# Energy in Chaos: GPU-Accelerated Lagrangian Particle Advection

A Real-Time Interactive Visualization

[Your Name/Entity]

`Tech Stack:` WebGL2, Three.js, GLSL



November 19, 2025

# 1 Abstract

"Energy in Chaos" is a real-time, interactive visualization of turbulent flow fields running entirely on the GPU. By leveraging GPGPU (General-Purpose computing on Graphics Processing Units) techniques, the system simulates up to $10^6$ particles at 60 FPS in a standard web browser.

The simulation employs Curl Noise potentials to generate divergence-free velocity fields, creating fluid-like motion without the computational cost of solving full Navier-Stokes equations. Additionally, it features a custom procedural audio engine and a dynamic proximity-based line topology system to visualize electrical connectivity.

# 2 Theoretical Framework

## 2.1 Divergence-Free Velocity Fields (Curl Noise)

The core aesthetic mimics the behavior of an incompressible fluid. In fluid dynamics, incompressible flow is characterized by a zero-divergence velocity field:

$$\nabla \cdot \vec{v} = 0$$

Standard Simplex noise is not divergence-free; using it directly results in particles collapsing into sinks. To solve this, we utilize the Helmholtz-Hodge decomposition. We assign noise not to the velocity $\vec{v}$, but to a vector potential field $\vec{\psi}(x, y, z)$. We then derive velocity as the curl of that potential:

$$\vec{v} = \nabla \times \vec{\psi}$$

By vector calculus identity, the divergence of a curl is always zero ($\nabla \cdot (\nabla \times \vec{\psi}) = 0$). This mathematically guarantees that particles flow around invisible obstacles and form closed vortices without changing fluid density.

## 2.2 Lagrangian Integration

The system treats the simulation as a Lagrangian particle system (tracking individual points) rather than an Eulerian system (tracking flow at grid cells). The position $P$ of a particle at time $t$ is updated using semi-implicit Euler integration:

$$P_{t+\Delta t} = P_t + \vec{v}(P_t, t) \cdot \Delta t$$

Where $\vec{v}$ is the summation of the Curl Noise field, the "Bombing" radial force triggered by audio transients, and the "8D" rotational torque.

# 3 Implementation Details

The implementation relies on "Ping-Pong" Buffers to bypass WebGL's limitation of reading and writing to the same texture simultaneously.

## 3.1 GPGPU State Management

State data is stored in Floating Point Textures (`RGBA32F`).

- **Resolution:** $N \times N$ (e.g., $1024 \times 1024$).
- **Capacity:** $N^2$ (approx. 1,000,000 particles).

Table 1: GPGPU Texture Data Layout

| Texture | Data Stored |
|---|---|
| Texture A (Velocity) | `vec3(vx, vy, vz)` and `float(decay)` |
| Texture B (Position) | `vec3(x, y, z)` and `float(life)` |

### 3.2 Compute Shaders (GLSL)

The simulation logic runs in fragment shaders:

- **Velocity Kernel:**
  - Samples 3D Simplex Noise.
  - Computes analytical derivatives for the Curl operation.
  - Injects audio-reactive forces (`explosiveForce` and `swirlForce`).
- **Position Kernel:**
  - Advects particles based on velocity.
  - Manages lifecycle: When `life < 0`, the particle is respawned in a dense "heart" cluster at the center to restart the cycle.

### 3.3 Dynamic Topology (Electrical Lines)

To visualize energy transfer, the system renders a secondary layer of `GL_LINES`.

- **Vertex Shader Logic:** Each line segment fetches the positions of two random particles from the GPGPU state texture.
- **Distance Culling:** The shader calculates the Euclidean distance between the two particles. If the distance exceeds a threshold ($d > 15.0$), the line's alpha is collapsed to zero. This creates a flickering, organic "arcing" effect that responds to the turbulence of the particle field.

### 3.4 Procedural Audio Synthesis

Instead of relying on microphone input, the application includes a custom `AudioContext` graph:

- **Oscillator 1 (Sawtooth):** Passed through a Low-Pass Filter modulated by an LFO to create a sci-fi drone.
- **Oscillator 2 (Sine):** Modulated by an exponential gain envelope to simulate a "Heartbeat" kick drum at 60 BPM.

The spectral data from this graph drives the visual "Bombing" effects via a uniform `audioLevel`.

## 4 Rendering Pipeline

To render 1 million particles efficiently, CPU draw calls are minimized using hardware instancing.

- **Instanced Billboards:**
  - Geometry is a single plane.

- The vertex shader stretches the plane along the velocity vector to simulate motion blur ("sparks").

- **High Dynamic Range (HDR):**
  - Colors are calculated in linear space with values exceeding 1.0.

- **Post-Processing:**
  - An `UnrealBloomPass` is applied to bleed high-intensity pixels into neighboring regions, creating the "neon" glow aesthetic.

# 5 Performance & Scalability

The application includes a dynamic configuration system (`SETTINGS.preset`) that reallocates GPU buffers on the fly:

Table 2: Performance Presets and Hardware Targets

| **Preset** | **Texture Size** | **Particle Count** | **Target Hardware** |
|---|---|---|---|
| Low | 256x256 | 65,536 | Mobile / Integrated Graphics |
| Medium | 512x512 | 262,144 | Entry-level Dedicated GPU |
| High | 700x700 | 490,000 | Mid-range Desktop |
| Ultra | 1024x1024 | 1,048,576 | High-end GPU (RTX 3060+) |