Informe: Shader Generativo del Conjunto de Mandelbrot

Raul Reguera Bravo

Informática Gráfica - *Practica 9*

1. Introducción y motivación

En esta práctica se desarrolla un *shader* de fragmentos generativo ejecutable en el editor de **The Book of Shaders**. El objetivo es explorar cómo un fragment shader puede implementar un patrón matemático no trivial, en este caso el **conjunto de Mandelbrot**, combinando:

- Un algoritmo iterativo sencillo.
- Una navegación interactiva con el ratón.
- Una coloración continua y visualmente agradable.

Los fragment shaders son especialmente adecuados para este tipo de visualizaciones porque se ejecutan por píxel y pueden calcular, de manera independiente, el color asociado a cada punto de la pantalla a partir de sus coordenadas. Esto encaja con el enfoque de *The Book of Shaders*, que introduce el uso de `gl_FragCoord`, la normalización de coordenadas y la manipulación de colores a partir de funciones matemáticas sobre el plano.

El **conjunto de Mandelbrot** es un fractal definido como el conjunto de números complejos (c) para los que la sucesión:

$$z_0 = 0, \quad z_{n+1} = z_n^2 + c$$

permanece acotada (no diverge). Visualmente, su frontera presenta una complejidad infinita: al aumentar el zoom aparecen niveles de detalle cada vez más finos. Esta combinación de regla muy simple y comportamiento extremadamente complejo lo convierte en un candidato ideal para un *shader* generativo.

La motivación principal del shader es, por tanto:

- Implementar el algoritmo de *escape time* del conjunto de Mandelbrot directamente en GLSL.
- Permitir una exploración interactiva básica del fractal usando el ratón (posición y zoom).
- Diseñar una paleta de color suave que haga el resultado estéticamente agradable y ejecutable sin modificaciones en el entorno de *The Book of Shaders*.

---

2. Descripción general del shader

El fragment shader final utiliza los *uniforms* estándar del entorno de *The Book of Shaders*:

- `u_resolution` (`vec2`): resolución de la ventana (ancho, alto).
- `u_mouse` (`vec2`): posición actual del ratón en píxeles.
- `u_time` (`float`): tiempo en segundos desde el inicio.

El comportamiento general es:

- El plano de la pantalla se mapea al **plano complejo** en el que se define el conjunto de Mandelbrot.
- La posición horizontal del ratón (X) marca el centro horizontal del zoom.
- La posición vertical del ratón (Y) controla el nivel de zoom, desde una vista global a un zoom profundo.
- En cada píxel se ejecuta el algoritmo iterativo (z_{n+1} = z_n^2 + c). Según cuántas iteraciones tarda en escapar, se asigna un color mediante una paleta suave basada en cosenos.

---

3. Código del shader de fragmentos

A continuación se muestra el shader completo en GLSL, en su versión "legible" para el editor de *The Book of Shaders*:

```
#ifdef GL_ES
precision mediump float;
#endif
```

```
uniform vec2 u_resolution;
uniform vec2 u_mouse;
uniform float u_time;

void main() {
    // Coordenadas normalizadas y centradas en píxeles
    vec2 uv = (gl_FragCoord.xy - 0.5 * u_resolution.xy) / u_resolution.y;

    // Ratón normalizado [0,1]
    vec2 m = u_mouse / u_resolution;
    float mx = clamp(m.x, 0.0, 1.0);
    float my = clamp(m.y, 0.0, 1.0);

    // --------- ZOOM CONTROLADO POR Y ----------
    // my = 0 → scale ~ 3.0   (vista amplia)
    // my = 1 → scale ~ 0.003 (zoom muy profundo)
    float scale = mix(3.0, 0.003, my);

    // --------- X MARCA EL CENTRO DEL ZOOM ----------
    // Curva suave para que el ratón sea menos brusco, pero recorre todo el
rango
    float k = 2.0; // 1.0 = lineal, >1 = más suave en el centro
    float mxAdj;
    if (mx < 0.5) {
        mxAdj = 0.5 * pow(mx * 2.0, k);
    } else {
        mxAdj = 1.0 - 0.5 * pow((1.0 - mx) * 2.0, k);
    }

    // Rango horizontal útil del Mandelbrot
    float left  = -2.2;
    float right =  0.8;
    float cx = mix(left, right, mxAdj);
    float cy = 0.0;
    vec2 center = vec2(cx, cy);

    // Coordenada compleja c
    vec2 c = uv * scale + center;
```

```
// Iteración Mandelbrot
vec2 z = vec2(0.0);
float n = 0.0;
const int MAX_IT = 180;

for (int i = 0; i < MAX_IT; i++) {
    if (dot(z, z) > 4.0) break;

    // z_{n+1} = z_n^2 + c
    z = vec2(
        z.x * z.x - z.y * z.y,
        2.0 * z.x * z.y
    ) + c;

    n = float(i);
}

vec3 color;

// Interior del conjunto → negro
if (dot(z, z) <= 4.0) {
    color = vec3(0.0);
} else {
    // Normalizamos iteraciones
    float t = n / float(MAX_IT);

    // Fondo azul oscuro
    vec3 base = vec3(0.02, 0.04, 0.10);

    // Paleta tipo nebulosa basada en cosenos
    vec3 pal = 0.5 + 0.5 * cos(6.28318 * (t + vec3(0.0, 0.33, 0.67)));

    // Mezcla entre fondo y paleta
    float mixFactor = smoothstep(0.0, 1.0, t);
    color = mix(base, pal, mixFactor);
}
```

```
    gl_FragColor = vec4(color, 1.0);
}
```

---

4. Desarrollo detallado

4.1. Sistema de coordenadas

Se parte de `gl_FragCoord.xy` y se resta la mitad de la resolución para centrar el origen en el centro de la pantalla:

```
vec2 uv = (gl_FragCoord.xy - 0.5 * u_resolution.xy) / u_resolution.y;
```

Después se divide por `u_resolution.y` para normalizar a unidades aproximadamente cuadradas y corregir el *aspect ratio*. De este modo, un "cuadrado" en el plano complejo no aparece deformado por el formato de la ventana.

4.2. Uso del ratón

Se normaliza `u_mouse` dividiendo por `u_resolution`:

```
vec2 m = u_mouse / u_resolution;
float mx = clamp(m.x, 0.0, 1.0);
float my = clamp(m.y, 0.0, 1.0);
```

La componente X (`mx`) controlará el centro horizontal del zoom, mientras que la componente Y (`my`) controlará el nivel de zoom.

4.3. Control de zoom

El parámetro `scale` se obtiene interpolando linealmente entre `3.0` (vista amplia) y `0.003` (zoom profundo) en función de `my`:

```
float scale = mix(3.0, 0.003, my);
```

A menor `scale`, más pequeño es el área del plano complejo que representa cada píxel y, por tanto, mayor es el zoom.

4.4. Centro horizontal y suavizado

Para evitar que el movimiento del ratón sea demasiado brusco, se aplica una función de potencia sobre `mx` para suavizar la respuesta cerca del centro:

```
float k = 2.0;
float mxAdj;
if (mx < 0.5) {
    mxAdj = 0.5 * pow(mx * 2.0, k);
} else {
    mxAdj = 1.0 - 0.5 * pow((1.0 - mx) * 2.0, k);
}
```

A partir de `mxAdj` se interpola entre `left = -2.2` y `right = 0.8`, que son valores típicos para encuadrar el conjunto de Mandelbrot en el eje real:

```
float cx = mix(left, right, mxAdj);
vec2 center = vec2(cx, 0.0);
```

4.5. Algoritmo de Mandelbrot

Para cada píxel se inicializa (z = 0) y se itera (z_{n+1} = z_n^2 + c) hasta un máximo de `MAX_IT` iteraciones o hasta que (|z|^2 > 4):

```
vec2 z = vec2(0.0);
float n = 0.0;
const int MAX_IT = 180;

for (int i = 0; i < MAX_IT; i++) {
    if (dot(z, z) > 4.0) break;

    z = vec2(
        z.x * z.x - z.y * z.y,
        2.0 * z.x * z.y
    ) + c;

    n = float(i);
}
```

El número de iteraciones `n` se usa como medida de cuánto tarda el punto en escapar. Si no escapa antes de `MAX_IT`, se considera que pertenece (o está muy cerca) del conjunto.

4.6. Coloración

Los puntos que no escapan (interior del conjunto) se pintan en negro:

```
if (dot(z, z) <= 4.0) {
   color = vec3(0.0);
}
```

Para el exterior, se normaliza `n` a (t \in [0,1]) y se construye una paleta basada en cosenos:

```
float t = n / float(MAX_IT);

vec3 base = vec3(0.02, 0.04, 0.10);
vec3 pal = 0.5 + 0.5 * cos(6.28318 * (t + vec3(0.0, 0.33, 0.67)));

float mixFactor = smoothstep(0.0, 1.0, t);
color = mix(base, pal, mixFactor);
```

De esta forma, los puntos que escapan rápidamente tienden al color de fondo azul oscuro, mientras que los cercanos al borde del conjunto muestran colores más vivos y variados.

---

5. Versión tiny code (< 512 bytes)

Además de la versión completa, se propone una versión "tiny" que cumple la restricción de tamaño (≈512 bytes), pensada también para The Book of Shaders:

```
#ifdef GL_ES
precision mediump float;
#endif
uniform vec2 u_resolution;
void main(){
 vec2 uv=(gl_FragCoord.xy-u_resolution*.5)/u_resolution.y;
```

```
 vec2 c=uv*3.+vec2(-0.7,0.0),z=vec2(0.);
 float n=0.;
 for(int i=0;i<80;i++){
  if(dot(z,z)>4.)break;
  z=vec2(z.x*z.x-z.y*z.y,2.*z.x*z.y)+c;
  n=float(i);
 }
 float t=n/80.;
 vec3 col=vec3(t*t,t,1.-t);
 gl_FragColor=vec4(col,1.);
}
```

Esta versión representa el mismo conjunto de Mandelbrot con un encuadre clásico, pero sin zoom ni control interactivo, usando una paleta sencilla `vec3(t*t, t, 1.-t)`.

---

6. Conclusiones

El shader desarrollado implementa el conjunto de Mandelbrot mediante el algoritmo de *escape time* directamente en un fragment shader GLSL. La combinación de zoom interactivo controlado con `u_mouse`, junto con una paleta de color suave, permite explorar visualmente la complejidad del fractal.

Se han conseguido los objetivos planteados: integrar un patrón matemático complejo en un shader generativo, hacerlo interactivo utilizando los *uniforms* estándar de *The Book of Shaders* y proporcionar tanto una versión completa y explicada como una versión *tiny code* adecuada para formatos de presentación muy compactos.

Entre las posibles mejoras futuras se incluyen el uso de técnicas de suavizado de iteración más avanzadas y la extensión de la navegación para permitir también desplazamientos verticales precisos que nos permitan observar con mayor claridad los conjuntos de Julia y permita cambios de centro más sofisticados.

---

7. Referencias

- *The Book of Shaders* (Patricio González Vivo, Jen Lowe).

- Documentación y artículos sobre el conjunto de Mandelbrot y algoritmos de *escape time*.
- Ejemplos de shaders fractales en plataformas como Shadertoy y recursos educativos sobre GLSL.
- Video explicativo sobre el conjunto de Mandelbrot y sus subconjuntos de Julia: https://www.youtube.com/watch?v=0OP9guFmWfs