

# Plasma (Tsoding)

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 **256×256×1**, 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 SXf = 256.0 ;  
const SYf = 256.0 ;  
const PI = 3.14159 ;
```

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

Pour chaque shader :

1. Clique **+Ajouter**.

2. Donne le nom.
3. Clique [Appliquer](#).
4. Colle le code WGSL.

## Shader [Compute2](#)

Workgroup: [8×8×1](#)

```
@compute @workgroup_size(8, 8, 1)
fn Compute2(@builtin(global_invocation_id) gid : vec3<u32>) {
    let index = gid.y * u32(SXf) + gid.x;
    let x = gid.x;
    let y = gid.y;
    let w = SXf ;
    let h = SYf ;
    let r = vec2<f32>(w, h) ;
    let t = (f32( step ) / 240.0) * 2.0 * PI;
    let FC = vec2<f32>(f32(x), f32(y));
    let p = (FC * 2.0 - r) / r.y;
    var l = vec2<f32>(0.0, 0.0);
    var i = vec2<f32>(0.0, 0.0);
    let s = 4.0 - 4.0 * abs(0.7 - dot(p, p));
    l = l + vec2<f32>(s, s);
    var v = p * l;
    // o accum
    var o = vec4<f32>(0.0, 0.0, 0.0, 0.0);
    for (var k : i32 = 1; k <= 8; k = k + 1) {
        let iy = f32(k);
        i.y = iy;
        let s4 = sin(vec4<f32>(v.x, v.y, v.y, v.x)) + vec4<f32>(1.0);
        o = o + s4 * abs(v.x - v.y);
        let c2 = cos(vec2<f32>(v.y, v.x) * iy + i + vec2<f32>(t, t)) /
iy + vec2<f32>(0.7, 0.7);
        v = v + c2;
    }
    let py4 = p.y * vec4<f32>(-1.0, 1.0, 2.0, 0.0);
    let num = 5.0 * exp(vec4<f32>(1.x - 4.0) - py4);
    let eps = vec4<f32>(1e-6);
    o = tanh(num / max(abs(o), eps));
    let R = u32(o.x * 255.0) ;
    let G = u32(o.y * 255.0) ;
    let B = u32(o.z * 255.0) ;
    texture1[index] = 0xFF000000u + ( R << 16) + (G << 8) + B ;
}
```

## 6) Configurer la Pass (onglet Pass)

Crée les pipelines/étapes dans l'ordre suivant :

- **Pipeline 1** : dispatch **32×32×1**

## 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**.