Plasma 2

This guide explains how to **recreate this project from scratch** in WebGPU Studio (without loading an example).

1) Goal and principle

We will create the buffers, paste the WGSL helper functions, write the compute shaders, then configure the Pass.

Steps (in order):

- **Pipeline 1**

2) Create a new project

1. Launch WebGPU Studio.
2. Click `New`.

3) Create the buffers (Buffers tab)

Create the following buffers (names must match exactly):

- **`texture1`**: size `128×64×128`, type `uint`, fill `random`

After each change, click `Apply`.

4) Add the helper library (Functions tab)

For each entry below:

1. Paste the WGSL.

Bibliothèque 1

```
const SX = 128 ;
const SY = 64  ;
const SZ = 128 ;
fn clamp01(x: f32) -> f32 {
  return clamp(x, 0.0, 1.0);
}
fn packAARRGGBB(a: f32, r: f32, g: f32, b: f32) -> u32 {
  let A: u32 = u32(clamp01(a) * 255.0 + 0.5);
  let R: u32 = u32(clamp01(r) * 255.0 + 0.5);
```

```
  let G: u32 = u32(clamp01(g) * 255.0 + 0.5);
  let B: u32 = u32(clamp01(b) * 255.0 + 0.5);
  return (A << 24u) | (R << 16u) | (G << 8u) | (B);
}
// Smooth plasma palette (polynomial-ish, no branches)
fn palette(t: f32) -> vec3<f32> {
  // t in [0,1]
  let tt = clamp01(t);
  let r = 0.55 + 0.45 * sin(6.2831853 * (tt + 0.00));
  let g = 0.55 + 0.45 * sin(6.2831853 * (tt + 0.33));
  let b = 0.55 + 0.45 * sin(6.2831853 * (tt + 0.67));
  return vec3<f32>(r, g, b);
}

// A cheap "fbm-like" sum of sines in 3D
fn plasmaField(p: vec3<f32>, t: f32) -> f32 {
  // p in roughly [-1,1]
  let p1 = p * 3.1 + vec3<f32>(0.0, 0.0, t * 0.35);
  let p2 = p * 5.3 + vec3<f32>(t * 0.25, 1.7, -t * 0.20);
  let p3 = p * 9.7 + vec3<f32>(-t * 0.18, t * 0.14, 2.3);
  let s1 = sin(p1.x) + sin(p1.y) + sin(p1.z);
  let s2 = sin(p2.x + sin(p2.y)) + sin(p2.y + sin(p2.z)) + sin(p2.z +
sin(p2.x));
  let s3 = sin(p3.x + p3.y) + sin(p3.y + p3.z) + sin(p3.z + p3.x);
  // radial modulation for a more "3D volumetric" look
  let r = length(p);
  let sr = sin(10.0 * r - t * 1.2);
  // combine and normalize to ~[-1,1]
  let v = 0.45 * (s1 / 3.0) + 0.35 * (s2 / 3.0) + 0.20 * (s3 / 3.0);
  return clamp(v + 0.25 * sr, -1.0, 1.0);
}
```

5) Create the compute shaders (Compute Shaders tab)

For each shader:

1. Paste the WGSL.

Shader `Compute1`

Workgroup: `8×8×1`

```
@compute @workgroup_size(4, 4, 4)
fn Compute1(@builtin(global_invocation_id) gid : vec3<u32>) {
  let index : u32 = gid.z * SX * SY + gid.y * SX + gid.x;
```

```
  if ( gid.x < SX/2 && gid.z < SZ/2 ) {
    texture1[index]  = 0x02AABBCCu ;
    return;
  }
  // Normalized voxel coords centered at 0, scaled to [-1,1] with aspect
// correction
  let fx = (f32(gid.x) + 0.5) / f32(SX);
  let fy = (f32(gid.y) + 0.5) / f32(SY);
  let fz = (f32(gid.z) + 0.5) / f32(SZ);
  var p = vec3<f32>(fx * 2.0 - 1.0, fy * 2.0 - 1.0, fz * 2.0 - 1.0);
  // aspect: keep spheres round if sizes differ
  p = p * vec3<f32>(f32(SX) / f32(SY), 1.0, f32(SZ) / f32(SY));
  let t = f32(step) * 0.035;
  // Field in [-1,1]
  let v = plasmaField(p, t);
  // Map to [0,1] for color ramp, with extra contrast
  let m = clamp01(0.5 + 0.5 * v);
  let mc = smoothstep(0.10, 0.95, m);
  // Base color from palette
  let col = palette(mc);
  // Volumetric-ish alpha:
  // - strongest near brighter plasma
  // - fade towards edges of the volume (gives a soft "fog in a box"
//feel)
  let edgeFade = clamp01(1.0 - pow(length(p) / 1.35, 2.0));
  let alpha = clamp01(0.10 + 0.90 * pow(mc, 1.6) * edgeFade);
  // Subtle glow: boost brighter parts
  let glow = 0.20 * pow(mc, 3.0);
  let rgb = clamp(col + vec3<f32>(glow), vec3<f32>(0.0),
vec3<f32>(1.0));
  texture1[index] = packAARRGGBB(alpha, rgb.r, rgb.g, rgb.b);
}
```

6) Configure the Pass (Pass tab)

Create the pipelines/steps in the following order:

- **Pipeline 1**: dispatch `32×16×32`

7) Compile and run

1. In the `Buffers` tab, select **`texture1`**.
2. View it in `2D` or `3D`.
3. Click `Compile`.
4. Click `Run` (or use `Step`).

8) Quick checks (if it doesn’t work)

- `Console` tab: read WGSL errors.
- Check buffer **names** match the WGSL code.
- Check buffer sizes (X/Y/Z) and Pass dispatch.

9) Save

Click `Save` to export the project as a `.wgstudio` file.