Vg
Declarative 2D
vector graphics for OCaml

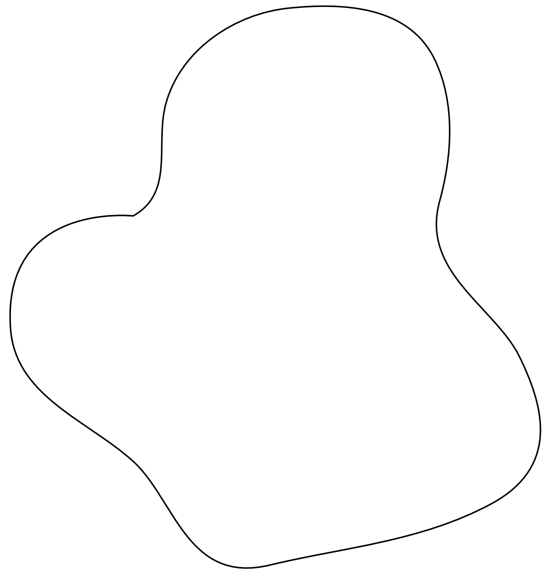
Daniel Bünzli, independent software engineer
http://erratique.ch

2013

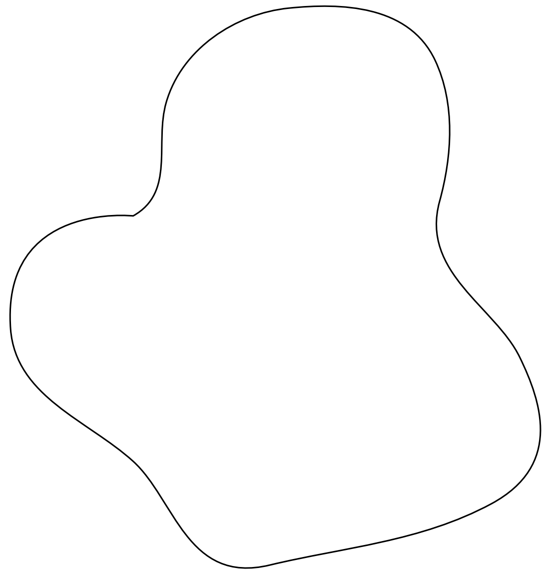
Vectors
vs.
Raster

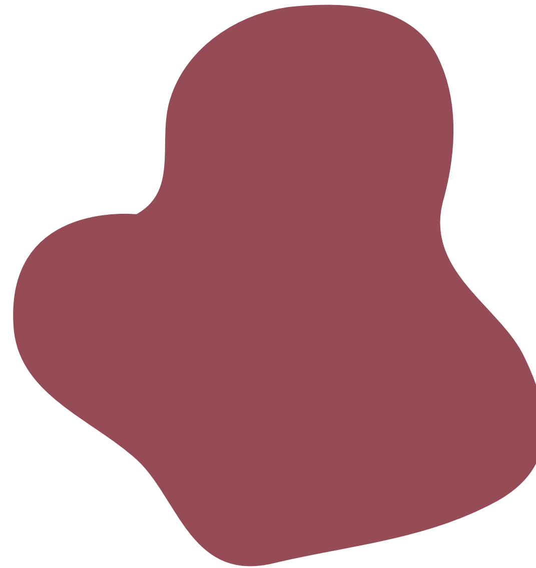
Declarative
vs.
Imperative

Painter model

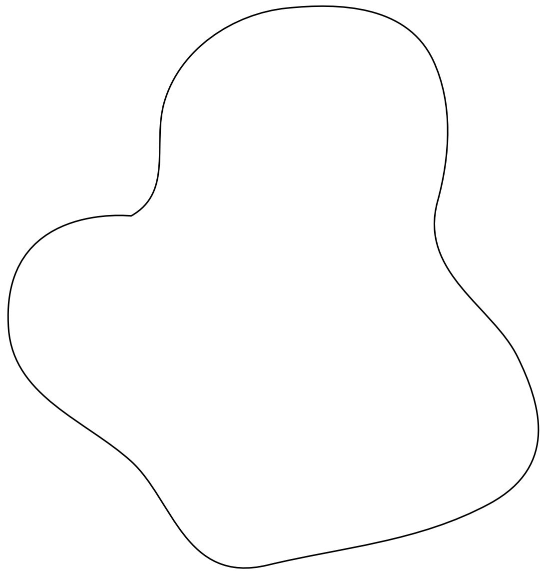


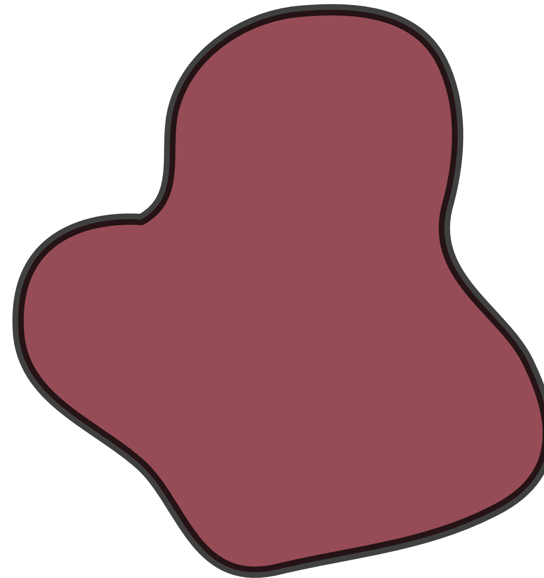
Painter model



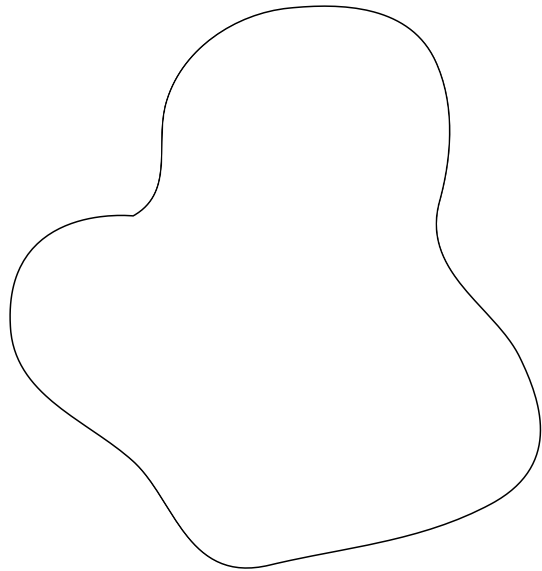


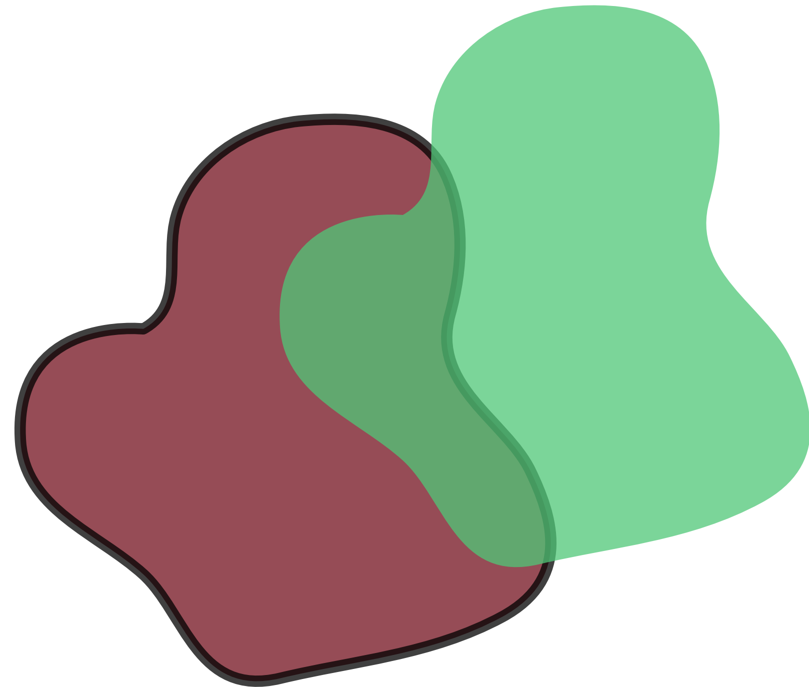
Painter model



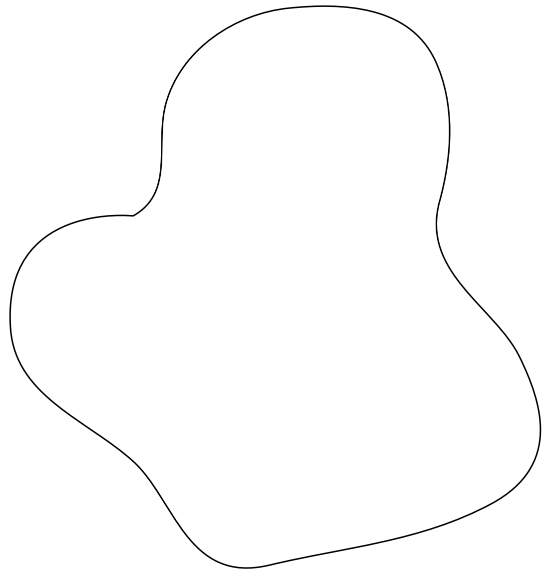


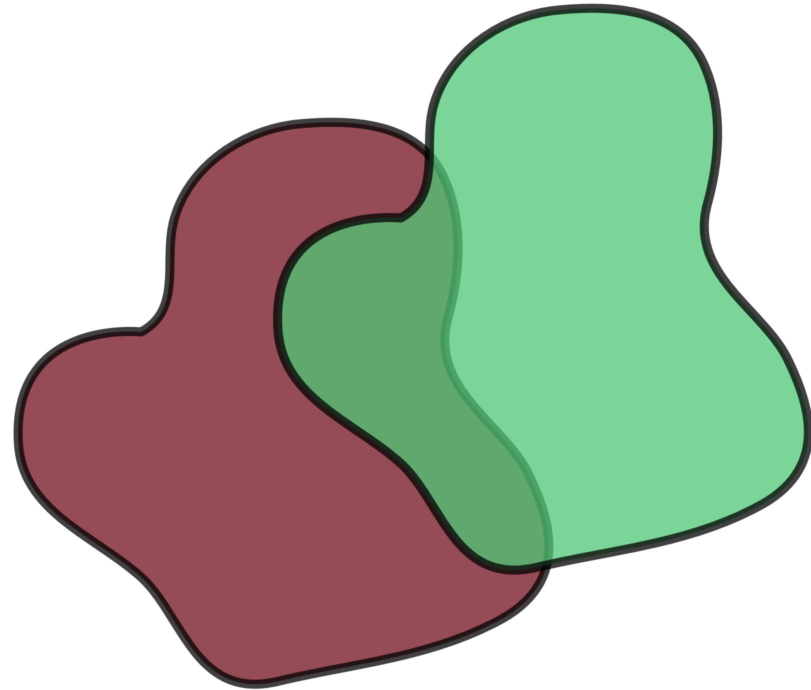
Painter model



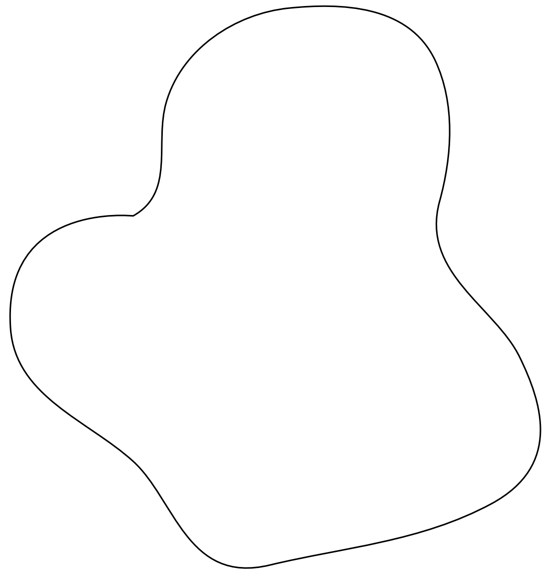


Painter model

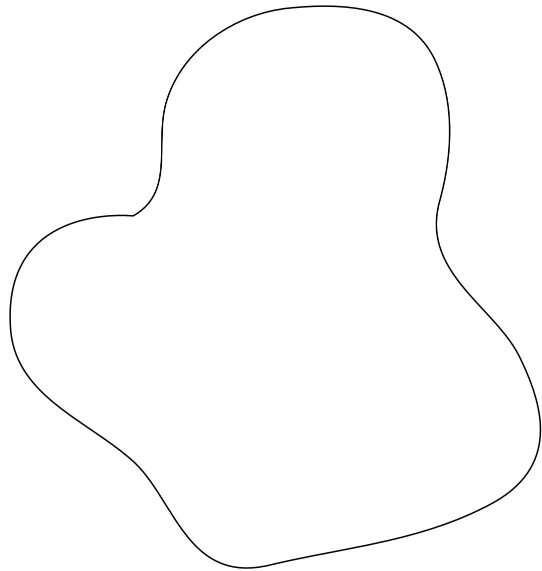




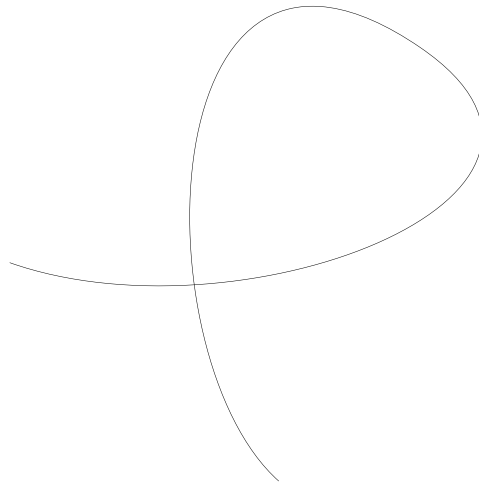
Painter model



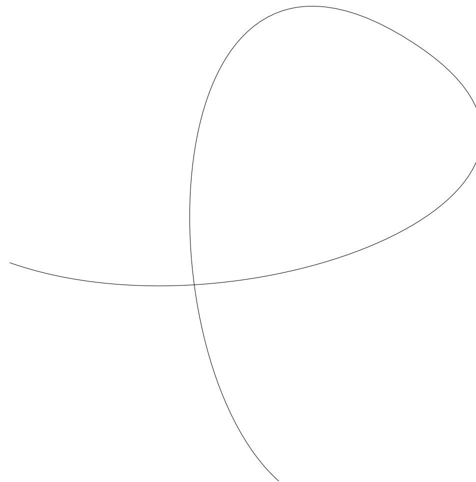
Painter model



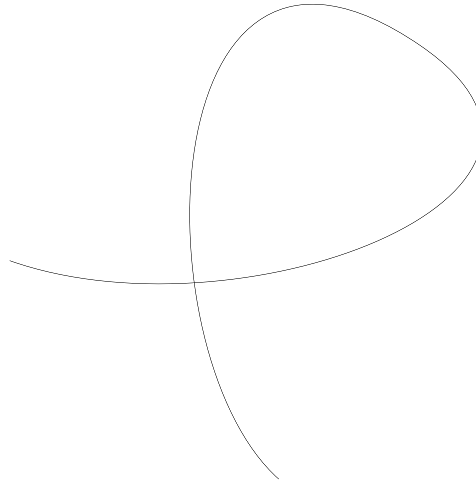
Stroke ?



Stroke ?



Stroke ?



Collage model

Infinite image

```
type image ≈ pt -> color
```

Constant image

```
let black ≈ fun _ -> Color.black
```

Constant image

```
let black ≈ fun _ -> Color.black

let black = I.const Color.black
```

Constant image

```
let black ≈ fun _ -> Color.black

let black = I.const Color.black

I.const : color -> image
```

Constant image

```
let black ≈ fun _ -> Color.black

let black = I.const Color.black

I.const : color -> image

I.const c ≈ fun _ -> c
```

`I.const Color.black`

```
I.const (Color.v_srgb 0.608 0.067 0.118)
```

```
I.const (Color.v_srgb 0.314 0.784 0.471)
```

```
I.const (Color.v_srgb 0.000 0.439 0.722)
```

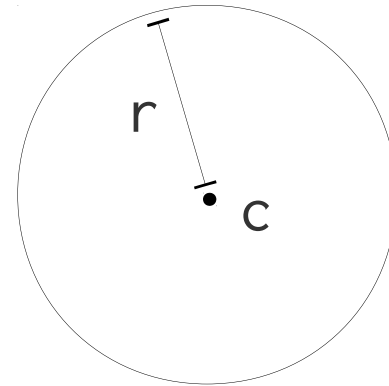
I.const **Color**.white

Areas as paths

```
type path ≈ pt -> bool
```

Circle path

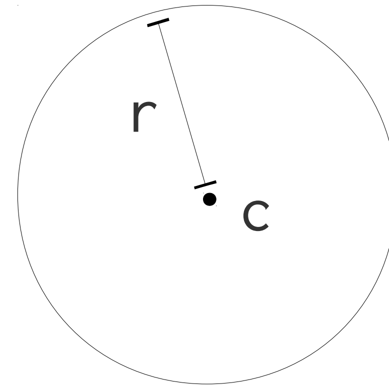
```
let c, r = P2.o, 1.
```




```
let circle ≈ fun pt -> V2.norm (pt - c) < r
```

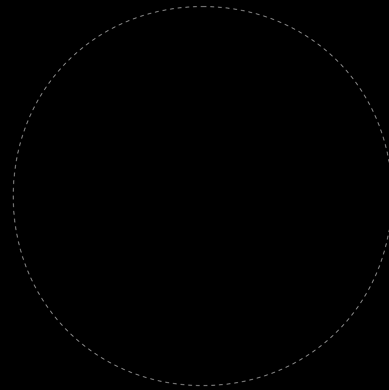
Circle path

```
let c, r = P2.o, 1.
```

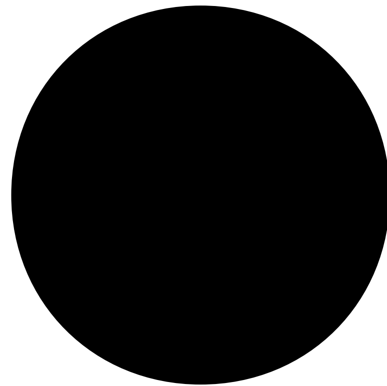



```
let circle ≈ fun pt -> V2.norm (pt - c) < r

let circle = P.empty >> P.circle c r
```

I.const **Color**.black



`I.const Color.black`



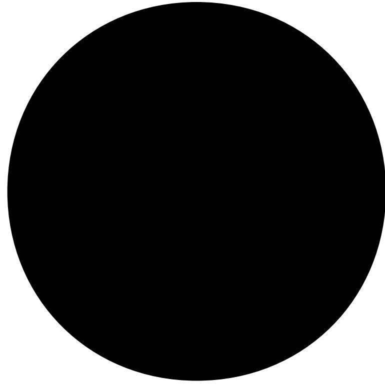
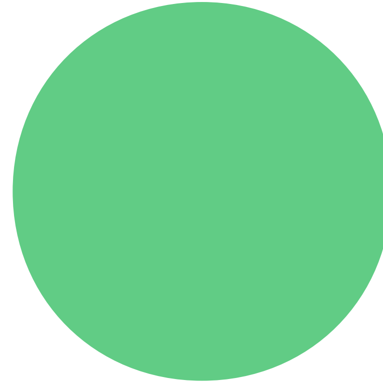
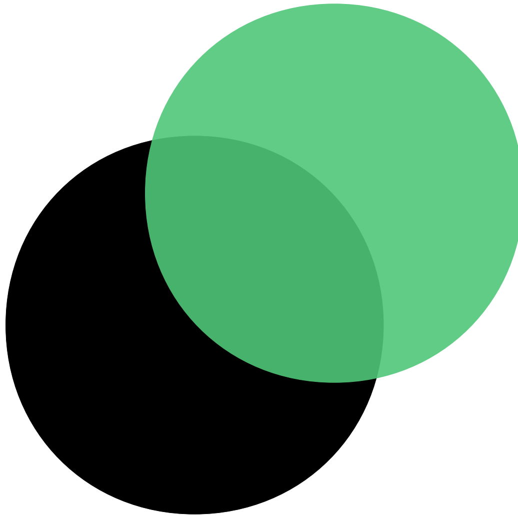
```
I.const Color.black >> I.cut circle
```

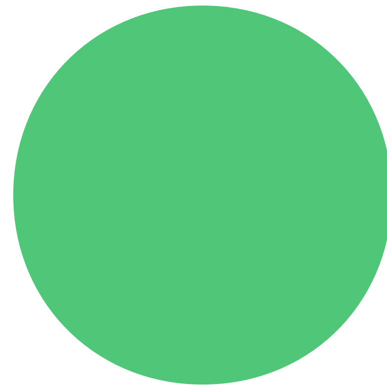
Cut !

```
I.cut : path -> image -> image
```

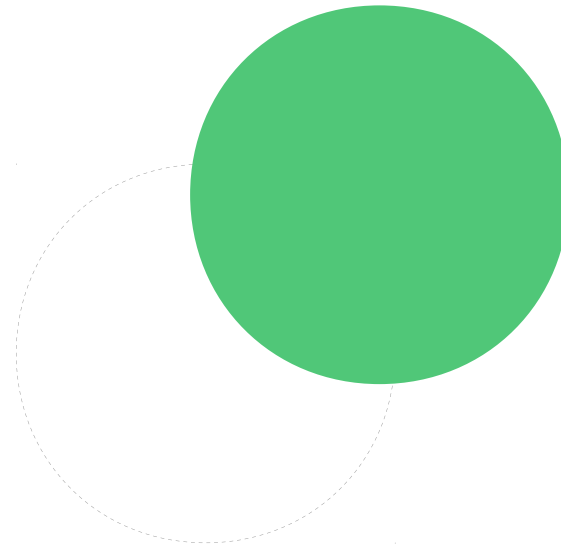
Cut!

```
I.cut : path -> image -> image

I.cut path img ≈
  fun pt ->
    if path pt then img pt else Color.void
```



```
I.const emerald >> I.cut circle
```



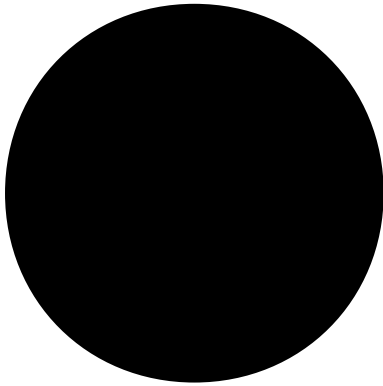
```
I.const emerald >> I.cut circle >>
I.move (V2.v 0.5 0.5)
```

Move

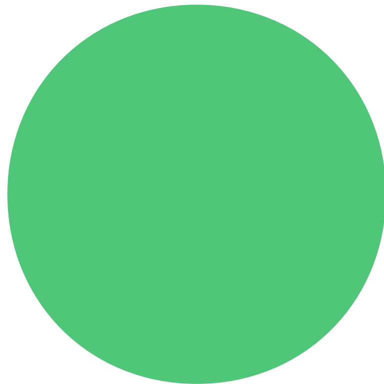
```
I.move : v2 -> image -> image
```

Move

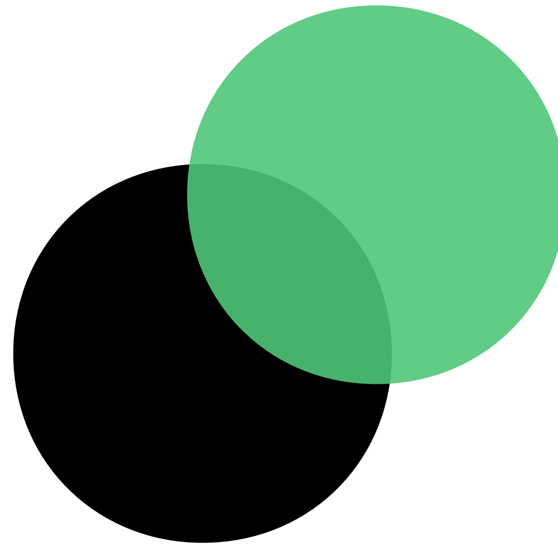
```
I.move : v2 -> image -> image

I.move v img ≈ fun pt -> img (pt - v)
```



```
let black_c =
  I.const Color.black >>
  I.cut circle
```



```
let emerald_c =
  I.const emerald >>
  I.cut circle >>
  I.move (V2.v 0.5 0.5)
```



```
black_c >> I.blend emerald_c
```

Blend

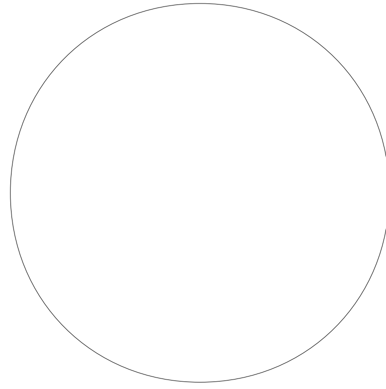
```
I.blend : image -> image -> image
```

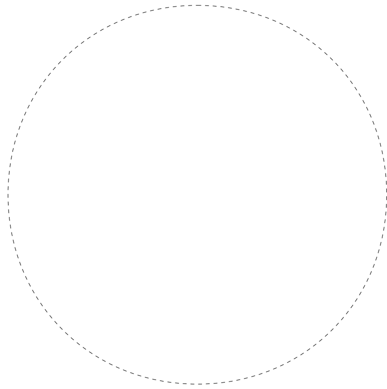
Blend

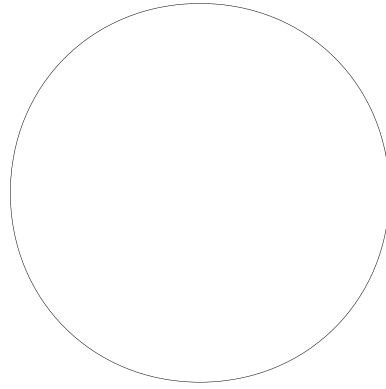
```
I.blend : image -> image -> image

I.blend img img' ≈
  fun pt -> Color.blend (img pt) (img' pt)
```

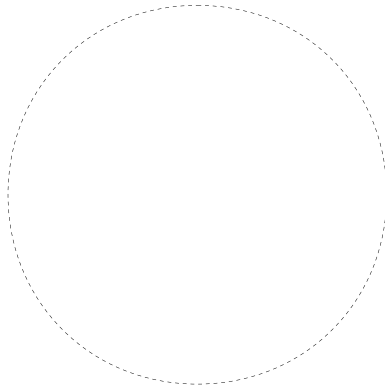
Blend

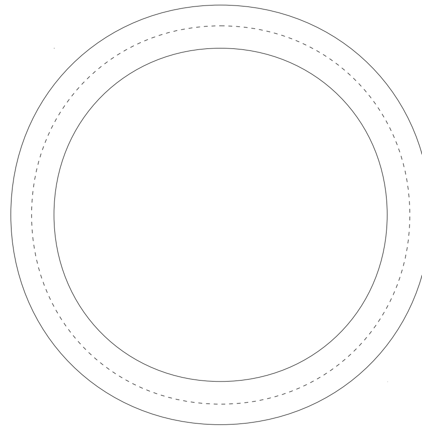
```
I.blend : image -> image -> image

I.blend img img' ≈
  fun pt -> Color.blend (img pt) (img' pt)

I.blend img (I.const Color.void) = img
I.blend (I.const Color.void) img = img
```











Areas as paths

```
type outline = { width : float; ... }
```

Areas as paths

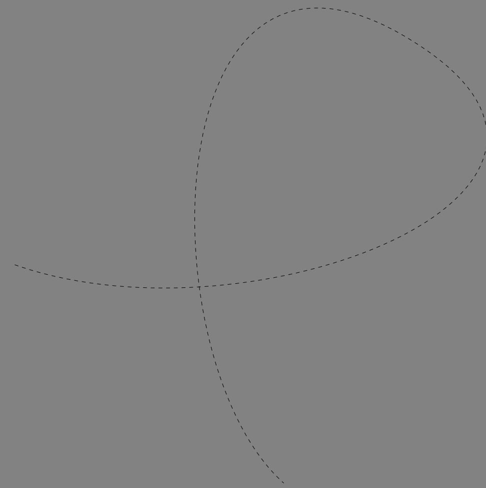
```
type outline = { width : float; ... }

type area = [ `A | `O of outline ]
```

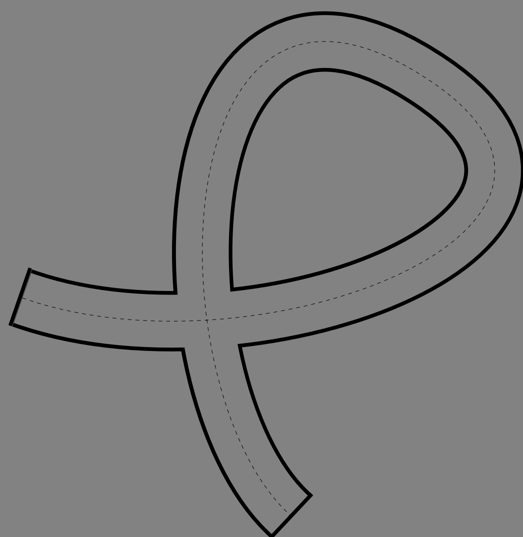
Areas as paths

```
type outline = { width : float; ... }

type area = [ `A | `O of outline ]

type path ≈ area -> pt -> bool
```

Areas as **paths**

```
type outline = { width : float; ... }

type area = [ `A | `O of outline ]

type path ≈ area -> pt -> bool

I.cut : ?area:area -> path -> image -> image
```





Render

PDF `(tbd)`**, SVG,**
HTML CANVAS `(via js_of_ocaml)`**,**
CUSTOM `(via Vg API)`

Insights

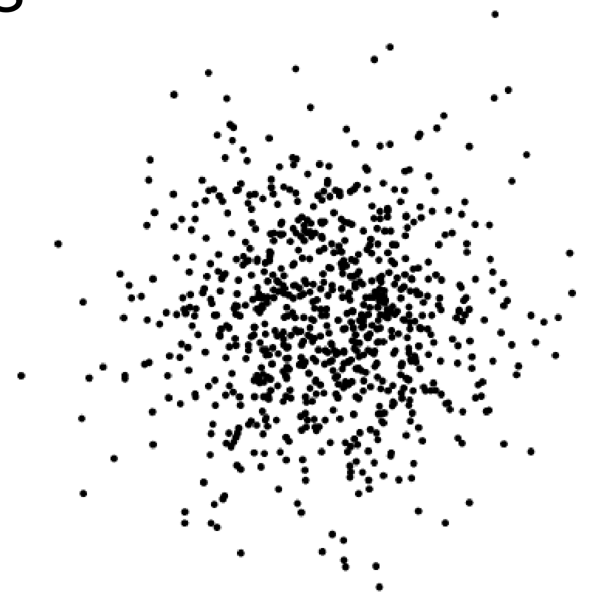
Conal Elliott. *Functional Image Synthesis, Proceedings of Bridges*, 2001.

Antony Courtney. *Haven : Functional Vector Graphics, chapter 6 in Modeling User Interfaces in a Functional Language*, Ph.D. Thesis, Yale University, 2004.

Try it !

http://erratique.ch/software/vg

```
let scatter_plot pts pt_width =
  let dot =
    let r = (0.5 *. pt_width) in
    let circle = P.empty >> P.circle P2.o r in
    I.const Color.black >> I.cut circle
  in
  let mark pt = dot >> I.move pt in
  let add acc pt = acc >> I.blend (mark pt) in
  List.fold_left add I.void pts
```



```
I.axial : Gg.Color.stops -> Gg.p2 -> Gg.p2 ->
          Vg.image
```

```
I.radial : Gg.Color.stops -> ?f:Gg.p2 ->
           Gg.p2 -> float -> Vg.image
```

```
let ( >> ) x f = f x
```