Implementation on the PDP-1 of a Subset of the Picture Calculus

Introduction

The subset of the picture calculus implemented consists of two unary operations: unary minus and blanking operator, and four binary operations: +, -, x, *.

Those operations perform on a set of up to 36 primitives, of which eight, numbered 0 through 7, represent the eight main directions and are built in the system. Their length is such that squares and diagonals can be drawn (fig. 1).

PDP-1 Displays (Philco "Read" consoles)

The Philco console draws vector in an incremental way; that is, the origin (tail) of each new vector is taken as being the extremity (head) of the preceding vector. Each vector is defined by a word in storage, so a picture will be a list of those words stored contiguously. There is, of course, a way of fixing the position of the tail (origin) of the first vector of the list.

Here, the list of word defining the picture currently displayed is stored in the array DB (Display Buffer).

There is a bit (in the word defining a vector) specifying if the vector is to be seen or be invisible (blank vector).

Data Structure

A primitive is a set of vectors (visible or blank), the extremity of each one being the origin of the next one.

In the beginning, there are only eight primitives, but any picture built with those eight primitives, using the operations +, -, x, *, unary minus and blanking operator, can be stored in a location designated by a letter of the alphabet, and can be used subsequently just as one of the original primitives.

Each primitive is stored in a block of variable length, whose format is given by fig 2.

$\Delta X$ is the X-coordinate of the head of the primitive with respect to its tail.

$\Delta Y$ is the Y-coordinate of the head of the primitive with respect to its tail.

$E$ is the address of the first word outside that block.

$V$ is a variable length space that holds the words representing the vectors making up the primitive.
The position of the tail of any picture displayed on the screen is arbitrary; so we fixed it to the center of the screen.

To find a given primitive, there is an array corresponding to the digits and letters of the alphabet, whose cells contain either the address of the primitive associated to that character, or zero (no primitive associated) (fig. 3).

**Implementation of the Operations**

Let us suppose that \(a\) is the picture already on the display, and that \(b\) is the primitive to be added (fig. 4).

**Binary operations**

- \(\text{a+b}:\) the string of vectors corresponding to the primitive \(b\) is added to the list of vectors corresponding to \(a\), which is in the display buffer DB; so the origin of the first vector of \(b\) (tail of \(b\)) is the extremity (head) of the last vector of \(a\). This works only because the display console works in an incremental way.

- \(\text{a-b}:\) a string of blank vectors (dashed lines on fig. 4), going from the head of \(b\) to its tail, is added to the buffer DB; then \(b\) is added. To compute the string of blank vector, use is made of the \(\Delta X\) and \(\Delta Y\) corresponding to \(b\).

- \(\text{axb}:\) a string of blank vectors going from the head of \(a\) to its tail is added to the buffer DB; then \(b\) is added. To compute the string of blank vectors, one needs to know the coordinates of the head of \(a\) with respect to its tail; this is saved and updated in cells \(\text{chn}\) and \(\text{chy}\).

- \(\text{a*b}(a:b\text{ in this implementation}):\) First \(\text{chn}\) and \(\Delta X\), \(\text{chy}\) and \(\Delta Y\) are compared; if they are not equal, then the operation \(*\) is undefined for those two primitives. Else, that operation is equivalent to \(\text{a-b}\).

**Unary operations**

- \(\text{Unary minus (÷ \text{\text{b}} in this implementation}):\) a string of blank vectors going from the head of \(b\) to its tail is added both before and after the string of vectors representing \(b\).

- \(\text{Blanking operator (.\text{\text{b}} in this implementation):}\) the blanking bit is set in all the vectors of \(b\).

**Other features**

A backspace will erase the last primitive added; a space will get it back (both are recursive).
Hitting the "!" key will mark the tail and head of the picture currently displayed.

Hitting the "tab" key will erase the whole screen.

Storing a picture in a letter is done by hitting the "=" key, then the letter key.

Then that picture becomes a primitive, and can be used as any other primitive. Ex: a can be 3, and b can be c (if there is something in c).
fig. 1 Basic primitives

fig. 2 Format of primitives in storage
Fig 3. Storage of primitives.
fig. 4. Operations
BEGIN
ORG 0
IOT 53
LAW ST+50
DAC @S

INIT:
DZM @CHX
DZM @CHY
DZM NMD
CLF 7
STF 1
LAW AP
DAC @T
LAW DB+2
DAC @P

DISP:
DZM @BLANK
LAC P
DAC I T
IDX T
LAC CHX
DAC I T
IDX T
LAC CHY
DAC I T
IDX T
DAC @MAX

DISPLAY:
LAC P
SUB [DB]
DAC NMT
LIO NMD
DAC NMD
DIO TNMD
LAW AIO3
IOT 3

READ:
IOT 7
AND [77]
SUB [IO]
SPA
JMP PASLET
SUB [44]
SMA
JMP PASLET
ADD [IA+44]
DAP .+1
LAC .
SZA I
JMP ERR3
DAC @J
JSP CALCUL
SZF I J
JMP SUIT
ADD CHX
DAC CHX
IDX J
JSP CALCUL
ADD CHY
DAC CHY

COMMON:
IDX J
LAC I J
DAC @N
IDX J

LOOP1:
LAC I J
LIO BLANK
SNI I
IOR [400000]
SZF 6
XOR [100200]
DAC I P
IDX P
IDX J
SAS N
JMP LOOP1
CLF 6
JMP DISP

ERR3:
IOT 17

TEXT //
THERE IS NOTHING SAVED IN THERE//
JMP READ
SUIT:
SZF I 2
JMP SUIT1
LIA
IDX J
JSP CALCUL

MIN:
SPI I
STF I
SPI
CMI
DIO @SDX
SMA
STF 5
SPA
CMA
DAC @SDY
SIR 77
DIO @NX
SAR 77
DAC N
SUB NX
SPA
DIO N
LAC N
SZA I
JMP VECT2
SAD NX
JMP XGTR
CLA
LIO SDX
SIL I
DIV N

ALPHA:
0
DAC ALPHA
LAW 100
DAC BETA
JMP APRES

XGTR:
LAW 100
DAC ALPHA
CLA
LIO SDY
SIL I
DIV N

BETA:
0
DAC BETA

APRES:
MUL N
LAI
SAR I
CMA
ADD SDY
DAC SDY
LAC ALPHA
MUL N
LAI
SAR I
CMA
ADD SDX
DAC SDX
LAC ALPHA
SAL 377
IOR BETA
SZF 1
IOR [100000]
SZF 5
IOR [200]
IOR [400000]
LIA
LAC N
CMA
DAC N

VECT1:
  DIO I P
  IDX P
  ISP N
  JMP VECT1

VECT2:
  LAC SDY
  SCR 377
  LAC SDX
  SCL 377
  SZF 1
  IOR [100000]
  SZF 5
  IOR [200]
  IOR [400000]
  CLF 1
  CLF 5
  DAC I P
  IDX P
  JMP COMMON
SUIT1: SZF I 3
      JMP SUIT2
      LIO CHY
      DIO N
      LIO CHX
      DAC CHX
      IDX J
      JSP CALCUL
      DAC CHY
      LAC N
      JMP MIN

CALCUL: DAP CALCUE
        LAC I J
        SZF 6
        CMA

CALCUE: JMP .
SUIT2:  SAS CHX
       JMP ERR1
       LIA
       IDX J
       JSP CALCUL
       SAD CHY
       JMP MIN

ERR1:  IOT 17

TEXT //
OPERATION : UNDEFINED FOR THOSE TWO PRIMITIVES//
       JMP READ
PASLET: SAD [-1]
STF 6
SAD [23]
JMP ERASE
SAD [4]
JMP STORE
SAD [15]
JMP INIT
SAD [-10]
JMP BACK
SAD [22]
JMP MARK
SAD [21]
DAC BLANK
CDF
CLF 7
SAD [11]
STF 1
SAD [6]
STF 2
SAD [3]
STF 3
SAD [-2]
STF 4
SZF 1 7
CDF
JMP READ
MARK: LAC P
       DAC N
       LAW MARKIX
       DAC J

LOOP2: LAC I J
       DAC I N
       IDX N
       IDX J
       SAS [MARKTX+6]
       JMP LOOP2
       LAC T
       DAC MAX
       LAC N
       JMP DISPLAY+1
STORE:

IOT 7
AND [77]
SAD [71]
JMP INIT
SUB [22]
SPA
JMP ERR2
SUB [32]
SMA
JMP ERR2
ADD [TA+44]
DAP .+2
LAC S
DAC .
LAC CHX
DAC I S
IDX S
LAC CHY
DAC I S
IDX S
ADD P
SUB [DB+1]
DAC I S
IDX S
LAW DB+2
DAC J

LOOP3:

LAC I J
DAC I S
IDX S
IDX J
SAS P
JMP LOOP3
JMP READ
ERASE:       LAC T
           SAD [AP+3]
           JMP READ
           SUB [6]
           DAC T

ERAS:       LAC I T
           DAC P
           IDX T
           LAC I T
           DAC CHX
           IDX T
           LAC I T
           DAC CHY
           IDX T
           JMP DISPLAY

ERR2:       IOT 17

TEXT //
ERROR...GIVE A LETTER PLEASE//
           JMP STORE
BACK: LAC T
SAD MAX
JMP READ
JMP ERAS
AI03:  DB
       30000
NMT:   0
NMD:   0
TNMD:  0
MARKTX: 747670
       310000
       747614
SKLT:  600402
       747670
       450000
TA:    ST
       ST+4
       ST+10
       ST+14
       ST+20
       ST+24
       ST+30
       ST+34
       ORG .+34
AP:    ORG .+600
DB:    747617
       600402
       ORG .+510
ST:  40
    0
  .+2  020000  40
  .+2  020040  0
  .+2  000040  -40
  .+2  120040  -40
        0
  .+2  120000  -40
        -40
  .+2  120240  0
        -40
  .+2  000240  40
        -40
  .+2  020240
END