ALGORITMOS Y ESTRUCTURAS DE DATOS

Guía 2. Gráficos, simulación y recursividad

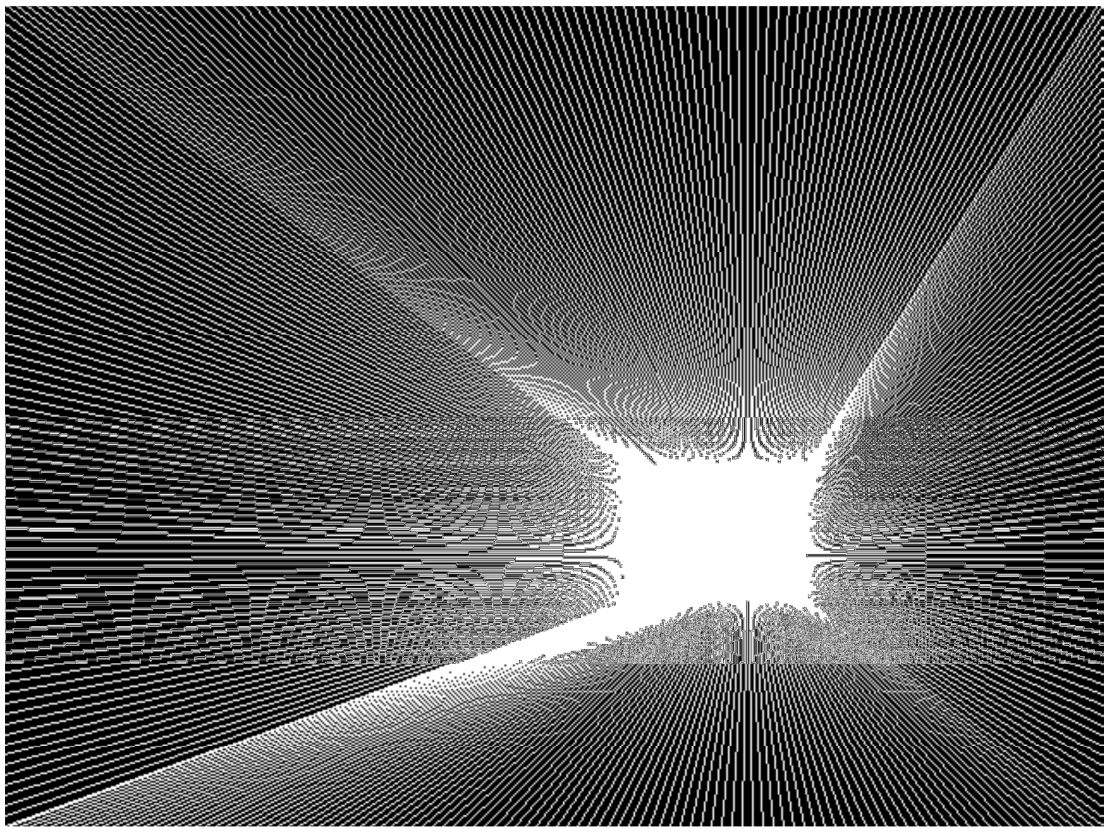
1. Se pide construir un proyecto (la elección del IDE y la plataforma quedan a criterio y gusto del alumno, siendo la recomendación de la cátedra Code:Blocks o Visual Studio 2010 corriendo en Windows) para la librería gráfica Allegro (http://alleg.sourceforge.net) y compilar el ejercicio de ejemplo que se encuentra en IK.

Los ejercicios que siguen deben realizarse utilizando la librería gráfica Allegro.

2. Un patrón de Moiré ocurre cuando dos patrones repetitivos pero ligeramente distintos se superponen. También puede ocurrir en gráficas de computadora, por la diferencia entre los pixeles físicos y los puntos matemáticos (los pixeles tienen área finita, los puntos no). Se pide realizar un programa que imprima un patrón de Moiré, cumpliendo con lo siguiente:

a. Debe comenzar con la pantalla en negro.

b. Debe recorrer el perímetro de la pantalla, avanzando de a n pixeles. Debe trazar una línea blanca entre cada punto recorrido y un cierto punto (cx, cy). Los parámetros cx, cy y n se pasan por línea de comando.



3. Se pide simular la física de una pelota de fútbol (de 0.2m de diámetro) que se encuentra en una "habitación" rectangular de x_m = 4m de ancho e y_m = 3m de alto. Se debe cumplir que:

a. La pelota se encuentra inicialmente en (x0, y0), y tiene velocidad inicial (vx0, vy0). Los parámetros x0, y0, vx0 y vy0 se deben pasar por línea de comando.

b. La pelota cumple las leyes de Newton. Por lo tanto:
   v'y(t) = a = g (constante gravitatoria)
   y'(t) = vy(t)
   v'x(t) = 0
   x'(t) = vx(t)

c. Podemos aproximar estas ecuaciones de la siguiente manera:
   (vy(n) - vy(n - 1)) / Δt = g
   vy(n) = vy(n - 1) + aΔt
   (y(n) - y(n - 1)) / Δt = vy(n)
   y(n) = y(n - 1) + vy(n)Δt
   (vx(n) - vx(n - 1)) / Δt = 0
   vx(n) = vx(n - 1)
   (x(n) - x(n - 1)) / Δt = vx(n)
   x(n) = x(n - 1) + vx(n)Δt

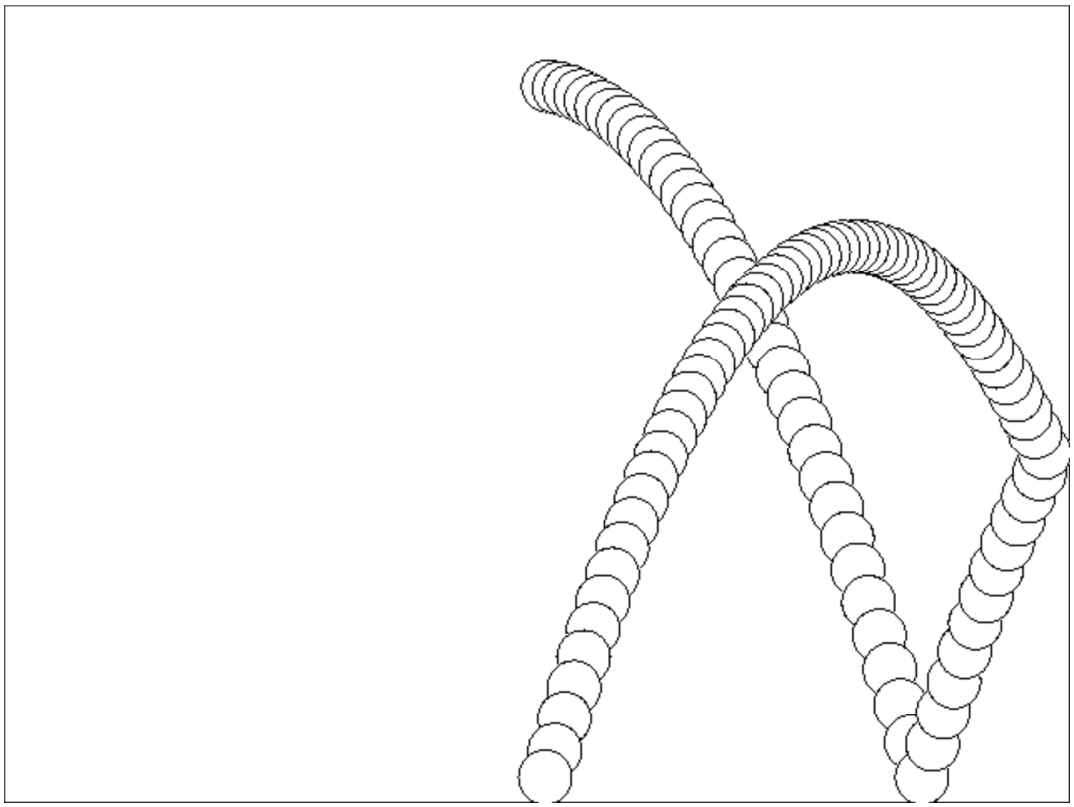
d. Para la simulación calculamos primero vx(n) y vy(n), y luego usamos estos valores para calcular x(n) e y(n). Δt debe valer por defecto 0.1 (segundos), pero ese valor se debe poder cambiar por línea de comando.

e. La habitación se encuentra en un lugar con constante gravitatoria g constante. Por defecto la constante es 9.8 (m/s2), pero es posible cambiarla por línea de comando.

f. Las paredes son parcialmente elásticas: una colisión en una pared hace que el signo de vx(n) o vy(n) se invierta, y su valor se multiplique por una constante de elasticidad. El valor de esta constante es 0.9 por defecto, pero debe poder cambiarse desde la línea de comando.

g. Se pide graficar la simulación en tiempo real. O sea: que un segundo de la simulación corresponda a un segundo real.

h. Lamparita final: se recomienda que las variables sean double, y que se usen las mismas unidades para evitar confusiones. Por ejemplo: metros para distancias y segundos para el tiempo.

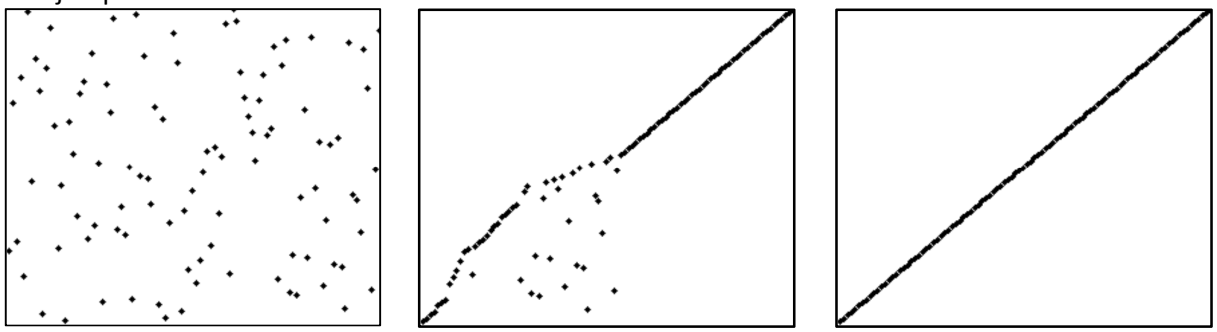


4. Se pide realizar un programa que implemente el algoritmo de ordenamiento bubble sort. El pseudo-código de bubble sort es:

```
bubbleSort(L)
{
  do
  {
    isSwapped = false;
    foreach(i = 0:(length(A) - 2))
    {
      if (L[i] > L[i + 1])
      {
        swap(L[i], L[i + 1])
        isSwapped = true
      }
    }
  } while (isSwapped)
}
```

Para probar el código se deberá generar un arreglo de 256 ints con estos valores: x[0] = 0, x[1]= 1, x[2] = 2, ..., x[255] = 255. Los valores se deberán permutar al azar, para simular que han sido desordenados (averiguar en internet como se pueden permutar valores al azar de manera eficiente). Se debe usar el algoritmo bubble sort para re-ordenar el arreglo. Para cada iteración del algoritmo bubble sort se pide graficar el arreglo, representando índice vs. valor del arreglo en ese índice.

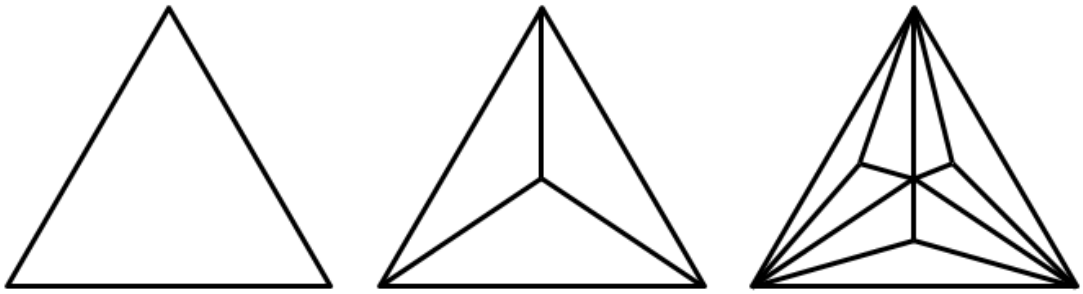
Ejemplo:



(el gráfico del arreglo al comenzar el programa, a medio camino, y al final)

Para entregar:
5. Se pide realizar un programa que permita dibujar tres tipos de fractales:

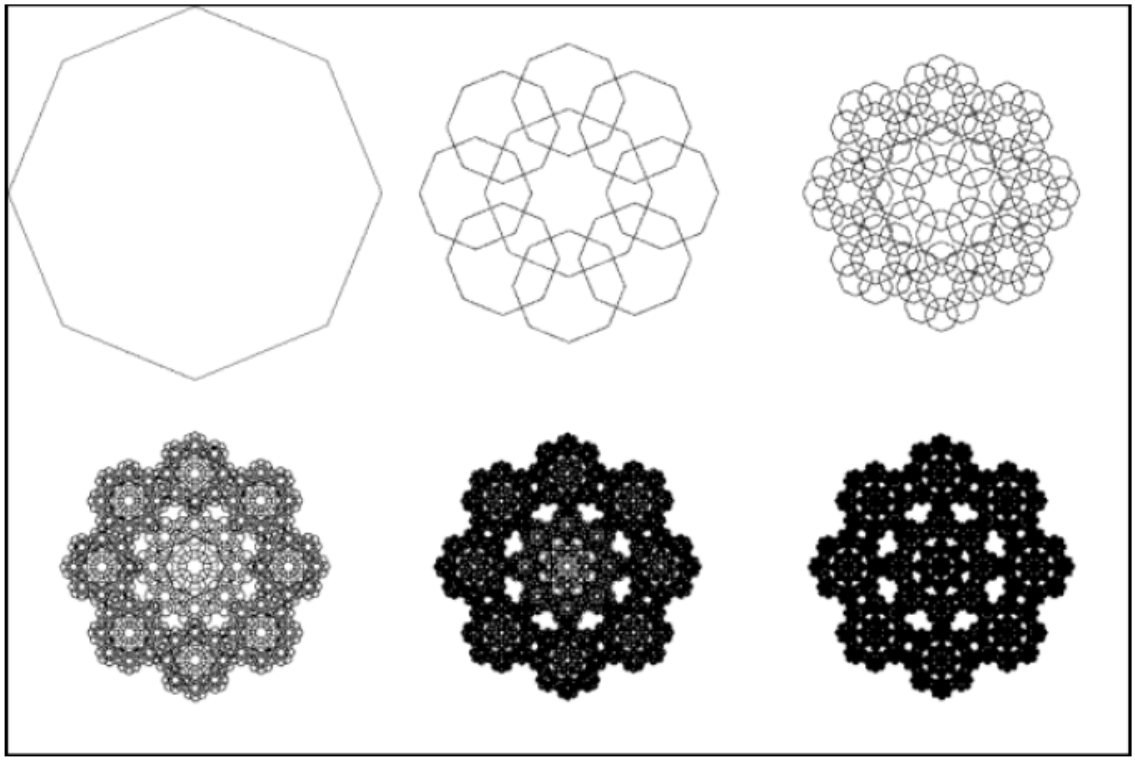
A. Triángulo de centro de masa uniforme:



Este fractal presenta la siguiente dinámica:

a. Se empieza con un triángulo ubicado en (x,y), cuya base tienen una longitud l = lStart y su lados izquierdo y derecho un ángulo leftAngle y rightAngle respectivamente medidos desde la base (de 0 a 90 en el caso derecho y de -90 a 0 en el caso izquierdo). El triángulo
b. Se le agrega al triángulo tres líneas desde sus vértices al centro de masa como muestra la figura arriba. Se continúa de esta forma por cada nuevo triángulo generado dentro del triángulo original.
c. La recursión termina cuando el tamaño de alguna de las líneas agregadas resulta inferior a cierto valor lEnd.

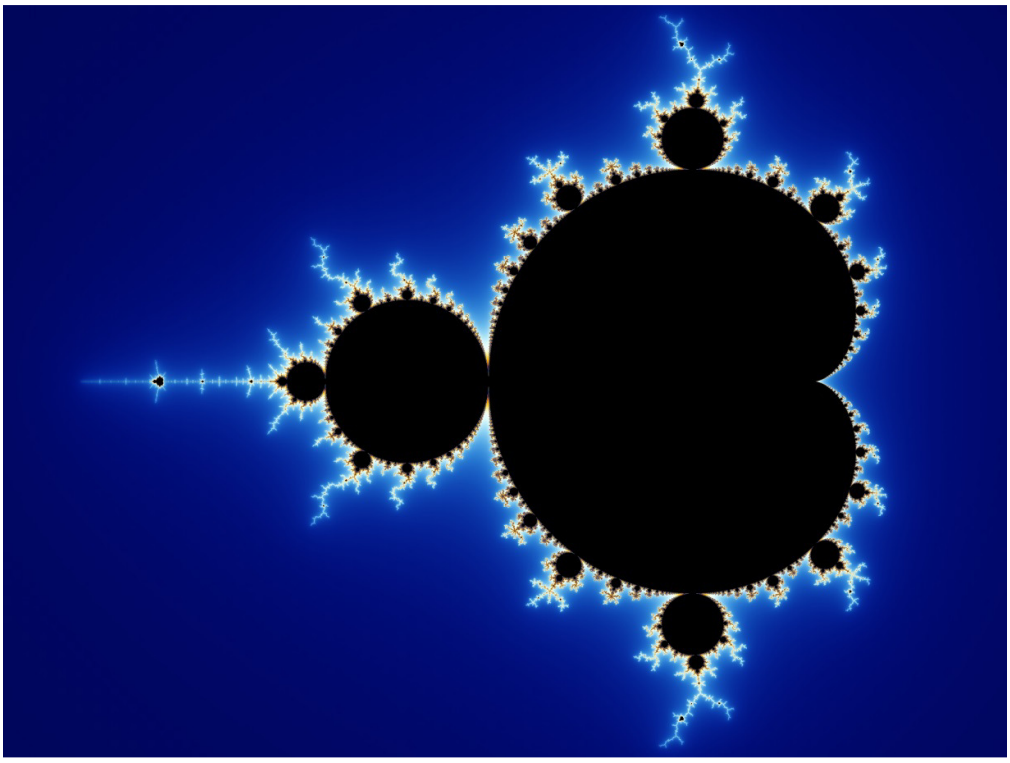
B. Fractal Octogonal:



Este fractal presenta la siguiente dinámica:

a. Se empieza con un octógono de longitud de lado l = lStart ubicado en (x, y).
b. Se dibujan ocho octógonos, en cada uno de sus vértices de lado l' igual a l multiplicado por una constante = lConstant.
c. La recursión termina cuando l es inferior a cierto valor lEnd.

C. Fractal Mandelbrot:



Este fractal presenta la siguiente dinámica:

a. Se toma un campo complejo representado por el plano $[a_0+jb_0; a_f+jb_f]$ donde el número complejo a_0+jb_0 representa el punto inferior izquierdo del plano y a_f+jb_f representa el punto superior derecho del plano.

b. Se toma también una pantalla de tamaño xMax x yMax. De forma que la granularidad de la parte real sea gReal = (a_f-a_0)/xMax y la granularidad de la parte imaginaria sea gImag = (b_f-b_0)/yMax.

c. Para cada punto Z_i en $[a_0+jb_0; a_f+jb_f]$ dadas las granularidades gReal y gImag, se evalúa la función:

$$f_n = (f_{n-1})^2 + Z_i \text{ donde } f_0 = 0.$$

Iterando hasta Nmax o hasta que la función diverja (se considera que la función diverge cuando mod(F_n) no pertenece a la circunferencia que se forma con el lado mayor del plano complejo definido), lo que ocurra primero. Se toma la profundidad del algoritmo como (n-Nmax) donde n representa la cantidad de iteraciones para cada Z_i. Nótese que para el caso en que la profundidad es 0, el punto se considera del conjunto Mandelbrot (ya que no diverge).

d. Para graficar existen dos posibilidades:

I. Se grafican los puntos del conjunto Mandelbrot de negro y el fondo de otro color.
II. Se corresponde cada pixel con cada punto del plano (mediante la granularidad) y para cada punto evaluado se colorea su correspondiente pixel según la profundidad arrojada, pintando de negro cuando la profundidad es 0 y con algún otro color cuando lo profundidad es Nmax, tomando los valores intermedios con colores de la gama entre los extremos.

e. Bibliografía adicional:
I. https://en.wikipedia.org/wiki/Mandelbrot_set
II. http://jonisalonen.com/2013/lets-draw-the-mandelbrot-set/

Pautas generales:

a. El programa recibe los siguientes parámetros por línea de comandos:
-type {UNIFORME", "OCTOGONO", "MANDELBROT"}
-lStart (0…100]
-lEnd (0…100).
-lConstant (0…1).
-leftAngle [-90…90].
-rightAngle [-90…90].
-x0, y0, xf, yf (pueden tomar valores flotantes).

b. Se deben verificar los parámetros recibidos y validarlos según lo que se esté graficando. Por ejemplo, si recibiera especificación de ángulos y se desea graficar un fractal OCTOGONO se debe señalar que ocurrió un error.

c. El dibujo del fractal MANDELBROT en "C" fue dibujado con:
–type MANDELBROT –x0 -2 –y0 -2 –yf 2 –xf 2

d. Se deberán comprobar los casos límite para asegurar el correcto funcionamiento del programa.

e. Se deberá además jugar con el uso de color en cada iteración.

f. Como opcional pueden agregarle música mientras se ejecuta la recursión.

g. El grupo que lo desee puede modificar el fractal OCTOGONO para que permita dibujar un polígono de N lados. En este caso se deberá agregar el parámetro adicional –N a recibir por línea de comandos y modificar el –type de OCTOGONO a POLIGONO.

h. Los parámetros mencionados tanto en "a" como en "g" son case insenstive.