

# Entrega Final PF CVI: Ciudad Futurista Inundada con Naves Espaciales

Marco Alejandro Ramírez Camacho

20 de mayo de 2025

## 1. Introducción

En esta cuarta iteración del proyecto final, he implementado dos sistemas de animación en tiempo real que transforman por completo la experiencia visual de la escena. Primero, he desarrollado **un sistema de movimiento dinámico para las naves espaciales** que incluye patrones de vuelo realistas como órbitas y patrullajes. Segundo, he creado **un sistema de olas oceánicas en movimiento** que convierte la superficie de agua estática en un océano vivo con ondas en constante movimiento.

## 2. Sistema de Movimiento Dinámico de Naves Espaciales

### 2.1. Estructura de Datos

He implementado un sistema completo de dinámicas que permite a cada nave tener patrones de movimiento individualizados:

```
1 // creando esta funcion con 2 movimientos disponibles para las
  // naves
2 enum class SpaceshipMovementType {
3     ORBITAL,           // Orbita alrededor de un punto
4     LINEAR_PATROL,    // Patrulla en linea recta
5 };
6
7 struct SpaceshipDynamics {
8     float3 basePosition;          // Posicion base
9     SpaceshipMovementType movementType;
10    float currentTime;           // Tiempo acumulado
11    float scale;                 // Escala de la nave
12
13};
```

## 2.2. Algoritmo de Actualización

La función `UpdateSpaceshipMovement()` se ejecuta cada frame y actualiza las posiciones según patrones matemáticos:

```
1 void Tutorial22_HybridRendering::UpdateSpaceshipMovement(float
2   deltaTime) {
3     for (size_t i = 0; i < m_SpaceshipDynamics.size(); i++) {
4       SpaceshipDynamics& dynamics = m_SpaceshipDynamics[i];
5       dynamics.currentTime += deltaTime * m_SpaceshipSpeed;
6
7       // Alternar entre movimiento orbital y patrullaje
8       bool isOrbital = (i % 2 == 0);
9
10      if (isOrbital) {
11        // Movimiento circular usando trigonometria
12        float phase = dynamics.currentTime * 0.3f;
13        newPosition = basePos + orbit(cos(phase), sin(phase));
14      } else {
15        // Movimiento lineal de patrullaje
16        float patrol = sin(dynamics.currentTime * patrolSpeed);
17        newPosition = basePos + patrolDirection * patrol;
18      }
19
20      // Actualizar matriz de transformacion del objeto
21      m_Scene.Objects[shipIndex].ModelMat = newTransform;
22    }
}
```

## 3. Sistema de Olas Oceánicas en Movimiento

### 3.1. Extensión de Estructuras

Añadí nuevos campos a la estructura de constantes globales para controlar las olas:

```
1 struct GlobalConstants {
2   float Time;           // Tiempo para animacion
3   float WaveStrength; // Intensidad de las olas (0.0-1.0)
4   float WaveSpeed;    // Velocidad de las olas (1.0-5.0)
5 };
```

### 3.2. Malla Densa y Shader de Olas

Transformé el plano de agua de 4 vértices a 1024 vértices (malla 32x32) para permitir deformación suave, con el fin de que se vea más lindo y realista. El vertex shader detecta automáticamente el agua y aplica múltiples ondas sinusoidales (busqué tutoriales faciles como hacer movimientos de olas, y lo más facil era con funciones de seno):

```
1 // En Rasterization.vsh
2 void main(in VSInput VSIn, out PSInput PSIn) {
3   // Transformar a espacio de mundo
```

```

4     PSIn.WPos = mul(float4(VSIn.Pos, 1.0), Obj.ModelMat);
5
6     // Detectar si es agua por posicion Y del objeto
7     bool isWater = (PSIn.WPos.y > -2.0 && PSIn.WPos.y < -1.0);
8
9     if (isWater) {
10         // Combinar multiples ondas sinusoidales
11         float wave1 = sin(x * 2.0 + time * waveSpeed) * 0.3;
12         float wave2 = sin(z * 1.5 + time * waveSpeed * 0.8) * 0.4;
13         float wave3 = sin((x + z) + time * waveSpeed * 1.2) * 0.2;
14
15         // Aplicar deformacion
16         PSIn.WPos.y += (wave1 + wave2 + wave3) * waveStrength;
17
18         // PARTE FUNDAMENTAL
19         // Calcular normales dinamicas para iluminacion
20         PSIn.Norm = calculateWaveNormal(position, time);
21     }
22 }
```

## 4. Integración con Ray Tracing

### 4.1. Sincronización CPU-GPU

Un aspecto crítico es mantener el sistema de ray tracing actualizado con los objetos en movimiento (naves espaciales y las olas):

```

1 void Tutorial22_HybridRendering::Render() {
2     // 1. Actualizar tiempo para animaciones
3     float time = getCurrentTime();
4     constants.Time = time;
5     constants.WaveStrength = m_WaveStrength;
6     constants.WaveSpeed = m_WaveSpeed;
7
8     // 2. MUY IMPORTANTE: Actualizar buffer ANTES del TLAS
9     updateObjectBuffer(m_Scene.Objects);
10
11    // 3. Reconstruir estructura de aceleracion
12    UpdateTLAS();
13
14    // 4. Renderizar con objetos actualizados
15    renderScene();
16 }
```

La secuencia es fundamental: primero actualizar las posiciones en CPU, luego enviar a GPU, y finalmente reconstruir el TLAS para que el ray tracing funcione correctamente.

## 5. Errores en el camino

Intenté primero, implementar la arena que se usaba en el tutorial 23 en la escena que ya tenía creada, pero cuando corría el código con ese approach, la geometría se deformaba de eje, no sé por qué.

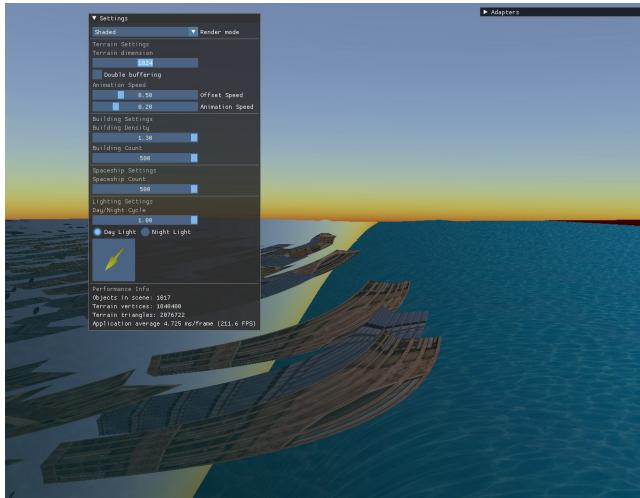


Figura 1: Prueba con approach de arena del tutorial 23

Luego, pensé que había multiplicado mal algun par de matrices y salí de esa forma por ese error, e intenté cambiar la arena para cambiar de eje y estar con los edificios.

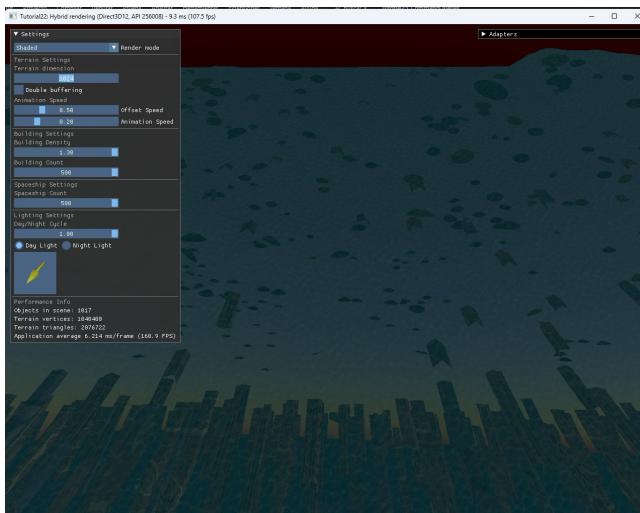


Figura 2: Nuevos errores con diferente approach

Pero al final ese approach no funcionó después de muchas horas e intentos de corregirlo, por eso cambié a cambiar la mesh del agua y darle muchísimos más vértices

## 6. Resultados Visuales

### 6.1. Antes de las Animaciones

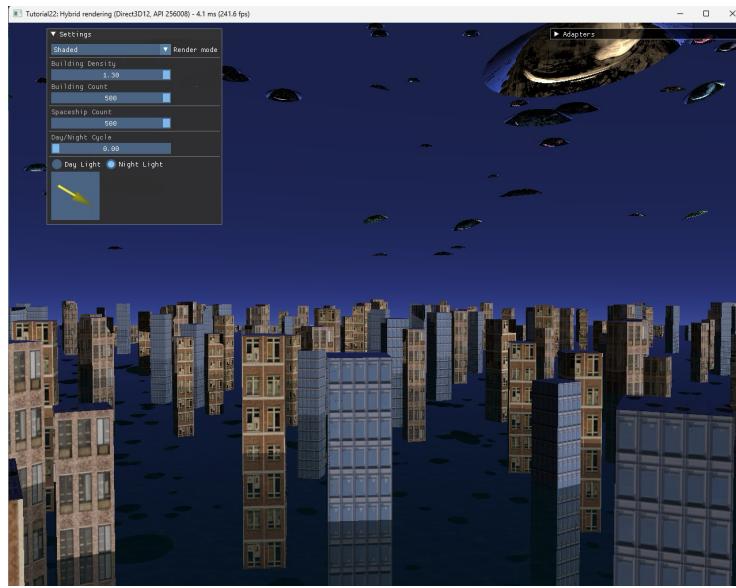


Figura 3: Escena estática antes de implementar las animaciones. Se observa el agua plana y las naves inmóviles.

### 6.2. Escena con Animaciones Completas

Las animaciones transforman completamente la experiencia visual de la escena:

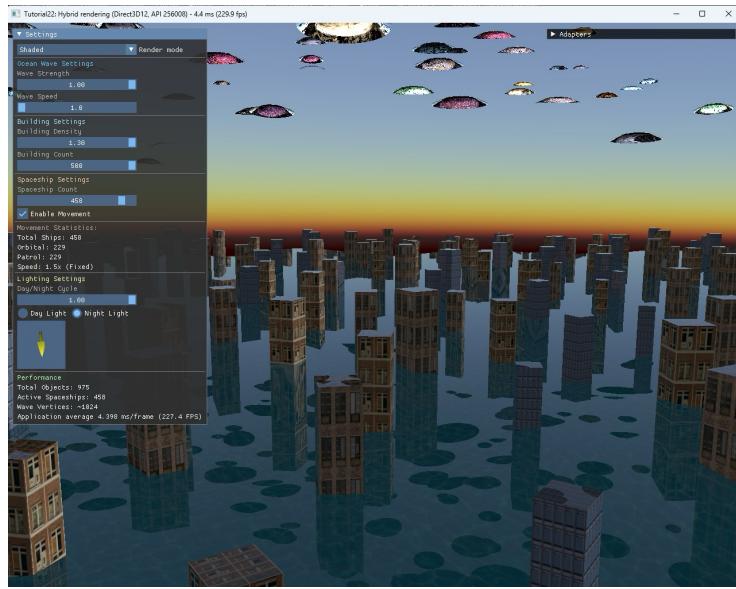


Figura 4: Vista completa de la escena durante el día con todas las animaciones activas: naves espaciales en movimiento orbital y de patrullaje, océano con olas ondulantes, y reflejos dinámicos en la superficie del agua.



Figura 5: La misma escena durante la noche, mostrando cómo las animaciones mantienen su impacto visual en diferentes condiciones de iluminación. Las naves espaciales y las olas son claramente visibles contra el cielo nocturno.

### 6.3. Demostración en Video

Para apreciar completamente el dinamismo de las animaciones implementadas, se puede consultar el siguiente video demostrativo:

#### **Video Demostrativo: Animaciones Dinámicas en Tiempo Real**

El video muestra:

- Patrones de movimiento orbital y de patrullaje de las naves espaciales
- Ondas oceánicas en tiempo real con diferentes intensidades
- Transiciones entre día y noche con animaciones activas
- Interacciones entre ray tracing y objetos en movimiento
- Controles de usuario en tiempo real

### 6.4. Transformación Visual

Las animaciones han transformado completamente la experiencia, **alcanzando lo que se quería lograr en la propuesta de proyecto:**

- **Dinamismo aéreo:** El cielo antes vacío ahora tiene naves espaciales que se mueven, no solamente con un tipo de movimiento.
- **Océano vivo:** La superficie de agua ahora tiene un movimiento calculado con senos, que el usuario puede cambiar con los controles de usuario añadidos (cantidad de oleaje, velocidad del oleaje)
- **Interacciones complejas:** Ray tracing captura sombras y reflejos en movimiento de los objetos dinámicos y estáticos.