Simulación interactiva de pasto en Diligent Engine

David Pérez Cárdenas

28 de mayo de 2025

1. Objetivo y motivación

La motivación del proyecto es realizar un buen trabajo para poder recibir una buena nota y aprovar el curso, sin embargo, hay diferentes objetivos:

- Generar un grid de pasto extenso que simule una larga extensión
- Permitir que una persona a través de un objeto de mueva sobre la extensión del pasto
- El pasto debe reaccionar conforme pasa el jugador, este debe seguir un comportamiento similar a lo que ocurriria si un bloque pasa por encima de el

2. Código de base

El punto de partida fue el código oficial de Diligent Engine *Tutorial 11 – ResourceUpdates* [1]. Dicho proyecto muestra cómo:

- Crear y actualizar buffers dinámicos.
- Subir regiones de textura de forma incremental
- Gestionar estados de recursos y enlaces de ShaderResourceBinding

3. Análisis del código extendido

3.1. Resumen de cambios estructurales

Se añadieron comentarios marcados con hashtag para identificar fácilmente que cambios fueron realizados por mi a diferencia de comentarios ya existentes de Dilligent.

3.2. Extensión de la arquitectura

Jugador: Se añadió el arreglo `SimpleCubeVerts` y su correspondiente índice `SimpleCubeIndices`. Así se cuenta con un mesh ligero para detectar colisiones con el pasto. Así como también `float m_PlayerX = 0.0f;` y `float m_PlayerZ = 0.0f;` para poder guardar la información de su posición.

Se agregó la función `UpdatePlayerVelocity()` que permitió actualizar la velocidad del jugador de manera dinámica para múltiples cálculos, principalmente para el pasto:

Métrica	Valor
Líneas originales (tutorial)	570 (cpp) + 84 (hpp)
Líneas actuales	814 (cpp) + 122 (hpp)
Clases nuevas	Ninguna (se amplió Tutorial11_ResourceUpdates)
Buffers/meshes añadidos	3 (jugador, pasto, plano)
Shaders nuevos	No (se reutilizó `cube.vsh/psh`)

Cuadro 1: Resumen de métricas del proyecto

```
void Tutorial11_ResourceUpdates::UpdatePlayerVelocity(float dt)
{
    float vx = (m_PlayerX - m_PrevPlayerX) / dt;
    float vz = (m_PlayerZ - m_PrevPlayerZ) / dt;

    const float SMOOTH = 10.0f;

    const float threshold = 0.2f;
    if (std::abs(vx) > threshold || std::abs(vz) > threshold)
    {
        m_PlayerVel.x = vx * 0.7f + m_PlayerVel.x * 0.3f;
        m_PlayerVel.z = vz * 0.7f + m_PlayerVel.z * 0.3f;
    }
    else
    {
        m_PlayerVel.x += (vx - m_PlayerVel.x) * (1.f - expf(-SMOOTH * dt));
        m_PlayerVel.z += (vz - m_PlayerVel.z) * (1.f - expf(-SMOOTH * dt));
    }

    if (std::abs(vx) < 0.01f && std::abs(vz) < 0.01f)
    {
        m_PlayerVel.x *= 0.85f;
        m_PlayerVel.z *= 0.85f;
    }
}
```

Figura 1: `UpdatePlayerVelocity()`

Plano de terreno Se definió un quad texturizado que usa la imagen `grass0.png` (esto es basicamente un rectangulo grande). El plano se renderiza primero para aprovechar el *z–buffer* y evitar sobre dibujadar.

Pasto animado El arreglo original de vértices se mejoro para un objeto de pasto mas claro. Cada instancia de pasto se dibuja con `DrawCube` pero recibe su propia matriz de mundo, donde se acumulan rotaciones en x,y,z para simular la distorcion.

3.3. Distorcion del pasto

1. En `Render` se calcula la distancia entre cada pasto y el jugador.
2. Si d es menor que R se aplica un peso $w = (1 - d/R)^2$.
3. El vector de dobles es la suma de dos términos:
   - **Posicion**: empuja el pasto radialmente lejos del jugador.
   - **Velocidad**: añade segun `m_PlayerVel` para un efecto más natural.
4. El ángulo máximo se controla con `MAX_BEND`.

3.4. Movimiento en reposo (“idle”)

Para que el pasto se mueva independiente del jugador se modifico `MapDynamicBuffer`:

- Se duplicó la frecuencia controla la variable `disp`:
  `disp = amp  sin(2.0m_CurrTime + phase);`.

```
// #Generacion del grid de pasto
for (int gz = 0; gz < GRID; ++gz)
{
    for (int gx = 0; gx < GRID; ++gx)
    {
        float xPos = -HALF + gx * STEP;
        float zPos = -HALF + gz * STEP;

        float dx = xPos - m_PlayerX;
        float dz = zPos - m_PlayerZ;
        float d2 = dx * dx + dz * dz;

        float bendX = 0.f, bendZ = 0.f;

        if (d2 < RADIUS * RADIUS)
        {
            float dist = std::sqrt(d2);
            float w    = 1.f - dist / RADIUS;
            w          = w * w;

            float inv = dist > 1e-4f ? 1.f / dist : 0.f;
            float ux  = dx * inv;
            float uz  = dz * inv;

            // #Bend dependiendo de la pos
            float posBendX = uz * MAX_BEND * w * POSITION_INFLUENCE;
            float posBendZ = -ux * MAX_BEND * w * POSITION_INFLUENCE;

            // #Velocidad
            float velBendX = -vel.z * MAX_BEND * w * VELOCITY_INFLUENCE;
            float velBendZ = vel.x * MAX_BEND * w * VELOCITY_INFLUENCE;

            bendX = posBendX + velBendX;
            bendZ = posBendZ + velBendZ;
        }

        float4x4 World = float4x4::RotationX(bendX) *
            float4x4::RotationZ(bendZ) *
            float4x4::Translation(xPos, 0.f, zPos);

        DrawCube(World * ViewProj,
                 m_CubeVertexBuffer[2],
                 m_SRBs[1]);
    }
}
```

Figura 2: Generación de grid de pasto dentro de `Render()`

- Se aumentó `baseAmp` de 0.06 a 0.10.

Esto hace que los tallos se muevan más rápido de lado a lado cuando no se estan moviendo.

3.5. Función `MapDynamicBuffer()`

Esta función es clave para generar efecto del pasto. Esta función:

- Actualiza en cada frame el *vertex buffer* dinámico, evitando reconstruir toda la malla.
- Permite combinar el movimiento natural con las interacciones del jugador (cambio de `m_MovementDirection`).

Al mantener separada la lógica de mapeo y oscilación, el rendimiento se ve menos afectado.

4. Conclusiones y trabajo futuro

El proyecto demuestra cómo, partiendo de un ejemplo de actualización de recursos, es posible implementar interactividad básica utilizando pocos recursos adicionales:

- La malla de pasto reutiliza el mismo *vertex buffer* para todas las instancias
- Las rotaciones se calculan en CPU sin texturas adicionales

```
void Tutorial11_ResourceUpdates::MapDynamicBuffer(Diligent::Uint32 BufferIndex)
{
    MapHelper<Vertex> Vtx(m_pImmediateContext,
                          m_CubeVertexBuffer[BufferIndex],
                          MAP_WRITE,
                          MAP_FLAG_DISCARD);

    for (Uint32 v = 0; v < _countof(CubeVerts); ++v)
    {
        const auto& src = CubeVerts[v];
        Vtx[v].UV       = src.UV;

        if (v < 4)
        {
            Vtx[v].Pos = src.Pos;
            continue;
        }

        float phase        = 0.30f * v;
        float baseAmp      = 0.06f;
        float heightFactor = src.Pos.y / 3.0f;
        float amp          = baseAmp * (heightFactor * 1.5f + 0.1f);

        if (v >= 4 && v <= 11) amp *= 1.3f;

        float disp = (amp * kIdleA) *
            std::sin(m_CurrTime * 0.8f * kIdleF + phase);

        float tilt  = src.Pos.y * 0.02f;
        float swayX = 0, swayY = 0, swayZ = 0;

        if (m_MovementDirection == 0)
        {
            swayX = disp;
            swayY = -disp * tilt;
        }
        else
        {
            swayZ = disp;
            swayY = -disp * tilt;
        }

        Vtx[v].Pos = float3{src.Pos.x + swayX,
                            src.Pos.y + swayY,
                            src.Pos.z + swayZ};
    }
}
```

Figura 3: Función `MapDynamicBuffer()`

Sin embargo hay bastante espacio para la mejora, se podria programar que el pasto se deforme de manera mas natural, lo que ocurre en el proyecto es que a veces se puede ver bastante extraño, además, hay un pequeño error en cuanto el jugador se aleja más, la distorción del pasto lo empieza a seguir en lugar de estar con el, no es muy notorio pero ocurre.

Referencias

[1] Diligent Graphics. "Tutorial 11 – Resource Updates". Disponible en: https://diligentgraphics.com/diligent-engine/guide/tutorials/tutorial11 *(consultado: agosto 2025)*.

[2] Peercy, M. *et al.* "Vegetation Rendering with Shells and Fin Textures". *GPU Gems* vol.2̃, cap.7̃, NVIDIA, 2005. Disponible en: https://developer.nvidia.com/gpugems/gpugems2/part-i-geometric-complexity/chapter-7-vegetation-rendering-shells-and-fin.

[3] Rodríguez, J. "3D Grass Shader with Distortion Wind". gameidea.org, 2023. Disponible en: https://gameidea.org/2023/12/03/3d-grass-shader-with-distortion-wind/.

[4] Foro Unreal Engine. "Grass spread away from player material?". 2016. Disponible en: [https://forums.

unrealengine.com/t/grass-spread-away-from-player-material/94298](https://forums.unrealengine.com/t/grass-spread-away-from-player-material/94298).