Grafiske brugergrænseflader i F#

Programmering og problemløsning

Jon Sporring

System.Timers

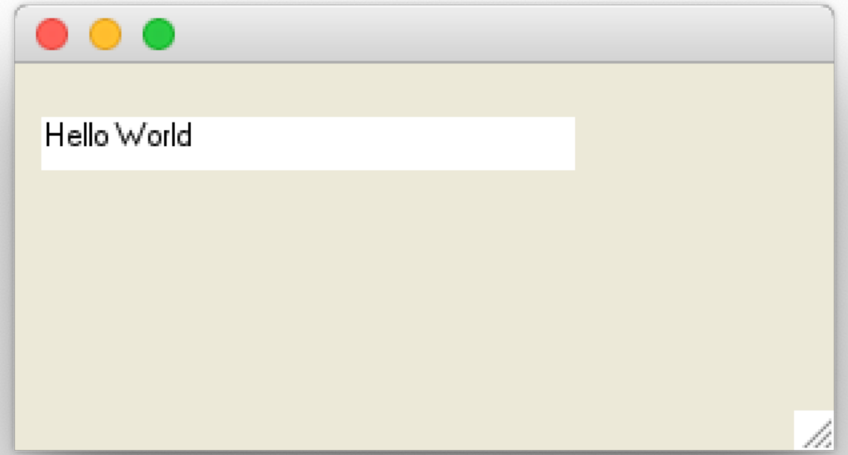
fsharpi

```
let t = new System.Timers.Timer()
t.Interval <- 1000.0
t.Elapsed.Add (fun e -> printfn "%s" (string System.DateTime.Now));;
t.Start();;
t.Stop();;
```

Skrive på skærmen

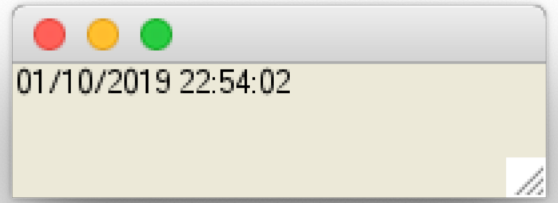
label.fsx

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 100)

// make a label to show time
let label = new Label()
win.Controls.Add label
label.Width <- 200
label.Location <- new Point (10, 20)
label.Text <- "Hello World"
label.BackColor <- Color.White
label.Height <- 20

Application.Run win // start event-loop
```



System.Windows.Forms

clock.fsx


01/10/2019 22:54:02

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 50)

// make a label to show time
let label = new Label()
win.Controls.Add label
label.Width <- 200
label.Text <- string System.DateTime.Now // get present time and date

// make a timer and link to label
let timer = new Timer()
timer.Interval <- 1000 // create an event every 1000 millisecond
timer.Enabled <- true // activate the timer
timer.Tick.Add (fun e -> label.Text <- string System.DateTime.Now)

Application.Run win // start event-loop
```

System.Windows.Forms

movingClock.fsx

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 50)

// make a label to show time
let label = new Label()
win.Controls.Add label
label.Text <- string System.DateTime.Now // get present time and date
let textSz = TextRenderer.MeasureText(label.Text,label.Font)
label.Width <- textSz.Width
label.Height <- textSz.Height
label.BackColor <- Color.White
```

System.Windows.Forms

movingClock.fsx

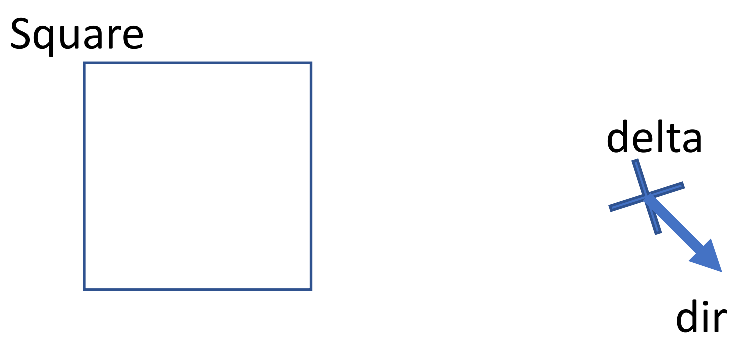
```
// make a timer and link to label
let timer = new Timer()
timer.Interval <- 100 // create an event every 1000 millisecond
timer.Enabled <- true // activiate the timer
let mutable pos = (0,0)
let mutable dir = (1,1)
let performTick (e : System.EventArgs) =
  printfn "%A %A" pos dir
  if fst pos + fst dir > win.ClientSize.Width - label.Width - 1
    || fst pos + fst dir < 0 then
     dir <- (-fst dir, snd dir);
  if snd pos + snd dir > win.ClientSize.Height - label.Height – 1
    || snd pos + snd dir < 0 then
     dir <- (fst dir, -snd dir);
  pos <- (fst pos + fst dir, snd pos + snd dir)
  label.Location <- Point (fst pos, snd pos);
  label.Text <- string System.DateTime.Now
timer.Tick.Add performTick

Application.Run win // start event-loop
```

System.Windows.Forms

movingSquare.fsx

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 50)

. . .

// make a timer
let timer = new Timer()
timer.Interval <- 10 // create an event every 10 millisecond
timer.Enabled <- true // activiate the timer
timer.Tick.Add (fun e -> win.Refresh())

Application.Run win // start event-loop
```

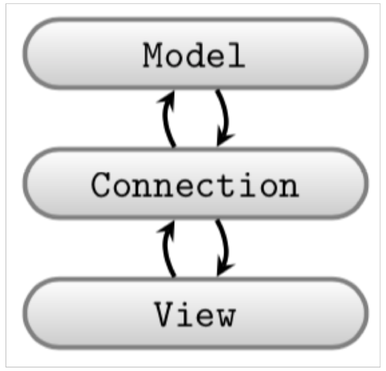
System.Windows.Forms



movingSquare.fsx

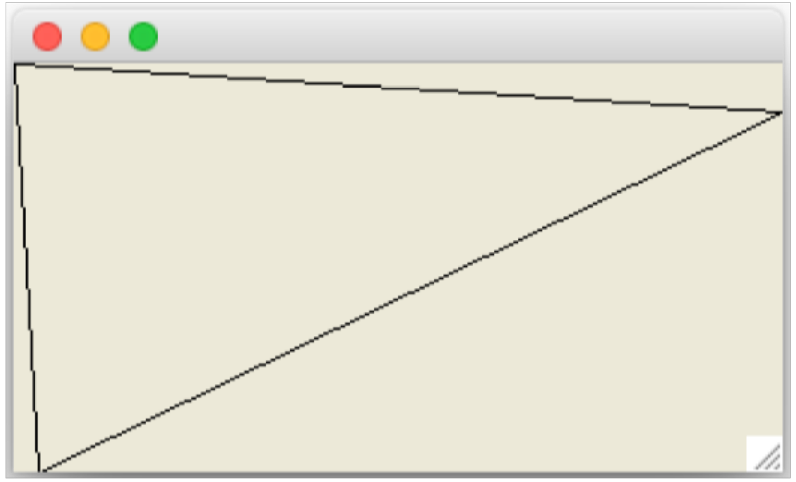
```
let mutable delta = Point (0,0)
let mutable dir = Point (1,1)
let polygonSz = Point (10,10);
let polygon = [|Point (0,0); Point (polygonSz.X,0); polygonSz; Point
(0,polygonSz.Y); Point (0,0)|]
let paint (e : PaintEventArgs) : unit =
  let pen = new Pen (Color.Black)
  if delta.X + dir.X < 0
    || delta.X + dir.X + polygonSz.X > win.ClientSize.Width – 1 then
      dir <- Point (-dir.X, dir.Y);
  if delta.Y + dir.Y < 0
    || delta.Y + dir.Y + polygonSz.Y > win.ClientSize.Height – 1 then
      dir <- Point (dir.X, -dir.Y);
  delta <- Point (delta.X + dir.X, delta.Y + dir.Y)
  let add (p : Point) -> Point (p.X + delta.X, p.Y + delta.Y)
  let points = Array.map add polygon
  e.Graphics.DrawLines (pen, points)
win.Paint.Add paint
```

Adskil model og view



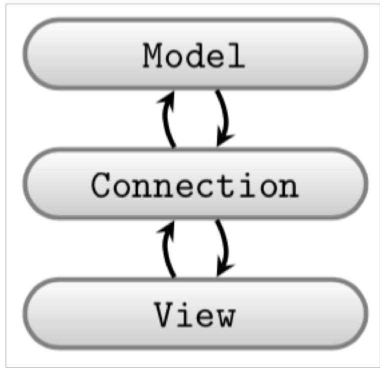
triangleClientSize.fsx

```
// Open often used libraries, beware of namespace pollution!
open System.Windows.Forms
open System.Drawing

// Prepare window form
let win = new Form ()
win.ClientSize <- Size (320, 170)

// Set paint call-back function
let paint (e : PaintEventArgs) : unit =
  let pen = new Pen (Color.Black)
  let points =
    [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
  e.Graphics.DrawLines (pen, points)
win.Paint.Add paint

Application.Run win // Start the event-loop.
```

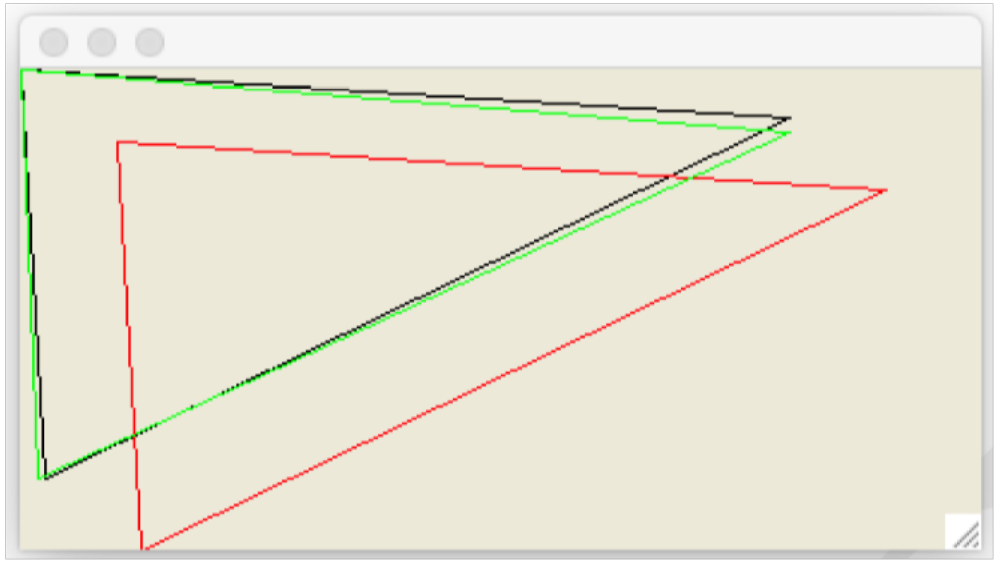


triangleOrganied.fsx

```
open System.Windows.Forms
open System.Drawing

////////////// WinForm specifics //////////////
/// Setup a window form and return function which can activate it
let view (sz : Size) (pen : Pen) (pts : Point []) : (unit -> unit) =
  let win = new Form ()
  win.ClientSize <- sz
  win.Paint.Add (fun e -> e.Graphics.DrawLines (pen, pts))
  fun () -> Application.Run win // function as return value

////////////// Model //////////////
// A black triangle, using winform primitives for brevity
let model () : Size * Pen * (Point []) =
  let size = Size (320, 170)
  let pen = new Pen (Color.FromArgb (0, 0, 0))
  let lines =
    [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
  (size, pen, lines)

////////////// Connection //////////////
// Tie view and model together and enter main event loop
let (size, pen, lines) = model ()
let run = view size pen lines
run ()
```



Tegn og transformer mange linjer



$$(a,b) = (x + \Delta x, y + \Delta y)$$

$$(a,b) = (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$$

$$(a,b) = (sx, sy)$$

Model **transformWindows.fsx**

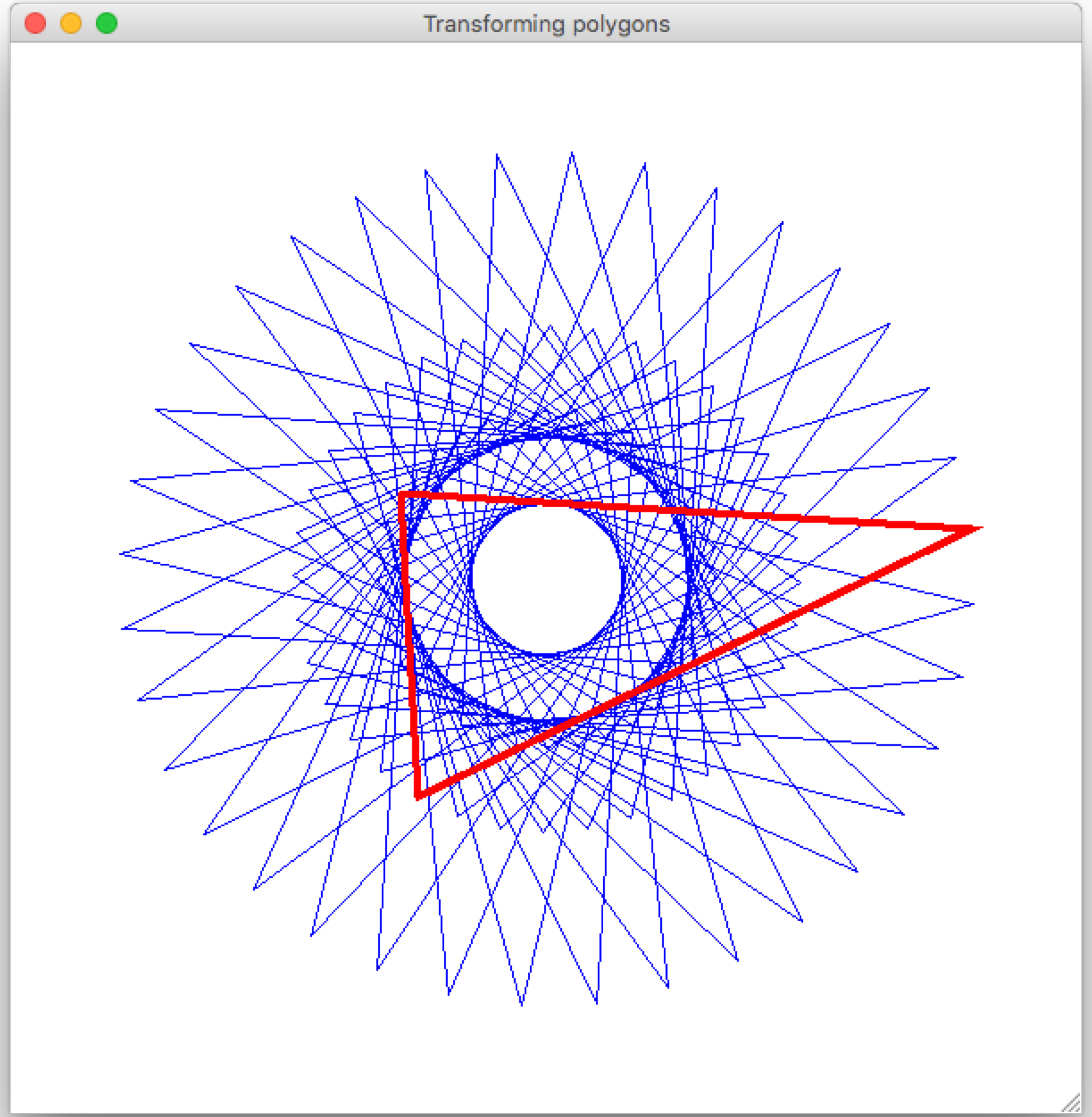
```
///////////// Model /////////////
// A black triangle, using WinForm primitives for brevity
let model () : Size * ((Pen * (Point [])) list) =
  /// Translate a primitive
  let translate (d : Point) (arr : Point []) : Point [] =
    let add (d : Point) (p : Point) : Point =
      Point (d.X + p.X, d.Y + p.Y)
    Array.map (add d) arr

  /// Rotate a primitive
  let rotate (theta : float) (arr : Point []) : Point [] =
    let toInt = int << round
    let rot (t : float) (p : Point) : Point =
      let (x, y) = (float p.X, float p.Y)
      let (a, b) = (x * cos t - y * sin t, x * sin t + y * cos t)
      Point (toInt a, toInt b)
    Array.map (rot theta) arr

  let size = Size (400, 200)
  let lines =
    [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
  let black = new Pen (Color.FromArgb (0, 0, 0))
  let red = new Pen (Color.FromArgb (255, 0, 0))
  let green = new Pen (Color.FromArgb (0, 255, 0))
  let shapes =
    [(black, lines);
     (red, translate (Point (40, 30)) lines);
     (green, rotate (1.0 *System.Math.PI / 180.0) lines)]
  (size, shapes)
```

View+forbindelse **transformWindows.fsx**

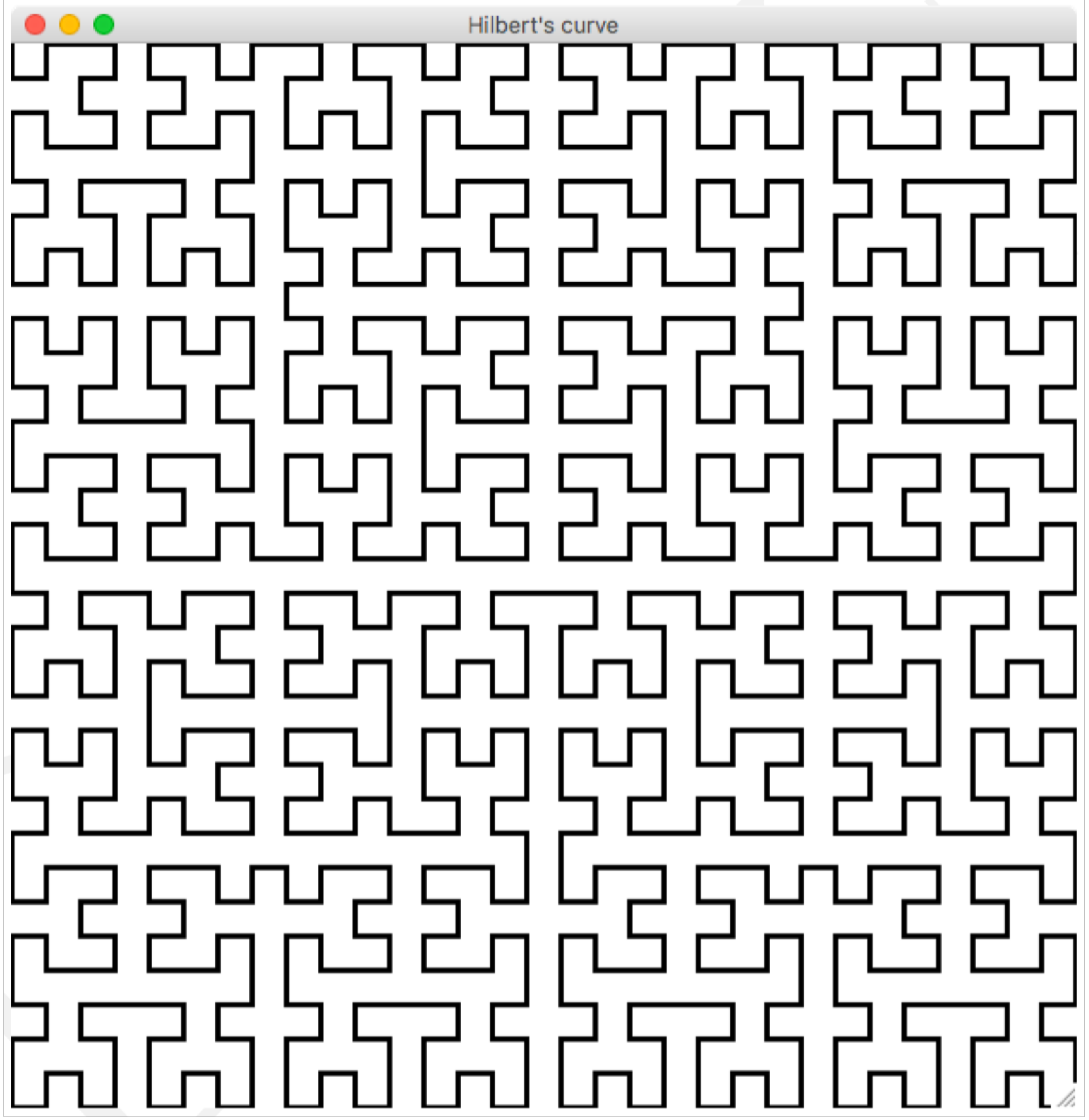
```
// Open often used libraries, beware of namespace polution!
open System.Windows.Forms
open System.Drawing

////////////// WinForm specifics //////////////
/// Setup a window form and return function to activate
let view (sz : Size) (shapes : (Pen * (Point [])) list) : (unit -> unit) =
  let win = new Form ()
  win.ClientSize <- sz
  let paint (e : PaintEventArgs) ((p, pts) : (Pen * (Point []))) : unit =
    e.Graphics.DrawLines (p, pts)
  win.Paint.Add (fun e -> List.iter (paint e) shapes)
  fun () -> Application.Run win // function as return value
```

```
////////////// Connection ///////////////
// Tie view and model together and enter main event loop
let (size, shapes) = model ()
let run = view size shapes
run ()
```

rotationalSymmetry.fsx



hilbert.fsx


Hilbert's curve