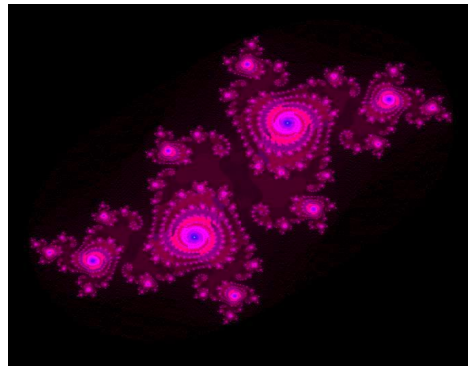


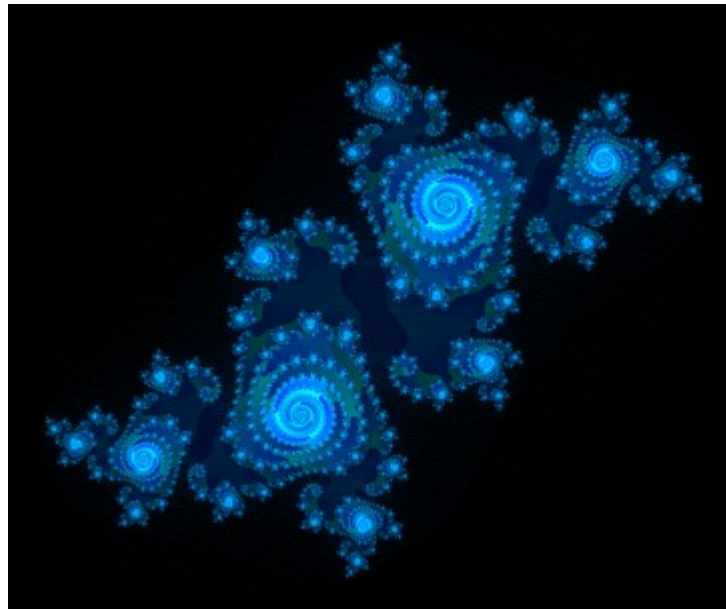
# Construyendo Fractales con programación funcional



Janeth De Anda Gil

# Fractal

Un fractal es un **objeto cuya estructura se repite a diferentes escalas**. Es decir, por mucho que nos acerquemos o alejemos del objeto, observaremos siempre la misma estructura.



# Conjunto de mandelbrot

-Colección de puntos

**type Point= (Float, Float)**

-Se construye una función que nos ayuda a crear un punto de una secuencia

**next::Point->Point->Point**

**next (u,v) (x,y)=(x\*x-y\*y+u,2\*x\*y+v)**

-Se aplica la función next p repetidamente a z para generar una infinita secuencia

[z , next p z , next p (next p z ), next p (next p (next p z )), . . .

**mandelbrot ::Point->[Point]**

**mandelbrot p=iterate (next p) (0,0)**

(Iterate se basa en una evaluación perezosa)

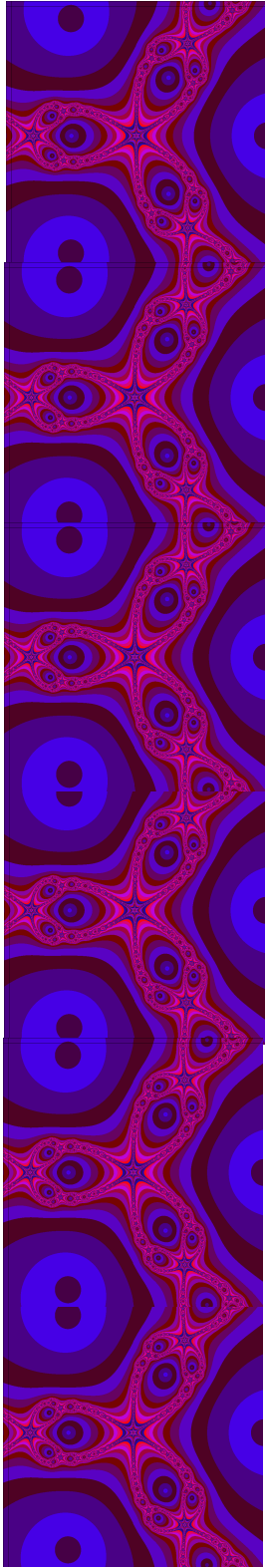


-Se define una función que indica si el punto se encuentra en el conjunto de mandelbrot (usando el teorema de pitagoras, la distancia sea menor a 10)

**cerca::Point->Bool**  
**cerca (u,v)=(u\*u+v\*v)<100**

-Se verifican si los puntos de la secuencia se encuentran en el conjunto de mandelbrot , se hace uso de *all*: acepta un predicado ,una lista y devuelve verdadero si todos los elementos de la lista cumplen el predicado:

**estaEnMandelbrot ::Int-> Point ->Bool**  
**estaEnMandelbrot n p= all cerca (take**  
**n(mandelbrot p))**



-En vez de verificar si todos los puntos de la secuencia se encuentran cerca del origen, se verificara cuantos puntos cumplen con la prueba de estar cerca del origen,  $n$  puntos, se toma el elemento  $n$  de la paleta de colores.

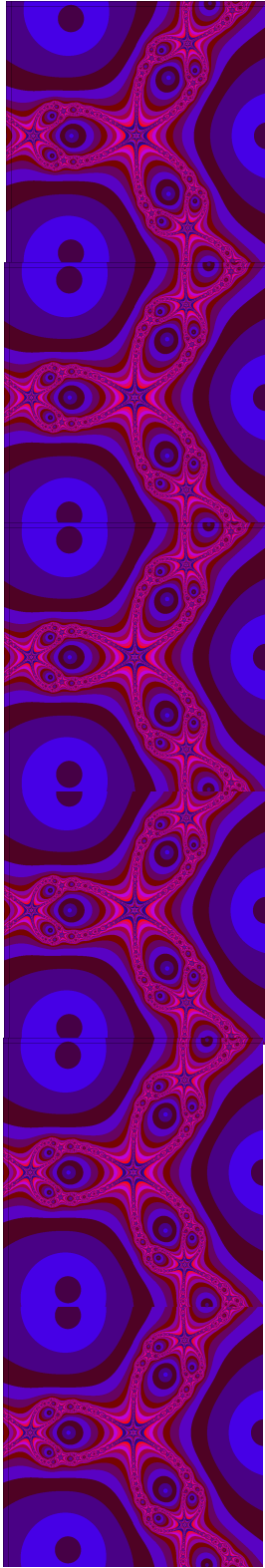
**chooseColor :: [color] -> [Point] -> color**  
**chooseColor paleta = (paleta!!).length.take**  
**n.takeWhile cerca where n=length paleta-1**

Mide el tamaño de esta lista de puntos que cumplen con que la función cerca sea verdadera y toma el elemento (color) que se encuentre en la posición de la longitud de la paleta.

-Una imagen es un mapeo que asigna el valor de un color a cada punto

**type Image color = Point → color**



A decorative vertical border on the left side of the slide, featuring a complex fractal pattern in shades of blue, purple, and magenta. The pattern consists of repeating, self-similar shapes that resemble stylized eyes or spirals.

El conjunto de mandelbrot puede ser pensado como una variable de tipo Image Bool, mapeando puntos que estan en el conjunto como verdaderos y dejando fuera los puntos Falsos. Para obtener una imagen más colorida

**fracImage::(Point->[Point])->[color]->ImageFunction  
color**

**fracImage fractal paleta= chooseColor paleta.fractal**

ChooseColor paleta se aplica a fractal (función) , en este caso se aplicara la función mandelbrot, y nos devuelve una función donde evalua un punto y devuelve un color usando el chooseColor

En los dispositivos se especifican colores para puntos en un espacio rectangular acotado por filas y columnas.

**type Grid a= [[a]]**



Cada segmento del valor mínimo al máximo debe de estar dividido en tramos iguales

```
for :: Int->Float-> Float ->[Float]  
for n min max =take n [min, min + delta ..] where  
delta = (max - min)/fromIntegral (n - 1)
```

-Se crea una malla con c número de filas y r número de renglones

```
grid :: Int -> Int ->Point -> Point-> Grid Point  
grid c r (xmin , ymin ) (xmax , ymax )= [(x , y) | x<-  
for c xmin xmax ] | y<- for r ymin ymax ]
```

-Dado un punto en la rejilla podemos aplicarle un color, ahora para toda la rejilla entonces se crea la función sample, la cual hace un doble map ya que la malla es una lista de listas.

```
sample :: Grid Point-> ImageFunction color->Grid  
color  
sample points image = map (map image) points
```

A vertical decorative border on the left side of the slide, featuring a complex fractal pattern in shades of blue, purple, and magenta. The pattern consists of repeating, nested, and distorted shapes that create a sense of depth and movement.

**draw ::Grid Point-> (Point ->[Point])-> [color]->(Grid  
color ->image)->image  
draw points fractal palette render = render (sample  
points (fracImage fractal palette))**

En este caso fracImage regresa image(que convierte los puntos a colores) y sample regresa la malla de colores a partir de una malla de puntos y del image color; render convierte la malla de colores a un objeto (imagen donde los colores pueden ser caracteres o colores , etc)



# Fractales con caracteres

-Definimos una paleta de caracteres

```
charPalette :: [Char]
charPalette= " ,o-!|?/<>X+={^O#%&8*$"
```

-putStr imprime en pantalla un salto de línea cada que encuentra un \n y unlines devuelve un String que concatena los elementos de una lista de String separados por un \n

```
charRender :: Grid Char->IO()
charRender=putStr.unlines
```

# A usar la paleta!

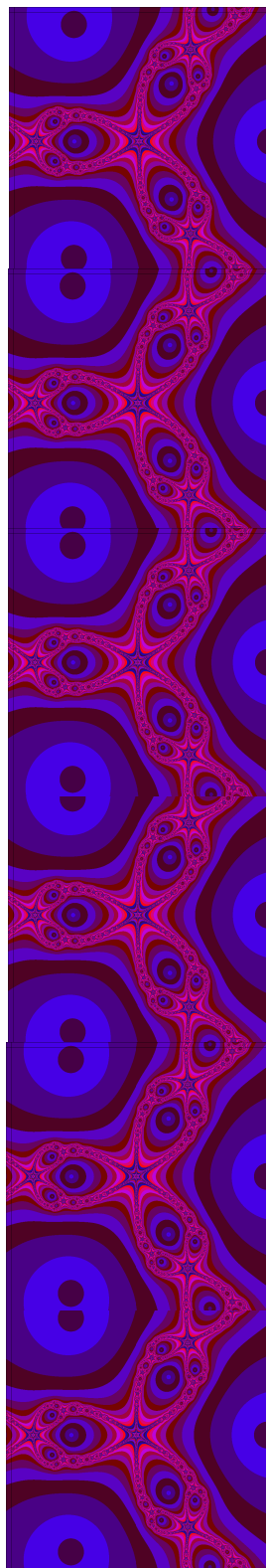
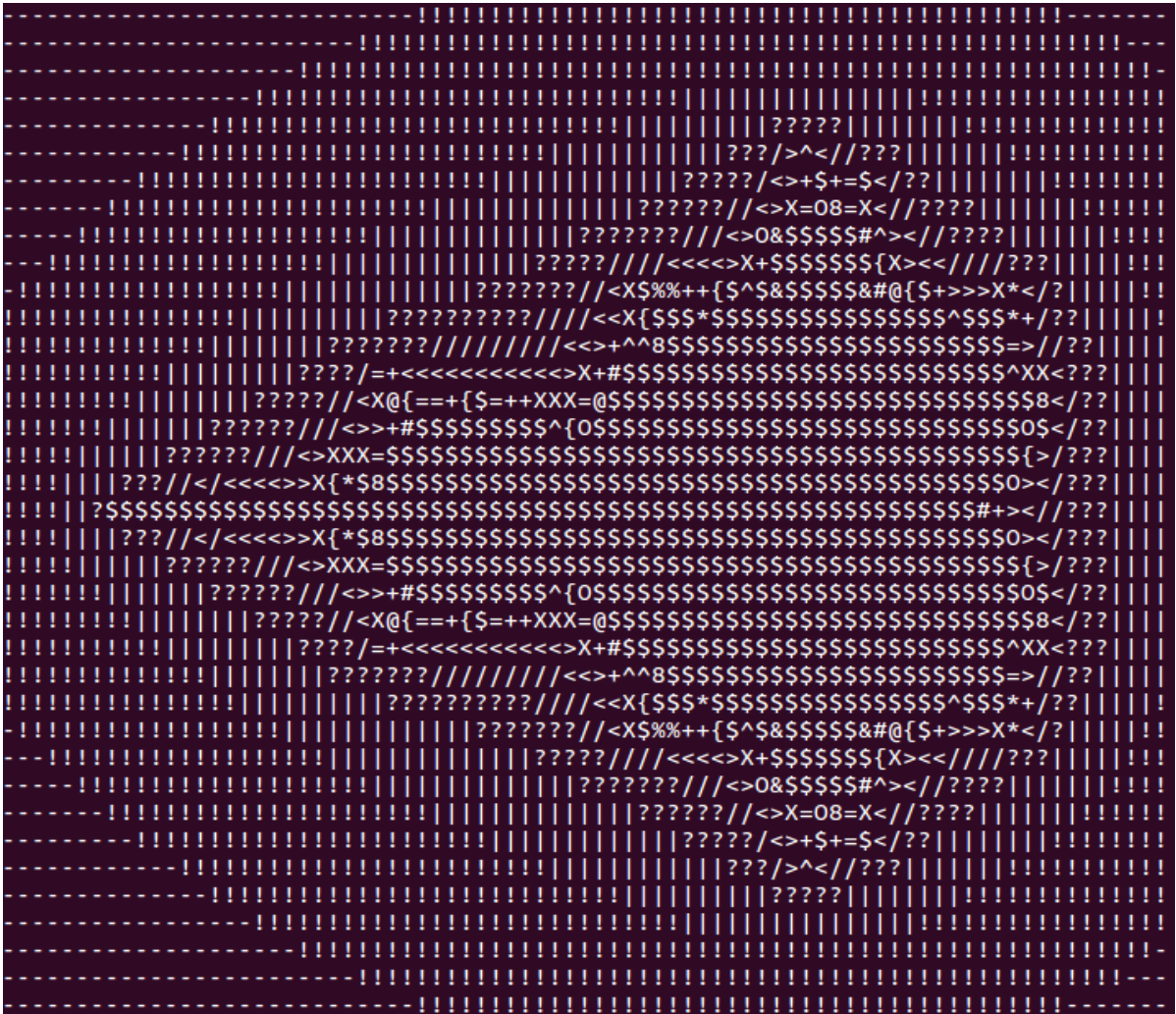
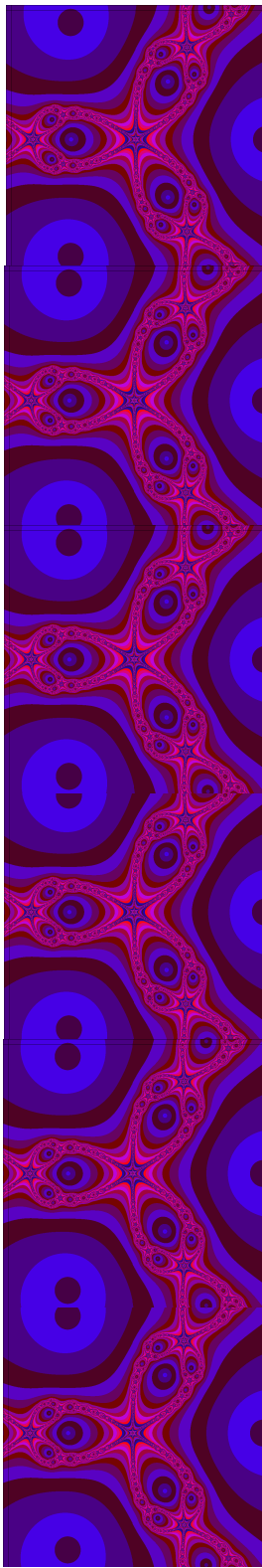
**figure1 = draw points mandelbrot charPalette  
charRender where points = grid 79 37 (-2.25, -1.5)  
(0.75, 1.5)**

```

,,,,,,,,,,,,,
,,,,,,,,,00000,
,,,,,,,,,000-|X!-000,
,,,,,,,,,00000-!|/8/<$!-00,
,,,,,,,,,000000-!|?<+#!-0000,
,,,,,,,,,0000000-!|+^$$$$$X|!-0000,
,,,,,,,,,00000-!|!|?/@$$$$$>?!-000,
,,,,,,,,,0000000-!?$[{//>$X$^&$$$$$^=0>$/|||?%! -0,
,,,,,,,,,0000000000-!?!?>&$%$$$$$$$$$$$$$$$$$$$X$$$%/-00,
,,,,,,,,,0000000-!|/XX#$$$$$$$$$$$$$$$$$$$$$$$$$8<|-00,
,,,,,,,,,0000-!|!|!|!|!|!|!|?/=$$$$$$$$$$$$$$$$$$$$$$$$$X?!!000,
,,,,,,,,,00000-!|?0><@</??<0$$$$$$$$$$$$$$$$$$$$$$$$$$$$$#!-00,
,,,,,,,,,000000-!|!|/$*$$$$$$X>+$$$$$$$$$$$$$$$$$$$$$$$$$$$$$+#!-00,
,,,,,,,,,000000-!|???<$$$$$$$$$$$$$&$$$$$$$$$$$$$$$$$$$$$$$$$$$$$>|-000,
,,,,,,,,,000-!|!|!|!|!|?>%$#$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$+!|-000,
o&$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$=/!|-000,
,,,,,,,,,000-!|!|!|!|!|?>%$#$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$+!|-000,
,,,,,,,,,000000-!|???<$$$$$$$$$$$$$&$$$$$$$$$$$$$$$$$$$$$$$$$$$$$>|-000,
,,,,,,,,,000000-!|!|/$*$$$$$$X>+$$$$$$$$$$$$$$$$$$$$$$$$$$$$$+#!-00,
,,,,,,,,,00000-!|?0><@</??<0$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$#!-00,
,,,,,,,,,0000-!|!|!|!|!|!|!|?/=$$$$$$$$$$$$$$$$$$$$$$$$$$$$$X?!!000,
,,,,,,,,,0000000-!|/XX#$$$$$$$$$$$$$$$$$$$$$$$$$$$$$8<|-00,
,,,,,,,,,0000000000-!?!?>&$%$$$$$$$$$$$$$$$$$$$$$$$X$$$%/-00,
,,,,,,,,,0000000-!?$[{//>$X$^&$$$$$^=0>$/|||?%! -0,
,,,,,,,,,00000-!|!|?/@$$$$$>?!-000,
,,,,,,,,,0000000-!|+^$$$$$X|!-0000,
,,,,,,,,,000000-!|?<+#!-0000,
,,,,,,,,,00000-!|/8/<$!-00,
,,,,,,,,,000-|X!-000,
,,,,,,,,,00000,
,,,,,,,,,,,,,

```







A vertical banner with a complex, repeating pattern. The design features stylized, colorful eyes and faces in shades of blue, purple, and pink, set against a dark background. The pattern is dense and intricate, with a central vertical axis of symmetry. The eyes are large and prominent, with detailed eyelids and pupils. The faces are smaller and more abstract, often appearing as part of the larger eye structures. The overall effect is a vibrant, almost hypnotic visual rhythm.

[illegible]

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,  
,,,oo-!=\$|/!-oooooooo-!!<|-ooo,,,  
,,,o-?\$^+\$\$\$\$\$&\$|!-!!X\$\$\$\$\$\$+\$\$\$<-o,,,  
,,,o-\$\$\$\$\$\$\$\${+X</??/\$\$\$\$\$\$\$\$\$\$\$\$O\$!o,,,  
,,,oo-!\$\$\$\$\$\$\$@{+X><X&\$\$\$\$\$\$\$\$\$\$\$%+>/|!-o,,,  
,,,o-/\$^<>\$\$\$\$\$\$\$\$\$\$\${0\*\$&\$-o,,,  
,,,o-=\$\$%[^\$?o,,,  
,,,o!\$=\$>!o,,,  
,,,o!OX\$Xo,,,  
,,,o?\${+XXXX+={\$\$\${\$o,,,  
,,,o-?+\$8@\$+\$X>>>>X+=#\$\$\$0-o,,,  
,,,oo!\$0+X><<//<\$\$\$=|-o,,,  
,,,o-!<\$X></???|!!-ooo,,,  
,,,oo--!?\$8\$X</??|!!--ooo,,,  
,,,ooo--!|?/>\$>/?!-ooo,,,  
,,,ooo--!!|??/<X\$8\$?!-oo,,,  
,,,ooo--!!|???/<>X\$<!-o,,,  
,,,o-|=\$\$\$<//<>X+O\$!oo,,,  
,,,o-O\$\$#+=X>>>>X+\$\$@8\$+?-o,,,  
,,,o\${\$\$\${+=XXXX+{\$?o,,,  
,,,oX\$\$\$\$\$\$\$O\$\$\$\$\$\$%\$XO!o,,,  
,,,o!>=\$!o,,,  
,,,o?\$%^{\$\$-o,,,  
,,,o-\$&\$\*O{\$\$\$\$\$\$\$\$\$\$\$\$><^\$/-o,,,  
,,,o-!|/>+\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$&X<<>X+{\$\$\$\$\$\$\$\$\$!-oo,,,  
,,,o!\$O\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$/??/<X+{\$\$\$\$\$\$\$\$\$\$-o,,,  
,,,o-<\$\$\$+\$\$\$\$\$\$\$X!!-!|\$&\$\$\$\$\$+^\$?-o,,,  
,,,ooo-|<!!-oooooooo-!/\$\$=!-oo,,,  
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,



# Paleta de colores (linux)

-Necesitaremos una paleta de colores, es decir una lista de colores RGB

**paletaRGB :: [PixelRGB8]**

-Colores degradados para que la imagen tenga colores que combinaran

**degradado :: PixelRGB8 -> PixelRGB8 -> Int -> [PixelRGB8]**

**degradado (PixelRGB8 x y z) (PixelRGB8 x2 y2 z2) pasos  
= [PixelRGB8 (x+(division x2 x pasos)\*(fromIntegral i))  
(y+(division y2 y pasos)\*(fromIntegral i)) (z+(division z2 z  
pasos)\*(fromIntegral i)) | i <- [0..pasos]**

**division :: Pixel8 -> Pixel8 -> Int -> Pixel8**  
**division x y pasos= round ((fromIntegral x-fromIntegral y)  
/ (fromIntegral pasos))**



-Ahora si creamos una paleta de colores  
**paletaRGB = degradado (PixelRGB8 0 0 0)  
(PixelRGB8 100 175 225) 200**

Para crear una imagen se usa writePng seguido de la dirección donde se desea que se guarde la imagen, el panel de colores y el ancho y largo de la imagen

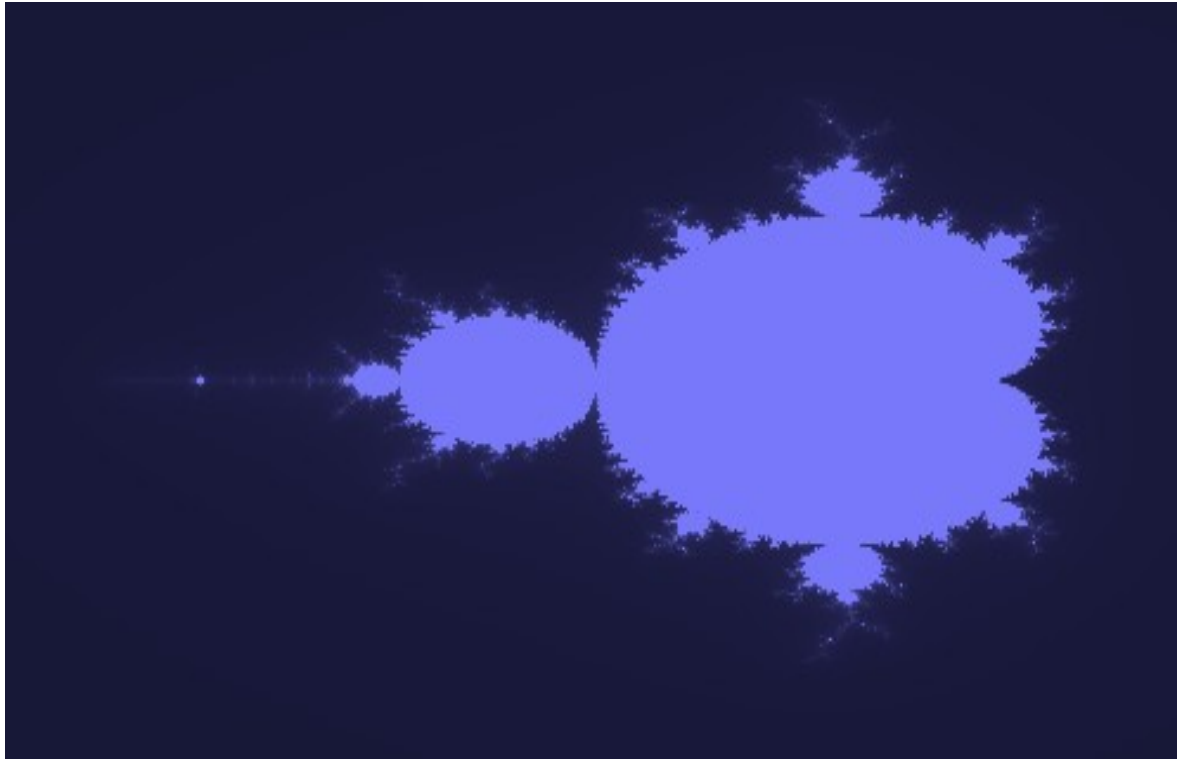
```
imageCreator :: String -> Int -> Int -> (Int->Int->PixelRGB8) -> IO ()  
imageCreator path width height pixelRenderer=  
writePng path $ generateImage pixelRenderer width  
height
```

-Ahora creamos una imagen dada una malla de pixeles

```
rgbRender :: Grid PixelRGB8 -> IO()  
rgbRender grid = imageCreator "fractal_j.png"  
(length (grid!!0)) (length grid) pixelRenderer where  
pixelRenderer x y = (grid!!y)!!x
```

figure3 = draw points mandelbrot paletaRGB rgbRender  
where points = grid 500 250 (-2.25, -1.5) (0.75, 1.5)

paletaRGB = degradado (PixelRGB8 20 20 50)  
(PixelRGB8 100 100 200) 100



PaletaRGB = degradado (PixelRGB8 50 50 80)  
(PixelRGB8 100 120 200) 200

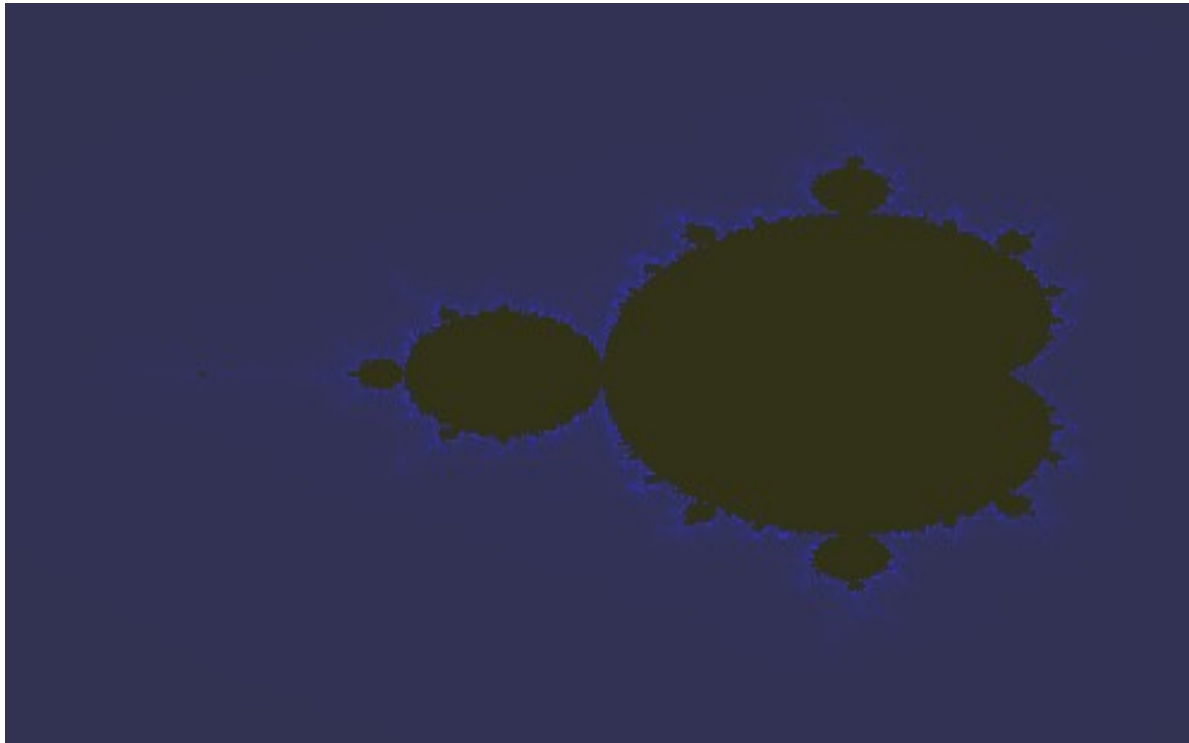
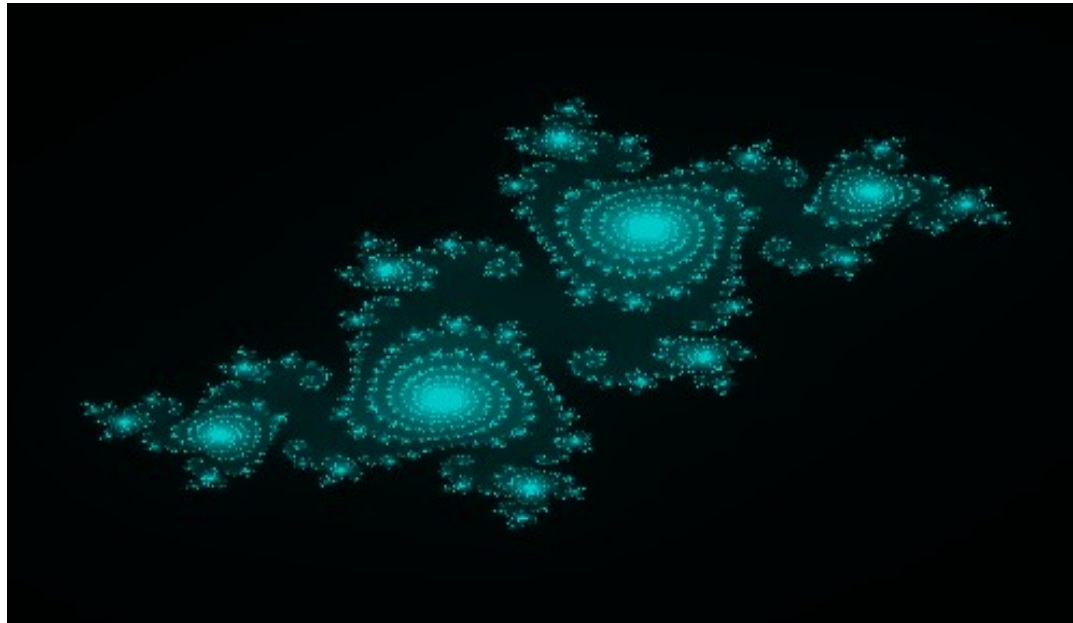


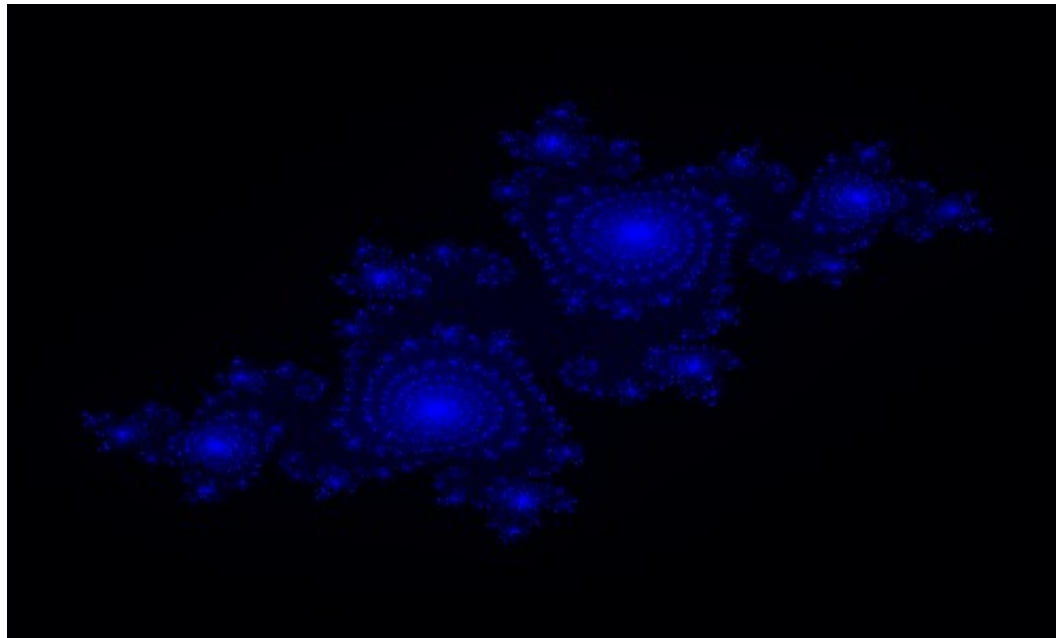


figure4 = draw points (julia (-0.194, 0.6557)) paletaRGB  
rgbRender where points = grid 500 250 (-1.5, -1.5) (1.5,  
1.5)

paletaRGB = degradado (PixelRGB8 0 0 0) (PixelRGB8  
100 175 225) 200



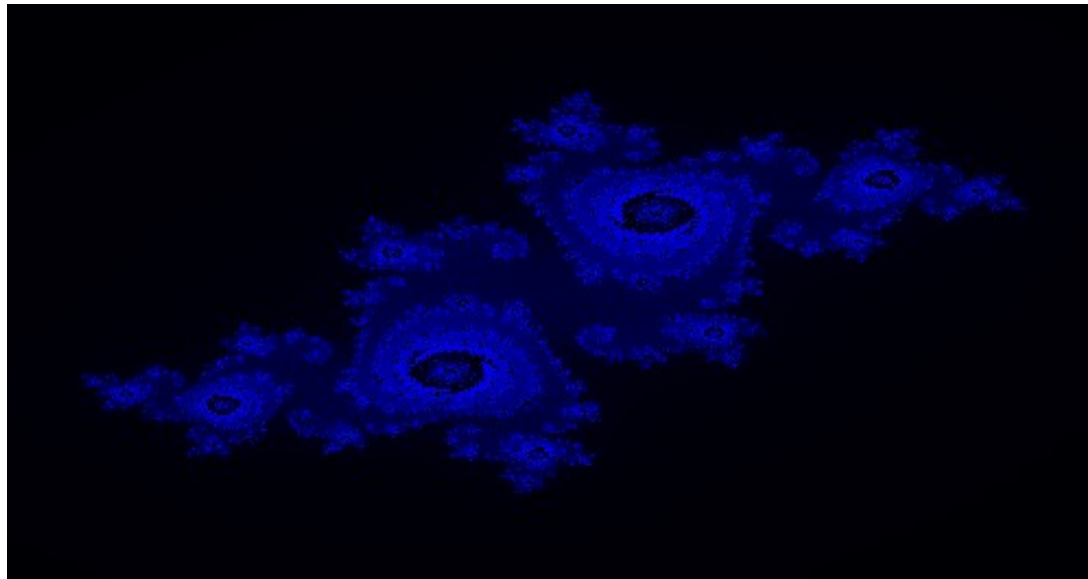
paletaRGB = degradado (PixelRGB8 0 0 0) (PixelRGB8  
100 100 220) 250



## Cambiando el degradado

```
degradado (PixelRGB8 x y z) (PixelRGB8 x2 y2 z2) pasos =  
  [PixelRGB8 (x+(division x2 x pasos)*(fromIntegral i)) (y+  
    (division y2 y pasos)*(fromIntegral i)) (z+(division z2 z  
    pasos)*(fromIntegral j)) | i <- [0..pasos],j <-[0,2..pasos*2]]
```

```
paletaRGB = degradado (PixelRGB8 0 0 0) (PixelRGB8 100  
  100 100) 100
```

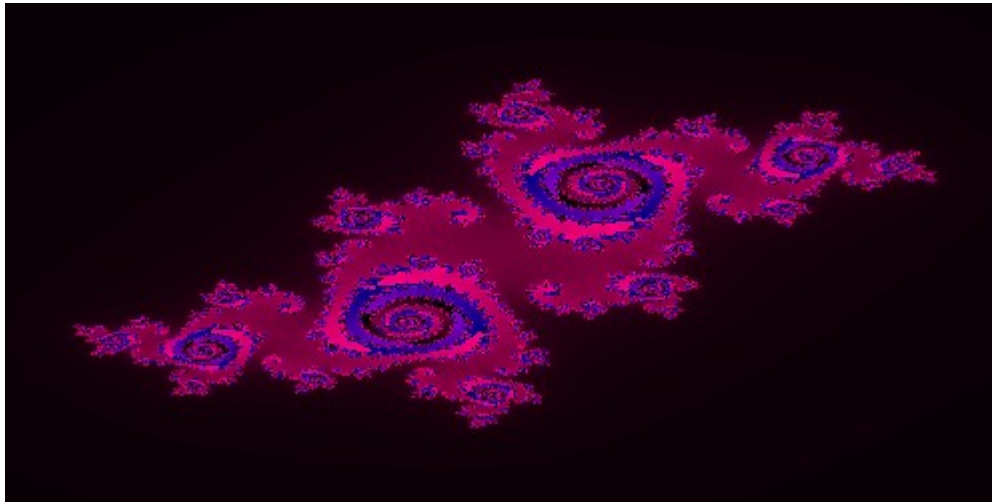




## Cambiando el degradado

```
degradado (PixelRGB8 x y z) (PixelRGB8 x2 y2 z2) pasos =  
  [PixelRGB8 (x+(division x2 x pasos)*(fromIntegral j)) (y+  
    (division y2 y pasos)*(fromIntegral i)) (z+(division z2 z  
    pasos)*(fromIntegral j)) | i <- [0..pasos],j <-[0,2..pasos*2]]
```

```
paletaRGB = degradado (PixelRGB8 0 0 0) (PixelRGB8 190  
  100 100) 100
```



paletaRGB = degradado (PixelRGB8 0 0 0) (PixelRGB8  
100 100 190) 100

