TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Desarrollo de una aplicación de modelado 3D

Motor de renderizado en tiempo real basada en SDFs

Autor
Pablo Cantudo Gómez

Directores
Juan Carlos Torres Cantero y Luis López Escudero

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Septiembre de 2025

Desarrollo software de una aplicación de modelado 3D basada en Signed Distance Functions (SDF)

Pablo Cantudo Gómez

Palabras clave: WebGPU, C++, ray marching, motor de renderizado, funciones de distancia con signo (SDF)

Resumen

Este trabajo presenta el desarrollo de Copper, un motor de renderizado 3D en tiempo real implementado en C++ utilizando la API WebGPU a través de Dawn. El motor emplea técnicas de ray marching sobre funciones de distancia con signo (SDF), lo que permite una representación eficiente y flexible de geometría tridimensional. Una de las principales ventajas de utilizar SDF es la capacidad de expresar de forma directa y compacta operaciones booleanas entre primitivas geométricas —como uniones, intersecciones o diferencias— mediante simples expresiones algebraicas. Además, a diferencia de los métodos tradicionales basados en mallas o polígonos, las SDF permiten aplicar operaciones booleanas suaves, como la unión con suavizado (smooth union), de forma prácticamente trivial desde el punto de vista computacional.

Este enfoque simplifica notablemente la construcción de formas complejas, evitando problemas típicos de la computación geométrica como el manejo de vértices, normales o topologías complejas. También se facilita la animación y transformación de objetos mediante funciones continuas. El motor incluye un sistema de sombreado en WGSL, con soporte para sombras suaves, operaciones modulares sobre la escena, seleccion de objetos, carga y guardado de modelos. Los resultados demuestran que el uso de SDF no solo ofrece un modelo más elegante para definir geometría, sino que permite construir escenas visualmente complejas con menos código y una mayor expresividad gráfica. En conjunto, el sistema demuestra la viabilidad técnica y creativa del uso de SDF en entornos modernos.

Development of a 3D modeling application based on Signed Distance Functions/Fields (SDF)

Pablo Cantudo Gomez

Keywords: WebGPU, C++, ray marching, rendering engine, Signed Distance Functions/Fields (SDF)

Abstract

This work presents the development of Copper, a real-time 3D rendering en- gine implemented in C++ using the WebGPU API through Dawn. The engi- ne employs ray marching techniques over Signed Distance Functions/Fields (SDF), enabling an efficient and flexible representation of three-dimensional geometry. One of the main advantages of using SDF is the ability to directly and compactly express Boolean operations between geometric primitives — such as unions, intersections, or differences— through simple algebraic ex- pressions. Furthermore, unlike traditional mesh- or polygon-based methods, SDF allows for smooth Boolean operations, such as smooth union, in a vir- tually trivial manner from a computational standpoint.

This approach greatly simplifies the construction of complex shapes, avoiding typical problems in geometric computing such as handling vertices, normals, or complex topologies. It also facilitates the animation and transformation of objects through continuous functions. The engine includes a shading system in WGSL, with support for soft shadows, modular scene operations, object selection, and model loading and saving. The results show that the use of SDF not only offers a more elegant model for defining geometry, but also enables the creation of visually complex scenes with less code and greater graphical expressiveness. Overall, the system demonstrates the technical and creative feasibility of using SDF in modern environments.

Yo, **Pablo Cantudo Gómez**, alumno de la titulación Grado en ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 78243264, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Pablo Cantudo Gómez

Granada a x de Septiembre de 2025.

D. **Tutores**, Profesores del Área de x del Departamento x de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Nombre***, ha sido realizado bajo su supervisión por **Tutores**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 11 de Noviembre de 2023.

Los directores:

Tutores

Agradecimientos

Quiero expresar mi agradecimiento a todas las personas que han contribuido directa o indirectamente a la realización de este trabajo. En primer lugar, a mi tutor Juan Carlos Torres Cantero y cotutor Luis López Escudero por la ayuda durante el desarrollo y la corrección del proyec- to, a mis amigos por las discusiones y las ideas, y a mi familia por su apoyo in- condicional.

Finalmente, me gustaría mencionar a la comunidad de desarrolladores y recursos abiertos, especialmente los foros, artículos y proyectos relacionados con WebGPU, SDF y ray marching, cuya documentación y ejemplos han sido una fuente de aprendizaje invaluable.

Índice general

Índice de figuras

Capítulo 1

Introducción

El desarrollo de los gráficos tridimensionales por computadora ha sido un campo de investigación activo y en constante evolución desde sus inicios. La capacidad de crear representaciones visuales de objetos y escenas en tres dimensiones ha revolucionado diversas industrias, desde el entretenimiento hasta la medicina y la ingeniería.
A medida que la tecnología avanza, también lo hacen las técnicas y herramientas utilizadas para generar gráficos 3D, lo que plantea nuevos desafíos y oportunidades para los investigadores y desarrolladores.

En sus inicios, la generación de gráficos 3D estaba limitada por la capacidad de cómputo y se basaba en *pipelines* gráficos fijos compuestos por etapas de transformación, iluminación y rasterización.
Posteriormente, con la llegada de los *shaders* programables en GPU (Nvidia GeForce 3 en 2001), fue posible sustituir los *pipelines* fijos por *pipelines* programables, lo que abrió un abanico de posibilidades para la creación de efectos visuales complejos y personalizados, como los algoritmos no basados en polígonos.
Esto impulsó la investigación en técnicas de representación más avanzadas, como el *ray tracing* y, en particular, el *ray marching*.

En el contexto del renderizado basado en funciones implícitas, las *Signed Distance Functions* (SDF) no son una invención reciente, sino que tienen sus raíces en trabajos mucho más antiguos.
El concepto de combinar funciones implícitas mediante operaciones booleanas se remonta al trabajo de Ricci en 1972 [Ric73], y fue ampliado en 1989 por B. Wyvill y G. Wyvill con el modelado de *soft objects* [WMW86].
Ese mismo año, Sandin, Hart y Kauffman aplicaron *ray marching* a SDF para renderizar fractales tridimensionales [HSK89].
Posteriormente, en 1995, Hart documentó de nuevo la técnica, aunque la denominó erróneamente *Sphere Tracing* [Har96].

La popularización moderna de las SDF en el ámbito del *renderizado en tiempo real* se debe en gran parte a la comunidad *demoscene*, especialmente

a partir de mediados de la década de 2000.
Trabajos como el de Crane (2005) y Evans (2006) introdujeron la idea de restringir el campo a una distancia euclidiana real, mejorando el rendimiento y la calidad visual.
Sin embargo, el uso de esta técnica no está tan extendido en aplicaciones de renderizado en tiempo real, a pesar de su potencial para crear gráficos visuales y eficientes.

1.1. Motivación

Como cualquier persona nacida en los dos mil, he crecido rodeado de videojuegos y el avance en la tecnología de gráficos 3D ha sido un aspecto fascinante de esta industria. Durante la carrera de informática, estudié diversas asignaturas relacionadas con gráficos por computadora, cuyos proyectos despertaron y consolidaron mi interés en este campo.

El desarrollo de motores gráficos y técnicas de renderizado siempre me ha resultado un área especialmente atractiva, no solo por su complejidad técnica, sino también por el impacto directo que tienen en sectores como el entretenimiento, la simulación o la realidad virtual. A lo largo de mis estudios me encontré con herramientas muy potentes, pero también con la dificultad que implica dominarlas o adaptarlas a entornos experimentales. Esto me llevó a plantearme la posibilidad de crear una aplicación propia que sirviera como espacio de exploración.

La motivación principal de este trabajo es profundizar en tecnologías emergentes, en concreto **WebGPU**, un estándar reciente que promete unificar el desarrollo gráfico multiplataforma con un acceso eficiente a las GPU modernas. Asimismo, me interesaba experimentar con el modelado mediante **funciones de distancia (SDF)**, que representan una alternativa flexible al modelado poligonal clásico. Considero que la combinación de ambas tecnologías constituye un terreno de investigación con un gran potencial, tanto en aplicaciones prácticas como en entornos educativos.

Finalmente, este proyecto me ofrece la oportunidad de afianzar mis conocimientos en programación gráfica, shaders y arquitecturas modernas de GPU, a la vez que desarrollo un software propio que pueda servir de base para futuros trabajos de investigación o aplicaciones más complejas en el ámbito del diseño 3D.

1.2. Descripción del problema

El campo del modelado y renderizado 3D ha estado tradicionalmente dominado por herramientas complejas y de gran envergadura, como Blender, Maya o 3ds Max. Si bien estas aplicaciones ofrecen una gran potencia

y versatilidad, presentan también limitaciones importantes: requieren elevados recursos de hardware, poseen curvas de aprendizaje pronunciadas y no siempre resultan adecuadas para entornos de experimentación ligera o proyectos educativos.

Por otro lado, las API gráficas más extendidas, como OpenGL o DirectX, han demostrado su eficacia a lo largo de los años, pero presentan restricciones en cuanto a eficiencia y portabilidad en plataformas modernas. El reciente estándar **WebGPU** surge como respuesta a estas carencias, ofreciendo un modelo de programación más cercano al hardware y multiplataforma, con el objetivo de unificar el desarrollo gráfico en navegadores y aplicaciones nativas.

En el ámbito del modelado, el paradigma poligonal sigue siendo el más utilizado, pero alternativas como las **funciones de distancia (SDF)** permiten representar geometrías complejas de manera más compacta y flexible, facilitando la combinación de primitivas y operaciones booleanas. Sin embargo, la integración de estas técnicas en aplicaciones prácticas todavía es limitada, especialmente en combinación con tecnologías emergentes como WebGPU.

El problema que aborda este trabajo consiste en la falta de herramientas ligeras que sirvan como demostración y entorno de experimentación para el modelado y renderizado basados en SDF sobre WebGPU.

1.3. Objetivos

Objetivo general

El objetivo principal de este Trabajo de Fin de Grado es el desarrollo de una aplicación de diseño 3D basada en **WebGPU** y en técnicas de **modelado mediante funciones de distancia (SDF)**, que permita explorar y demostrar el potencial de estas tecnologías como alternativa al modelado poligonal clásico y como herramienta de experimentación en el ámbito de los gráficos por computadora.

Objetivos específicos

Para alcanzar este objetivo general, se plantean los siguientes objetivos específicos:

- Investigar y comprender en profundidad el funcionamiento de la API WebGPU y su integración en aplicaciones nativas mediante la librería Dawn.
- Diseñar e implementar un motor de renderizado basado en *ray marching* sobre funciones de distancia, capaz de representar primitivas y combinaciones mediante operaciones booleanas y suaves.

- Incorporar técnicas de sombreado y efectos visuales (iluminación, sombras suaves) que mejoren la calidad del renderizado.
- Desarrollar una interfaz gráfica sencilla que permita al usuario interactuar con la escena y manipular las primitivas.

1.4. Estructura de la memoria

La presente memoria se organiza en los siguientes capítulos:

- **Introducción:** Se expone el contexto del trabajo, la motivación, los objetivos planteados y la justificación de la elección de las tecnologías empleadas.
- **Fundamentos teóricos:** Se revisan los conceptos clave de modelado y renderizado 3D, las funciones de distancia (*Signed Distance Functions, SDF*) y el estándar WebGPU, contextualizando el trabajo en el estado actual de la tecnología.
- **Arquitectura de la aplicación:** Se describe la estructura general de la aplicación Copper, detallando los principales módulos, componentes y la interacción entre ellos.
- **Implementación:** Este capítulo se divide en varias secciones:
  - **Ventana y motor de renderizado:** Se explica el proceso de inicialización de la ventana, integración con Dawn/WebGPU y el desarrollo del motor de renderizado basado en SDF y ray marching.
  - **Controles e interacción:** Se detalla la implementación de los controles de cámara y de manipulación de objetos mediante la interfaz gráfica.
  - **Shaders y efectos visuales:** Se describe el desarrollo de los shaders WGSL utilizados, incluyendo técnicas de sombreado, iluminación y efectos visuales implementados.
- **Pruebas y validación:** Se presentan las pruebas realizadas para evaluar el rendimiento, la estabilidad y la calidad visual de la aplicación, así como los resultados obtenidos.
- **Conclusiones y trabajos futuros:** Se realiza un balance del trabajo realizado, se revisa el grado de cumplimiento de los objetivos y se plantean posibles líneas de investigación y desarrollo futuras.
- **Bibliografía y anexos:** Se recopilan las referencias bibliográficas consultadas y se incluyen materiales complementarios relevantes para la comprensión y reproducibilidad del trabajo.

Capítulo 2

Fundamentos teóricos

El modelado y renderizado en 3D es posible gracias a una serie de técnicas matemáticas y computacionales que permiten representar, manipular y visualizar geometría en entornos virtuales. Copper se apoya principalmente en el modelado mediante funciones de distancia con signo (SDF), el renderizado por ray marching y el uso del estándar gráfico WebGPU. Este capítulo describe en profundidad cada uno de estos fundamentos.

2.1. Modelado tridimensional: Paradigmas y fundamentos

El modelado tridimensional tradicional utiliza mallas poligonales, donde los objetos se representan mediante vértices, aristas y caras conectadas entre sí. Este método, empleado en la mayoría de herramientas profesionales (*Blender, Maya, 3ds Max*), permite una gran flexibilidad, pero implica gestionar topología y almacenar grandes cantidades de datos, lo que complica la edición y la generación procedural.

Como alternativa, existen métodos de representación implícita, siendo los campos de distancia con signo (*Signed Distance Fields, SDF*) los más destacados. Una SDF es una función $f(\vec{x})$ que, para cada punto $\vec{x}$ del espacio, devuelve la distancia mínima a la superficie del objeto. El signo indica si el punto está en el interior (negativo), sobre la superficie (cero) o en el exterior (positivo). Este enfoque permite describir objetos mediante expresiones matemáticas, simplificando la combinación y manipulación de geometría compleja.

2.1.1. Ventajas de las SDF frente al modelado poligonal

- **Compacidad**: Las SDF están definidas por fórmulas matemáticas, no por listas de vértices, lo que reduce el espacio necesario para describir objetos.

- **Facilidad de combinación**: Pueden combinarse mediante operadores matemáticos (unión, intersección, resta) de forma eficiente.
- **Transformaciones geométricas**: Admiten fácilmente traslaciones, rotaciones y escalados aplicando transformaciones sobre la función.
- **Cálculo de normales**: La normal en un punto de la superficie se obtiene como el gradiente de la función de distancia.
- **Flexibilidad**: Permiten crear transiciones suaves entre objetos mediante operadores suavizados.

2.2. Funciones de distancia con signo (SDF)

Las SDF asignan a cada punto del espacio la distancia mínima a una superficie implícita. Formalmente, para una función $f(\vec{x})$, la superficie se define como el conjunto de puntos donde $f(\vec{x}) = 0$. Las SDF permiten describir primitivas básicas como:

- **Esfera**: $f_{esfera}(\vec{x}) = ||\vec{x} - \vec{c}|| - r$, donde $\vec{c}$ es el centro y r el radio.
- **Caja**: $f_{caja}(\vec{x}) = ||\max(|\vec{x} - \vec{c}| - \vec{s}, 0)|| + \min(\max(d_x, \max(d_y, d_z)), 0)$, donde $\vec{s}$ es el tamaño.

Las SDF pueden combinarse empleando operadores booleanos y suaves:

- **Unión**: $f_{union}(a, b) = \min(a, b)$
- **Intersección**: $f_{inter}(a, b) = \max(a, b)$
- **Resta**: $f_{resta}(a, b) = \max(a, -b)$
- **Unión suave**: Interpolación entre distancias y colores para crear transiciones continuas.

2.2.1. Propiedades y aplicaciones

Las SDF permiten:

- Evaluar la función en cualquier punto del espacio para detectar colisiones, calcular iluminación o generar geometría procedural.
- Calcular la normal en un punto como el gradiente $\nabla f(\vec{x})$.
- Construir geometría fractal y orgánica mediante funciones recursivas o combinaciones arbitrarias.

Se emplean en renderizado, simulación física, generación de terrenos y efectos visuales avanzados.

2.3. Ray Marching

El *ray marching* es una técnica de renderizado que, utilizando la SDF, avanza iterativamente un rayo en el espacio hasta aproximar la intersección con una superficie implícita.

El algoritmo sigue estos pasos:

1. Lanzar un rayo desde la cámara en una dirección determinada.
2. Evaluar la SDF en la posición actual para obtener la distancia mínima a la superficie más cercana.
3. Avanzar el punto a lo largo del rayo una distancia igual al valor obtenido.
4. Repetir hasta que la distancia sea menor que un umbral (colisión con la superficie) o se alcance un límite de pasos o distancia máxima.

2.3.1. Comparación con Ray Tracing

- **Ray tracing**: Calcula la intersección exacta con primitivas geométricas.
- **Ray marching**: Utiliza la distancia local para aproximar superficies implícitas y fractales.

Ray marching facilita la incorporación de sombras suaves, reflexión y refracción, y la renderización eficiente de geometría compleja.

2.3.2. Sphere Tracing

La técnica de **sphere tracing** es una variante eficiente del ray marching, utilizada específicamente para el renderizado de superficies implícitas definidas por funciones de distancia con signo (SDF). Su principal ventaja respecto a los métodos tradicionales de ray casting es que utiliza la propia SDF para calcular el avance óptimo en cada paso del rayo, evitando colisiones innecesarias y acelerando la detección de intersecciones.

El algoritmo se basa en la siguiente idea: desde el origen del rayo, en cada iteración se evalúa la SDF en la posición actual para obtener la distancia mínima a cualquier superficie. Este valor se interpreta como el radio de una esfera libre de obstáculos centrada en el punto actual. Así, se puede avanzar el rayo exactamente esa distancia sin riesgo de atravesar ninguna superficie. El proceso se repite hasta que la distancia es menor que un umbral prefijado (lo que indica que se ha alcanzado la superficie) o hasta que se supera una distancia máxima o número de pasos.

Sphere tracing es especialmente útil en escenas donde las funciones de distancia son suaves y bien definidas, y permite renderizar geometría compleja con costes computacionales bajos. Sin embargo, puede resultar menos eficiente en casos de superficies muy delgadas o SDFs poco continuas, donde el avance óptimo se reduce drásticamente.

La Figura 2.1 ilustra el funcionamiento del algoritmo: cada círculo representa el rango libre de obstáculos desde la posición actual del rayo, determinado por la evaluación de la SDF. El rayo avanza saltando de esfera en esfera hasta que la distancia mínima indica una colisión con la superficie.

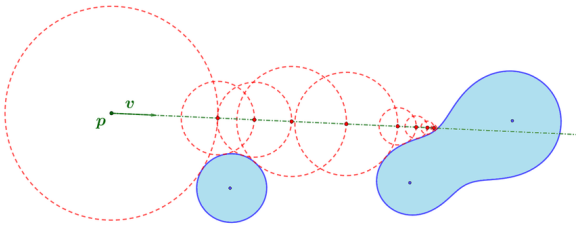

Figura 2.1: Representación visual del algoritmo de sphere tracing.

2.4. WebGPU: Acceso moderno a la GPU

WebGPU es un estándar gráfico de nueva generación que proporciona acceso eficiente y multiplataforma a la GPU, tanto en navegadores como en aplicaciones nativas. Sus principales características incluyen:

- Multiplataforma: Compatible con Windows, Linux, Mac y navegadores recientes.
- Eficiencia: Modelo de programación cercano al hardware, acceso directo a buffers y pipelines.
- Shaders en WGSL: Permite la programación de efectos personalizados y operaciones matemáticas avanzadas.

WebGPU ofrece ventajas frente a OpenGL y DirectX en control de recursos, computación general y portabilidad.

2.5. Shaders y lenguaje WGSL

Los **shaders** son programas que ejecutan operaciones matemáticas en la GPU para transformar vértices, calcular colores y simular efectos visuales.

WGSL (*WebGPU Shading Language*) es el lenguaje nativo de WebGPU, diseñado para expresar funciones de distancia, operadores booleanos, cálculos de iluminación y efectos visuales de forma eficiente.

- **Vertex shaders**: Transforman posiciones y atributos de vértices.
- **Fragment shaders**: Calculan el color final de cada píxel, aplicando modelos de iluminación como Blinn-Phong, efectos de sombras y combinaciones de SDF.

WGSL permite aprovechar la arquitectura de la GPU para realizar renderizado en tiempo real.

2.6. Herramientas auxiliares

El desarrollo de aplicaciones gráficas requiere gestionar ventanas, entrada de usuario y la interfaz gráfica:

- **GLFW**: Biblioteca multiplataforma para la gestión de ventanas y eventos.
- **ImGui**: Sistema de interfaz gráfica inmediata para la manipulación interactiva de primitivas y parámetros de escena.
- **GLM**: Biblioteca matemática para operaciones con vectores y matrices.
- **CMake**: Herramienta de compilación y gestión de dependencias.

Estas herramientas proporcionan la infraestructura básica para la interacción y visualización dentro del entorno de Copper.

Capítulo 3

Arquitectura

3.1. Tecnologías utilizadas

3.2. Estructura del proyecto

Capítulo 4

Implementación: Ventana

Capítulo 5

Implementación: Controles

Capítulo 6

Implementación: Shaders

Capítulo 7

Funcionalidades adicionales

Capítulo 8

Pruebas

Capítulo 9

Conclusiones

Bibliografía

[Ric73] A. Ricci. "A Constructive Geometry for Computer Graphics". En: *The Computer Journal* 16.2 (mayo de 1973), págs. 157-160. DOI: http://dx.doi.org/10.1093/comjnl/16.2.157.

[WMW86] Geoff Wyvill, Craig McPheeters y Brian Wyvill. "Data Structure for Soft Objects". En: *The Visual Computer* 2.4 (1986), págs. 227-234. DOI: 10.1007/BF01900346. URL: https://link.springer.com/article/10.1007/BF01900346.

[HSK89] John C. Hart, Daniel J. Sandin y Louis H. Kauffman. "Ray Tracing Deterministic 3-D Fractals". En: *Computer Graphics* 23.3 (jul. de 1989), págs. 289-296. DOI: 10.1145/74334.74340. URL: https://www.cs.drexel.edu/~david/Classes/Papers/rtqjs.pdf.

[Har96] John C. Hart. "Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces". En: *The Visual Computer* 12.10 (1996), págs. 527-545. DOI: 10.1007/s003710050084. URL: https://link.springer.com/article/10.1007/s003710050084.

[Quí08] Iñigo Quílez. *Raymarching Distance Fields*. https://iquilezles.org/articles/raymarchingdf/. Accessed: 2025-08-11. 2008.

Apéndice A

Código

Apéndice B

Escenas

Glosario

Constructive Solid Geometry (CSG) Método de modelado geométrico que combina primitivas mediante operaciones booleanas como unión, intersección y diferencia..

Demoscene Comunidad de programadores, artistas y músicos que crean producciones audiovisuales en tiempo real para mostrar destreza técnica y creatividad..

Función Implícita Función matemática que describe una superficie como el conjunto de puntos que satisfacen una ecuación dada, sin necesidad de una parametrización explícita..

Hypertexture Técnica de Ken Perlin y Louis Hoffert para renderizar volúmenes procedurales, que sirvió de base para el desarrollo del ray marching..

Ray Marching Técnica de renderizado que recorre un rayo en pasos discretos para encontrar intersecciones con superficies definidas implícitamente..

Signed Distance Function (SDF) Función que, dado un punto en el espacio, devuelve la distancia mínima a la superficie más cercana, con signo positivo si el punto está fuera y negativo si está dentro..

Smooth Blending Técnica para suavizar las transiciones entre primitivas geométricas en modelado implícito, evitando uniones abruptas..

Sphere Tracing Método propuesto por Hart en 1995 para recorrer campos de distancia en el renderizado de superficies implícitas..