https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145

Vol. 4, Núm. 1. enero-marzo 2026, págs. 37 - 49

Simulación de reflejos anidados en tiempo real mediante Render to Texture en el proyecto Museo de la Luz

Simulation of nested real-time reflections using Render to Texture in the project Museo de la Luz

José Larios Delgado

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

Universidad Nacional Autónoma de México

jlarios@unam.mx

ORCID: 0009-0003-5663-9962

Información del reporte:

Para citar este reporte técnico:

Larios Delgado, J. (2026). Simulación de reflejos anidados en tiempo real mediante Render to Texture en el proyecto Museo de la Luz. *Cuadernos Técnicos Universitarios de la DGTIC, 4* (1), páginas (37 - 49). https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145

Resumen

Se aborda la implementación en Unity de una simulación de reflejos anidados en entornos tridimensionales interactivos mediante la técnica *Render to Texture*. Con el objetivo de obtener un modelo capaz de reproducir en tiempo real el comportamiento físico de un periscopio, se priorizaron la exactitud física del fenómeno y el rendimiento del sistema. Se utilizó una metodología que consistió en generar cámaras asociadas a planos reflectantes, las cuales producen texturas de renderizado dinámicas, ajustando su posición y orientación en función del cálculo del vector reflejado, la dirección del vector de vista y la normal del plano que representa al espejo. El rendimiento del sistema se evaluó incrementando el número de reflejos y analizando métricas como el número de cuadros por segundo, el tiempo de renderizado y el uso de memoria. Se observó una reducción promedio de 40 cuadros por segundo y un aumento en el tiempo de renderizado de 1.28 milisegundos por cuadro por cada reflejo adicional. En cuanto a la exactitud física, la comparación con un periscopio real confirmó que la orientación final de los reflejos en la simulación coincide con la orientación observada físicamente, lo que permitió validar el algoritmo implementado. Se concluye que la simulación propuesta constituye una solución factible para representar reflejos anidados en tiempo real dentro de entornos interactivos para la web, ofreciendo un equilibrio adecuado entre exactitud visual y desempeño.

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 38 - 49

Palabras clave: Aprendizaje autodirigido, periscopio virtual, reflejo en tiempo real, renderizado a textura.

Abstract

The implementation of nested reflection simulation in interactive three-dimensional environments using the Render to Texture technique within Unity was addressed. The objective was to develop a model capable of reproducing the physical behavior of a periscope in real time, prioritizing both the physical accuracy of the phenomenon and system performance. The methodology consisted of generating cameras associated with reflective planes, which produced dynamic rendering textures by adjusting their position and orientation based on the calculation of the reflected vector, the view direction vector, and the normal of the plane representing the mirror. System performance was measured by increasing the number of reflections and analyzing metrics such as frames per second, rendering time, and memory usage. An average reduction of 40 frames per second and an increase of 1.28 milliseconds per frame for each additional reflection were observed. Regarding physical accuracy, comparison with a real periscope confirmed that the final orientation of the simulated reflections matched the physically observed orientation, thereby validating the implemented algorithm. It was concluded that the simulation represents a feasible solution for rendering nested reflections in real time within web-based interactive environments, offering an appropriate balance between visual accuracy and performance.

Keywords: *Real time reflection, render to texture, self–directed learning, virtual periscope.*

1. INTRODUCCIÓN

En la actualidad, los entornos digitales interactivos son una herramienta clave para comunicar conceptos de manera accesible y atractiva en espacios educativos y museísticos (Serrano Moral, 2014). Este tipo de herramientas permiten al público experimentar y fomentar el aprendizaje a través de la exploración activa.

En este contexto, el Museo de la Luz identificó la necesidad de contar con una exposición permanente que coadyuve a la comprensión del fenómeno de la reflexión de la luz, por lo que surgió la propuesta de crear un espacio digital interactivo que permitiera experimentar este fenómeno de manera remota y continua. Para ello, durante el 2025, un equipo multidisciplinario de la Dirección General de Cómputo y de Tecnologías de Información y Comunicación definió los objetivos y alcances para el proyecto, proponiendo un interactivo web que fomente la libre experimentación y el aprendizaje autónomo sin mediación docente, en concordancia con los enfoques contemporáneos que promueven la autorregulación y la exploración activa del conocimiento en entornos educativos digitales (Kemp, Davidson, Hurse, & Naik, 2024).

Se planteó un reto interactivo en donde el usuario se enfrenta a la tarea de construir un periscopio digital dentro de un ambiente simulado, un concierto multitudinario. Utilizando componentes como espejos y extensores, el usuario descubre los principios ópticos que le permitan visualizar el escenario.

Este desarrollo presentó dos desafíos técnicos centrales: la simulación de reflejos en tiempo real y su anidamiento, es decir, lograr que un espejo refleje la imagen de otro espejo sucesivamente. Se empleó el motor Unity, seleccionado por su capacidad de exportación a WebGL, su amplio conjunto de herramientas

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 39 - 49

para entornos 3D interactivos y su compatibilidad con técnicas avanzadas de renderizado[1], características ampliamente documentadas en estudios comparativos sobre motores de juego y entornos WebGL aplicados a la visualización interactiva (Zheng et al., 2023).

El objetivo del proyecto fue implementar una simulación de reflejos anidados en tiempo real, como los de un periscopio, dentro de un entorno educativo digital desarrollado en Unity para el proyecto Museo de la Luz.

2. DESARROLLO TÉCNICO

Simulación de reflejos en tiempo real

Existen múltiples técnicas para la simulación de reflejos, por ejemplo, el trazado de rayos en tiempo real, el cual presenta limitaciones importantes en contextos donde el rendimiento y la compatibilidad multiplataforma son prioritarios. En los últimos años, se han realizado numerosas mejoras en este algoritmo y se ha alcanzado un punto en el que los resultados son viables para tiempo real, gracias a técnicas avanzadas y hardware dedicado específicamente al trazado de rayos (Clemenz y Weydemann, 2021). Aunque tiene ventajas como la precisión de los reflejos que produce —incluyendo interreflexiones de manera nativa—, su dependencia de hardware especializado, como las GPU con núcleos para *Ray Tracing* (RT) (Tatzgern, et al., 2024), restringe su implementación en plataformas como WebGL .

Por esta razón, el uso de la técnica de *Render to Texture* (RTT) representó una alternativa eficiente y práctica para el proyecto, ya que permite obtener reflejos visualmente coherentes en tiempo real sin requerir hardware de aceleración de RT.

RTT permite capturar la escena desde una perspectiva específica y proyectar el resultado como una textura sobre una superficie, creando la ilusión de un reflejo coherente con el entorno y la posición del observador (Unity Technologies, 2023; Epic Games, 2024), la cual consiste en redirigir la salida del renderizado hacia una textura almacenada temporalmente en memoria. En lugar de mostrar la imagen directamente en pantalla, el motor gráfico la guarda en un *buffer*. Dicha textura puede aplicarse después a otro objeto dentro de la escena y actualizarse de forma dinámica en cada cuadro (Möller, Haines y Hoffman, 2018).

RTT en la simulación de reflejos

Para implementar reflejos en tiempo real usando RTT, se siguieron los siguientes pasos:

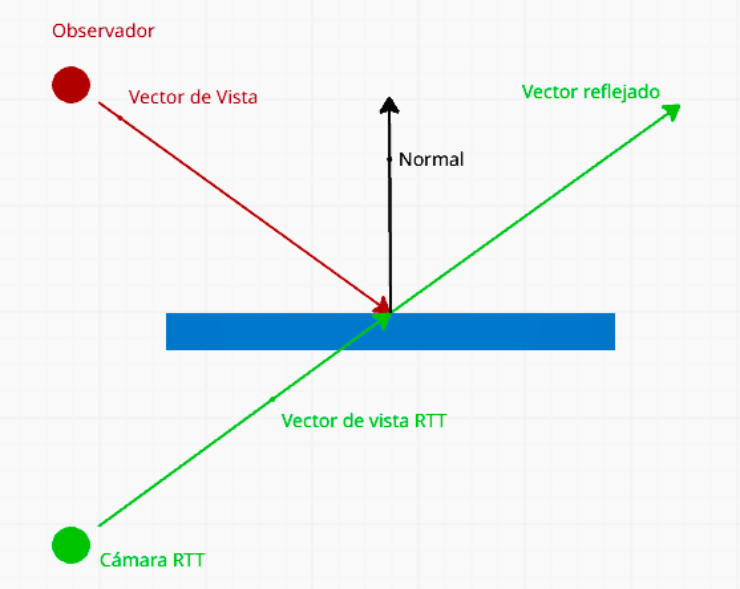
1. **Creación de una cámara para hacer el RTT**, cuya vista representa la "vista reflejada" de la escena. Esta cámara se posiciona de manera simétrica respecto al plano que simula el espejo.
2. **Dibujo de la escena de la cámara** a una textura, guardando lo que se "vería" en el reflejo. La dirección del vector de vista de la cámara debe coincidir con la dirección del vector reflejado, calculado a partir del vector de vista del observador y la normal del plano reflectante (ver Figura 1).
3. **Asignación de la textura (reflejo) como material del objeto reflectante,** textura que se le aplica al espejo.

1 Para la definición de éste y otros conceptos técnicos, véase el glosario contenido en el Anexo A.

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 40 - 49

Figura 1

Esquema de Render to Texture



Esta técnica permite obtener un reflejo coherente, que se actualiza en tiempo real, corresponde al movimiento del usuario o de los objetos en la escena.

Ventajas: Generar reflejos dinámicos y coherentes con el entorno (Shirley et al., 2021), permitir el reflejo de objetos que se mueven, de luces y efectos visuales en tiempo real, sin requerir del uso de técnicas de *ray tracing*; se puede implementar en múltiples plataformas, incluyendo WebGL.

Limitaciones: Implica dibujar la escena por lo menos dos veces cada cuadro, lo que incrementa el costo computacional (Akenine-Möller et al., 2018), en superficies curvas o con múltiples reflejos, el manejo de las cámaras encargadas del RTT se vuelve más complejo y propenso a errores (Akenine-Möller et al., 2018).

Simulación de reflejos anidados

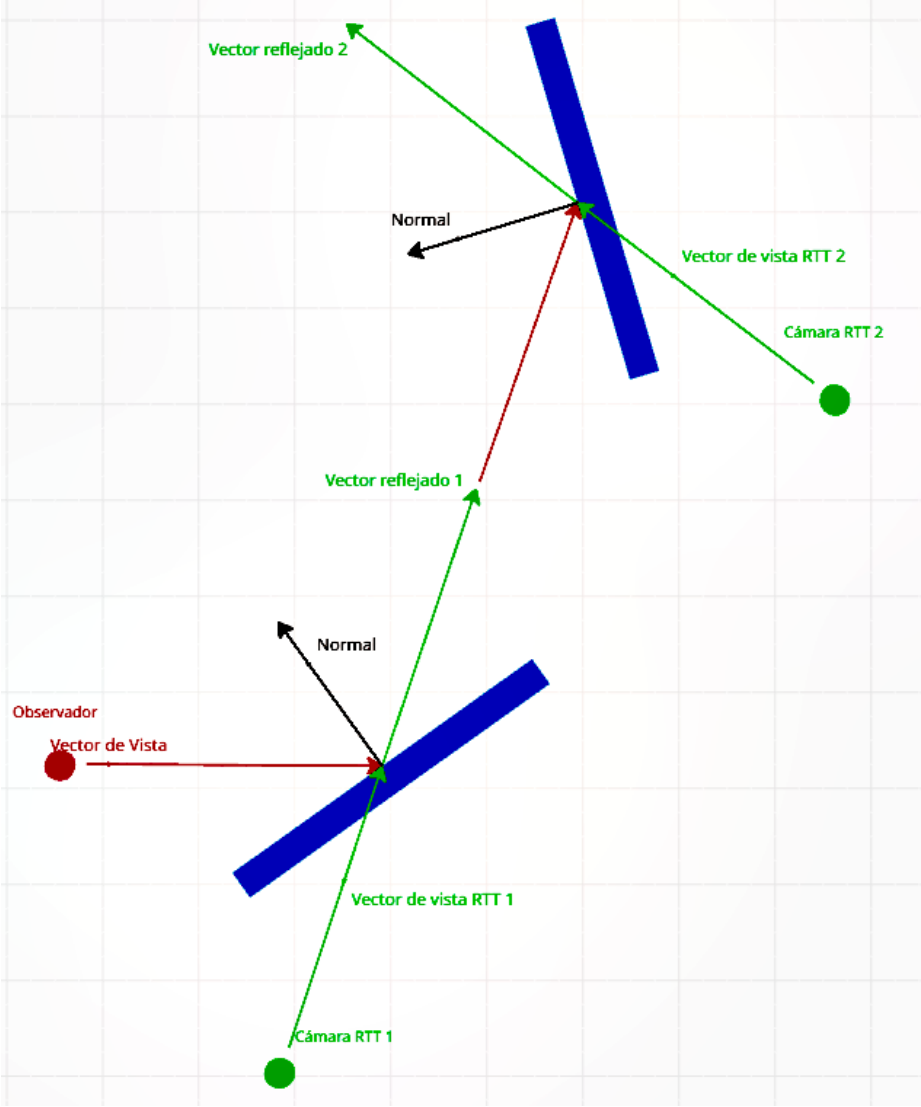
Una vez resuelto el reflejo de un espejo, se abordó el desarrollo de una solución para simular el funcionamiento del periscopio de forma digital. Esto requirió el anidamiento de los reflejos simulados de múltiples espejos.

El anidamiento de reflejos significa que, en tiempo real, el vector reflejado del primer espejo debe actualizarse en función de la dirección del vector de vista de la cámara principal (observador). El vector resultante actúa a su vez como el nuevo vector de vista para calcular el reflejo del segundo espejo, y así sucesivamente con los demás planos reflectantes (ver Figura 2).

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 41 - 49

Figura 2

Esquema reflejos anidados



Nota. Resalta la necesidad de controlar el orden de dibujo de las cámaras que realizan el RTT, ya que se observa que el primer reflejo simulado debe corresponder al último espejo, mientras que el último reflejo debe corresponder al primer espejo. Esto se debe a que este orden (inverso) garantiza la coherencia de la simulación y evita que ocurran errores por falta de sincronización en el dibujo de las texturas.

2.1 METODOLOGÍA

La implementación se realizó siguiendo un enfoque de prototipado evolutivo, ya que permite construir y refinar progresivamente las funcionalidades de un sistema, facilitando la validación temprana de resultados y la corrección iterativa del diseño (Pressman & Maxim, 2020).

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 42 - 49

A continuación, se describe el procedimiento que se siguió para desarrollar un prototipo en Unity que utiliza la técnica de *Render to Texture* para que simule el reflejo de un espejo.

Implementación de Render to Texture en Unity

Se creó un recurso de tipo RenderTexture y se le nombró *MirrorRT_A*; se configuró con los siguientes parámetros:

- **Resolución:** 1024 × 1024 (modificable para optimizar el rendimiento).
- ***Depth Buffer*:** 24 bits (información de profundidad).
- ***Color Format*:** R8G8B8A8.

En la jerarquía de la escena, se creó una nueva cámara, *RTCameraA*. La textura *MirrorRT_A* se le asignó en el campo *TargetTexture* de la cámara *RTCameraA* en el editor de Unity, con esto se logra que la salida del renderizado se guarde en MirrorRT_*A*.

Aplicación del reflejo

Se creó un nuevo material con el *shader* estándar de Unity y se le asignó *MirrorRT_A* como la textura base (*Base Map*). Este material se aplicó al plano que funciona como espejo. En tiempo de ejecución, la textura se actualiza, mostrando el punto de vista de la cámara *RTCameraA* (Möller, Haines y Hoffman, 2018).

Actualización de la posición y orientación del reflejo

La cámara que simula el reflejo (*RTCameraA*) se ubica y orienta sobre el vector reflejado de la vista del observador (*MainCamera*) respecto al plano del espejo (ver Figura 1). Su posición se calcula reflejando la posición de la cámara principal a través del plano reflejante y a la misma distancia que el observador está del espejo. Asimismo, su orientación, correspondiente al vector de vista, coincide con la dirección del vector reflejado.

El cálculo se realiza mediante el siguiente pseudocódigo:

```
posicion_camara_reflejo = Reflejo(posicion_camara_principal - posicion_espejo, normal_espejo) + posicion_espejo

orientacion_camara_reflejo = Reflejo(posicion_camara_principal - posicion_espejo, normal_espejo)
```

Recordando que la función de reflejo se define como (Akenine-Möller et al., 2018):

$$R = I - 2(I \cdot N)N$$

donde:

R-> vector reflejado

I-> vector incidente

N-> vector normal del plano reflectante (normalizado)

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 43 - 49

Desarrollo del anidamiento de reflejos

Para obtener el anidamiento de reflejos, se desarrolló un *script* nombrado MirrorController, el cual se asignó como un componente de la cámara encargada del reflejo (*RTCameraA*). Este *script* se encarga de actualizar la posición y orientación de la cámara en el método *Update()*, e implementa el pseudocódigo antes descrito en lenguaje C#.

El *script MirrorController* depende de parámetros configurables:

- **MainCam:** cámara desde la cual se observa el reflejo.
- **MirrorGO:** plano que representa el espejo.

De esta forma, el anidamiento se logra encadenando cámaras y espejos de la siguiente manera (ver Figura 2):

Primer espejo (RTCameraA), parámetros en *MirrorController***:**

MainCam: Main Camera (observador)

MirrorGO: MirrorA

Segundo espejo (RTCameraB), parámetros en *MirrorController***:**

MainCam: RTCameraA (primer espejo)

MirrorGO: MirrorB

Finalmente, para controlar el orden de dibujo de las cámaras, se configuró la propiedad *Priority* en el editor. Una prioridad más alta indica que la cámara se renderiza antes que las cámaras que tienen una prioridad menor. Así pues, la cámara del último reflejo se configuró con la prioridad más alta (5) y se asignaron valores decrecientes hasta llegar a la cámara del primer reflejo con la prioridad más baja (1).

3. RESULTADOS

Como resultado de la implementación anteriormente descrita, se identificaron dos aspectos fundamentales: Primero, el rendimiento de la simulación y sus límites (cuántos espejos pueden simularse y cuál es el impacto cada vez que se agrega un espejo más). Segundo, la correspondencia de la simulación respecto al fenómeno real de reflexión.

Rendimiento de la simulación

Se buscó determinar el impacto de agregar más espejos a la simulación, analizando la variación en el rendimiento (cuadros por segundo [FPS] y tiempo de renderizado por cuadro) al incrementar el número de cámaras que usan RTT en una escena al usar la misma configuración de hardware para todos los casos (ver Anexo A).

A continuación, se describe el procedimiento que se siguió: se tomó como base una escena desarrollada como prototipo para la aplicación y se hizo la implementación de cuatro espejos, se configuraron según lo descrito en el desarrollo. Posteriormente, al correr la simulación, se midió el rendimiento con un espejo

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 44 - 49

activo durante 30 segundos y se registraron los valores promedio de FPS, CPU y uso de memoria, que reportó el Profiler de Unity. Este proceso se repitió con dos, tres y cuatro espejos para generar la siguiente tabla.

Tabla 1

Resultados prueba de rendimiento

Nº de RTT	FPS promedio	CPU ms/frame	Uso de memoria (RTT) mb
1	246	4.05	341.5
2	195	5.12	395
3	159	6.27	446
4	126	7.9	497.5

3.1 ANÁLISIS

Se observó una disminución de aproximadamente 40 FPS por cada espejo adicional, también hubo un incremento promedio de 1.28 ms en cada cuadro (ver Tabla 1). Esto fue previsible debido a que cada RTT extra requiere un renderizado completo adicional. Con esta información, se proyectó un punto de inflexión a partir del cual la incorporación de más espejos degrada notoriamente el rendimiento y se estimó que, a partir de alrededor de seis espejos, la tasa de cuadros por segundo desciende al rango de 40 a 30 FPS.

Si se compara con el uso de técnicas como *Real-Time Rendering of Glossy Reflections*, el trazado de rayos por sí mismo tiene un costo significativo en el desempeño; se observan tiempos de renderizado por *frame* de ~5.83 ms a 1080p usando una GPU moderna como la AMD Radeon RX 7900 XTX. (Eto, Meunier, Harada, & Boissé, 2023).

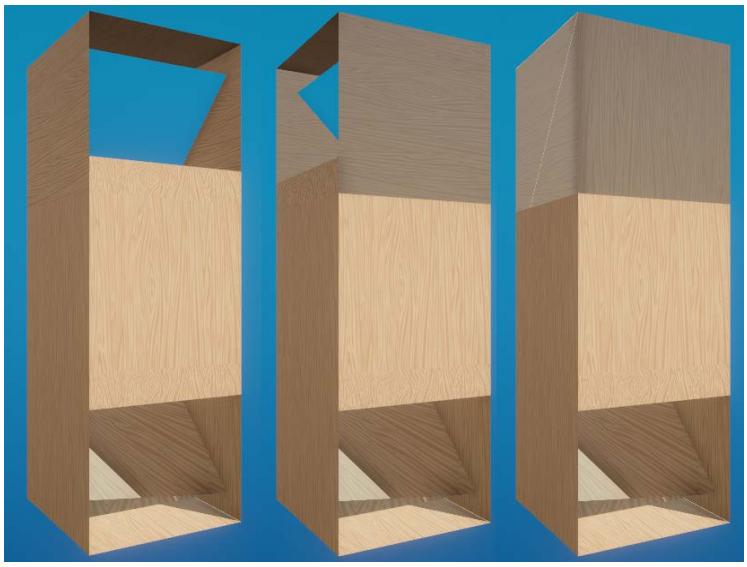
Exactitud de la simulación

Para evaluar la exactitud del modelo con el fenómeno físico, se ensamblaron tres configuraciones diferentes (ver Figura 3) usando dos espejos en un periscopio real y se observó la orientación del reflejo final (reflejo del primer espejo).

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 45 - 49

Figura 3

Configuraciones del periscopio virtual y real



Se observó que:

- En la primera configuración, el reflejo aparece invertido (rotado 180°).
- En la segunda, el reflejo se ve rotado 90° en sentido antihorario.
- En la tercera, el reflejo se muestra correctamente orientado.

Estas mismas configuraciones se reprodujeron en la aplicación, obteniéndose los siguientes resultados :

Figura 4

Reflejo simulado, primera configuración

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 46 - 49

Figura 5

Reflejo simulado, segunda configuración

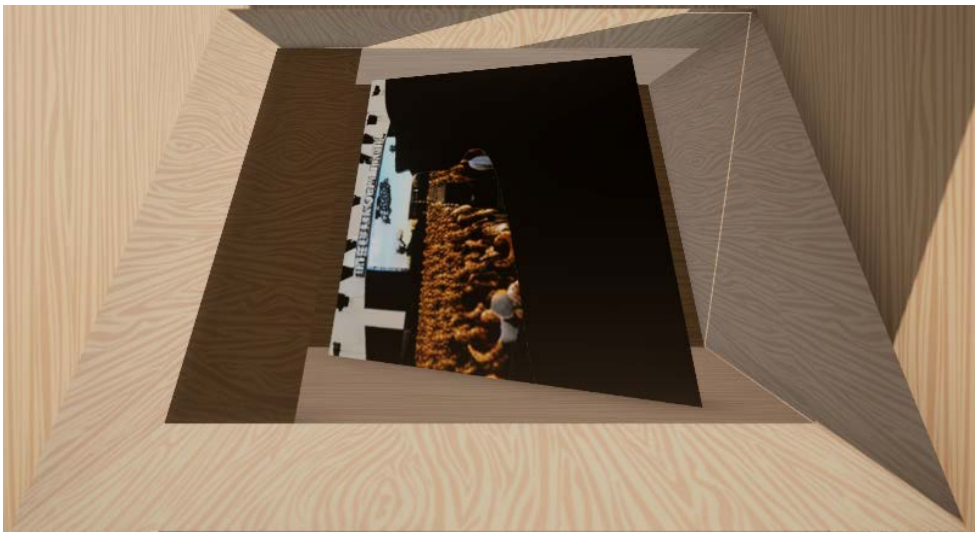
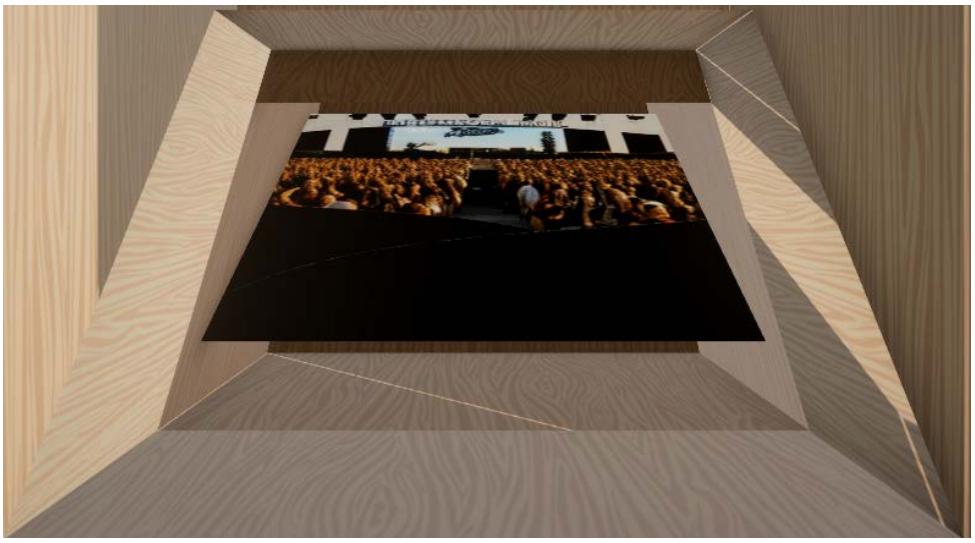


Figura 6

Reflejo simulado, tercera configuración



Observando las Figuras 4, 5 y 6, es evidente que los resultados de la simulación concuerdan fielmente con las características de los reflejos observados en el periscopio real. Esta prueba confirmó la validez del modelo de reflexión anidada implementado en el proyecto.

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 47 - 49

4. CONCLUSIONES

Con base en los resultados, se puede afirmar que se desarrolló un sistema capaz de simular reflejos anidados en tiempo real, y se resolvieron dos principales desafíos técnicos: la representación precisa del comportamiento físico de un periscopio y la optimización del rendimiento de la simulación para su correcta ejecución en un entorno WebGL.

Las pruebas de exactitud física (ver Anexo B) mostraron que la orientación del reflejo que se observó en la simulación coincide con el reflejo en el periscopio físico, esto sirvió para validar el modelo de reflejos anidados propuesto. En cuanto al rendimiento, se pudo determinar que 4 cámaras con RTT es el máximo número factible para usar en el proyecto sin afectar la tasa de cuadros mínima deseada (30FPS). Con este número máximo de espejos, aún es posible crear escenarios interesantes para la experimentación del usuario y el rendimiento es aceptable para funcionar en aplicaciones WebGL.

Por lo anterior, se concluyó que la implementación cumplió satisfactoriamente con el objetivo planteado, al lograr una simulación que combina exactitud física, interactividad y visualización en tiempo real en entornos WebGL. Como trabajo a futuro, se prevén oportunidades de mejora en la programación de optimizaciones en la gestión de los recursos gráficos que puedan incrementar el desempeño de la implementación en las pruebas de rendimiento.

REFERENCIAS

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

Clemenz, C., & Weydemann, L. (2021). Reflection techniques in real-time computer graphics. *KoG : Journal of the Croatian Computer Graphics Society, 25*(25), 87–95. https://doi.org/10.31896/k.25.10

Epic Games. (s. f.). *Textures in Unreal Engine. Epic Games Documentation.* https://dev.epicgames.com/documentation/en-us/unreal-engine/textures-in-unreal-engine

Eto, K., Meunier, S., Harada, T., & Boissé, G. (2023). Real-Time Rendering of Glossy Reflections Using Ray Tracing and Two-Level Radiance Caching. In *SIGGRAPH Asia 2023 Technical Communications*. Association for Computing Machinery (ACM). https://doi.org/10.1145/3610543.3626167

Kemp, K. L., Davidson, C. J., Hurse, D. N., & Naik, A. R. (2024). Exploring educational experiences that correlate with self-directed learning in college students seeking to pursue science, technology, engineering, math, and medical (STEMM) fields. *Frontiers in Education, 9*, Article 1393493. https://doi.org/10.3389/feduc.2024.1393493

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

Serrano Moral, C. (2014). ¿Museos del futuro? Comunicación, educación e interactividad. *International Journal of Educational Research and Innovation (IJERI), 2*, 129-140. Recuperado de https://www.upo.es/revistas/index.php/IJERI/article/view/1142

Shirley, P., Ashikhmin, M., Marschner, S., & Gleicher, M. (2021). *Fundamentals of Computer Graphics* (5th ed.). CRC Press.

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 **Vol. 4, Núm. 1.** enero-marzo 2026, págs. 48 - 49

Tatzgern, W., Weinrauch, A., Stadlbauer, P., Mueller, J. H., Winter, M., & Steinberger, M. (2024). Radiance caching with on-surface caches for real-time global illumination. *Proceedings of the ACM on Computer Graphics and Interactive Techniques, 7* (3), Article 38. https://doi.org/10.1145/3675382

Unity Technologies. (s. f.). *Render Texture* (Unity Manual). Recuperado de https://docs.unity3d.com/Manual/class-RenderTexture.html

Zheng, Y., Merchant, A., Laninga, J., Xiang, Z. X., Alshaebi, K., Arellano, N., Romaniuk, H., Fai, S., & Sun, D. H. (2023). Comparison of characteristics of BIM visualization and interactive application based on WebGL and game engine. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLVIII-M-2-2023,* 1671–1677. https://doi.org/10.5194/isprs-archives-XLVIII-M-2-2023-1671-2023

https://doi.org/10.22201/dgtic.30618096e.2026.4.1.145 Vol. 4, Núm. 1. enero-marzo 2026, págs. 49 - 49

ANEXO A. GLOSARIO

Buffer. Área de memoria temporal donde se almacenan datos (como imágenes, vértices o texturas) antes de ser procesados o enviados a la GPU durante el renderizado.

Cuadros por segundo (FPS). Medida que indica cuántas imágenes completas se muestran en pantalla cada segundo. A mayor FPS, mayor fluidez visual en juegos y aplicaciones gráficas.

Trazado de Rayos (*Ray Tracing*). Técnica avanzada de renderizado que simula el comportamiento real de la luz mediante el trazado de rayos, logrando reflejos, sombras e iluminación más realistas.

Renderizado. Proceso mediante el cual la computadora genera una imagen o animación a partir de modelos, texturas, luces y cámaras.

WebGL. API basada en JavaScript que permite renderizar gráficos 2D y 3D acelerados por hardware directamente en navegadores web, sin necesidad de plugins.

ANEXO B. CARACTERÍSTICAS DEL EQUIPO CON EL QUE SE HICIERON LAS PRUEBAS DE RENDIMIENTO

Procesador	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60 GHz
RAM instalada	64.0 GB
Tarjeta gráfica	NVIDIA GeForce RTX 2080 (8 GB), Intel(R) HD Graphics 630 (128 MB)
Tipo de sistema	Sistema operativo de 64 bits, procesador basado en x64