en Marmoset Toolbag 4 renderer), ideal para portfolios de arte 3D (integración directa con ArtStation), soporte de turntables automáticos y anotaciones de texto.

Desventajas. Orientado exclusivamente a visualización estática (turntables 360°), interactividad limitada a rotación orbital pasiva, no permite scripting de lógica custom, no diseñado para aplicaciones de ingeniería con despiece funcional.

Sketchfab. Sketchfab (adquirido por Epic Games en 2021) es una plataforma de hosting con visor WebGL embebible, conteniendo más de 5 millones de modelos 3D.

Ventajas. Gran biblioteca de modelos comunitarios, viewer embebible via iframe, soporte VR/AR listo (WebXR API), funcionalidad de anotaciones de información técnica.

Desventajas. Control limitado sobre lógica de interacción (API JavaScript básica), dependencia de plataforma externa (vendor lock-in), rest ricciones de branding en plan gratuito (logo Sketchfab obligatorio), límites de tamaño de archivo en planes free/basic.

PlayCanvas. PlayCanvas es una plataforma de desarrollo de juegos web con motor WebGL integrado y editor colaborativo en la nube desde 2011.

Ventajas. Editor en navegador con colaboración en tiempo real (similar a Google Docs pero para desarrollo de juegos), asset pipeline completo con optimización automática (compresión de texturas, bundling), deployment con un click a CDN global, tamaño de runtime pequeño (500KB gzip).

Desventajas. Menor ecosistema de assets comparado con Unity/Unreal, curva de aprendizaje para desarrolladores acostumbrados a motores desktop-first, dependencia de plataforma cloud (aunque existe versión self-hosted open-source), documentación menos exhaustiva que Three.js para casos de uso técnicos avanzados.

Tabla 1

Comparativa de Soluciones Web 3D

Tecnología	Build	Load	PBR	Lenguaje	Interact.	Costo
Three.js	600KB	$<$2s	Manual	JS	Total	Gratis
Babylon.js	2MB	$\sim$3s	Built-in	TS/JS	Total	Gratis
Unity	5-15MB	5-7s	URP	C# (WASM)	Total	Gratis*
UE Pixel	5MB	$\sim$2s	Photoreal	C++/BP	Latencia	Cloud
Spline	1MB	$\sim$2s	Basic	Visual	Limitada	Gratis/Pro
Marmoset	800KB	$\sim$3s	Advanced	Visual	Rotación	Licencia
Sketchfab	3MB	$\sim$4s	Good	N/A	Media	Gratis/Pro
PlayCanvas	500KB	$<$2s	Built-in	JS	Total	Gratis/Pro

Nota. *Unity gratuito para ingresos $<$ \$100K/año. Load Time medido en red 4G (10 Mbps downstream). UE Pixel Streaming requiere infraestructura cloud (AWS EC2 G4 instances). Build Size incluye compresión Brotli. *Fuente*. Autoría propia basada en documentación oficial y benchmarks de comunidad (Three.js Forum, Unity Forums, Babylon.js Documentation).

Gap Analysis y Justificación de Unity WebGL.

Descartados. La evaluación técnica identificó limitaciones críticas en tres categorías de soluciones:

Pixel Streaming. Latencia de red inaceptable ($>$100ms típica en redes 4G latinoamericanas) para interacción precisa en hotspots técnicos y animación de despiece paramétrico, que requiere respuesta $<$16ms para fluidez perceptual.

Spline/Marmoset/Sketchfab. Limitaciones de lógica custom (no permiten scripting de despiece paramétrico en C# con control fino de interpolación).

Three.js/Babylon.js/PlayCanvas. Tiempo de desarrollo excesivo para implementar pipeline de optimización completo (Occlusion Culling automático, sistema LOD de 3 niveles, Asset Bundles con carga asíncrona) desde cero, estimado en $>$200 horas adicionales de ingeniería.

Seleccionado Unity WebGL. Unity WebGL fue seleccionado por el equilibrio óptimo entre cuatro factores críticos:

Balance Fidelidad-Tamaño. Progressive Loading mitiga carga inicial (shell de 3MB + Asset Bundles bajo demanda).

Herramientas Visuales. Shader Graph, Timeline, Cinemachine esenciales para colaboración con artistas técnicos no-programadores.

Rendimiento Predecible. C# $\rightarrow$ IL2CPP $\rightarrow$ WASM garantiza ejecución determinista sin pauses de GC de JavaScript (crítico para mantener frame budget de 33.33ms).

Ecosistema Maduro. 15+ años de desarrollo, Asset Store con soluciones probadas para problemas comunes (oclusión, batching, compresión de texturas).

Marco Teórico

El diseño, desarrollo y evaluación de la plataforma web 3D se fundamenta en un marco teórico multidisciplinario que integra conocimientos de la ingeniería, el diseño, la psicología cognitiva y la computación gráfica.

Hipótesis de Usabilidad (Carga Cognitiva).. "La visualización 3D interactiva reduce la carga cognitiva intrínseca comparada con imágenes 2D estáticas para la comprensión de estructura interna".

Métrica de validación: Comparación de tiempo de comprensión y tasa de error en pruebas con usuarios (A/B test: 3D vs 2D).

El uso de metodología MVP justifica la exclusión intencional de características como multi-idioma, persistencia de estado, o analytics avanzados, priorizando la validación de viabilidad técnica.

Teoría de la Carga Cognitiva (Sweller, 1988; Sweller et al., 2019)

La Teoría de la Carga Cognitiva, desarrollada por Sweller (1988) y refinada durante más de 30 años, postula que la memoria de trabajo tiene capacidad limitada ($\sim$4 elementos simultáneos según Cowan, 2001) y que el diseño instruccional debe minimizar la carga impuesta sobre esta capacidad.

Sweller, van Merriënboer y Paas (2019) en su revisión de 20 años publicada en *Educational Psychology Review* identifican **tres tipos de carga cognitiva**:

Carga Intrínseca (Intrinsic Load). Complejidad inherente del material de aprendizaje, determinada por la interactividad de elementos. En este proyecto, la complejidad intrínseca es alta: un dron contiene docenas de componentes con relaciones espaciales jerárquicas (rotor $\rightarrow$

motor → ESC → batería). Esta carga es *irreducible* porque representa la naturaleza del dominio técnico.

Carga Extrínseca (Extraneous Load). Carga *innecesaria* impuesta por diseño instruccional deficiente que no contribuye al aprendizaje. Las imágenes 2D estáticas generan alta carga extrínseca al requerir *rotación mental* (usuario debe imaginar vistas no mostradas). Hegarty & Waller (2004) demostraron que la rotación mental consume recursos visuoes paciales finitos de la memoria de trabajo.

Reducción en este Proyecto. La interfaz 3D orbital *externaliza* la rotación mental al sistema (el usuario manipula directamente el objeto), convirtiendo carga extrínseca en interacción productiva.

Carga Relevante o Germana (Germane Load). Esfuerzo cognitivo dedicado a la construcción de esquemas mentales permanentes (aprendizaje profundo). Sweller et al. (2019) enfatizan que el diseño instruccional debe maximizar carga germana liberando recursos de memoria de trabajo.

Maximización en este Proyecto. La interacción directa con hotspots técnicos (click → información contextual) dirige recursos cognitivos hacia la integración de información estructural (ubicación + función + relaciones), facilitando construcción de modelo mental cohesivo.

Ecuación de Balanceo:

Navegación Espacial (Bowman et al., 2004; Darken & Sibert, 1996).. Bowman et al. (2004) en "3D User Interfaces: Theory and Practice" definen interacción 3D como "interacción humano-computador en la cual las tareas del usuario son ejecutadas directamente en un contexto espacial tridimensional".

Para este proyecto, controles orbitales (arcball rotation) siguen principios de Darken &

Sibert (1996) para prevenir desorientación:

- Rotación Constrained: Restringida a ejes Y (yaw) y X (pitch), eliminando roll no intuitivo.
- Punto de Enfoque Fijo: El centro del objeto permanece anclado (world origin), evitando deriva orbital.
- Suavizado de Movimiento: Interpolación ease-out (función cuadrática) reduce mareo cibernético.

Cognición Distribuida (Hutchins, 1995).. Hutchins en "Cognition in the Wild" postula que la cognición no reside únicamente en la mente, sino distribuida entre agente (usuario), artifacts (interfaz 3D), y entorno:

- Cámara orbital: Extensión cognitiva, externaliza rotación mental (Hegarty, 2004).
- Hotspots: Memoria externa, reducen necesidad de memorizar nombres técnicos.

Heurísticas de Usabilidad (Nielsen, 1994).. Nielsen formalizó 10 heurísticas. Las más relevantes para este proyecto:

1. Visibilidad del Estado: Feedback inmediato (cursor cambia, highlight en hover).
2. Correspondencia Sistema-Mundo Real: Terminología técnica coincide con nomenclatura de ingeniería.
3. Control y Libertad: Botón "Reset Camera" revierte rotación no deseada.
4. Reconocimiento vs Memoria: Hotspots siempre visibles (no requiere memorización de ubicaciones).

Rotación Mental y Visualización Espacial (Hegarty & Waller, 2004; Bartlett & Dorribo Camba, 2023).. Hegarty demostró que individuos con alta habilidad espacial usan estrategias holísticas (rotan objeto completo), mientras individuos con baja habilidad usan estrategias

piecemeal (rotan partes). Los controles orbitales democratizan esta capacidad al externalizar la rotación.

Investigación reciente de Bartlett & Dorribo Camba (2023) en *Spatial Cognition & Computation* confirma que la visualización espacial con modelos 3D interactivos facilita la interpretación gráfica de estructuras complejas, reduciendo la dependencia de habilidades espaciales innatas. Este hallazgo valida el uso de interfaces 3D manipulables para audiencias técnicas heterogéneas.

Teoría de Percepción Visual (Gestalt)

Principios de Gestalt (Wertheimer, 1923; Köhler, 1929)..

- Proximidad: Hotspots cercanos se agrupan visualmente.
- Similitud: Hotspots del mismo color indican misma categoría (azul=estructura, naranja=energía, gris=mecánica).
- Figura-Fondo: El hardware (figura) contrasta con fondo (gradiente neutral, no competitivo).

Teoría del Color (Ware, 2012; Miller, 1956).. Usar colores distinguibles para categorías (máximo 7±2 según Miller). Evitar rojo-verde para accesibilidad (deuteranopia afecta $\sim$8 % hombres).

Fundamentos Matemáticos de Optimización Gráfica

Presupuesto Poligonal..

$$P_{total} = \sum_{i=1}^{n} P_i \leq 50,000 \text{ triángulos}$$

donde P_i es el conteo de polígonos del modelo i (Akenine-Möller et al., 2018, Cap. 18).

Densidad de Texel Consistente..

$$\rho = \frac{R_{texture}}{A_{UV}} = 10{,}24\frac{px}{cm}$$

Reducción de Draw Calls (GPU Instancing)..

$$D_{optimizado} = \frac{D_{original}}{N_{instances}}$$

Frame Time Budget..

$$T_{frame} = T_{CPU} + T_{GPU} < 33{,}33ms$$

Modelo Matemático de Renderizado Físicamente Basado (PBR)

BRDF de Cook-Torrance (1982)..

$$f_{spec} = \frac{D \cdot F \cdot G}{4(\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{v})}$$

donde:

- D: Normal Distribution Function (GGX).
- F: Término de Fresnel (Schlick's approximation).
- G: Geometric Shadowing/Masking term.

Aproximación de Fresnel (Schlick, 1994)..

$$F = F_0 + (1 - F_0)(1 - \cos\theta)^5$$

GGX Distribution (Walter et al., 2007)..

$$D_{GGX} = \frac{\alpha^2}{\pi((\mathbf{n} \cdot \mathbf{h})^2(\alpha^2 - 1) + 1)^2}$$

donde $\alpha = roughness^2$.

Modelo Disney (Burley, 2012).. Modelo artista-friendly con parámetros intuitivos:

- Metallic (0-1): Interpola entre dieléctrico y conductor.
- Roughness (0-1): Controla directamente α en GGX.

Garantiza conservación de energía: $E_{reflected} + E_{absorbed} = E_{incident}$.

Marco Conceptual

Los siguientes conceptos técnicos fundamentan el proyecto (orden alfabético):

Albedo. Mapa de textura que define el color base de un material sin información de iluminación, input primario en workflows PBR.

Asset Bundle. Sistema de empaquetado de Unity que agrupa activos 3D en archivos comprimidos para carga dinámica, optimizando memoria.

Baking. Proceso de pre-calcular información compleja (iluminación, geometría de alta resolución) y almacenarla en mapas de textura para reducir costo computacional en tiempo real.

BRDF (Bidirectional Reflectance Distribution Function). Función matemática que describe cómo la luz es reflejada por una superficie en función del ángulo de incidencia y observación, fundamental para PBR.

Diffuse. Componente de reflexión mate o dispersa de un material, sin dirección preferente.

Draw Call. Comando enviado desde la CPU a la GPU para renderizar un conjunto de geometría, constituyendo el principal cuello de botella en aplicaciones con muchos objetos únicos.

Fragment Shader (Pixel Shader). Programa ejecutado por la GPU para cada píxel visible, determinando su color final mediante ecuaciones de iluminación.

Frame Time. Tiempo total requerido para renderizar un cuadro de imagen, compuesto por tiempo de CPU y GPU ($T_{frame} = T_{CPU} + T_{GPU}$), inversamente proporcional a FPS.

GPU Instancing. Técnica de optimización que renderiza múltiples copias de la misma malla con una sola llamada de dibujo, reduciendo sobrecarga de CPU en 99 % para objetos repetidos.

IL2CPP (Intermediate Language to C++). Backend de compilación de Unity que transpi-

la bytecode de C# a código C++, luego a código nativo (WASM para WebGL).

LOD (Level of Detail). Sistema que intercambia entre versiones de diferente complejidad poligonal según distancia a la cámara, manteniendo calidad visual mientras reduce costo computacional.

Metallic Map. Textura que define qué partes de un material son metálicas (1.0) vs dieléctricas (0.0), controlando la reflectancia a incidencia normal (F_0).

MVP (Producto Mínimo Viable). Versión de un producto con características mínimas suficientes para validar hipótesis mediante aprendizaje validado con usuarios reales (Ries, 2011).

Normal Map (Bump Map). Textura que almacena información de normales de superficie para simular detalles geométricos sin aumentar conteo de polígonos, esencial para baking de alta frecuencia.

Occlusion Culling. Técnica que evita renderizar objetos no visibles porque están bloqueados por otros objetos desde la perspectiva de la cámara.

PBR (Physically-Based Rendering). Modelo de iluminación que simula la interacción física de la luz con materiales usando conservación de energía y teoría de microfacetas.

Retopology. Proceso de reconstruir la topología de una malla 3D para optimizar estructura poligonal, típicamente pasando de NURBS/subdivisión a quads optimizados.

Roughness Map. Textura que define la microsuperficie de un material, controlando dispersión del lóbulo especular (0.0 = espejo perfecto, 1.0 = mate total).

Shader. Programa ejecutado en la GPU que determina el aspecto visual de vértices o píxeles, implementando el modelo de iluminación.

Specular. Componente de reflexión directa de un material, con dirección preferente hacia el ángulo de reflexión perfecto (Ley de Snell).

Texel Density. Relación entre resolución de textura y área de superficie 3D que cubre ($\rho = R_{texture}/A_{UV}$), medida en píxeles por centímetro, crucial para mantener nitidez uniforme.

UV Unwrapping. Proceso de proyectar la superficie 3D de un modelo en un espacio 2D (UV) para aplicar texturas, minimizando distorsión y maximizando aprovechamiento de resolución.

Vertex Shader. Programa ejecutado por la GPU para cada vértice de una malla, transformando su posición tridimensional a espacio de proyección.

WebAssembly (WASM). Formato binario de bajo nivel para código ejecutable diseñado para ejecución rápida en navegadores web, sirviendo como target de compilación para lenguajes como C++ y C#.

WebGL (Web Graphics Library). API de JavaScript basada en OpenGL ES que expone funcionalidad de aceleración gráfica por hardware directamente en el navegador web sin necesidad de plugins.

Metodología

Se utilizará una metodología de Investigación Aplicada con un enfoque de Design Science Research (DSR) (Hevner et al., 2004), estructurada en desarrollo Ágil (Scrum Adaptado) con Sprints de 4 semanas.

Framework de Design Science Research

El proyecto sigue el framework DSR de 6 etapas (Hevner et al., 2004):

1. Problem Identification: Brecha de información técnica en medios 2D.
2. Objectives: Prototipo que valida viabilidad de WebGL para visualización de hardware.
3. Design & Development: Implementación en Unity siguiendo principios de HCI.
4. Demonstration: Despliegue online accesible.
5. Evaluation: Pruebas de usabilidad (SUS), profiling de rendimiento.
6. Communication: Documentación de pipeline de optimización.

Desarrollo en Sprints

Sprint 1: Análisis y Fundamentación Técnica (Semana 1-4)

- Selección del modelo CAD de referencia (Dron de Alta Gama).
- Benchmarking: Pruebas de rendimiento en dispositivos objetivo (Snapdragon 7 series).
- Definición de KPIs: Polígonos ($<$50,000), Draw Calls ($<$50), VRAM ($<$150MB).

Sprint 2: Pipeline de Arte Técnico (Semana 5-8)

- Retopología: Conversión de mallas CAD (NURBS) a polígonos optimizados en Blender.
- UV Mapping: Organización de islas UV para maximizar densidad de texel.

- Baking: Transferencia de detalles de alta frecuencia a mapas de normales.

Sprint 3: Ingeniería e Implementación (Semana 9-16)

- Configuración de Unity URP con soporte WebGL 2.0.
- Programación de scripts C# (compilados a WASM) para sistema de "Despiece" (Exploded View).
- Implementación de UI responsiva.

Sprint 4: Validación y Despliegue (Semana 17-24)

- Profiling: Uso de Unity Memory Profiler para detectar fugas de memoria.
- Pruebas de Usuario: Evaluación mediante cuestionarios SUS (System Usability Scale).
- Despliegue: Publicación en servidor con compresión Gzip/Brotli.

Tamaño de Muestra para Pruebas de Usabilidad

Nielsen (2000) demostró que 5 usuarios detectan $\sim$85 % de problemas de usabilidad. Para este proyecto:

- Muestra: 8-12 usuarios (target: ingenieros/técnicos).
- Justificación: Supera el mínimo de Nielsen, permitiendo detección de issues menos frecuentes.
- Protocolo: Think-Aloud + SUS Questionnaire.

KPIs Técnicos (Cuantitativos)

1. Polygon Budget: $\sum P_i < 50,000$ triángulos.
2. Texel Density: $\rho = 10{,}24 \pm 2$ px/cm (±20 % tolerancia).
3. Draw Calls: $D < 50$ (GPU Instancing + Static Batching).

4. Frame Time: $T_{frame} < 33{,}33$ms (30 FPS) en Snapdragon 7 Gen 1.

5. Memory: Heap Usage $<$ 150MB (medido via Unity Profiler).

6. Load Time: TTI (Time to Interactive) $<$ 5s en 4G (10 Mbps).

Cronograma de Actividades

Tabla 2

Cronograma General del Proyecto

Fase	Actividad Principal	M1	M2	M3	M4	M5-6
1. Análisis	Rev. Lit. & Benchmarking	X				
	Definición KPIs	X				
2. Arte Técnico	Retopología		X			
	UV Mapping & Baking		X	X		
3. Implementación	Conf. Unity & Scripts C#			X	X	
	Integración 3D & UI			X	X	
4. Validación	Profiling & Pruebas				X	X
	Documentación & Entrega					X

Nota. M = Mes. Fases 3 y 4 tienen overlap intencional (desarrollo iterativo). *Fuente*. Autoría propia.

Recursos Necesarios

Tabla 3

Presupuesto Estimado de Recursos

RECURSO	DESCRIPCIÓN	PRESUPUESTO (COP)
1. Equipo Humano	Estudiante investigador (Dedicación intensiva 6 meses). Consulta puntual externa (si aplica).	2,000,000
2. Equipos y Software	Hardware: PC con GPU Dedicada (existente). Software: Unity 6 (Licencia Estudiante), Blender 4.0 (Open Source), VS Code (Gratuito). Hosting/Dominio (∼150,000).	150,000
3. Bibliografía	Acceso a bases de datos IEEE Xplore, ACM Digital Library (Biblioteca UNAD).	0 (Institucional)
4. Materiales	Assets 3D/Texturas (Opcional, ∼300,000). Incentivos Pruebas Usuario (8-12 x ∼15,000 = ∼180,000).	480,000
TOTAL		**2,630,000**

Nota. Presupuesto ajustado a 6 meses. Se priorizará software gratuito. No incluye matrícula. *Fuente*. Autoría propia.

Resultados o Productos Esperados

Tabla 4

Resultados o Productos Esperados

RESULTADO/PRODUCTO	INDICADOR	BENEFICIARIO
1. Prototipo de Software WebGL Funcional	- URL pública del prototipo. - Implementación verificada de 3+ características interactivas. - Cumplimiento de KPIs ($<$50K polígonos, $>$30 FPS).	- Comunidad Académica. - Industria del Hardware. - Estudiante.
2. Conjunto de Modelos 3D Optimizados	- Reducción tamaño archivo $\geq$ 90 %. - Archivos .glb disponibles. - Documentación de texel density.	- Plataforma web. - Estudiantes de Multimedia.
3. Documento de Trabajo de Grado	- Documento final aprobado. - Cumplimiento normas APA 7 UNAD. - Pipeline replicable documentado.	- Estudiante. - UNAD. - Comunidad Académica.
4. Informe de Evaluación de Usabilidad	- Informe entregado. - Identificación 3+ hallazgos clave UX. - Resultados SUS documentados.	- Estudiante. - Jurados.

Nota. Los indicadores buscan medir la consecución de los resultados clave del proyecto. *Fuente*. Autoría propia.

Referencias Bibliográficas

Akenine-Möller, T., Haines, E., & Hoffman, N. (2018). *Real-Time Rendering* (4th ed.). CRC Press.

Bartlett, K. A., & Dorribo Camba, J. (2023). The Role of a Graphical Interpretation Factor in the Assessment of Spatial Visualization: A Critical Analysis. *Spatial Cognition & Computation*, *23*(1), 1–30. https://doi.org/10.1080/13875868.2021.1987375

Blank, S. (2013). Why the Lean Start-Up Changes Everything. *Harvard Business Review*, *91*(5), 63–72.

Bowman, D. A., Kruijff, E., LaViola Jr, J. J., & Poupyrev, I. (2004). *3D User Interfaces: Theory and Practice*. Addison-Wesley.

Burley, B. (2012). Physically-Based Shading at Disney. *ACM SIGGRAPH 2012 Course Notes: Practical Physically Based Shading in Film and Game Production*. ACM.

Cook, R. L., & Torrance, K. E. (1982). A reflectance model for computer graphics. *ACM Transactions on Graphics*, *1*(1), 7–24. https://doi.org/10.1145/357290.357293

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, *24*(1), 87–114.

Darken, R. P., & Sibert, J. L. (1996). Wayfinding Strategies and Behaviours in Large Virtual Worlds. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96)*, 142–149. https://doi.org/10.1145/238386.238459

Denes, G., Hall, A., & Persson, M. (2024). Using WebGPU in a Godot Web Build. *Proceedings of the 2024 IEEE Conference on Games (CoG)*, 1–4. IEEE. https://doi.org/10.1109/CoG60054.2024.1064

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Houghton Mifflin.

Hegarty, M. (2004). Mechanical reasoning by mental simulation. *Trends in Cognitive Sciences*, *8*(6), 280–285.

Hegarty, M., & Waller, D. (2004). Spatial Abilities at Different Scales: Individual Differences in Aptitude-Test Performance and Spatial-Layout Learning. *Intelligence*, *32*(2), 151–176.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105.

Hutchins, E. (1995). *Cognition in the Wild*. MIT Press.

Khronos Group. (2011). *WebGL Specification 1.0*. https://www.khronos.org/registry/webgl/specs/latest/1.0/

Köhler, W. (1929). *Gestalt Psychology*. Liveright.

Mayer, R. E. (2005). *The Cambridge Handbook of Multimedia Learning*. Cambridge University Press.

Mayer, R. E. (Ed.). (2021). *The Cambridge Handbook of Multimedia Learning* (3rd ed.). Cambridge University Press. https://doi.org/10.1017/9781108894333

Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review*, *63*(2), 81–97.

Cabello, R. [mrdoob]. (2024). *three.js - JavaScript 3D Library* [GitHub repository]. GitHub. https://github.com/mrdoob/three.js (110,000 stars as of November 2024)

Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.

Nielsen, J. (2000). Why You Only Need to Test with 5 Users. Nielsen Norman Group. https://www.nngroup.com/you-only-need-to-test-with-5-users/

Norman, D. A. (2013). *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.

Paivio, A. (1986). *Mental Representations: A Dual Coding Approach*. Oxford University Press.

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business.

Schlick, C. (1994). An Inexpensive BRDF Model for Physically-based Rendering. *Computer Graphics Forum*, *13*(3), 233–246.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, *12*(2), 257–285.

Sweller, J., van Merriënboer, J. J. G., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, *31*(2), 261–292. https://doi.org/10.1007/s10648-019-09465-5

Unity Technologies. (2021). *Advances in Real-Time Rendering in Games: Part I*. ACM SIGGRAPH 2021 Courses. ACM. https://doi.org/10.1145/3450508.3464571

Unity Technologies. (2024). *Memory in WebGL*. Unity Documentation. https://docs.unity3d.com/Manual/webgl-memory.html

Unity Technologies. (2024). *WebGL: Interacting with browser scripting*. Unity Documentation. https://docs.unity3d.com/Manual/webgl-interactingwithbrowserscripting.html

Walter, B., Marschner, S. R., Li, H., & Torrance, K. E. (2007). Microfacet Models for Refraction through Rough Surfaces. *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, 195–206.

Ware, C. (2012). *Information Visualization: Perception for Design* (3rd ed.). Morgan Kaufmann.

Wertheimer, M. (1923). Untersuchungen zur Lehre von der Gestalt II. *Psychologische Forschung*, *4*(1), 301–350.

Yu, G., Liu, C., Fang, T., Jia, J., Lin, E., He, Y., Fu, S., Wang, L., Wei, L., & Huang, Q. (2023). A survey of real-time rendering on Web3D application. *Virtual Reality Intelligent Hardware*,

5(5), 390–394. https://doi.org/10.1016/j.vrih.2022.04.002