Design Document: Genomancer Visual Synthesizer (Successor to Radial Geometry Explorer v3)

1. Introduction & Vision

Project Title: "Genomancer Visual Synthesizer" (GVS)

Vision: To create an interactive, web-based artistic tool that empowers users to design and visualize evolving, aesthetically pleasing geometric patterns based on polar parametric equations. Users will work with multiple **Layers** to construct a **Scene**, and multiple Scenes can be sequenced into **Compositions** with **Transitions**. The tool will build upon the foundations of "Radial Geometry Explorer v2," offering enhanced flexibility in curve definition (in later stages), more sophisticated artistic controls, and a more robust and intuitive user interface designed for scalability and creative exploration. The emphasis remains on the beauty of mathematical art, facilitated by the p5.js library leveraging the HTML 2D Canvas.

Core Idea: Users will select and control predefined polar curves for (initially up to three via the UI) **Layers** within a **Scene**. The underlying architecture will support an arbitrary number of Layers. The theta range for curve completion will be automatically determined where applicable. Each Layer's parameters and visual representation (color, line weight, fill, effects) can animate over time via a global t parameter, initially through simple amplitude/speed controls with a choice of linear or sinusoidal response, and later with more advanced animation envelopes. Scenes can then be arranged into **Compositions** with a timeline and transitions, allowing for dynamic, multi-part visual narratives. The tool aims to be a powerful creative outlet, an educational instrument, and a more streamlined experience than its predecessor.

Motivation for "Fresh Restart": "Radial Geometry Explorer v2" (RGE) was an exploratory project. As features were added, UI event conflicts and architectural limitations began to emerge. "Genomancer Visual Synthesizer" aims for a cleaner initial design, anticipating a fuller feature set and ensuring the UI can gracefully handle complexity, including the new Composition structure and flexible Layer management.

2. Review of Predecessor: "Radial Geometry Explorer v2"

"Radial Geometry Explorer v2" is a single-page web application allowing users to:

- **Configure up to three independent geometric "Figures":** Each figure is displayed on its own canvas and controlled by a dedicated set of UI elements.

- **Select from predefined figure types:** Centered Circles, Rhodonea Curve, Epitrochoid, Hypotrochoid.
- **Control figure parameters:** Including basic properties, geometric parameters, transformations, animation, and fill options.
- **View a Combined Output (termed "Overlay" in RGE):** All active figures are rendered onto a larger, shared canvas with blending modes.
- **Manage Snapshots:** Save/load the entire state to localStorage.

Technology: Vanilla JavaScript with direct DOM manipulation for UI and 2D Canvas API for rendering.

(Note: For brevity, some details of RGE are omitted here but are present in the previous version of this document if needed.)

3. Comparison and Evolution: V2 vs. Genomancer Visual Synthesizer

Feature Area	Radial Geometry Explorer v2	Genomancer Visual Synthesizer (Proposed)	Key Changes/Improvements
Core Library	Vanilla JS, 2D Canvas API	**p5.js leveraging HTML 2D Canvas**, HTML, CSS, JS	Leverage p5.js for drawing loop, math functions, color modes (HSB), and potentially simplified event handling for the canvas.
Curve Definition	Predefined types (Centered Circles, Rhodonea, Trochoids)	**Initially predefined types (Rhodonea, Epitrochoid, Hypotrochoid)** for each Layer. Future: User-defined polar equations r=f(θ,t).	Phased approach. Start with robust predefined types, then expand. User-defined equations are a significant future enhancement.
Structural Units	Up to 3 "Figures" combined into an "Overlay".	**MVP UI: 3 "Layers"** per **"Scene"** (architecture supports arbitrary Layers). Multiple Scenes form a	Major structural enhancement. Introduces temporal sequencing. New nomenclature. Scalable Layer

		"Composition" with **"Transitions"**.	architecture.
Evolution/Animation	Animation of size, rotation, line width via sinusoidal/linear change per Figure.	Global **time parameter t** per Scene. Layer parameters animated via **MVP: simple speed/amplitude sliders with linear/sinusoidal choice; Post-MVP: easing functions/curves.** Composition timeline controls Scene playback.	More powerful and generic evolution within Scenes. Phased approach to animation complexity. Compositions add higher-level temporal control.
Artistic Styling	Basic color, line, fill (solid, 2-color gradients, presets).	Advanced color (gradients along curve, palette cycling), line styles, **post-processing effects** (bloom, blur) per Layer/Scene.	Deeper artistic control. p5.js shaders could be a future goal.
UI Framework	Vanilla JS, manual DOM manipulation. Multi-panel layout.	HTML/CSS/JS with p5.js. **Slide-out Save/Load Hub (Layers, Scenes, Compositions). Main area with top tabs ("Scene Editor", "Composition Editor"). Draggable divider in Scene Editor.**	Centralized Save/Load. Clearer separation of Scene vs. Composition editing contexts. Addresses RGE UI complexity. Scalability.
Preset System	Save/Load full state (3 figures + overlay) as "Snapshots".	Save/Load individual **Layers**, **Scenes**, and **Compositions** via dedicated Save/Load Hub.	More granular control over saving/loading components. "Snapshots" concept evolves into specific data structures.

4. Target Audience (Consistent)

- Artists & Designers, Students & Educators, Hobbyists & Enthusiasts.
- Aim for usability without prior coding, but with depth for advanced users.

5. Core Features of Genomancer Visual Synthesizer

5.1. Layer Configuration (within the "Scene Editor" Tab)

- **Multiple Layers per Scene:**
  - **MVP UI:** Will provide controls for up to 3 configurable Layers per Scene.
  - **Architecture:** The underlying data structures and rendering logic will support an arbitrary number of Layers per Scene.
- **Predefined Equation Selector per Layer:**
  - User selects a curve type from a predefined list (initially: Rhodonea, Epitrochoid, Hypotrochoid) for each active Layer.
- **Parameter Controls per Layer:**
  - A variety of UI controls appropriate for each parameter of the selected predefined equation. The specific groups are: Name, Shape, Line, Fill, Size, Rotation.
- **Curve Resolution/Steps per Layer:** Control over the number of points/steps for rendering each Layer's curve, affecting its smoothness. The actual theta range required for curve completion will be automatically calculated by the application for relevant curve types.
- **(Future) User-Defined Equations per Layer.**

5.2. Scene Definition (within the "Scene Editor" Tab)

- **Scene Composition:** A Scene is defined by the configuration of its Layers, their individual parameters, styles, and how they are blended together.
- **Scene-Level Time (t):** Each Scene will have its own instance of a global time parameter t that drives animations within its Layers.
- **Blending Modes:** Control how Layers within a Scene are blended together.

5.3. Composition Authoring (within the "Composition Editor" Tab - Functionality Deferred Post-MVP, UI Shell in MVP)

- **Composition Timeline:** A visual or list-based interface to arrange multiple Scenes in sequence.
- **Scene Duration:** Users can specify how long each Scene plays within the Composition.
- **Transitions:**

    - Users can select or define transitions between Scenes.
    - **Early Post-MVP Implementation:** Simple crossfade via opacity levels.
    - Transition duration control.
- **Composition Playback:** Controls to play, pause, and scrub through the entire Composition.

5.4. Evolution & Animation (controls within "Scene Editor" Tab)

- **Layer Parameter Animation:**
    - **MVP Approach:** For animatable parameters within a Layer (e.g., n, scale, rotation), users will have dedicated UI controls for:
        - A base value.
        - An animation toggle (on/off).
        - Animation speed (slider).
        - Animation amplitude (slider, defining the range, e.g., baseValue +/- amplitude).
        - Animation type (dropdown: "Linear Ramp", "Sinusoidal Oscillation").
            - *Linear Ramp:* currentValue = baseValue + t * speed * (amplitude / typical_duration_estimate) or similar, effectively using amplitude as a rate multiplier.
            - *Sinusoidal Oscillation:* currentValue = baseValue + amplitude * sin(t * speed).
    - **Post-MVP Enhancements:**
        - Introduction of animation envelopes (e.g., based on Penner easing functions, symmetric Penner pairs, cyclic pulse with decay) for more nuanced control over how a parameter changes over the Scene's duration or in response to t.
        - Eventually, possible full envelope control.
        - Ability to define animation curves more directly (e.g., specifying parameter = base + scaleFactor * sin(t * frequency + phase)).
- **Scene Animation Controls:** Play, pause, reset, speed control for the current Scene's t parameter.

5.5. Artistic Rendering & Styling (controls within "Scene Editor" Tab)

- **Color:** p5.js HSB Mode, stroke/fill, advanced gradients, palettes, alpha.
- **Line Style:** Weight, dashes/dots.
- **Fill:** For closed paths.
- **Background:** Per Scene (or a default).
- **(Future) p5.js Effects:** Shadows, blur, bloom.

5.6. User Interface & Experience (Major Revision)

- **Overall Layout:**
  - **Left Slide-Out Save/Load Hub:**
    - Collapsible panel, activated by a button/icon.
    - Contains vertical tabs: "Layers," "Scenes," "Compositions."
    - Each tab provides UI for listing, saving, loading, and deleting the respective data items.
  - **Main Workspace Area:**
    - Governed by two primary horizontal tabs at the top: "Scene Editor" and "Composition Editor."
- **"Scene Editor" Tab (MVP Primary Focus):**
  - **Visual Area (Top, resizable portion):**
    - A large main canvas displaying the rendered Scene (composite of layers). Max size 800x800px, centered.
    - A row of three small preview canvases below the main canvas, one for each active Layer.
  - **Controls Area (Bottom, resizable portion, below canvases, separated by draggable horizontal divider):**
    - A single, unified panel for controls. The content dynamically reflects the settings for the **currently selected Layer Preview**.
    - This panel contains subsections (fully visible, no accordions for MVP): Name, Shape (Curve Type selector, dynamic Parameter Controls for the selected curve, Curve Resolution/Steps input), Line, Fill, Size, Rotation.
    - A section for "Global Scene Settings": Background color, overall Layer Blending mode, Scene-level animation controls (for t).
- **"Composition Editor" Tab (MVP Shell):**
  - Placeholder UI for the composition timeline, scene sequence management, and transition settings. Visually distinct (e.g., grayed out) to indicate future functionality.
- **Real-time Preview:** Essential for both individual Layers and the combined Scene.
- **"Surprise Me" / Randomization:** For Layer parameters.
- **Context Menus:** For quick actions like "Save Layer Snapshot" on Layer Previews.

5.7. Save/Load Functionality (New Section, Centralized in Hub)

- **Save/Load Hub:** A dedicated slide-out panel.
- **Layers:**
  - Ability to save the configuration of an individual Layer (LayerConfig object).
  - Ability to load a saved Layer configuration into any of the three Layer slots in the current Scene.
- **Scenes:**

    - Ability to save the current Scene configuration (SceneConfig object, including all its Layer configurations).
    - Ability to load a saved Scene, replacing the current Scene.
- **Compositions (Post-MVP for full functionality, MVP shell for save/load buttons):**
    - Ability to save the current Composition (CompositionConfig object).
    - Ability to load a saved Composition.
- **Storage:** Likely localStorage.

6. Technical Design (p5.js centric)

(This section would be significantly impacted if a framework is chosen. For now, it reflects the vanilla JS + p5.js baseline, but would need revision based on Section 11's outcome).

6.1. Core Technologies

- **p5.js leveraging HTML 2D Canvas**, HTML5, CSS3, JavaScript (ES6+).
- *(If a framework is adopted, it would be listed here, e.g., React with p5.js).*

6.2. Key p5.js Features to Leverage

- setup() & draw() loop.
- Coordinate System: translate(), rotate(), scale(). push()/pop().
- Drawing Primitives: beginShape(), vertex(), endShape(CLOSE).
- Color Modes: colorMode(HSB) is highly recommended. fill(), stroke(), background().
- Math Functions: sin(), cos(), noise(), map().
- Instance Mode: Highly recommended for managing multiple p5.js sketches (Layer Previews, Main Scene).

6.3. Data Structures

- **LayerParameterConfig Object (for a single animatable parameter):**
    - baseValue: The static or starting value.
    - isAnimated: boolean.
    - animationType: string (MVP: "linearRamp", "sinusoidalOscillation").
    - animationSpeed: number.
    - animationAmplitude: number.
    - (Post-MVP) animationEnvelopeType: string (e.g., "pennerEaseInQuad").
    - (Post-MVP) animationEnvelopeParams: object for envelope-specific settings.
- **LayerConfig Object:**
    - id: Unique identifier.

    - name: String (user-defined).
    - isActive: boolean.
    - curveType: String (e.g., "rhodonea").
    - curveParams: Object where keys are parameter names (e.g., n, d, scale, rotationDegrees, lineWidth) and values are LayerParameterConfig objects or simple static values if not animatable (e.g. color hex string).
    - curveResolution: Number (e.g., 500 steps).
    - styleParams: Object for non-animatable style properties (e.g., fill type, gradient definitions if not directly part of animatable color parameters). (ThetaMin, ThetaMax removed from explicit config here, will be derived or fixed based on curve type)
- **SceneConfig Object:**
    - id: Unique identifier.
    - name: String (user-defined).
    - layers: Array of LayerConfig objects [MVP UI handles 3, architecture supports more].
    - layerBlendingMode: String (global blending for layers within the scene).
    - backgroundColor.
    - sceneTimeParams: { animationSpeedFactor: 1.0, currentTime_t: 0 }
- **(Post-MVP) TransitionConfig Object & CompositionConfig Object:** (As previously defined).
- **Global State / Playback Engine:**
    - activeWorkspaceTab: String ("sceneEditor", "compositionEditor").
    - activeScene: Reference to the currently loaded/active SceneConfig.
    - currentlySelectedLayerIndex: Number (0, 1, or 2 for MVP).
    - isPlayingScene: Boolean.
    - (Post-MVP) currentComposition, compositionPlayhead, isPlayingComposition.

6.4. Core Algorithms

- **Layer Rendering:**
    1. Determine appropriate theta range for the selected curveType (e.g., for Rhodonea r=acos(kθ), if k=n/d, range is 0 to 2πd/gcd(n,d) or simply 2πd if n,d are coprime). For spirals, a fixed large range might be used.
    2. For each parameter in layerConfig.curveParams: (Calculate animated value).
    3. Iterate θ over the determined range using layerConfig.curveResolution steps.
    4. Convert (r,θ) to Cartesian (x,y).
    5. Apply styling (colors, line weight from styleParams or animated curveParams).
    6. vertex(x, y).
- **Scene Rendering:**

1. Clear main scene canvas with activeScene.backgroundColor.
2. For each active LayerConfig in activeScene.layers:
   a. Set p5.js blendMode() based on activeScene.layerBlendingMode (or individual layer blend mode if implemented later).
   b. Render the layer (either by redrawing its geometry or by drawing its pre-rendered preview canvas/buffer).

- **(Post-MVP) Composition Playback Engine & Transition Logic.**

7. Aesthetics and Artistic Principles

- Emphasis on flow, movement, harmonic proportions, and color theory.
- HSB color mode in p5.js will be a significant aid.
- (Post-MVP) Temporal composition and narrative through Scene sequencing.

8. UI Event Handling and State Management Strategy (Baseline without dedicated framework)

- UI controls within the "Scene Editor" tab's unified control panel update the activeScene.layers[currentlySelectedLayerIndex] or activeScene (for global settings).
- Selection of a Layer Preview updates currentlySelectedLayerIndex and triggers a refresh of the control panel to show that layer's settings.
- The Save/Load Hub interacts with localStorage and updates the activeScene or individual layers within it.
- Robust functions to populate all UI controls from loaded LayerConfig, SceneConfig objects are critical.
- *(This section would be entirely rewritten based on the chosen framework's patterns if one is adopted).*

9. Potential Future Enhancements (by priority)

1. Full Composition and Transition functionality.
2. UI for Dynamic Layer Addition/Removal in a Scene.
3. Export (PNG, SVG for Scenes; GIF/MP4 for Compositions).
4. Advanced animation envelopes.
5. Audio Reactivity.
6. User-Defined Polar Equations per Layer.
7. Advanced Equation Editor.
8. Shader Integration.
9. Community Sharing.
10. Undo/Redo.

10. Open Questions & Discussion Points

- **Initial Predefined Equation Set:** Confirming Rhodonea, Epitrochoid, Hypotrochoid for Layers.
- **MVP Focus for GVS (Revised UI Structure):**
  - **Save/Load Hub (Slide-Out Left Panel):**
    - Functional Save/Load for individual **Scenes**.
    - UI shell for Save/Load of **Layers** and **Compositions**.
  - **Main Workspace with Top Tabs:**
    - **"Scene Editor" Tab:**
      - Main Scene Canvas (max 800x800px, centered) at the top.
      - Row of 3 small Layer Preview canvases below the main canvas.
      - Unified controls panel below previews (separated by draggable horizontal resizer), dynamically showing settings for the selected Layer:
        - Name input.
        - Subsections (fully visible, no accordions for MVP): Shape (Curve Type, specific params, Resolution), Line, Fill, Size, Rotation.
        - Each animatable parameter has base value, anim toggle, speed/amplitude sliders, and anim type dropdown (Linear/Sinusoidal).
      - Global Scene Settings section (background, blending, scene time controls).
      - p5.js rendering.
    - **"Composition Editor" Tab:** (UI Shell as previously defined)
  - **Crucially, robust Save/Load of individual Scene configurations must be part of the MVP.**
- **(Post-MVP) Transition Implementation:** Initial simple crossfade.
- **(Post-MVP) Advanced Animation:** Penner easing functions, etc.

11. State Management and UI Framework Considerations

The Genomancer Visual Synthesizer (GVS), with its multiple interactive layers, real-time parameter animation, dynamic UI updates based on selection, and future plans for compositions, presents significant challenges in managing application state and ensuring the UI remains consistent and performant. This section explores considerations for tackling these challenges.

11.1. Challenges

- **Complex State:** Managing the parameters for up to three (or more,

architecturally) layers, each with numerous animatable and styleable properties, plus scene-level settings, selection states, and future composition data.
- **UI-State Synchronization:** Ensuring that changes in the underlying JavaScript data model are accurately and efficiently reflected in the numerous UI controls (sliders, inputs, pickers, etc.), and vice-versa.
- **Dynamic UI:** The controls displayed for a layer change based on the selected curve type. The active layer's settings panel content also changes based on user selection.
- **Inter-component Communication:** Changes in one part of the UI (e.g., selecting a layer preview) need to trigger updates in other parts (e.g., the settings panel).
- **Maintainability and Scalability:** As features like compositions and more advanced animations are added, the complexity of managing state with direct DOM manipulation can grow exponentially, potentially leading to bugs and difficult maintenance (as experienced with RGE).

11.2. Approach 1: Vanilla JavaScript + p5.js (Current Baseline)

This approach involves using vanilla JavaScript for all UI logic and state management, with p5.js dedicated to canvas rendering.

- **Pros:**
  - Direct control over the DOM and event handling.
  - No additional framework learning curve if the developer is already proficient in JS.
  - Potentially less initial setup overhead for a project primarily focused on p5.js visuals.
- **Cons:**
  - **Manual DOM Manipulation:** Requires developers to write explicit code to create, update, and remove UI elements, which can be verbose and error-prone.
  - **State Synchronization:** Keeping the JavaScript data model and the UI (HTML controls) in sync manually can be complex. Data flows in two directions, and ensuring consistency requires careful event handling.
  - **Scalability:** As the application grows, the interconnectedness of different UI parts and state variables can become difficult to trace and manage, increasing the risk of bugs and making refactoring harder.
  - **Code Organization:** Requires strong discipline and patterns (e.g., custom event systems, a clear model-view separation) to keep the codebase maintainable.

11.3. Approach 2: UI Framework (e.g., React, Vue, Svelte) + p5.js

This approach involves using a modern JavaScript framework to manage the UI components and application state, while p5.js instances handle canvas rendering.

- **Pros:**
  - **Component-Based Architecture:** Breaks the UI into smaller, reusable, and manageable components.
  - **Declarative UI:** Simplifies UI update logic.
  - **State Management Solutions:** Offers robust patterns for managing complex application state.
  - **Improved Scalability and Maintainability.**
  - **Developer Experience & Ecosystem.**
- **Cons:**
  - **Learning Curve & Initial Setup.**
  - **p5.js Integration Strategy Needed:** Requires careful planning to combine the framework's lifecycle with p5.js's rendering loop (typically via instance mode and wrapper components).
  - **Bundle Size:** Potentially larger, though mitigatable.

11.4. Recommendation and Framework Choice Considerations for GVS (Revised)

Given the planned complexity and interactive nature of GVS, **adopting a modern JavaScript UI framework is strongly recommended** over a purely vanilla JavaScript approach for managing the UI and application state. The benefits in terms of organization, state management, scalability, and long-term maintainability are likely to be substantial, especially considering the lessons from RGE.

The choice among popular frameworks like React, Vue, and Svelte depends on several factors:

- **React:**
  - **Strengths:** Largest ecosystem and community support, robust patterns for complex state management (Context API, Redux, Zustand), and a component model that's well-suited for large applications. Its explicitness can also be beneficial for clarity, including when collaborating with AI coding partners. Established patterns exist for p5.js integration.
  - **Considerations:** Can have a steeper learning curve for those unfamiliar with JSX and its state management paradigms.
- **Vue.js:**
  - **Strengths:** Often cited for its gentle learning curve and excellent documentation, making it approachable. It's a progressive framework that can be adopted incrementally. Vue 3 with Pinia offers a very intuitive state management solution. p5.js integration is straightforward through component

wrappers.
    - **Considerations:** While its ecosystem is large and mature, it's not as vast as React's.
- **Svelte:**
    - **Strengths:** Acts as a compiler, resulting in highly performant vanilla JavaScript with minimal runtime overhead and small bundle sizes—very attractive for a creative tool. Its syntax is close to plain HTML/JS/CSS, and reactivity is a core feature. p5.js integration is also clean.
    - **Considerations:** Has the smallest (though rapidly growing) ecosystem of the three. The "compiler as a framework" approach, while powerful, might feel less transparent to developers accustomed to runtime frameworks.

General Guidance for GVS:

- **If developer familiarity exists with one of these frameworks, that often becomes the most pragmatic choice.**
- For a project of GVS's potential complexity aiming for rich interactivity and future expansion (like compositions), **React** is a very strong candidate due to its mature ecosystem and patterns for large-scale state management.
- **Vue** offers a compelling balance of ease of use, performance, and a solid state management story with Pinia.
- **Svelte** is an excellent choice if peak performance, minimal bundle size, and a very clean developer experience are top priorities, provided its ecosystem meets the project's needs.

Regardless of the specific choice, a clear strategy for integrating p5.js (likely using instance mode and wrapper components that receive state as props) will be essential. The chosen framework will primarily handle the UI controls, application logic outside of pure rendering, and overall state, passing necessary data to the p5.js sketches for visualization.

Next Steps if a Framework is Adopted:
If the decision is made to proceed with a framework, Section 6 (Technical Design) and Section 8 (UI Event Handling and State Management Strategy) of this document will require significant revisions to detail:

- The chosen framework and its version.
- Key libraries to be used (e.g., for state management, routing if applicable).
- The component hierarchy.
- The specific strategy for state flow between framework components and p5.js sketches.
- The build process and development environment setup.

This updated document now includes a dedicated section on these important architectural considerations.