CCBOL 2011

L-SYSTEM

Carlos Gomez

Conferencia de Ciencias de la Computación Bolivia
Jornadas de Programación Funcional
CCBOL-2011

25 de Octubre, 2011

Contenido

CCBOL 2011

1 Introducción

2 Desarrollo

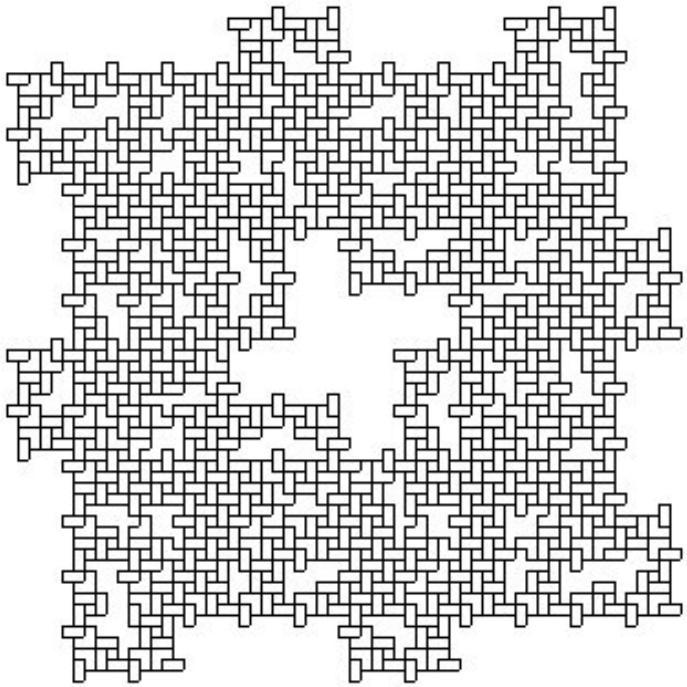
3 Resultados

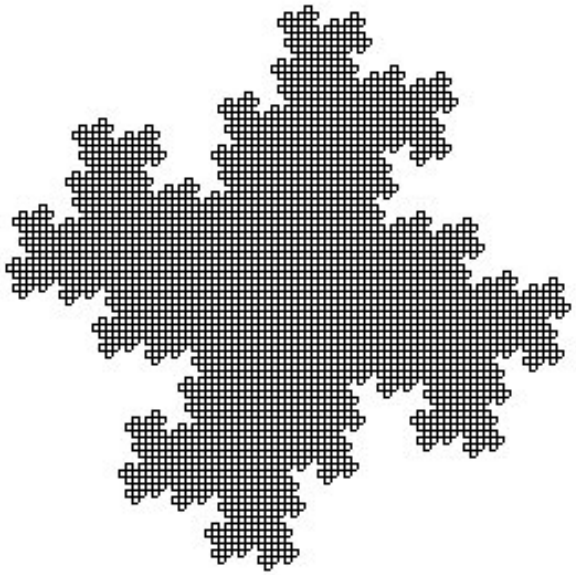
4 Conclusiones

Introducción

Motivación

Como se podría generar esto:

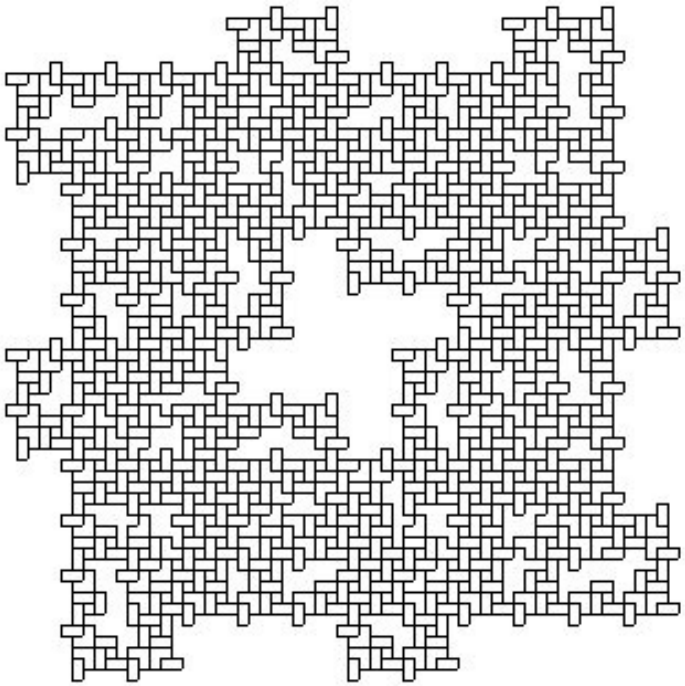




Motivación

Como se podría generar esto:

Pueden se expresados utilizando una descripción de un Sistema-L:



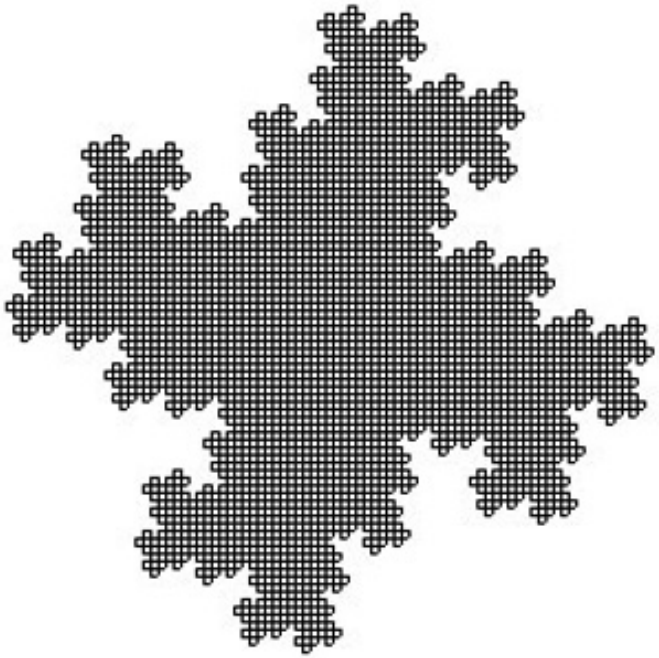
```
Gramatica
    koch C
        { alfabeto = [F,+,-]
        ; inicio = [F,-,F,-,F,-,F]
        ; prducciones
            = [F -> [F,F,-,F,-,-,F,-,F]]
        }

    Iteraciones = 5
    Angulo = 90°
```

Motivación

Como se podría generar esto:

Pueden se expresados utilizando una descripción de un Sistema-L:



Gramatica

koch E

{ alfabeto = [F,+,-]
; inicio = [F,-,F,-,F,-,F]
; prducciones
= [F -> [F,-,F,F,-,-,F,-,F]]
}

Iteraciones = 5
Angulo = 90°

Motivación

Desarrollar un pequeño programa para demostrar como se podría dibujar fractales del Sistema-L utilizando el lenguaje de programación funcional Haskell.

Conceptos Básicos

Sistema-L

Los Sistemas-L (Sistemas de Lindenmayer) son gramáticas formales que pueden ser utilizadas para modelar fractales y plantas.
Fueron introducidas y desarrolladas por el Biólogo y Botánico **Aristid Lindenmayer**.

Conceptos Básicos

Movimiento Tortuga

El Movimiento Tortuga es un sistema de graficación basado en la interpretación de símbolos o comandos.

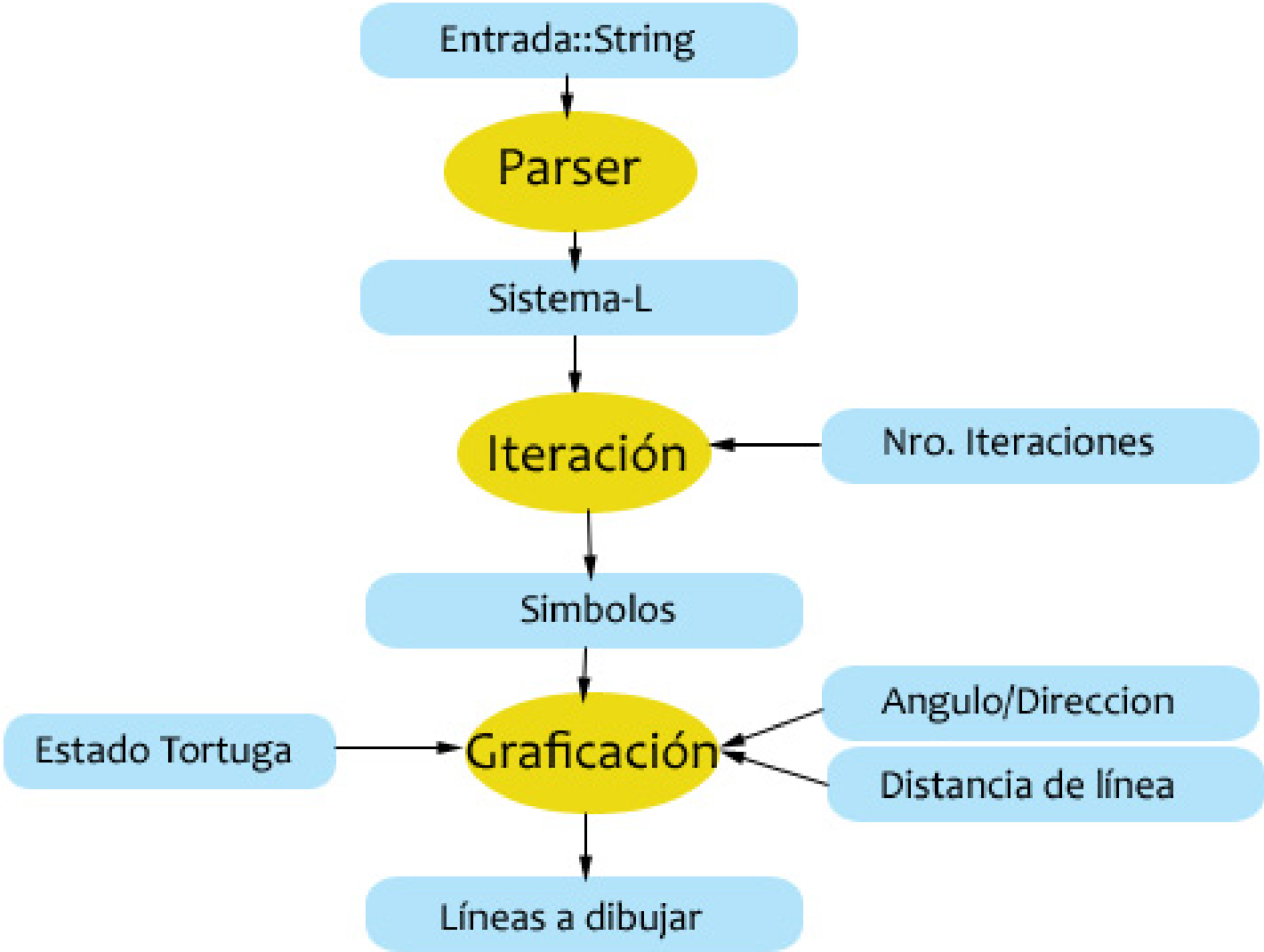
- Tiene un Estado = Posición, Ángulo y Distancia para avanzar.
- Algunos comandos son:
  - **F**, avanzar una Distancia en dirección de un Ángulo, y dibujar una linea.
  - **f**, avanzar una Distancia en dirección de un Ángulo, y NO dibujar una linea.
  - $+$, girar el ángulo hacia la derecha.
  - $-$, girar el ángulo hacia la izquierda.

Conceptos Básicos

Sistema-L
+
Movimiento Tortuga
= Imágenes de Fractales y Plantas

Desarrollo

Flujo de Datos Principal



Tipos de Datos

Sistema-L

CCBOL 2011

Sistema-L

```
Sierpinski gasket                    Nombre
   { alfabeto = [Fl,Fr,+,-]          Alfabeto
   ; inicio   = [Fr]                 Simbolos Iniclaes
   ; producciones
        = [ Fl -> [Fr,+,Fl,+,Fr]     Producciones
          ; Fr -> [Fl,-,Fr,-,Fl]
          ]
   }
```

Tipos de Datos

Sistema-L


```
DATA SistemaL
     | SistemaL nombre        : String
              alfabeto       : Alfabeto
              inicio         : Inicio
              producciones   : Producciones

DATA Simbolo
     | Simbolo String

TYPE Alfabeto       = [Simbolo]
TYPE Inicio         = [Simbolo]
TYPE Producciones   = [Produccion]
TYPE Produccion     = (Simbolo, Succesor)
TYPE Succesor       = [Simbolo]
```

Parser para Sistema-L


```
pSistemaL :: Parser SistemaL
pSistemaL = SistemaL <$> pNombre
                     <* pSymbol "{" <*> pAlfabeto
                     <* pSymbol ";" <*> pInicio
                     <* pSymbol ";" <*> pProducciones
                     <* pSymbol "}"
```

Parser para Sistema-L


```
pAlfabeto :: Parser Alfabeto
pAlfabeto = pKeyword "alfabeto"
              *> pSymbol "="
              *> pListaS1 (pSymbol ",") pSLSimbolo

pInicio :: Parser Inicio
pInicio = pKeyword "inicio"
              *> pSymbol "="
              *> pListaS1 (pSymbol ",") pSLSimbolo

pProducciones :: Parser Producciones
pProducciones = pKeyword "producciones"
                  *> pSymbol "="
                  *> pListaS1 (pSymbol ";") pProduccion

pProduccion :: Parser Produccion
pProduccion = (,) <$> pSLSimbolo
                  <* pSymbol "->"
                  <*> pListaS1 (pSymbol ",") pSLSimbolo
```

Iteraciones

Gramatica

```
Koch F
  { alfabeto  = [F, +, -]
  ; inicio    = [F,-,F,-,F,-,F]
  ; producciones
      = [ F -> [F,-,F,+,F,-,F,-,F] ]
  }

Angulo   = 90°
Distancia = 5px
```

Iteraciones

Koch F, N=1

Iteraciones

Inicio = F-F-F-F

N = 1

Gen = F-F+F-F-F-F-F+F-F-F-F-F+F-F-F-F-F+F-F-F

Iteraciones

Koch F, N=2

CCBOL 2011

Iteraciones

N = 2

Gen = F-F+F-F-F-F-F+F-F-F+F-F+F-F-F-F-F+
F-F-F-F-F+F-F-F-F-F+F-F-F-F-F+F-F-F ...

Iteraciones

Koch F, N=3

CCBOL 2011

Iteraciones

N = 3

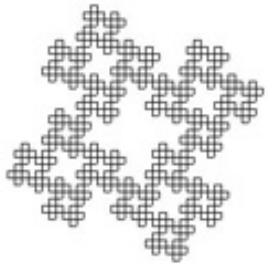
Gen = F-F+F-F-F-F-F+F-F-F+F-F+F-F-F-F-F+
F-F-F-F-F+F-F-F-F-F+F-F-F-F-F+F-F-F ...

Iteraciones

Koch F, N=4

CCBOL 2011

Iteraciones



N = 4

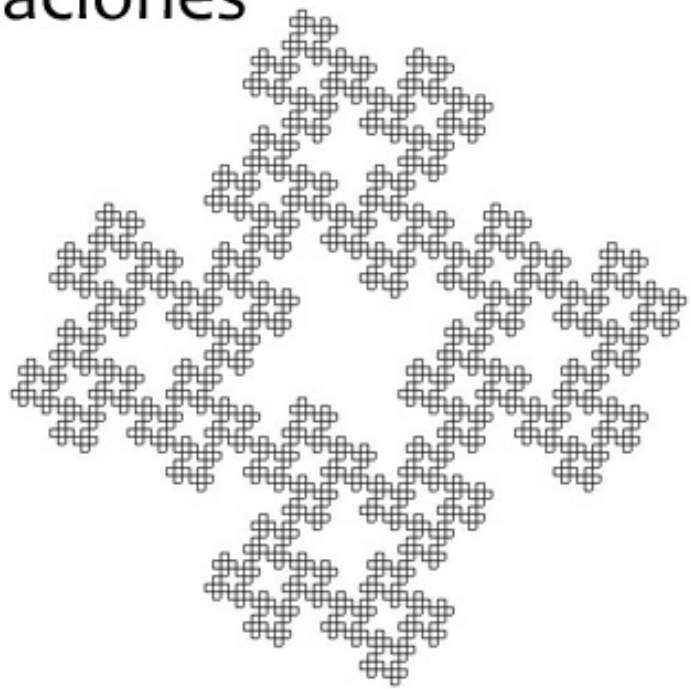
Gen = F-F+F-F-F-F-F+F-F-F+F-F+F-F-F-F-F+
F-F-F-F-F+F-F-F-F-F+F-F-F-F-F+F-F-F ...

Iteraciones

Koch F, N=5

CCBOL 2011

Iteraciones



N = 5

Gen = F-F+F-F-F-F-F+F-F-F-F-F+F-F-F-F-F+F ....

Iteraciones

Funcion para generar las iteraciones

Funcion para generar las iteraciones

```
generate :: SistemaL -> Int -> [Simbolo]
generate (SistemaL _ _ init prods) n
    = generate' init prods n

        where generate' seq prods 0
                  = seq
              generate' seq prods n
                  = let seq' = concat [ lst
                                      | s1 <- seq
                                      , (s2,lst) <- prods
                                      , s1 == s2
                                      ]
                    in generate' seq' prods (n-1)
```

Graficación

Tipos de Datos

Tipos de Datos Movimiento Tortuga

```
type Tortuga
    = ( Estado          -- estado actual de la tortuga
      , Distancia       -- distancia que que avanza la tortuga
      , Angulo          -- angulo con que gira
      , Color           -- color con el que dibuja
      )

type Distancia = Double
type Angulo    = Double
type PosX      = Double
type PosY      = Double

-- | Representa el estado actual de una tortuga
type Estado
    = ( PosX          -- su posicion x
      , PosY          -- su posicion y
      , Angulo        -- su angulo
      )
```

Graficación

Interpretación de Comandos

Interpretación de Comandos Tortuga

```
doCommand :: Simbolo -> Tortuga -> Points -> (Points,Tortuga)
doCommand simb state@((x,y,an),d,bn,c) points
  = let getCmd     (Simbolo str) = head str
    in case getCmd simb of
        'F' -> let (x', y') = ( x + d * cos an
                              , y + d * sin an)
                   ptline = ( (round x , round y)
                            , (round x', round y')
                            , c)
                   newState  = ((x', y', an), d, bn, c)
               in (ptline : points, newState)
        'f' -> let (x', y') = ( x + d * cos an
                              , y + d * sin an)
                   newState = ((x', y', an), d, bn, c)
               in (points, newState)
        '+' -> let newState = ((x, y, an + bn), d, bn, c)
               in (points, newState)
        '-' -> let newState = ((x, y, an - bn), d, bn, c)
               in (points, newState)
```

Graficación

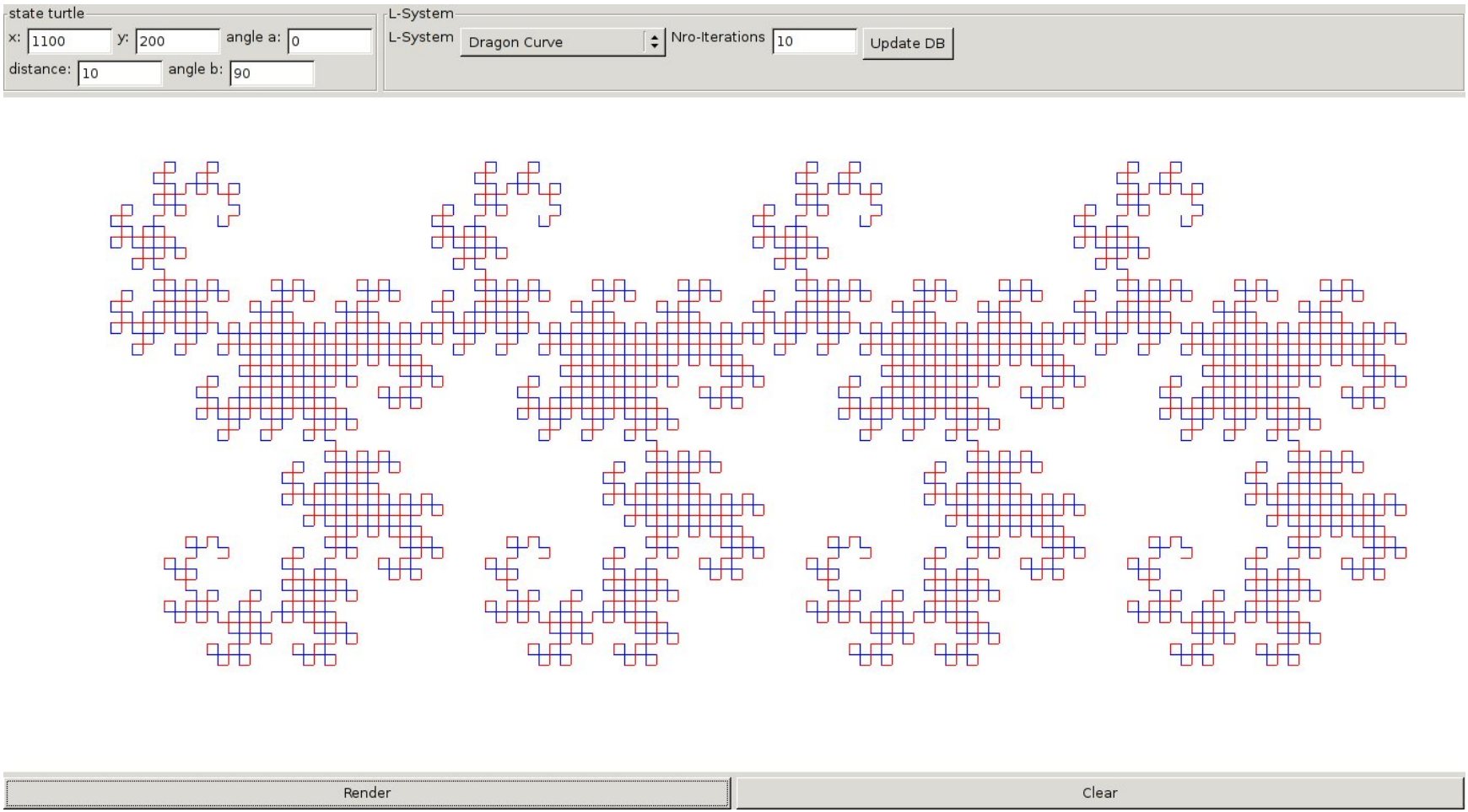
Interpretación de Comandos

Interpretación de Comandos Tortuga

```
processSequence :: [Simbolo] -> Tortuga -> Points -> Points
processSequence seq state points
    = let (newPoints,newState)
              = foldl funMix (points,state) seq
      in newPoints
    where funMix (points, state) cmd
              = doCommand cmd state points
```

Resultados

Ventana Principal



Funcionalidad

Principales Funcionalidades Implementadas

- Validación de Gramáticas
- Node Rewriting
- Edge Rewritting
- Graficación con Colores

OnLine

Puedes descargarlo del siguiente enlace:

```
http://hackage.haskell.org/package/lsystem
```

Conclusiones

Conclusión

Haskell , como lenguaje de programación, facilita el proceso de desarrollo de aplicaciones; De manera que el proceso sea rápido, comprensible, y en una cantidad relativamente pequeña de código.

Principales Ventajas

- La gramática del Sistema-L se implemento directamente en Haskell . Se utilizo la librería de parsers uu-parsinglib .
- Se utilizo la herramienta uuagc para la validación de gramáticas.
- Se utilizo la librería wxHaskell para la graficación y GUI.

Recomendaciones para trabajos futuros

Este proyecto es simplemente una pequeña demostración de lo que se puede hacer con Haskell . Es posible extenderlo de varias maneras, entre ellas:

- Dar soporte para la graficación de plantas
- Dar soporte para el graficado en 3 dimensiones
- Optimizar la forma de graficado.

Fin de la Presentación

Carlos Gomez
carliros.g@gmail.com