

# Arbre

Ce guide explique comment **reproduire ce projet depuis zéro** dans WebGPU Studio (sans charger un exemple).

## 1) Objectif et principe

On va créer les buffers, coller les fonctions WGSL, écrire les compute shaders, puis configurer la Pass.

Étapes (dans l'ordre) :

- **Pipeline 1**

## 2) Créer le projet

1. Lance WebGPU Studio.
2. Clique **Nouveau**.

## 3) Créer les buffers (onglet Buffers)

Crée les buffers suivants (noms **exactement** identiques) :

- **texture1** : taille **128×128×128**, type **uint**, remplissage **random**

À chaque création/modification : clique **Appliquer**.

## 4) Ajouter la bibliothèque de fonctions (onglet Fonctions)

Pour chaque entrée ci-dessous :

1. Colle le code WGSL.

### Bibliothèque 1

```
const SX : u32 = 128u;
const SY : u32 = 128u;
const SZ : u32 = 128u;
fn clamp01(x: f32) -> f32 {
    return clamp(x, 0.0, 1.0);
}
fn packRGB(r: f32, g: f32, b: f32) -> u32 {
    let R: u32 = u32(clamp01(r) * 255.0);
    let G: u32 = u32(clamp01(g) * 255.0);
```

```

    let B: u32 = u32(clamp01(b) * 255.0);
    return (0xFFu << 24u) | (R << 16u) | (G << 8u) | B;
}
// Distance point → segment
fn distPointSegment(p: vec3<f32>, a: vec3<f32>, b: vec3<f32>) -> f32 {
    let ab = b - a;
    let t = clamp(dot(p - a, ab) / max(dot(ab, ab), 1e-6), 0.0, 1.0);
    let q = a + t * ab;
    return length(p - q);
}
struct Node {
    pos: vec3<f32>,
    dir: vec3<f32>,
    len: f32,
    rad: f32,
    depth : i32,
};

```

## 5) Créer les compute shaders (onglet Compute Shaders)

Pour chaque shader :

1. Colle le code WGSL.

### Shader Compute1

Workgroup: 8×8×1

```

@compute @workgroup_size(4, 4, 4)
fn Compute1(@builtin(global_invocation_id) gid : vec3<u32>) {
    let index = gid.z * SX * SY + gid.y * SX + gid.x;
    // Voxel → espace centré
    let p = vec3<f32>(
        (f32(gid.x) / f32(SX)) * 2.0 - 1.0,
        (f32(gid.y) / f32(SY)) * 2.0 - 1.0,
        (f32(gid.z) / f32(SZ)) * 2.0 - 1.0
    );
    // Arbre
    let rootPos = vec3<f32>(0.0, -1.0, 0.0);
    let rootDir = vec3<f32>(0.0, 1.0, 0.0);
    let maxDepth = 6;
    var stack : array<Node, 64>;
    var sp = 0;
    stack[0] = Node(rootPos, rootDir, 0.45, 0.09, maxDepth);
    sp = 1;
}

```

```

var minD = 1e9;
var leaf = 0.0;
loop {
    if (sp <= 0) { break; }
    sp -= 1;
    let n = stack[sp];
    let a = n.pos;
    let b = n.pos + n.dir * n.len;
    minD = min(minD, distPointSegment(p, a, b) - n.rad);
    if (n.depth <= 1) {
        leaf = max(leaf, exp(-length(p - b) * 25.0));
    }
    if (n.depth == 0) { continue; }
    let bend = 0.55;
    let d1 = normalize(n.dir + vec3<f32>( bend, 0.6, 0.0));
    let d2 = normalize(n.dir + vec3<f32>(-bend, 0.6, 0.0));
    if (sp + 2 < 64) {
        stack[sp] = Node(b, d1, n.len * 0.72, n.rad * 0.65, n.depth - 1);
        sp += 1;
        stack[sp] = Node(b, d2, n.len * 0.72, n.rad * 0.65, n.depth - 1);
        sp += 1;
    }
}
// Masque solide
if (minD > 0.0 && leaf < 0.02) {
    texture1[index] = 0x01FF0033u; // fond noir opaque
    return;
}
let trunk = vec3<f32>(0.45, 0.28, 0.14);
let leaves = vec3<f32>(0.20, 0.85, 0.35);
let color = clamp(trunk + leaves * leaf, vec3<f32>(0.0),
vec3<f32>(1.0));
texture1[index] = packRGB(color.r, color.g, color.b);
}

```

## 6) Configurer la Pass (onglet Pass)

Crée les pipelines/étapes dans l'ordre suivant :

- **Pipeline 1** : dispatch 32×32×32

## 7) Compiler et exécuter

1. Dans l'onglet **Buffers**, sélectionne **texture1**.
2. Visualise en 2D ou 3D.

3. Clique **Compile**.
4. Clique **Run** (ou avance avec **Step**).

## 8) Vérifications rapides (si ça ne marche pas)

- Onglet **Console** : lis les erreurs WGSL.
- Vérifie les **noms** des buffers (ils doivent correspondre au code WGSL).
- Vérifie les **tailles** (X/Y/Z) et le **dispatch** dans la Pass.

## 9) Sauvegarder

Clique **Sauver** pour exporter le projet en **.wgstudio**.