Bit Dot Night

https://github.com/dkconnect/bit-city | https://bit-city.vercel.app/

***Abstract*—Bit Dot Night is a generative art project which combines aesthetic design with mathematical and computational principles. Created using JavaScript, HTML, and CSS, the application produces reproducible, dynamic city scapes of pixels. This paper presents the project's architecture, focusing on the seeding mechanism, mathematical foundations, and rendering pipeline that enable randomized yet repeatable artwork.**

***Index Terms*—Generative Art, Procedural Graphics, Randomness, JavaScript, Canvas API, Seeding, Generation**

I. Introduction

Bit Dot Night is an example of generative art where visuals are produced not manually but algorithmically. The project generates ever-changing city scapes of pixels, using JavaScript for logic, HTML for structure, and CSS for styling and UI. This paper explores its architecture, mathematical underpinnings, and mechanisms for reproducible randomness.

II. Foundation

At the heart of every generative art lies randomness. For reproducibility, randomness must be controlled. "Bit Dot Night" achieves this using seeded pseudo-random generation. The system begins by initializing an `fxhash` value, a Base58-encoded string. Using the `b58dec` function, the string is decoded into integers, which seed the SFC32 pseudo-random number generator. Unlike `Math.random()`, SFC32 produces deterministic sequences, ensuring identical images for a given hash.

The project builds further with a custom `Random` class, seeded from SFC32. It provides methods such as `float`, `int`, and `bool`, internally using a xorshift algorithm. Xorshift, a linear-feedback shift register generator, offers speed and long periods, enabling consistent randomness for color palettes, structures, and visual traits.

III. Mathematics

The pixel mosaics stem from applying operators to coordinates via the `applyOperator` function. The result determines brightness or color.

A. Bitwise Operators

- AND (&) creates checkerboard-like patterns.
- OR (|) produces dense, less structured textures.
- XOR () generates diagonal stripes and tile-like designs.

B. Arithmetic Operators

Addition and multiplication yield smooth gradients, while division produces repeating, complex textures.

C. Combined Operator

The operator $(x - y)^(x + y)$ merges subtraction, addition, and XOR, generating complex, unpredictable visuals.

Randomized selection of background and window operators ensures distinct output in every generation.

IV. Code Mechanism

The project's logic is implemented with JavaScript functions and classes. The `Random` class provides deterministic randomness throughout.

The `startNewGeneration()` function resets state, cancels animations, clears the canvas, and sets parameters such as operator types, flags (e.g., earthquake, grayscale), and color values. It calls `drawBackground()` and initiates an animation loop via `requestAnimationFrame(update)`.

The `update()` function drives animation, continuously drawing buildings and windows while applying effects such as grayscale or inversion. A `compositeImage()` function applies global composite operations to integrate final effects.

The `downloadArt()` function allows users to save images by converting high-resolution canvas data into PNG format.

V. Visual Rendering

The HTML5 `<canvas>` element handles rendering. To minimize flicker, off-screen rendering is used: a hidden high-resolution canvas (4096 x 2048) performs drawing before transferring to the main canvas.

The `drawBackground()` function renders skies and moons in HSL color space. Difference blending creates inverse-color lunar effects. The `update()` function then adds buildings and windows, with flags such as `earthquake`, `thinBuildings`, `isRainbow`, and `isGrayscale` introducing variations.

CSS, styled with Tailwind, ensures responsiveness. The property `image-rendering: pixelated` preserves pixel-art sharpness.

VI. Conclusion

This project shows how seeded randomness, procedural mathematics, and efficient rendering produce reproducible generative art. Through controlled pseudo-randomness, operator logic, and canvas optimization, the project balances structured design with creative unpredictability.

References

[1] ”Bit Dot Night” Source Code, 2025. Available: https://github.com/dkconnect/bit-city

[2] fx(hash) Platform. Available: https://fxhash.xyz

[3] SFC32 Pseudo-Random Number Generator, 2018. [Online]. Available: https://github.com/bryc/code/blob/master/jshash/PRNGs.md

[4] Base58 Encoding.

[5] HTML Canvas API. Available: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

[6] MDN Web Docs. Available: https://developer.mozilla.org