Anexo B

Procedimentos gráficos

Os procedimentos gráficos disponibilizados pelas várias implementações do *Scheme* são, em geral, sofisticados e requerem alguma experiência para uma utilização correcta. Para os exercícios propostos que requeiram saída gráfica, optou-se por um conjunto de procedimentos simples que funcionam numa janela gráfica. Trata-se de uma abstracção, a *janela gráfica*, à qual se associa uma caneta que pode desenhar, pintar[1], ou escrever, com cores à escolha. O *ponto corrente* é o ponto onde se encontra a caneta.

⇒ ***(janela largura altura titulo)***
Cria janela gráfica com a dimensão *largura* x *altura* e com um *título* (cadeia de caracteres);

⇒ ***(cor indice)***
De uma tabela de 16 cores, escolhe uma delas para cor da caneta, através de *indice*, inteiro situado entre 0 e 15. Se *indice = 'temp* (temporário), o efeito de uma primeira passagem da caneta é anulado por uma segunda passagem. É com esta cor que se deve desenhar ou pintar um objecto animado, sem se perder os gráficos visualizados e sobre os quais o objecto se desloca. Para tal, bastará desenhar o objecto animado com a cor *'temp* e, quando ele vai sair, deverá ser novamente desenhado com a cor *'temp*, no local onde se encontra, reaparecendo o que ele tapava;

⇒ ***(move pt)***
Move a caneta, sem desenhar, para *pt* (*pt* é uma lista com as coordenadas *x* e *y*);

⇒ ***(move-rel desloca)***
Sendo Px e Py as coordenadas do *ponto corrente*, e o argumento *desloca* uma lista com *dx* e *dy*, a caneta é movida, sem desenhar, para o ponto (Px + dx Py + dy);

⇒ ***(desenha lista-de-pontos)***
Desenha, movendo a caneta ao longo da linha poligonal que une os vários pontos em *lista-de-pontos*. A caneta fica sobre o último ponto;

⇒ ***(desenha-rel lista-de-pontos)***
Como *desenha*, mas cada ponto em *lista-de-pontos* é definido relativamente ao ponto onde se encontra a caneta, antes desta começar a mover-se na sua direcção;

⇒ ***(pinta lista-de-pontos)***
Como *desenha*, mas preenche com a cor da caneta o polígono definido por *lista-de-pontos*. Se o último ponto de *lista-de-pontos* não coincidir com o primeiro, o polígono é pintado como se esta coincidência existisse, mas a caneta fica no último ponto de *lista-de-pontos*;

⇒ ***(pinta-rel lista-de-pontos)***
Como *pinta*, mas cada ponto em *lista-de-pontos* é definido relativamente ao ponto onde se encontra a caneta, antes desta começar a mover-se na sua direcção;

⇒ ***(ponto pt)***
Como *move*, mas pinta *pt* com a caneta;

⇒ ***(ponto-rel pt)***
Como *ponto*, mas *pt* é relativo ao *ponto corrente*;

⇒ ***(desenha-txt texto)***
A partir do ponto corrente, escreve *texto* e deixa a caneta no canto inferior direito do último caracter de *texto*;

⇒ ***(caneta)***
Devolve o *ponto corrente*;

⇒ ***(limpa)***
Limpa a janela.

---

[1] Pintar significa preencher o interior de polígonos com a cor da caneta

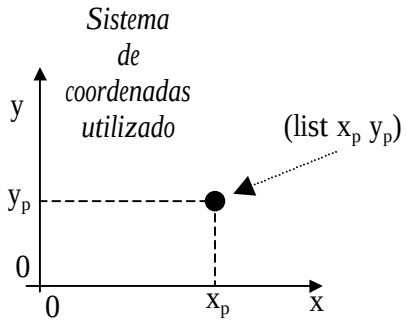
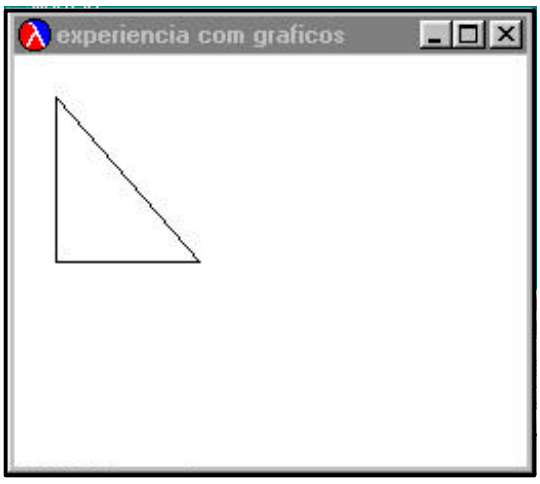


Tabela de cores utilizada

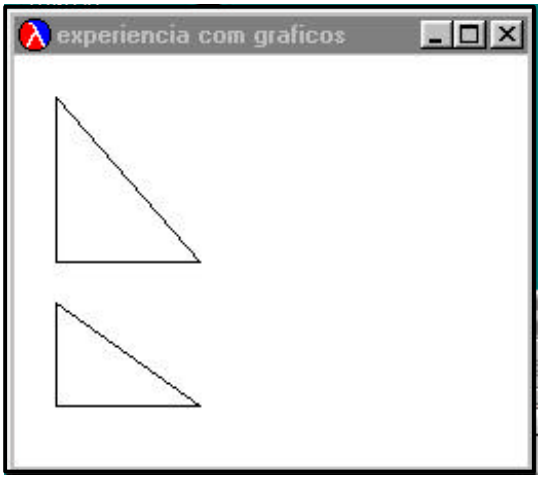
índice	R	G	B	cor
0	0	0	0	preto
1	0	0	127	azul escuro
2	0	127	0	verde escuro
3	0	127	127	cyan escuro
4	127	0	0	vermelho escuro
5	127	0	127	magenta escuro
6	127	127	0	amarelo escuro
7	127	127	127	branco-cinza escuro
8	191	191	191	branco-cinza
9	0	0	255	azul
10	0	255	0	verde
11	0	255	255	cyan
12	255	0	0	vermelho
13	255	0	255	magenta
14	255	255	0	amarelo
15	255	255	255	branco

```
↦(janela 250 200 "experiencia com graficos")
```

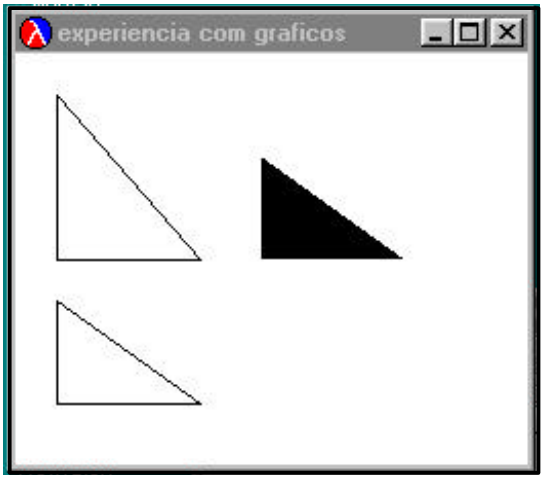

```
↦(desenha (list '(20 100)'(20 180)'(90 100)'(20 100)))
```



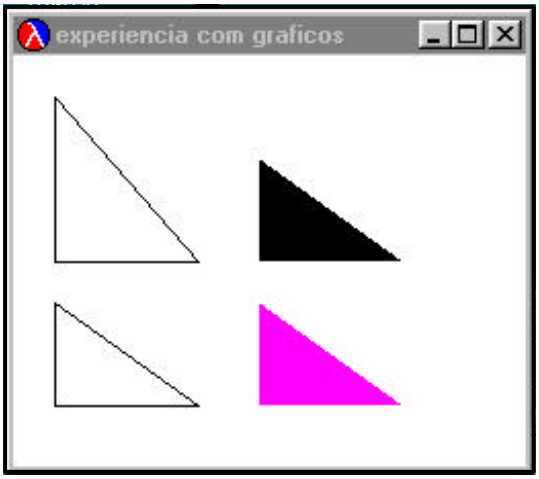
```
↦(move (list 20 30))
↦(desenha-rel (list '(0 0)'(0 50)'(70 -50)'(-70 0)))
```




```
↦(move (list 120 100))
↦(pinta-rel (list '(0 0)'(0 50)'(70 -50)'(-70 0)))[2]
```

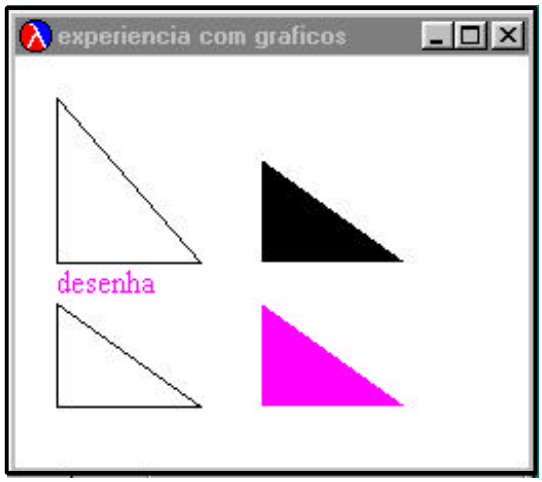



```
↦(cor 13)
↦(pinta '((120 30)(120 80)(190 30)))
```

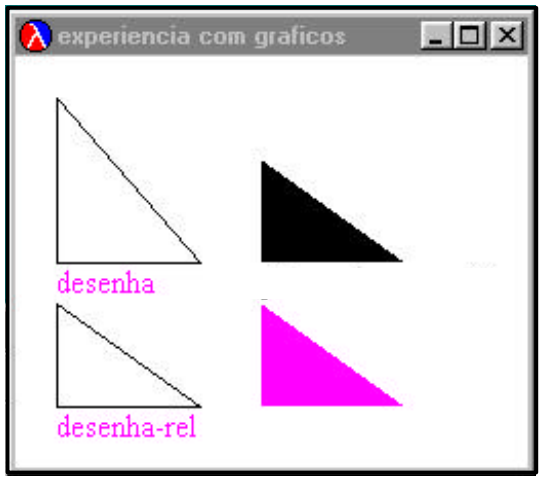


[2] ou: ↦(pinta-rel (list '(0 0)'(0 50)'(70 -50)))

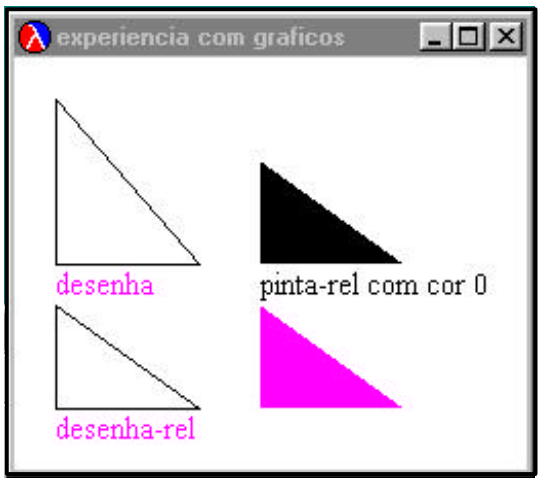
```
↦(move '(20 85))
↦(desenha-txt "desenha")
```



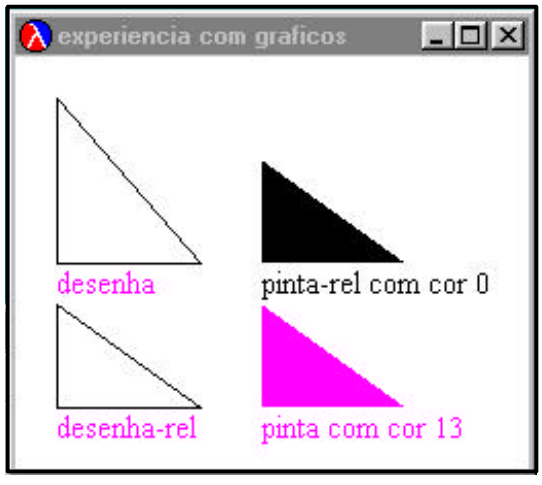

```
↦(move '(20 15))
↦(desenha-txt "desenha-rel")
```




```
↦(move '(120 85))
↦(cor 0)
↦(desenha-txt "pinta-rel com cor 0")
```




```
↦(move (list 120 15))
↦(cor 13)
↦(desenha-txt "pinta com cor 13")
```



Apresentam-se duas implementações dos procedimentos gráficos, uma relativa ao *EdScheme* e outra ao DrScheme.

```
;
;
; implementação para o EdScheme, versão 4.3 ou seguintes.
;
;
(define posi (list 0 0))   ; ponto corrente (posiX posiY)

(define janela             ; cria uma janela gráfica
  (lambda (larg alt titulo)
    (let ((jan (make-graphics-window (list larg alt))))
      (begin
        (turtle-hide jan)
        (window-set-title titulo jan)
        (graphics-set-origin '(0 0) jan)
        jan))))

(define cor                ; para seleccionar a cor da caneta ...
  (lambda (indice)           ; indice = 'temp (passando duas vezes com esta cor
    (let ((tab-cores         ; repõe a cor existente antes da 1ª passagem
            '((0 0 0)(0 0 127)(0 127 0)(0 127 127)
              (127 0 0)(127 0 127)(127 127 0)(127 127 127)
              (191 191 191)(0 0 255)(0 255 0)(0 255 255)
              (255 0 0)(255 0 255)(255 255 0)(255 255 255))))
      (cond ((equal? indice 'temp)
             (pen-reverse))
            (else
              (let ((cor (list-ref tab-cores indice)))
                (pen-set-color cor)
                (fill-set-style 0)
                (fill-set-foreground-color cor)
                (pen-down)))))))

(define move               ; move caneta para ponto
  (lambda (ponto)
    (set-car! posi (car ponto))
    (set-car! (cdr posi) (cadr ponto))))

(define move-rel           ; move caneta, em relação ao ponto corrente
  (lambda (d)                 ; ou seja, para (posiX + dx  posiY + dy)
    (set-car! posi (+ (car posi)
                      (car d)))
    (set-car! (cdr posi) (+ (cadr posi)
                            (cadr d)))))

(define desenha            ; desenha polígono definido por
  (lambda (seq-pontos)     ; seq-pontos e deixa caneta no último ponto
    (func-graf polygon seq-pontos)))

(define pinta              ; pinta polígono definido por
  (lambda (seq-pontos)     ; seq-pontos e deixa caneta no último ponto
    (func-graf polygon-paint seq-pontos)))

(define func-graf          ; desenha ou pinta
  (lambda (func seq-pontos)
    (let ((ultimo (car (reverse seq-pontos))))
      (func seq-pontos)
```

```
      (set-car! posi (car ultimo))
      (set-car! (cdr posi) (cadr ultimo)))))

(define desenha-rel         ; como desenha, mas cada ponto é definido
  (lambda (seq-pontos)            ; relativamente à posição da caneta, no
    (let ((aux                    ; início do traçado de cada segmento
            (lambda (elem)
              (begin
                (set-car! posi (+ (car elem)
                                  (car posi)))
                (set-car! (cdr posi) (+ (cadr elem)
                                        (cadr posi)))
                (list (car posi) (cadr posi))))))
      (desenha (map aux seq-pontos)))))

(define pinta-rel           ; como pinta, mas cada ponto é definido
  (lambda (seq-pontos)              ; relativamente à posição da caneta, no
    (let ((aux                      ; início do traçado de cada segmento
            (lambda (elem)
              (begin
                (set-car! posi (+ (car elem)
                                  (car posi)))
                (set-car! (cdr posi) (+ (cadr elem)
                                        (cadr posi)))
                (list (car posi) (cadr posi))))))
      (pinta (map aux seq-pontos)))))

(define ponto               ; move a caneta para o ponto p
  (lambda (p)               ; e desenha p com a caneta
    (draw-point p)
    (set-car! posi (car p))
    (set-car! (cdr posi) (cadr p))))

(define ponto-rel           ; como ponto, mas p é definido relativo ao
  (lambda (p)               ; ponto corrente
    (ponto (list (+ (car p)
                    (car posi))
                 (+ (cadr p)
                    (cadr posi))))))

(define desenha-txt         ; desenha texto a partir do ponto corrente
  (lambda (texto)           ; e deixa a caneta no final do último caracter
    (pen-up)
    (turtle-set-position posi)
    (pen-down)
    (turtle-display texto)
    (let ((ultimo (turtle-position)))
      (set-car! posi (car ultimo))
      (set-car! (cdr posi) (cadr ultimo)))))

(define caneta              ; devolve o ponto corrente
  (lambda ()
    (list (car posi)
          (cadr posi))))

(define limpa               ; limpa a janela gráfica activa
  (lambda ()
    (clean)))
```

```
;
;
; implementação para o DrScheme, versão 53 ou seguintes.
;
;
(define posi (list 0 0))                    ; posicao da caneta
(define cor-caneta (make-rgb 0 0 0))        ; cor da caneta
(define cor-temp? #f)                       ; cor temporaria?
(define vp 0)                               ; viewport
(define altura 0)                           ; altura do viewport

(define janela                  ; cria uma janela gráfica
  (lambda (larg alt titulo)
    (open-graphics)
    (set! altura alt)
    (let ((vport (open-viewport titulo larg alt 1.0)))
      (set! vp vport))))

(define cor                 ; para seleccionar a cor da caneta ...
  (lambda (indice)            ; indice = 'temp (passando duas vezes com esta cor
    (let ((tab-cores          ; repõe a cor existente antes da 1ª passagem
            '((0 0 0)(0 0 0.5)(0 0.5 0)(0 0.5 0.5)
              (0.5 0 0)(0.5 0 0.5)(0.5 0.5 0)(0.5 0.5 0.5)
              (0.75 0.75 0.75)(0 0 1.0)(0 1.0 0)(0 1.0 1.0)
              (1.0 0 0)(1.0 0 1.0)(1.0 1.0 0)(1.0 1.0 1.0))))
      (cond ((equal? indice 'temp)
             (set! cor-temp? #t))
            (else
              (let ((cor-aux (list-ref tab-cores indice)))
                (let ((cor-drscm (make-rgb (car cor-aux)
                                           (cadr cor-aux)
                                           (caddr cor-aux))))
                  (set! cor-temp? #f)
                  (set! cor-caneta cor-drscm))))))))

(define move                    ; move caneta para ponto
  (lambda (ponto)
    (set-car! posi (car ponto))
    (set-car! (cdr posi) (cadr ponto))))

(define move-rel            ; move caneta, em relação ao ponto corrente
  (lambda (d)               ; ou seja, para (posiX + dx  posiY + dy)
    (set-car! posi (+ (car posi)
                      (car d)))
    (set-car! (cdr posi) (+ (cadr posi)
                            (cadr d)))))

(define desenha             ; desenha polígono definido por
  (lambda (seq-pontos)      ; seq-pontos e deixa caneta no último ponto
    (if cor-temp?
        ((flip-polygon vp)
         (list->posn-list seq-pontos '(0 0))
         (make-posn 0 0)
         cor-caneta)
        ((draw-polygon vp)
         (list->posn-list seq-pontos '(0 0))
         (make-posn 0 0)
         cor-caneta))                           ;  e deixa caneta no
    (set! posi (car (reverse seq-pontos)))))    ; último ponto

(define pinta               ; pinta polígono definido por
  (lambda (seq-pontos)      ; seq-pontos e deixa caneta no último ponto
```

```
    (if cor-temp?
        ((flip-solid-polygon vp)
         (list->posn-list seq-pontos '(0 0))
         (make-posn 0 0)
         cor-caneta)
        ((draw-solid-polygon vp)
         (list->posn-list seq-pontos '(0 0))
         (make-posn 0 0)
         cor-caneta))                          ;  e deixa caneta no
    (set! posi (car (reverse seq-pontos)))))   ; último ponto

(define list->posn-list
  (lambda (s-pontos pos)
    (let
      ((x-posi (car pos))
       (y-posi (cadr pos)))
      (let
        ((aux
           (lambda (elem)
             (make-posn
               (+ (car elem)
                  x-posi)
               (- altura
                  (+ (cadr elem)
                     y-posi))))))
        (map aux s-pontos)))))

(define desenha-rel        ; como desenha, mas cada ponto é definido
  (lambda (seq-pontos)             ; relativamente à posição da caneta, no
    (let ((aux                     ; início do traçado de cada segmento
            (lambda (elem)
              (begin
                (set-car! posi (+ (car elem)
                                  (car posi)))
                (set-car! (cdr posi) (+ (cadr elem)
                                        (cadr posi)))
                (list (car posi)
                      (cadr posi))))))
      (desenha (map aux seq-pontos)))))

(define pinta-rel          ; como pinta, mas cada ponto é definido
  (lambda (seq-pontos)              ; relativamente à posição da caneta, no
    (let ((aux                      ; início do traçado de cada segmento
            (lambda (elem)
              (begin
                (set-car! posi (+ (car elem)
                                  (car posi)))
                (set-car! (cdr posi) (+ (cadr elem)
                                        (cadr posi)))
                (list (car posi)
                      (cadr posi))))))
      (pinta (map aux seq-pontos)))))

(define ponto              ; move a caneta para o ponto p
  (lambda (p)              ; e desenha p com a caneta
    ((draw-pixel vp) (make-posn (car p)
                                (- altura (cadr p))) cor-caneta)
    (set-car! posi (car p))
    (set-car! (cdr posi) (cadr p))))

(define ponto-rel           ; como ponto, mas p é definido relativo ao
  (lambda (p)               ; ponto corrente
    (ponto (list (+ (car p)
                    (car posi))
```

```
                  (+ (cadr p)
                     (cadr posi))))))

(define desenha-txt
  (lambda (texto)
    ((draw-string vp) (make-posn (car posi)
                       (- altura
                          (cadr posi)))
     texto cor-caneta)                              ;  e deixa caneta no
    (let ((larg-alt ((get-string-size vp) texto)))  ; último ponto
      (set-car! posi (+ (car larg-alt)
                (car posi))))))

(define caneta             ; devolve o ponto corrente
  (lambda ()
    (list (car posi)
      (cadr posi))))

(define limpa              ; limpa a janela gráfica activa
  (lambda ()
    ((clear-viewport vp))))
```