GRASS

Programar tus propios efectos de vídeo en Sandin FX — tutorial del lenguaje

El módulo GRASS de Sandin FX rinde homenaje al **GRAphics Symbiosis System** de Tom DeFanti (1974), el lenguaje que permitía a los artistas animar la imagen en tiempo real antes de enviarla al Sandin Image Processor. Aquí, el mismo gesto: escribes unas líneas de código, se compilan al instante, y la imagen que producen atraviesa después todos los módulos de la aplicación.

1. Dónde ocurre

En Sandin FX, pulsa el botón **GRASS** de la barra de transporte: se abre una ventana de edición. Ahí escribes el **cuerpo de una función de efecto** y luego **Compilar y aplicar**: el efecto se activa inmediatamente en la vista previa y en la exportación, y se guarda (se recarga en el próximo arranque). El menú **Ejemplos** ofrece una docena de efectos listos para modificar.

Tu efecto se inserta **al principio de la cadena**: lo que devuelve pasa después por el Adder/Multiplier, el Comparator, el Colorizer, el feedback… En la barra lateral, el módulo **GRASS — efecto programado** (verde) ajusta su **opacidad** (mezcla con el vídeo), su **modo de fusión** (Normal, Suma, Multiplicación, Pantalla) y los dos **potenciómetros p1 / p2** expuestos a tu código.

2. El lenguaje

El código es **Metal Shading Language** (muy cercano al C). Punto esencial: tu código se ejecuta **una vez por píxel**, en cada fotograma. No dibujas «sobre» la imagen: respondes a la pregunta «¿qué color para este píxel?» — y terminas siempre con `return` de un `float3` (rojo, verde, azul, cada uno de 0.0 a 1.0).

Las variables a tu disposición

Variable	Tipo	Función
`col`	`float3`	El color del píxel actual del vídeo (tras la separación cromática). El punto de partida más sencillo.
`uv`	`float2`	La posición del píxel en la imagen: `uv.x` de 0 (izquierda) a 1 (derecha), `uv.y` de 0 (arriba) a 1 (abajo). El centro es `(0.5, 0.5)`.
`t`	`float`	El tiempo en segundos. Es lo que anima todo.
`p1`, `p2`	`float`	Los dos potenciómetros del módulo GRASS (0 a 1) — tus ajustes en directo, sin recompilar. Asignables por MIDI.
`video`, `smp`	textura	El vídeo de origen completo: `video.sample(smp, coord).rgb` lee el color en cualquier coordenada `coord` (un `float2` en 0–1).

Los tipos y cómo se escriben

```
float  a = 0.5;                      // un número
float2 p = float2(0.3, 0.7);         // un punto / vector 2D (x, y)
float3 c = float3(1.0, 0.5, 0.0);    // un color (r, v, a) — aquí naranja
float3 gris = float3(0.5);           // atajo: (0.5, 0.5, 0.5)

// «swizzle»: acceder a las componentes
float rojo = c.r;                    // también .g .b, o .x .y para un float2
float3 inverso = c.bgr;              // reordena los canales de una vez
```

Trampa n.º 1: escribe siempre los números con decimal — `2.0` y no `2`. Metal es estricto: `uv * 2` puede negarse a compilar donde `uv * 2.0` pasa. **Trampa n.º 2:** cada instrucción termina con punto y coma. **Trampa n.º 3:** siempre hay que alcanzar un `return`.

Las operaciones vectoriales

Los operadores trabajan componente a componente: `col * 0.5` oscurece, `1.0 - col` invierte, `col + float3(0.1, 0.0, 0.0)` enrojece. Es lo que hace el código tan corto.

La caja de herramientas (las funciones que sirven siempre)

Función	Qué hace
`sin(x)`, `cos(x)`	La oscilación — la base de toda animación. `sin(t)` ondula entre –1 y 1; `0.5 + 0.5*sin(t)` lo devuelve a 0–1.
`fract(x)`, `floor(x)`	Parte fraccionaria / entera. `fract` hace bucles (rampas, desplazamientos), `floor` cuantifica (bloques, escalones).
`mix(a, b, k)`	Fundido de `a` a `b` según `k` (0–1). La herramienta n.º 1 para dosificar un efecto.
`clamp(x, min, max)`	Acota un valor — útil para mantener los colores en 0–1.
`step(umbral, x)`, `smoothstep(a, b, x)`	Umbral neto / umbral suavizado — para recortes al estilo Comparator.
`length(v)`, `distance(a, b)`	Longitud de un vector, distancia entre dos puntos — círculos, halos, túneles.
`atan2(y, x)`	El ángulo de un punto alrededor del centro — espirales, caleidoscopios.
`dot(col, float3(0.299, 0.587, 0.114))`	La **luminancia** de un color — para razonar en «claro/oscuro».
`abs(x)`, `min`, `max`, `pow`, `fmod`	Los clásicos: valor absoluto, límites, potencia, módulo.

3. Primeros pasos — cinco efectos desde cero

a. Tocar los colores

```
return 1.0 - col;                    // negativo
```

```
float l = dot(col, float3(0.299, 0.587, 0.114));   // luminancia
return float3(l * 1.1, l * 0.8, l * 0.5);          // sepia
```

b. Animar con el tiempo

```
// pulsación: la imagen respira al ritmo de p1
float pulse = 0.75 + 0.25 * sin(t * (1.0 + p1 * 8.0));
return col * pulse;
```

c. Desplazar la lectura del vídeo (el gran clásico)

En lugar de leer el vídeo «aquí» (`uv`), léelo **en otro sitio**: toda distorsión nace de ahí.

```
// ondas horizontales: cada línea se desplaza según su altura
float2 d = float2(sin(uv.y * 30.0 + t * 2.0) * 0.02 * p1, 0.0);
return video.sample(smp, uv + d).rgb;
```

d. Recortar el espacio

```
// pantalla dividida: mitad izquierda normal, mitad derecha en negativo
if (uv.x > 0.5) { return 1.0 - col; }
return col;
```

e. Generar sin el vídeo (el GRASS puro)

```
// anillos concéntricos animados — ignora col, crea la imagen
float r = distance(uv, float2(0.5, 0.5));
float v = 0.5 + 0.5 * sin(r * 40.0 - t * 3.0);
return float3(v * 0.9, v * 0.4, 1.0 - v);
```

Un generador puro cubre el vídeo: baja la **Opacidad** del módulo GRASS o pasa la **Fusión** a Suma/Pantalla para superponerlo — y deja que el Colorizer y el feedback hagan el resto.

4. Usar bien p1 y p2

Escribe tus constantes interesantes en función de `p1` / `p2` para tocar el efecto en directo:

```
float velocidad = 1.0 + p1 * 8.0;          // p1 en 0 → lento, en 1 → rápido
float sectores  = 2.0 + floor(p2 * 10.0);  // p2 cuantificado: de 2 a 12 sectores
```

Truco: `floor(p * n)` convierte un potenciómetro continuo en un selector con posiciones.

5. Leer los errores de compilación

En caso de error, el mensaje del compilador aparece en rojo bajo el editor. Busca `error:` seguido de una descripción y un fragmento. Los más habituales:

Mensaje (fragmento)	Causa probable
`expected ';'`	Punto y coma olvidado en la línea anterior.
`cannot convert ... 'int'`	Número escrito sin decimal (`2` en vez de `2.0`).
`use of undeclared identifier`	Nombre mal escrito (`uv` , `col` , `video` …) o variable no declarada.
`control reaches end of non-void function`	Un camino del código no alcanza ningún `return` .

6. Rendimiento

El código se ejecuta para cada píxel, 60 veces por segundo — mantenlo ligero: evita los bucles largos (unas decenas de iteraciones como máximo, como el ejemplo Lissajous y sus 48 puntos) y los `sample` en gran número. Todo lo demás — `sin` , `mix` , aritmética — es casi gratis en la GPU.

7. Los ejemplos incluidos

Ejemplo	Qué enseña	p1 / p2
Negativo	La operación más sencilla sobre `col`	—
Pixelado	Cuantificar `uv` con `floor`	tamaño de los píxeles
Ondulaciones	Desplazamiento radial de la lectura del vídeo	fuerza / frecuencia
Caleidoscopio espejo	Plegar `uv` con `abs`	—
Caleidoscopio angular	Coordenadas polares (`atan2` , `fmod`)	sectores / rotación
Glitch en bloques	Azar determinista por celda, intercambio de canales	intensidad / tamaño
TV desajustada	Desplazamiento con `fract` , eventos por escalones de tiempo	velocidad / desgarro
Plasma	Superposición de senos, modulación de la señal	frecuencia / intensidad
Túnel psicodélico	Generador puro en polares	velocidad / ramas
Lissajous luminoso	Bucle, acumulación de halos	razones X / Y
Julia (fractal animado)	Iteración z^2+c, coloración suave	constante c (x / y)
Espiral logarítmica	Coordenadas polares + log	brazos / apretado

Método recomendado: parte de un ejemplo cercano a lo que imaginas, cambia *una* constante, recompila, observa. Era la pedagogía de GRASS en 1974 — sigue funcionando.

8. Anexo — el código completo de los ejemplos

Los ejemplos del menú, tal como se entregan (se pueden copiar y pegar en el editor). Aquí los comentarios están traducidos; el código es idéntico.

Negativo

```
// Negativo de vídeo
return 1.0 - col;
```

Pixelado

```
// Pixelado en mosaico — p1: tamaño de los píxeles
float n = 8.0 + (1.0 - p1) * 120.0;
float2 q = (floor(uv * n) + 0.5) / n;
return video.sample(smp, q).rgb;
```

Ondulaciones

```
// Ondulaciones concéntricas que deforman el vídeo — p1: fuerza, p2: frecuencia
float2 d = uv - 0.5;
float r = length(d) + 0.0001;
float rip = sin(r * (20.0 + p2 * 70.0) - t * 4.0) * (0.002 + p1 * 0.03);
return video.sample(smp, uv + (d / r) * rip).rgb;
```

Caleidoscopio espejo

```
// Caleidoscopio: la imagen plegada sobre sí misma
float2 m = float2(abs(uv.x * 2.0 - 1.0), abs(uv.y * 2.0 - 1.0));
return video.sample(smp, m).rgb;
```

Caleidoscopio angular

```
// Caleidoscopio angular — p1: número de sectores, p2: rotación
float2 d = uv - 0.5;
float n = 2.0 + floor(p1 * 10.0);
float a = atan2(d.y, d.x) + p2 * 6.28318;
a = abs(fmod(a, 6.28318 / n) - 3.14159 / n);
float r = length(d);
return video.sample(smp, 0.5 + r * float2(cos(a), sin(a))).rgb;
```

Glitch en bloques

```
// Glitch en bloques, estilo señal corrupta — p1: intensidad, p2: tamaño de los bloques
float2 cell = floor(uv * (4.0 + (1.0 - p2) * 28.0));
float h = fract(sin(dot(cell, float2(12.9898, 78.233)) + floor(t * 8.0)) * 43758.5453);
float2 off = (h < p1 * 0.5) ? float2(h - 0.5, fract(h * 7.0) - 0.5) * 0.2 : float2(0.0);
float3 c = video.sample(smp, uv + off).rgb;
if (h < p1 * 0.15) { c = c.bgr; }   // algunos bloques intercambian sus canales
return c;
```

TV desajustada

```
// TV desajustada: desplazamiento vertical + desgarro — p1: velocidad, p2: desgarro
float roll = fract(uv.y + t * p1 * 1.5);
float tear = (fract(sin(floor(t * 6.0) * 7.13) * 43758.5453) - 0.5) * p2 * 0.25;
float x = uv.x + ((roll < 0.07) ? tear : 0.0);
float3 c = video.sample(smp, float2(x, roll)).rgb;
if (roll < 0.02) { c *= 0.3; }      // barra negra de sincronía
return c;
```

Plasma

```
// Plasma modulado por la señal — p1: frecuencia, p2: intensidad
float f = 8.0 + p1 * 30.0;
float v = sin(uv.x * f + t) * sin(uv.y * f * 0.75 - t * 1.3)
        + sin((uv.x + uv.y) * f * 0.6 + t * 0.7);
return col * (1.0 - p2 * 0.5 + p2 * 0.4 * v);
```

Túnel psicodélico

```
// Túnel psicodélico generado (ignora la fuente, como un GRASS puro)
// p1: velocidad — p2: número de ramas
float2 d = uv - 0.5;
float a = atan2(d.y, d.x);
float r = length(d);
float v = sin(10.0 / (r + 0.1) - t * (1.0 + p1 * 6.0))
        * sin(a * floor(2.0 + p2 * 10.0) + t);
return float3(0.5 + 0.5 * v, 0.2 + 0.3 * v, 0.7 - 0.4 * v);
```

Julia (fractal animado)

```
// Fractal de Julia animado (z^2 + c) — p1/p2: desplazamiento de la constante c
float2 z = (uv - 0.5) * float2(3.2, 2.4);
float2 c = float2(-0.8 + p1 * 0.6 + 0.04 * sin(t * 0.31),
                  0.156 + (p2 - 0.5) * 0.4 + 0.04 * cos(t * 0.23));
float m = 0.0;
for (int i = 0; i < 48; i++) {
    z = float2(z.x * z.x - z.y * z.y, 2.0 * z.x * z.y) + c;
    if (dot(z, z) > 4.0) { m = float(i) / 48.0; break; }
}
if (m == 0.0) { return float3(0.02); }
float k = fract(m * 3.0 + t * 0.05);
float3 ca = float3(0.85, 0.05, 0.75);
float3 cb = float3(1.0, 0.85, 0.15);
float3 cc = float3(0.88, 0.9, 0.93);
return (k < 0.5) ? mix(ca, cb, k * 2.0) : mix(cb, cc, k * 2.0 - 1.0);
```

Espiral logarítmica

```
// Espiral logarítmica luminosa — p1: número de brazos, p2: apretado
float2 d = uv - 0.5;
float r = length(d) + 0.0001;
float a = atan2(d.y, d.x);
float brazos = 2.0 + floor(p1 * 8.0);
float v = sin(a * brazos + log(r) * (4.0 + p2 * 14.0) - t * 2.0);
float m = smoothstep(0.0, 0.35, v) * smoothstep(1.0, 0.55, r * 2.0);
return col * 0.3 + float3(0.75, 0.7, 1.0) * m;
```

Lissajous luminoso

```
// Curva de Lissajous luminosa sobre el vídeo atenuado — p1/p2: razones X/Y
float g = 0.0;
for (int i = 0; i < 48; i++) {
    float ph = t * 0.6 + float(i) * 0.13;
    float2 pt = 0.5 + 0.38 * float2(sin(ph * (2.0 + floor(p1 * 4.0))),
                                    sin(ph * (3.0 + floor(p2 * 4.0)) + 1.57));
    float d2 = distance(uv, pt);
    g += 0.003 / (0.003 + d2 * d2 * 400.0);
}
return col * 0.25 + float3(0.45, 1.0, 0.55) * g;
```