¿Puede una obra sobrevivir a los medios?

Distribuciones: estrategias visuales para un arte óptico interactivo

Emiliano Causa

1. Introducción

La obra interactiva *Distribuciones*, desarrollada por mí al interior del colectivo Biopus (biopus.ar), fue creada específicamente para acompañar una edición digital cuya temática central es la distribución. El punto de partida del proyecto fue, precisamente, la reflexión sobre esta palabra: "distribución" no sólo como tema, sino como principio estructurante de una experiencia visual e interactiva. A partir de esta noción, se diseñó un sistema visual que pone en tensión dos comportamientos opuestos: la distribución de formas en el espacio como mecanismo de orden y apertura, y la concentración como estado de repliegue, desorganización y ausencia de actividad.

Desde el punto de vista formal, *Distribuciones* es una obra de arte óptico interactivo construida a partir de composiciones sobre grillas modulares. Utiliza figuras geométricas simples —principalmente cuadrados y círculos— organizadas en tres capas visuales que se superponen. Las variaciones de posición, rotación y desplazamiento entre estas capas generan efectos de interferencia visual que remiten a las estrategias del arte óptico clásico. Estas interferencias se intensifican con el movimiento y la interacción, generando composiciones que no son estáticas ni previsibles, sino que se transforman en tiempo real en función de las acciones del usuario.

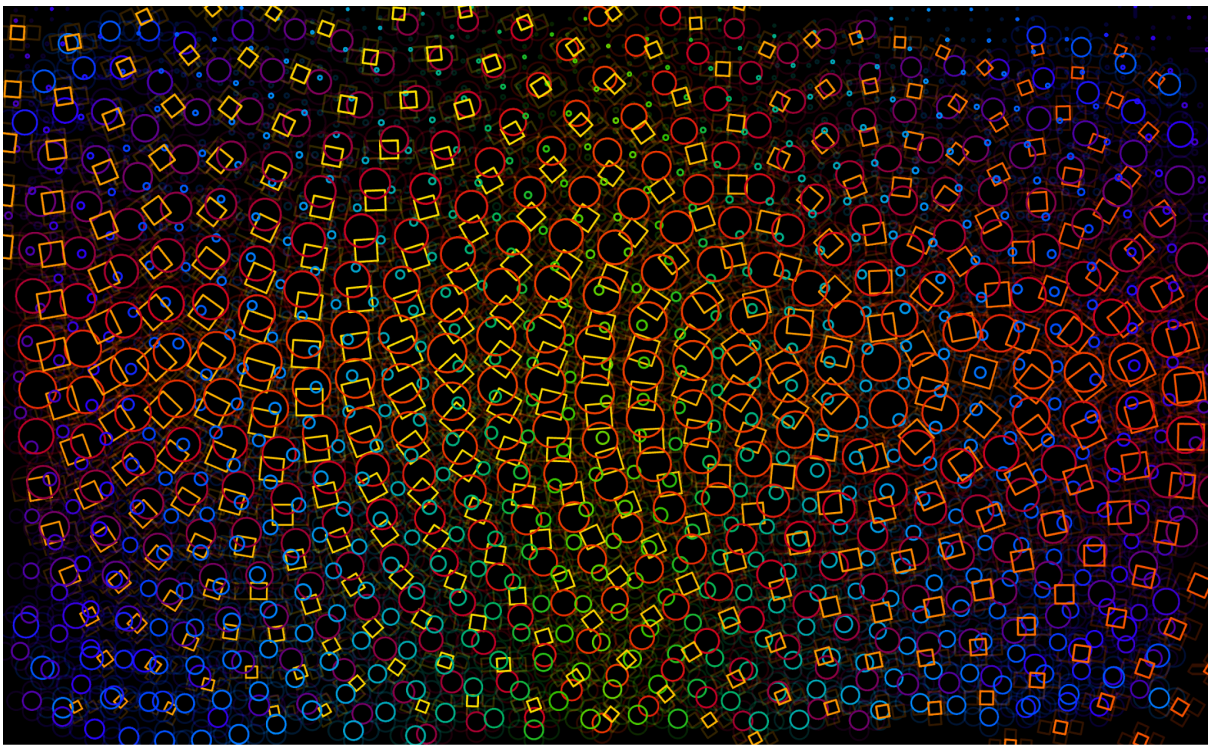


Figura 1 - *Distribuciones* #0

La obra está compuesta por una serie de variaciones generativas. Cada variación produce composiciones distintas a partir de parámetros que se definen parcialmente al azar, tales como la cantidad de elementos, las direcciones de rotación, el uso del color o las formas de desplazamiento. En cada caso, ciertos elementos permanecen fijos, mientras que otros se modifican, permitiendo explorar múltiples configuraciones dentro de un mismo marco estructural.

El comportamiento de la obra se activa a través de la interacción del usuario, ya sea con el cursor o mediante gestos táctiles. Sin intervención, las formas geométricas tienden a agruparse hacia el centro o los bordes de la pantalla; al ser activadas, se reorganizan en estructuras visuales en movimiento.

Uno de los objetivos principales del desarrollo fue asegurar que la obra pudiera ejecutarse en distintos tipos de dispositivos, desde computadoras de alta performance hasta teléfonos móviles con capacidades limitadas. Para lograr esto, se implementaron dos tipos de estrategias visuales diferenciadas. En dispositivos con alto rendimiento gráfico, la obra presenta composiciones completas con transiciones suaves y movimientos sutiles. En cambio, en dispositivos de baja performance, se emplean técnicas de dibujo parcial y simplificación progresiva, donde las formas se construyen en etapas a través del movimiento, manteniendo así la idea de distribución como dinámica central, aunque con otros recursos técnicos. Esta adaptación no busca replicar exactamente la misma experiencia, sino conservar la lógica compositiva y conceptual que define a la obra en sus distintas versiones.

2. Composición visual: grillas y formas espirales

La estructura visual de *Distribuciones* se basa en dos sistemas principales de generación de puntos: las grillas regulares y las formas circulares organizadas en espirales. Ambas estrategias fueron desarrolladas mediante funciones escritas en JavaScript, utilizando la biblioteca p5.js como entorno de trabajo. Estas funciones permiten crear configuraciones geométricas a partir de parámetros controlables como el espaciado, el ángulo de rotación y la distancia entre elementos.

Por un lado, la función `generateGridPoints(D)` genera una cuadrícula regular de puntos distribuidos equidistantemente en la pantalla, con un centrado que asegura un margen homogéneo. Esta grilla sirve como base compositiva para la colocación de figuras simples, permitiendo una organización espacial ordenada y simétrica.

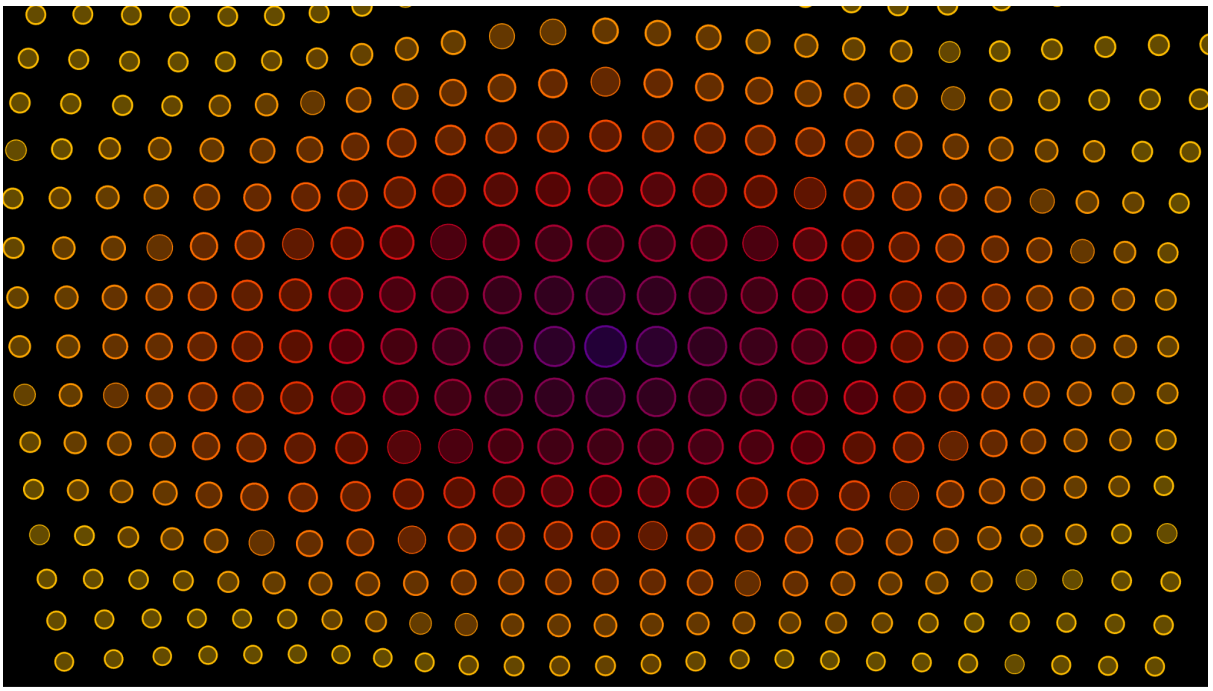


Figura 2 - Composición en forma de Grilla

Por otro lado, se diseñaron dos clases (`ArcPoints` y `Spiral`) que permiten generar puntos a lo largo de trayectorias curvas. La clase `ArcPoints` calcula puntos equidistantes sobre un arco de circunferencia a partir de un centro, un radio, un ángulo de inicio y otro de fin. La clase `Spiral`, construida sobre la anterior, concatena arcos alternados para generar trayectorias espirales expandibles que se desplazan gradualmente en el espacio. Estas funciones fueron desarrolladas en parte con la colaboración de ChatGPT, lo que facilitó la resolución de problemas algorítmicos específicos y aceleró el diseño estructural del código.

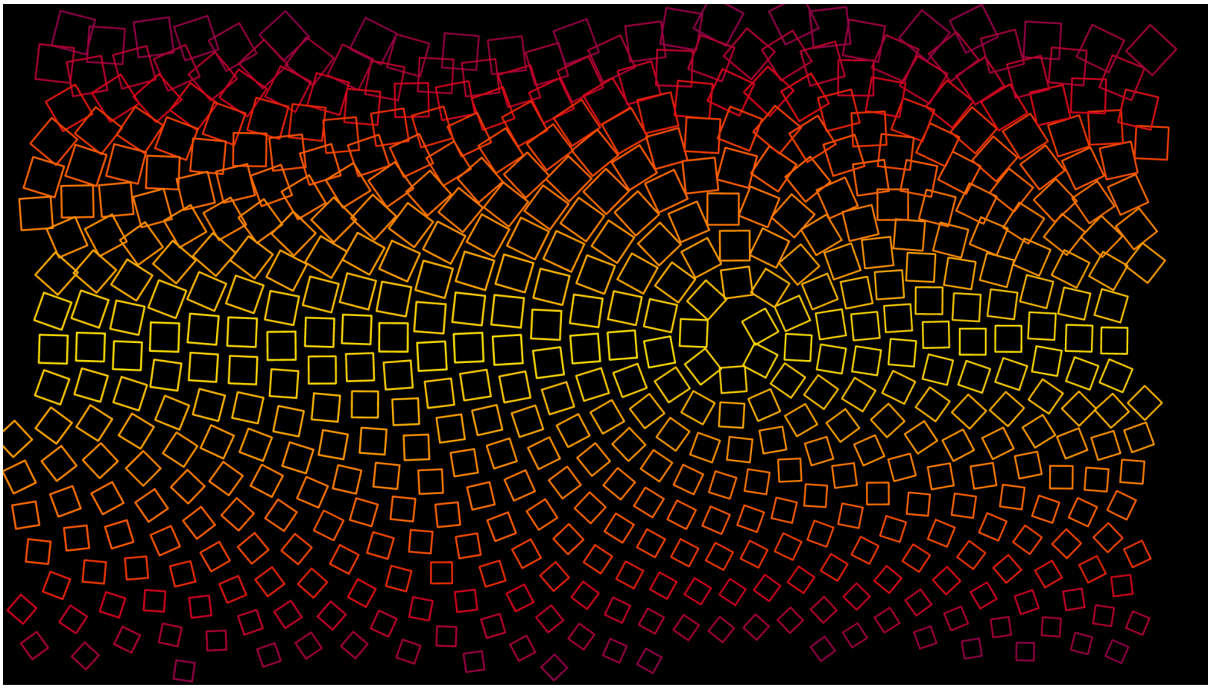


Figura 3 - Composición en forma de Espiral

La obra utiliza estas trayectorias para ubicar figuras geométricas (principalmente cuadrados) con una orientación determinada por el ángulo de rotación de cada punto en la espiral. En particular, se implementó una lógica que valida si cada nueva figura puede ser colocada sin superposición con las anteriores, utilizando un buffer auxiliar (`PGraphics`) para simular la ocupación espacial y evitar colisiones visuales. De este modo, se logra una distribución densa pero no caótica, donde las formas mantienen una cierta autonomía espacial. A esta composición se la denominó de “muchos espirales”.

La estrategia de las espirales tiene, además, una variante optimizada para dispositivos de menor rendimiento, en la que se dibujan solo los puntos visibles dentro de la pantalla (`generateCompositionFullSpiral`). Esta decisión permite mantener la fluidez del movimiento y la riqueza compositiva sin comprometer la performance general del sistema.

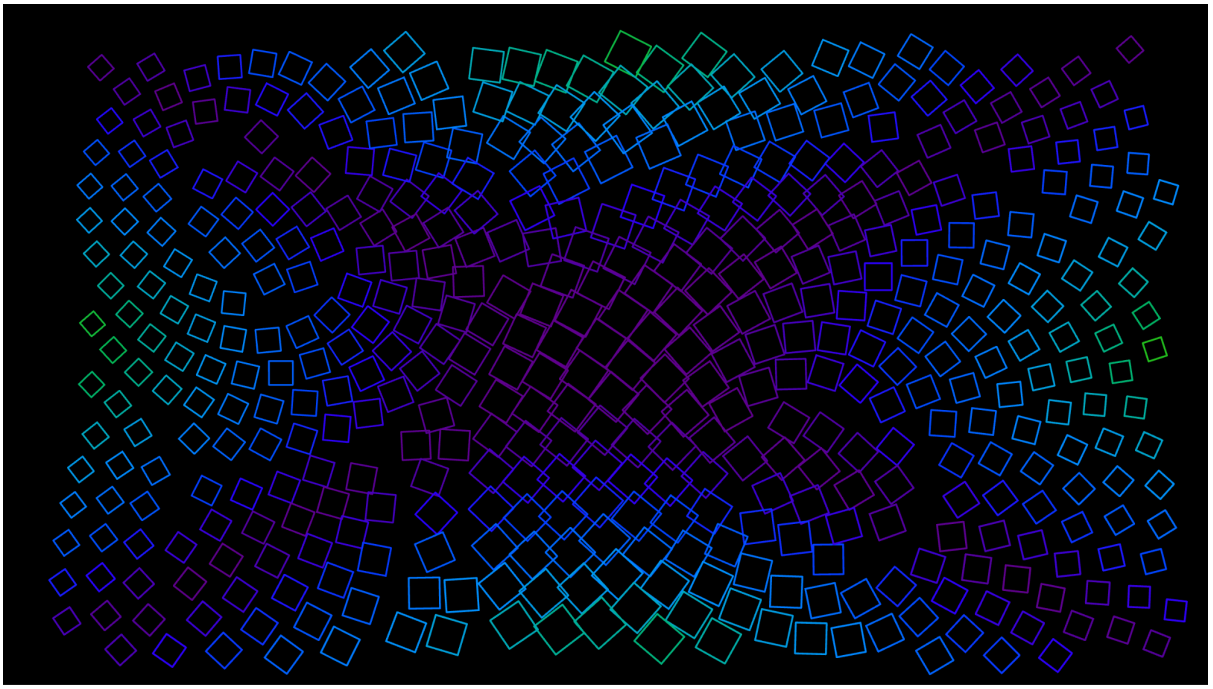


Figura 4 - Composición en forma de Muchos Espirales

3. Composición visual: modulación de formas

Otra sección fundamental del sistema visual es el uso de capas de modulación (Clase `Capa`) que controlan la apariencia de las figuras que se dibujan sobre las grillas o a lo largo de las espirales. Cada instancia de `Capa` afecta atributos como el color, el tamaño y la rotación de las formas geométricas. Lo interesante es que estas modulaciones no son aleatorias ni estáticas, sino que responden a patrones definidos por mapas de gradientes de base, que operan como máscaras de control.

Estos gradientes —también llamados en el código simplemente *gradientes* o instancias de `Evolucion`— pueden tener forma de cruz, radial o lineal (de lado a lado), y son utilizados para leer valores interpolados que modifican dinámicamente cada parámetro visual. Cada

capa mantiene internamente tres `Evolucion`: una para color, otra para tamaño, y otra para rotación. Estas clases leen píxeles de una imagen (`this.esteGradiente`) que actúa como mapa de control. A partir del valor del canal rojo del píxel correspondiente a la posición normalizada `(x, y)` en pantalla, se obtiene un número entre 0 y 1 (posiblemente invertido aleatoriamente), que luego se mapea según las necesidades de modulación.

Por ejemplo:

- El color se interpola entre dos extremos (`color1` y `color2`) definidos en el espacio de color de la paleta general, usando la función `darColor2`.
- El tamaño de cada figura se define por interpolación entre `minTam` y `maxTam`.
- La rotación se modula entre 0 y PI, generando así variaciones armónicas y suaves.

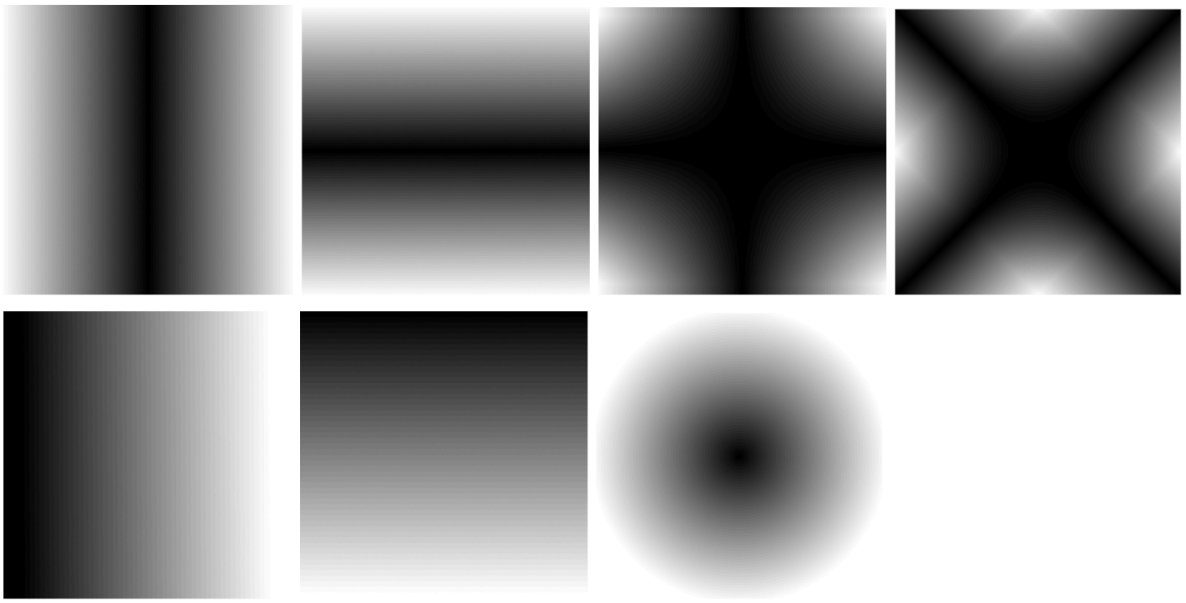


Figura 5 - Tipos de gradientes

El sistema también contempla diferentes modos de dibujo (`P_RELLENO`, `P_CONTORNO_GRUESO`, etc.) y tipos de figuras (`F_ARCOS`, `F_CRUCES`, etc.), haciendo que una misma capa pueda tener múltiples expresiones visuales.

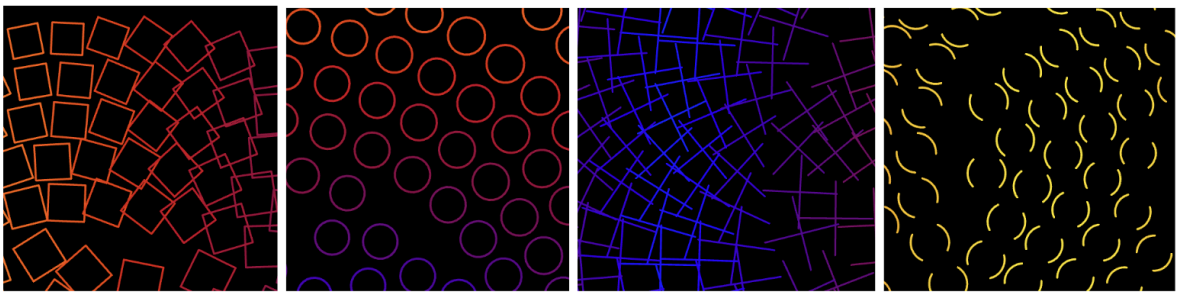


Figura 6 - Tipos de figuras: cuadrados, círculos, cruces, arcos.

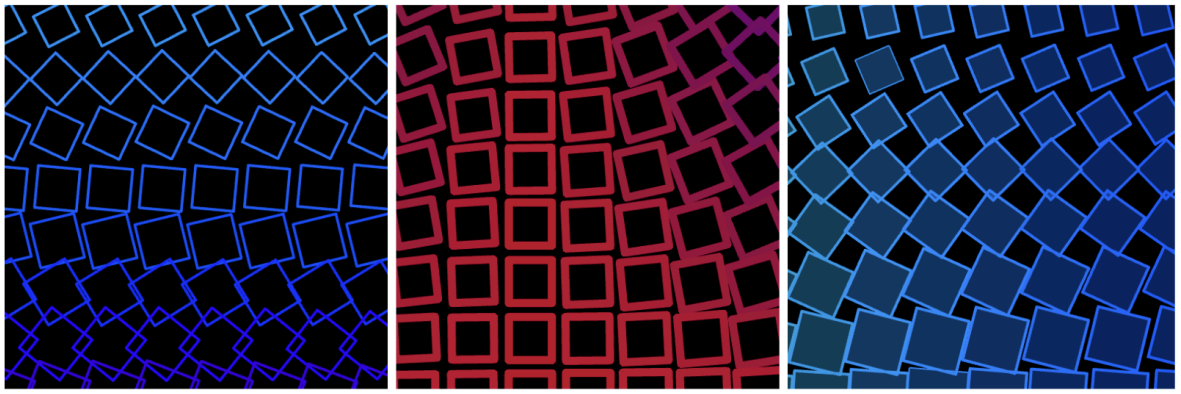


Figura 7 - Modos de dibujo: contorno, contorno grueso, relleno.

El resultado es una modulación compleja pero controlada, donde las figuras que se repiten en la grilla o en el trazo espiral tienen un comportamiento coherente, visualmente vibrante, y profundamente influido por la estructura de los gradientes base.

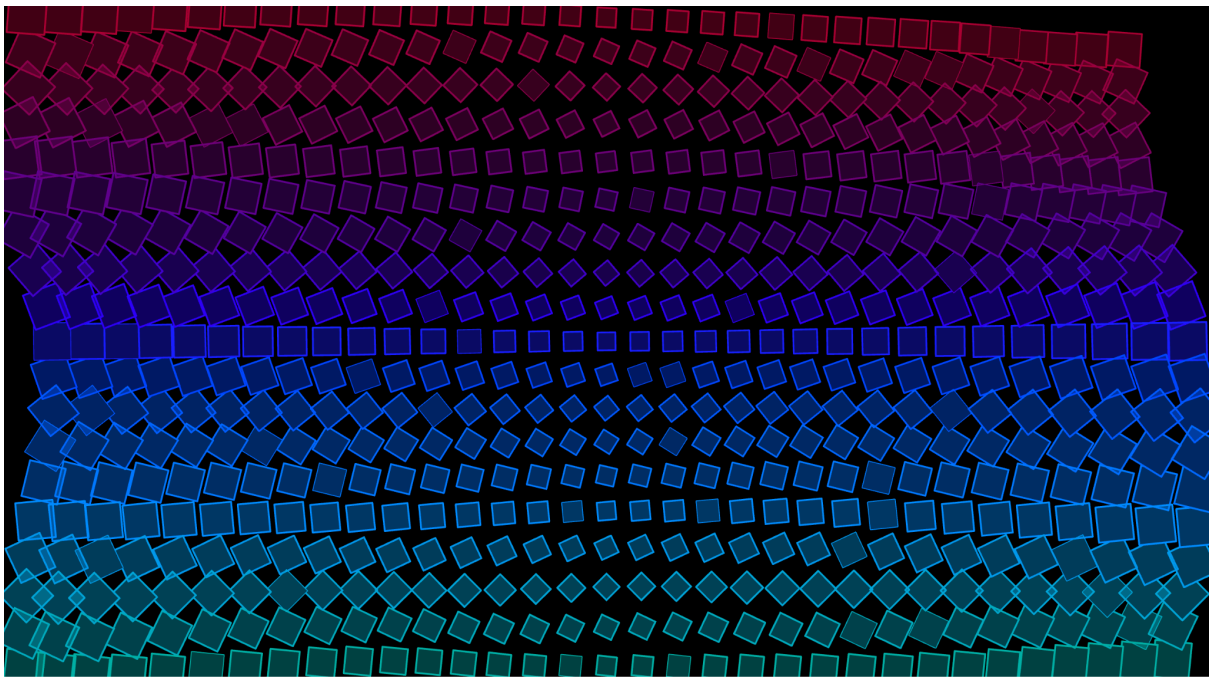


Figura 8 - Muestra de una capa

En la figura 8 se presenta un ejemplo de capa donde se pueden observar claramente las tres dimensiones principales de variación. El color muestra una transición vertical suave, definida por un gradiente lineal que recorre la figura de arriba hacia abajo. El tamaño, en cambio, responde a un gradiente horizontal de tipo parabólico: crece hacia el centro y decrece hacia los bordes, alcanzando su valor máximo en la zona media. Por último, la rotación está modulada por un gradiente en cruz, es decir, una combinación de variaciones verticales y horizontales que genera una distribución más compleja y rica en desplazamientos angulares.

El resultado es una modulación compleja pero controlada, donde las figuras que se repiten en la grilla o en el trazo espiral tienen un comportamiento coherente, visualmente vibrante, y profundamente influido por la estructura de los gradientes base.

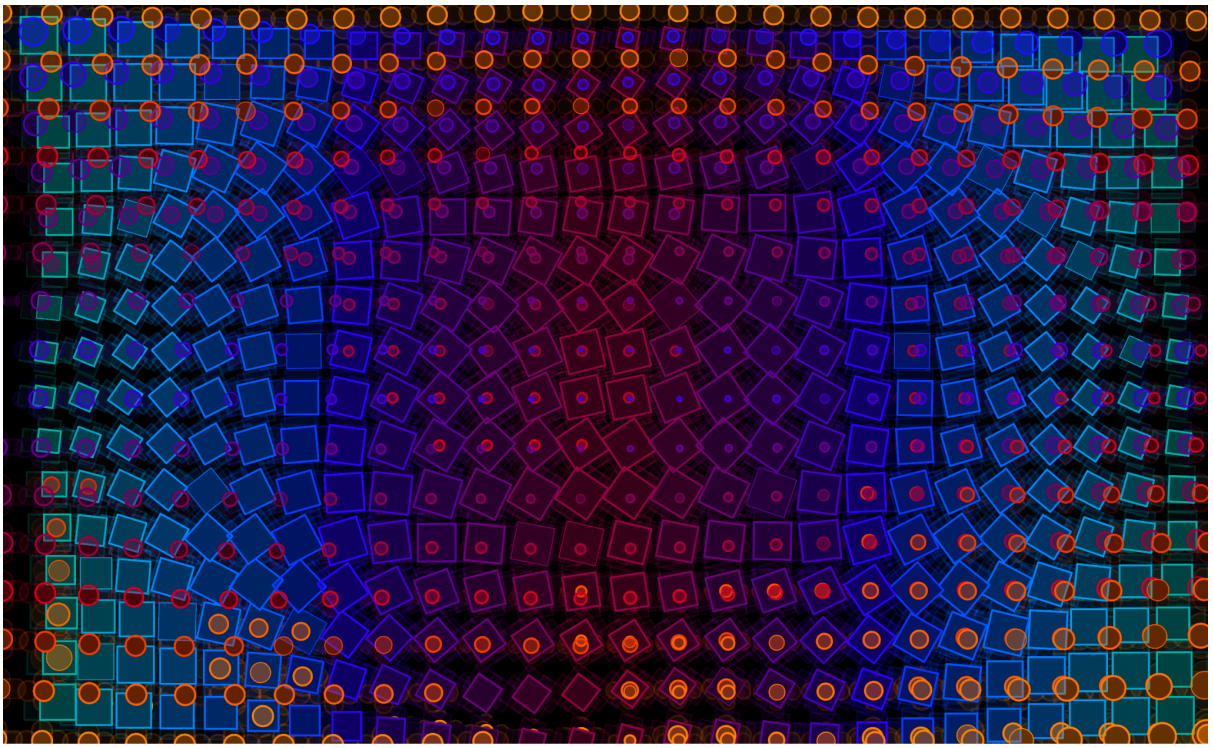


Figura 9 - Muestra de composición compuesta por 3 capas

La figura 9 muestra una composición generada a partir de la articulación de tres capas superpuestas: dos de ellas construidas con figuras circulares y una tercera basada en cuadrados. Cada capa opera con sus propios gradientes de color, tamaño y rotación, y se organiza según diferentes disposiciones espaciales. La interferencia visual resultante —producto del entrelazamiento entre estas estructuras independientes— da lugar a un efecto característico del arte óptico, donde las formas parecen vibrar, desfasarse o generar pulsaciones. Este efecto, que es central en la obra *Distribuciones*, no emerge de una figura aislada sino de la relación dinámica entre múltiples sistemas visuales en convivencia. La composición final es el resultado de esa tensión armónica entre capas, y su riqueza perceptual radica precisamente en esa superposición cuidadosamente orquestada.

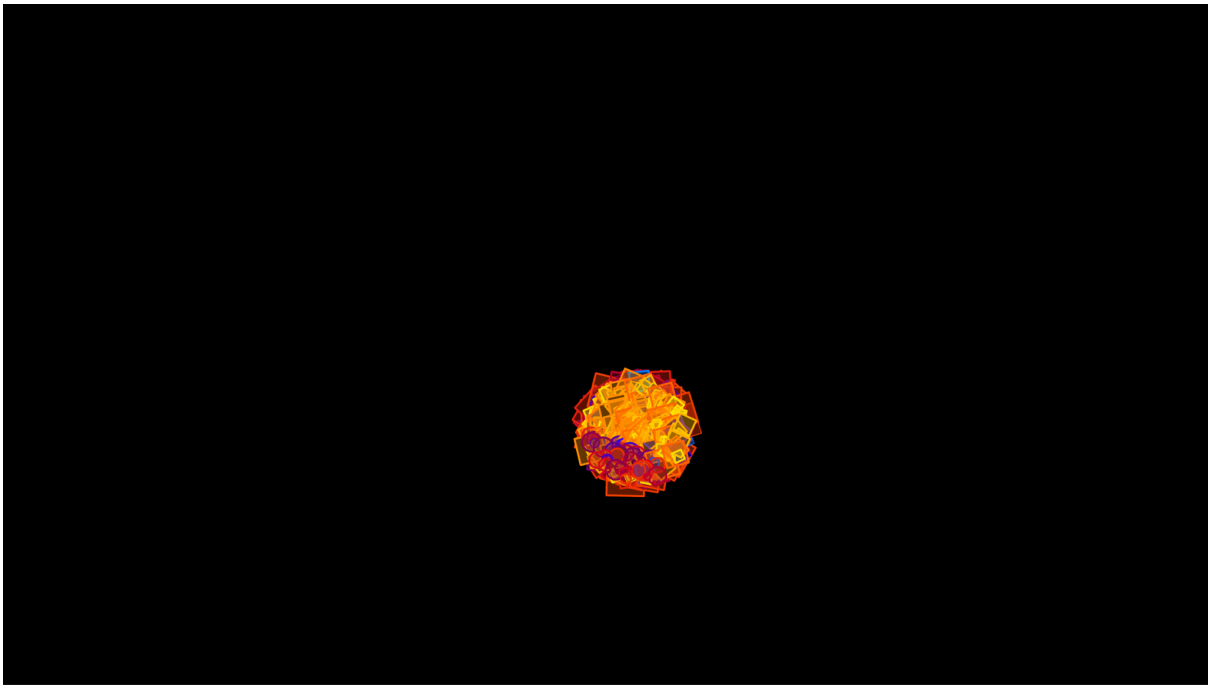
4. El comportamiento y el movimiento

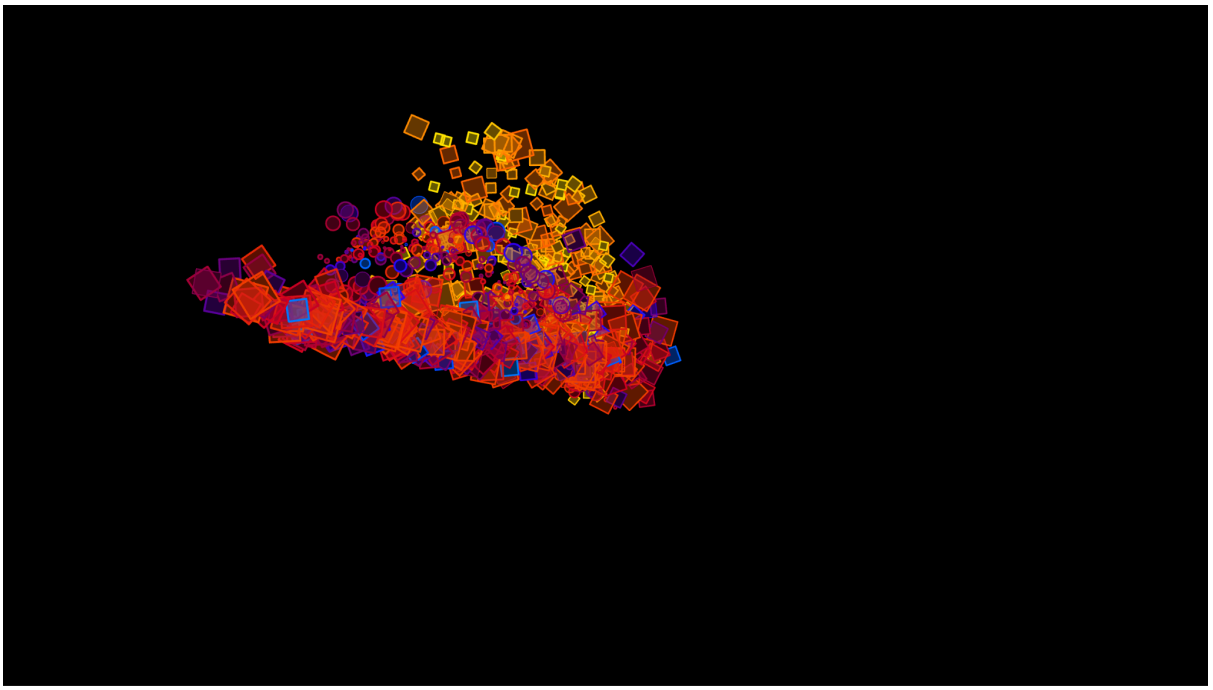
La clase `Pintura` es la responsable de generar la composición final de la obra de arte óptico. No se trata de un concepto abstracto, sino de una clase concreta que organiza y coordina los distintos elementos del sistema. Al iniciar una nueva composición, `Pintura` le solicita a una función de composición (de las que ya describimos anteriormente) un conjunto de puntos. Estos puntos representan la estructura espacial de la obra, y cada uno de ellos se convierte en el destino de un caminante.

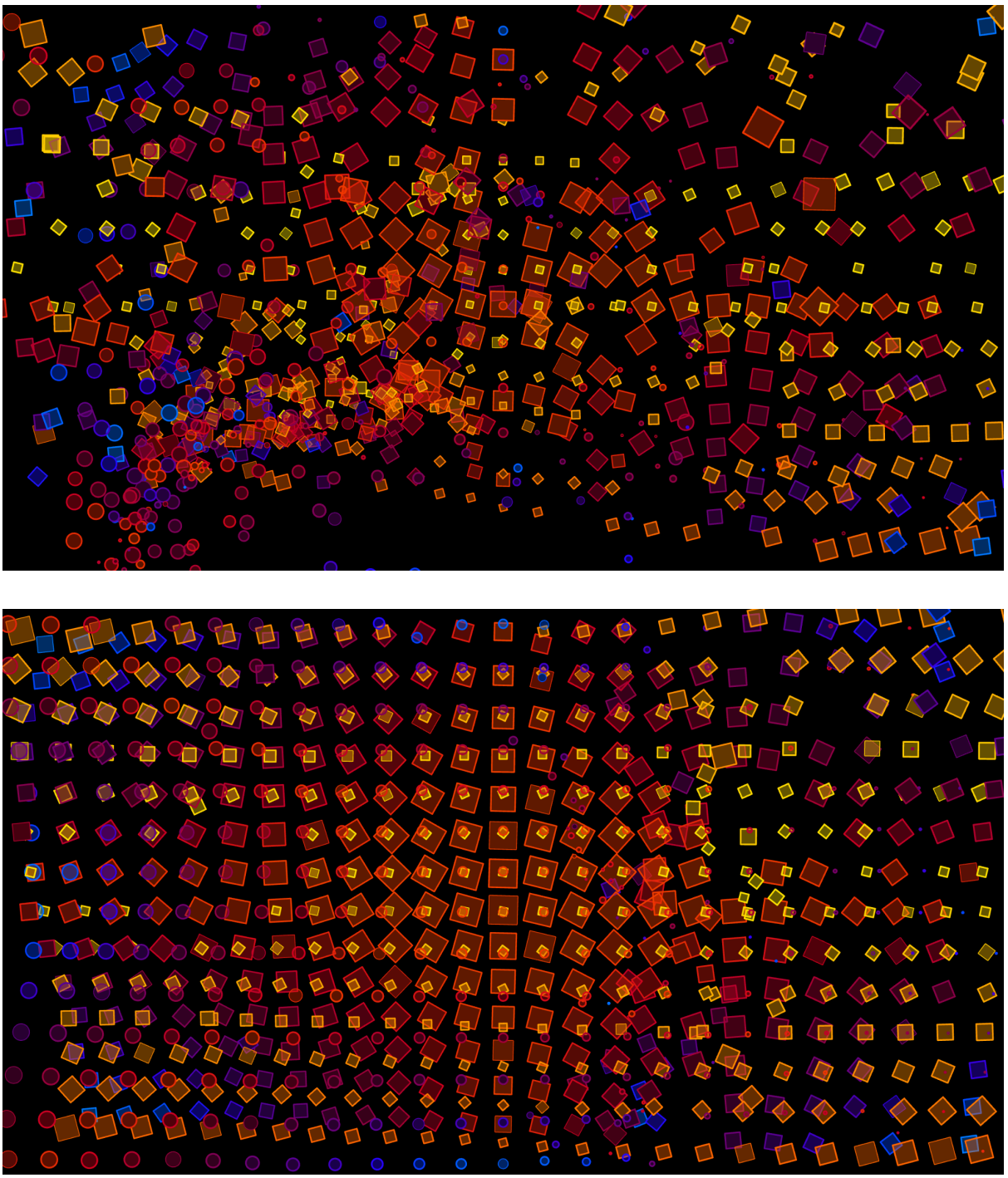
Cada caminante, al recibir su destino, se asocia además con una capa, que le proporciona dos elementos fundamentales: una figura con la que va a pintar y una serie de gradientes que definen cómo va a evolucionar en el tiempo su tamaño, su rotación y su color. Así, cada caminante sabe exactamente qué rol ocupa dentro de la composición: en qué posición debe estar, con qué forma debe pintar, y cómo deben transformarse sus atributos visuales a medida que avanza el tiempo.

Sin embargo, si no hay interacción por parte del usuario, los caminantes tienden a concentrarse en una misma región del espacio, desarmando la composición inicial. Es un comportamiento deliberado que genera una cierta tensión en la imagen, una deriva visual. Pero cuando el usuario hace clic sobre la obra, todos los caminantes vuelven a acomodarse en el lugar que les corresponde dentro de la composición, restaurando el orden visual. En ese instante, la imagen reaparece con claridad.

La obra transita, entonces, entre dos estados: uno de **concentración**, donde los caminantes se agrupan y la composición se disuelve, y otro de **distribución**, donde cada caminante ocupa su posición asignada y la estructura se hace visible. Este segundo estado, además, tiene una dimensión interactiva: al detectar la presencia del usuario mediante un clic, no solo se reactiva la distribución compositiva, sino que también se desencadena un movimiento tridimensional en profundidad (eje *Z*).







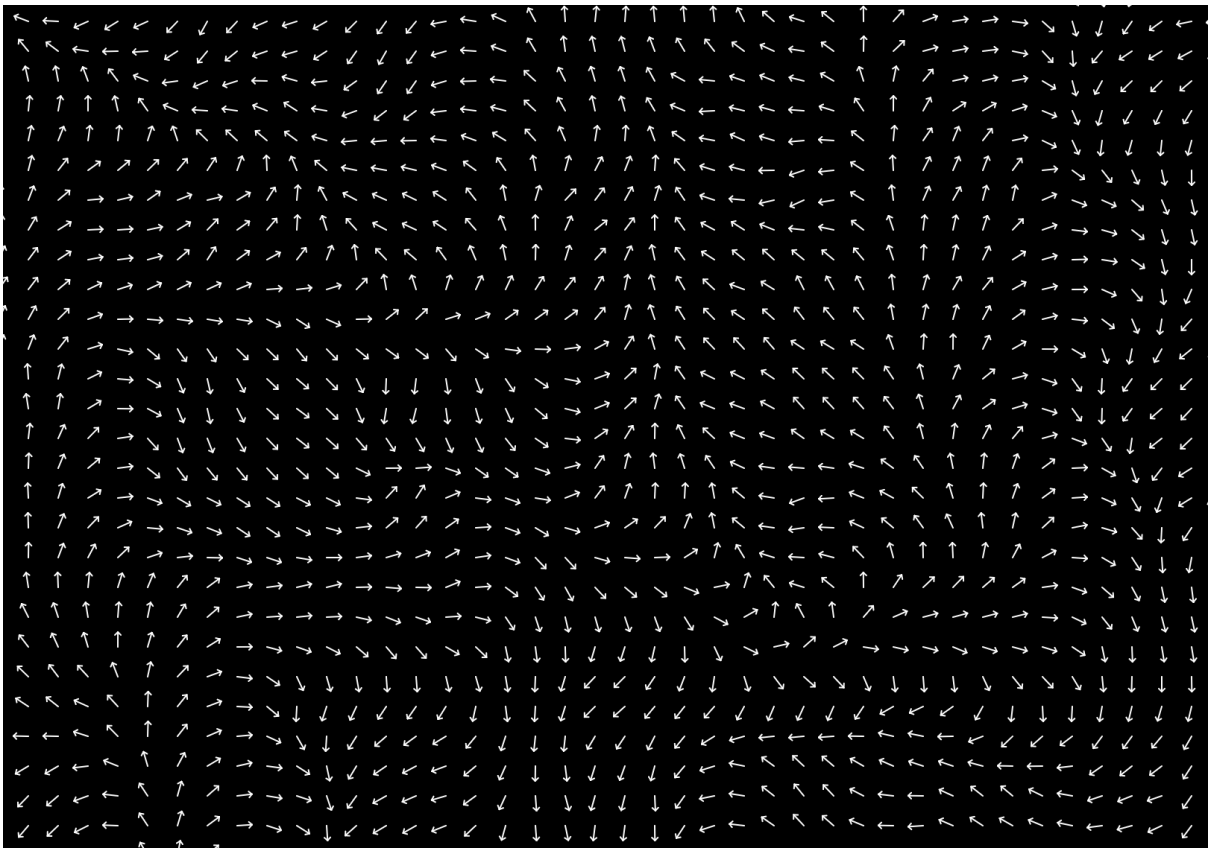
Figuras 10/11/12/13 - Transición que muestra los caminantes yendo del estado de Concentración hacia la Distribución

Este desplazamiento en *Z* no es instantáneo ni arbitrario, sino que está resuelto mediante una simulación física simple. Cada caminante cuenta con una posición en *Z*, una velocidad y una aceleración, y cuando se activa el clic, se le asigna un destino (por lo general, *z = 0*). A partir de ese momento, cada caminante se mueve hacia ese punto como si estuviera conectado por un resorte invisible: se calcula una aceleración proporcional a la distancia, se acumula velocidad, y se simula fricción para suavizar el movimiento. Este tipo de respuesta

genera una oscilación breve, un pulso tridimensional, como si la imagen entera respirara ante el gesto del espectador.

Un aspecto fundamental del comportamiento visual de los caminantes en *Distribuciones* es la forma en que se desplazan durante las transiciones entre los estados de **concentración** y **distribución**. Estos desplazamientos no siguen trayectorias lineales o deterministas, sino que están mediados por una lógica de desvío controlado a través de ruido Perlin (*Perlin noise*), el cual opera como un campo vectorial fluido que modifica la dirección de movimiento en cada fotograma.

Este mecanismo se articula principalmente en el método `mover()` de la clase `Caminante`. Allí, en lugar de orientar directamente al caminante hacia su destino, se calcula primero una dirección ideal —una línea recta hacia el punto de llegada— y luego se mezcla con un ángulo de desvío obtenido a partir del ruido Perlin (*Perlin noise*) en función de la posición del caminante. En concreto, el ruido se consulta a partir de sus coordenadas `x` e `y`, pero a estas se les suma una tercera dimensión —el paso del tiempo— que actúa como desplazamiento progresivo en el eje del ruido. Esta dimensión temporal se construye mediante un *offset* que se actualiza cuadro a cuadro y que permite que el “territorio de direcciones” generado por el Perlin Noise no sea estático, sino que evolucione suavemente a lo largo del tiempo.



Figuras 14 - Ejemplo de campo vectorial producido con Perlin noise

Gracias a esta tercera dimensión, las direcciones posibles que puede adoptar cada caminante se transforman constantemente, lo que evita repeticiones y patrones previsibles.

El resultado son trayectorias orgánicas, dinámicas y fluidas, donde cada caminante parece responder a condiciones del entorno que cambian lentamente, como si se desplazara dentro de un campo de fuerzas viviente. Este tipo de movimiento se asemeja al fluir del agua, al comportamiento de cardúmenes o de bandadas de aves, y contribuye a esa sensación de vida autónoma que caracteriza la pieza.

Como se observa en la figura 14, el Perlin Noise puede ser representado como un campo vectorial distribuido sobre una grilla. Cada celda de esta grilla contiene una flecha cuya orientación es el resultado de consultar el Perlin Noise en esa posición específica del espacio. Si un caminante pasara por alguno de estos cuadros, tomaría la dirección indicada por la flecha correspondiente, adoptando así un comportamiento guiado por este campo vectorial suave y continuo. Este mecanismo es el que permite que las trayectorias no sean rectas ni previsibles, sino que se desplacen con una fluidez orgánica, como si respondieran a corrientes invisibles que recorren el espacio.

```
let nueva = atan2( this.destino.y-this.posi.y , this.destino.x-this.posi.x );

let toff = (frameCount + this.desplazaNoise) * 0.001;
let xoff = toff+map( this.posi.x , 0 , width , 0.0 , 1.0 );
let yoff = toff+map( this.posi.y , 0 , height , 0.0 , 1.0 );

let desvio = map( noise(xoff,yoff) , 0.2 , 0.8 , 0 , PI*4)

this.dir = desvio * f + nueva * (1-f);
```

Figuras 15 - Extracto del método mover() del caminante

Lo que se ve en la figura 15 es un extracto del código donde se define la lógica mediante la cual los caminantes determinan su dirección de movimiento. Esta lógica no responde a un cálculo recto y previsible, sino a una combinación entre una dirección directa al destino y una desviación sugerida por un campo vectorial generado con Perlin Noise. En primer lugar, se calcula el ángulo que une la posición actual del caminante con su punto de destino, utilizando `atan2`, que entrega la dirección geométrica más directa. Paralelamente, se calcula un desplazamiento temporal a partir del `frameCount` y un valor de ruido individual (`desplazaNoise`) que permite consultar el Perlin Noise no solo en función del espacio (las coordenadas X e Y del caminante), sino también en función del tiempo. Esta tercera dimensión temporal hace que el "territorio" de direcciones del Perlin Noise se transforme suavemente a lo largo del tiempo, generando un campo vectorial que evoluciona constantemente. Luego, con ese valor de ruido se obtiene un ángulo de desviación que es interpolado con la dirección directa al destino. El parámetro `f` controla cuánto pesa esa desviación en relación con la dirección original, permitiendo que al principio del recorrido el caminante esté más influido por el comportamiento fluido y orgánico del campo de ruido, y que gradualmente se oriente más hacia su destino. Esta combinación genera trayectorias que no son completamente aleatorias ni enteramente deterministas, sino que evocan comportamientos más naturales y complejos, como los que observamos en fenómenos colectivos del mundo natural: un cardumen, una bandada o un sistema de partículas en flujo.

Por su parte, el método `actualizar()` introduce el otro gran componente del movimiento: la atracción hacia el destino y la interacción con el usuario. Allí, la figura acumula velocidad a partir de una aceleración que responde a la diferencia de profundidad (componente Z) entre su posición y el destino. Esta lógica permite que el movimiento tenga cierto "peso", como si los caminantes tuvieran inercia. Además, cuando el usuario presiona el mouse, se produce un impulso adicional sobre el eje Z que empuja a los caminantes, generando una respuesta interactiva directa, casi táctil.

```
if( mouseIsPressed ){
    let distancia = dist( mouseX , mouseY , this.posi.x , this.posi.y );
    if( distancia < ALCANCE_MOUSE * random(1) ){
        this.aceleracion.z = this.empuje;
    }
}

this.velocidad.add( this.aceleracion );
this.aceleracion.set(0,0,0);

let vectorAtraccion = createVector( 0 , 0 , (this.destino.z-this.posi.z) * this.atraccion);

let friccion = 0.9;

this.velocidad.add( vectorAtraccion );
this.velocidad.mult( friccion );

this.posi.add( this.velocidad );
```

Figuras 16 - Extracto del método actualizar() del caminante

En este fragmento de código, que pertenece al método `actualizar`, se define el comportamiento dinámico del caminante en relación con su movimiento vertical, especialmente en torno a su eje Z. Esta dimensión cobra protagonismo cuando el usuario interactúa con el sistema mediante el mouse: si el botón está presionado y el caminante se encuentra dentro de un cierto rango de proximidad al cursor (definido por `ALCANCE_MOUSE` multiplicado por un valor aleatorio para introducir variabilidad), se le aplica una fuerza de empuje en el eje Z. Esta fuerza se suma a la aceleración del caminante, generando un efecto de “despertar” o sobresalto vertical.

Luego, esa aceleración se incorpora al vector de velocidad, que es el que determina finalmente el desplazamiento. Inmediatamente después se reinicia la aceleración a cero para evitar acumulaciones, como sucede comúnmente en simulaciones físicas por pasos discretos. A continuación, se introduce un vector de atracción también en el eje Z, que actúa como una fuerza restauradora hacia el valor deseado en el destino, con una intensidad regulada por el parámetro `atraccion`. Esta fuerza tiende a devolver al caminante a su posición de equilibrio luego de ser perturbado, generando una oscilación que recuerda a un sistema físico con amortiguación.

Para suavizar este movimiento y evitar comportamientos erráticos o descontrolados, se aplica un factor de fricción que reduce ligeramente la velocidad en cada ciclo. Finalmente, esa velocidad (ya modificada por la fricción y la atracción) se suma a la posición actual,

actualizando la ubicación del caminante en el espacio tridimensional. En conjunto, este sistema permite que los caminantes tengan una respuesta sensible y dinámica a las acciones del usuario, mientras conservan un comportamiento controlado y orgánico, enriqueciendo la expresividad del sistema. Estos algoritmos logran una animación rica en matices, donde las figuras nunca se mueven de forma idéntica y en la que lo determinista convive con lo azaroso.

5. Primeras pruebas y fracasos

En las primeras pruebas que realicé mostrando la aplicación desde un servidor en línea, me encontré con un problema inesperado: mientras que en computadoras, especialmente notebooks de buena performance, la obra corría sin inconvenientes, en teléfonos celulares —incluso en modelos de alta gama— el comportamiento era desastroso. La fluidez se perdía, los cuadros por segundo caían drásticamente, y la experiencia estética de la obra se veía completamente comprometida.

La primera estrategia que ensayé para resolver esto fue eliminar un shader que había implementado para simular profundidad de campo. Ese shader tenía como objetivo hacer que las figuras, al alejarse en el eje *Z*, perdieran foco, produciendo un desenfoque suave que reforzaba la sensación de espacio. Era un recurso visual potente, pero computacionalmente exigente. Decidí entonces retirarlo temporalmente, con la esperanza de que la obra pudiera sostenerse gráficamente sin esa capa de complejidad.

Además, tomé otra medida: reducir la resolución de las grillas que componen la obra. Esto se traduce en una disminución en la cantidad de filas y columnas, es decir, en la cantidad de elementos que deben ser procesados y renderizados por la máquina. Cuantos menos caminantes haya en pantalla, menor será la carga de trabajo para el sistema gráfico. Esta lógica funcionó parcialmente, pero trajo consigo un nuevo problema.

La aplicación estaba diseñada para adaptarse dinámicamente al rendimiento del dispositivo, midiendo el `deltaTime` entre frames y ajustando en tiempo real la densidad de la grilla con el objetivo de mantener un mínimo de 20 cuadros por segundo. Pero en dispositivos con muy baja performance —como algunas computadoras viejas o teléfonos particularmente limitados— el sistema comenzaba a bajar tanto la resolución que la composición visual de la obra se desdibujaba por completo. Lo que debía ser una adaptación elegante se transformaba en un colapso visual: la grilla quedaba tan vacía que la estructura compositiva desaparecía, dejando solo rastros dispersos que no alcanzaban a sostener la imagen.

Esta situación me obligó a repensar el balance entre optimización técnica y fidelidad visual. ¿Hasta qué punto vale la pena sostener una experiencia en dispositivos que no pueden reproducirla adecuadamente? ¿Conviene simplificar la obra para que funcione en todos los dispositivos, o mantener su densidad estética aún si eso implica excluir ciertos entornos? Estas preguntas siguen abiertas, pero la experiencia dejó en claro que la adaptabilidad, si no está cuidadosamente regulada, puede atentar contra la integridad de la obra misma.

6. Otras estrategias

El fracaso de la estrategia basada en disminuir la resolución de las grillas en dispositivos de baja performance me obligó a repensar el enfoque de optimización. Reducir la cantidad total de elementos en pantalla afectaba de forma directa la estructura de la composición, al punto de desintegrarla visualmente. No podía permitirme seguir por ese camino. La nueva estrategia debía permitir conservar la cantidad de elementos necesarios para sostener la composición, pero sin comprometer la fluidez de la experiencia.

La solución apareció en una idea simple pero efectiva: alternar la frecuencia con la que se actualiza y se dibuja cada uno de los elementos. En lugar de intentar actualizar todos los caminantes en cada fotograma, implementé un sistema de turnos. Para eso introduje una variable llamada `UNO_DE_CADA`, que funciona como un filtro en los ciclos `for` que controlan tanto la lógica de movimiento como el dibujo en pantalla. Esta variable determina si se ejecutan las operaciones para todos los elementos (valor 1), para uno de cada dos (valor 2), uno de cada cuatro (valor 4), y así sucesivamente.

```
for( let i=this.puntoDeInicioCiclo ; i<this.figuras.length ; i+=UNO_DE_CADA ){
    push();
    translate( this.figuras[ i ].getX() ,
    this.figuras[ i ].getY() ,
    this.figuras[ i ].getZ() );

    rotate( this.figuras[ i ].getRotacion() );

    let cualCapa = this.figuras[ i ].cualCapa;

    this.capa[ cualCapa ].dibujarPantalla(
        this.figuras[ i ].nx ,
        this.figuras[ i ].ny );

    pop();
}

this.puntoDeInicioCiclo = ( this.puntoDeInicioCiclo+1 ) % UNO_DE_CADA;
```

Figuras 17 - Extracto del método dibujar() en la clase Pintura

Como se ve en la figura 17, el ciclo `for` que se muestra en el código recorre las figuras en un orden específico determinado por la variable `this.puntoDeInicioCiclo`. Esta variable define el índice desde donde comienza el ciclo, y su valor se incrementa en cada iteración. El ciclo avanza a través de los elementos de `this.figuras` saltando una cantidad de pasos definidos por la constante `UNO_DE_CADA`, lo que permite procesar solo ciertas figuras en intervalos regulares. Esto significa que, dependiendo del valor de `UNO_DE_CADA`, el ciclo puede procesar, por ejemplo, solo una de cada dos o tres figuras.

Al finalizar cada ciclo, la última línea `this.puntoDeInicioCiclo = (this.puntoDeInicioCiclo + 1) % UNO_DE_CADA` asegura que el índice de inicio se ajuste para la siguiente iteración. La expresión `(this.puntoDeInicioCiclo + 1) % UNO_DE_CADA` incrementa el valor de `this.puntoDeInicioCiclo` en 1, pero lo limita al rango definido por `UNO_DE_CADA`. Esto crea un patrón repetitivo en el ciclo, permitiendo que el procesamiento de las figuras se distribuya de forma eficiente y escalonada a lo largo de múltiples cuadros sin procesar todas las figuras de manera secuencial.

Decidí trabajar con una escala exponencial —potencias de 2— para tener un control más predecible y estable de la carga computacional: 1, 2, 4, 8, etc. De esta manera, no se eliminaban elementos de la composición, sino que se espaciaba su actualización a lo largo del tiempo. Lo importante es que todos los elementos seguían estando presentes en la escena, pero no todos eran actualizados o dibujados en cada ciclo de animación. Por otra parte, como esto atentaba contra la velocidad de los caminantes, ya que al actualizar sus movimientos de forma espaciada su velocidad disminuia, tuve que multiplicar la velocidad por esa misma variable `UNO_DE_CADA` para compensar.

Con este mecanismo, logré que la aplicación pudiera sostener un framerate por encima de los 20 cuadros por segundo, incluso en dispositivos con capacidades gráficas limitadas. En vez de sacrificar complejidad visual, la obra adoptó una lógica más parecida a la de una orquesta que entra por secciones, permitiendo que la imagen se construya con ritmo y persistencia, sin perder su densidad formal.

La estrategia de alternar la frecuencia de actualización de los elementos, si bien efectiva para sostener la performance, trajo consigo una nueva dificultad: el refresco de pantalla. No podía simplemente aplicar un `background(0)` en todos los fotogramas, porque eso implicaba borrar todo lo que no se había dibujado ese mismo ciclo, y como los caminantes se actualizaban por turnos, el resultado era una composición construida por fragmentos visibles, incompleta, parcial.

Por otro lado, si el fondo no se refrescaba en absoluto, el efecto acumulativo de los dibujos —sumado al leve movimiento de las figuras— terminaba produciendo un “pastiche” visual caótico, en el que las formas se superponían indefinidamente, desdibujando la intención compositiva. Necesitaba entonces una lógica más sofisticada para el refresco: un balance entre persistencia visual y renovación de la imagen.

```
if( this.hayMovimiento() ){
    push();
    fill(0, int(255/UNO_DE_CADA) );
    translate(0,0,-300);
    rectMode( CENTER );
    rect( width/2 , height/2 , width*2 , height*2 );
    pop();
}else{
    if( frameCount % (UNO_DE_CADA * CANTIDAD_CICLOS_REFRESCO) == 0 ){
        push();
        fill(0, 100 );
        translate(0,0,-300);
        rectMode( CENTER );
        rect( width/2 , height/2 , width*2 , height*2 );
        pop();
    }
}
```

Figuras 18 - Extracto del método dibujar() en la clase Pintura

La primera decisión fue condicionar el comportamiento del fondo al estado de movimiento de las partículas. Mientras las figuras se encontraran en movimiento —es decir, mientras la composición estuviera en tránsito, ya sea desde la concentración hacia la distribución o viceversa—, el fondo se actualizaba constantemente mediante la superposición de un rectángulo negro semitransparente en cada fotograma. Esto se lograba aplicando un `fill(0, int(255/UNO_DE_CADA))` seguido de un `rect()` que cubría toda la pantalla, generando así un leve velo que no borraba del todo el trazo anterior. La transparencia ajustada según la variable `UNO_DE_CADA` permitía que los movimientos dejaran un rastro tenue, contribuyendo a la sensación de fluidez sin generar acumulaciones visuales permanentes.

Sin embargo, cuando las figuras quedaban quietas —ya organizadas en su disposición final o concentradas en un punto—, la lógica cambiaba. En esos casos, el fondo no se actualizaba en cada fotograma, sino de manera espaciada: solo se aplicaba un refresco visual cuando el contador de cuadros (`frameCount`) alcanzaba un múltiplo de `UNO_DE_CADA` por `CANTIDAD_CICLOS_REFRESCO`. Esta operación definía una frecuencia de limpieza más pausada, que respetaba el ritmo de dibujo pero sin estar perfectamente alineada a él. Esto evitaba que las figuras desaparecieran demasiado rápido o en sincronía constante, desarmando la monotonía temporal y permitiendo que la composición se completara con naturalidad visual.

Esta estrategia rompía cualquier regularidad molesta y prevenía el parpadeo rítmico que suele aparecer cuando el refresco del fondo ocurre de forma demasiado predecible. Al intercalar momentos de limpieza con lapsos más largos de acumulación, se mantenía parte del “pastiche” —ese tapiz hecho de huellas persistentes—, pero sin que eso interfiriera con la estabilidad o la legibilidad de la imagen.

7.Conclusión

A lo largo del desarrollo de esta obra se fue desplegando un sistema que no solo pone en juego una lógica visual y compositiva, sino también una profunda sensibilidad hacia las condiciones técnicas de su entorno de ejecución. A pesar de que la experiencia visual difiere marcadamente entre dispositivos de alta performance —donde se lucen los gradientes más complejos, la profundidad de campo y una actualización constante de todos los elementos— y los de baja performance —donde se reducen los recursos gráficos y se alterna el dibujo por turnos—, en ambos casos se sostiene una misma estrategia compositiva de fondo. Esa consistencia estructural es la que le permite a la obra "sobrevivir" a las limitaciones técnicas, sin traicionar su identidad estética.

En ese sentido, la obra no es solo interactiva —en tanto permite al público reorganizar la composición mediante su presencia o sus clics—, sino también adaptativa: se adapta a las capacidades del dispositivo que la ejecuta, como si reconociera el entorno en el que está siendo desplegada. Esa adaptabilidad le permite existir en condiciones diversas, sin dejar de ser ella misma.

Más aún, es interesante observar cómo, entre los dos extremos de reproducción —el de máxima y el de mínima performance—, aparecen diferencias visuales notables, pero también una sorprendente continuidad estética. Como si la obra supiera mutar sin quebrarse, conservar su espíritu mientras reconfigura sus formas. Esa tensión entre cambio y permanencia le da un carácter especial: es una obra que se transforma sin perderse, que se reinventa sin disolverse.

Actualmente, la pieza cuenta con cerca de 18 variaciones posibles, de las cuales 12 logran ejecutarse correctamente en todos los dispositivos. Y dentro de cada una de esas variaciones existe, además, un nivel de variabilidad interna que impide que la misma imagen se repita: los gradientes, los valores aleatorios y la distribución de las figuras garantizan que cada ejecución sea única, irrepetible. Esto refuerza la idea de que no se trata de una imagen cerrada, sino de una obra en estado de devenir, capaz de dialogar con el contexto técnico, con la presencia del espectador y con su propio azar.

Emiliano Causa
mayo 2025

Postdata IA: Este texto ha sido elaborado con la asistencia de la herramienta de inteligencia artificial ChatGPT, utilizada únicamente para acompañar procesos de redacción, estructuración narrativa y claridad expositiva. Todos los conceptos, ideas, decisiones formales y enfoques aquí desarrollados son de mi completa autoría, y la herramienta fue empleada como un apoyo técnico en la escritura. Si bien considero que en un futuro este tipo de aclaraciones no serán necesarias —dado que el uso de asistentes será una práctica común e integrada a los modos de producción textual—, en el presente contexto de transición y debate en torno a estas tecnologías, me parece un acto de honestidad intelectual explicitar su utilización.