

Computer Science I

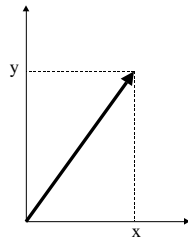
Professor Tom Ellman
Lecture 19

A Picture Language

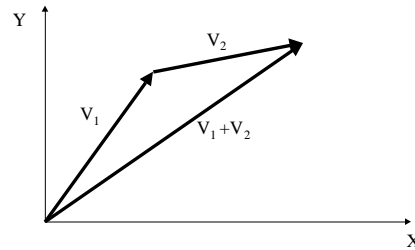
- Demonstrates the power of procedures that build other procedures.
- Taken from “The Structure and Interpretation of Computer Programs”, (2/e) MIT Press, by Abelson and Sussman.

Vectors

- A vector may be visualized as an arrow.
- A vector has a length and a direction.
- A vector lying in a plane may be represented by two numbers: x and y .

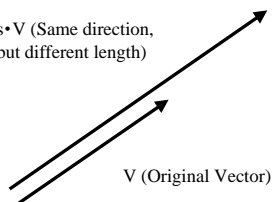


A vector may be added to another vector:



A vector may be scaled by a number:

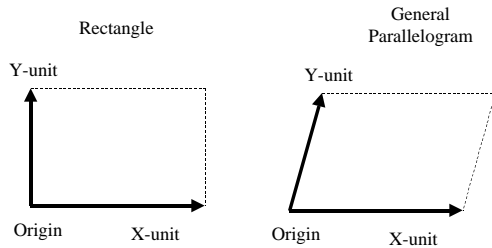
$s \cdot V$ (Same direction,
but different length)



The Vector Data Type

```
(define make-vector (lambda (x y) (list 'vector x y)))  
  
(define vector-x (lambda (vector) (cadr vector)))  
  
(define vector-y (lambda (vector) (caddr vector)))  
  
(define vector-sum  
  (lambda (p1 p2) (make-vector (+ (vector-x p1) (vector-x p2))  
                               (+ (vector-y p1) (vector-y p2)))))  
  
(define vector-scale  
  (lambda (s p) (make-vector (* s (vector-x p)) (* s (vector-y p)))))  
  
(define vector? (lambda (vector) (equal? (car vector) 'vector)))
```

A “FRAME” Defines a Parallelogram



Defining the FRAME Data Type

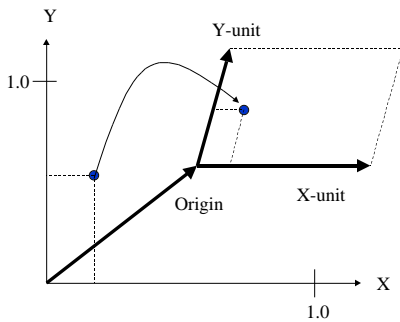
```
(define make-frame
  (lambda (origin x-unit y-unit)
    (list 'frame origin x-unit y-unit)))

(define frame-origin (lambda (frame) (cadr frame)))

(define frame-x-unit (lambda (frame) (caddr frame)))

(define frame-y-unit (lambda (frame) (caddrd frame)))
```

A Frame Defines a New Coordinate System



Converting a Frame into a Coordinate Mapping Function

```
(define frame-map
  (lambda (frame)
    (lambda (vector)
      (vector-sum (frame-origin frame)
                   (vector-sum (vector-scale (vector-x vector)
                                             (frame-x-unit frame))
                                (vector-scale (vector-y vector)
                                             (frame-y-unit frame)))))))
```

The FRAME-MAP function takes a frame as input and returns a function that maps a vector to a vector.

Drawing a Line w.r.t. a Frame

```
(define paint-line
  (lambda (p1 p2 f)
    (let* ((m (frame-map f))
           (p1-new (m p1))
           (p2-new (m p2)))
      (begin (goto (vector-x p1-new) (vector-y p1-new))
              (drawto (vector-x p2-new) (vector-y p2-new))))))
```

We use the frame's coordinate map to convert the given points into points defined relative to the frame.

Painter Procedures

- A *painter* takes a frame as input.
- It draws a picture using the frame as a coordinate system.
- The picture will normally lie inside the parallelogram defined by the frame.

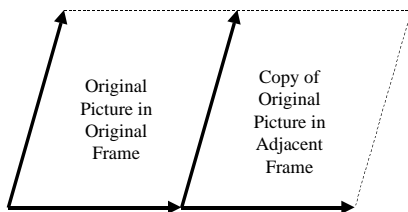
A Cone-Head Stick Figure Painter

```
(define cone-head-painter
  (lambda (frame)
    (let* ((p1 (make-vector 0.0 0.0)) (p2 (make-vector 0.5 0.35))
          (p3 (make-vector 1.0 0.0)) (p4 (make-vector 0.5 0.6))
          (p5 (make-vector 0.0 0.5)) (p6 (make-vector 1.0 0.5))
          (p7 (make-vector 0.5 0.7)) (p8 (make-vector 0.4 0.8))
          (p9 (make-vector 0.6 0.8)) (p10 (make-vector 0.5 1.0)))
      (begin (paint-outline frame) (paint-line p1 p2 frame)
             (paint-line p2 p3 frame) (paint-line p2 p4 frame)
             (paint-line p4 p5 frame) (paint-line p4 p6 frame)
             (paint-line p4 p7 frame) (paint-line p7 p8 frame)
             (paint-line p7 p9 frame) (paint-line p8 p10 frame)
             (paint-line p9 p10 frame))))))
```

Constructing Complex Painters Out of Simple Ones

- We can define a wide variety of operations on painters using higher order functions.
- Each takes one or more painters as input and returns a new, more complex painter as its output.

(SELF-AND-RIGHT <Painter>)

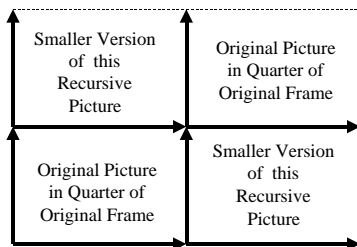


The SELF-AND-RIGHT Procedure

```
(define self-and-right
  (lambda (painter)
    (lambda (frame)
      (begin (painter frame)
             (painter (frame-right frame))))))

(define frame-right
  (lambda (frame)
    (make-frame (vector-sum (frame-origin frame)
                           (frame-x-unit frame))
                (frame-x-unit frame)
                (frame-y-unit frame))))
```

(CORNERS <Painter>)



The CORNERS Procedure

```
(define corners
  (lambda (painter n)
    (lambda (frame)
      (if (= n 0)
          (painter frame)
          (let ((quarter (frame-quarter frame))
                (recursive-painter (corners painter (- n 1))))
              (begin (painter quarter)
                     (painter (frame-above (frame-right quarter)))
                     (recursive-painter (frame-above quarter))
                     (recursive-painter (frame-right quarter))))))))))
```

The CORNERS Procedure

- The CORNERS procedure is not directly recursive.
- It does not call itself.
- It simply returns a painter procedure.
- The painter may in turn call CORNERS.
- Therefore the CORNERS procedure is indirectly recursive.

