The Algorithmic Umwelt

Tech Stack: HTML5, CSS3, Three.js (WebGL), ml5.js (Computer Vision), Gemini 2.5 Flash API (LLM/VLM), WebXR.

Creator:: *Weijun Wang*

The Algorithmic Umwelt (internally designated as AI_VIEW: ENHANCED_VISUAL_STREAM) is an interactive, web-based spatial installation and machine vision simulator. Inspired by Jakob von Uexküll's concept of the *Umwelt* (the self-centered world of a subjective organism) and Timothy Morton's *Hyperobjects*, this project attempts to construct and visualize the perceptual reality of an autonomous, non-human intelligence.

Instead of human-centric RGB imagery, the system perceives reality as a volumetric data void—a persistent matrix of discrete coordinates where "life" is reduced to spatial perturbations, entropy, and mathematical anomalies.

2. Conceptual Framework

This system challenges fundamental assumptions about human agency and machine intelligence:

- **Asymmetrical Interaction: The machine does not adapt to the human; the human must adapt to the machine's strict parameters of depth and motion.**
- **Redefining "Life": To the system, biology is merely high-frequency noise. Life is calculated as frame-by-frame voxel disparity (motion).**
- **Inverse Thermal Logic: Defying human intuition, the distant background (unknown void) is rendered in warm, chaotic amber/red (raw entropy),**

while proximate objects (scanned and parsed humans) are rendered in cold, electric blue (calculated data).

3. Technical Architecture

The project is built on a high-performance, real-time web stack:

- **Rendering Engine: Three.js (WebGL). Replaces traditional 2D mesh stretching with a true 3D volumetric point cloud utilizing custom ShaderMaterial and dynamic particle size attenuation.**
- **Neural Network (Local): ml5.js (FaceMesh). Executes real-time biological tracking, overlaying a coordinate mesh onto detected facial structures.**
- **Cognitive Processor (Cloud): Google Gemini 2.5 Flash API. Acts as the "analytical brain" of the system, interpreting raw optical feeds into cold, tactical text reports.**
- **Spatial Tracking: WebXR API. Supports SBS (Side-by-Side) 3D stereoscopic rendering for immersive VR headsets.**

4. Core Modules & Mechanics

4.1 Volumetric Abyss (Exponential Depth Mapping)

To eliminate the flat "bas-relief" effect typical of basic depth-maps, the system utilizes a persistent 260×180 spatial grid. Depth (Z-axis) is calculated using an exponential multiplier based on pixel luminosity:

$Z = \text{brightness}^{2.2} \times \text{Depth\_Exaggeration}$

This creates a profound parallax effect—distant backgrounds collapse into a dark void, while bright foreground subjects violently project toward the observer.

4.2 Biological Threat Detection (The Hostility Index)

The system calculates a real-time HOSTILITY metric (ranging from 0.00 to

1.00) based on motion detection (pixel disparity between frames).

- **Passive State: The point cloud remains stable, rendering in its designated data spectrum.**
- **Hostile State: Excessive biological movement triggers system-wide overload. Affected voxels experience violent Z-axis jitter (a "boiling" effect), colors shift to glaring red, and the UnrealBloomPass intensifies, simulating sensory overload.**

5. User Interface (HUD)

The interface is designed with a dystopian, industrial sci-fi aesthetic (Glassmorphism, custom CSS scanlines, chamfered borders).

- **Render Modes:**
  - AI_EYE DEFAULT: Deep blue with dynamic red hostility shifts.
  - DEPTH MAP: Turbo spectrum radar mapping.
  - THERMAL HEATMAP: Classic infrared aesthetic.
  - WHITEHOT: Raw greyscale intensity.
- **Controls:** Users can calibrate the NOISE GATE (to filter out background static) and Z-AXIS EXAGGERATION (to manipulate the physical thickness of the point cloud).

6. Deployment & Setup

1. **Requirements:** A modern web browser (Chrome/Edge recommended), a webcam, and an active internet connection.
2. **API Configuration:** Locate the const apiKey = ""; variable in the core script and insert a valid Google Gemini API Key.
3. **Execution:** Launch the .html file via a local server (e.g., VS Code Live Server) to bypass browser CORS restrictions for webcam access.