https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170

Vol. 4, Núm. 2. abril-junio 2026, págs. 29 - 48

Modelo lógico para la transferencia de líquidos en entornos tridimensionales web e inmersivos

Logical Model for Liquid Transfer in Web and Immersive Three-Dimensional Environments

Información del reporte:

Para citar este reporte técnico:

Cruz Lovera, T. M. (2026). Modelo lógico para la transferencia de líquidos en entornos tridimensionales web e inmersivos. *Cuadernos Técnicos Universitarios de la DGTIC, 4*(2), páginas (29 - 48). https://doi.org/10.22201/dgtic.30618096e.2026.4.2.169

Tayde Martín Cruz Lovera

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

Universidad Nacional Autónoma de México

taydevr@comunidad.unam.mx

ORCID:0009-0003-9519-8805

Resumen

Este trabajo presenta el diseño y la implementación de un modelo lógico para la transferencia de líquidos entre objetos virtuales dentro de una aplicación interactiva tridimensional. En lugar de utilizar simulación continua de fluidos físicos, el enfoque propuesto representa el comportamiento del líquido mediante estados lógicos y reglas de transferencia. Esto permite controlar tanto el volumen total como su composición, reduciendo además la complejidad computacional del sistema.

El modelo se basa en la identificación de las entidades clave involucradas en el proceso (recipientes, instrumentos de laboratorio y líquido) y está estructurado siguiendo una arquitectura modular que separa la lógica del sistema, la representación visual y los mecanismos de interacción. Esta separación permitió trabajar los componentes de forma independiente y reutilizar el modelo en distintos contextos de interacción. La representación visual del líquido refleja el volumen y la composición mediante cambios en el nivel y apariencia dentro de los recipientes virtuales usando *shaders*.

El modelo fue diseñado, implementado y evaluado mediante pruebas funcionales en una aplicación desarrollada en Unity y se aplicó a un instrumento de laboratorio virtual tridimensional, concretamente una micropipeta. La solución se logró implementar tanto en una aplicación web como en una aplicación de realidad

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 **Vol. 4, Núm. 2.** abril-junio 2026, págs. 30 - 48

virtual, lo que demostró su adaptabilidad a diferentes plataformas y manteniendo un comportamiento lógico consistente.

Palabras clave: Abstracción funcional, aplicación interactiva, laboratorios virtuales, realidad virtual, Unity.

Abstract

This work presents the design and implementation of a logical interaction model for liquid transfer between virtual objects within a three-dimensional interactive application. Instead of relying on continuous physical fluid simulation, the proposed approach represents liquid behavior through logical states and transfer rules. This enables control over both total volume and composition while reducing the computational complexity of the system.

The model is based on the identification of the key entities involved in the process (containers, instruments, and liquid) and is structured following a layered architecture that separates system logic, visual representation, and interaction mechanisms. This separation allowed components to be developed independently and enabled the reuse of the model in different interaction contexts. The visual representation of the liquid is derived from its logical state, showing volume and composition through changes in level and appearance within the virtual containers using shaders.

The model was implemented in an application developed with Unity and applied to a three-dimensional virtual instrument, specifically a micropipette, allowing its behavior to be evaluated in scenarios requiring precise and repeated liquid transfers. The solution was deployed both as a web application and as a virtual reality application, demonstrating its adaptability across platforms while maintaining consistent logical behavior

Keywords: *Functional abstraction, interactive application, Unity, virtual laboratories, virtual reality.*

1. INTRODUCCIÓN

A finales de 2021, la Facultad de Odontología de la UNAM se acercó al Observatorio Ixtli (ahora LAD UNAM) en busca de una alternativa basada en realidad virtual para llevar a cabo prácticas de laboratorio. Esta iniciativa surgió debido a que, durante la pandemia, las clases se impartían mediante videos, lo que impedía replicar los procesos en casa.

A partir de ese acercamiento inicial, se realizó una colaboración donde se desarrollaron interactivos tridimensionales correspondientes a dos prácticas de laboratorio, cuya primera implementación en versión web se realizó entre 2023 y 2024. Posteriormente, durante 2025 y 2026, se llevó a cabo la adaptación e implementación de las versiones en realidad virtual, etapa en la que se centra el presente reporte técnico.

En los últimos años, las aplicaciones interactivas tridimensionales se han consolidado como una herramienta relevante en contextos educativos, particularmente a partir del uso de entornos de realidad virtual aplicados a la enseñanza y el entrenamiento, los cuales han mostrado un amplio potencial en múltiples dominios educativos (Radianti *et al.*, 2020). Los laboratorios virtuales representan un caso particular de estas aplicaciones, ya que permiten la simulación de procedimientos experimentales sin los

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 31 - 48

riesgos y costos asociados al entorno físico real. En estas aplicaciones, la interacción debe ser coherente y comprensible, reflejando el comportamiento esperado de los instrumentos y materiales representados.

Uno de los procesos más comunes en entornos de laboratorio es la transferencia de líquidos entre recipientes, ya sea mediante instrumentos especializados o de forma directa. La representación de la transferencia de líquidos en entornos interactivos no requiere necesariamente una simulación física completa del comportamiento del fluido. A pesar de que existen técnicas avanzadas para simular dinámicas complejas de fluidos, estas implican un alto costo computacional derivado del cálculo numérico continuo de variables físicas, innecesario para la interacción planteada en este trabajo.

El objetivo del presente reporte técnico, en el contexto del desarrollo de una aplicación interactiva tridimensional, es proponer un modelo lógico de interacción desacoplado de los mecanismos de entrada que permita la transferencia de líquidos entre objetos virtuales en entornos web y de realidad virtual, de modo que pueda integrarse con distintos sistemas sin modificar su lógica interna.

2. DESARROLLO TÉCNICO

El desarrollo técnico de este trabajo describe la construcción y el uso de un modelo lógico para la transferencia de líquidos entre objetos virtuales dentro de una aplicación interactiva tridimensional. A partir de la identificación de los elementos involucrados en el proceso, recipientes, instrumentos y líquido, se propuso una solución basada en una abstracción lógica que sustituye el uso de simulación física.

En entornos interactivos en tiempo real, como aplicaciones web y de realidad virtual autónomas, la simulación física de fluidos presenta desafíos significativos de rendimiento y complejidad. Estudios recientes sobre modelado de fluidos en contextos inmersivos muestran que incluso las soluciones optimizadas requieren técnicas especializadas para mantener tasas de actualización estables, lo que limita su viabilidad en aplicaciones interactivas generales (Cen *et al*., 2024).

Con base en estas consideraciones, el modelo se definió como una alternativa para regular la transferencia de líquidos entre objetos virtuales, evitando el uso de simulaciones físicas, mediante una abstracción lógica que permite controlar el proceso, así como la cantidad y composición de líquido en cada objeto virtual.

La literatura reciente en ingeniería de *software* subraya la importancia de la abstracción como estrategia para abordar la complejidad propia de los sistemas, lo que permite centrar el diseño en el comportamiento relevante del sistema sin incorporar detalles innecesarios (Bencomo *et al*., 2024).

2.1 METODOLOGÍA

Como parte del proceso de desarrollo, se identificaron las siguientes necesidades y requerimientos del sistema:

- Representar la transferencia de líquidos sin recurrir a simulaciones físicas, dado que no se requiere un comportamiento realista de fluidos.
- Controlar el volumen total y la composición del líquido durante la transferencia.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 32 - 48

- Permitir transferencias consecutivas.
- Poder configurar o variar el volumen a transferir.
- Mantener consistencia lógica entre plataformas (web y realidad virtual).

A partir de estos requerimientos, se definió el modelo lógico propuesto, desarrollado mediante el siguiente proceso metodológico. Este se dividió en cuatro etapas principales:

1. Análisis conceptual: identificación de las entidades involucradas (recipientes, instrumentos y líquido), así como de sus relaciones dentro del proceso de transferencia.
2. Diseño del modelo: definición de los estados, reglas de transferencia y estructura de datos que conforman el modelo lógico.
3. Implementación: integración del modelo en una aplicación desarrollada en Unity.
4. Pruebas: ejecución de escenarios controlados para validar su comportamiento mediante pruebas funcionales.

2.2 MODELO LÓGICO PARA TRANSFERENCIA DE LÍQUIDOS

Para el diseño del modelo, se llevó a cabo un análisis conceptual de los elementos involucrados en el proceso de transferencia de líquidos, identificando como componentes principales a los recipientes, los instrumentos y el líquido contenido en ambos. Desde estos elementos, se definieron las relaciones y reglas que estructuran el modelo:

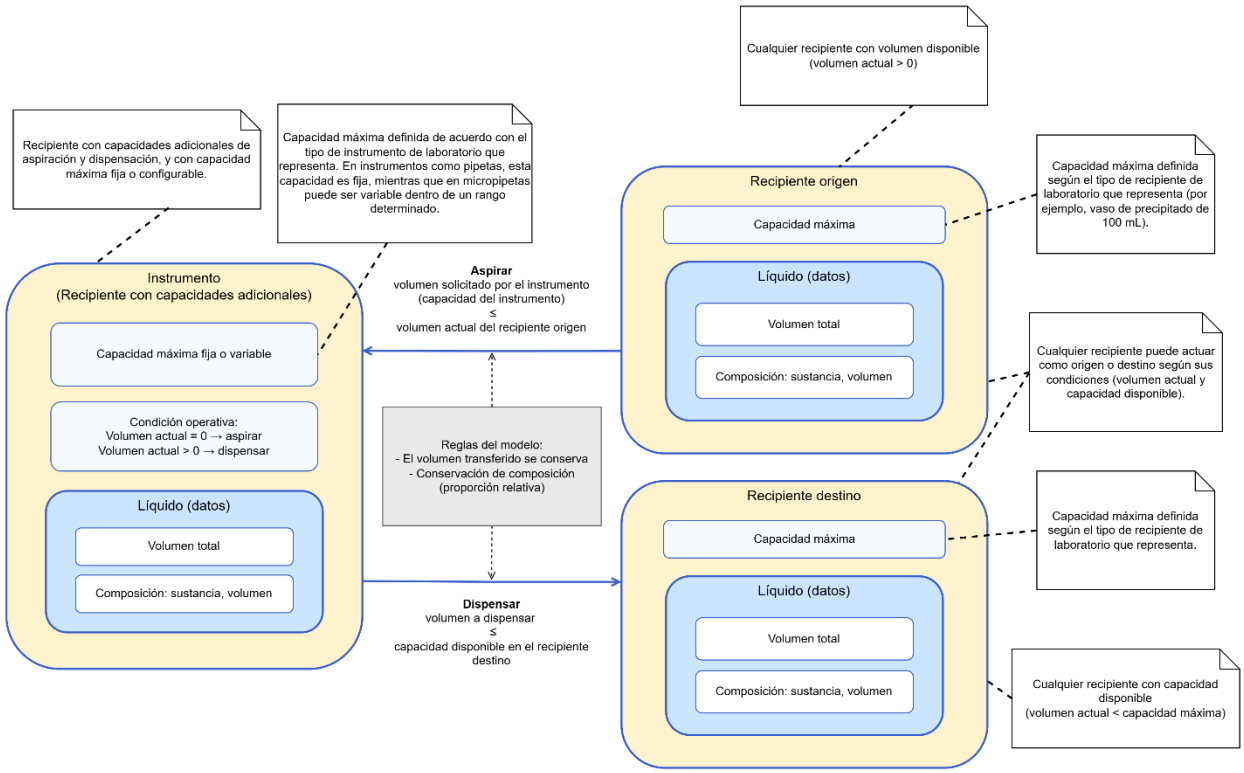
- Recipientes: elementos que actúan como contenedores capaces de almacenar y proporcionar líquido; cada uno tiene parámetros que indican su capacidad máxima y la cantidad actual de líquido, lo que determina el volumen que puede recibir o transferir.
- Instrumentos: elementos que funcionan como intermediarios en la transferencia de líquido entre recipientes, al trasladarlo de un recipiente a otro; en el modelo, se consideran como un tipo de recipiente con capacidades adicionales que permiten la aspiración y dispensación controlada del líquido.
- Líquido: estructura de datos que describe la composición del contenido en términos de volumen total y proporción relativa de sus componentes, la cual es manipulada durante el proceso de transferencia.

La relación lógica entre recipientes, instrumentos y líquido se representa en el diagrama de la Figura 1.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 33 - 48

Figura 1

Modelo lógico de interacción para la transferencia de líquidos

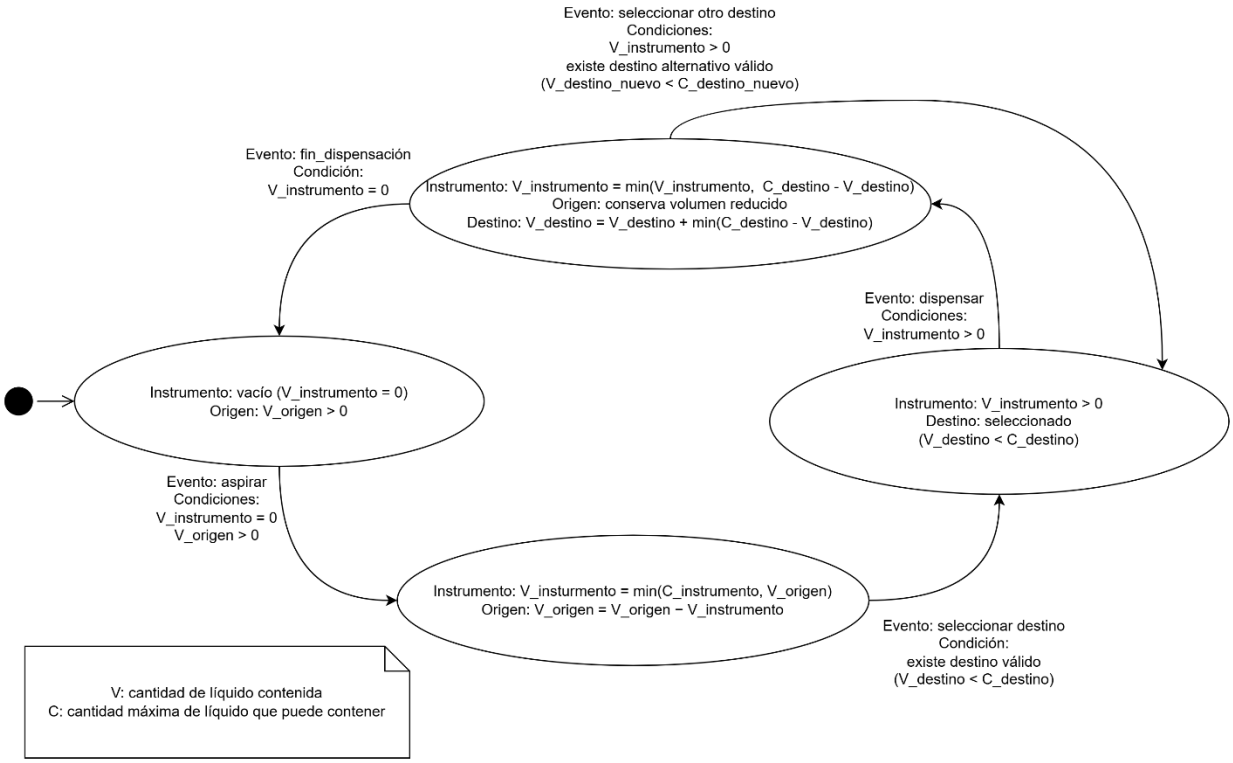


El proceso de transferencia puede resumirse esquemáticamente en la Figura 2.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 34 - 48

Figura 2

Secuencia típica de estados del modelo



Se muestra cómo cambia el estado del sistema a lo largo de una secuencia típica.

2.3 INTEGRACIÓN DEL MODELO EN UNA APLICACIÓN INTERACTIVA TRIDIMENSIONAL

El modelo fue implementado en una aplicación desarrollada en Unity, siguiendo un principio de separación en módulos con el objetivo de facilitar su adaptación a distintas plataformas. Diversos trabajos han señalado que el diseño modular es una estrategia eficaz para gestionar la complejidad de sistemas de *software*, permitiendo que distintos componentes evolucionen de manera independiente (Guntakandla, 2025).

Este diseño distingue tres módulos principales: la lógica del sistema, la representación visual y el mecanismo de interacción. La Figura 3 muestra estos módulos y su relación:

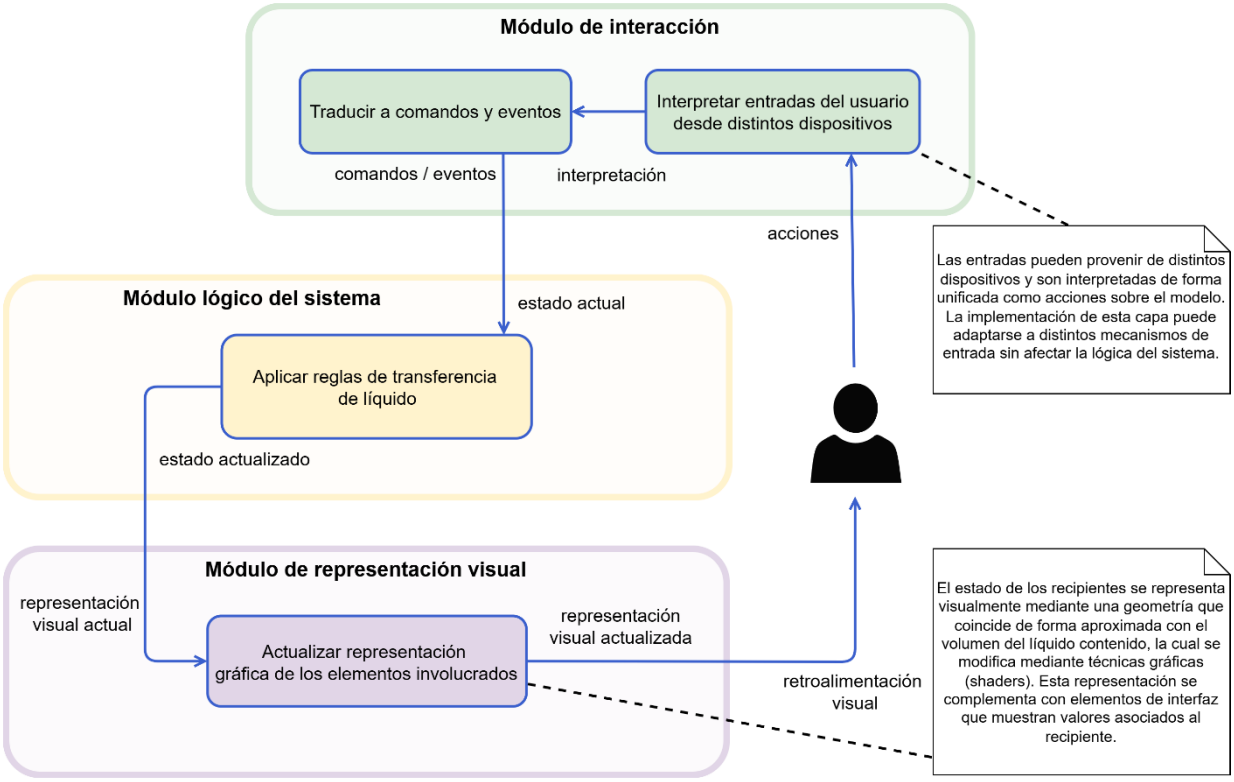
- Módulo lógico del sistema: responsable de gestionar el estado y las reglas que rigen la transferencia de líquidos.
- Módulo de representación visual: encargado de mostrar visualmente el estado actual de los elementos del sistema.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 35 - 48

- Módulo de interacción: interpreta las acciones del usuario y las traduce en operaciones sobre el modelo lógico.

Figura 3

Diseño modular de la aplicación interactiva tridimensional



Las acciones del usuario activan el instrumento virtual por medio del módulo de interacción, el cual se comunica con el módulo lógico para controlar el estado de los recipientes involucrados y el proceso de transferencia de líquido. Finalmente, el módulo lógico se comunica con el módulo de representación visual para actualizar gráficamente el estado de los recipientes.

En el módulo lógico, el comportamiento se define a partir de la representación y manipulación del estado del líquido en cada recipiente, el cual se describe en términos de volumen total y proporción relativa de sus componentes. Esta representación se implementa en los *scripts* LiquidDefinitions.cs y Liquid.cs, mostrados en las Figuras 4 y 5. Los fragmentos de código presentados corresponden a una versión simplificada y funcional de la implementación, utilizada con fines ilustrativos.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 36 - 48

Figura 4

Definición de la estructura de datos para la composición del líquido (LiquidDefinitions.cs)

```
// Archivo: LiquidDefinitions.cs

using System;

// Define la estructura que representa un componente dentro de un líquido
[Serializable]
public struct LiquidData
{
    public LiquidType type;   // Tipo de sustancia
    public float amount;      // Cantidad en mL

    // Constructor del componente
    public LiquidData(LiquidType type, float amount)
    {
        this.type = type;
        this.amount = amount;
    }
}

// Enum que define los tipos posibles de sustancias
public enum LiquidType
{
    Water,
    Alcohol,
    ReagentA,
    ReagentB
}
```

Figura 5

Implementación del recipiente y estado del líquido (Liquid.cs)

```
// Archivo: Liquid.cs

using System.Collections.Generic;
using UnityEngine;

// Representa un recipiente o volumen de líquido
public class Liquid : MonoBehaviour
{
    [Header("Capacidad")]

    [SerializeField] protected float capacity = 10f;     // Capacidad máxima del recipiente
    [SerializeField] protected float currentVolume = 0f; // Volumen actual contenido

    // Getter de la capacidad máxima
    public float GetCapacity() { return capacity; }

    // Getter del volumen actual
    public float GetCurrentVolume() { return currentVolume; }

    // Modifica volumen (uso interno controlado)
    public void ModifyVolume(float delta)
    {
        currentVolume = Mathf.Clamp(currentVolume + delta, 0f, capacity);
    }

    // Setter protegido del volumen (limitado entre 0 y la capacidad)
    protected void SetCurrentVolume(float value)
    {
        currentVolume = Mathf.Clamp(value, 0f, capacity);
    }

    [Header("Composición")]

    [SerializeField] protected List<LiquidData> composition = new List<LiquidData>(); // Componentes del líquido

    // Getter de la composición (uso para visualización o depuración)
    public List<LiquidData> GetComposition() { return composition; }
}
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 37 - 48

La transferencia de líquido entre recipientes se realiza mediante un cálculo proporcional que preserva la composición del líquido durante el proceso. Esta se implementa en el *script* Instrument.cs, mostrado en la Figura 6.

Figura 6

Implementación de la transferencia de líquido mediante el instrumento (Instrument.cs)

```
// Archivo: Instrument.cs

using UnityEngine;

// Instrumento que transfiere líquido entre recipientes
public class Instrument : Liquid
{
    [Header("Transferencia")]

    // Indica si el instrumento permite configurar un volumen de transferencia menor a su capacidad.
    [SerializeField] private bool allowsVariableTransferVolume = true;

    // Volumen configurado para aspirar o dispensar cuando la transferencia variable está habilitada.
    [SerializeField] private float configuredVolume = 50; // Volumen configurado

    // Getter de la bandera que indica si el instrumento permite volumen de transferencia variable.
    public bool AllowsVariableTransferVolume() { return allowsVariableTransferVolume; }

    // Setter de la bandera que habilita o deshabilita el uso de un volumen de transferencia configurable.
    public void SetAllowsVariableTransferVolume(bool value)
    {
        allowsVariableTransferVolume = value;
    }

    // Getter del volumen configurado
    public float GetConfiguredVolume() { return configuredVolume; }

    // Setter del volumen configurado (limitado por el máximo)
    public void SetConfiguredVolume(float value)
    {
        configuredVolume = Mathf.Clamp(value, 0f, capacity);
    }

    // Getter del volumen efectivo de transferencia según la configuración del instrumento.
    public float GetTransferVolume()
    {
        return allowsVariableTransferVolume ? Mathf.Clamp(configuredVolume, 0f, capacity) : capacity;
    }

    // Getter del volumen contenido en el instrumento
    public float GetInstrumentVolume() { return currentVolume; }

    // Setter del volumen interno del instrumento (limitado por su capacidad)
    public void SetInstrumentVolume(float value)
    {
        currentVolume = Mathf.Clamp(value, 0f, capacity);
    }

    // Transfiere volumen entre source y target de forma proporcional
    public void Transfer(float volume, Liquid source, Liquid target)
    {
        if (volume <= 0f || source.GetCurrentVolume() <= 0f) return;

        volume = Mathf.Min(volume, source.GetCurrentVolume());
        volume = Mathf.Min(volume, target.GetCapacity() - target.GetCurrentVolume());

        var sourceComp = source.GetComposition();
        var targetComp = target.GetComposition();

        float sourceVolume = source.GetCurrentVolume();

        for (int i = 0; i < sourceComp.Count; i++)
        {
            var data = sourceComp[i];

            float fraction = data.amount / sourceVolume;
            float amount = volume * fraction;

            // Restar del source
            data.amount -= amount;
            sourceComp[i] = data;

            // Agregar al target
            bool found = false;
            for (int j = 0; j < targetComp.Count; j++)
            {
                if (targetComp[j].type == data.type)
                {
                    var t = targetComp[j];
                    t.amount += amount;
                    targetComp[j] = t;
                    found = true;
                    break;
                }
            }

            if (!found)
                targetComp.Add(new LiquidData(data.type, amount));
        }

        // Actualizar volúmenes usando método controlado
        source.ModifyVolume(-volume);
        target.ModifyVolume(volume);
    }

    // Aspira líquido desde un recipiente al instrumento
    public void Aspirate(Liquid source)
    {
        Transfer(GetTransferVolume(), source, this);
    }

    // Dispensa líquido desde el instrumento a un recipiente
    public void Dispense(Liquid target)
    {
        Transfer(GetTransferVolume(), this, target);
    }
}
```

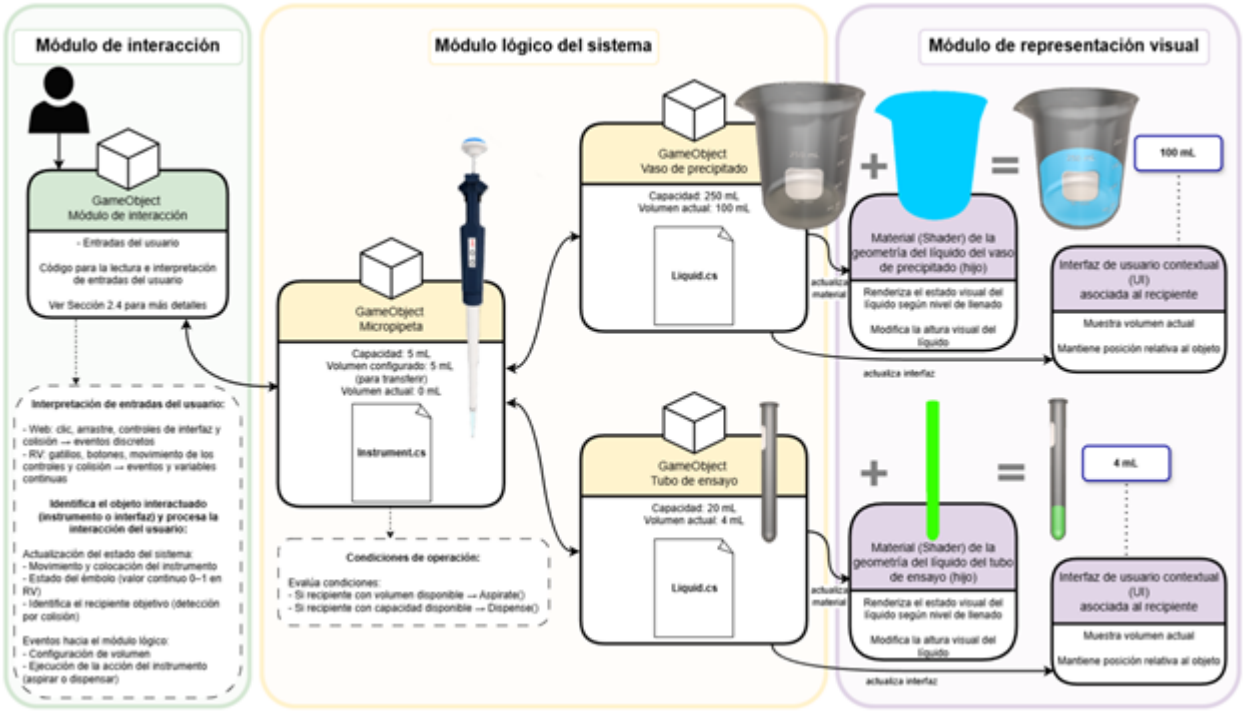
El modelo se implementó sobre objetos virtuales tridimensionales, en los que los instrumentos actúan como recipientes con capacidades adicionales para la transferencia de líquido, mientras que los recipientes actúan como origen o destino según la operación.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 38 - 48

En la aplicación implementada, el instrumento usado es una micropipeta, la cual permite controlar con precisión el volumen transferido en escenarios que requieren múltiples transferencias entre distintos recipientes, con parámetros específicos de volumen inicial y capacidad máxima. La Figura 7 resume la implementación del modelo en Unity.

Figura 7

Implementación del modelo en Unity: relación entre módulos, objetos de la escena y scripts



Nota. Modelos 3D de micropipeta, vaso de precipitado y tubo de ensayo elaborados por Víctor Hugo Franco Serrano.

La representación visual del líquido se realiza mediante un *shader* que ajusta dinámicamente el nivel y color en función de la composición y volumen del líquido en el recipiente (véase Anexo A).

2.4 ADAPTACIÓN DEL MODELO A ENTORNOS WEB Y DE REALIDAD VIRTUAL

El modelo lógico de transferencia de líquidos se diseñó de forma independiente de la plataforma, permitiendo su integración con distintos dispositivos de entrada mediante una abstracción de las interacciones. Esta separación permite mapear diversas formas de interacción física a un mismo conjunto de operaciones lógicas.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 **Vol. 4, Núm. 2.** abril-junio 2026, págs. 39 - 48

En ambas plataformas, los elementos interactivos se representan mediante objetos 3D (instrumentos y recipientes), complementados con interfaces de usuario contextuales que muestran estados como volumen configurado y volumen actual.

2.4.1 WEB

En el entorno web, por su mayor alcance y accesibilidad, la aplicación se orienta al uso de dispositivos de entrada convencionales (ratón o pantalla táctil). En este contexto, las interacciones se basan en la interpretación de eventos de interfaz de usuario (clic, arrastre y controles de interfaz, como botones y controles deslizantes), que se traducen en eventos discretos del sistema.

2.4.2 REALIDAD VIRTUAL

En realidad virtual, se implementaron interacciones mediante los controles físicos del dispositivo (Meta Quest 2, 3 o 3S), de forma que la manipulación del instrumento se asemeje a su uso en la vida real. Este tipo de interacción en realidad virtual, basada en acciones funcionales, permite que el usuario comprenda y anticipe el comportamiento del objeto virtual, favoreciendo una interacción coherente y significativa, aun cuando ésta se realice mediante dispositivos intermediarios como controles de realidad virtual (Fusaro *et al*., 2025).

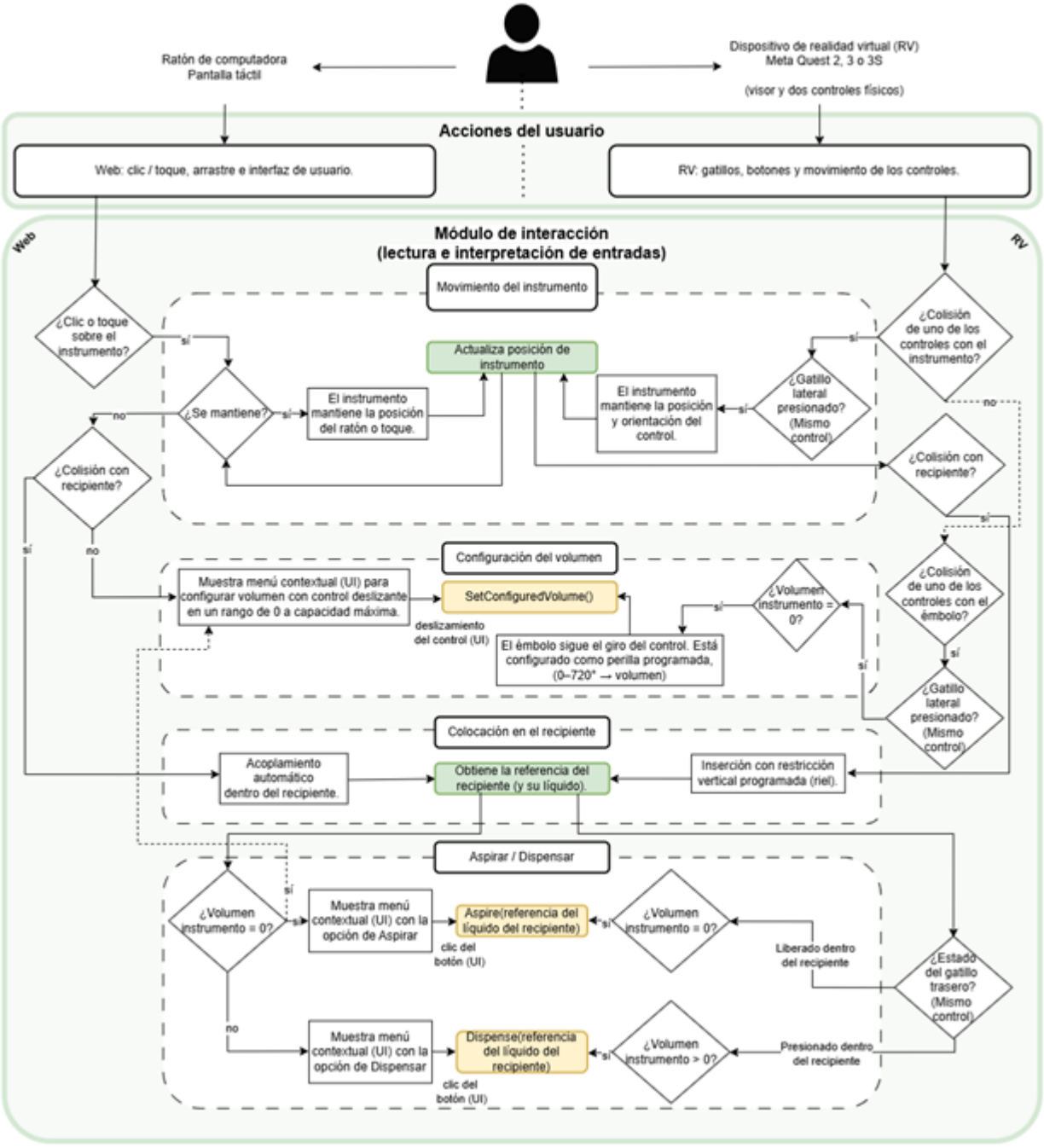
2.4.3 MODELO GENERAL DE INTERACCIÓN

Las acciones del usuario se traducen en llamadas a la misma lógica de actualización del estado del sistema sin importar la plataforma usada, como se muestra en la Figura 8.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 40 - 48

Figura 8

Traducción de interacciones del usuario al modelo lógico de la micropipeta



https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 41 - 48

3. RESULTADOS

El comportamiento del modelo lógico de interacción para la transferencia de líquidos fue evaluado mediante pruebas funcionales basadas en escenarios controlados dentro de la aplicación. Las condiciones de validación consistieron en la ejecución de secuencias de transferencia entre recipientes con diferentes configuraciones de volumen inicial, capacidad máxima y volumen de transferencia, incluyendo transferencias consecutivas. También se evaluaron casos límite como aspiración de volúmenes mayores al disponible, dispensación en recipientes con capacidad insuficiente, aspiración desde recipientes vacíos y dispensación en recipientes llenos.

Los criterios de validación considerados fueron:

- Conservación del volumen total tras cada transferencia.
- Preservación de la proporción de componentes.
- Manejo correcto de casos límite (desbordamiento y volumen insuficiente).
- Consistencia del comportamiento en transferencias consecutivas.
- Consistencia del comportamiento entre plataformas.

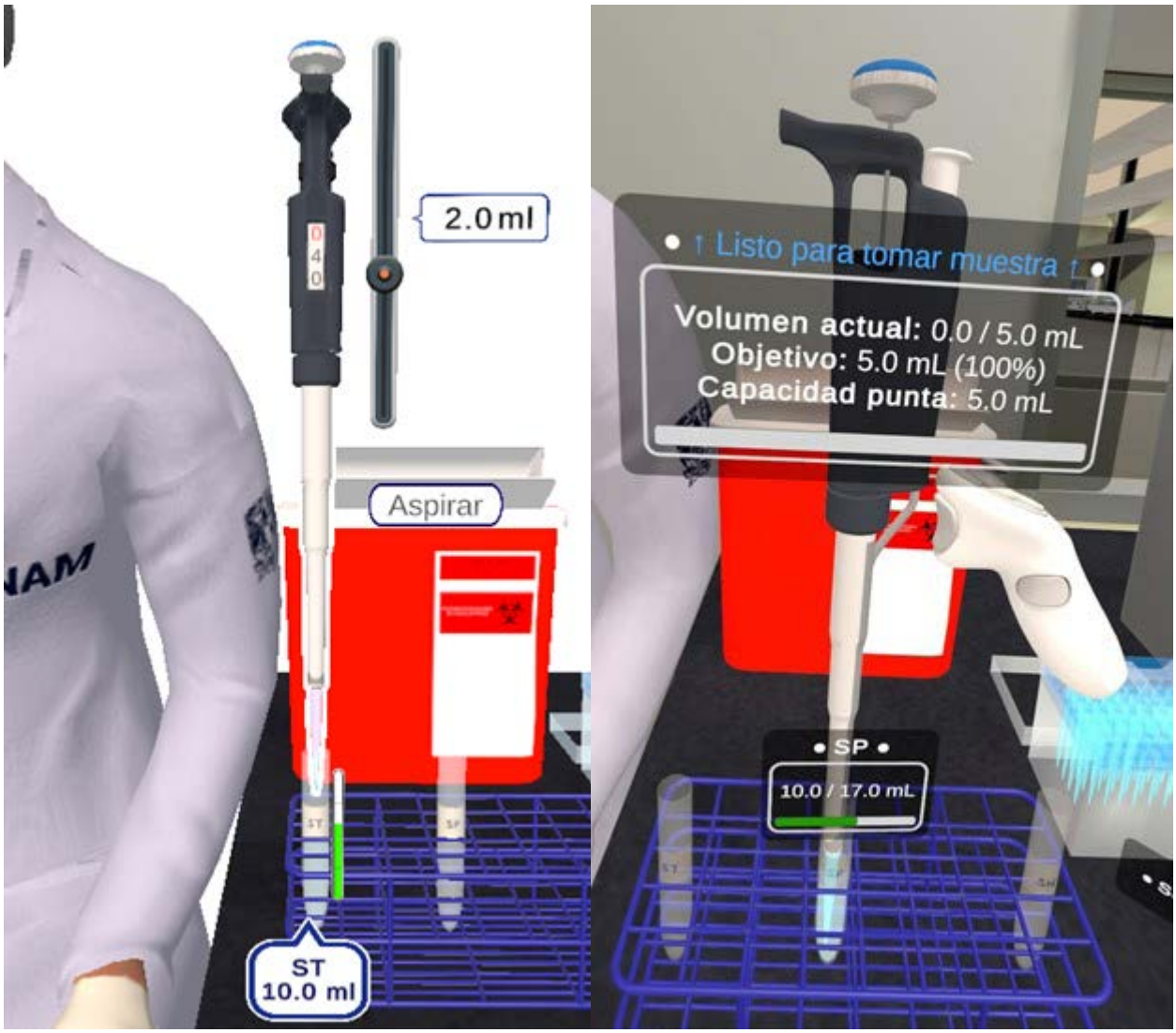
El modelo cumplió con estos criterios en todos los escenarios evaluados, manteniendo un comportamiento consistente.

En cuanto a la adaptación a distintas plataformas, el modelo mantuvo un comportamiento lógico consistente tanto en entornos web como en aplicaciones de realidad virtual. La Figura 9 presenta una comparación visual entre ambas versiones.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 42 - 48

Figura 9

Micropipeta en versiones web y de realidad virtual



Nota. La versión web se muestra a la izquierda y la de realidad virtual a la derecha.

El modelo presenta ciertas limitaciones derivadas de su enfoque abstracto; en particular, sólo soporta mezclas homogéneas, por lo que no considera fenómenos asociados a sistemas heterogéneos como la separación de fases, las reacciones químicas o los cambios derivados de las interacciones entre líquidos.

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 43 - 48

4. CONCLUSIONES

El modelo lógico de interacción presentado en este trabajo, basado en estados y reglas lógicas, permitió regular de forma consistente el volumen y la composición del líquido transferido, sin recurrir a simulaciones físicas de fluidos. El modelo es capaz de realizar los procesos de aspiración y dispensación mediante una micropipeta, preservando la proporción relativa de cada sustancia en la composición del líquido.

La separación del modelo en módulos facilitó la adaptación del modelo a aplicaciones web y de realidad virtual, lo que le da independencia de la plataforma destino.

La decisión de emplear una abstracción lógica basada en datos en lugar de simulación física representa una alternativa viable para la transferencia de líquidos en entornos interactivos, especialmente donde no se requiere simulación física realista de fluidos.

AGRADECIMIENTOS

Se agradece a la Dra. Silvia Maldonado Frías de la Facultad de Odontología de la UNAM por su colaboración para poder desarrollar las prácticas de laboratorio. Se reconoce también la colaboración del Mtro. Víctor Hugo Franco Serrano por el material elaborado y presentado en la Figura 7 de este reporte técnico.

REFERENCIAS

Bencomo, N., Cabot, J., Chechik, M., Cheng, B. H. C., Combemale, B., Wąsowski, A., & Zschaler, S. (2024). Abstraction Engineering. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2408.14074

Cen, Y., Deng, H., Ma, Y., & Liang, X. (2024). A real-time and interactive fluid modeling system for mixed reality. *IEEE Transactions on Visualization and Computer Graphics,* 30(11), 7310–7320. https://doi.org/10.1109/TVCG.2024.3456140

Fusaro, M., Lisi, M. P., Era, V., Porciello, G., Candidi, M., Aglioti, S. M., & Boukarras, S. (2025). The transformative power of virtual reality: redefining interactions in virtual platforms, education, healthcare, and workplaces. *Topoi,* 44(4), 1071–1086. https://doi.org/10.1007/s1124 5-025-10216-1

Guntakandla, A. R. (2025). Modular architecture: A scalable and efficient system design approach for enterprise applications. *World Journal of Advanced Research and Reviews, 26*(1), 3114–3126. https://doi.org/10.30574/wjarr.2025.26.1.1340

Radianti, J., Majchrzak, T. A., Fromm, J., & Wohlgenannt, I. (2020). A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. *Computers & Education, 147,* 103778. https://doi.org/10.1016/j.compedu.2019.103778

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 44 - 48

ANEXO A. REPRESENTACIÓN VISUAL DEL LÍQUIDO MEDIANTE SHADER

A.1 Instrucciones

1. Adjuntar el *script* LiquidShaderController al objeto que contiene la malla del líquido.
2. Asignar un material con el *shader* DRDE/Liquid/SimpleFill a la geometría interior que representa el líquido con el recipiente lleno. La malla debe corresponder únicamente al volumen del líquido. Para una prueba rápida se puede usar un cilindro.
3. Desde el componente responsable de la transferencia, al finalizar la operación, solicitar al recipiente (Liquids.cs) la actualización de los valores de fillLevel (en el rango de 0 y 1) y liquidColor, según el volumen actual del líquido y su composición.

Los fragmentos de código correspondientes al *shader* y al *script* de control se presentan en las Secciones A.5 y A.6, respectivamente.

A.2 Funcionamiento

El *script* LiquidShaderController calcula automáticamente la altura mínima y máxima de la malla del líquido. El *shader* trabaja con un valor de llenado normalizado entre 0 y 1, pero la malla puede tener cualquier altura o escala local. Para ello, el *script* toma los límites reales verticales de la malla y los envía al *shader* para normalizar la altura internamente.

A.3 Consideraciones

- Este ejemplo es ilustrativo. Los fragmentos han sido simplificados con fines de claridad.
- La malla del líquido debe corresponder a la malla interior del recipiente.
- La altura visible no representa necesariamente el volumen real del líquido de forma exacta.
- La relación entre volumen y altura depende de la forma del recipiente y, en caso necesario, debe aproximarse mediante un cálculo adicional.
- Si la malla incluye partes que no pertenecen al volumen del líquido, el *script* no calculará adecuadamente los valores de MinY y MaxY.
- Si MinY y MaxY no describen adecuadamente la geometría útil del líquido, el nivel visual resultante será incorrecto.

A.4 Versiones de implementación

Unity 6000.4.0f1

URP 17.4.0

XR Interaction Toolkit 3.3.1 (para la versión de RV)

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 45 - 48

A.5 Código del *shader* SimpleFill

```
// Archivo SimpleFill.shader

// Shader simple para representar el nivel visible de un liquido dentro de una malla.
Shader "DRDE/Liquid/SimpleFill"
{
    Properties
    {
        // Color uniforme del líquido, incluyendo su transparencia.
        [MainColor] _LiquidColor ("Liquid Color", Color) = (0.2, 0.5, 0.9, 0.75)
        // Nivel de llenado normalizado del 0 al 1.
        _FillLevel ("Fill Level", Range(0, 1)) = 0.5
        // Alturas locales del volumen del líquido, usadas para normalizar la Y.
        _MinY ("Min Y", Float) = 0
        _MaxY ("Max Y", Float) = 1
        // Control explícito de caras visibles sin requerir geometría duplicada.
        [Enum(Caras delanteras, 0, Caras traseras, 1, Ambas, 2)] _RenderFaces ("Render Faces", Float) = 2
        // Superficie superior aproximada en el plano del nivel de llenado.
        [Toggle] _TopSurface ("Top Surface", Float) = 0
    }

    SubShader
    {
        Tags
        {
            "RenderPipeline" = "UniversalPipeline"
            "Queue" = "Transparent"
            "RenderType" = "Transparent"
        }

        HLSLINCLUDE
        #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"

        CBUFFER_START(UnityPerMaterial)
            half4 _LiquidColor;
            float _FillLevel;
            float _MinY;
            float _MaxY;
            float _RenderFaces;
            float _TopSurface;
        CBUFFER_END

        struct Attributes
        {
            float4 positionOS : POSITION;
        };

        struct Varyings
        {
            float4 positionHCS : SV_POSITION;
            float localY : TEXCOORD0;
        };

        Varyings vert(Attributes input)
        {
            Varyings output;
            // Se proyecta la geometría normalmente y se conserva la Y local
            // para decidir luego qué parte del volumen se dibuja.
            output.positionHCS = TransformObjectToHClip(input.positionOS.xyz);
            output.localY = input.positionOS.y;
            return output;
        }

        Varyings vertTopSurface(Attributes input)
        {
            Varyings output;
            float fillY = lerp(_MinY, _MaxY, saturate(_FillLevel));
            float3 topPositionOS = float3(input.positionOS.x, fillY, input.positionOS.z);
            output.positionHCS = TransformObjectToHClip(topPositionOS);
            output.localY = fillY;
            return output;
        }

        half4 fragBase(Varyings input) : SV_Target
        {
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 46 - 48

```
            // Se normaliza la altura local de la malla usando el rango vertical
            // útil del volumen del líquido. Esto evita exigir una malla 0..1.
            float heightRange = max(_MaxY - _MinY, 0.0001);
            float normalizedY = saturate((input.localY - _MinY) / heightRange);
            // Se recorta todo lo que quede por encima del nivel de llenado.
            float cutoff = (_FillLevel * 1.0001) - 0.0001;
            clip(cutoff - normalizedY);
            // El color es uniforme.
            return _LiquidColor;
        }

        half4 fragFront(Varyings input) : SV_Target
        {
            if (_RenderFaces > 0.5 && _RenderFaces < 1.5)
            {
                clip(-1);
            }
            return fragBase(input);
        }

        half4 fragBack(Varyings input) : SV_Target
        {
            if (_RenderFaces < 0.5)
            {
                clip(-1);
            }
            return fragBase(input);
        }

        half4 fragTopSurface(Varyings input) : SV_Target
        {
            if (_TopSurface < 0.5)
            {
                clip(-1);
            }
            return fragBase(input);
        }
        ENDHLSL

        Pass
        {
            Name "BackFaces"
            Tags { "LightMode" = "UniversalForward" }
            // Mezcla transparente simple.
            Blend SrcAlpha OneMinusSrcAlpha
            ZWrite Off
            Cull Front
            HLSLPROGRAM
            #pragma vertex vert
            #pragma fragment fragBack
            ENDHLSL
        }

        Pass
        {
            Name "FrontFaces"
            Tags { "LightMode" = "SRPDefaultUnlit" }
            Blend SrcAlpha OneMinusSrcAlpha
            ZWrite Off
            Cull Back
            HLSLPROGRAM
            #pragma vertex vert
            #pragma fragment fragFront
            ENDHLSL
        }

        Pass
        {
            Name "TopSurface"
            Tags { "LightMode" = "SRPDefaultUnlit" }
            Blend SrcAlpha OneMinusSrcAlpha
            ZWrite On
            ZTest Less
            Cull Off
            HLSLPROGRAM
            #pragma vertex vertTopSurface
            #pragma fragment fragTopSurface
            ENDHLSL
        }
    }
}
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 47 - 48

A.6 Código del script de control del *shader*

```
// Archivo LiquidShaderController.cs

using UnityEngine;

// Controlador simple para el shader del liquido.
[ExecuteAlways]
[RequireComponent(typeof(MeshFilter), typeof(Renderer))]
public class LiquidShaderController : MonoBehaviour
{
    public enum FaceRenderMode
    {
        FrontFaces = 0,
        BackFaces = 1,
        Both = 2
    }

    [Header("Visual")]
    [Range(0f, 1f)]
    public float fillLevel = 0.5f;
    public Color liquidColor = new Color(0.2f, 0.5f, 0.9f, 0.75f);
    public FaceRenderMode renderFaces = FaceRenderMode.Both;
    public bool topSurface = false;

    [Header("Bounds")]
    public bool autoCalculateBounds = true;
    public float manualMinY = 0f;
    public float manualMaxY = 1f;

    [Header("Update")]
    public bool updateContinuously = false;

    static readonly int FillLevelId = Shader.PropertyToID("_FillLevel");
    static readonly int LiquidColorId = Shader.PropertyToID("_LiquidColor");
    static readonly int MinYId = Shader.PropertyToID("_MinY");
    static readonly int MaxYId = Shader.PropertyToID("_MaxY");
    static readonly int RenderFacesId = Shader.PropertyToID("_RenderFaces");
    static readonly int TopSurfaceId = Shader.PropertyToID("_TopSurface");

    Renderer m_renderer;
    MeshFilter m_meshFilter;
    MaterialPropertyBlock m_propertyBlock;

    void OnEnable()
    {
        CacheComponents();
        ApplyToShader();
    }

    void OnValidate()
    {
        CacheComponents();
        ApplyToShader();
    }

    void Update()
    {
        if (!updateContinuously)
        {
            return;
        }
        ApplyToShader();
    }

    // Método público para cambiar el nivel de llenado desde otros scripts.
    public void SetFillLevel(float newFillLevel)
    {
        fillLevel = Mathf.Clamp01(newFillLevel);
        ApplyToShader();
    }
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.2.170 Vol. 4, Núm. 2. abril-junio 2026, págs. 48 - 48

```
    // Método público para cambiar el color del líquido desde otros scripts.
    public void SetLiquidColor(Color newLiquidColor)
    {
        liquidColor = newLiquidColor;
        ApplyToShader();
    }

    // Método público de conveniencia para actualizar nivel y color al mismo tiempo.
    public void SetFillLevelAndColor(float newFillLevel, Color newLiquidColor)
    {
        fillLevel = Mathf.Clamp01(newFillLevel);
        liquidColor = newLiquidColor;
        ApplyToShader();
    }

    // Método público para recalcular los límites de la malla si esta cambia.
    public void RefreshShaderBounds()
    {
        ApplyToShader();
    }

    void CacheComponents()
    {
        if (m_renderer == null)
        {
            m_renderer = GetComponent<Renderer>();
        }
        if (m_meshFilter == null)
        {
            m_meshFilter = GetComponent<MeshFilter>();
        }
        if (m_propertyBlock == null)
        {
            m_propertyBlock = new MaterialPropertyBlock();
        }
    }

    void ApplyToShader()
    {
        if (m_renderer == null || m_meshFilter == null)
        {
            return;
        }
        Mesh mesh = m_meshFilter.sharedMesh;
        if (mesh == null)
        {
            return;
        }
        float minY;
        float maxY;
        if (autoCalculateBounds)
        {
            // Mesh.bounds está en espacio local del mesh.
            // El objeto puede cambiar de escala en la escena.
            Bounds bounds = mesh.bounds;
            minY = bounds.min.y;
            maxY = bounds.max.y;
        }
        else
        {
            minY = manualMinY;
            maxY = manualMaxY;
        }
        m_renderer.GetPropertyBlock(m_propertyBlock);
        m_propertyBlock.SetFloat(FillLevelId, fillLevel);
        m_propertyBlock.SetColor(LiquidColorId, liquidColor);
        m_propertyBlock.SetFloat(MinYId, minY);
        m_propertyBlock.SetFloat(MaxYId, maxY);
        m_propertyBlock.SetFloat(RenderFacesId, (float)renderFaces);
        m_propertyBlock.SetFloat(TopSurfaceId, topSurface ? 1f : 0f);
        m_renderer.SetPropertyBlock(m_propertyBlock);
    }
}
```