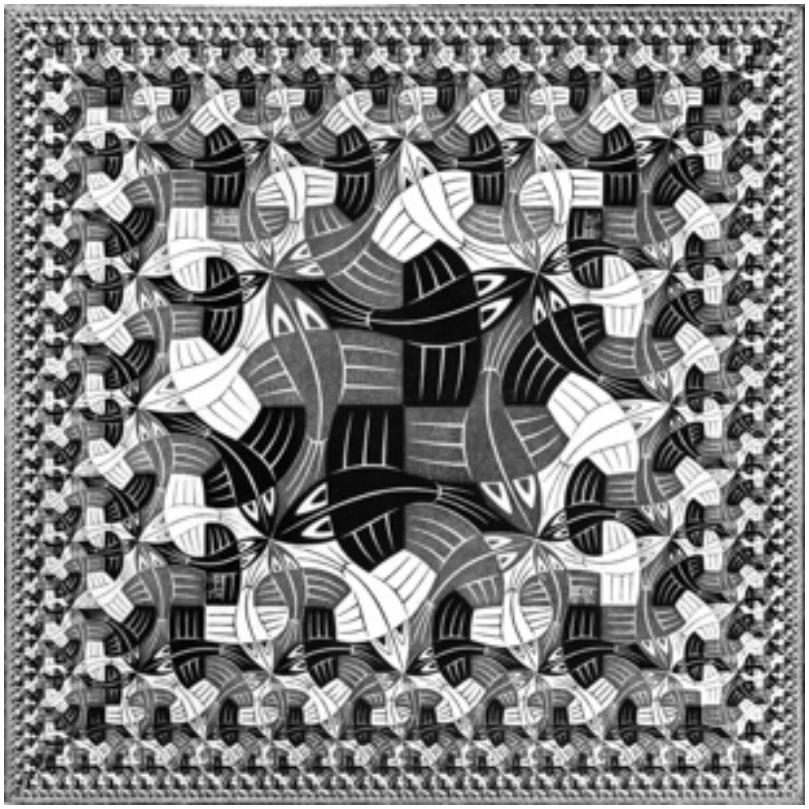


Functional Geometry

Peter Henderson
Department of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
p.henderson@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~ph

October, 2002

Abstract. An algebra of pictures is described that is sufficiently powerful to denote the structure of a well-known Escher woodcut, Square Limit. A decomposition of the picture that is reasonably faithful to Escher's original design is given. This illustrates how a suitably chosen algebraic specification can be both a clear description and a practical implementation method. It also allows us to address some of the criteria that make a good algebraic description.

Keywords: Functional programming, graphics, geometry, algebraic style, architecture, specification.

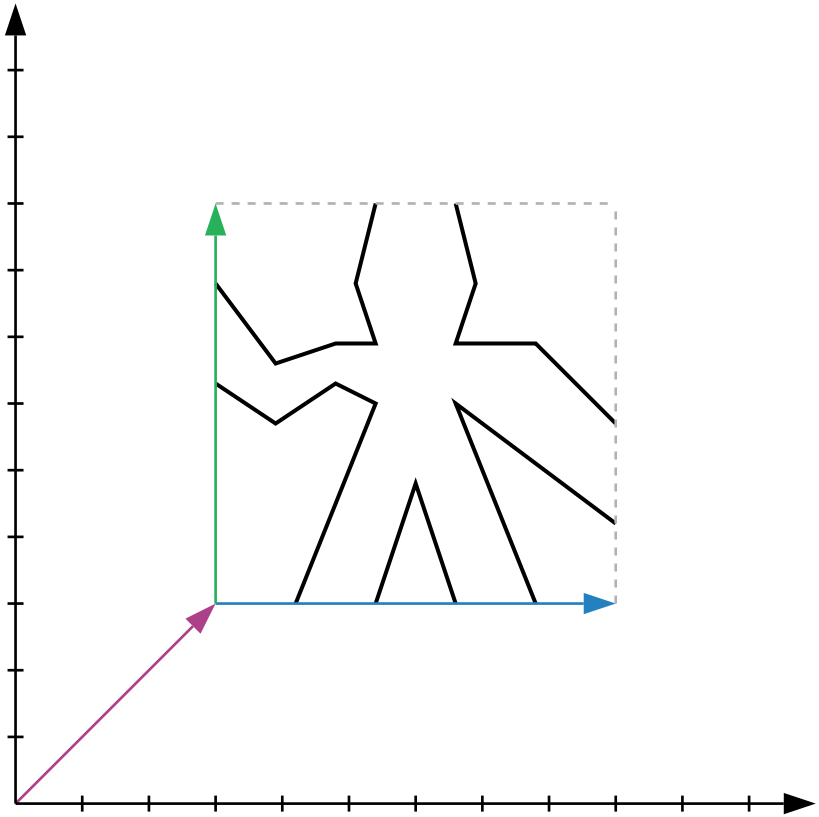
A picture is an example
of a complex object that
can be described in terms
of its parts.

Let us define a picture as a function which takes three arguments, each being two-space vectors and returns a set of graphical objects to be rendered on the output device.

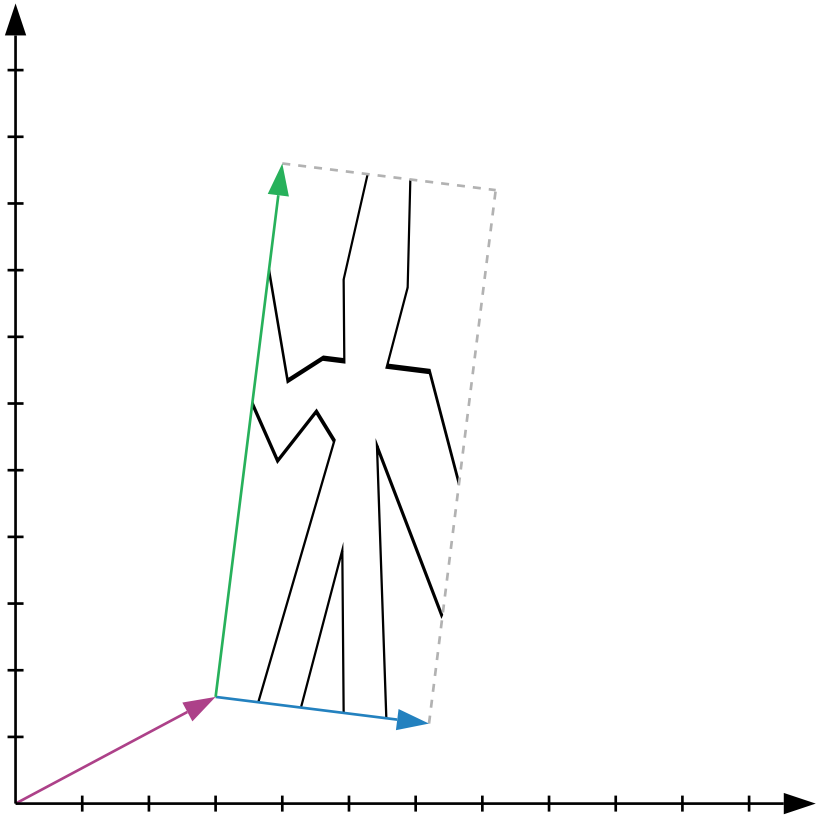
```
type Box = { a : Vector
           , b : Vector
           , c : Vector }

type Picture = Box -> Rendering
```

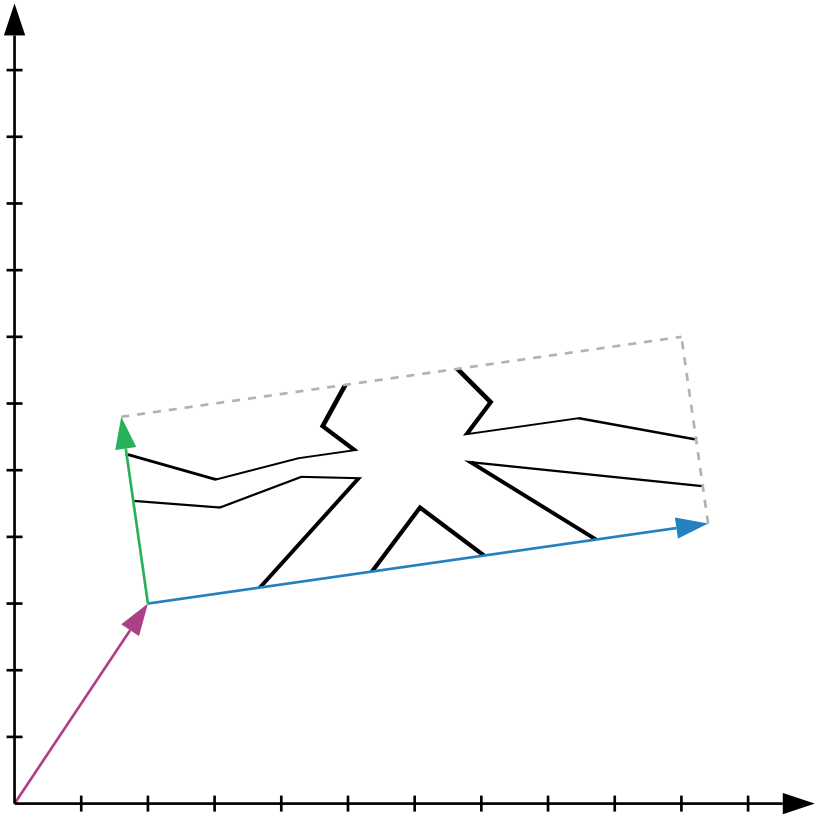
george



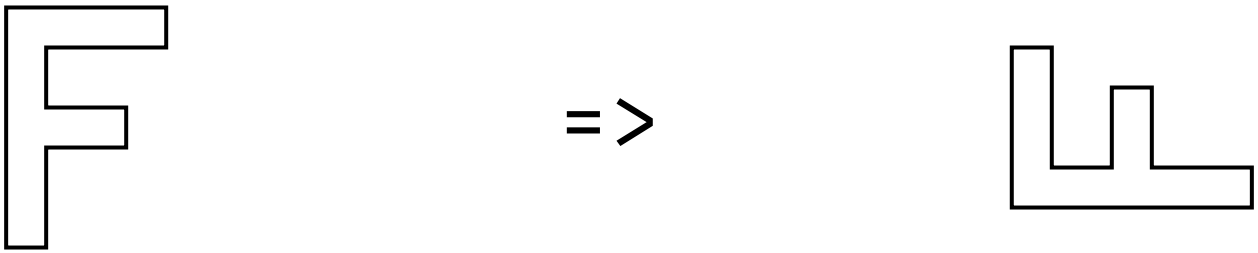
also george



still george



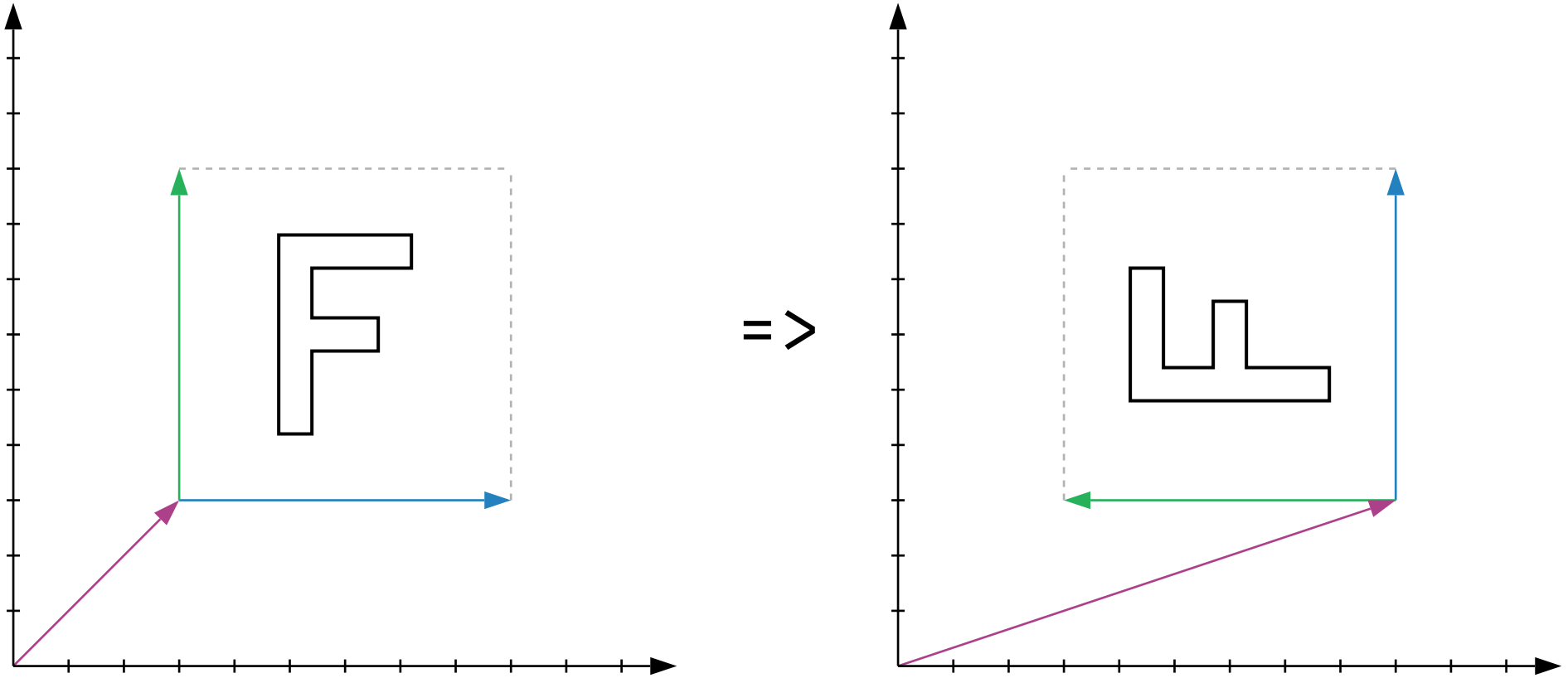
turn



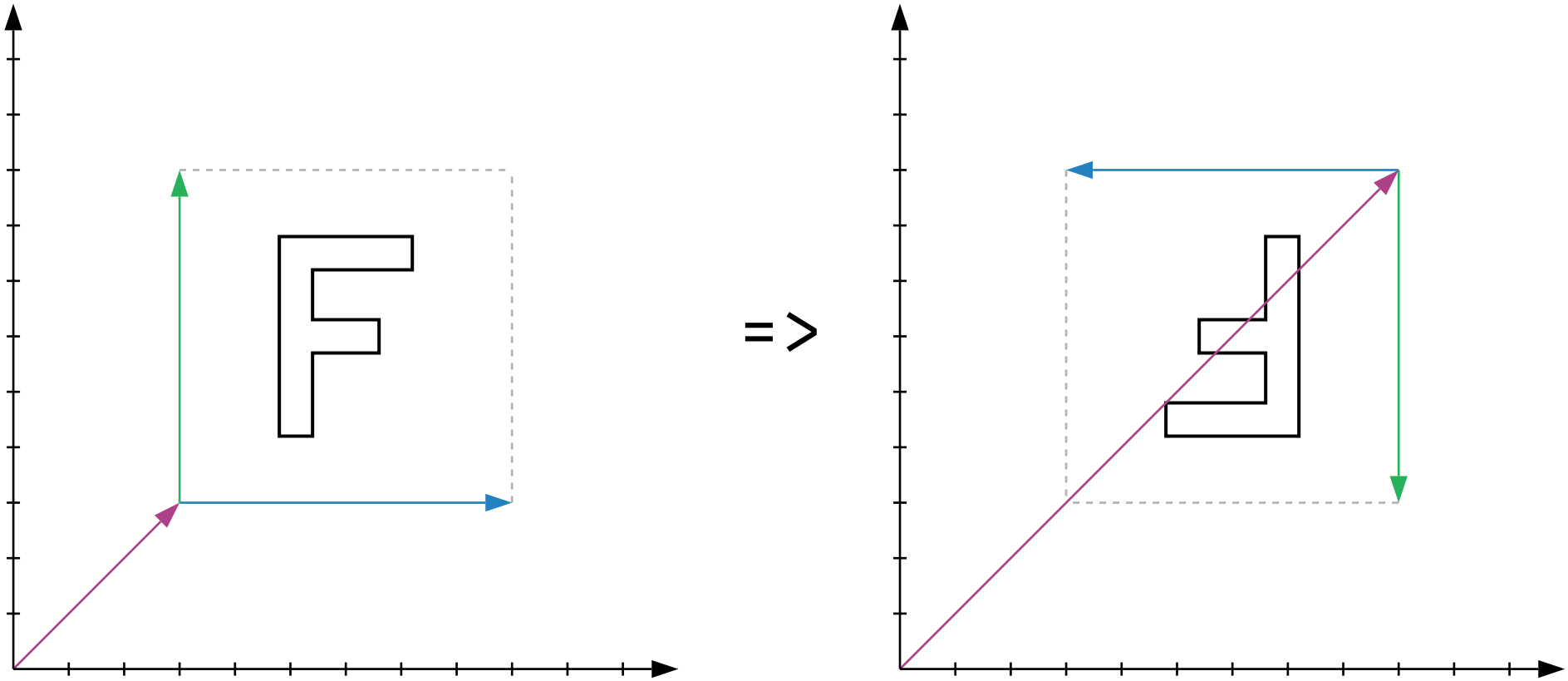
```
turnBox : Box -> Box
turnBox { a, b, c } = { a = add a b
                      , b = c
                      , c = neg b }

turn : Picture -> Picture
turn p = turnBox >> p
```

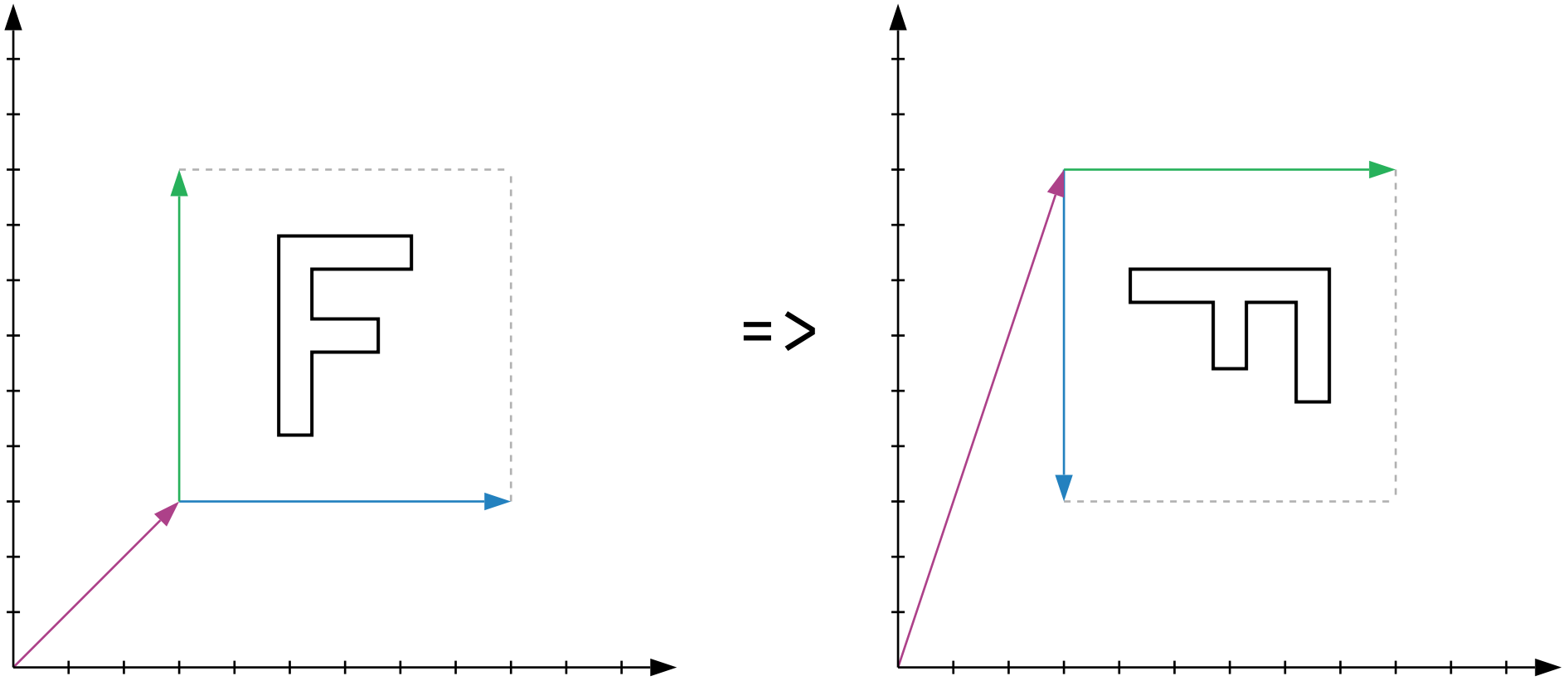
turn



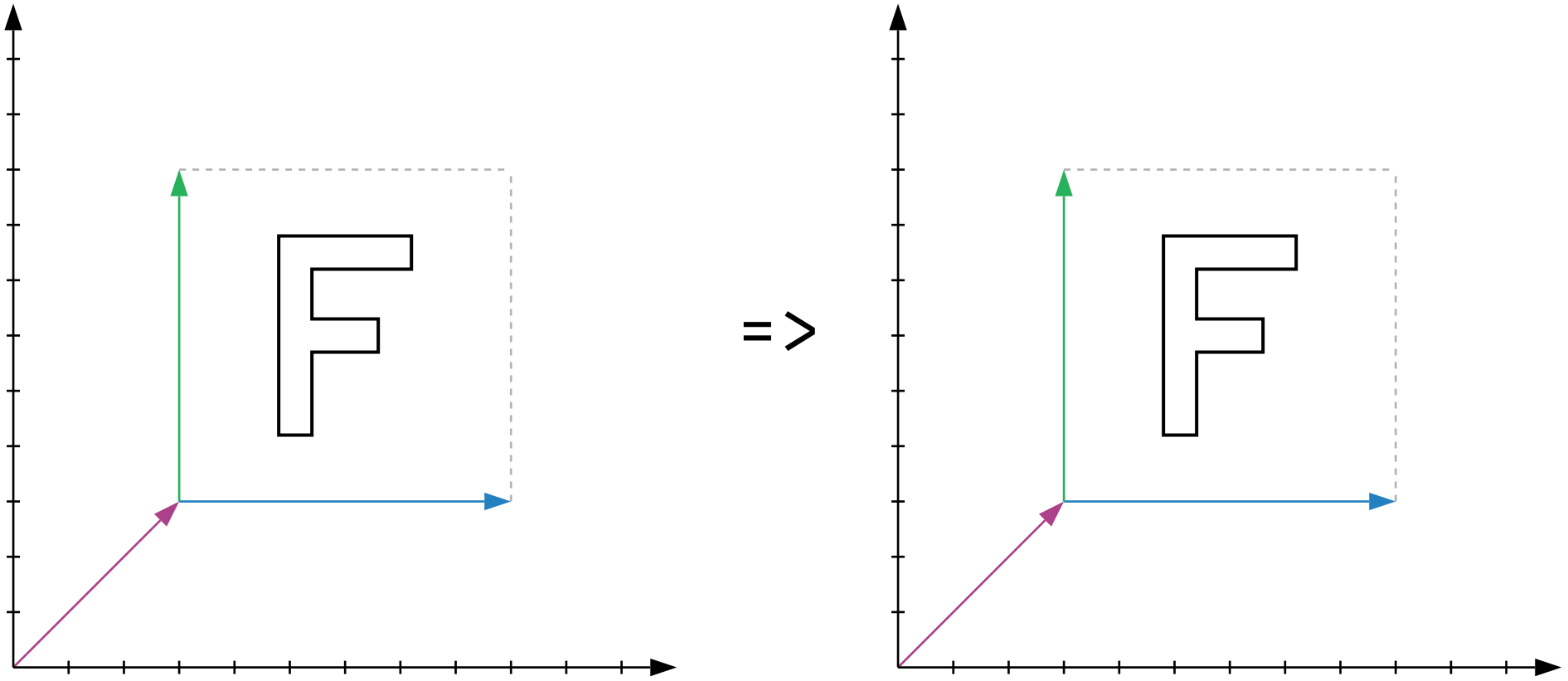
turn >> turn



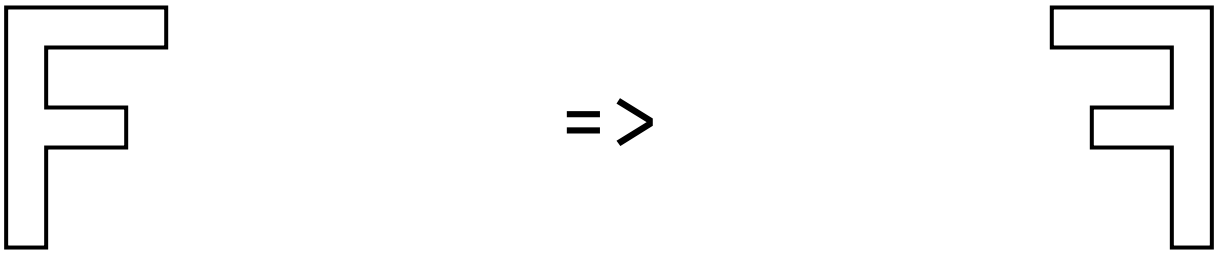
```
turn >> turn >> turn
```



```
turn >> turn >> turn >> turn
```



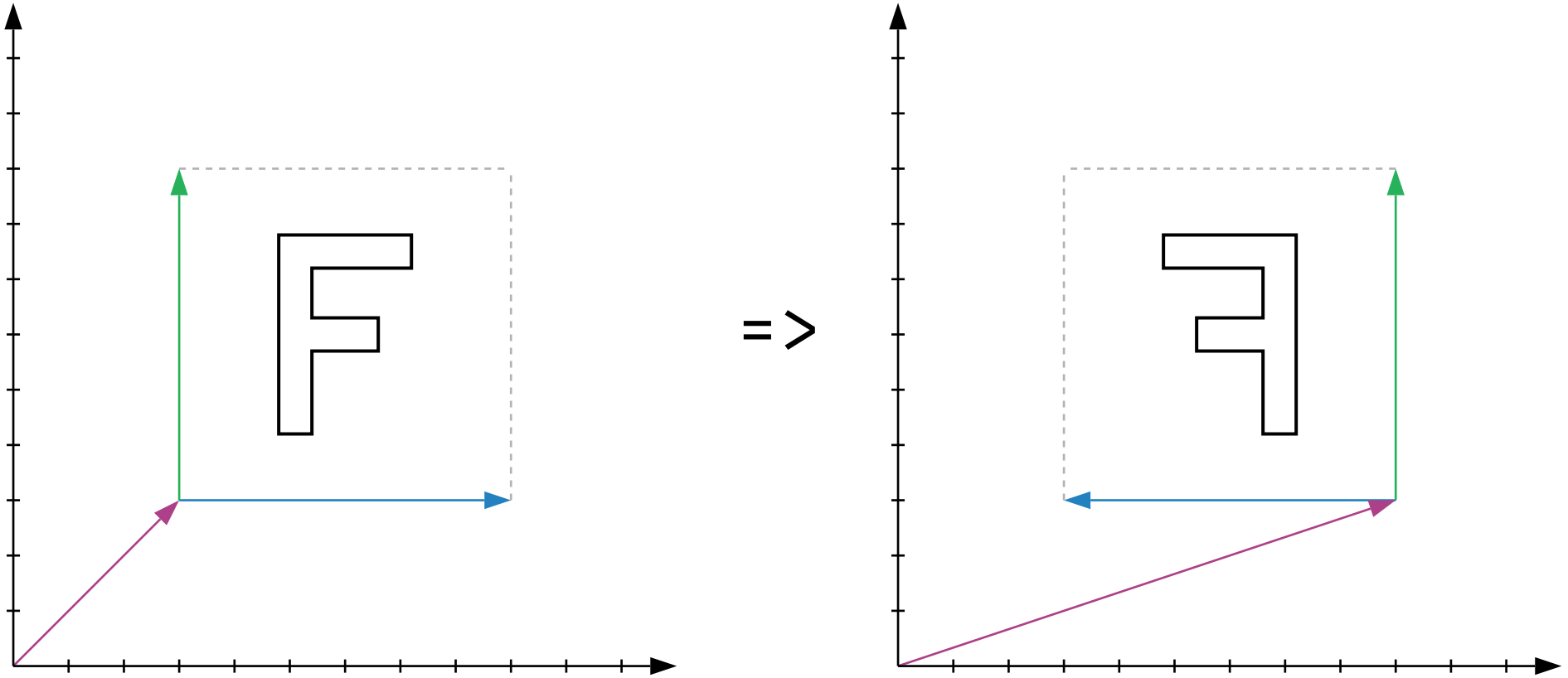
flip



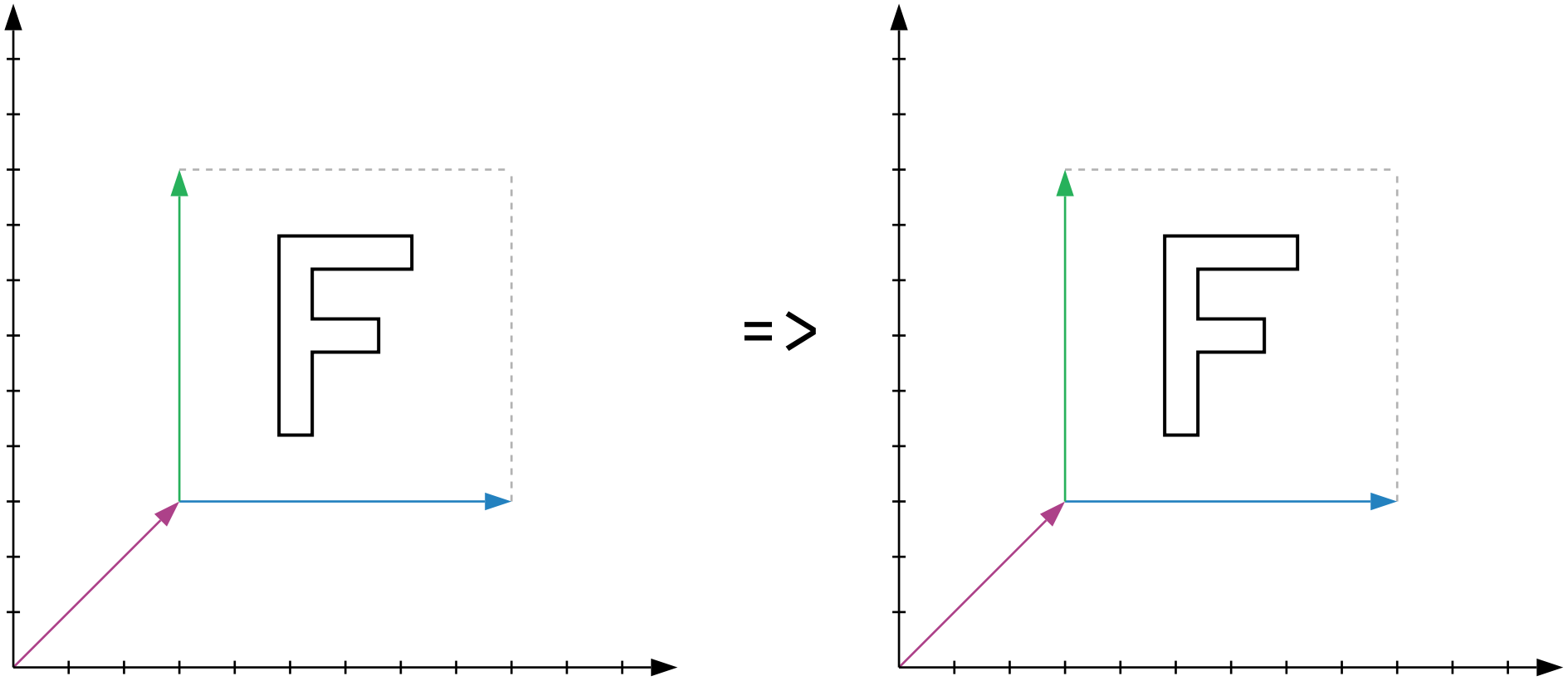
```
flipBox : Box -> Box
flipBox { a, b, c } = { a = add a b
                      , b = neg b
                      , c = neg c }

flip : Picture -> Picture
flip p = flipBox >> p
```

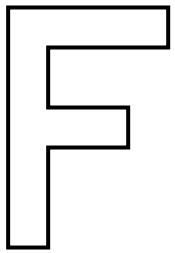
flip



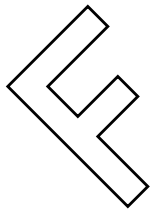
flip >> flip



toss



=>



```
tossBox : Box -> Box
tossBox { a, b, c } =
    { a = add a (scale 0.5 (add b c))
    , b = scale 0.5 (add b c)
    , c = scale 0.5 (sub c b) }

toss : Picture -> Picture
toss p = tossBox >> p
```

toss