Grafiske brugergrænseflader i F#

Programmering og problemløsning

Jon Sporring

Simpel grafik med System.Drawing: Farver

Method/Property	Description
`A`	Get the value of the alpha channel of a color.
`B`	Get the value of the blue channel of a color.
`Black`	Get a predefined color with ARGB value of `0xFF000000`.
`Blue`	Get a predefined color with ARGB value of `0xFF0000FF`.
`Brown`	Get a predefined color with ARGB value of `0xFFA52A2A`.

`FromArgb : int -> Color`
`FromArgb : int*int*int -> Color`
`FromArgb : int*int*int*int -> Color`
	Create a color structure.

`G`	Get the value of the green channel of a color.
`Gray`	Get a predefined color with ARGB value of `0xFF808080`.
`Green`	Get a predefined color with ARGB value of `0xFF00FF00`.
`Orange`	Get a predefined color with ARGB value of `0xFFFFA500`.
`Purple`	Get a predefined color with ARGB value of `0xFF800080`.
`R`	Get the value of the red channel of a color.
`Red`	Get a predefined color with ARGB value of `0xFFFF0000`.
`ToArgb : Color -> int`	Get the 32-bit integer representation of a color.
`White`	Get a predefined color with ARGB value of `0xFFFFFFFF`.
`Yellow`	Get a predefined color with ARGB value of `0xFFFFFF00`.

https://msdn.microsoft.com/en-us/library/system.drawing.color(v=vs.110).aspx

System.Drawing: Andre structurer

Constructor	Description
`Bitmap(int, int)`	Create a new empty `Image` of specified size.
`Bitmap(Stream)` `Bitmap(string)`	Create a `Image` from a `System.IO.Stream` or from a file specified by a filename.
`Font(string, single)`	Create a new font from the font's name and em-size.

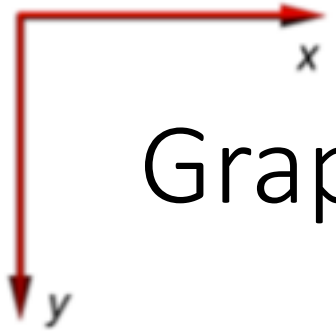
`Pen(Brush)`
`Pen(Brush), single)`
`Pen(Color)`
`Pen(Color, single)`
	Create a pen to paint either with a brush or solid color and possibly with specified width.

`Point(int, int)`
`Point(Size)`
`PointF(single, single)`
	Create an ordered pair of integers or singles specifying x- and y-coordinates in the plane.

`Size(int, int)`
`Size(Point)`
`SizeF(single, single)`
`SizeF(PointF)`
	Create an ordered pair of integers or singles specifying height and width in the plane.

`SolidBrush(Color)` `TextureBrush(Image)`	Create a `Brush` as a solid color or from an image to fill the interior of a geometric shapes.

https://msdn.microsoft.com/en-us/library/system.drawing(v=vs.110).aspx



Graphics metoder

Constructor	Description
`DrawImage : Image * (Point []) -> unit` `DrawImage : Image * (PointF []) -> unit`	Draw an image at a specific point and size.
`DrawImage : Image * Point -> unit` `DrawImage : Image * PointF -> unit`	Draw an image at a specific point.
`DrawLines : Pen * (Point []) -> unit` `DrawLines : Pen * (PointF []) -> unit`	Draw a series of lines between the n'th and the $n+1$'th points.
`DrawString :` `  string * Font * Brush * PointF -> unit`	Draw a string at the specified point.

https://msdn.microsoft.com/en-us/library/system.drawing.graphics_methods(v=vs.110).aspx

My first drawing

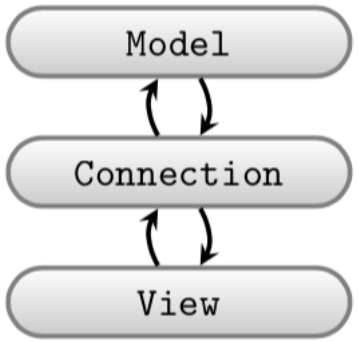
Listing 23.4 winforms/triangle.fsx:
Adding line graphics to a window. See Figure 23.5

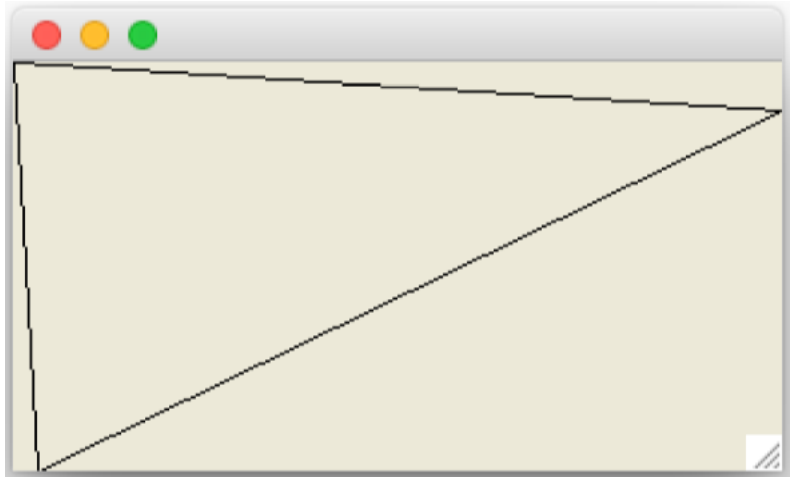
```
1  // Open often used libraries, be ware of namespace polution!
2  open System.Windows.Forms
3  open System.Drawing
4
5  // Prepare window form
6  let win = new Form ()
7  win.Size <- Size (320, 170)
8
9  // Set paint call-back function
10 let paint (e : PaintEventArgs) : unit =
11   let pen = new Pen (Color.Black)
12   let points =
13     [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
14   e.Graphics.DrawLines (pen, points)
15 win.Paint.Add paint
16
17 // Start the event-loop.
18 Application.Run win // Start the event-loop.
```

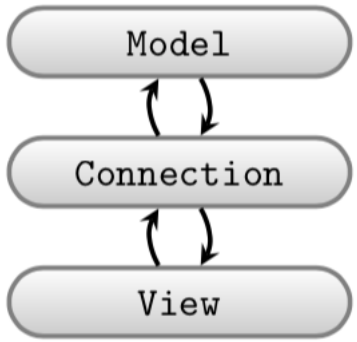
Winforms, Gtk#, Cocoa: Adskil model og view

Listing 23.5 winforms/triangleClientSize.fsx:
Adding line graphics to a window. See Figure 23.6.

```
1  // Open often used libraries, be ware of namespace polution!
2  open System.Windows.Forms
3  open System.Drawing
4
5  // Prepare window form
6  let win = new Form ()
7  win.ClientSize <- Size (320, 170)
8
9  // Set paint call-back function
10 let paint (e : PaintEventArgs) : unit =
11   let pen = new Pen (Color.Black)
12   let points =
13     [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
14   e.Graphics.DrawLines (pen, points)
15 win.Paint.Add paint
16
17 // Start the event-loop.
18 Application.Run win // Start the event-loop.
```



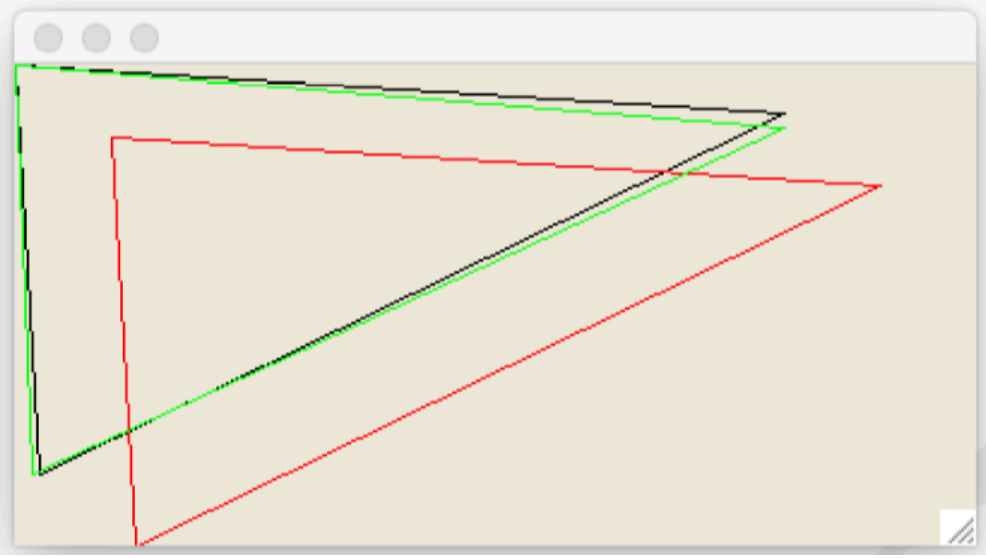




Listing 23.6 winforms/triangleOrganized.fsx:
Improved organization of code for drawing a triangle. See Figure 23.8.

```
1  // Open often used libraries, be ware of namespace polution!
2  open System.Windows.Forms
3  open System.Drawing
4
5  ////////////// WinForm specifics //////////////
6  /// Setup a window form and return function to activate
7  let view (sz : Size) (pen : Pen) (pts : Point []) : (unit -> unit) =
8    let win = new System.Windows.Forms.Form ()
9    win.ClientSize <- sz
10   win.Paint.Add (fun e -> e.Graphics.DrawLines (pen, pts))
11   fun () -> Application.Run win // function as return value
12
13 ////////////// Model //////////////
14 // A black triangle, using winform primitives for brevity
15 let model () : Size * Pen * (Point []) =
16   let size = Size (320, 170)
17   let pen = new Pen (Color.FromArgb (0, 0, 0))
18   let lines =
19     [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
20   (size, pen, lines)
21
22 ////////////// Connection //////////////
23 // Tie view and model together and enter main event loop
24 let (size, pen, lines) = model ()
25 let run = view size pen lines
26 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

Listing 23.7 winforms/transformWindows.fsx:
Model of a triangle and simple transformations of it. See also Listing 23.8 and 23.9, and Figure 23.9.

```
15 ////////////// Model //////////////
16 // A black triangle, using winform primitives for brevity
17 let model () : Size * ((Pen * (Point [])) list) =
18   /// Translate a primitive
19   let translate (d : Point) (arr : Point []) : Point [] =
20     let add (d : Point) (p : Point) : Point =
21       Point (d.X + p.X, d.Y + p.Y)
22     Array.map (add d) arr
23
24   /// Rotate a primitive
25   let rotate (theta : float) (arr : Point []) : Point [] =
26     let toInt = int << round
27     let rot (t : float) (p : Point) : Point =
28       let (x, y) = (float p.X, float p.Y)
29       let (a, b) = (x * cos t - y * sin t, x * sin t + y * cos t)
30       Point (toInt a, toInt b)
31     Array.map (rot theta) arr
32
33   let size = Size (400, 200)
34   let lines =
35     [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
36   let black = new Pen (Color.FromArgb (0, 0, 0))
37   let red = new Pen (Color.FromArgb (255, 0, 0))
38   let green = new Pen (Color.FromArgb (0, 255, 0))
39   let shapes =
40     [(black, lines);
41      (red, translate (Point (40, 30)) lines);
42      (green, rotate (1.0 *System.Math.PI / 180.0) lines)]
43   (size, shapes)
```

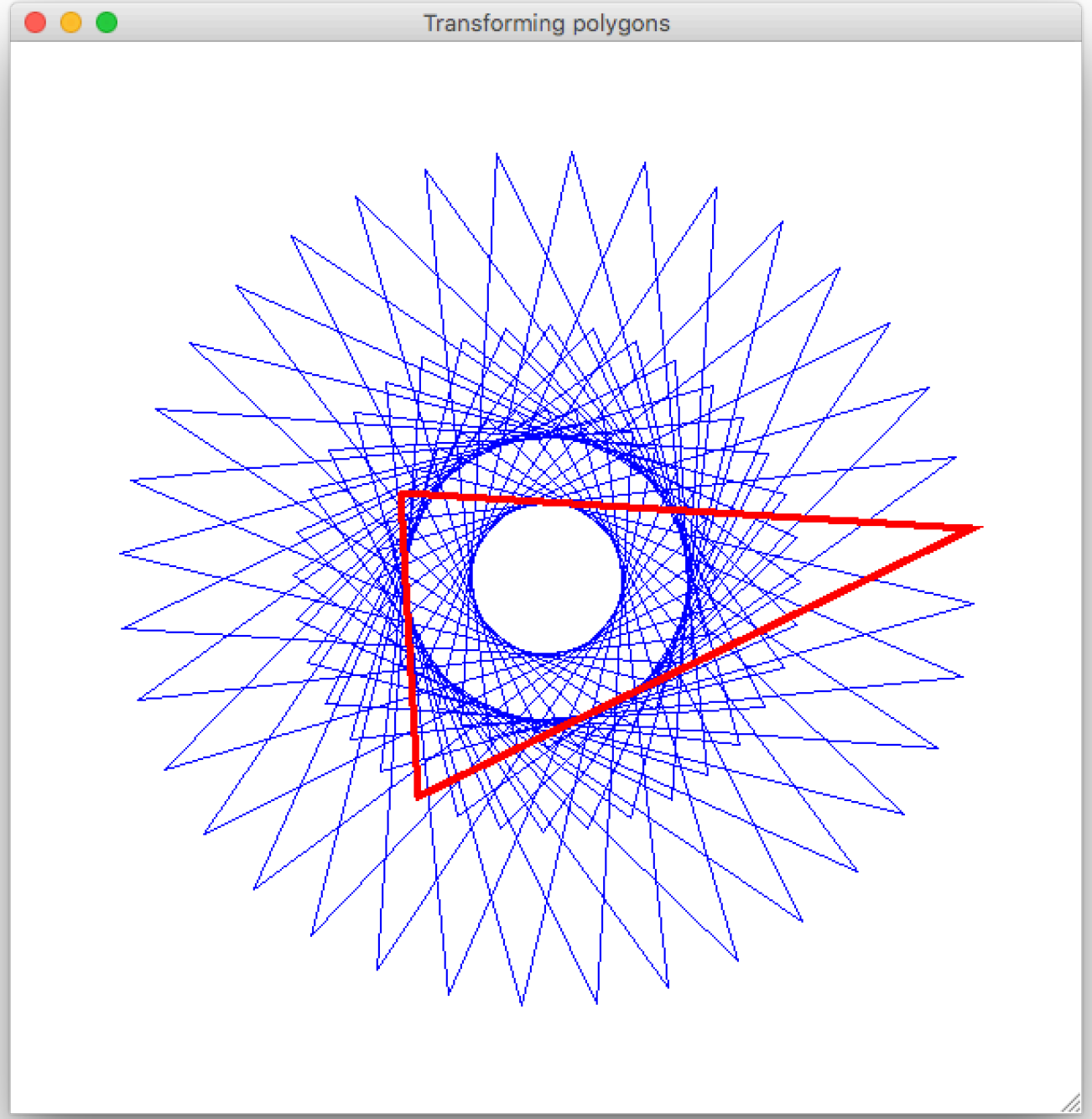
View+control

Listing 23.9 winforms/transformWindows.fsx:
Model of a triangle and simple transformations of it. See also Listing 23.7 and 23.8, and Figure 23.9.

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

Listing 23.8 winforms/transformV
View of lists of pairs of pen and
Figure 23.9.

```
1  // Open often used libraries, be ware of namespace polution!
2  open System.Windows.Forms
3  open System.Drawing
4
5  ////////////// WinForm specifics //////////////
6  /// Setup a window form and return function to activate
7  let view (sz : Size) (shapes : (Pen * (Point [])) list) : (unit -> unit)
       =
8    let win = new System.Windows.Forms.Form ()
9    win.ClientSize <- sz
10   let paint (e : PaintEventArgs) ((p, pts) : (Pen * (Point []))) : unit =
11     e.Graphics.DrawLines (p, pts)
12   win.Paint.Add (fun e -> List.iter (paint e) shapes)
13   fun () -> Application.Run win // function as return value
```


Transforming polygons

Event-styret programmering

Window callback	Kaldes når brugeren ….
Move	flytter vinduet
Resize	ændrer vinduets størrelse
MouseMove	flytter musen i forhold til et aktivt vindue
MouseDown	trykker venstre musetast ned
MouseUp	slipper venstre musetast
MouseClick	klikker i et vindue
KeyPress	trykker på en tast

windowEvents.fsx

DateTime Class

Properties	Description
Day	The day of month as a number (0..31)
Hour	The hour in (0..23)
Millisecond	The millisecond as a number (0..999)
Minute	The minute (0..59)
Month	The month as a number (1..12)
Now	Return a new DateTime instance of current time and date
Year	The year (1..9999)

```
let now = System.DateTime.Now
printf "%d/%d/%d" now.Year now.Month now.Day
printfn " %d:%02d:%02d:%03d" now.Hour now.Minute now.Second now.Millisecond;;
```

Stopur (Timer Class)

```
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();;
```

Stopur og WinForms

cl

Listing 23.13 winforms/clock.fsx:
Using `System.Windows.Forms.Timer` and `System.DateTime.Now` to update the display of the present date and time. See Figure 23.11 for the result.

Refresh

```
18 timer.Tick.Add (fun e -> label.Text <- string System.DateTime.Now)
19
20 Application.Run win // start event-loop
```

Input fra brugeren via Controls

Listing 23.14 winforms/buttonControl.fsx:
Create the button and an event, see also Figure 23.12.

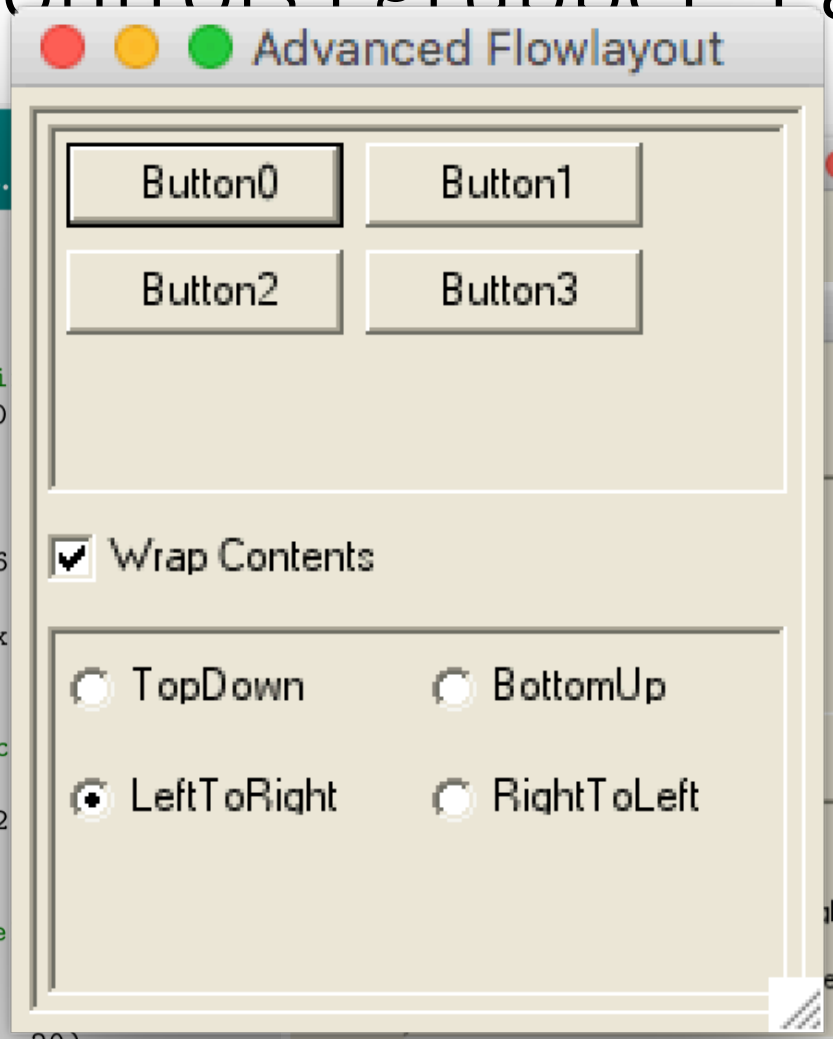
```
1  open System.Windows.Forms
2  open System.Drawing
3  open System
4
5  let win = new Form () // make a window form
6  win.ClientSize <- Size (140, 120)
7
8  // Create a label
9  let label = new Label()
10 win.Controls.Add label
11 label.Location <- new Point (20, 20)
12 label.Width <- 120
13 let mutable clicked = 0
14 let setLabel clicked =
15   label.Text <- sprintf "Clicked %d times" clicked
16 setLabel clicked
17
18 // Create a button
19 let button = new Button ()
20 win.Controls.Add button
21 button.Size <- new Size (100, 40)
22 button.Location <- new Point (20, 60)
23 button.Text <- "Click me"
24 button.Click.Add (fun e -> clicked <- clicked + 1; setLabel clicked)
25
26 Application.Run win // Start the event-loop.
```

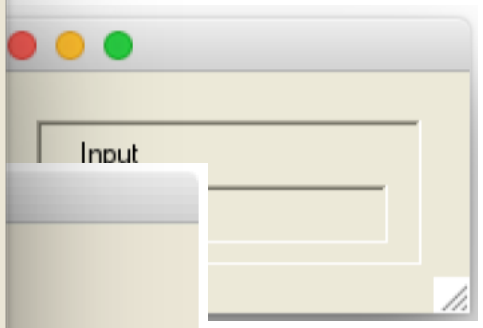
buttonControl.fsx

Organisering af Controls i grupper: Panels

Listing 23.18 winforms/panel.fsx:
Create a panel, label, text input controls.

```
1  open System.Drawing
2  open System.Windows.Forms
3
4  let win = new Form () // Create a wi
5  win.ClientSize <- new Size (200, 100
6
7  // Customize the Panel control
8  let panel = new Panel ()
9  panel.ClientSize <- new Size (160, 6
10 panel.Location <- new Point (20,20)
11 panel.BorderStyle <- BorderStyle.Fix
12 win.Controls.Add panel // Add panel
13
14 // Customize the Label and TextBox c
15 let label = new Label ()
16 label.ClientSize <- new Size (120, 2
17 label.Location <- new Point (15,5)
18 label.Text <- "Input"
19 panel.Controls.Add label // add labe
20
21 let textBox = new TextBox ()
22 textBox.ClientSize <- new Size (120, 20)
23 textBox.Location <- new Point (20,25)
24 textBox.Text <- "Initial text"
25 panel.Controls.Add textBox // add textbox to panel
26
27 Application.Run win // Start the event-loop
```





[flowLayoutPanel.fsx]

[flowLayoutPanelAdvanced.fsx]

Hilberts kurve

Problem 23.1

Consider a curve consisting of piecewise straight lines all with the same length but with varying angles 0°, 90°, 180°, or 270° w.r.t. the horizontal axis. To draw this curve we need 3 basic operations: Move forward (F), turn right (R), and turn left (L). The turning is w.r.t. the present direction. A Hilbert Curve is a space-filling curve, which can be expressed recursively as:

$$A \to LBFRAFARFBL \tag{23.1}$$
$$B \to RAFLBFBLFAR \tag{23.2}$$

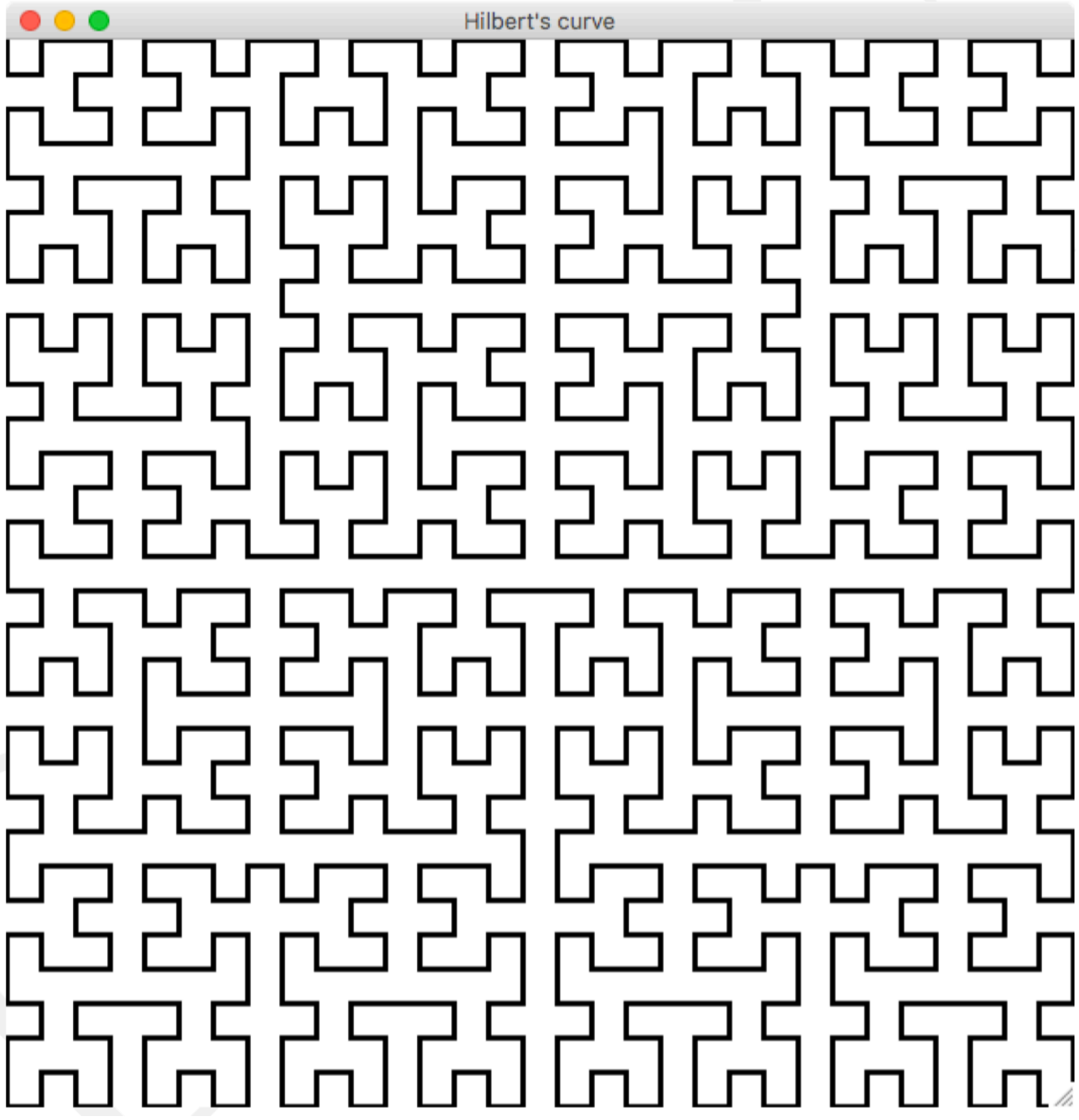
starting with A. For practical illustrations, we typically only draw space filling curves to a specified depth of recursion, which is called the order of the curve. Hence, to draw a first order curve, we don't recurs at all, i.e., ignore all occurrences of the symbols A and B on the right-hand-side of (23.1), and get

$$A \to LFRFRFL.$$

For the second order curve, we recurse once, i.e.,

$$\begin{aligned} A \to & LBFRAFARFBL \\ \to & L(RAFLBFBLFAR)F \\ & R(LBFRAFARFBL)F(LBFRAFARFBL) \\ & RF(RAFLBFBLFAR)L \\ \to & LRFLFLFRFRLFRFRFLFLFRFRFLRFRFLFLFRL. \end{aligned}$$

Since $LR = RL = \emptyset$ then the above simplifies to $FLFLFRFFRFRFLFLFRFRFFRFLFLF$. Make a program, that given an order produces an image of the Hilbert curve.


Hilbert's curve