**Study of Equation‑Based Generative Art and Real‑Time Animation Control via User Input and Audio Feedback**

**Author:** Shivang

**Abstract**  
This report explores mathematical and computational methods for equation‑based generative art and systems that enable real‑time animation control through user input (mouse, touch, MIDI, UI sliders) and audio feedback (live audio analysis or pre-recorded audio). We present mathematical formalisms, procedural primitives, generative systems (ODEs, fractals, L‑systems, reaction‑diffusion), interactive control techniques, audio‑reactive mappings, efficient rendering strategies (GPU shaders, WebGL), evaluation metrics, and example pipelines and experiments. The goal: provide a comprehensive, reproducible blueprint for building expressive, responsive, and high‑performance generative art installations and applications.

**1. Introduction and scope**

Equation‑based generative art uses mathematical functions, differential equations, iterative maps, and procedural noise to create visual content. When combined with real‑time control and audio reactivity, these systems become expressive instruments that can be performed or explored interactively.

This document covers:

* Core mathematical building blocks and equations used in generative visuals.
* Architectures for real‑time control with low latency.
* Audio analysis and mapping strategies for expressive animation.
* Implementation techniques (GPU, Web Audio + WebGL, Unity/Unreal, Node + socket streams).
* Design patterns, UX considerations, and evaluation criteria.

**2. Mathematical foundations and primitives**

**2.1 Continuous functions and fields**

Treat images as functions over space and time:

Visuals are generated by evaluating parametric functions at pixel coordinates and time .

Common primitives:

* Sinusoids: .
* Radial basis: .
* Polynomials and rational maps.

**2.2 Iterated function systems & fractals**

Iterated maps produce self‑similar structure. Example: complex quadratic map (Julia/Mandelbrot):

Colouring escape time and distance estimators produce detailed fractal visuals.

**2.3 Dynamical systems and ODEs**

Continuous dynamics can be simulated with ODEs and visualized as trajectories or field integrals. Example: Lorenz attractor

Parameter modulation (via user input or audio) drives changes in chaos and structure.

**2.4 Reaction–diffusion**

Reaction–diffusion PDEs (Gray–Scott, Turing patterns) produce organic patterns. Discretized form:

Finite difference schemes on a grid with carefully chosen yield diverse textures.

**2.5 Noise functions**

Procedural noise (Perlin, Simplex, OpenSimplex) is crucial for organic textures. Multiscale fractal noise:

**2.6 Vector fields and flow visualization**

Define a time‑varying vector field . Particle advection and LIC (line integral convolution) visualize flow:

Particle update:

**3. Interaction & control modalities**

**3.1 User input sources**

* Mouse / touch: continuous 2D coordinates, pressure (if supported).
* Keyboard: discrete events, shortcuts, step control.
* MIDI / OSC: high‑precision controllers (knobs, faders) and note triggers.
* UI Sliders and presets: for discoverability and saving configurations.
* External sensors: camera (pose), microphone (audio), accelerometer.

**3.2 Mapping strategies**

Mapping raw inputs to parameters must balance expressivity and stability.

* **Direct mapping:** slider → parameter (linear or log scaling).
* **Scalarization & normalization:** scale inputs to normalized range ([0,1]) or log domain.
* **Nonlinear mappings:** exponential/log, power curves for perceptual control.
* **Stateful controllers:** velocity, momentum, easing to smooth abrupt changes.
* **Mode switching & context:** map same controls to different parameter sets depending on mode.

**3.3 Latency and responsiveness**

Aim for overall system latency < 30–50 ms for a responsive feel. Key techniques:

* Use WebAudio or native audio APIs for low‑latency audio capture.
* Run heavy simulation tasks on GPU shaders or WebWorkers.
* Send lightweight control messages (OSC/MIDI) and avoid synchronous blocking operations.

**4. Audio analysis & mapping for reactivity**

**4.1 Audio features**

Extract features in real‑time (short frames, e.g., 512–2048 samples):

* RMS / loudness.
* Spectral centroid, bandwidth.
* Mel‑spectrogram / chroma features.
* Onset detection and beat tracking.
* Per‑band energy (subbands) via filterbank or STFT.

**4.2 Time–frequency pipelines**

Use STFT: frame size , hop length . Complex STFT yields magnitude spectrogram .

Mel spectrogram and MFCC pipelines compress spectral info to perceptual bands.

**4.3 Feature smoothing and envelope following**

Apply exponential moving average to stabilize control signals:

Use attack/release times to control responsiveness.

**4.4 Mapping audio to visuals**

Design mapping strategy:

* **Low‑level mapping:** band energy → particle emission rate or stroke thickness.
* **High‑level mapping:** beat detection → spawn events, chord changes → colour palettes.
* **Perceptual mappings:** spectral centroid → colour temperature; spectral flux → motion intensity.

Normalize and compress audio features to avoid clipping and maintain dynamic range.

**5. Rendering & performance**

**5.1 GPU shaders and WebGL**

Write fragment and vertex shaders to compute visuals per pixel with massive parallelism. Benefits:

* Real‑time reaction at 60+ FPS for complex formulas.
* Deterministic, high‑throughput computations for per‑pixel PDEs, noise, and raymarching.

Shaders commonly used:

* Fragment shaders for field evaluation and colour mapping.
* Compute shaders (WebGPU / desktop GL) for particle updates and cellular automata.

**5.2 Compute budgets and precision**

* Use single precision floats for speed; consider 16‑bit float where available.
* For iterative solvers (reaction‑diffusion), use ping‑pong framebuffers.

**5.3 Hybrid CPU/GPU pipelines**

* CPU handles control logic, audio analysis (unless using GPU FFT), and scene graph.
* GPU handles heavy per‑pixel calculations and particle advecting.

**5.4 Frame timing and synchronization**

* Keep simulation timestep fixed for stability (e.g., 1/60s) and interpolate for rendering.
* Audio latency alignment: use timestamps from audio API to align visuals to audio frames.

**6. Example equation families & mappings (with equations)**

**6.1 Oscillatory field**

where and are spatial envelopes (noise, radial falloff).

**6.2 Flow with curl noise**

Define velocity field:

where is a multi‑octave noise. This yields divergence‑free swirling motion.

**6.3 Reaction–diffusion (discrete update)**

Finite difference scheme on grid point :

with discrete Laplacian .

**6.4 Particle advection with audio-driven birth rate**

Particle count creation rate:

Particle velocity influenced by audio envelope:

**7. System architecture and dataflow**

A robust interactive system typically includes these components:

1. **Input layer:** Mouse/touch, MIDI/OSC, audio capture.
2. **Analysis layer:** Audio feature extractor, input smoothing, beat tracker.
3. **Mapper / Controller:** Maps features to parameters, applies easing and mode logic.
4. **Simulation layer:** GPU shaders, particle systems, ODE solvers, reaction‑diffusion.
5. **Renderer:** Final compositing, postprocessing (bloom, color grading), UI overlay.
6. **Recorder/export:** Capture frames, video, or serialized parameter logs for replay.

Use lightweight messaging (WebSocket / WebRTC / local OSC) to connect remote controllers or distributed nodes.

**8. UX, design patterns, and affordances**

* Provide clear defaults and a small set of knobs for live performance.
* Offer presets and randomize function for serendipity.
* Visual feedback for control mappings (e.g., show which parameter a knob controls).
* Allow parameter recording and automation for repeatable sequences.

**9. Evaluation and metrics**

Quantitative evaluation is limited for aesthetic systems, but useful proxies include:

* **Responsiveness:** end‑to‑end latency in ms.
* **Stability:** frame rate (median, 1% lows) under load.
* **Control coverage:** sensitivity and dynamic range of controls.
* **Perceptual studies:** user preference, expressivity ratings by human subjects.

**10. Experiments & example projects**

**10.1 Audio‑reactive flow painter**

* Use curl noise field with particle advection; map low‑band energy to emission rate and high‑band centroid to color.
* Implement particle trails via framebuffer accumulation and fade.

**10.2 Live sketching via ODEs**

* Use controlled Lorenz or Rössler attractors; map MIDI knobs to parameters and draw trajectories in 2D colour channels.

**10.3 Reaction–diffusion instrument**

* Real‑time parameter modulation via touch or MIDI; use GPU to compute RD and apply palette mapping to visualize transitions.