INFORME TÉCNICO COMPLETO

Free Animation Glitch

37 Modos de Distorsión Visual en Tiempo Real con p5.js

Responsable del Proyecto

Eduardo Andrés Fierro Duque

Santiago de Chile — Agosto 2026

Índice

1. Resumen Ejecutivo

Free Animation Glitch es una herramienta de procesamiento visual en tiempo real que transforma video (archivo local o cámara web) mediante 37 modos de distorsión artística. Construida sobre p5.js, la herramienta aplica algoritmos de manipulación de píxeles, geometría computacional y procesamiento de señales para producir efectos que abarcan desde el puntillismo clásico (estilo Seurat) hasta efectos glitch digitales contemporáneos como datamosh y chromatic aberration.

La herramienta funciona procesando cada frame de video píxel por píxel: extrae los datos de imagen mediante getImageData, aplica el algoritmo del modo seleccionado, y renderiza el resultado en un canvas p5.js. El proceso se repite a 30-60 FPS, creando una experiencia de video en tiempo real completamente transformada. Incluye capacidades de exportación a PNG estático, SVG vectorial, GIF animado y video WebM.

2. Stack Tecnológico

Capa	Tecnología
Motor gráfico	p5.js 1.11.1 (CDN) — Framework creative coding
Canvas	Canvas 2D API via p5.js (960×540 nativo)
Video	HTMLVideoElement + getUserMedia (cámara) + Drag & Drop (archivos)
Exportación GIF	gif.js (encoder LZW con Web Workers, 0-2 workers)
Exportación Video	MediaRecorder API + canvas.captureStream(30)
Exportación SVG	Generación procedural de SVG inline con círculos por píxel
Frontend	HTML5 + CSS3 con diseño responsive, sin frameworks CSS
Autenticación	PHP session_start() con $_SESSION['email']
Idiomas	Solo español (la herramienta tiene poco texto de interfaz)

3. Arquitectura del Sistema

La arquitectura se basa en el ciclo de vida de p5.js: setup() → draw() en bucle infinito. En cada iteración de draw(), se captura el frame actual del elemento <video> mediante p5.image(), se extrae el array de píxeles (Uint8ClampedArray de tamaño 960×540×4 = 2,073,600 bytes), se aplica el algoritmo del modo activo, y se renderiza el resultado. Los controles de la UI modifican las variables de configuración (cfg) que el sketch de p5.js lee en cada frame. La comunicación entre la UI vanilla JS y el sketch p5.js se realiza mediante variables globales compartidas (cfg, hasVideo, videoReady, myP5).

4. Catálogo Completo de los 37 Efectos

PUNTILLISMO Y ESTILOS

- clasico: Puntillismo clásico (Seurat): puntos blancos sobre fondo negro. Tamaño inversamente proporcional al brillo: zonas oscuras = puntos grandes.
- organico: Puntillismo orgánico (Natural): puntos coloreados con jitter de Perlin Noise sobre fondo crema. Simula textura de pintura acrílica.
- neon: Neón digital (Glow): 3 capas concéntricas de puntos (halo exterior, medio, núcleo brillante) con colores RGB del video original.
- vivo: Partículas vivas (Fluido): sistema de partículas con dinámica de fluidos. Cada partícula tiene posición, velocidad, tamaño oscilante (sinusoidal) y fase aleatoria.

HALFTONE Y COLOR

- halftone: Halftone CMYK: separación de color en 4 planchas (Cian, Magenta, Amarillo, Negro) con puntos de diferente tamaño según la intensidad de cada canal.
- posterize: Posterizar (Pop Art): reducción de la paleta de colores a n niveles mediante cuantización por división entera.
- solarize: Solarizar (Invertido): inversión parcial de colores. Los píxeles por encima de un umbral de brillo se invierten (efecto Sabattier).
- heat_map: Heat Map (Térmico): mapea el brillo a una escala de color térmica (azul frío → rojo caliente) utilizando interpolación HSL.

GLITCH DIGITAL

- rgb_shift: RGB Shift (Glitch cromático): desplaza los canales R y B en direcciones opuestas con offset sinusoidal. Simula chromatic aberration.
- pixel_sort: Pixel Sort (Datamosh): ordena los píxeles de cada fila horizontal por brillo, creando el característico efecto de "derretimiento digital".

- scanlines: Scanlines (CRT): líneas horizontales oscuras cada 3px + viñeta radial + distorsión de borde. Emula un monitor CRT vintage.
- chromatic_fringe: Chromatic Fringe (Edge Shift): desplaza canales de color en los bordes de alto contraste, acentuando la aberración cromática.
- pixel_stretch: Pixel Stretch (Horizontal): estira los píxeles horizontalmente basado en su brillo. Píxeles brillantes = más estiramiento.
- block_shift: Block Shift (Brutal): desplaza bloques rectangulares aleatorios del frame con temporizador. Efecto de "glitch de transmisión".
- channel_delay: Channel Delay (RGB Time): retrasa los canales R, G y B por diferentes cantidades de frames usando frameHistory[].
- displacement: Displacement (Derretir): desplaza píxeles según un mapa de ruido Perlin. Crea efecto de imagen derritiéndose.

DEFORMACIÓN GEOMÉTRICA

- wave_warp: Wave Warp (Ondulante): desplaza filas de píxeles horizontalmente con función sinusoidal de amplitud y frecuencia variables.
- fisheye: Fisheye (Ojo de pez): distorsión de lente gran angular. Los píxeles se alejan del centro proporcionalmente a su distancia.
- swirl: Swirl (Remolino): rota los píxeles alrededor del centro con fuerza proporcional a la distancia. Ángulo = dist² × factor.
- ripple: Ripple (Ondas): desplazamiento sinusoidal radial desde el centro, simulando ondas en el agua.
- pixelate: Pixelate (Mosaico): reduce la resolución efectiva agrupando píxeles en bloques cuadrados y promediando su color.
- lenticular: Lenticular (Espejo): refleja columnas alternas de píxeles horizontalmente, creando patrón de espejo vertical.
- caleidoscopio: Caleidoscopio (Mirror): refleja la imagen en múltiples planos radiales (típicamente 8-12 segmentos).

EFECTOS TEMPORALES

- motion_trails: Motion Trails (Ghost): superpone frames anteriores con opacidad decreciente. background(0, alpha) controla la persistencia.
- motion_map: Motion Map (Trayectoria): calcula la diferencia entre frame actual y anterior. Los píxeles que cambiaron se colorean.
- afterimage: Afterimage (Fantasmas): mantiene una "estela" de frames pasados con decaimiento exponencial de opacidad.
- time_slit: Time Slit (Temporal): divide el canvas en columnas verticales, cada una mostrando un frame de diferente momento temporal.

- strobe_freeze: Strobe Freeze (Shutter): congela el frame actual durante n frames, luego captura uno nuevo. Simula luz estroboscópica.

ALGORITMOS AVANZADOS

- triangulacion: Triangulación (Mesh): divide el canvas en triángulos (Delaunay simplificado) y colorea cada uno con el color promedio de sus 3 vértices.
- voronoi: Voronoi (Celdas): diagrama de Voronoi con semillas aleatorias. Cada píxel se colorea según la semilla más cercana (distancia euclidiana).
- flow_field: Flow Field (Corrientes): partículas que se mueven siguiendo un campo vectorial generado por ruido Perlin 3D (x, y, tiempo).
- feedback: Feedback (Recursivo): utiliza un p5.Graphics buffer para realimentar la imagen. Cada frame se escala ligeramente y se superpone.

POST-PROCESADO

- ascii_art: ASCII Art (Terminal): convierte cada bloque de píxeles a un carácter ASCII según su brillo. Mapeo: " .:-=+*#%@" de oscuro a claro.
- edge_detect: Edge Detection (Contornos): aplica kernel Sobel 3×3 para detectar bordes. Magnitud del gradiente = color del borde.
- emboss: Emboss (Relieve): aplica kernel de relieve ([[-2,-1,0],[-1,1,1],[0,1,2]]) y añade offset de gris para centrar el resultado.
- dot_matrix: Dot Matrix (Matricial): simula una impresora matricial con puntos CMYK en patrón de roseta.
- paint_drip: Paint Drip (Escurrimiento): simula pintura escurriendo hacia abajo. Las filas se desplazan verticalmente con ruido Perlin acumulativo.

5. Fórmulas Matemáticas por Categoría de Efecto

5.1 Puntillismo Clásico (modo "clasico")

Para cada punto en la grilla (x, y) con paso step = max(cfg.density, 2):

R = píxeles[idx], G = píxeles[idx+1], B = píxeles[idx+2]

brillo = (R + G + B) / 3

tamaño = map(brillo, 0→255, maxSz→minSz)

círculo_blanco(x, y, max(tamaño, 0.5)) sobre fondo negro

La función map() de p5.js es una interpolación lineal: map(val, a, b, c, d) = (val-a)/(b-a) × (d-c) + c

tamaño(brillo) = (brillo/255) × (minSz - maxSz) + maxSz

= -(brillo/255) × (maxSz - minSz) + maxSz

Es una relación lineal inversa: a mayor brillo (zonas claras), menor tamaño de punto (más espacio blanco visible). A menor brillo (zonas oscuras), mayor tamaño de punto (se ve más negro). Esto es exactamente el principio del puntillismo de Seurat.

5.2 Neón Digital (modo "neon")

3 capas concéntricas por punto:

Capa 1 (halo): círculo con radio = tamaño × 2.8, alpha = 25

Capa 2 (medio): círculo con radio = tamaño × 1.6, alpha = 70

Capa 3 (núcleo): círculo con radio = tamaño × 0.7, alpha = 230

La superposición de 3 círculos concéntricos con opacidad creciente hacia el centro crea el característico efecto de "tubo de neón" con difusión gaussiana aproximada. El radio del halo (2.8×) comparado con el núcleo (0.7×) produce una razón de difusión de 4:1, que es visualmente similar a un desenfoque gaussiano con $\sigma \approx$ tamaño.

5.3 Halftone CMYK (modo "halftone")

Conversión RGB → CMYK simplificada:

R' = R/255, G' = G/255, B' = B/255

K = 1 - max(R', G', B')

C = (1 - R' - K) / (1 - K) si K < 1, sino 0

M = (1 - G' - K) / (1 - K)

Y = (1 - B' - K) / (1 - K)

Tamaño de punto para cada canal:

size_C = C × baseSize

size_M = M × baseSize

size_Y = Y × baseSize

size_K = K × baseSize

Cada canal se desplaza ligeramente (roseta offset):

Cyan: (x, y)

Magenta: (x + step×0.22, y)

Amarillo:(x - step×0.22, y)

Negro: (x, y + step×0.1)

El patrón de roseta con offsets crea la textura característica de la impresión CMYK. Los ángulos de trama estándar en impresión son C=15°, M=75°, Y=0°, K=45°; aquí se aproximan con desplazamientos cartesianos para simplificar el cómputo.

5.4 RGB Shift — Chromatic Aberration

Offset dinámico basado en funciones trigonométricas:

offset_x = floor(sin(frameCount × 0.05) × 6 + 3)

offset_y = floor(cos(frameCount × 0.05) × 3)

Canal Rojo: pixel[(y + offset_y) × W + (x - offset_x)]

Canal Verde: pixel[y × W + x] (sin desplazar)

Canal Azul: pixel[(y - offset_y) × W + (x + offset_x)]

Las funciones seno y coseno desfasadas 90° producen un movimiento circular del desplazamiento cromático. El período es 2π/0.05 ≈ 125 frames ≈ 4 segundos a 30fps. La amplitud de 6px en X y 3px en Y crea una aberración más pronunciada horizontalmente, como ocurre en lentes reales donde la dispersión cromática es mayor en los bordes horizontales del encuadre.

5.5 Voronoi (modo "voronoi")

1. Generar N semillas aleatorias en el canvas: seeds[i] = (randomX, randomY)

2. Para cada píxel (px, py):

minDist = ∞, nearestSeed = -1

Para cada semilla s:

d = √((px - s.x)² + (py - s.y)²)

Si d < minDist: minDist = d, nearestSeed = s

color_pixel = color_promedio en la region de la semilla nearestSeed

Complejidad: O(W×H×N) por frame. Con N=50 semillas y canvas 960×540, son ~26 millones de operaciones por frame. En la práctica, p5.js puede manejar esto a 15-20 FPS en hardware moderno gracias a la optimización del motor V8 de JavaScript para operaciones numéricas.

5.6 Perlin Noise (modos "organico" y "flow_field")

Función de ruido Perlin 3D: noise(x, y, t) ∈ [0, 1]

Modo Orgánico (jitter de puntos):

nx = x + map(noise(x×0.012, y×0.012, t×0.004), 0,1, -jitter, +jitter)

ny = y + map(noise(x×0.012+100, y×0.012+100, t×0.004), 0,1, -jitter, +jitter)

Flow Field (dirección de partículas):

ángulo = noise(x×0.004, y×0.004, t×0.005) × TWO_PI × 2

vx = cos(ángulo) × velocidad

vy = sin(ángulo) × velocidad

Los offsets +100 en la segunda llamada a noise() garantizan que los desplazamientos X e Y sean independientes (muestras diferentes del campo de ruido). Las frecuencias espaciales bajas (0.004-0.012) producen ondulaciones suaves; frecuencias más altas producirían turbulencia caótica.

5.7 ASCII Art (modo "ascii_art")

Mapeo de brillo a carácter:

chars = " .:-=+#%@" (10 niveles, de oscuro a claro)*

idx = floor(map(brillo, 0→255, 0→9))

char = chars[idx]

Conversión equivalente: idx = floor((255 - brillo) / 255 × 9)

A brillo=255 (blanco): idx = 0 → " " (espacio). A brillo=0 (negro): idx = 9 → "@". La paleta de 10 caracteres fue seleccionada empíricamente para maximizar el contraste perceptual entre niveles adyacentes, siguiendo el estándar históricamente usado en arte ASCII desde los años 70.

5.8 Edge Detection — Kernel Sobel

Kernel Sobel horizontal (Gx): Kernel Sobel vertical (Gy):

[-1 0 +1] [-1 -2 -1]

[-2 0 +2] [0 0 0]

[-1 0 +1] [+1 +2 +1]

Magnitud del gradiente: $|G| = \sqrt{(Gx^2 + Gy^2)}$

Aproximación rápida: $|G| \approx |Gx| + |Gy|$

La aproximación |Gx| + |Gy| tiene un error máximo del ~11% respecto a la magnitud euclidiana verdadera, pero es significativamente más rápida (2 sumas + 2 valores absolutos vs. 2 multiplicaciones + 1 suma + 1 raíz cuadrada).

6. Métodos Web Utilizados

- p5.js 1.11.1: Framework creative coding: setup(), draw(), image(), loadPixels(), updatePixels(), map(), noise(), sin(), cos(), random(), TWO_PI, frameCount, createGraphics(). El bucle draw() se ejecuta automáticamente a ~60 FPS sincronizado con requestAnimationFrame.
- Canvas 2D API (via p5): p5.drawingContext accede al contexto nativo para operaciones avanzadas: createRadialGradient, addColorStop, fillRect con gradientes.
- getUserMedia: Acceso a cámara web con constraints {video:{width:{ideal:640}, height:{ideal:480}, facingMode}}. Selección de cámara frontal/trasera.
- HTMLVideoElement: Reproducción de video con playsInline, muted, loop. Eventos: loadstart, canplay, playing, waiting, stalled, ended, error. Control de playbackRate.
- Drag & Drop API: Eventos dragenter, dragover, dragleave, drop en el canvas wrapper. Acepta archivos de video (video/*).
- File API: Aceptación de archivos vía <input type="file"> y drag & drop. URL.createObjectURL para crear src de video.
- gif.js: Encoder GIF con algoritmo LZW. 2 Web Workers para codificación paralela. Configuración: quality=10, repeat=0 (infinito). Callback on('finished', blob).
- MediaRecorder: canvas.captureStream(30) → stream de 30 FPS. MediaRecorder con MIME video/webm;codecs=vp9. Eventos ondataavailable y onstop.
- localStorage: No utilizado en esta herramienta. Todo es volátil.

7. Modelo de RAM y Rendimiento

Canvas p5.js: 960×540 píxeles

Array de píxeles: 960 × 540 × 4 = 2,073,600 bytes ≈ 2 MB

Copias temporales (motion trails, feedback): 2-10 MB adicionales

Video en memoria: 5-50 MB dependiendo de la duración

Total típico: 10-60 MB

Rendimiento: los modos simples (clasico, neon, halftone, rgb_shift, pixelate) alcanzan 55-60 FPS. Los modos intensivos (voronoi, triangulacion, flow_field con muchas partículas) pueden bajar a 15-25 FPS. El cuello de botella es el bucle anidado de procesamiento píxel por píxel, que es inherentemente O(W×H).

8. Flujo de Datos Completo

[ENTRADA] Video archivo (Drag & Drop) o Cámara web (getUserMedia)
■
■■■ HTMLVideoElement (oculto, reproducción en bucle)
■
■■■ p5.js draw() loop (60 FPS):
■■■ p5.image(videoEl, 0, 0, CANVAS_W, CANVAS_H)
■■■ p5.loadPixels() → array de 2,073,600 bytes (RGBA)
■■■ Algoritmo del modo activo:
■ ■■■ renderClasico / renderOrganico / renderNeon
■ ■■■ renderHalftone / renderVivo / renderRGBShift
■ ■■■ renderPixelSort / renderScanlines
■ ■■■ renderTriangulacion / renderVoronoi
■ ■■■ renderFisheye / renderSwirl / renderRipple
■ ■■■ renderPixelate / renderLenticular
■ ■■■ renderAscii / renderEdgeDetect / renderEmboss
■ ■■■ renderMotionTrails / renderMotionMap
■ ■■■ renderAfterimage / renderTimeSlit / renderStrobeFreeze
■ ■■■ renderFeedback / renderHeatMap / renderPosterize
■ ■■■ renderSolarize / renderChannelDelay
■ ■■■ renderDisplacement / renderFlowField / renderPaintDrip
■ ■■■ renderBlockShift / renderWaveWarp / renderDotMatrix
■■■ p5.updatePixels() → canvas visible

[EXPORTACIÓN]
PNG: p5.saveCanvas('puntillismo-' + timestamp, 'png')
SVG: generar string SVG con <circle> por cada punto → Blob → download
GIF: gifRecorder.addFrame(canvas, {delay:...}) × 90 frames → render()
Video: canvas.captureStream(30) → MediaRecorder → WebM blob

9. Conclusiones

Free Animation Glitch es un laboratorio de procesamiento visual que demuestra cómo algoritmos clásicos de computación gráfica — diagramas de Voronoi, ruido Perlin, separación CMYK, detección de bordes Sobel — pueden combinarse para crear efectos artísticos en tiempo real. La

herramienta funciona como un puente entre el arte generativo y la producción de video, permitiendo a creadores sin conocimientos de programación aplicar transformaciones visuales complejas a su material audiovisual.

La decisión de usar p5.js como motor gráfico fue estratégica: su API de alto nivel (image(), loadPixels(), map(), noise()) reduce drásticamente el código boilerplate comparado con Canvas 2D nativo, permitiendo implementar 37 efectos en un solo sketch. El costo es una pequeña sobrecarga de rendimiento (~5-10% vs Canvas nativo) que es aceptable dado que la mayoría de los efectos ya son computacionalmente intensivos.

El diseño de la arquitectura de estado (objeto cfg + variables de sketch) permite que los 37 modos compartan infraestructura de entrada/salida (video, cámara, exportación) mientras mantienen sus algoritmos de renderizado completamente independientes. Es un ejemplo de patrón Strategy aplicado a creative coding: cada modo es una estrategia de renderizado intercambiable en tiempo de ejecución.

— Fin del Documento 05 — Free Animation Glitch —