Dynamic NMR Microscopy

動態核磁顯微成像

VOLUME TWO

Appendices: Software Development

A thesis presented in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy in Physics
at
Massey University

by

Yang Xia

夏 陽

1992

Contents

Appendix 1 TI-980A Software Development

The TI-980A software development was based on the imaging software which had been developed from the original JEOL assembly language source code. The JEOL source code is in the form of machine language. No disassembler package is available for this source code in the TI-980A computer. Therefore any modification has to be done at the machine code level.

A1.1 Quadrant phase compensation

The routine for the quadrant phase compensation, which is used for the flow imaging, follows the **k**-gradient update routine.

```
...
36F1    7C00    BRU     (36F2)      ; at the end of k-gradient update
36F2    3760    DATA                ;   routine, go to adjust phase P0
36F3    CE00

...
375F    CE00    IDL
3760    0400    LDA     ((3761))    ; load q-gradient level
3761    341F    DATA                ;   address (3500-)
3762    271F    ADD     1F          ; add 1F16 so that (A) contains
3763    C506    RMO     A, B        ;   the current q phase address

3764    0900    LDE     ((B) + 0)   ; load E register with phase value P0
3765    0400    LDA     ((3766))    ; load Gx sign (0=+Gx, 1=-Gx)
3766    3211    DATA
3767    CC80    SNZ                 ; go to 3769 if Gx -ve
3768    780D    BRU     3776        ; go to 3776 if Gx +ve

3769    C510    RMO     E, A        ; add or sub the phase adjustment
376A    2F2C    SUB     2C          ;   when Gx -ve
376B    C501    RMO     A, E        ;   (27xx = add xx; 2Fxx = sub xx)
```

```
376C    0400    LDA     ((376D))    ; load Gz sign (0=+Gz, 1=-Gz)
376D    3210    DATA
376E    CC80    SNZ                 ; go to 3770 if Gz -ve (#3 quadrant)
376F    7803    BRU     3773        ; go to 3773 if Gz +ve (#2 quadrant)

3770    C510    RMO     E, A        ; add or sub the phase adjustment
3771    2F01    SUB     1           ;   when Gz -ve and Gx -ve
3772    C501    RMO     A, E        ;   (27xx = add xx; 2Fxx = sub xx)

3773    8C00    STE     ((3774))    ; store phase value P0
3774    0245    DATA
3775    7809    BRU     377F        ; done phase compensation

3776    0400    LDA     ((3777))    ; load Gz sign when +ve Gx
3777    3210    DATA
3778    CC80    SNZ                 ; go to 377A if -Gz (#4 quadrant)
3789    7803    BRU     377D        ; go to 377D if +Gz (#1 quadrant)

377A    C510    RMO     E, A        ; add or sub the phase adjustment
377B    2F03    SUB     3           ;   when Gz -ve and Gx +ve
377C    C501    RMO     A, E        ;   (27xx = add xx; 2Fxx = sub xx)

377D    8C00    STE     ((377E))    ; store phase value P0
377E    0245    DATA                ; done phase compensation

377F    0000    LDA     (3780)      ; load base register address
3780    0200    DATA
3781    C506    RMO     A, B        ; 0200 -> (B)
3782    0145    LDA     ((B) + 45)  ; load (0245), phase value ZP0
3783    C881    ALA     1
3784    9903    MPY     ((B) + 3)   ; multiply (0203), ZN
3785    5800    DIV     (3786)      ; divided by 720
3786    02D0    DATA
3787    8184    STA     ((B) + 84)  ; store at (0284)
3788    7C00    BRU     (3789)      ; done!
3789    1F38
378A    CE00
```

Note: several addresses in TI-980A memory

0200		; address of base register
0203	ZN	; number of data points
0245	ZP0	; phase value
0284	ZZP0	; this is the phase which is used!

A1.2 "One-shot" velocity microscopy

The software development for the 'one-shot' velocity microscopy was based on the original static PR imaging software. The major modification is the rf and gradient phase cycling routine. In the software, there are three places where the phases of rf and gradient pulses need to be altered: the phase cycling routine which is cycled for every signal acquisition in a four step sequence, the projection reset routine which resets the rf and gradient phases every time when the **k** gradient is updated, and the 'grand' reset routine which resets the rf and gradient phases when a new imaging experiment is started.

There are two different versions of the phase cycling routine: the v-sensitive version and calibration version. They are placed at the end of the CNC (Coherent Noise Cancellation) routine. These two phase cycling routines are listed as the following with detailed comments. The rf and gradient phase reset commands used in the projection and 'grand' reset routines are similar with that used in the two phase cycling routines so that only the machine level codes are given.

The phase of the rf and gradient pulses are set via the spare bits at the '04' register. However, the 'natural' logic and the logic levels in the TI-980A memory are inverted due to historical reasons, as the following:

Logic of '04' register

bit	(a) in theory								(b) in the TI-980A memory								
	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F	
	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0^o phase
	0	0	1	0	0	0	0	0	1	1	0	1	1	1	1	1	90^o phase
	0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	180^o phase
	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	-ve G_z or G_y
	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	-ve G_x

A1.2.1 The phase cycling routine of the calibration version

```
...
374F    0000    DATA                ; cycling counter (1 to 4)
...
3804    CE00    IDL
3805    0400    LDA     ((3806))    ; load CNC flag
3806    0231    DATA
3807    6700    CPL     0           ; CNC required?
3808    CDA0    SNE                 ; yes, go to 380A
3809    78E0    BRU     37EA        ; no, go to 37EA

380A    0000    LDA     (380B)      ; load the starting address of
380B    3000    DATA                ;  the pulse sequence
380C    C506    RMO     A, B        ;  and save it

380D    0100    LDA     ((B) + 0)

380E    6704    CPL     4           ; is it rf command (04 register)?
380F    CDA0    SNE                 ; no, go to 3811
3810    7804    BRU     3815        ; yes, go to 3815

3811    6700    CPL     0           ; is it the end of the pulse?
3812    CDA0    SNE
3813    782E    BRU     3842        ; no, go to 383F
3814    7829    BRU     383E        ; yes, go to 3843
```

```
3815    0400    LDA     ((3816))    ; have found 04 register, now
3816    374F    DATA                ;   determine cycling sequence

3817    6701    CPL     1           ; is it the first acquisition?
3818    CDA0    SNE                 ; no, go to 381A
3819    780E    BRU     3828        ; yes, go to 3828

381A    6702    CPL     2           ; is it the second acquisition?
381B    CDA0    SNE                 ; no, go to 381D
381C    7812    BRU     382F        ; yes, go to 382F

381D    6703    CPL     3           ; is it the third acquisition?
381E    CDA0    SNE                 ; no, must be the 4th, go to 3820
381F    7817    BRU     3837        ; go to 3837

3820    0101    LDA     ((B) + 1)   ; the current acquisition is the forth
3821    DB1D    TABO    D           ;   so have to prepare for the first
3822    7802    BRU     3825        ;   use +g and 0° rf phase (add)
3823    DB4D    SABZ    D           ; set bit D to ZERO
3824    7801    BRU     3826
3825    DB5D    SABO    D           ; set bit D to ONE
3826    DB5F    SABO    F           ; set bit F to ONE
3827    7815    BRU     383D        ; done

3828    0101    LDA     ((B) + 1)   ; the current acquisition is the first
3829    DB1D    TABO    D           ;   so have to prepare for the second
382A    7802    BRU     382D        ;   use +g and 180° rf phase (add)
382B    DB4D    SABZ    D           ; set bit D to ZERO
382C    7801    BRU     382E
382D    DB5D    SABO    D           ; set bit D to ONE
382E    780E    BRU     383D        ; done

382F    0101    LDA     ((B) + 1)   ; the current acquisition is the second
3830    DB1D    TABO    D           ;   so have to prepare for the third
3831    7802    BRU     3834        ;   use -g and 0° rf phase (add)
3832    DB4D    SABZ    D           ; set bit D to ZERO
3833    7801    BRU     3835
3834    DB5D    SABO    D           ; set bit D to ONE
```

```
3835    DB4F    SABZ    F           ; set bit F to ZERO
3836    7806    BRU     383D        ; done

3837    0101    LDA     ((B) + 1)   ; the current acquisition is the third
3838    DB1D    TABO    D           ;  so have to prepare for the forth
3839    7802    BRU     383C        ;  use -g and 180° rf phase (add)
383A    DB4D    SABZ    D           ; set bit D to ZERO
383B    7801    BRU     383D
383C    DB5D    SABO    D           ; set bit D to ONE, done

383D    8101    STA     ((B) + 1)   ; search for the other rf pulses in
383E    C560    RMO     B, A        ;  the pulse sequence
383F    2705    ADD     5
3840    C506    RMO     A, B
3841    78CB    BRU     380D        ; go to to search again

3842    7C00    BRU     (3843)      ; searching finished!
3843    37E0    DATA
3844    CE00    IDL
```

A1.2.2 The phase cycling routine of the v-sensitive version

```
...
374F    0000    DATA                ; cycling counter (1 to 4)
...
3804    CE00    IDL
3805    0400    LDA     ((3806))    ; load CNC flag
3806    0231    DATA
3807    6700    CPL     0           ; CNC required?
3808    CDA0    SNE                 ; yes, go to 380A
3809    78E0    BRU     37EA        ; no, go to 37EA

380A    0000    LDA     (380B)      ; load the starting address of
380B    3000    DATA                ;  the pulse sequence
380C    C506    RMO     A, B        ;  and save it

380D    0100    LDA     ((B) + 0)
```

```
380E    6704    CPL     4       ; is it rf command (04 register)?
380F    CDA0    SNE             ; no, go to 3811
3810    7804    BRU     3815    ; yes, go to 3815

3811    6700    CPL     0       ; is it the end of the pulse?
3812    CDA0    SNE
3813    782A    BRU     383E    ; yes, go to 383E
3814    7825    BRU     383A    ; no, go to 383A

3815    0400    LDA     ((3816)); have found 04 register, now
3816    374F    DATA            ;  determine cycling sequence

3817    6701    CPL     1       ; is it the first acquisition?
3818    CDA0    SNE             ; no, go to 381A
3819    780B    BRU     3825    ; yes, go to 3825

381A    6702    CPL     2       ; is it the second acquisition?
381B    CDA0    SNE             ; no, go to 381D
381C    7811    BRU     382E    ; yes, go to 382E

381D    6703    CPL     3       ; is it the third acquisition?
381E    CDA0    SNE             ; no, must be the 4th, go to 3820
381F    7812    BRU     3832    ; go to 3832
3820    7800    BRU     3821

3821    0101    LDA     ((B) + 1); the current acquisition is the forth
3822    DB5F    SABO    F       ;  so have to prepare for the first
3823    8101    STA     ((B) + 1);  use +g and 0° rf phase (add)
3824    7815    BRU     383A    ; done

3825    0101    LDA     ((B) + 1); the current acquisition is the first
3826    DB1D    TABO    D       ;  so have to prepare for the second
3827    7802    BRU     382A    ;  use +g and 180° rf phase (add)
3828    DB4D    SABZ    D       ; set bit D to ZERO
3829    7801    BRU     382B
382A    DB5D    SABO    D       ; set bit D to ONE
382B    DB5F    SABO    F       ; set bit F to ONE
```

```
382C    8101    STA     ((B) + 1)
382D    780C    BRU     383A        ; done

382E    0101    LDA     ((B) + 1)   ; the current acquisition is the second
382F    DB4F    SABZ    F           ;   so have to prepare for the third
3830    8101    STA     ((B) + 1)   ;   use -g and 180° rf phase (sub.)
3831    7808    BRU     383A        ; done

3832    0101    LDA     ((B) + 1)   ; the current acquisition is the third
3833    DB1D    TABO    D           ;   so have to prepare for the forth
3834    7802    BRU     3837        ;   use -g and 0° rf phase (sub.)
3835    DB4D    SABZ    D           ; set bit D to ZERO
3836    7801    BRU     3838
3837    DB5D    SABO    D           ; set bit D to ONE
3838    DB4F    SABZ    F           ; set bit F to ZERO
3839    8101    STA     ((B) + 1)   ; done

383A    C560    RMO     B, A        ; search for the other rf pulses in
383B    2705    ADD     5           ;   the pulse sequence
383C    C506    RMO     A, B
383D    78CF    BRU     380D        ; go to to search again

383E    7C00    BRU     (383F)      ; searching finished!
383F    37E0    DATA
3841    CE00    IDL
```

A1.2.3 Other modifications in the 'one-shot' software

These modifications are common to both the velocity-sensitive and calibration experiments.

Phase cycling counter update routine:

```
37DF    CE00    IDL
37E0    0400    LDA     ((37E1))    ; load phase cycling counter
37E1    374F    DATA
37E2    6704    CPL     4           ; =4?
```

```
37E3    CDA0    SNE             ; no, go to 37E5
37E4    7802    BRU     37E7    ; yes, go to 37E7
37E5    2701    ADD     1       ; not 4, add 1
37E6    7801    BRU     37E8    ; go to 37E8
37E7    0701    LDA     1       ; =4, reset to 1
37E8    8400    STA     ((37E9)); save it
37E9    374F    DATA
37EA    7C00    BRU     (37EB)  ; go to the transfer routine, transfer
37EB    1F38    DATA            ;  the pulse to the pulse sequencer
37EC    CE00    IDL
```

Projection reset routine:

This routine is executed every time when the **k** gradient is updated. Note that the first part of this routine (from 3690 to 36F0) is the 360° **k**-gradient update routine which is identical with that used in the flow imaging software. At the end of the gradient update routine, the phases of the rf and gradient pulses are reset.

```
3844    CE00    IDL             ; this is the end of the CNC routine
3845    7C00    BRU     (3846)
3846    3690    DATA            ; go to reset pulse phases
3847    CE00
...

3690    0700    8400    321F    0400    3200    9C00    321F    C510
3698    2000    3220    2400    321E    C501    0000    3210    2400
36A0    321F    C504    C516    0100    C546    8100    C311    0400
36A8    321F    2000    3201    C504    C516    0100    C546    8500
36B0    5400    321F    0400    321F    6702    CD20    78DC    0000
36B8    3000    C506    0100    6704    CDA0    7807    6700    CDA0
36C0    781F    C560    2705    C506    78F5    C366    7C00    36D2
36C8    CE00    CE00    CE00    CE00    CE00    CE00    CE00    CE00
36D0    CE00    CE00    0400    3211    6701    CDA0    7804    0100
36D8    DB58    8100    7803    0100    DB48    8100    C766    78E1
36E0    0400    384F    6701    CD20    7804    5400    321E    5400
36E8    321E    0400    3200    6400    321E    CD40    7802    7C00
36F0    0200
```

36F0		0701	8400	374F	8400	374E	0000	3000
36F8	C506	0100	6704	CDA0	7804	6700	CDA0	7823
3700	781E	0400	374E	6701	CD20	7814	0101	DB0D
3708	780B	0702	8400	374E	0101	DB0D	7802	DB5D
3710	7801	DB4D	DB5F	780A	0703	8400	374E	0101
3718	DB5F	7804	6702	CD20	78FA	78EE	8101	C560
3720	2705	C506	78D6	7C00	1F38	CE00	CE00	CE00

Note: At the end of the projection reset routine, the address 1F38 starts to transfer the current pulse sequence to the TI-980A computer.

"Grand" reset routine:

This routine is only executed every time when a new imaging experiment is going to be started, through the manual 'IMAGING' <=> 'NORMAL' reset.

1CA0	0700	LDA	0	
1CA1	8400	STA	((1CA2))	
1CA2	321F	DATA		; gradient counter
1CA3	8400	STA	((1CA4))	
1CA4	321E	DATA		; projection counter
1CA5	8400	STA	((1CA6))	
1CA6	321D	DATA		
1CA7	8400	STA	((1CA8))	
1CA8	3654	DATA		
1CA9	7C00	BRU	(1CAA)	; go to reset rf and gradient phases
1CAA	3751	DATA		
1CAB	CE00			

...

3748	CE00	CE00	CE00	CE00	CE00	CE00	0003	0004
3750	CE00	0701	8400	374F	8400	374E	0000	3000
3758	C506	0100	6704	CDA0	7804	6700	CDA0	7823
3760	781E	0400	374E	6701	CD20	7814	0101	DB0D
3768	780B	0702	8400	374E	0101	DB0D	7802	DB5D
3770	7801	DB4D	DB5F	780A	0703	8400	374E	0101
3778	DB5F	7804	6702	CD20	78FA	78EE	8101	C560
3780	2705	C506	78D6	7C00	1C30	CE00	CE00	CE00

Note: (374E) and (374F) are two counters for phase cycling. At the end of the projection reset routine, the address 1C30 starts the imaging procedure.

A1.3 Flow imaging at transverse direction (x-direction)

The software modification for the flow imaging along the transverse direction (x) was based on the longitudinal direction (y) flow imaging software. Refer to Chapter 8 for more details.

```
3787    Quadrant phase compensation routine finished.
3788    0400    LDA     ((3789))    ; load Gx sign
3789    3211    DATA
378A    CC80    SNZ                 ; go to 378C if Gx -ve
378B    7802    BRU     378E        ; go to 378E if Gx +ve
378C    07BF    LDA     BF          ; switch the extra rf pulse ON
378D    7801    BRU     378F
378E    07FF    LDA     FF          ; switch the extra rf pulse OFF
378F    C501    RMO     A, E        ; save into 3169 which is the
3790    8C00    STE     ((3791))    ;  address of ON/OFF of extra
3791    3169    DATA                ;  90° rf pulse in VDSEM6XB
3792    7C00    BRU     (3793)      ; done!
3793    1F38    DATA
3794    CE00    IDL
```

A1.4 128×128 image reconstruction

```
...
1DA4    0100    DATA                ; 100(16) = 2×128
...
1569    615A    CPL     ((B) + 5A)  ; 128 points, at (025A)
156A    CDC0    SLE
156B    015A    LDA     ((B) + 5A)  ; 128 points, at (025A)
```

A2 Hitachi computer software

A2.1 128×128 back-projection subroutine at 360°

```
;*************************************************************************
;*                                BP128C.ASM                             *
;*     Intel-8088 assembly routine for the back projection operation in the filtered BP imaging  *
;*                (128x128 images, 360 degree projection, square pixels)  *
;*                (modified from an assembly routine written by C.D. Eccles)  *
;********************************************************** 25/4/90 **********

FRAME   STRUC
SAVEBP  DW              ; Storage for base pointer
SAVERT  DD              ; Storage for return address
T       DD      ?       ; Projection angle
ISEG    DD      ?       ; Data segment of image
DSEG    DD      ?       ; Data segment of filtered profile
FRAME   ENDS


DATA    SEGMENT
;
; SINE LOOK UP TABLE [ sin(P0) for 0<=P0<=90 ]
;
TRIG    DW      0000H, 011EH, 023CH, 0359H, 0477H, 0594H, 06B1H, 07CDH
        DW      08E8H, 0A03H, 0B1DH, 0C36H, 0D4EH, 0E66H, 0F7CH, 1090H
        DW      11A4H, 12B6H, 13C7H, 14D6H, 15E4H, 16F0H, 17FAH, 1902H
        DW      1A08H, 1B0CH, 1C0EH, 1D0EH, 1E0CH, 1F07H, 2000H, 20F6H
        DW      21EAH, 22DBH, 23CAH, 24B5H, 259EH, 2684H, 2767H, 2847H
        DW      2923H, 29FDH, 2AD3H, 2BA6H, 2C75H, 2D41H, 2E0AH, 2ECEH
        DW      2F90H, 304DH,3107H, 31BDH, 326FH, 331DH, 33C7H, 346DH
        DW      350FH, 35ADH, 3646H, 36DCH, 376DH, 37FAH, 3882H, 3906H
        DW      3986H, 3A01H, 3A78H, 3AEAH, 3B57H, 3BC0H, 3C24H, 3C83H
        DW      3CDEH, 3D34H, 3D85H, 3DD2H, 3E19H, 3E5CH, 3E9AH, 3ED3H
        DW      3F07H, 3F36H, 3F61H, 3F86H, 3FA6H, 3FC2H, 3FD8H, 3FEAH
```

```
        DW      3FF6H, 3FFEH, 4000H

THETA   DW      ?       ; Back projection angle
TORIG   DW      ?       ; Original angle
COS     DW      ?       ; Cos(theta)
SIN     DW      ?       ; Sin(theta)
X       DW      ?       ; Horizontal coordinate
Y       DW      ?       ; Vertical coordinate
ADRS    DW      ?       ; Address of profile data to be projected
SEGI    DW      ?       ; Segment for image data
SEGD    DW      ?       ; Segment of incoming data
DATA    ENDS

CODE    SEGMENT 'CODE'
DGROUP  GROUP   DATA
        ASSUME  CS:CODE, ES:DGROUP, DS:DGROUP

BP128C  PROC    FAR
        PUBLIC  BP128C
        PUSH    BP              ; Save caller's BP
        MOV     BP, SP          ; Set frame base
        PUSH    DS

        MOV     AX, DATA
        MOV     DS, AX          ; Data base segment

        LES     BX, [BP]+T
        MOV     AX, ES:[BX]
        MOV     TORIG, AX
        SHL     AX, 1
        MOV     THETA, AX       ; Load 2 * projection angle

        LES     BX, [BP]+ISEG
        MOV     AX, ES:[BX]
        MOV     SEGI, AX

        LES     BX, [BP]+DSEG
        MOV     AX, ES:[BX]
```

```
        MOV     SEGD, AX        ; Load data segment

; Sine and cosine calculation

        MOV     BX, OFFSET TRIG
        CMP     THETA, 360      ; Is THETA > 360?
        JL      GT0
        SUB     THETA, 360

GT0:    CMP     THETA, 180      ; Is ANGLE>90?
        JG      GT90            ; Yes, go to GT90

        MOV     AX, THETA       ; 0<=ANGLE<=90
        ADD     BX, AX
        MOV     AX, DS:[BX]     ; AX=SIN(ANGLE)
        MOV     SIN, AX

        MOV     BX, OFFSET TRIG
        MOV     AX, 180
        SUB     AX, THETA
        ADD     BX, AX
        MOV     AX, DS:[BX]     ; AX=COS(ANGLE)
        MOV     COS, AX
        JMP     CALC

GT90:   MOV     AX, 360         ; 90<ANGLE<=180
        SUB     AX, THETA       ; 180-ANGLE
        ADD     BX, AX
        MOV     AX, DS:[BX]     ; AX=SIN(180-ANGLE)
        MOV     SIN, AX

        MOV     BX, OFFSET TRIG
        MOV     AX, THETA
        SUB     AX, 180         ; ANGLE-90
        ADD     BX, AX
        MOV     AX, DS:[BX]
        NEG     AX              ; AX=-COS(ANGLE-90)
```

```
                MOV     COS, AX

; Square pixels

CALC:           MOV     AX, 100         ; If changing AX from 100 to 128, but keeping
                IMUL    SIN             ;    BX as 100, the pixel size would be
                MOV     BX, 100         ;    rectangular
                IDIV    BX
                MOV     SIN, AX

; Determine R=X*COS(THETA)+Y*SIN(THETA)

; Initialise X and Y values
                MOV     AX, -64         ; Half the pixel array size in X direction
                MOV     X, AX
                MOV     AX, -64         ; Half the pixel array size in Y direction
                MOV     Y, AX           ; Because X=Y, array is square

LOOP:           MOV     AX, X
                ADD     AX, 64
                SHL     AX, 1           ; X=X*2
                MOV     BX, Y
                ADD     BX, 64
                MOV     CL, 8
                SHL     BX, CL          ; Y=Y*128
                ADD     AX, BX
                MOV     ADRS, AX        ; ADRS=X*2+Y*128

                MOV     AX, X
                IMUL    COS             ; DX:AX=X*COS(THETA)
                MOV     DI, AX
                MOV     CX, DX          ; CX:DI=X*COS(THETA)
                MOV     AX, Y
                IMUL    SIN             ; DX:AX=Y*SIN(THETA)

                CLC
                ADC     AX, DI
                ADC     DX, CX          ; DX:AX=X*COS(THETA)+Y*SIN(THETA)
```

```
        CLC
        ADC     AX, 8192
        ADC     DX, 0           ; DX:AX=INT((DX,AX+.5))*4000H

        SAL     DX, 1
        SAL     DX, 1           ; 2^(-14) = 2^(-16) * 2^2
        JNO     A1              ; If the data was positive skip the next line
        OR      DX, 8000H       ; Keep negative

A1:     MOV     CL, 14
        SHR     AX, CL          ; AX=AX/4000H
        ADD     DX, AX          ; DX=INT(X*COS(THETA)+Y*SIN(THETA)+.5)

; check if angle>=180 degree

        MOV     AX, TORIG
        CMP     AX, 180
        JL      TFRV
        NEG     DX
        DEC     DX              ; DX --> -DX-1

; Test for R value outside range -32<R<31

TFRV:   CMP     DX, 63
        JG      NCAR
        CMP     DX, -64
        JL      NCAR            ; If DX<-32 or DX>31 then skip calculation

; Load projection value

        ADD     DX, 64          ; Remove all negative values
        SHL     DX, 1
        MOV     BX, DX
        MOV     AX, SEGD
        MOV     ES, AX
        MOV     DX, ES:[BX]     ; Load projection value
```

```
                MOV     AX, SEGI        ; Sample base segment for image data
                MOV     ES, AX

                MOV     BX, ADRS
                MOV     AX, ES:[BX]
                ADD     AX, DX
                MOV     ES:[BX], AX     ; (ADRS)=(ADRS)+DX {Add projection value}

; Next x

NCAR:           INC     X
                MOV     AX, X
                CMP     AX, 63          ; Increment X and test
                JG      LOOP1
                JMP     LOOP

; Next y

LOOP1:          MOV     AX, -64
                MOV     X, AX
                INC     Y

                MOV     AX, Y
                CMP     AX, 63          ; Increment Y and test
                JG      FIN
                JMP     LOOP

FIN:            POP     DS
                POP     BP
                RET     12              ; Return to calling program

BP128C          ENDP

CODE            ENDS
                END
```

A2.2 128×128 back-projection subroutine at 180°

```
;*****************************************************************************
;*                                  BP128D.ASM                                 *
;*     Intel-8088 assembly routine for the back projection operation in the filtered BP imaging     *
;*                  (128x128 images, 180 degree projection, square pixels)                       *
;*                  (modified from an assembly routine written by C.D. Eccles)                   *
;************************************************************ 4/5/90 ***************

FRAME   STRUC
SAVEBP  DW              ; Storage for base pointer
SAVERT  DD              ; Storage for return address
T       DD      ?       ; Projection angle
ISEG    DD      ?       ; Data segment of image
DSEG    DD      ?       ; Data segment of filtered profile
FRAME   ENDS

DATA    SEGMENT
;
; SINE LOOK UP TABLE [ sin(P0) for 0<=P0<=90 ]
;
TRIG    DW      0000H, 011EH, 023CH, 0359H, 0477H, 0594H, 06B1H, 07CDH
        DW      08E8H, 0A03H, 0B1DH, 0C36H, 0D4EH, 0E66H, 0F7CH, 1090H
        DW      11A4H, 12B6H, 13C7H, 14D6H, 15E4H, 16F0H, 17FAH, 1902H
        DW      1A08H, 1B0CH, 1C0EH, 1D0EH, 1E0CH, 1F07H, 2000H, 20F6H
        DW      21EAH, 22DBH, 23CAH, 24B5H, 259EH, 2684H, 2767H, 2847H
        DW      2923H, 29FDH, 2AD3H, 2BA6H, 2C75H, 2D41H, 2E0AH, 2ECEH
        DW      2F90H, 304DH,3107H, 31BDH, 326FH, 331DH, 33C7H, 346DH
        DW      350FH, 35ADH, 3646H, 36DCH, 376DH, 37FAH, 3882H, 3906H
        DW      3986H, 3A01H, 3A78H, 3AEAH, 3B57H, 3BC0H, 3C24H, 3C83H
        DW      3CDEH, 3D34H, 3D85H, 3DD2H, 3E19H, 3E5CH, 3E9AH, 3ED3H
        DW      3F07H, 3F36H, 3F61H, 3F86H, 3FA6H, 3FC2H, 3FD8H, 3FEAH
        DW      3FF6H, 3FFEH, 4000H

THETA   DW      ?       ; Back projection angle
COS     DW      ?       ; Cos(theta)
SIN     DW      ?       ; Sin(theta)
X       DW      ?       ; Horizontal coordinate
```

```
Y       DW      ?       ; Vertical coordinate
ADRS    DW      ?       ; Address of profile data to be projected
SEGI    DW      ?       ; Segment for image data
SEGD    DW      ?       ; Segment of incoming data
DATA    ENDS


CODE    SEGMENT 'CODE'
DGROUP  GROUP   DATA
        ASSUME  CS:CODE, ES:DGROUP, DS:DGROUP

BP128D  PROC    FAR
        PUBLIC  BP128D
        PUSH    BP              ; Save caller's BP
        MOV     BP, SP          ; Set frame base
        PUSH    DS

        MOV     AX, DATA
        MOV     DS, AX          ; Data base segment

        LES     BX, [BP]+T
        MOV     AX, ES:[BX]
        SHL     AX, 1
        MOV     THETA, AX       ; Load 2 * projection angle

        LES     BX, [BP]+ISEG
        MOV     AX, ES:[BX]
        MOV     SEGI, AX

        LES     BX, [BP]+DSEG
        MOV     AX, ES:[BX]
        MOV     SEGD, AX        ; Load data segment

;       Sine and cosine calculation

        MOV     BX, OFFSET TRIG
        CMP     THETA, 180      ; Is THETA>90?
```

```
                JG      GT180           ; Yes

                MOV     AX, THETA
                ADD     BX, AX
                MOV     AX, DS:[BX]     ; AX=SIN(THETA)
                MOV     SIN, AX

                MOV     BX, OFFSET TRIG
                MOV     AX, 180
                SUB     AX, THETA
                ADD     BX, AX
                MOV     AX, DS:[BX]     ; AX=COS(THETA)
                MOV     COS, AX
                JMP     CALC

GT180:          MOV     AX, 360         ; 90>THETA>=180
                SUB     AX, THETA
                ADD     BX, AX
                MOV     AX, DS:[BX]     ; AX=SIN(THETA)
                MOV     SIN, AX

                MOV     BX, OFFSET TRIG
                MOV     AX, THETA
                SUB     AX, 180
                ADD     BX, AX
                MOV     AX, DS:[BX]
                NEG     AX              ; AX=-COS(THETA)
                MOV     COS, AX

; Square pixels

CALC:           MOV     AX, 100         ; Square pixels because AX=BX
                IMUL    SIN
                MOV     BX, 100
                IDIV    BX
                MOV     SIN, AX

; Determine R=X*COS(THETA)+Y*SIN(THETA)
```

```
        MOV     AX, -64         ; Square pixels because X=Y
        MOV     X, AX
        MOV     AX, -64
        MOV     Y, AX           ; Initialise X and Y values

LOOP:   MOV     AX, X
        ADD     AX, 64
        SHL     AX, 1           ; X=X*2
        MOV     BX, Y
        ADD     BX, 64
        MOV     CL, 8
        SHL     BX, CL          ; Y=Y*256
        ADD     AX, BX
        MOV     ADRS, AX        ; ADRS=X*2+Y*256

        MOV     AX, X
        IMUL    COS             ; DX:AX=X*COS(THETA)
        MOV     DI, AX
        MOV     CX, DX          ; CX:DI=X*COS(THETA)
        MOV     AX, Y
        IMUL    SIN             ; DX:AX=Y*SIN(THETA)

        CLC
        ADC     AX, DI
        ADC     DX, CX          ; DX:AX=X*COS(THETA)+Y*SIN(THETA)
        CLC
        ADC     AX, 8192
        ADC     DX, 0           ; DX:AX=INT((DX,AX+.5))*4000H

        SAL     DX, 1
        SAL     DX, 1           ; 2^(-14) = 2^(-16) * 2^2
        JNO     A1              ; If the data was positive skip the next line
        OR      DX, 8000H       ; Keep negative

A1:     MOV     CL, 14
        SHR     AX, CL          ; AX=AX/4000H
```

```
            ADD     DX, AX          ; DX=INT(X*COS(THETA)+Y*SIN(THETA)+.5)

; Test for R value outside range -64<R<63

            CMP     DX, 63
            JG      NCAR
            CMP     DX, -64
            JL      NCAR            ; If DX<-64 or DX>63 then skip calculation

; Load projection value

            ADD     DX, 64          ; Remove all negative values
            SHL     DX, 1
            MOV     BX, DX

            MOV     AX, SEGD
            MOV     ES, AX
            MOV     DX, ES:[BX]     ; Load projection value

            MOV     AX, SEGI        ; Sample base segment for image data
            MOV     ES, AX

            MOV     BX, ADRS
            MOV     AX, ES:[BX]
            ADD     AX, DX
            MOV     ES:[BX], AX     ; (ADRS)=(ADRS)+DX {Add proj. value}

; Next x

NCAR:       INC     X
            MOV     AX, X
            CMP     AX, 63          ; Increment X and test
            JG      LOOP1
            JMP     LOOP

; Next y

LOOP1:      MOV     AX, -64
```

```
                MOV     X, AX
                INC     Y

; Update data segment if y>=0

                MOV     AX, Y
                CMP     AX, 63          ; Increment Y and test
                JG      FIN
                JMP     LOOP

FIN:            POP     DS
                POP     BP

                RET     12              ; Return to calling program

BP128D          ENDP

CODE            ENDS
                END
```

A2.3 128×128 PR image reconstruction program

```
**************************************************************************
*                                FBP128.FOR                              *
*     This program accepts filtered profile data (128x128) from the FX-60 spectrometer and  *
*                back project into ONE square images.                    *
*  [Various Intel-8088 assembly subroutines can be found in Eccles (1987) and Xia (1988) thesises] *
********************************************************** 4/5/90 *********

$STORAGE: 2

      PROGRAM  FBP128

c Data types

      INTEGER*2     AMP, PA, SF, SFG, THETA, INC, W, DSEG
      INTEGER*2     CSC, RESP, RESP1, MAX
      REAL          AMPR

c Constants

      DSEG = 16000
      MAX = -9999
      SFG = 7

c Set screen mode 640*400

      CALL SCMD(183)
      CALL RESET
      CALL SETCOL(0, 14, 0, 0)

 ********************************** Input data ***********************************

      WRITE (*, 1)
1     FORMAT( ' *** Image array = 128x128 , Square pixels, PR *** ' )
      WRITE (*, 2)
2     FORMAT( ' ' )
```

```
      WRITE (*, 5)
5     FORMAT( ' Enter angle increment (XX) ')
      READ (*, 10) INC
10    FORMAT( I2)

      WRITE (*, 15)
15    FORMAT( ' Do you wish to set a scale factor (Y=1 N=0)')
      READ (*, 20) RESP
20    FORMAT( I1)

      IF(RESP .EQ. 1) THEN
            WRITE(*, 25)
25          FORMAT( ' Enter scale factor (XXX)')
            READ (*, 30) SF
30          FORMAT( I3)
      ENDIF

      WRITE (*, 35)
35    FORMAT( ' Enter total projection angle (XXX) ')
      READ (*, 40) PA
40    FORMAT( I3)

      WRITE (*, 45)
45    FORMAT( ' Enter colour scale code (X) ')
      READ (*, 50) CSC
50    FORMAT( I1)

****************** FILTER BACK PROJECTED DATA IMAGES LOOP ******************

c Remove cursor

      CALL CURSOR(1)

      DO 90 THETA=0, PA-INC, INC

c Obtain data from spectrometer(only use the 1st half data)

            CALL INTFAC(DSEG, 256)
```

```
c Determine scaling factor if required

                IF(THETA .EQ. 0 .AND. RESP .EQ. 0) THEN
                        DO 70 W=0, 127
                                CALL PEEK(AMP, DSEG, W*2)
                                IF(AMP .GT. MAX) MAX = AMP
70                      CONTINUE

                        SF=NINT((REAL(MAX)/REAL(INC)*REAL(PA))/32768.0)
                        SFG=NINT(3.3219281*ALOG10(2.0*REAL(MAX)/150.0))
                ENDIF

c Clear screen and then write angle and display data

                CALL CLS(0)
                CALL GRAPH(DSEG, 0, 128, 50, 200, 3, 11, SFG, 50, 350)
                CALL OUTPUT(66, 2, 14, 0, 0, 'Theta = ', 1, THETA, 0, '°', 10)

c Divide data by the number of projections to prevent overflow

                DO 80 W=0, 127
                        CALL PEEK(AMP, DSEG, W*2)
                        AMPR=REAL(AMP)/REAL(SF)
                        CALL POKE(NINT(AMPR), DSEG, W*2)
80              CONTINUE

c Perform back projection

                IF (PA .EQ. 180) CALL BP128D(DSEG, 9088, THETA)
                IF (PA .EQ. 360) CALL BP128C(DSEG, 9088, THETA)

c Display images

                CALL CLS(0)
                CALL IMAGEA(9088, 0, 128, 128, 64, 11, 0, CSC)

                CALL DRLINE(128, 64, 384, 64, 7)
```

```
            CALL DRLINE(128, 320, 384, 320, 7)
            CALL DRLINE(128, 64, 128, 320, 7)
            CALL DRLINE(384, 64, 384, 320, 7)

            CALL COLORA(560, 112, 8, 6, CSC)
            CALL DRLINE(560, 112, 560, 368, 7)
            CALL DRLINE(608, 112, 608, 368, 7)
            CALL DRLINE(560, 112, 608, 112, 7)
            CALL DRLINE(560, 368, 608, 368, 7)

            CALL OUTPUT(67, 2, 14, 0, 0, 'Theta = ', 1, THETA, 0, '°', 10)
            CALL OUTPUT(68, 4, 14, 0, 0, ' SF = ', 1, SF, 8)

90    CONTINUE

****************************** Transfer the image to Macintosh ************************

95      WRITE (*, 100)
100     FORMAT( ' Transfer the image to Mac? (Yes=1 No=0)')
        READ (*, 110) RESP
110     FORMAT( I1)

        IF(RESP .EQ. 1) THEN
            CALL RDUMP(9088, 32)

            WRITE (*, 120)
120         FORMAT( ' Image at Mac OK? (Yes=1 No=0)')
            READ (*, 130) RESP1
130         FORMAT( I1)

            IF(RESP1 .EQ. 0) GOTO 95

        ENDIF

200   CALL CURSOR(2)

END
```

A2.4 Program in simulation of the square Helmholtz coil

```
100 '******************************** Gz - Z Gradient ********************************
110 '   Program to calculate the Z gradient of Gz planar coil for FX-60 'super-Gy' imaging probe   *
120 '************************************************************ 23/3/89 *********
130 '
140 DIM I(16),UI(25,16),WI(16,25),WJ(16,25),VJ(25,16),EI(25,16),CI(25,16),GRADY(6,6)
150 '
160 'Input data
170 '
180 PRINT " Planar coil to generate Z gradient: Gz "
190 INPUT " Choose a working plane [ (Y-Z)=1,(X-Y)=2,(Z-X)=3 ]: ",PLANE
200 INPUT " Input coil width (mm): ",CWidth
210 INPUT " Input coil height (mm): ",CHeight
250 '
260 ' Read in constants
270 '
280 M=6                  :'points to be calculated along one axis [=DIM of GRADY( , )]
290 FINAL1=.003          :'range to be calculated along one axis (m)
300 WT=.26               :'wire diameter in millimetre
310 NT=10                :'number of turns in one plane
320 MT=1                 :'number of planes
330 PI=3.1415927#        :'a well-known constant
340 X1=0:Y1=0:Z1=0
350 FINAL=FINAL1+.0000001#
360 STEPS=FINAL1/M
370 DJ=CWidth/2000
380 CH=CHeight/2
390 MU=4*PI*.0000001#
450 '
460 GOSUB 1700           :'Read in arrays
480 '
490 ' Main program
500 '
510 LPRINT "------- Planar coil to generate Z gradient: Gz -------"
520 LPRINT "(No. of turns =";NT;"*";MT;", Wire diameter ="; WT;"mm)"
530 LPRINT "(Coil width =";CWidth;"mm; Coil length =";CHeight;"mm)"
540 LPRINT
```

```
550 IF PLANE=1 THEN GOTO 600
560 IF PLANE=2 THEN GOTO 700
570 IF PLANE=3 THEN GOTO 800
590 '
600 LPRINT"(Y(mm),Z(mm))",:LPRINT"Gz (T/m/A)",: LPRINT"Field distortion":LPRINT
601 PRINT"(Y(mm),Z(mm))",:PRINT"Gz (T/m/A)",:PRINT"Field distortion":PRINT
602 FOR Y1=0 TO FINAL STEP STEPS
605   FOR Z1=0 TO FINAL STEP STEPS
610     GOSUB 1100
615     IF Y1=0 AND Z1=0 THEN GZ0=GZ
620     CORDA=INT(Y1*M/FINAL1+.5)
625     CORDB=INT(Z1*M/FINAL1+.5)
630     GOSUB 1500
635     N$="YZ"
640   NEXT Z1
645 NEXT Y1
650 GOTO 900
690 '
700 LPRINT"(X(mm),Y(mm))",:LPRINT"Gz (T/m/A)",: LPRINT"Field distortion":LPRINT
701 PRINT"(X(mm),Y(mm))",:PRINT"Gz (T/m/A)",:PRINT"Field distortion":PRINT
705 FOR X1=0 TO FINAL STEP STEPS
710   FOR Y1=0 TO FINAL STEP STEPS
720     GOSUB 1100
730     IF X1=0 AND Y1=0 THEN GZ0=GZ
740     CORDA=INT(X1*M/FINAL1+.5)
745     CORDB=INT(Y1*M/FINAL1+.5)
750     GOSUB 1500
755     N$="XY"
760   NEXT Y1
765 NEXT X1
770 GOTO 900
790 '
800 LPRINT"(Z(mm),X(mm))",:LPRINT"Gz (T/m/A)",: LPRINT"Field distortion":LPRINT
801 PRINT"(Z(mm),X(mm))",:PRINT"Gz (T/m/A)",:PRINT"Field distortion":PRINT
805 FOR Z1=0 TO FINAL STEP STEPS
810   FOR X1=0 TO FINAL STEP STEPS
815     GOSUB 1100
820     IF Z1=0 AND X1=0 THEN GZ0=GZ
```

```
825       CORDA=INT(Z1*M/FINAL1+.5)
830       CORDB=INT(X1*M/FINAL1+.5)
835       GOSUB 1500
840       N$="ZX"
845    NEXT X1
850 NEXT Z1
860 '
900 ' STOP
910 ' Save the data
920 '
930 A$="Z-Gz("+N$+").DAT"
940 OPEN A$ FOR OUTPUT AS#1
950 FOR CORDA=0 TO M
960    FOR CORDB=0 TO M
970       PRINT#1,GRAD(CORDA,CORDB)
980    NEXT CORDB
990 NEXT CORDA
1000 CLOSE #1
1010 END
1090 '
1091 ' Subroutine to calculate the gradient
1092 '
1100  GZ=0
1110  FOR Q=1 TO MT
1120     FOR P=1 TO NT
1120        FOR N=1 TO 8
1140 '
1150 ' i terms (parallel with Y axis)
1160 '
1170        AI2=(X1-UI(P,N))^2+(Z1-WI(N,Q))^2
1180        TERM1=(Y1-CI(P,N))/SQR(AI2+(Y1-CI(P,N))^2)
1190        TERM2=(Y1-EI(P,N))/SQR(AI2+(Y1-EI(P,N))^2)
1200        TERM3=TERM1-TERM2-TERM1^3/3+TERM2^3/3
1210        TERM4=3*I(N)*(X1-UI(P,N))*(Z1-WI(N,Q))/AI2^2
1220         GZ=GZ+TERM3*TERM4
1240 '
1250 ' j terms (parallel with X axis)
1260 '
```

```
1270          AJ2=(Y1-VJ(P,N))^2+(Z1-WJ(N,Q))^2
1280          TERM1=(X1-DJ)/SQR(AJ2+(X1-DJ)^2)
1290          TERM2=(X1+DJ)/SQR(AJ2+(X1+DJ)^2)
1300          TERM3=TERM1-TERM2-TERM1^3/3+TERM2^3/3
1310          TERM4=(-3)*I(N)*(Y1-VJ(P,N))*(Z1-WJ(N,Q))/AJ2^2
1320          GZ=GZ+TERM3*TERM4
1330        NEXT N
1340      NEXT P
1350 NEXT Q
1360 GZ=GZ*MU/(4*PI)
1400 RETURN
1500 '
1510 'Print subroutine
1520 '
1530 GRAD(CORDA,CORDB)=(GZ-GZ0)*100/GZ0
1540 PRINT "(";CORDA/2;",";CORDB/2;")",
1550 LPRINT "(";CORDA/2;",";CORDB/2;")",
1560 PRINT GZ,
1570 LPRINT GZ,
1580 PRINT USING "##.###";GRAD(CORDA,CORDB);
1590 LPRINT USING "##.###";GRAD(CORDA,CORDB);
1600 PRINT"%"
1610 LPRINT"%"
1650 RETURN
1670 '
1680 ' Read data for current and wire sign arrays
1690 '
1700 RESTORE 2030
1705 FOR N=1 TO 8
1710   READ A
1715   A=A*CWidth/2
1720   FOR P=1 TO NT
1725     UI(P,N)=(A+WT*(P-NT/2-.5))*.001
1730   NEXT P
1735 NEXT N
1740 RESTORE 2040
1745 FOR N=1 TO 4
1750   READ A
```

```
1754   A=A*CWidth/2
1755   FOR Q=1 TO MT
1760     WI(N,Q)=(A+WT*(Q-MT/2-.5))*.001
1765     WJ(N,Q)=WI(N,Q)
1770   NEXT Q
1775 NEXT N
1780 FOR N=5 TO 8
1790   READ A
1794   A=A*(32-CWidth/2)
1795   FOR Q=1 TO MT
1800     WI(N,Q)=(A+WT*(Q-MT/2-.5))*.001
1805     WJ(N,Q)=WI(N,Q)
1810   NEXT Q
1820 NEXT N
1830 RESTORE 2050
1840 FOR N=1 TO 8
1850   READ A
1854   A=A*CH
1855   FOR P=1 TO NT
1860     VJ(P,N)=(A+WT*(P-NT/2-.5))*.001
1865   NEXT P
1870 NEXT N
1880 RESTORE 2060
1890 FOR N=1 TO 8
1900   READ A
1904   A=A*CH
1905   FOR P=1 TO NT
1910     EI(P,N)=(A+WT*(P-NT/2-.5))*.001
1915   NEXT P
1920 NEXT N
1930 RESTORE 2070
1940 FOR N=1 TO 8
1950   READ A
1954   A=A*CH
1955   FOR P=1 TO NT
1960     CI(P,N)=(A+WT*(P-NT/2-.5))*.001
1965   NEXT P
1970 NEXT N
```

```
1980 RESTORE 2080
1990 FOR N=1 TO 8
2000 ` READ I(N)
2010 NEXT N
2020 RETURN
2024 '
2025 'Data arrays
2026 '
2030 DATA -1,-1,1,1,-1,-1,1,1
2040 DATA -1,1,-1,1,-1,1,-1,1
2050 DATA 1,1,-1,-1,1,1,-1,-1
2060 DATA -1,-1,-1,-1,-1,-1,-1,-1
2070 DATA 1,1,1,1,1,1,1,1
2080 DATA -1,1,1,-1,-0.6,0.6,0.6,-0.6
```

A3 Macintosh software: ImageShow™

The ImageShow™ software consists of a PASCAL program (ImageShow.p), a resource program (ImageShow.r) and a compile program (ImageShow.make).

A3.1 ImageShow.p

```
{[j=20}
{-------------------------------------------------------------------------}
PROGRAM ImageShow;    {version 4.0}
{-------------------------------------------------------------------------}

  USES
    Memtypes, Quickdraw, Osintf, Toolintf, Packintf, Palettemgr, Printtraps;

  CONST
    Maxwindows      = 40;       {maximum number of windows on the screen at one time}
    MaxQSlice       = 18;       {maximum number of q-slices used in dynamic analysis}
    MaxCount        = 1000;     {maximum number of different values in statistical routines}
    MaxFFTNo        = 1024;     {biggest digital array size in FFT}

    ndat2           = 131072;   {256x256x2}
    ndimp           = 2;        {2 dimensional in 2_D FT}

    GoodArray1      = 131072;   {256x256 array}
    GoodArray2      = 32768;    {128x128 array}
    GoodArray3      = 8192;     {64x64 array}

    Leftkey         = 123;      {key codes in Apple Extended key-board}
    Rightkey        = 124;
    Upkey           = 126;
    Downkey         = 125;

    Clut32          = 128;
    Clut16          = 129;
    Clut8           = 130;
    Clut4           = 131;
    Greyclut        = 132;
    Monoclut        = 133;
    Flow15clut      = 134;

    Plttid          = 128;

    Applemenu       = 128;
    Imagemenu       = 129;
    Editmenu        = 130;
    Scalemenu       = 131;
    Cursormenu      = 132;
    Displaymenu     = 133;
    Operationmenu   = 134;
    Multislicesmenu = 135;

  TYPE
    Bitmapptr   = ^Bitmap;      {used to coerce PixelMaps into BitMaps}
```

```
    Intptr          = ^Integer;          {to read the raw data}
    Image           = RECORD             {needed to handle the display of each image}
        Window          : Windowptr;     {the window for each Image}
        Wrecord         : Windowrecord;
        Colscalew       : Windowptr;     {window for the Scale}
        Colscalerec     : Windowrecord;
        Thecolours      : Palettehandle;
        Pixmap          : Pixmaphandle;  {PixelMap for the Image}
        Data            : Handle;        {raw Data}
        Imagesize       : Longint;       {size of image data array}
        ImageWidth      : Longint;       {width of image array, for example, 256}
        Showcscale      : Boolean;       {Show/Hide Scale}
        Cscaletype      : Boolean;       {static/dynamic colour scale}
        Baseline        : Integer;       {baseline = min(black)}
        Upper           : Integer;       {Upper = max(white)}
        Scale           : Integer;       {how many scale divisions}
        Dispdimension   : Integer;       {window dimension}
        Square          : Boolean;       {true = square window}
    END;

VAR
    Myhandle        : Array [Applemenu..Multislicesmenu] OF Menuhandle;
    Ourimages       : Array [0..Maxwindows] OF Image;
    Ourrect         : Array [0..Maxwindows] OF Rect;
    Myevent         : Eventrecord;
    Err             : Oserr;
    Dragrect        : Rect;
    Whichwindow     : Windowptr;     {used for handling events}
    Mycurwindow     : Windowptr;     {windows for cursor or text}
    Curwrecord      : Windowrecord;
    Mydialog        : Dialogptr;
    Texth           : Tehandle;
    Ibeamhdl        : Curshandle;

    Quit                        : Boolean; {true if quit}
    Cursmode                    : Boolean; {true in cursor mode}
    Scalemode                   : Boolean; {true in scale mode}
    Qmode                       : Boolean; {true if crosscursor select in flow analysis}
    CFmode                      : Boolean; {true if in Pixel Statistics menu}
    Hismode                     : Boolean; {true if in Histogram}
    STImage                     : Boolean; {true if in stack-plot mode}
    NoOpen                      : Boolean; {true if no image has been opened}
    Type2RData, Type2IData      : Boolean; {true if from flowanalysis}
    Firstdraw, Sorry, Over      : Boolean; {used in the statistical routines}

    NI, z, DFacter, FWHM, Vel, Scfac, Xshift, Yshift, Xspace    : Longint;
    Mean1, NMean1, Mean2, NMean2, FFTNo, CurrentArray           : Longint;
    Mymask, Images, MaxNum, X, Y, Scbase                        : Integer;
    Large, Small, C1, C2, TNum, ZNum, NTNum                     : Integer;
    KMax, A, B, A2, B2, Power                                   : Extended;

    QSpectrum                   : Array [0..MaxFFTNo] OF Longint;
    FinalC                      : Array [0..MaxCount] OF Longint;
    FinalV                      : Array [0..MaxCount] OF Integer;
    CX, CY                      : Array [1..MaxQSlice] OF Integer;
    KStep, Noise, NormSig       : Array [1..MaxQSlice] OF Extended;
    Data2                       : Array [1..ndat2] OF Real;

{-------------------------------------------------------------------------------}
{-------------------------------------------------------------------------------}

PROCEDURE Menuset;
```

```
    BEGIN
       Initmenus;
       Myhandle[Applemenu] := Getmenu(Applemenu);
       Addresmenu(Myhandle[Applemenu], 'DRVR');
       Insertmenu(Myhandle[Applemenu], 0);

       Myhandle[Imagemenu] := Getmenu(Imagemenu);
       Insertmenu(Myhandle[Imagemenu], 0);

       Myhandle[Editmenu] := Getmenu(Editmenu);
       Insertmenu(Myhandle[Editmenu], 0);

       Myhandle[Scalemenu] := Getmenu(Scalemenu);
       Insertmenu(Myhandle[Scalemenu], 0);

       Myhandle[Cursormenu] := Getmenu(Cursormenu);
       Insertmenu(Myhandle[Cursormenu], 0);

       Myhandle[Displaymenu] := Getmenu(Displaymenu);
       Insertmenu(Myhandle[Displaymenu], 0);

       Myhandle[Operationmenu] := Getmenu(Operationmenu);
       Insertmenu(Myhandle[Operationmenu], 0);

       Myhandle[Multislicesmenu] := Getmenu(Multislicesmenu);
       Insertmenu(Myhandle[Multislicesmenu], 0);

       Drawmenubar; {draw completed menubar}
    END;

{-------------------------------------------------------------------------------}
   {$S Part1}

   PROCEDURE Enablemenus;

      BEGIN
         IF (Images >= 0) THEN
            BEGIN
               Enableitem(Myhandle[Imagemenu], 3);
               Enableitem(Myhandle[Imagemenu], 4);
               Enableitem(Myhandle[Imagemenu], 5);
               Enableitem(Myhandle[Imagemenu], 7);
               Enableitem(Myhandle[Scalemenu], 0);
               Enableitem(Myhandle[Cursormenu], 0);
               Enableitem(Myhandle[Displaymenu], 0);
               Enableitem(Myhandle[Operationmenu], 0);
               Disableitem(Myhandle[Multislicesmenu], 1);
               Disableitem(Myhandle[Multislicesmenu], 2);
            END;
      END;

{-------------------------------------------------------------------------------}
   {$S Part1}
   { Highlght the default button. }

   PROCEDURE SetOKButton(aDialogPtr: DialogPtr);

      VAR
         itemtype  : Integer;
         item      : handle;
         box       : Rect;
         oldport   : Grafptr;
```

```
    BEGIN
      GetPort(oldPort);
      SetPort(aDialogptr);
      GetDItem(aDialogPtr, 1, itemType, item, box);
      Pensize(3, 3);
      InsetRect(box, -4, -4);
      FrameRoundRect(box, 16, 16);
      SetPort(oldPort);
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { Show dialogue box with the notice. }

  PROCEDURE Donotice(I: Integer);

    VAR
      Itemhit : Integer;

    BEGIN
      Mydialog := Getnewdialog(I, nil, Pointer( - 1));
      SetOKButton(MyDialog);

      REPEAT
        Modaldialog(nil, Itemhit);
      UNTIL (Itemhit = 1);

      Disposdialog(Mydialog);
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}

  PROCEDURE DrawMyCross(Mypencolour: Pattern; CXoffset, CYoffset, CFacter: Integer);

    VAR
      I          : Integer;

    BEGIN
      FOR I := 1 TO z DO
        BEGIN
          PenPat(Mypencolour);
          Moveto(CXoffset + CFacter*CX[I] - 3, CYoffset + CFacter*CY[I]);
          Lineto(CXoffset + CFacter*CX[I] + 3, CYoffset + CFacter*CY[I]);
          Moveto(CXoffset + CFacter*CX[I], CYoffset + CFacter*CY[I] - 3);
          Lineto(CXoffset + CFacter*CX[I], CYoffset + CFacter*CY[I] + 3);
        END;
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { Part of the Update procedure which updates a particular scale (which), redraws it with the
    appropriate size rectangles. }

  PROCEDURE Updatescale(Which: Integer);

    VAR
      R              : Rect;
      Factor, Count  : Integer;
      Thecolour      : Rgbcolor;

    BEGIN
      IF Ourimages[Which].Showcscale THEN
```

```
            BEGIN
               Setport(Ourimages[Which].Colscalew);
               Beginupdate(Ourimages[Which].Colscalew);
               Eraserect(Ourimages[Which].Colscalew^.Portrect);
               WITH Ourimages[Which].Colscalew^.Portrect DO
                  BEGIN
                     Factor := (Bottom - Top) DIV 32;
                     FOR Count := 0 TO 31 DO
                        BEGIN
                           Setrect(R, 0, ((Count) * Factor), Right, ((Count) * Factor) + Factor);
                           Getentrycolor(Ourimages[Which].Thecolours, Count, Thecolour);
                           Rgbforecolor(Thecolour);
                           Paintrect(R);
                        END;
                  END;
               Endupdate(Ourimages[Which].Colscalew);
            END;
      END;

{-------------------------------------------------------------------------------}
   {$S Part1}
   { Sets the menu for the appropriate scale. }

   PROCEDURE Setthescale;

      VAR
         Num          : Integer;

      BEGIN
         IF (Frontwindow <> nil) THEN
            BEGIN
               Num := Getwrefcon(Frontwindow);
               IF ((Num > - 1) AND (Num < Maxwindows)) THEN
                  BEGIN
                     IF Ourimages[Num].Showcscale = true THEN
                        Setitem(Myhandle[Scalemenu], 1, 'Hide Scale')
                     ELSE
                        Setitem(Myhandle[Scalemenu], 1, 'Show Scale');
                     IF Ourimages[Num].Square = True THEN
                        Setitem(Myhandle[Displaymenu], 6, 'Non-Square')
                     ELSE
                        Setitem(Myhandle[Displaymenu], 6, 'Square');
                  END;
            END;     {if frontwindow}
      END;

{-------------------------------------------------------------------------------}
   {$S Part1}
   { Creates a new menu window, gives it the appropriate pallette and sets the show flag to true i.e.
     Scale is at present displayed, sets menu to 'Hide Scale'. It does not draw the menu since the
     Update event generated will cause it to be displayed. }

   PROCEDURE Createcscale(Num: Integer);

      VAR
         Windowsize      : Rect;
         WinPInc         : Integer;

      BEGIN
         WinPInc := Num - 2*NI;
         IF (NOT Type2RData) AND (NOT Type2IData) AND (NOT STImage) THEN
            Setrect(Windowsize, (265 + (WinPInc * 10)), (45 + (WinPInc * 10)), (315 +
                                                   (WinPInc * 10)), (301 + (WinPInc * 10)));
```

```
        Ourimages[Num].Colscalew := Newcwindow(@Ourimages[Num].Colscalerec,
                                                Windowsize, 'S', True, 0, Pointer( - 1), False, 0);
        Setwrefcon(Ourimages[Num].Colscalew, Num);
        Ourimages[Num].Showcscale := True;
        Setitem(Myhandle[Scalemenu], 1, 'Hide Scale');
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { Closes the appropriate scale and sets the flag to false and the menu to 'Show Scale'. }

  PROCEDURE Closecscale(Num: Integer);

    BEGIN
        Closewindow(Ourimages[Num].Colscalew);
        Ourimages[Num].Showcscale := False;
        Setitem(Myhandle[Scalemenu], 1, 'Show Scale');
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { Calculate the default upper/lower display limits for the image. }

  PROCEDURE FindLimits(Num : Integer);

    VAR
        Place, Plate, Top, Bottom  : Longint;
        Myarray                    : Intptr;

    BEGIN
        IF (NOT Type2RData) AND (NOT Type2IData) THEN
            BEGIN
                Place := 0;
                Top := 1;
                Bottom := 0;

                REPEAT
                    Myarray := Intptr(Ord(Ourimages[Num].Data^) + Place);
                    Plate := Myarray^;

                    IF (Plate > Top) THEN Top := Plate
                        ELSE IF ( Plate < Bottom) THEN Bottom := Plate;

                    Place := Place + 2;
                UNTIL Place >= Ourimages[Num].Imagesize;

                IF (Round(1.5*Top) > 32767) THEN Ourimages[Num].Upper := 32767
                    ELSE Ourimages[Num].Upper := Round(1.5*Top);

                IF ((Round(100.0*Top/Abs(Bottom)) <120) AND (Top < 400)) OR
                            ((Round(100.0*Top/Abs(Bottom)) <120) AND (Top > 6000)) THEN
                    BEGIN
                        IF (Round(1.5*Bottom) < -32767) THEN
                                Ourimages[Num].Baseline := -32767
                            ELSE Ourimages[Num].Baseline := -Ourimages[Num].Upper;
                    END
                ELSE Ourimages[Num].Baseline := 0;
            END;
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { Makes the palettehandle 'ourpalette' equal to appropriate colours. }
```

```
PROCEDURE Changeourpalette(Num: Integer; Theclut: Integer);

   VAR
      Ourctable      : Ctabhandle;

   BEGIN
      Ourctable := Getctable(Theclut);
      WITH Ourimages[Num] DO
         BEGIN
            Disposctable(Pixmap^^.Pmtable);
            Pixmap^^.Pmtable := Ourctable;
            Setport(Window);
            Invalrect(Window^.Portrect);
            IF Showcscale THEN
               BEGIN
                  Setport(Colscalew);
                  Invalrect(Colscalew^.Portrect);
               END;
            Ctab2palette(Ourctable, Thecolours, Pmtolerant, 0);
         END;
   END;

{-----------------------------------------------------------------------}
   {$S Part1}

   PROCEDURE Setourpalette(Num: Integer);

      VAR
         Thepalette     : Palettehandle;
         Ourctable      : Ctabhandle;
         Myarray        : Intptr;
         Plate          : Longint;
         Temp           : Extended;

      BEGIN
         IF (NOT Type2RData) AND (NOT Type2IData) THEN
            BEGIN
               IF (Ourimages[Num].Baseline = 0) THEN
                  Ourctable := Getctable(Clut32)
               ELSE
                  BEGIN
                     Temp := 100.0*Ourimages[Num].Upper/Abs(Ourimages[Num].Baseline);
                     IF ((Round(Temp) > 80) AND (Round(Temp) < 120)) THEN
                        Ourctable := Getctable(Flow15clut)
                     ELSE
                        Ourctable := Getctable(Clut32);
                  END;
            END
         ELSE Ourctable := Getctable(Flow15clut);

         Thepalette := Getnewpalette(Plttid);

         WITH Ourimages[Num] DO
            BEGIN
               Thecolours := Newpalette(32, nil, Pmtolerant, 0);
               Disposctable(Pixmap^^.Pmtable);
               Pixmap^^.Pmtable := Ourctable;
               Setpalette(Window, Thepalette, True);
               Activatepalette(Window);
               Ctab2palette(Ourctable, Thecolours, Pmtolerant, 0);
               Invalrect(Window^.Portrect);
            END;
```

```
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { Main routine to display a color image. }

  PROCEDURE Newpiximage(Name: Str255; Size: Longint);

    VAR
      Windowrect  : Rect;
      Pixmapptr   : Ptr;
      Ourcursorh  : Curshandle;
      WinPInc     : Integer;

    BEGIN
      IF (Images < Maxwindows) THEN
        BEGIN
          Ourcursorh := Getcursor(4);
          Setcursor(Ourcursorh^^);

          Images := Images + 1;
          WinPInc := Images - 2*NI;
          Createcscale(Images);

          IF (NOT Type2RData) AND (NOT Type2IData) THEN
            Setrect(Windowrect, (5 + (WinPInc * 10)), (45 + (WinPInc * 10)), (261 +
                                        (WinPInc * 10)), (301 + (WinPInc * 10)))
          ELSE IF Type2RData THEN
            Setrect(Windowrect, (502 - (Images * 5)), (50 + (Images * 2)), (630 -
                                                (Images * 5)), (178 + (Images * 2)))
          ELSE IF Type2IData THEN
            Setrect(Windowrect, (507 - (Images * 5)), (260 + (Images * 2)), (635 -
                                                (Images * 5)), (388 + (Images * 2)));

          Ourimages[Images].Window := Newcwindow(@Ourimages[Images].Wrecord,
                                        Windowrect, Name, True, 0,Pointer( - 1), True, 0);

          Setport(Ourimages[Images].Window);
          Selectwindow(Ourimages[Images].Window);
          WITH Ourimages[Images] DO
            BEGIN
              Setwrefcon(Window, Images);
              Imagesize := Size;
              ImageWidth := Round(Sqrt(Imagesize DIV 2));
              Data := Newhandle(Size);          {initialing data into the fields}

              IF (NOT Type2RData) AND (NOT Type2IData) THEN
                BEGIN
                  Baseline := 0;
                  Upper := 32767;
                END
              ELSE
                BEGIN
                  Baseline := -32767;
                  Upper := 32767;
                END;
              Dispdimension := 256;
              Square := True;                   {start window as square}
              Scale := 31;                      {this is the number of bits in our scale}
              Setrect(Ourrect[Images], 0, 0, ImageWidth, ImageWidth);

              Pixmap := Newpixmap;
              Pixmapptr := Newptr(Size DIV 2);
```

```
                    WITH Pixmap^^ DO
                        BEGIN
                            Baseaddr := Pixmapptr;
                            Rowbytes := ImageWidth + $8000;
                                    {the $8000 tells the mac this is a colour window}
                            Bounds := Ourrect[Images];
                            Pixelsize := 8;
                        END;
                    Setourpalette(Images);
                END;
            END
        ELSE
            Donotice(134);
    END;

{-------------------------------------------------------------------------------}
    {$S Part1}
    { Scale the raw data pixels. }

    PROCEDURE Adjust(Num: Integer);

        VAR
            Place, Incr, Factor   : Longint;
            Thedata               : Integer;
            Theptr                : Intptr;
            Mybits                : Ptr;

        BEGIN
            Place := 0;
            Incr := 0;

            WITH Ourimages[Num] DO
                BEGIN
                    Factor := (Ord4(Upper) - Baseline);
                    REPEAT
                        Theptr := Intptr(Ord(Data^) + Place);
                        Thedata := Theptr^;
                        IF Thedata > Upper THEN Thedata := Upper;
                        IF Thedata < Baseline THEN Thedata := Baseline;
                        Mybits := Ptr(Ord(Pixmap^^.Baseaddr) + Incr);
                        Mybits^ := (((Ord4(Upper) - Thedata) * Ord4(Scale)) DIV Factor);
                        Incr := Incr + 1;
                        Place := Place + 2;
                    UNTIL Place >= Imagesize;
                END;
            Initcursor;
        END;

{-------------------------------------------------------------------------------}
    {$S Part1}
    { Save the image as the type "IMGE". }

    PROCEDURE Savearray;

        VAR
            Where                 : Point;
            Vrefnum, Refnum, Num  : Integer;
            Filecount             : Longint;
            Reply                 : Sfreply;
            Outstr1               : Str255;

        BEGIN
            Num := Getwrefcon(Frontwindow);
```

```
    Where.V := 100;
    Where.H := 180;
    Outstr1 := ' SaveAs ImageShow File: ';
    SFPUTFile(where, Outstr1, '', nil, reply);
    Filecount := Ourimages[Num].Imagesize;

    IF Reply.Good = True THEN
       BEGIN
          Err := Create(Reply.Fname, Reply.Vrefnum, 'MASY', 'IMGE');
          Err := Fsopen(Reply.Fname, Reply.Vrefnum, Refnum);
          Err := Fswrite(Refnum, Filecount, Ourimages[Num].Data^);
          Err := Fsclose(Refnum);
          Err := Flushvol(nil, Reply.Vrefnum);
       END;
 END;

{-------------------------------------------------------------------------------}
 {$S Part1}
 { Save cross sections of the image at the cursor position as a CricketGraph™ file. }

 PROCEDURE SaveAsCG(X, Y, Num: Integer; XShift, YShift: Longint);

    VAR
       Dim, Xscaled, YScaled, XPlace, YPlace                  : Longint;
       XAmp2, YAmp2, XAmp3, YAmp3, XAmp4, YAmp4               : Longint;
       XAmp5, YAmp5, XAmp6, YAmp6                             : Longint;
       Outstr, Outstr1, Outstr2, Outstr3, Outstr4, Outstr5    : Str255;
       Outstr6, Outstr7, Outstr8, Outstr9, Outstr10, Outstr11 : Str255;
       f             : text;
       Where         : point;
       reply         : SFReply;
       i             : Integer;
       Tab           : Char;
       Creat, Ctype  : OSType;
       Myfileinfo    : FInfo;
       Datptr        : Intptr;
       Ourcursorh    : Curshandle;

    BEGIN
       Dim := Ourimages[Num].ImageWidth;
       Xscaled := Round(Dim*X/(Ourimages[Num].Dispdimension)) + Xshift;
       Yscaled := Round(Dim*Y/(Ourimages[Num].Dispdimension)) + Yshift;
       Tab := char($09);
       Where.v := 50;
       Where.h := 200;
       Outstr1 := ' SaveAs Cricket Graph File: ';

       SFPUTFile(Where, Outstr1,'',nil, reply);
       IF (reply.good = true) THEN
          BEGIN
             Err := SetVol(nil,reply.vrefnum);
             Creat := OSType('CGRF');
             NumToString(Yscaled - 2, Outstr);
             Outstr2 := Concat('Amp(Y=',Outstr,')');
             NumToString(Yscaled - 1, Outstr);
             Outstr3 := Concat('Amp(Y=',Outstr,')');
             NumToString(Yscaled, Outstr);
             Outstr4 := Concat('Amp(Y=',Outstr,')');
             NumToString(Yscaled + 1, Outstr);
             Outstr5 := Concat('Amp(Y=',Outstr,')');
             NumToString(Yscaled + 2, Outstr);
             Outstr6 := Concat('Amp(Y=',Outstr,')');
```

```
                NumToString(Xscaled - 2, Outstr);
                Outstr7 := Concat('Amp(X=',Outstr,')');
                NumToString(Xscaled - 1, Outstr);
                Outstr8 := Concat('Amp(X=',Outstr,')');
                NumToString(Xscaled, Outstr);
                Outstr9 := Concat('Amp(X=',Outstr,')');
                NumToString(Xscaled + 1, Outstr);
                Outstr10 := Concat('Amp(X=',Outstr,')');
                NumToString(Xscaled + 2, Outstr);
                Outstr11 := Concat('Amp(X=',Outstr,')');

                Rewrite(f, reply.fName);
                WriteLn (f, '*');
                WriteLn (f, Outstr2, Tab, Outstr3, Tab, Outstr4, Tab, Outstr5, Tab, Outstr6, Tab,
                            Outstr7, Tab, Outstr8, Tab, Outstr9, Tab, Outstr10, Tab, Outstr11);

                XPlace := 2*Xscaled;
                YPlace := 2*(Dim * Yscaled);
                FOR i := 0 TO Dim-1 DO
                   BEGIN
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + YPlace - 4*Dim + 2*i);
                      XAmp2 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + YPlace - 2*Dim + 2*i);
                      XAmp3 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + YPlace + 2*i);
                      XAmp4 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + YPlace + 2*Dim + 2*i);
                      XAmp5 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + YPlace + 4*Dim + 2*i);
                      XAmp6 := Datptr^;

                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + XPlace - 4 + 2*i*Dim);
                      YAmp2 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + XPlace - 2 + 2*i*Dim);
                      YAmp3 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + XPlace + 2*i*Dim);
                      YAmp4 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + XPlace + 2 + 2*i*Dim);
                      YAmp5 := Datptr^;
                      Datptr := Intptr(Ord(Ourimages[Num].Data^) + XPlace + 4 + 2*i*Dim);
                      YAmp6 := Datptr^;
                      WriteLn (f, XAmp2, Tab, XAmp3, Tab, XAmp4, Tab, XAmp5, Tab,
                                  XAmp6, Tab, YAmp2, Tab, YAmp3, Tab, YAmp4, Tab,
                                  YAmp5, Tab, YAmp6);
                   END;

                Close (f);
                Err := GetFInfo(reply.fName, reply.vRefNum, Myfileinfo);
                Myfileinfo.fdcreator := Creat;
                Err := SetFInfo (reply.fName, reply.vRefNum, Myfileinfo);
             END;
          Ourcursorh := Getcursor(2);
          Setcursor(Ourcursorh^^);
       END;

{-------------------------------------------------------------------------------}
   {$S Part1}
   { The main initialization procedure called when a new Image is read in. It allocates the various fields
     in the Image Record and reads in the Raw Data. Also sets up the parameters for the Image to be
     displayed. }

   PROCEDURE Doarray;
```

```
VAR
   Logeof       : Longint;
   Refnum       : Integer;
   Typelist     : Sftypelist;
   Where        : Point;
   Reply        : Sfreply;
   Ourcursorh   : Curshandle;

BEGIN
   Where.V := 140;                    {position for standard file get procedure}
   Where.H := 150;
   Typelist[0] := 'IMGE';
   Sfgetfile(Where, '', nil, 1, Typelist, nil, Reply);
   IF Reply.Good = True THEN      {if it is a good file, proceed}
      BEGIN
         Ourcursorh := Getcursor(4);
         Setcursor(Ourcursorh^^);

         Err := Fsopen(Reply.Fname, Reply.Vrefnum, Refnum); {open the IMGE type of file}
         Err := Setfpos(Refnum, Fsfromstart, 0);
         Err := Geteof(Refnum, Logeof);                    {get the length of the file}
         IF (Logeof = GoodArray3) OR (Logeof = GoodArray2) OR (Logeof = GoodArray1)
               THEN
            BEGIN
               Newpiximage(Reply.Fname, Logeof);
               Err := Fsread(Refnum, Logeof, Ourimages[Images].Data^);
               Err := Fsclose(Refnum);

               FindLimits(Images);
               Setourpalette(Images);
               Adjust(Images);
               Initcursor;

               Enablemenus;
               IF (NI = 0) THEN  Disableitem(Myhandle[Displaymenu], 8);
               DrawMenubar;
            END
         ELSE
            BEGIN
               Initcursor;
               DoNotice(133);
            END;
      END
   ELSE NoOpen := True;
END;

{-------------------------------------------------------------------------------------------------}
{$S Part1}
{ This procedure is called when the toolbox generates an update event. It redraws each window but is
      clipped to regions which need updating. It then clears the update flag as well as calling the
      Updatescale procedure. }

PROCEDURE Updatewindow;

   VAR
      Count     : Integer;

   BEGIN
      Setthescale;

      FOR Count := 0 TO Images DO
         BEGIN
            WITH Ourimages[Count] DO
```

```
                BEGIN
                    Beginupdate(Window);
                    Copybits(Bitmapptr(Pixmap^)^, Window^.Portbits, Ourrect[Count],
                            Window^.Portrect, 0, Window^.Visrgn);
                    Endupdate(Window);
                    Updatescale(Count);
                END;
            END;
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This procedure is called when the toolbox generates an update event. It redraws only last window
      without clipping. It then clears the update flag as well as calling the Updatescale procedure. }

    PROCEDURE UpdateAllwindow;

        BEGIN
            Setthescale;

            WITH Ourimages[Images] DO
                BEGIN
                    Beginupdate(Window);
                    Copybits(Bitmapptr(Pixmap^)^, Window^.Portbits, Ourrect[Images],
                            Window^.Portrect, 0, nil);
                    Endupdate(Window);
                    Updatescale(Images);
                END;
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This routine is for the adjustment of the upper and lower limits of the current image. It sets the
        numbers in the edit boxes to the value stored in the image upper/baseline fields. }

    PROCEDURE Setcontrols(Num: Integer);

        VAR
            Thevalue, Itemtype  : Integer;
            Item                : Handle;
            Box                 : Rect;
            Sometext            : Str255;

        BEGIN
            Thevalue := Ourimages[Num].Baseline;
            Numtostring(Thevalue, Sometext);
            Getditem(Mydialog, 9, Itemtype, Item, Box);
            Setitext(Item, Sometext);
            Thevalue := Ourimages[Num].Upper;
            Numtostring(Thevalue, Sometext);
            Getditem(Mydialog, 10, Itemtype, Item, Box);
            Setitext(Item, Sometext);
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This routine handles the upper/lower threshold adjustment and calls procedure Setcontrols. }

    PROCEDURE Showcontrols;

        VAR
            Itemhit, Num, Theone, Itemtype  : Integer;
            Item                            : Handle;
```

```
    Box                         : Rect;
    Sometext                    : Str255;
    Thevalue                    : Longint;
    Ourcursorh                  : Curshandle;

BEGIN
  IF Frontwindow <> nil THEN
    BEGIN
      Num := Getwrefcon(Frontwindow);
      Mydialog := Getnewdialog(129, nil, Pointer( - 1));
      SetOKButton(MyDialog);
      Setcontrols(Num);

      REPEAT
        Modaldialog(nil, Itemhit);
        CASE Itemhit OF
          1:
            BEGIN
              Ourcursorh := Getcursor(4);
              Setcursor(Ourcursorh^^);

              Getditem(Mydialog, 9, Itemtype, Item, Box);
              Getitext(Item, Sometext);
              Stringtonum(Sometext, Thevalue);
              Ourimages[Num].Baseline := Thevalue;
              Getditem(Mydialog, 10, Itemtype, Item, Box);
              Getitext(Item, Sometext);
              Stringtonum(Sometext, Thevalue);
              Ourimages[Num].Upper := Thevalue;
              Adjust(Num);
              Setport(Ourimages[Num].Window);
              Invalrect(Ourimages[Num].Window^.Portrect);
              Updatewindow;
              Initcursor;
            END;
          3:
            BEGIN
              IF Ourimages[Num].Baseline > 31767 THEN
                                    Ourimages[Num].Baseline := 32767
              ELSE
                Ourimages[Num].Baseline := Ourimages[Num].Baseline + 1000;
              Setcontrols(Num);
            END;
          4:
            BEGIN
              IF Ourimages[Num].Upper > 31767 THEN
                                    Ourimages[Num].Upper := 32767
              ELSE
                Ourimages[Num].Upper := Ourimages[Num].Upper + 1000;
              Setcontrols(Num);
            END;
          5:
            BEGIN
              IF Ourimages[Num].Baseline < - 31766 THEN
                                    Ourimages[Num].Baseline := - 32766
              ELSE
                Ourimages[Num].Baseline := Ourimages[Num].Baseline - 1000;
              Setcontrols(Num);
            END;
          6:
            BEGIN
              IF Ourimages[Num].Upper < - 31766 THEN
                                    Ourimages[Num].Upper := - 32766
```

```
                        ELSE
                            Ourimages[Num].Upper := Ourimages[Num].Upper - 1000;
                        Setcontrols(Num);
                    END;
                END;
            UNTIL (Itemhit = 2);
            Closedialog(Mydialog);
        END;
    END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { Set up a window record for display. }

    PROCEDURE Coordcursor(Num: Integer);

        VAR
            Ourcursorh    : Curshandle;
            Windowsize    : Rect;

        BEGIN
            Ourcursorh := Getcursor(2);
            Setcursor(Ourcursorh^^);
            IF CursMode THEN
                Setrect(Windowsize, 10, 380, 630, 470)
            ELSE IF QMode THEN
                Setrect(Windowsize, 330, 30, 630, 470);
            Mycurwindow := Newcwindow(@Curwrecord, Windowsize, 'Cursor', True,
                                                    Dboxproc, Pointer( - 1), True, 0);
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { Plot cross section of the image. }

    PROCEDURE Plotslice(Num,Amplitude, Scbase: Integer;
                                        Scfac, Xscaled, Yscaled, Xspace: Longint);

        VAR
            Ampl, Place, Offset, Scaledampl   : Longint;
            I, I1, Space, Dim                 : Integer;
            Datptr                            : Intptr;
            Slicerect                         : Rect;

        BEGIN
            Setrect(Slicerect, 185, 0, 450, 90);
            Eraserect(Slicerect);
            Textmode(Srccopy);          {now plot slice, first erase old}
            FOR I := 0 TO 100 DO
                BEGIN
                    Moveto(190, I);
                    Drawstring('
                END;
            Pennormal;
            Framerect(Slicerect);

            Dim := Ourimages[Num].ImageWidth;
            IF (Ourimages[Num].Imagesize = GoodArray1) THEN     Space := 1
                ELSE IF (Ourimages[Num].Imagesize = GoodArray2) THEN    Space := 2
                ELSE IF (Ourimages[Num].Imagesize = GoodArray3) THEN    Space := 4;

            IF (Xspace = 1) THEN
                BEGIN
```

```
        Place := 2 * (Dim * Yscaled);
        FOR I := Xscaled DOWNTO 0 DO
          BEGIN
            Offset := 2 * I;
            Datptr := Intptr(Ord(Ourimages[Num].Data^) + Place + Offset);
            Ampl := Datptr^;
            Scaledampl := (Ampl * Scfac) DIV (32767);
            Moveto(190 + Space*I*Xspace, Scbase - Scaledampl);
            Line(0, 0);
          END;
        FOR I := Xscaled TO Dim-1 DO
          BEGIN
            Offset := 2*I;
            Datptr := Intptr(Ord(Ourimages[Num].Data^) + Place + Offset);
            Ampl := Datptr^;
            Scaledampl := (Ampl * Scfac) DIV (32767);
            Moveto(190 + Space*I*Xspace, Scbase - Scaledampl);
            Line(0, 0);
          END;
        Pensize(3, 3);
        Forecolor(205);
        Ampl := Amplitude;                          {convert int to Longint}
        Scaledampl := (Scfac*Ampl) DIV (32767);
        Moveto(190 + Space*Xscaled, Scbase - Scaledampl);
        Line(0, 0);
        Forecolor(33);
      END
    ELSE
      BEGIN
        Place := 2*(Dim*Yscaled);
        I1 := 1;
        FOR I := Xscaled DOWNTO 0 DO
          BEGIN
            Offset := 2*I;
            Datptr := Intptr(Ord(Ourimages[Num].Data^) + Place + Offset);
            Ampl := Datptr^;
            Scaledampl := (Ampl*Scfac) DIV (32767);
            IF (328 - Space*I1*Xspace >= 190) THEN
              BEGIN
                Moveto(328 - Space*I1*Xspace, Scbase - Scaledampl);
                Line(0, 0);
              END;
            I1 := I1 + 1;
          END;
        I1 := 1;
        FOR I := Xscaled TO Dim-1 DO
          BEGIN
            Offset := 2*I;
            Datptr := Intptr(Ord(Ourimages[Num].Data^) + Place + Offset);
            Ampl := Datptr^;
            Scaledampl := (Ampl*Scfac) DIV (32767);
            IF (328 + Space*I1*Xspace <= 446) THEN
              BEGIN
                Moveto(328 + Space*I1*Xspace, Scbase - Scaledampl);
                Line(0, 0);
              END;
            I1 := I1 + 1;
          END;
        Pensize(3, 3);
        Forecolor(205);
        Ampl := Amplitude;                          {convert int to Longint}
        Scaledampl := (Scfac*Ampl) DIV (32767);
        Moveto(328, Scbase - Scaledampl);           {now highlight chosen point}
```

```
            Line(0, 0);
            Forecolor(33);
        END;
    END;

{-------------------------------------------------------------------------}
  {$S Part1}
  { Calculate the cursor position and write them on the screen. }

  PROCEDURE Dispcoord(Num, Scbase1: Integer; X, Y, Scfac1, Xshift, Yshift: Longint);

    VAR
      Xscaled, Yscaled, Place, Offset, Ampl, Xspace  : Longint;
      Amplitude, I, Itemhit                          : Integer;
      Outstr1, Outstr2, Outstr3                      : Str255;
      Mypenstate                                     : Penstate;
      Datptr                                         : Intptr;

    BEGIN
      Setport(Mycurwindow);
      Textfont(21);
      Textface([Bold]);
      Moveto(5, 10);
      Drawstring('Cursor Parameters');

      Pennormal;
      Textmode(Srccopy);
      Moveto(10, 35);
      Drawstring('                    ');
      Moveto(10, 55);
      Drawstring('                    ');
      Moveto(10, 75);
      Drawstring('                    ');

      Xscaled := Round(Ourimages[Num].ImageWidth*X/(Ourimages[Num].Dispdimension)) +
                                                                        Xshift;
      Yscaled := Round(Ourimages[Num].ImageWidth*Y/(Ourimages[Num].Dispdimension)) +
                                                                        Yshift;
      Place := 2*(Ourimages[Num].ImageWidth*Yscaled + Xscaled);

      Datptr := Intptr(Ord(Ourimages[Num].Data^) + Place);
      Amplitude := Datptr^;

      Moveto(10, 35);
      Textfont(1);
      Textface([]);
      Textmode(Srcor);
      Xspace := 1;

      IF ((Ourimages[Num].Imagesize = GoodArray1) AND (Xscaled >= 0) AND
                    (Xscaled <= 255) AND (Yscaled >= 0) AND (Yscaled <= 255)) OR
         ((Ourimages[Num].Imagesize = GoodArray2) AND (Xscaled >= 0) AND
                    (Xscaled <= 127) AND (Yscaled >= 0) AND (Yscaled <= 127)) OR
         ((Ourimages[Num].Imagesize = GoodArray3) AND (Xscaled >= 0) AND
                    (Xscaled <= 63) AND (Yscaled >= 0) AND (Yscaled <= 63)) THEN
        BEGIN
          Numtostring(Xscaled, Outstr1);
          Numtostring(Yscaled, Outstr2);
          Numtostring(Amplitude, Outstr3);
          Drawstring(' ( x , y ) =  (');
          Drawstring(Outstr1);
          Drawstring(', ');
          Drawstring(Outstr2);
```

```
        Drawstring(')');
        Moveto(10, 55);
        Drawstring('Amplitude = ');
        Drawstring(Outstr3);

        IF Scalemode THEN
          BEGIN
            Mydialog := Getnewdialog(132, nil, Pointer( - 1));

            REPEAT
              Modaldialog(nil, Itemhit);
              CASE Itemhit OF
              1:
                BEGIN
                  Scbase1 := Scbase1 - 5;
                  Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled,
                                                                    Xspace);
                END;
              2:
                BEGIN
                  Scfac1 := Scfac1*2;
                  Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled,
                                                                    Xspace);
                END;
              5:
                BEGIN
                  Scfac1 := (Scfac1) DIV (2);
                  Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled,
                                                                    Xspace);
                END;
              6:
                BEGIN
                  Scbase1 := Scbase1 + 5;
                  Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled,
                                                                    Xspace);
                END;
              7:
                BEGIN
                  Xspace := Xspace + 1;
                  Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled,
                                                                    Xspace);
                END;
              8:
                BEGIN
                  Xspace := Xspace - 1;
                  IF (Xspace < 1) THEN Xspace := 1;
                  Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled,
                                                                    Xspace);
                END;
              END;
            UNTIL (Itemhit = 10);

            Scalemode := False;
            Closedialog(Mydialog);

            Scfac := Scfac1;
            Scbase := Scbase1;
          END
        ELSE
          Plotslice(Num, Amplitude, Scbase1, Scfac1, Xscaled, Yscaled, Xspace);
      END
    ELSE
      BEGIN
```

```
            Drawstring('Move towards image!');
            Xshift := 0;
            Yshift := 0;
        END;
    END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { Change the menu for the cursor function. }

    PROCEDURE Setcurmenu;

        VAR
            Num   : Integer;

        BEGIN
            IF (Frontwindow <> nil) THEN
                BEGIN
                    Num := Getwrefcon(Frontwindow);
                    IF Cursmode THEN
                        Setitem(Myhandle[Cursormenu], 1, 'Hide Cursor')
                    ELSE
                        Setitem(Myhandle[Cursormenu], 1, 'Show Cursor');
                END;
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This is the routine which is able to modify the raw image pixel value. The modified image will
        not be saved automatically into disk. A call of Save As" is needed. }

    PROCEDURE DoRetouch;

        VAR
            i, j, newdata, Thevalue, place   : Longint;
            Itemhit, Itemtype, Num           : Integer;
            Sometext                         : Str255;
            Item                             : Handle;
            Ourcursorh                       : Curshandle;
            myarray1                         : IntPtr;
            Box                              : Rect;

        BEGIN
            Num := Getwrefcon(Frontwindow);
            Mydialog := Getnewdialog(143, nil, Pointer( - 1));
            SetOKButton(MyDialog);

            REPEAT
                ModalDialog(nil, Itemhit);
                CASE Itemhit OF
                    1:
                        BEGIN
                            Getditem(Mydialog, 4, Itemtype, Item, Box);
                            Getitext(Item, Sometext);
                            Stringtonum(Sometext, Thevalue);
                            i := Thevalue;

                            Getditem(Mydialog, 6, Itemtype, Item, Box);
                            Getitext(Item, Sometext);
                            Stringtonum(Sometext, Thevalue);
                            j := Thevalue;

                            Place := 2*(Ourimages[Num].ImageWidth*j + i);
```

```
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                Thevalue:=Myarray1^;
                NumToString(Thevalue,Sometext);
                Getditem(Mydialog, 11, Itemtype, Item, Box);
                Setitext(Item, Sometext);
            END;    {end of case 1}
        12:
            BEGIN
                Getditem(Mydialog, 4, Itemtype, Item, Box);
                Getitext(Item, Sometext);
                Stringtonum(Sometext, Thevalue);
                i := Thevalue;

                Getditem(Mydialog, 6, Itemtype, Item, Box);
                Getitext(Item, Sometext);
                Stringtonum(Sometext, Thevalue);
                j := Thevalue;

                Getditem(Mydialog, 8, Itemtype, Item, Box);
                Getitext(Item, Sometext);
                Stringtonum(Sometext, Thevalue);
                newdata := Thevalue;

                Ourcursorh := Getcursor(4);
                Setcursor(Ourcursorh^^);

                Place := 2*(Ourimages[Num].ImageWidth*j + i);

                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                Myarray1^:= newdata;
                FindLimits(Num);
                Setourpalette(Num);
                Adjust(Num);
                UpdateAllWindow;
                Updatewindow;

                Initcursor;
            END;    {end of case 12}
        END;
    UNTIL (Itemhit = 2);

    Disposdialog(Mydialog);
END;

{-------------------------------------------------------------------------------}
{$S Part1}
{ This is the routine which draws the stacked plot on the screen. }

PROCEDURE Stackplot(PlotFrom: Integer; imfact: Longint; Squared: Boolean);

    VAR
        x0, y0, sx, sy, sx0, sy0, syp, x2, y2, i0, i1, i2, dat, dat1, dat2    : Longint;
        Zmaxp, x, y, z, i, j, gain, place, jmin, jmax, jmid, imin, imax    : Longint;
        Zmax, Zmin                  : Array [0..500] OF Longint;
        arr, Dim, HDim, Space, Num  : Integer;
        myarray1                    : Intptr;
        OutStr1                     : Str255;
        Ourcursorh                  : Curshandle;

    BEGIN
        Ourcursorh := Getcursor(4);
        Setcursor(Ourcursorh^^);
        Num := Getwrefcon(Frontwindow);
```

```
GetWTitle(Frontwindow, Outstr1);

IF (PlotFrom = -1) THEN     Outstr1:=Concat(Outstr1,'.F')
   ELSE IF (PlotFrom = 1) THEN     Outstr1:=Concat(Outstr1, '.B')
   ELSE IF (PlotFrom = -2) THEN    Outstr1:=Concat(Outstr1, '.L')
   ELSE IF (PlotFrom = 2) THEN     Outstr1:=Concat(Outstr1, '.R');
IF Squared THEN  Outstr1:=Concat(Outstr1, '(Squ.)');
Newpiximage(Outstr1, 131072);

FOR arr:=0 TO 500 DO
   BEGIN
      Zmax[arr] := 500;
      Zmin[arr] := -500;
   END;
Dim := Ourimages[Num].ImageWidth;
HDim := Dim DIV 2;
IF (Dim = 64) THEN  Space := 1
   ELSE IF (Dim = 128) THEN    Space := 2
   ELSE IF (Dim = 256) THEN    Space := 4;

gain := 32767 DIV imfact;
jmin := -31;
jmax := 31;
imin := -127;
imax := 127;

IF (Dim = 256) THEN
   BEGIN
      x0 := 80;
      y0 := 200;
      jmid := 50;
   END
ELSE IF (Dim = 128) THEN
   BEGIN
      x0 := 92;
      y0 := 189;
      jmid := 37;
   END
ELSE IF (Dim = 64) THEN
   BEGIN
      x0 := 104;
      y0 := 175;
      jmid := 25;
   END;

sx := 0;
Zmaxp := 0;

FOR j:=jmin TO jmax DO
BEGIN
   y := 2*j;
   sx0 := (x0 + imin) + (y + jmid);
   IF (sx<0) THEN sx := 0;
   sy0 := (y0 - (y + jmid));
   moveto(sx0, sy0);
   syp := sy0;

   FOR i := imin TO imax DO
   BEGIN
      x := i;
      i0 := i DIV 4;
      IF (i0>0) THEN i0 := i0 ELSE i0 := i0 - 1;
      i1 := i - 4*i0;
```

```
i2 := (Space*i) DIV 4;

IF (PlotFrom = -1) OR (PlotFrom = 1) THEN
   place := 2*(Dim*(PlotFrom*Space*j + HDim) + i2 + HDim)
ELSE IF (PlotFrom = -2) THEN
   place := 2*(Space*j + HDim + Dim*(i2 + HDim))
ELSE IF (PlotFrom = 2) THEN
   place := Ourimages[Num].Imagesize - 2*(Space*j - HDim + Dim*(i2 + HDim));

myarray1 := Intptr(Ord(Ourimages[Num].Data^) + place);

IF (Ourimages[Num].ImageSize =GoodArray1) OR
                        (Ourimages[Num].ImageSize =GoodArray2) THEN
   BEGIN
      IF NOT Squared THEN
         dat := (myarray1^*gain) DIV 300
      ELSE
         dat := ((myarray1^)*(myarray1^)*gain) DIV 300;
   END
ELSE   IF (Ourimages[Num].ImageSize = GoodArray3) THEN
   BEGIN
      IF NOT Squared THEN
         dat1 := (myarray1^*gain) DIV 300
      ELSE
         dat1 := ((myarray1^)*(myarray1^)*gain) DIV 300;

      IF (i0<31) THEN
         BEGIN
            IF (PlotFrom = -1) OR (PlotFrom = 1) THEN
               place := 2*(Dim*(PlotFrom*j + HDim) + i0 + 1 + HDim)
            ELSE IF (PlotFrom = -2) THEN
               place := 2*(j + HDim + Dim*(i0 + 1 + HDim))
            ELSE IF (PlotFrom = 2) THEN
               place := 2*(4096 - j + HDim - Dim*(i0 + 1 + HDim));

            IF NOT Squared THEN
               dat2 := (myarray1^*gain) DIV 300
            ELSE
               dat2 := ((myarray1^)*(myarray1^)*gain) DIV 300;
         END
      ELSE
         dat2 := 0;
         dat := ((4 - i1)*dat1 + i1*dat2) DIV 4;
   END;

z := y0 - (y + jmid) - dat;
sx := (x0 + x) + (y + jmid);
IF (sx < 0) THEN sx := 0;
arr := sx;
sy := z;
IF i = imin THEN
   BEGIN
      IF (Zmax[sx] <s y) THEN moveto(sx, Zmax[sx]) ELSE moveto(sx, sy);
   END;
IF (j=jmin) THEN
   BEGIN
      Zmax[arr] := z;
      lineto(sx, sy);
   END
ELSE
   BEGIN
      IF (sy <= Zmax[arr]) AND (syp <= Zmaxp) THEN
         BEGIN
```

```
                    lineto(sx, sy);
                    Zmax[arr] := z;
                END;
            IF (syp <= Zmaxp) AND (sy > Zmax[arr]) AND (sy > syp)
                                                THEN lineto(sx, Zmax[arr]);
            IF (syp > Zmaxp) AND (sy <= Zmax[arr]) THEN
                BEGIN
                    IF (Zmax[arr] = 500) THEN moveto(sx, sy)
                                         ELSE moveto(sx, Zmax[arr]);
                    lineto(sx, sy);
                END;
            IF (syp > Zmaxp) AND (sy > Zmax[arr]) THEN moveto(sx,sy);
            syp :=s y;
            Zmaxp := Zmax[arr];
        END;
    END; {of i}
END; {of j}

WITH Ourimages[Images] DO
    Copybits(Window^.Portbits, Bitmapptr(Pixmap^)^, Window^.Portrect,
                ourRect[Images], 0, Window^.Visrgn);
STImage := False;
Initcursor;
END;

{------------------------------------------------------------------------}
{$S Part1}
{ This is the stack-plot calling routine which interfaces with the user. }

PROCEDURE DoStackPlot;

VAR
    Itemhit, Itemtype, PlotFrom   : Integer;
    Thevalue, imfact              : Longint;
    Sometext                      : Str255;
    Box                           : Rect;
    Item                          : Handle;
    Squared                       : Boolean;

BEGIN
    PlotFrom := -1;
    Squared := false;
    Mydialog := Getnewdialog(142, nil, Pointer( - 1));
    SetOKButton(MyDialog);

    GetDItem(mydialog, 6, ItemType, Item, Box);
    SetCtlValue(ControlHandle(Item), 1);
    GetDItem(mydialog, 9, ItemType, Item, Box);
    SetCtlValue(ControlHandle(Item), 1);

    REPEAT
        ModalDialog(nil, Itemhit);

        CASE Itemhit OF
            9,10,11,12:
                BEGIN
                    CASE PlotFrom OF
                        -1:
                            BEGIN
                                GetDItem(mydialog, 9, ItemType, Item, Box);
                                SetCtlValue(ControlHandle(Item), 0);
                            END;
                        1:
```

```
                BEGIN
                    GetDItem(mydialog, 10, ItemType, Item, Box);
                    SetCtlValue(ControlHandle(Item), 0);
                END;
            -2:
                BEGIN
                    GetDItem(mydialog, 11, ItemType, Item, Box);
                    SetCtlValue(ControlHandle(Item), 0);
                END;
            2:
                BEGIN
                    GetDItem(mydialog, 12, ItemType, Item, Box);
                    SetCtlValue(ControlHandle(Item), 0);
                END;
        END;

        GetDItem(mydialog, Itemhit, ItemType, Item, Box);
        SetCtlValue(ControlHandle(Item), 1);
    END;

  6:
    BEGIN
        GetDItem(mydialog, 6, ItemType, Item, Box);
        SetCtlValue(ControlHandle(Item), 1);
        GetDItem(mydialog, 7, ItemType, Item, Box);
        SetCtlValue(ControlHandle(Item), 0);
        Squared := false;
    END;

  7:
    BEGIN
        GetDItem(mydialog, 7, ItemType, Item, Box);
        SetCtlValue(ControlHandle(Item), 1);
        GetDItem(mydialog, 6, ItemType, Item, Box);
        SetCtlValue(ControlHandle(Item), 0);
        Squared := true;
    END;
  END;

  CASE Itemhit OF
    6: Squared := false;
    7: Squared := true;
    9: PlotFrom := -1;
    10: PlotFrom := 1;
    11: PlotFrom := -2;
    12: PlotFrom := 2;
  END;
UNTIL (Itemhit=1) or (Itemhit=2);

CASE Itemhit OF
  1:
    BEGIN
      Getditem(Mydialog, 4, Itemtype, Item, Box);
      Getitext(Item, Sometext);
      Stringtonum(Sometext, Thevalue);
      imfact := Thevalue;

      Disposdialog(Mydialog);
      STImage := True;
      Stackplot(PlotFrom, imfact, Squared);
    END;     {end of case 1}
  2: Disposdialog(Mydialog);
END;
```

```
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine doing the convolution on the image. }

  PROCEDURE Conv(Upperleft, Upper, Upperright, Left, Centre, Right, Lowerleft,
                                                        Lower, Lowerright: Longint);

    VAR
      Jacross, Idown, Scaler, Across2, Down2, Nsmall, Num    : Integer;
      I, J, M, N, Place, Place1, Dval, Dval1, Val            : Longint;
      Myarray1, Myarray2                                     : Intptr;
      Ourcursorh                                             : Curshandle;

    BEGIN
      Num := Getwrefcon(Frontwindow);
      Newpiximage('Convolved Image', Ourimages[Num].Imagesize);
      Ourcursorh := Getcursor(4);
      Setcursor(Ourcursorh^^);

      Jacross := Ourimages[Num].ImageWidth-1;
      Idown := Ourimages[Num].ImageWidth-1;
      Scaler := Ourimages[Num].ImageWidth;
      Across2 := 2*Ourimages[Num].ImageWidth-2;
      Down2 := 2*Ourimages[Num].ImageWidth-2;
      Nsmall := 0;

      FOR J := 0 TO Jacross DO
        BEGIN
          FOR I := 0 TO Idown DO
            BEGIN
              Val := 0;
              M := I - 1;
              IF M < 0 THEN M := - M;
              N := J - 1;
              IF N < Nsmall THEN N := - N;
              Place1 := 2 * (Scaler * N + M);
              Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
              Dval := Myarray1^;
              Dval1 := (Dval * Upperleft) DIV (100);
              Val := Val + Dval1;

              M := I - 1;
              IF M < 0 THEN M := - M;
              N := J;
              Place1 := 2 * (Scaler * N + M);
              Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
              Dval := Myarray1^;
              Dval1 := (Dval * Left) DIV (100);
              Val := Val + Dval1;

              M := I - 1;
              IF M < 0 THEN M := - M;
              N := J + 1;
              IF N > Jacross THEN N := Across2 - N;
              Place1 := 2 * (Scaler * N + M);
              Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
              Dval := Myarray1^;
              Dval1 := (Dval * Lowerleft) DIV (100);
              Val := Val + Dval1;

              M := I;
```

```
                N := J - 1;
                IF N < Nsmall THEN N := - N;
                Place1 := 2 * (Scaler * N + M);
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                Dval := Myarray1^;
                Dval1 := (Dval * Upper) DIV (100);
                Val := Val + Dval1;

                M := I;
                N := J;
                Place1 := 2 * (Scaler * N + M);
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                Dval := Myarray1^;
                Dval1 := (Dval * Centre) DIV (100);
                Val := Val + Dval1;

                M := I;
                N := J + 1;
                IF N > Jacross THEN N := Across2 - N;
                Place1 := 2*(Scaler * N + M);
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                Dval := Myarray1^;
                Dval1 := (Dval * Lower) DIV (100);
                Val := Val + Dval1;

                M := I + 1;
                IF M > Idown THEN M := Down2 - M;
                N := J - 1;
                IF N < Nsmall THEN N := - N;
                Place1 := 2 * (Scaler * N + M);
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                Dval := Myarray1^;
                Dval1 := (Dval * Upperright) DIV (100);
                Val := Val + Dval1;

                M := I + 1;
                IF M > Idown THEN M := Down2 - M;
                N := J;
                Place1 := 2*(Scaler * N + M);
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                Dval := Myarray1^;
                Dval1 := (Dval * Right) DIV (100);
                Val := Val + Dval1;

                M := I + 1;
                IF M > Idown THEN M := Down2 - M;
                N := J + 1;
                IF N > Jacross THEN N := Across2 - N;
                Place1 := 2*(Scaler * N + M);
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                Dval := Myarray1^;
                Dval1 := (Dval * Lowerright) DIV (100);
                Val := Val + Dval1;

                Place := 2*(Scaler * J + I);
                Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                Myarray2^ := Val;
            END;
        END;
    FindLimits(Images);
    Setourpalette(Images);
    Adjust(Images);
    Initcursor;
```

```
    END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the convolution calling routine. }

  PROCEDURE Doconvol;

    VAR
      Upperleft, Upper, Left, Centre, Right        : Longint;
      Lowerleft, Lower, Lowerright, Upperright     : Longint;
      Itemhit, Itemtype                            : Integer;
      Box                                          : Rect;
      Item                                         : Handle;
      Sometext                                     : Str255;

    BEGIN
      Mydialog := Getnewdialog(138, nil, Pointer( - 1));
      SetOKButton(MyDialog);

      Modaldialog(nil, Itemhit);
      CASE Itemhit OF
        1:
          BEGIN
            Getditem(Mydialog, 3, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Left);

            Getditem(Mydialog, 4, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Lowerleft);

            Getditem(Mydialog, 5, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Upper);

            Getditem(Mydialog, 6, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Centre);

            Getditem(Mydialog, 7, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Lower);

            Getditem(Mydialog, 8, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Upperright);

            Getditem(Mydialog, 9, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Right);

            Getditem(Mydialog, 10, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Lowerright);

            Getditem(Mydialog, 11, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Upperleft);

            Disposdialog(Mydialog);

            Conv(Upperleft, Upper, Upperright, Left, Centre, Right,
```

```
                                                    Lowerleft, Lower, Lowerright);
            END;
          2: Disposdialog(Mydialog);
        END;
      END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine doing the edge detection. }

  PROCEDURE Edge(Step, Range: Longint);

    VAR
      I, J, M, N, Val, Place, Place1, Dval, Dval1  : Longint;
      Myarray1, Myarray2, Myarray3                 : Intptr;
      Num                                          : Integer;
      Ourcursorh                                   : Curshandle;

    BEGIN
      Ourcursorh := Getcursor(4);
      Setcursor(Ourcursorh^^);

      Num := Getwrefcon(Frontwindow);
      Newpiximage('Edged Image', Ourimages[Num].Imagesize);

      FOR J := 0 TO Ourimages[Num].ImageWidth - 1 DO
        BEGIN
          FOR I := 0 TO Ourimages[Num].ImageWidth - 1 DO
            BEGIN
              Val := 0;
              Place := 2*(Ourimages[Num].ImageWidth * J + I);
              Myarray2 := Intptr(Ord(Ourimages[Num].Data^) + Place);
              Dval := Myarray2^;

              M := I - Range;
              IF M < 0 THEN M := 0;
              N := J - Range;
              IF N < 0 THEN N := 0;
              Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
              Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
              Dval1 := Myarray1^;
              IF (Dval - Dval1) > Step THEN Val := 32767;

              M := I - Range;
              IF M < 0 THEN M := 0;
              N := J;
              Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
              Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
              Dval1 := Myarray1^;
              IF (Dval - Dval1) > Step THEN Val := 32767;

              M := I - Range;
              IF M < 0 THEN M := 0;
              N := J + Range;
              IF N > Ourimages[Num].ImageWidth - 1 THEN
                                              N := Ourimages[Num].ImageWidth - 1;
              Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
              Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
              Dval1 := Myarray1^;
              IF (Dval - Dval1) > Step THEN Val := 32767;

              M := I;
              N := J - Range;
```

```
                    IF N < 0 THEN N := 0;
                    Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
                    Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                    Dval1 := Myarray1^;
                    IF (Dval - Dval1) > Step THEN Val := 32767;

                    M := I;
                    N := J;
                    Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
                    Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                    Dval1 := Myarray1^;
                    IF (Dval - Dval1) > Step THEN Val := 32767;

                    M := I;
                    N := J + Range;
                    IF N > Ourimages[Num].ImageWidth-1 THEN
                                                    N := Ourimages[Num].ImageWidth - 1;
                    Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
                    Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                    Dval1 := Myarray1^;
                    IF (Dval - Dval1) > Step THEN Val := 32767;

                    M := I + Range;
                    IF M > Ourimages[Num].ImageWidth-1 THEN
                                                    M := Ourimages[Num].ImageWidth-1;
                    N := J - Range;
                    IF N < 0 THEN N := - N;
                    Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
                    Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                    Dval1 := Myarray1^;
                    IF (Dval - Dval1) > Step THEN Val := 32767;

                    M := I + Range;
                    IF M > Ourimages[Num].ImageWidth-1 THEN
                                                    M := Ourimages[Num].ImageWidth-1;
                    N := J;
                    Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
                    Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                    Dval1 := Myarray1^;
                    IF (Dval - Dval1) > Step THEN Val := 32767;

                    M := I + Range;
                    IF M > Ourimages[Num].ImageWidth-1 THEN
                                                    M := Ourimages[Num].ImageWidth-1;
                    N := J + Range;
                    IF N > Ourimages[Num].ImageWidth-1 THEN
                                                    N := Ourimages[Num].ImageWidth-1;
                    Place1 := 2*(Ourimages[Num].ImageWidth * N + M);
                    Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place1);
                    Dval1 := Myarray1^;
                    IF (Dval - Dval1) > Step THEN Val := 32767;

                    Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                    Myarray3^ := Val;
                END;
            END;
        FindLimits(Images);
        Setourpalette(Images);
        Adjust(Images);

        Initcursor;
    END;
```

```
{-------------------------------------------------------------------------}
  {$S Part1}
  { This is the edge-detection calling routine. }

  PROCEDURE Doedgedet;

    VAR
      Itemhit, Itemtype    : Integer;
      Box                  : Rect;
      Step, Range          : Longint;
      Item                 : Handle;
      Sometext             : Str255;

    BEGIN
      Mydialog := Getnewdialog(139, nil, Pointer( - 1));
      SetOKButton(MyDialog);

      Modaldialog(nil, Itemhit);
      CASE Itemhit OF
        1:
          BEGIN
            Getditem(Mydialog, 3, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Step);

            Getditem(Mydialog, 6, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Range);

            Disposdialog(Mydialog);

            Edge(Step, Range);
          END;
        2: Disposdialog(Mydialog);
      END;
    END;

{-------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine which performs various arithmatic operations on the image. }

  PROCEDURE DoArith;

    VAR
      Itemhit, Itemtype, Num, i, fun    : Integer;
      Thevalue, Place, Plate1, Plate3   : Longint;
      Sometext,Outstr1                  : Str255;
      Box                               : Rect;
      Myarray3, Myarray1, Myarray2      : Intptr;
      Item                              : Handle;
      Ourcursorh                        : Curshandle;

    BEGIN
      Num := Getwrefcon(Frontwindow);
      Mydialog := Getnewdialog(146, nil, Pointer( - 1));
      SetOKButton(MyDialog);

      GetDItem(mydialog,4, ItemType, Item, Box);
      SetCtlValue(ControlHandle(Item), 1);
      fun:=4;

      REPEAT
        ModalDialog(nil, Itemhit);
```

```
CASE Itemhit OF
   4:
      BEGIN
         FOR i:=4 TO 11 DO
            BEGIN
               GetDItem(mydialog, i, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
            END;
         GetDItem(mydialog, 4, ItemType, Item, Box);
         SetCtlValue(ControlHandle(Item), 1);
         fun := 4;
      END;
   5:
      BEGIN
         FOR i:=4 TO 11 DO
            BEGIN
               GetDItem(mydialog, i, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
            END;
         GetDItem(mydialog, 5, ItemType, Item, Box);
         SetCtlValue(ControlHandle(Item), 1);
         fun := 5;
      END;
   6:
      BEGIN
         FOR i:=4 TO 11 DO
            BEGIN
               GetDItem(mydialog, i, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
            END;
         GetDItem(mydialog, 6, ItemType, Item, Box);
         SetCtlValue(ControlHandle(Item), 1);
         fun := 6;
      END;
   7:
      BEGIN
         FOR i:=4 TO 11 DO
            BEGIN
               GetDItem(mydialog, i, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
            END;
         GetDItem(mydialog, 7, ItemType, Item, Box);
         SetCtlValue(ControlHandle(Item), 1);
         fun := 7;
      END;
   8:
      BEGIN
         FOR i:=4 TO 11 DO
            BEGIN
               GetDItem(mydialog, i, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
            END;
         GetDItem(mydialog, 8, ItemType, Item, Box);
         SetCtlValue(ControlHandle(Item), 1);
         fun := 8;
      END;
   9:
      BEGIN
         FOR i:=4 TO 11 DO
            BEGIN
               GetDItem(mydialog, i, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
```

```
            END;
          GetDItem(mydialog, 9, ItemType, Item, Box);
          SetCtlValue(ControlHandle(Item), 1);
          fun := 9;
        END;
      10:
        BEGIN
          FOR i:=4 TO 11 DO
            BEGIN
              GetDItem(mydialog, i, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 0);
            END;
          GetDItem(mydialog, 10, ItemType, Item, Box);
          SetCtlValue(ControlHandle(Item), 1);
          fun := 10;
        END;
      11:
        BEGIN
          FOR i:=4 TO 11 DO
            BEGIN
              GetDItem(mydialog, i, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 0);
            END;
          GetDItem(mydialog, 11, ItemType, Item, Box);
          SetCtlValue(ControlHandle(Item), 1);
          fun := 11;
        END;
      END;

UNTIL (Itemhit=1) or (Itemhit=2);

CASE Itemhit OF
  1:
    BEGIN
      CASE fun OF
        8:
          BEGIN
            Getditem(Mydialog, 12, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue);
          END;
        9:
          BEGIN
            Getditem(Mydialog, 13, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue);
          END;
        10:
          BEGIN
            Getditem(Mydialog, 14, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue);
          END;
        11:
          BEGIN
            Getditem(Mydialog, 15, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue);
          END;
      END; {END of case fun}

      Disposdialog(Mydialog);
      Ourcursorh := Getcursor(4);
```

```
Setcursor(Ourcursorh^^);

CASE fun OF
   4: Outstr1 := 'Absolute Image';
   5: Outstr1 := 'Square Image';
   6: Outstr1 := 'Square Root Image';
   7: Outstr1 := 'Reverse Image';
   8: Outstr1 := Concat('Image Plus by ', Sometext);
   9: Outstr1 := Concat('Image Minus by ', Sometext);
   10: Outstr1 := Concat('Image Multiply by ', Sometext);
   11: Outstr1 := Concat('Image Divided by ', Sometext);
END;

Newpiximage(Outstr1, Ourimages[Num].Imagesize);
Place := 0;

CASE fun OF
   4:
      BEGIN
         REPEAT
         Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
         Plate1 := Myarray1^;
         Plate3 := Abs(Plate1);
         Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
         Myarray3^ := Plate3;

         Place := Place + 2;
         UNTIL Place >= Ourimages[Num].Imagesize;
      END;
   5:
      BEGIN
         REPEAT
         Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
         Plate1 := Myarray1^;
         Plate3 := (Plate1)*(Plate1);
         Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
         Myarray3^ := Plate3;

         Place := Place + 2;
         UNTIL Place >= Ourimages[Num].Imagesize;
      END;
   6:
      BEGIN
         REPEAT
         Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
         Plate1 := Myarray1^;
         Plate3 := Round(Sqrt(Plate1));
         Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
         Myarray3^ := Plate3;

         Place := Place + 2;
         UNTIL Place >= Ourimages[Num].Imagesize;
      END;
   7:
      BEGIN
         REPEAT
         Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
         Plate1 := Myarray1^;
         Plate3 := (-1)*Plate1;
         Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
         Myarray3^ := Plate3;

         Place := Place + 2;
```

```
                    UNTIL Place >= Ourimages[Num].Imagesize;
                END;
            8:
                BEGIN
                    REPEAT
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Plate3 := Plate1 + Thevalue;
                        Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                        Myarray3^ := Plate3;

                        Place := Place + 2;
                    UNTIL Place >= Ourimages[Num].Imagesize;
                END;
            9:
                BEGIN
                    REPEAT
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Plate3 := Plate1 - Thevalue;
                        Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                        Myarray3^ := Plate3;

                        Place := Place + 2;
                    UNTIL Place >= Ourimages[Num].Imagesize;
                END;
            10:
                BEGIN
                    REPEAT
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Plate3 :=Plate1 * Thevalue;
                        Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                        Myarray3^ := Plate3;

                        Place := Place + 2;
                    UNTIL Place >= Ourimages[Num].Imagesize;
                END;
            11:
                BEGIN
                    IF (Thevalue <> 0) THEN
                        BEGIN
                            REPEAT
                                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                                Plate1 := Myarray1^;
                                Plate3 :=Round(Plate1 / Thevalue);
                                Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                                Myarray3^ := Plate3;

                                Place := Place + 2;
                            UNTIL Place >= Ourimages[Num].Imagesize;
                        END;
                END;
        END; {end of case fun}

        FindLimits(Images);
        Setourpalette(Images);
        Adjust(Images);
        Initcursor;
    END;    {end of case 1}
  2: Disposdialog(Mydialog);
 END;
END;
```

```
{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine which performs the various mathmatic operations on the image. }

  PROCEDURE DoMath;

    VAR
      Itemhit, Itemtype, Num, i, fun    : Integer;
      Sometext, Outstrl                 : Str255;
      Box                               : Rect;
      Thevalue, Place, Temp             : Longint;
      RTemp                             : Extended;
      Myarray1, Myarray2                : Intptr;
      Item                              : Handle;
      Ourcursorh                        : Curshandle;

    BEGIN
      Num := Getwrefcon(Frontwindow);
      Mydialog := Getnewdialog(157, nil, Pointer( - 1));
      SetOKButton(MyDialog);

      GetDItem(mydialog, 4, ItemType, Item, Box);
      SetCtlValue(ControlHandle(Item), 1);
      fun := 4;

      REPEAT
        ModalDialog(nil, Itemhit);

        CASE Itemhit OF
          4:
            BEGIN
              FOR i:= 4 TO 6 DO
                BEGIN
                  GetDItem(mydialog, i, ItemType, Item, Box);
                  SetCtlValue(ControlHandle(Item), 0);
                END;
              GetDItem(mydialog, 4, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 1);
              fun := 4;
            END;
          5:
            BEGIN
              FOR i:=4 TO 6 DO
                BEGIN
                  GetDItem(mydialog, i, ItemType, Item, Box);
                  SetCtlValue(ControlHandle(Item), 0);
                END;
              GetDItem(mydialog, 5, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 1);
              fun := 5;
            END;
          6:
            BEGIN
              FOR i:=4 TO 6 DO
                BEGIN
                  GetDItem(mydialog, i, ItemType, Item, Box);
                  SetCtlValue(ControlHandle(Item), 0);
                END;
              GetDItem(mydialog, 6, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 1);
              fun := 6;
            END;
```

```
    END;

UNTIL (Itemhit=1) or (Itemhit=2);

CASE Itemhit OF
   1:
      BEGIN
         Getditem(Mydialog, 7, Itemtype, Item, Box);
         Getitext(Item, Sometext);
         Stringtonum(Sometext, Thevalue);

         Disposdialog(Mydialog);
         Ourcursorh := Getcursor(4);
         Setcursor(Ourcursorh^^);

         CASE fun OF
            4: Outstr1:= Concat(Sometext,'*Arc Sin(Image)');
            5: Outstr1:= Concat(Sometext,'*Arc Cos(Image)');
            6: Outstr1:= Concat(Sometext,'*Arc Tan(Image)');
         END;

         Newpiximage(Outstr1, Ourimages[Num].Imagesize);
         Place := 0;

         CASE fun OF
            4:
               BEGIN
                  REPEAT
                     Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                     Temp := Myarray1^;
                     IF (Temp > 1000) THEN Temp := 1000
                        ELSE IF (Temp < -1000) THEN Temp := -1000;
                     RTemp := Temp/1000;
                     Temp := Round(Thevalue*Arctan(RTemp/Sqrt(1 -
                                                         RTemp*RTemp)));
                     Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                     Myarray2^ := Round(Temp);

                     Place := Place + 2;
                  UNTIL Place >= Ourimages[Num].Imagesize;
               END;
            5:
               BEGIN
                  REPEAT
                     Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                     Temp := Myarray1^;
                     IF (Temp > 1000) THEN Temp := 1000
                        ELSE IF (Temp < -1000) THEN Temp := -1000;
                     RTemp := Temp/1000;
                     Temp := Round(Thevalue*(Pi/2 - Arctan(RTemp/Sqrt(1 -

                                                         RTemp*RTemp))));
                     Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                     Myarray2^ := Round(Temp);

                     Place := Place + 2;
                  UNTIL Place >= Ourimages[Num].Imagesize;
               END;
            6:
               BEGIN
                  REPEAT
                     Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                     Temp := Myarray1^;
```

```
                        Temp := Round(Thevalue*Arctan(Temp));
                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                        Myarray2^ := Round(Temp);

                        Place := Place + 2;
                    UNTIL Place >= Ourimages[Num].Imagesize;
                END;
            END; {end of case fun}

            FindLimits(Images);
            Setourpalette(Images);
            Adjust(Images);
            Initcursor;
        END;    {end of case 1}
      2: Disposdialog(Mydialog);
    END;
  END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine which performs various intensity-filtering operations. }

  PROCEDURE DoFilter;

    VAR
      Thevalue1, Thevalue2, Thevalue3, Place, Plate1, Plate2   : Longint;
      Itemhit, Itemtype, Num, i, fun                           : Integer;
      Sometext, Outstr1                                        : Str255;
      Box                                                      : Rect;
      Item                                                     : Handle;
      Ourcursorh                                               : Curshandle;
      Myarray1, Myarray2                                       : Intptr;

    BEGIN
      Num := Getwrefcon(Frontwindow);
      Mydialog := Getnewdialog(153, nil, Pointer(-1));
      SetOKButton(MyDialog);

      GetDItem(mydialog, 4, ItemType, Item, Box);
      SetCtlValue(ControlHandle(Item), 1);
      fun := 4;

      REPEAT
        ModalDialog(nil, Itemhit);

        CASE Itemhit OF
          3:
            BEGIN
              FOR i:=3 TO 5 DO
                BEGIN
                  GetDItem(mydialog, i, ItemType, Item, Box);
                  SetCtlValue(ControlHandle(Item), 0);
                END;
              GetDItem(mydialog, 3, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 1);
              fun := 3;
            END;
          4:
            BEGIN
              FOR i:=3 TO 5 DO
                BEGIN
                  GetDItem(mydialog, i, ItemType, Item, Box);
                  SetCtlValue(ControlHandle(Item), 0);
```

```
            END;
          GetDItem(mydialog, 4, ItemType, Item, Box);
          SetCtlValue(ControlHandle(Item), 1);
          fun := 4;
        END;
      5:
        BEGIN
          FOR i:=3 TO 5 DO
            BEGIN
              GetDItem(mydialog, i, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 0);
            END;
          GetDItem(mydialog, 5, ItemType, Item, Box);
          SetCtlValue(ControlHandle(Item), 1);
          fun := 5;
        END;
      END;

UNTIL (Itemhit=1) or (Itemhit=2);

CASE Itemhit OF
  1:
    BEGIN
      CASE fun OF
        3:
          BEGIN
            Getditem(Mydialog, 14, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue2);
            Getditem(Mydialog, 12, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue1);
          END;
        4:
          BEGIN
            Getditem(Mydialog, 9, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue2);
            Getditem(Mydialog, 7, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue1);
          END;
        5:
          BEGIN
            Getditem(Mydialog, 16, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue1);
            Getditem(Mydialog, 18, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue2);
            Getditem(Mydialog, 20, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Thevalue3);
          END;
      END; {end of case fun}

      Disposdialog(Mydialog);
      Ourcursorh := Getcursor(4);
      Setcursor(Ourcursorh^^);

      CASE fun OF
        3: Outstr1 := Concat('Low Pass (',Sometext,')');
        4: Outstr1 := Concat('High Pass (',Sometext,')');
```

```
                5: Outstr1 := 'Band Pass';
            END;
            Newpiximage(Outstr1, Ourimages[Num].Imagesize);
            Place := 0;

            CASE fun OF
                3:
                    BEGIN
                        REPEAT
                            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                            Plate1 := Myarray1^;
                            IF (Plate1 > TheValue1) THEN
                                Plate2 := TheValue2
                            ELSE Plate2 := Plate1;
                            Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                            Myarray2^ := Plate2;

                            Place := Place + 2;
                        UNTIL Place >= Ourimages[Num].Imagesize;
                    END;
                4:
                    BEGIN
                        REPEAT
                            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                            Plate1 := Myarray1^;
                            IF (Plate1 < TheValue1) THEN
                                Plate2 := TheValue2
                            ELSE Plate2 := Plate1;
                            Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                            Myarray2^ := Plate2;

                            Place := Place + 2;
                        UNTIL Place >= Ourimages[Num].Imagesize;
                    END;
                5:
                    BEGIN
                        REPEAT
                            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                            Plate1 := Myarray1^;
                            IF (Plate1 > TheValue1) OR (Plate1 < TheValue2)  THEN
                                Plate2 := TheValue3
                            ELSE Plate2 := Plate1;
                            Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);
                            Myarray2^ := Plate2;

                            Place := Place + 2;
                        UNTIL Place >= Ourimages[Num].Imagesize;
                    END;
            END; {end of case fun}

            FindLimits(Images);
            Setourpalette(Images);
            Adjust(Images);
            Initcursor;
        END;     {end of case 1}
    2: Disposdialog(Mydialog);
  END;
END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine doing the radial-averaging on the image. }
```

```
PROCEDURE RadialAve(LeftX, RightX, TopY, BottomY, Num: Integer);

  VAR
    Horiz, Verti, CentreX, CentreY, Place, Plate1, Plate2      : Longint;
    Dim, HDim, HSize, DataVal1, DataVal2, i, j, Radius, w      : Longint;
    m                                                          : Integer;
    Myarray1, Myarray2                                         : Intptr;
    Amp, N                                                     : Array [0..127] OF Longint;
    Ourcursorh                                                 : Curshandle;

  BEGIN
    Ourcursorh := Getcursor(4);
    Setcursor(Ourcursorh^^);

    Horiz := Round((RightX - LeftX)/2);
    Verti := Round((BottomY - TopY)/2);
    CentreX := LeftX + Horiz;
    CentreY := TopY + Verti;
    Radius := Round((Horiz + Verti)/2);

    Dim := Ourimages[Num].ImageWidth;
    HDim := Dim DIV 2;
    HSize := Ourimages[Num].Imagesize DIV 2;

    Newpiximage('RadialAverage', Ourimages[Num].Imagesize);

    FOR m:=0 TO HDim-1 DO
      BEGIN
        Amp[m] := 0;
        N[m] := 0;
      END;

    FOR j:=0 TO Dim-1 DO
      BEGIN
        FOR i:=0 TO Dim-1 DO
          BEGIN
            place := 2*(j*Dim + i);
            IF (((i - CentreX)*(i - CentreX) + (j - CentreY)*(j - CentreY))
                                  <= (Radius + 1)*(Radius + 1)) THEN
              BEGIN
                Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                DataVal1 := Myarray1^;

                w := HSize*((i - CentreX)*(i - CentreX) + (j - CentreY)*(j - CentreY));
                m := Round(Sqrt(w/HSize));
                Amp[m] := Amp[m] + DataVal1;
                N[m] :=N[m] + 1;
              END;
          END;    {end of i}
      END;    {end of j}

    FOR m:=0 TO HDim-1 DO
        IF N[m] <> 0 THEN Amp[m] := Round(Amp[m]/N[m])  ELSE Amp[m] := 0;

    FOR j:=0 TO Dim-1 DO
      BEGIN
        FOR i:= 0 TO Dim-1 DO
          BEGIN
            place := 2*(j*Dim + i);
            Myarray2 := Intptr(Ord(Ourimages[Images].Data^) + Place);

            IF (((i - CentreX)*(i - CentreX) + (j - CentreY)*(j - CentreY))
                                  > (Radius + 1)*(Radius + 1)) THEN
```

```
                    BEGIN
                        Myarray2^ := 0;
                    END
                ELSE
                    BEGIN
                        w := HSize*((i - CentreX)*(i - CentreX) + (j - CentreY)*(j - CentreY));
                        m := Round(Sqrt(w/HSize));
                        Plate2 := Amp[m];
                        Myarray2^ := Plate2;
                    END;
            END;    {end of i}
        END;    {end of j}

    FindLimits(Images);
    Setourpalette(Images);
    Adjust(Images);
    Initcursor;
END;

{-------------------------------------------------------------------------------}
{$S Part1}
{ This is the radial-averaging calling routine which deals with the interfacing with the user. }

PROCEDURE DoRadialAve;

    VAR
        LeftX, RightX, TopY, BottomY, Itemhit, Num, Itemtype : Integer;
        Item        : Handle;
        Box         : Rect;
        Sometext    : Str255;
        Thevalue    : Longint;

    BEGIN
        IF Frontwindow <> nil THEN
            BEGIN
                Num := Getwrefcon(Frontwindow);
                Mydialog := Getnewdialog(144, nil, Pointer( - 1));
                SetOKButton(MyDialog);

                Modaldialog(nil, Itemhit);
                CASE Itemhit OF
                    1:
                        BEGIN
                            Getditem(Mydialog, 8, Itemtype, Item, Box);
                            Getitext(Item, Sometext);
                            Stringtonum(Sometext, Thevalue);
                            TopY := Thevalue;
                            Getditem(Mydialog, 9, Itemtype, Item, Box);
                            Getitext(Item, Sometext);
                            Stringtonum(Sometext, Thevalue);
                            LeftX := Thevalue;
                            Getditem(Mydialog, 10, Itemtype, Item, Box);
                            Getitext(Item, Sometext);
                            Stringtonum(Sometext, Thevalue);
                            BottomY := Thevalue;
                            Getditem(Mydialog, 11, Itemtype, Item, Box);
                            Getitext(Item, Sometext);
                            Stringtonum(Sometext, Thevalue);
                            RightX := Thevalue;

                            Disposdialog(Mydialog);
                            UpdateWindow;
```

```
                    RadialAve(LeftX,RightX,TopY,BottomY,Num);

                END;
              2: Disposdialog(Mydialog);
            END;     {end of case}
          END;     {end of Frontwindow <> nil}
      END;

{-------------------------------------------------------------------------------}
  {$S Part1}
  { This is the routine which applys the slope-adjustnent on the image. }

  PROCEDURE DoAnti_Sloping;

    VAR
      Thevalue, place, Theplace, Plate1, Plate2, i, j, Dim    : Longint;
      Itemhit, Itemtype, Num, func                            : Integer;
      Sometext, Outstr1, Outstr2                              : Str255;
      Item                                                    : Handle;
      myarray1, myarray2                                      : IntPtr;
      Box                                                     : Rect;

    BEGIN
      Num := Getwrefcon(Frontwindow);
      Mydialog := Getnewdialog(159, nil, Pointer( - 1));
      SetOKButton(MyDialog);

      GetDItem(mydialog, 4, ItemType, Item, Box);
      SetCtlValue(ControlHandle(Item), 1);
      func := 1;

      REPEAT
        ModalDialog(nil, Itemhit);
        CASE Itemhit OF
          4:
            BEGIN
              GetDItem(mydialog, 5, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 0);
              GetDItem(mydialog, 4, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 1);
              func := 1;
            END;
          5:
            BEGIN
              GetDItem(mydialog, 4, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 0);
              GetDItem(mydialog, 5, ItemType, Item, Box);
              SetCtlValue(ControlHandle(Item), 1);
              func := 2;
            END;
        END;
      UNTIL (Itemhit=1) or (Itemhit=2);

      CASE Itemhit OF
        1:
          BEGIN
            CASE func OF
              1:
                BEGIN
                  Getditem(Mydialog, 7, Itemtype, Item, Box);
                  Getitext(Item, Sometext);
                  Stringtonum(Sometext, Theplace);
                END;
```

```
    2:
        BEGIN
            Getditem(Mydialog, 8, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            Stringtonum(Sometext, Theplace);
        END;
END;

Getditem(Mydialog, 13, Itemtype, Item, Box);
Getitext(Item, Sometext);
Stringtonum(Sometext, Thevalue);

CASE func OF
    1:
        BEGIN
            NumToString(Thevalue, Sometext);
            NumToString(Theplace, Outstr2);
            Outstr1:= Concat('X(',Outstr2,') ramped ', Sometext);
        END;
    2:
        BEGIN
            NumToString(Thevalue, Sometext);
            NumToString(Theplace, Outstr2);
            Outstr1:= Concat('Y(',Outstr2,') ramped ', Sometext);
        END;
END;
Disposdialog(Mydialog);

Newpiximage(Outstr1, Ourimages[Num].Imagesize);
Dim := Ourimages[Num].Imagewidth;
Place := 0;

CASE func OF
    1:
        BEGIN
            FOR j:=0 TO Dim-1 DO
                BEGIN
                    FOR i:=0 TO Dim-1 DO
                        BEGIN
                            Place := 2*(j*Dim + i);
                            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                            Plate1 := Myarray1^;

                            IF (i < Theplace) THEN
                                BEGIN
                                    Plate2 := Plate1 - Thevalue*(Theplace - i);
                                END
                            ELSE IF (i >= Theplace) THEN
                                BEGIN
                                    Plate2 := Plate1 + Thevalue*(i - Theplace);
                                END;

                            Myarray2 := Intptr(Ord(Ourimages[Images].Data^)+Place);
                            Myarray2^ := Plate2;
                        END;     {end of i}
                END;     {end of j}
        END;
    2:
        BEGIN
            FOR j:=0 TO Dim-1 DO
                BEGIN
                    FOR i:= 0 TO Dim-1 DO
                        BEGIN
```

```
                                        Place := 2*(j*Dim + i);
                                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                                        Plate1 := Myarray1^;

                                        IF (j < Theplace) THEN
                                           BEGIN
                                              Plate2 := Plate1 - Thevalue*(Theplace - j);
                                           END
                                        ELSE IF (j >= Theplace) THEN
                                           BEGIN
                                              Plate2 := Plate1 + Thevalue*(j - Theplace);
                                           END;

                                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^)+Place);
                                        Myarray2^ := Plate2;
                                     END;     {end of i}
                                  END;     {end of j}
                               END;
                            END;

                            FindLimits(Images);
                            Setourpalette(Images);
                            Adjust(Images);
                            UpdateAllWindow;
                            Updatewindow;

                         END;     {end of case 1}
                      2: Disposdialog(Mydialog);
                   END;
                END;

{-------------------------------------------------------------------------------}
   {$S Part1}
   { This is the routine which does the operation of linear combination. }

   PROCEDURE Linearcom(Currentimfact, Previousimfact: Longint);

      VAR
         Place, Plate1, Plate2, Plate3      : Longint;
         Myarray3, Myarray1, Myarray2       : Intptr;
         Num                                : Integer;
         Ourcursorh                         : Curshandle;

      BEGIN
         Ourcursorh := Getcursor(4);
         Setcursor(Ourcursorh^^);

         Num := Getwrefcon(Frontwindow);
         Newpiximage('Image Linear Combination', Ourimages[Num].Imagesize);
         Place := 0;

         REPEAT
            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
            Plate1 := Myarray1^;
            Myarray2 := Intptr(Ord(Ourimages[Num - 1].Data^) + Place);
            Plate2 := Myarray2^;
            Plate3 := Currentimfact * Plate1 + Previousimfact * Plate2;
            Plate3 := (Plate3) DIV (100);
            Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
            Myarray3^ := Plate3;

            Place := Place + 2;
```

```
        UNTIL Place >= Ourimages[Num].Imagesize;

        FindLimits(Images);
        Setourpalette(Images);
        Adjust(Images);
        Initcursor;
    END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This is the linear-combination calling routine. }

    PROCEDURE Dolinearcom;

        VAR
            Currentimfact, Previousimfact, Thevalue : Longint;
            Itemhit, Itemtype                       : Integer;
            Sometext                                : Str255;
            Box                                     : Rect;
            Item                                    : Handle;

        BEGIN
            Mydialog := Getnewdialog(131, nil, Pointer( - 1));
            SetOKButton(MyDialog);

            Modaldialog(nil, Itemhit);
            CASE Itemhit OF
                1:
                    BEGIN
                        Getditem(Mydialog, 3, Itemtype, Item, Box);
                        Getitext(Item, Sometext);
                        Stringtonum(Sometext, Thevalue);
                        Currentimfact := Thevalue;

                        Getditem(Mydialog, 6, Itemtype, Item, Box);
                        Getitext(Item, Sometext);
                        Stringtonum(Sometext, Thevalue);
                        Previousimfact := Thevalue;
                        Disposdialog(Mydialog);

                        Linearcom(Currentimfact, Previousimfact);
                    END;
                2: Disposdialog(Mydialog);
            END;
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This is the routine which does the modulus operation. }

    PROCEDURE Modulus;

        VAR
            Place, Plate1, Plate2, Plate3, DataVal, i, j, k, m  : Longint;
            Myarray3, Myarray1, Myarray2                        : Intptr;
            Num                                                 : Integer;
            Ourcursorh                                          : Curshandle;

        BEGIN
            Ourcursorh := Getcursor(4);
            Setcursor(Ourcursorh^^);

            Num := Getwrefcon(Frontwindow);
```

```
        Newpiximage('Modulus', Ourimages[Num].Imagesize);
        Place := 0;

        REPEAT
            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
            Plate1 := Myarray1^;
            Myarray2 := Intptr(Ord(Ourimages[Num - 1].Data^) + Place);
            Plate2 := Myarray2^;
            Plate3 := Plate1 * Plate1 + Plate2 * Plate2;
            Plate3 := Trunc(SQRT(Plate3));
            Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
            Myarray3^ := Plate3;

            Place := Place + 2;
        UNTIL Place >= Ourimages[Num].Imagesize;

        FindLimits(Images);
        Setourpalette(Images);
        Adjust(Images);
        Initcursor;
    END;

{-----------------------------------------------------------------------}
    {$S Part1}
    { This is the modulus calling routine. }

    PROCEDURE DoModulus;

        VAR
            Itemhit    : Integer;

        BEGIN
            Mydialog := Getnewdialog(145, nil, Pointer( - 1));
            SetOKButton(MyDialog);

            Modaldialog(nil, Itemhit);
            CASE Itemhit OF
                1:
                    BEGIN
                        Disposdialog(Mydialog);
                        Modulus;
                    END;
                2: Disposdialog(Mydialog);
            END;
        END;

{-----------------------------------------------------------------------}
    {$S Part1}
    { This is the routine which does the ratio operation of the two images. }

    PROCEDURE Ratio(Thresh: Longint);

        VAR
            Place, Plate1, Plate2, Plate3      : Longint;
            Myarray3, Myarray1, Myarray2       : Intptr;
            Num                                : Integer;
            Ourcursorh                         : Curshandle;

        BEGIN
            Ourcursorh := Getcursor(4);
            Setcursor(Ourcursorh^^);

            Num := Getwrefcon(Frontwindow);
```

```
        Newpiximage('1000*Second/First', Ourimages[Num].Imagesize);
        Place := 0;

        REPEAT
            Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
            Plate1 := Myarray1^;
            Myarray2 := Intptr(Ord(Ourimages[Num - 1].Data^) + Place);
            Plate2 := Myarray2^;

            IF (Plate1 >= Thresh) AND (Plate2 >= Thresh) THEN
                Plate3 := Round(1000*Plate1/Plate2)
            ELSE    Plate3 := 0;

            Myarray3 := Intptr(Ord(Ourimages[Images].Data^) + Place);
            Myarray3^ := Plate3;

            Place := Place + 2;
        UNTIL Place >= Ourimages[Num].Imagesize;

        FindLimits(Images);
        Setourpalette(Images);
        Adjust(Images);
        Initcursor;
    END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This is the ratio calling routine. }

    PROCEDURE DoRatio;

        VAR
            Itemhit, Itemtype    : Integer;
            Box                  : Rect;
            Item                 : Handle;
            Sometext             : Str255;
            TheValue             : Longint;

        BEGIN
            Mydialog := Getnewdialog(152, nil, Pointer( - 1));
            SetOKButton(MyDialog);

            Modaldialog(nil, Itemhit);
            CASE Itemhit OF
                1:
                    BEGIN
                        Getditem(Mydialog, 7, Itemtype, Item, Box);
                        Getitext(Item, Sometext);
                        StringToNum(Sometext, TheValue);
                        Disposdialog(Mydialog);

                        Ratio(TheValue);
                    END;
                2: Disposdialog(Mydialog);
            END;
        END;

{-------------------------------------------------------------------------}
    {$S Part1}
    { This is the routine which shifts the position of the image around. }

    PROCEDURE DoShifting;
```

```
VAR
   Thevalue, i, j, Place, Plate1, Temp, Temp1  : Longint;
   Itemhit, Itemtype, Num, fun, k              : Integer;
   Sometext, Outstr1                           : Str255;
   Box                                         : Rect;
   Myarray1, Myarray2                          : Intptr;
   Item                                        : Handle;
   Ourcursorh                                  : Curshandle;

BEGIN
   Num := Getwrefcon(Frontwindow);
   Mydialog := Getnewdialog(156, nil, Pointer( - 1));
   SetOKButton(MyDialog);

   GetDItem(mydialog, 4, ItemType, Item, Box);
   SetCtlValue(ControlHandle(Item), 1);
   fun := 1;

   REPEAT
      ModalDialog(nil, Itemhit);

      CASE Itemhit OF
         4:
            BEGIN
               FOR k:=4 TO 7 DO
                  BEGIN
                     GetDItem(mydialog, k, ItemType, Item, Box);
                     SetCtlValue(ControlHandle(Item), 0);
                  END;
               GetDItem(mydialog, 4, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 1);
               fun := 1;
            END;
         5:
            BEGIN
               FOR k:=4 TO 7 DO
                  BEGIN
                     GetDItem(mydialog, k, ItemType, Item, Box);
                     SetCtlValue(ControlHandle(Item), 0);
                  END;
               GetDItem(mydialog, 5, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 1);
               fun := 2;
            END;
         6:
            BEGIN
               FOR k:=4 TO 7 DO
                  BEGIN
                     GetDItem(mydialog, k, ItemType, Item, Box);
                     SetCtlValue(ControlHandle(Item), 0);
                  END;
               GetDItem(mydialog, 6, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 1);
               fun := 3;
            END;
         7:
            BEGIN
               FOR k:=4 TO 7 DO
                  BEGIN
                     GetDItem(mydialog, k, ItemType, Item, Box);
                     SetCtlValue(ControlHandle(Item), 0);
                  END;
               GetDItem(mydialog, 7, ItemType, Item, Box);
```

```
                SetCtlValue(ControlHandle(Item), 1);
                fun := 4;
            END;
        END;
UNTIL (Itemhit=1) or (Itemhit=2);

CASE Itemhit OF
   1:
      BEGIN
         CASE fun OF
            1,2,3,4:
               BEGIN
                  Getditem(Mydialog, 7+fun, Itemtype, Item, Box);
                  Getitext(Item, Sometext);
                  Stringtonum(Sometext, Thevalue);
               END;
         END; {end of case fun}

         Disposdialog(Mydialog);
         Ourcursorh := Getcursor(4);
         Setcursor(Ourcursorh^^);

         CASE fun OF
            1: Outstr1 := 'Image Up';
            2: Outstr1 := 'Image Down';
            3: Outstr1 := 'Image Left';
            4: Outstr1 := 'Image Right';
         END;

         Newpiximage(Outstr1, Ourimages[Num].Imagesize);

         CASE fun OF
            1:
               BEGIN
                  Temp := 2*Thevalue*Ourimages[Num].ImageWidth;
                  Temp1 := Ourimages[Num].Imagesize - Temp;

                  Place := Temp;
                  WHILE Place < Ourimages[Num].Imagesize DO
                     BEGIN
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^)
                                                          + Place - Temp);
                        Myarray2^ := Plate1;

                        Place := Place + 2;
                     END;

                  Place := 0;
                  WHILE Place < Temp DO
                     BEGIN
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^)
                                                          + Place + Temp1);
                        Myarray2^ := Plate1;

                        Place := Place + 2;
                     END;
               END;
            2:
               BEGIN
```

```
            Temp := 2*Thevalue*Ourimages[Num].ImageWidth;
            Temp1 := Ourimages[Num].Imagesize - Temp;

            Place := Temp1;
            WHILE Place < Ourimages[Num].Imagesize DO
               BEGIN
                  Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                  Plate1 := Myarray1^;
                  Myarray2 := Intptr(Ord(Ourimages[Images].Data^)
                                                      + Place - Temp1);
                  Myarray2^ := Plate1;

                  Place := Place + 2;
               END;

            Place := 0;
            WHILE Place < Temp1 DO
               BEGIN
                  Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                  Plate1 := Myarray1^;
                  Myarray2 := Intptr(Ord(Ourimages[Images].Data^)
                                                        + Place + Temp);
                  Myarray2^ := Plate1;

                  Place := Place + 2;
               END;
         END;
      3:
         BEGIN
            FOR i := 1 TO Ourimages[Images].ImageWidth DO
               BEGIN
                  Temp := 2*(Ourimages[Images].ImageWidth - Thevalue);
                  FOR j := 1 TO Thevalue DO
                     BEGIN
                        Place := 2*((i-1)*Ourimages[Images].ImageWidth + j-1);
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^) +
                                                          Place + Temp);
                        Myarray2^ := Plate1;
                     END;

                  FOR j:=Thevalue + 1 TO Ourimages[Images].ImageWidth DO
                     BEGIN
                        Place := 2*((i-1)*Ourimages[Images].ImageWidth + j-1);
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                        Plate1 := Myarray1^;
                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^) +
                                                        Place - 2*Thevalue);
                        Myarray2^ := Plate1;
                     END;
               END;
         END;
      4:
         BEGIN
            FOR i := 1 TO Ourimages[Images].ImageWidth DO
               BEGIN
                  Temp := 2*(Ourimages[Images].ImageWidth-Thevalue);
                  FOR j := 1 TO Thevalue DO
                     BEGIN
                        Place := 2*((i - 1)*Ourimages[Images].ImageWidth +
                                                            j - 1) + Temp;
                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
```

```
                                        Plate1 := Myarray1^;
                                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^) +
                                                                        Place - Temp);
                                        Myarray2^ := Plate1;
                                    END;

                                FOR j := 1 TO Ourimages[Images].ImageWidth-Thevalue DO
                                    BEGIN
                                        Place := 2*((i-1)*Ourimages[Images].ImageWidth + j-1);
                                        Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
                                        Plate1 := Myarray1^;
                                        Myarray2 := Intptr(Ord(Ourimages[Images].Data^) +
                                                                    Place + 2*Thevalue);
                                        Myarray2^ := Plate1;
                                    END;
                            END;
                        END;
                    END; {end of case fun}

                    FindLimits(Images);
                    Setourpalette(Images);
                    Adjust(Images);
                    Initcursor;
                END;    {end of case 1}
            2: Disposdialog(Mydialog);
        END;
    END;

{-------------------------------------------------------------------------}
{-------------------------------------------------------------------------}
    {$S PictSave}
    { This is part of the PICT file saving procedure. }

    FUNCTION ColourQDExists :boolean;

        CONST
            ROM85Loc      = $28E;
            TwoHighMask   = $C000;

        TYPE
            WordPtr = ^Integer;

        VAR
            WD : WordPtr;

        BEGIN
            Wd := POINTER(ROM85Loc);
            ColourQDExists := (BitAnd(Wd^, TwoHighMask) = 0);
        END;

{-------------------------------------------------------------------------}
    {$S PictSave}
    { This is part of the PICT file saving procedure. }

    PROCEDURE GetPic (VAR thePic: PicHandle;
                      VAR IsWindow: boolean; QDExists: boolean; wholescreen: boolean);

        TYPE
            BitMapPtr = ^BitMap;
            BitMapHdl = ^BitMapPtr;

        VAR
            sorcRect, destRect, mapREct   : Rect;
```

```
    OldPort, WPort          : GrafPtr;
    clip, vis               : RgnHandle;
    FWin                    : WindowPtr;
    PixH                    : BitMapHdl;
    tempMap                 : BitMap;

BEGIN
    GetPort(OldPort);

    IF QDExists THEN
        BEGIN
            GetCWMgrPort(CGrafPtr(WPort));
            PixH := BitMapHdl(WPort^.portbits.baseAddr);
            mapRect := PixH^^.bounds;
        END
    ELSE
        BEGIN
            GetWMgrPort(WPort);
            mapRect := WPort^.portBits.bounds;
        END;

    FWin := FrontWindow;
    IsWindow := (NOT wholescreen) AND (FWin <> nil);
    IF IsWindow THEN
        BEGIN
            SetPort(FWin);
            sorcRect := FWin^.portRect;
            LocalToGlobal(sorcRect.topLeft);
            LocalToGlobal(sorcRect.botRight);
            OffSetRect(sorcRect, mapRect.left, mapRect.top);
        END
    ELSE
        sorcRect := MapRect;
    destRect.left := 0;
    destRect.right := sorcRect.right - sorcRect.left;
    destRect.top := 0;
    destRect.bottom := sorcRect.bottom - sorcRect.top;
    SetPort(WPort);
    vis := WPort^.visRgn;
    clip := WPOrt^.clipRgn;
    WPort^.visRgn := NewRgn;
    WPort^.clipRgn := NewRgn;
    SetRectRgn(WPort^.visRgn, -3000, -3000, 3000, 3000);
    SetRectRgn(WPort^.clipRgn, -3000, -3000, 3000, 3000);
    thePic := OpenPicture(destRect);
    CopyBits(WPort^.portBits, WPort^.Portbits, sorcRect, destRect, srcCopy, nil);
    ClosePicture;
    DisposeRgn(WPort^.visRgn);
    DisposeRgn(WPort^.clipRgn);
    WPort^.visRgn := vis;
    WPort^.clipRgn := clip;

    SetPort(OldPort);
END;

{-------------------------------------------------------------------------------}
  {$S PictSave}
  { This is part of the PICT file saving procedure. }

  PROCEDURE PickFIle(VAR theReply: SFReply; IsWindow: Boolean; QDExists: Boolean);

    VAR
      Pt        : Point;
```

```
    S1, S2    : str255;

BEGIN
    Pt.v := 60;
    Pt.h := 60;
    IF QDExists THEN
        S1 := 'Save Colour '
    ELSE
        S1 := 'Save ';
    IF IsWindow THEN
        S2 := 'Window PICT:'
    ELSE
        S2 := 'Screen PICT:';
    SFPutFile(Pt,CONCAT(S1, S2) ,'', nil,theReply);
END;
```

{----------}

```
{$S PictSave}
{ This is main part of the PICT file saving procedure. }

PROCEDURE SaveColourImage (thePic: PicHandle; theReply: SFReply);

    TYPE
        BufType = PACKED Array [1..128] OF Longint;

    VAR
        theErr          : OSErr;
        OK              : Boolean;
        FileRef, count  : Integer;
        BufSize         : Longint;
        Buf             : BufType;

    BEGIN
        OK := False;
        FileRef := 0;
        theErr := FSDelete(theReply.fname, theReply.vRefNum);
        IF (theErr = noErr) OR (theErr = fnfErr) THEN
            BEGIN
                theErr := Create(theReply.fname, theReply.vRefNum, 'MDRW', 'PICT');
                IF (theErr = noErr) THEN
                    BEGIN
                        theErr := FSOpen (theReply.fname, theReply.vRefNum, FileRef);
                        IF (theErr = noErr) THEN
                            BEGIN
                                FOR count := 1 TO 128 DO
                                Buf[count] := 0;
                                BufSize := 512;
                                theErr := FSWrite (FileRef, BufSize, @Buf);
                                IF (theErr = NoErr) THEN
                                    BEGIN
                                        BufSize := GetHandleSize(Handle(thePic));
                                        HLock(Handle(thePic));
                                        theErr := FSWrite (FileRef, BufSize, Ptr(thePic^));
                                        HUnLock(Handle(thePic));
                                        IF (theErr = NoErr) THEN
                                            BEGIN
                                                theErr := FSClose(FileRef);
                                                FileRef := 0;
                                                IF (theErr = noErr) THEN
                                                    BEGIN
                                                        theErr := FlushVol(nil, theReply.vRefNum);
                                                        OK := TRUE;
                                                    END;
```

```
                                END;
                            END;
                        END;
                    END;
                END;
            IF NOT Ok THEN SysBeep(60);
            IF (FileRef <> 0 ) THEN theErr := FSClose (FileRef);
        END;

{-------------------------------------------------------------------------}
    {$S PictSave}
    { This is part of the PICT file saving procedure. }

    PROCEDURE PictSave(wholescreen: boolean);

        VAR
            thePic          : PicHandle;
            theReply        : SFReply;
            IsWindow        : Boolean;
            QDExist         : Boolean;

        BEGIN
            QDExist := ColourQDEXists;
            GetPic(thepic, IsWindow, QDExist, wholescreen);
            PickFile(theReply, IsWindow, QDExist);
            IF theReply.good THEN SaveColourImage(thePic, theReply);
            KillPicture(thePic);
        END;

{-------------------------------------------------------------------------}
{-------------------------------------------------------------------------}
    {$S FFT2D}
    { This is the routine which applies the filter in the 2-D FFT procedure. }

    PROCEDURE ApFilter(HCenter, HExp, VCenter, VExp: Longint; Num: Integer);

        VAR
            Myarray             : Intptr;
            i, j, Temp2, Half   : Longint;
            Temp1, Temp3        : Extended;
            k                   : Integer;

        BEGIN
            Half := Ourimages[Num].ImageWidth DIV 2;
            FOR k:=3 DOWNTO 2 DO
                FOR j:=1 TO Ourimages[Num].ImageWidth DO
                    BEGIN
                        Temp2 := 2*(j - 1)*Ourimages[Num].ImageWidth;
                        FOR i := 1 TO Ourimages[Num].ImageWidth DO
                            BEGIN
                                Myarray := Intptr(Ord(Ourimages[Images-k].Data^) + 2*(i - 1) + Temp2);

                                IF (HExp <> 0) AND (VExp <> 0) THEN
                                    BEGIN
                                        Temp3 := ABS(i - HCenter)/HExp + ABS(j - VCenter)/VExp;
                                        Temp1 := Exp(-Temp3);
                                    END
                                ELSE Temp1 := 1.0;
                                Myarray^ := Round(Temp1*Myarray^);
                            END;
                    END;
        END;
```

```
{-------------------------------------------------------------------------}
{$S FFT2D}
{ This is the routine does the 2-D FFT (from Numerical Recipe). }

PROCEDURE FFT2D(Num: Integer; isign: Longint);

  VAR
    i, j, k, l, Half, Block, SF, SF1, max, min, NGAIN, ii1, ii2, ii3      : Longint;
    OffsetA, OffsetB, OS1, OS2, OS3, OS4, nprev, nrem, ntot               : Longint;
    i1, i2, i3, i2rev, i3rev, ibit, idim, ip1, ip2, ip3, ifp1, ifp2, k1, k2, n : Longint;
    tempi, tempr, theta, wi, wpi, wpr, wr, wtemp                          : Extended;
    Itemhit, Itemtype          : Integer;
    Box                        : Rect;
    Item                       : Handle;
    sometext, message, message1 : Str255;
    ReData2, ImData2           : Intptr;


  BEGIN
    {Arrange Re & Im into one Array for 2D_FT}

    Half := Ourimages[Num].ImageWidth DIV 2;
    Block := Ourimages[Num].ImageWidth*Half*2;

    FOR l:=0 TO 1 DO
      FOR k:=0 TO 1 DO
        FOR j:=1 TO Half DO
          BEGIN
            OS1 := (1 - k)*(Block + 2*(2*j - 1)*Half) + k*(4*(j - 1)*Half);
            OS2 := k*(Block + 2*(2*j - 1)*Half) + (1 - k)*(4*(j - 1)*Half);
            OS3 := (1 - k)*(Block + 4*(j - 1)*Half) + k*(2*(2*j - 1)*Half);
            OS4 := k*(Block + 4*(j - 1)*Half) + (1 - k)*(2*(2*j - 1)*Half);

            OffsetA := (1 - l)*OS1 + l*OS3;
            OffsetB := (1 - l)*OS2 + l*OS4;

            FOR i:=1 TO Half DO
              BEGIN
                ReData2 := Intptr(Ord(Ourimages[Images - 3].Data^) + 2*(i - 1) +
                                                                        OffsetA);
                Data2[2*i - 1 + OffsetB] := ReData2^;
                ImData2 := Intptr(Ord(Ourimages[Images - 2].Data^) + 2*(i - 1) +
                                                                        OffsetA);
                Data2[2*i + OffsetB] := ImData2^;
              END;
          END;

    {===== 2D_FT routine =====}

    ntot := 1;
    FOR idim := 1 TO ndimp DO
        ntot := ntot*Ourimages[Num].ImageWidth;
    nprev := 1;
    for idim := ndimp DOWNTO 1 DO
      BEGIN
        n := Ourimages[Num].ImageWidth;
        nrem := ntot DIV (n * nprev);
        ip1 := 2 * nprev;
        ip2 := ip1 * n;
        ip3 := ip2 * nrem;
        i2rev := 1;

        {this is the bit reversal part of the routine}
```

```
for ii2 := 0 TO ((ip2 - 1) DIV ip1) DO
   BEGIN
      i2 := 1 + ii2 * ip1;
      IF i2 < i2rev THEN
         BEGIN
            for ii1 := 0 TO ((ip1 - 2) DIV 2) DO
               BEGIN
                  i1 := i2 + ii1 * 2;
                  for ii3 := 0 TO ((ip3 - i1) DIV ip2) DO
                     BEGIN
                        i3 := i1 + ii3 * ip2;
                        i3rev := i2rev + i3 - i2;
                        tempr := Data2[i3];
                        tempi := Data2[i3 + 1];
                        Data2[i3] := Data2[i3rev];
                        Data2[i3 + 1] := Data2[i3rev + 1];
                        Data2[i3rev] := tempr;
                        Data2[i3rev + 1] := tempi;
                     END
               END
         END;
      ibit := ip2 DIV 2;
      WHILE ((ibit >= ip1) and (i2rev > ibit)) DO
         BEGIN
            i2rev := i2rev - ibit;
            ibit := ibit DIV 2;
         END;
      i2rev := i2rev + ibit
   END;

{Here begins the Danielson-Lanczos section of the routine}
ifp1 := ip1;
WHILE ifp1 < ip2 DO
   BEGIN
      ifp2 := 2 * ifp1;
      theta := isign*2*Pi/(ifp2 DIV ip1);
      wpr := -2.0 * sqr(sin(0.5 * theta));
      wpi := sin(theta);
      wr := 1.0;
      wi := 0.0;
      for ii3 := 0 TO ((ifp1 - 1) DIV ip1) DO
         BEGIN
            i3 := 1 + ii3 * ip1;
            for ii1 := 0 TO ((ip1 - 2) DIV 2) DO
               BEGIN
                  i1 := i3 + ii1 * 2;
                  for ii2 := 0 TO ((ip3 - i1) DIV ifp2) DO
                     BEGIN
                        i2 := i1 + ii2 * ifp2;
                        k1 := i2;
                        k2 := k1 + ifp1;
                        tempr := wr * Data2[k2] - wi * Data2[k2 + 1];
                        tempi := wr * Data2[k2 + 1] + wi * Data2[k2];
                        Data2[k2] := Data2[k1] - tempr;
                        Data2[k2 + 1] := Data2[k1 + 1] - tempi;
                        Data2[k1] := Data2[k1] + tempr;
                        Data2[k1 + 1] := Data2[k1 + 1] + tempi;
                     END
               END;
            wtemp := wr;
            wr := wr * wpr - wi * wpi + wr;
            wi := wi * wpr + wtemp * wpi + wi;
         END;
```

```
            ifp1 := ifp2
          END;
        nprev := n * nprev
    END;

{═══ 2D_FT routine finished ═══}

{check for data overflow}
SF := 1;
NGAIN := 0;
max := 32767;
min := -32767;

j := Ourimages[Num].ImageWidth*Ourimages[Num].ImageWidth;
FOR i := 1 TO j DO
    IF (Round(Data2[i]) > max) THEN max := Round(Data2[i])
        ELSE IF (Round(Data2[i]) < min) THEN min := Round(Data2[i]);

IF (max > 32767) OR (min < -32767) THEN
    BEGIN
        Mydialog := Getnewdialog(137, nil, Pointer( - 1));
        SetOKButton(MyDialog);

        IF (max > 32767) THEN
            BEGIN
                NumToString(max, sometext);
                message := Concat('[Maximum = ', sometext,']');
                GetDItem(mydialog, 6, itemType, item, box);
                SetIText(item, message);
            END
        ELSE IF (min < -32767) THEN
            BEGIN
                NumToString(min, message);
                message := Concat('[Minimum = ', sometext,']');
                GetDItem(mydialog, 6, itemType, item, box);
                SetIText(item, message);
            END;

        IF (max > ABS(min)) THEN     SF1 := max DIV 32767
            ELSE    SF1 := ABS(min) DIV 32767;
        NGAIN := Trunc(Ln(SF1)/0.69314718) +1;

        NumToString(NGAIN, sometext);
        message1 := Concat('2**',sometext,' ?');
        GetDItem(mydialog, 7, itemType, item, box);
        SetIText(item, message1);

        ModalDialog(nil, Itemhit);

        CASE Itemhit OF
            1:
                BEGIN
                    SF := 2**NGAIN;
                    Disposdialog(Mydialog);
                END;
            2:
                BEGIN
                    SF := 1;
                    NGAIN := 0;
                    Disposdialog(Mydialog);
                END;
        END;
    END;
```

```
        {Now ReArrange the output}
        FOR I := 0 TO 1 DO  {l=0 for top-left to bottom-right and bottom-right to top-left
                             l=1 for top-right to bottom-left and bottom-left to top-right}
           FOR k := 0 TO 1 DO        {k=0 for top to bottom; k=1 for bottom to top}
              FOR j := 1 TO Half DO
                 BEGIN
                    OS1 := (1 - k)*(Block + 2*(2*j - 1)*Half) + k*(4*(j - 1)*Half);
                    OS2 := k*(Block + 2*(2*j - 1)*Half) + (1 - k)*(4*(j - 1)*Half);
                    OS3 := (1 - k)*(Block + 4*(j - 1)*Half) + k*(2*(2*j - 1)*Half);
                    OS4 := k*(Block + 4*(j - 1)*Half) + (1 - k)*(2*(2*j - 1)*Half);

                    OffsetA := (1 - l)*OS1 + l*OS3;
                    OffsetB := (1 - l)*OS2 + l*OS4;

                    FOR i := 1 TO Half DO
                       BEGIN
                          ReData2:=Intptr(Ord(Ourimages[Images - 1].Data^) + 2*(i - 1) +
                                                                                OffsetA);
                          ReData2^ := Round(Data2[2*i - 1 + OffsetB]/SF);
                          ImData2 := Intptr(Ord(Ourimages[Images].Data^) + 2*(i - 1) + OffsetA);
                          ImData2^ := Round(Data2[2*i + OffsetB]/SF);
                       END;
                 END;

        {Now write the Ngain into the first pixel of each Re and Im images}
        ReData2 := Intptr(Ord(Ourimages[Images - 1].Data^));
        ReData2^ := NGAIN;
        ImData2 := Intptr(Ord(Ourimages[Images].Data^));
        ImData2^ := NGAIN;
     END;

{-------------------------------------------------------------------------}
   {$S FFT2D}
   { This is part of the 2-D FFT procedure. It sets up the windows etc. }

   PROCEDURE Prepare2D_FT(HCenter, HExp, VCenter, VExp, isign: Longint);

      VAR
         Num   : Integer;

      BEGIN
         Num := Getwrefcon(Frontwindow);
         Newpiximage('FI.R', Ourimages[Num].Imagesize);
         Newpiximage('FI.I', Ourimages[Num].Imagesize);

         ApFilter(HCenter,HExp,VCenter,VExp,Num);
         FFT2D(Num,isign);

         FindLimits(Images-1);
         Setourpalette(Images-1);
         Adjust(Images-1);
         Ourimages[Images].Baseline := Ourimages[Images-1].Baseline;
         Ourimages[Images].Upper := Ourimages[Images-1].Upper;
         Setourpalette(Images);
         Adjust(Images);
      END;

{-------------------------------------------------------------------------}
   {$S FFT2D}
   { This is the 2-D FFT calling routine. }

   PROCEDURE Do2D_FT;
```

```
VAR
   HCenter, HExp, VCenter, VExp, isign   : Longint;
   Itemhit, Itemtype                     : Integer;
   Box                                   : Rect;
   Item                                  : Handle;
   Sometext                              : Str255;

BEGIN
   Mydialog := Getnewdialog(160, nil, Pointer( - 1));
   SetOKButton(MyDialog);

   GetDItem(mydialog, 9, ItemType, Item, Box);
   SetCtlValue(ControlHandle(Item), 1);
   isign := 1;

   REPEAT
      ModalDialog(nil, Itemhit);

      CASE Itemhit OF
         9:
            BEGIN
               GetDItem(mydialog, 9, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 1);
               GetDItem(mydialog, 10, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
               isign := 1;
            END;
         10:
            BEGIN
               GetDItem(mydialog, 10, ItemType, Item, Box);
               SetCtlValue(ControlHandle(Item), 1);
               GetDItem(mydialog, 9, temType, Item, Box);
               SetCtlValue(ControlHandle(Item), 0);
               isign := -1;
            END;
         END;
   UNTIL (Itemhit=1) or (Itemhit=2);

   CASE Itemhit OF
      1:
         BEGIN
            Getditem(Mydialog, 4, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            StringToNum(Sometext, HCenter);
            Getditem(Mydialog, 6, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            StringToNum(Sometext, HExp);
            Getditem(Mydialog, 12, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            StringToNum(Sometext, VCenter);
            Getditem(Mydialog, 14, Itemtype, Item, Box);
            Getitext(Item, Sometext);
            StringToNum(Sometext, VExp);
            Disposdialog(Mydialog);

            Prepare2D_FT(HCenter, HExp, VCenter, VExp, isign);
         END;
      2: Disposdialog(MydiaIog);
   END;
END;

{-------------------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------------------}
{$S DynamicData}
{ This is the routine which reads the data images from the disk into the memory. }

PROCEDURE ReadFlow(FileName: Str255);

   VAR
      vRefNum, Refnum, I    : Integer;
      Logeof                : Longint;
      volName               : StringPtr;
      FName, sometext       : Str255;
   LABEL 123;

   BEGIN
      Err := GetVol(volName, vRefNum);

      FOR I := NI DOWNTO 1 DO
         BEGIN
            NumToString(I, sometext);

            Type2RData := True;
            FName := Concat(FileName, '.', sometext, '.R');
            Err := FSOpen(FName, vRefNum, Refnum);
            IF (Err <> noErr) AND (I = NI) THEN
               BEGIN
                  Sorry := True;
                  DoNotice(154);
                  GOTO 123;
               END
            ELSE IF (Err <> noErr) THEN
               BEGIN
                  Sorry := True;
                  DoNotice(151);
                  GOTO 123;
               END;
            Err := Setfpos(Refnum, Fsfromstart, 0);
            Err := Geteof(Refnum, Logeof);
            IF ((Logeof <> GoodArray3) AND (Logeof <> GoodArray2)) THEN
               BEGIN
                  DoNotice(155);
                  GOTO 123;
               END;
            CurrentArray := Logeof;
            Newpiximage(FName, Logeof);
            Err := FSRead(Refnum, Logeof, Ourimages[Images].Data^);
            Err := Fsclose(Refnum);
            Setourpalette(Images);
            Adjust(Images);
            Type2RData := False;

            Type2IData := True;
            NumToString(I, sometext);
            FName := Concat(FileName, '.', sometext, '.I');
            Err := FSOpen(FName, vRefNum, Refnum);
            IF Err <> noErr THEN
               BEGIN
                  Sorry := True;
                  DoNotice(151);
                  GOTO 123;
               END;
            Err := Setfpos(Refnum, Fsfromstart, 0);
            Err := Geteof(Refnum, Logeof);
            Newpiximage(FName, Logeof);
```

```
            Err := FSRead(Refnum, Logeof, Ourimages[Images].Data^);
            Err := Fsclose(Refnum);
            Setourpalette(Images);
            Adjust(Images);
            Type2IData := False;
         END;
      123:
   END;

{-------------------------------------------------------------------------}
{-------------------------------------------------------------------------}
   {$S Part2}
   { This is the routine which subtracts the noise power from the signal in Stejskal_Tanner. }

   PROCEDURE DynamicData(Place : Longint);

      VAR
         Sig                  : Array [1..MaxQSlice] OF Extended;
         ReDataPtr, ImDataPtr : Intptr;
         ReData, ImData       : Longint;
         I                    : Integer;

      BEGIN
         I := 0;
         REPEAT
            I := I+1;
            ReDataPtr := Intptr(Ord(Ourimages[2*NI - I*2].Data^) + Place);
            ReData := ReDataPtr^;
            ImDataPtr := Intptr(Ord(Ourimages[2*NI - I*2 + 1].Data^) + Place);
            ImData := ImDataPtr^;
            Sig[I] := ReData*ReData + ImData*ImData;
         UNTIL (Sig[I] - Noise[I] <= 1.0) OR (I = NI);

         IF (Sig[I] - Noise[I] > 1.0) THEN z := I ELSE    z := I - 1;
         FOR I := 1 TO z DO
            BEGIN
               Sig[I] := SQRT(Sig[I] - Noise[I]);
               NormSig[I] := (-1.0)*Ln(Sig[I]/Sig[1]);
            END;
      END;

{-------------------------------------------------------------------------}
   {$S Part2}
   { This is the routine which performs the linear regression in Stejskal_Tanner analysis. }

   PROCEDURE DoubleLR;

      VAR
         SX, SY, STX2, SS, SXOSS, TX, Weight      : Extended;
         SDeviation           : Array [1..MaxQSlice] OF Extended;
         I                    : Integer;

      BEGIN
         {Do initial linear regression to calculate the first fit without weighting}

         SX := 0;                                {Initial Linear regression sums}
         SY := 0;
         STX2 := 0;
         B := 0;

         FOR I := 1 TO z DO
            BEGIN
               SX := SX + KStep[I];
```

```
            SY := SY + NormSig[I];
         END;

      SXOSS := SX / z;

      FOR I := 1 TO z DO
         BEGIN
            TX := KStep[I] - SXOSS;
            STX2 := STX2 + TX * TX;
            B := B + TX * NormSig[I];
         END;

      B := B/STX2;
      A := (SY - SX*B)/z;

      {refit data using Yi=A+BXi to calculate standard diviation}

      SX := 0;
      SY := 0;
      SS := 0;
      STX2 := 0;
      B2 := 0;

      FOR I := 1 TO z DO
         BEGIN
            SDeviation[I] := Abs(NormSig[I] - (A + B*KStep[I]));
            Weight := 1/(SDeviation[I]*SDeviation[I]);
            SS := SS + Weight;
            SX := SX + KStep[I]*Weight;
            SY := SY + NormSig[I]*Weight;
         END;

      SXOSS := SX / SS;

      FOR I := 1 TO z DO
         BEGIN
            TX := (KStep[I] - SXOSS)/SDeviation[I];
            STX2 := STX2 + TX * TX;
            B2 := B2 + (TX * NormSig[I])/SDeviation[I];
         END;

      B2 := B2/STX2;
      A2 := (SY - SX*B2)/SS;
   END;

{-------------------------------------------------------------------------------}
   {$S Part2}
   { This routine prepares the parameters used in the Stejskal_Tanner analysis. }

   PROCEDURE QParameters(Dur, Sep, Grad, NumberPGSE: Longint);

      VAR
         I, j                    : Integer;
         GS, GM, K, RDur, RSep   : Extended;

      BEGIN
         IF (DFacter = -7) THEN Power := 1E10
            ELSE IF (DFacter = -8) THEN Power := 1E11
            ELSE IF (DFacter = -9) THEN Power := 1E12
            ELSE IF (DFacter = -10) THEN Power := 1E13
            ELSE IF (DFacter = -11) THEN Power := 1E14
            ELSE IF (DFacter = -12) THEN Power := 1E15
            ELSE IF (DFacter = -13) THEN Power := 1E16
```

```
        ELSE Power := 1E12;

      RSep := Sep/1000000.0;
      RDur := Dur/1000000.0;
      GM := Grad/1000.0;
      GS := GM/(MaxQSlice-1);
      K := (7.1824E16)*NumberPGSE*RDur*RDur*(RSep - RDur/3.0);
      KMax := K*GM*GM;
      FOR I := 1 TO NI DO    KStep[I] := K*((I - 1)*GS)*((I - 1)*GS);
   END;

{-------------------------------------------------------------------------------}
   {$S Part2}
   {Calculate the mean squared noise power for each pair of images using the first 128 pixels}

   PROCEDURE FindNoise;

      VAR
         ReDataPtr, ImDataPtr   : Intptr;
         ReData, ImData         : Longint;
         I, j                   : Integer;

      BEGIN
         FOR I := 1 TO NI DO
            BEGIN
               Noise[I] := 0.0;
               FOR j := 1 TO 128 DO
                  BEGIN
                     ReDataPtr := Intptr(Ord(Ourimages[2*NI - I*2].Data^) + 2*(j - 1));
                     ReData := ReDataPtr^;
                     ImDataPtr := Intptr(Ord(Ourimages[2*NI - I*2 + 1].Data^) + 2*(j - 1));
                     ImData := ImDataPtr^;
                     Noise[I] := Noise[I] + (ReData*ReData + ImData*ImData)/128.0;
                  END;
            END;
      END;

{-------------------------------------------------------------------------------}
   {$S Part2}
   { This routine constructs the diffusion image using the Stejskal_Tanner method. }

   PROCEDURE Stejskal_Tanner(Thresh: Longint; FileName: Str255);

      VAR
         DifPtr, DataPtr          : Intptr;
         DValue, Amp, Place       : Longint;
         I, j, Power1, Ball, BallC : Integer;
         FName, Outstr1           : Str255;
         Ourcursorh               : Curshandle;
      LABEL 30;

      BEGIN
         Ball := 128;
         BallC := 1;
         Power1 := Abs(DFacter) + 3;
         Numtostring(Power1, Outstr1);
         FName := Concat(FileName, '.D*1E', Outstr1, ' (LR)');
         Newpiximage(FName, CurrentArray);
         Ourcursorh := Getcursor(Ball);
         Setcursor(Ourcursorh^^);

         {Process data using the Stejskal Tanner method}
```

```
        Place := 0;
        REPEAT
            DataPtr := Intptr(Ord(Ourimages[2*NI - 2].Data^) + Place);      {test first real data}
            Amp := DataPtr^;
            IF (Amp > Thresh) THEN                    {Analyze data only if bigger than threshord}
                BEGIN
                    DynamicData(Place);
                    IF (z < 4) THEN GOTO 30;
                    DoubleLR;

                    DValue := Round(B2*Power/100)*100;
                    DifPtr := Intptr(Ord(Ourimages[2*NI].Data^) + Place);
                    DifPtr^ := DValue;
                END
            ELSE
30:             BEGIN
                    DValue := 0;
                    DifPtr := Intptr(Ord(Ourimages[2*NI].Data^) + Place);
                    DifPtr^ := DValue;
                END;

            BallC := BallC+1;
            IF (BallC = 50) THEN
                BEGIN
                    BallC := 1;
                    Ball := Ball + 1;
                    IF (Ball=132) THEN Ball := 128;
                END;
            Ourcursorh := Getcursor(Ball);
            Setcursor(Ourcursorh^^);

            Place := Place + 2;
        UNTIL Place >= CurrentArray;

        FindLimits(2*NI);
        Setourpalette(2*NI);
        Adjust(2*NI);
        InitCursor;
    END;

{-------------------------------------------------------------------------------}
    {$S Part2}
    { This is the FFT routine used in the analysis of Dynamic NMR Microscopy. }

    PROCEDURE FFT(Place: Longint);

        VAR
            Data                                : Array [1..2*MaxFFTNo] OF Longint;
            ReDataPtr, ImDataPtr                : Intptr;
            i, j, k, p, q, m, n, IStep, MMax, N2 : Integer;
            Arg, WTemp, WR, WI, WPR, WPI        : Extended;
            TReal, TImag                        : Longint;

        BEGIN
            FOR i := 1 TO NI DO                                {Read data}
                BEGIN
                    ReDataPtr := Intptr(Ord(Ourimages[2*NI - i*2].Data^) + Place);
                    Data[2*i - 1] := ReDataPtr^;
                    ImDataPtr := Intptr(Ord(Ourimages[2*NI - i*2 + 1].Data^) + Place);
                    Data[2*i] := ImDataPtr^;
                END;

            FOR i := 2*NI + 1 TO 2*FFTNo DO                    {zero filling the rest}
```

```
   Data[i] := 0;

n := 2*FFTNo;                                    {FFT routine}
i := 1;
FOR q :=1 TO FFTNo DO
   BEGIN
      k := 2*q - 1;
      IF (i > k) THEN
         BEGIN
            TReal := Data[i];
            TImag := Data[i + 1];
            Data[i] := Data[k];
            Data[i + 1] := Data[k + 1];
            Data[k] := TReal;
            Data[k + 1] := TImag;
         END;
      m := n DIV 2;
      WHILE ((m >= 2) AND (i > m)) DO
         BEGIN
            i := i - m;
            m := m DIV 2;
         END;
      i := i + m;
   END;

MMax := 2;

WHILE (n > MMAx) DO
   BEGIN
      IStep := 2*MMax;
      Arg := 6.283185/MMax;
      WPR := -2.0*SQR(Sin(0.5* Arg));
      WPI := Sin(Arg);
      WR := 1.0;
      WI := 0.0;

      FOR q := 1 TO (MMax DIV 2) DO
         BEGIN
            m := 2*q - 1;
            FOR p := 0 TO ((n - m) DIV IStep) DO
               BEGIN
                  k := m + p*IStep;
                  i := k + MMax;
                  TReal := Round(WR*Data[i] - WI*Data[i + 1]);
                  TImag := Round(WR*Data[i + 1] + WI*Data[i]);
                  Data[i] := Data[k] - TReal;
                  Data[i + 1] := Data[k + 1] - TImag;
                  Data[k] := Data[k] + TReal;
                  Data[k + 1] := Data[k + 1] + TImag;
               END;
            WTemp := WR;
            WR := WR*WPR - WI*WPI + WR;
            WI := WI*WPR + WTemp*WPI + WI;
         END;
      MMax := IStep;
   END;

{Swap the data into -FFTNo/2 to FFTNo/2-1 format}

N2 := FFTNo DIV 2;
FOR i := 1 TO N2 DO    QSpectrum[N2 - i + 1] := Data[2*i - 1];
FOR i := 1 TO N2-1 DO      QSpectrum[N2 + i] := Data[2*(FFTNo - i) + 1];
QSpectrum[0] := Data[FFTNo + 1];
```

```
    END;

{-------------------------------------------------------------------------}
  {$S Part2}
  { Search for peak and calculate FWHM. }

  PROCEDURE Searching;

    VAR
      Base                : Extended;
      i, LeftP, RightP, j : Integer;
      Guess               : Longint;
    LABEL 31;

    BEGIN
      Guess := -60000;                                  {search for peak}
      FOR i := 0 TO FFTNo - 1 DO
        BEGIN
          IF (QSpectrum[i] > Guess) THEN
            BEGIN
              Guess := QSpectrum[i];
              Vel := i;
            END;
        END;

      Base := 0.0;                                      {calculate true half value}
      IF (Vel >= FFTNo/3) AND (Vel <= 2*FFTNo/3) THEN
        BEGIN
          IF (FFTNo = 256) THEN j := 32
            ELSE IF (FFTNo = 512) THEN j := 64
            ELSE IF (FFTNo = 1024) THEN j := 128;
          FOR i := 0 TO j-1 DO    Base := Base + QSpectrum[i]/(2*j);
          FOR i := (FFTNo - 1) DOWNTO (FFTNo - j - 1) DO
                                  Base := Base + QSpectrum[i]/(2*j);
        END
      ELSE IF (Vel < FFTNo/3) THEN
        BEGIN
          IF (FFTNo = 256) THEN
            FOR i := (FFTNo-1) DOWNTO (FFTNo-65) DO
                                              Base := Base + QSpectrum[i]/64
          ELSE IF (FFTNo = 512) THEN
            FOR i := (FFTNo-1) DOWNTO (FFTNo-129) DO
                                              Base := Base + QSpectrum[i]/128
          ELSE IF (FFTNo = 1024) THEN
            FOR i := (FFTNo-1) DOWNTO (FFTNo-257) DO
                                              Base := Base + QSpectrum[i]/256;
        END
      ELSE IF (Vel > 2*FFTNo/3) THEN
        BEGIN
          IF (FFTNo = 256) THEN
            FOR i := 0 TO 63 DO    Base := Base + QSpectrum[i]/64
          ELSE IF (FFTNo = 512) THEN
            FOR i := 0 TO 127 DO   Base := Base + QSpectrum[i]/128
          ELSE IF (FFTNo = 1024) THEN
            FOR i := 0 TO 255 DO   Base := Base + QSpectrum[i]/256;
        END;

      Guess := Round((Guess - Base)/2 + Base);

      i := Vel;                                         {calculate the FWHM}
      REPEAT
        i := i - 1;
```

```
            IF (i <= 0) THEN
               BEGIN
                  LeftP := 0;
                  GOTO 31;
               END;
            UNTIL (QSpectrum[i] <= Guess);
            LeftP := i;

            i := Vel;
            REPEAT
               i := i + 1;
               IF (i >= FFTNo - 1) THEN
                  BEGIN
                     RightP := FFTNo - 1;
                     GOTO 31;
                  END;
            UNTIL (QSpectrum[i] <= Guess);
            IF (QSpectrum[i] < Guess) THEN
               RightP := i - 1
            ELSE IF (QSpectrum[i] = Guess) THEN RightP := i;

31:         FWHM := RightP - LeftP;
            Vel := Vel - (FFTNo DIV 2);
         END;

{-------------------------------------------------------------------------------}
   {$S Part2}
   { This routine constructs the velocity and FWHM maps. }

   PROCEDURE FFTMaping(Thresh: Longint; FileName: Str255);

      VAR
         DataPtr, DifPtr, VelPtr    : Intptr;
         Place, Amp                 : Longint;
         I, j, Power1, Ball, BallC  : Integer;
         FName, Outstr1             : Str255;
         Ourcursorh                 : Curshandle;

      BEGIN
         FName := Concat(FileName, '.FWHM');
         Newpiximage(FName, CurrentArray);
         FName := Concat(FileName, '.V (FFT)');
         Newpiximage(FName, CurrentArray);
         Ball := 128;
         BallC := 1;
         Ourcursorh := Getcursor(Ball);
         Setcursor(Ourcursorh^^);

         Place := 0;
         REPEAT
            DataPtr := Intptr(Ord(Ourimages[2*NI - 2].Data^) + Place);      {test first real data}
            Amp := DataPtr^;
            IF (Amp > Thresh) THEN                  {Analyze data only if bigger than threshord}
               BEGIN
                  FFT(Place);
                  Searching;

                  DifPtr := Intptr(Ord(Ourimages[2*NI].Data^) + Place);
                  DifPtr^ := FWHM;
                  VelPtr := Intptr(Ord(Ourimages[2*NI + 1].Data^) + Place);
                  VelPtr^ := Vel;

                  Ball := Ball + 1;
```

```
            IF (Ball = 132) THEN Ball := 128;
            Ourcursorh := Getcursor(Ball);
            Setcursor(Ourcursorh^^);
         END
      ELSE
      BEGIN
            DifPtr := Intptr(Ord(Ourimages[2*NI].Data^) + Place);
            DifPtr^ := 0;
            VelPtr := Intptr(Ord(Ourimages[2*NI + 1].Data^) + Place);
            VelPtr^ := 0;
         END;
      Place := Place + 2;
   UNTIL Place >= CurrentArray;

   FindLimits(2*NI);
   Setourpalette(2*NI);
   Adjust(2*NI);
   FindLimits(2*NI + 1);
   Setourpalette(2*NI + 1);
   Adjust(2*NI + 1);
   InitCursor;
END;

-------------------------------------------------------------------------------------------------------}
{$S Part2}
{ This is the user interfacing routine in the velocity and diffusion analysis. }

PROCEDURE DoDynamicAnalysis(SelMenu: Longint);

   VAR
      Itemhit, Itemtype                : Integer;
      Sometext, FileName               : Str255;
      TH, Sep, Dur, Grad, NumberPGSE   : Longint;
      Box                              : Rect;
      Item                             : Handle;
   LABEL 121;

   BEGIN
      Mydialog := Getnewdialog(150, nil, Pointer( -1 ));
      SetOKButton(MyDialog);

      Modaldialog(nil, Itemhit);
      CASE Itemhit OF
         1:
            BEGIN
               Getditem(Mydialog, 7, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, NI);

               Getditem(Mydialog, 9, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, TH);

               Getditem(Mydialog, 11, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, Dur);

               Getditem(Mydialog, 13, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, Sep);

               Getditem(Mydialog, 15, Itemtype, Item, Box);
               Getitext(Item, Sometext);
```

```
                Stringtonum(Sometext, Grad);

                Getditem(Mydialog, 20, Itemtype, Item, Box);
                Getitext(Item, FileName);

                Getditem(Mydialog, 22, Itemtype, Item, Box);
                Getitext(Item, Sometext);
                Stringtonum(Sometext, NumberPGSE);

                Getditem(Mydialog, 24, Itemtype, Item, Box);
                Getitext(Item, Sometext);
                Stringtonum(Sometext, DFacter);

                Getditem(Mydialog, 27, Itemtype, Item, Box);
                Getitext(Item, Sometext);
                Stringtonum(Sometext, FFTNo);

                Disposdialog(Mydialog);

                IF (NI > 18) THEN
                   BEGIN
                      DoNotice(147);
                      Quit := True;
                      GOTO 121;
                   END;

                ReadFlow(FileName);
                IF NOT Sorry THEN
                   BEGIN
                      Enablemenus;
                      Drawmenubar;
                      QParameters(Dur, Sep, Grad, NumberPGSE);
                      FindNoise;
                      IF (SelMenu = 1) THEN
                            Stejskal_Tanner(TH, FileName)
                      ELSE IF (SelMenu = 2) THEN
                            FFTMaping(TH, FileName);
                   END
                ELSE Quit := True;
121:        END;
            2: Disposdialog(Mydialog);
         END;
      END;

{-------------------------------------------------------------------------}
   {$S Part2}
   { This is the routine which updates the menu for the v&d plot display. }

   PROCEDURE SetQMenu;

      VAR
         Num   : Integer;

      BEGIN
         IF (Frontwindow <> nil) THEN
            BEGIN
               Num := Getwrefcon(Frontwindow);
               IF QMode THEN
                  Setitem(Myhandle[Displaymenu], 8, 'Hide q Plots ')
               ELSE
                  Setitem(Myhandle[Displaymenu], 8, 'Show q Plots ');
            END; {IF frontwindow}
      END;
```

```
{-------------------------------------------------------------------------------}
  {$S Part2}
  { This is the routine which displays the FFT and Stejskal_Tanner plots on the screen. }

  PROCEDURE ShowQPlots(Num: Integer; X, Y, Xshift, Yshift: Longint);

    VAR
      Xscaled, Yscaled, TheValue, Place, Dim  : Longint;
      I, J, Power1, y1, Inc                   : Integer;
      Outstr1, Outstr2, Outstr                : Str255;

    BEGIN
      IF ((Ourimages[Num].ImageWidth = 64) OR (Ourimages[Num].ImageWidth = 128))
        THEN
        BEGIN
          Power1 := Abs(DFacter) + 3;
          Dim := Ourimages[Num].ImageWidth;
          Xscaled := Round(Dim * X / (Ourimages[Num].Dispdimension)) + Xshift;
          Yscaled := Round(Dim * Y / (Ourimages[Num].Dispdimension)) + Yshift;
          Place := 2*(Dim * Yscaled + Xscaled);

          Setport(Mycurwindow);
          Textfont(21);
          Textface([Bold]);
          Moveto(100, 10);
          Drawstring('Cursor Position');
          Moveto(10, 48);
          Drawstring('Stejskal-Tanner Plot');
          Moveto(10, 285);
          Numtostring(FFTNo,Outstr1);
          Outstr := Concat('Fourier Transform (', Outstr1, ') plot');
          Drawstring(Outstr);
          Pensize(2, 2);
          Moveto(0, 33);
          Lineto(300, 33);
          Moveto(0, 270);
          Lineto(300, 270);

          Pennormal;
          Textmode(Srccopy);
          Moveto(10, 25);
          Drawstring('                                    ');
          Drawstring('                                    ');
          Moveto(10, 62);
          Drawstring('                                        ');
          Drawstring('                                        ');
          Moveto(10, 300);
          Drawstring('                                        ');
          Moveto(10, 315);
          Drawstring('                                        ');

          Textfont(1);
          Textface([]);
          Textmode(Srcor);

          Moveto(10, 25);
          IF (Xscaled >= 0) AND (Xscaled <= Dim-1) AND (Yscaled >= 0)
                        AND (Yscaled <= Dim-1) THEN     {Cursor is inside the image}
            BEGIN
              IF (FFTNo = 256) THEN Inc := 1
                ELSE IF (FFTNo = 512) THEN Inc := 2
                ELSE IF (FFTNo = 1024) THEN Inc := 4;
```

```
Numtostring(Xscaled, Outstr1);
Numtostring(Yscaled, Outstr2);
Outstr := Concat(' ( x , y ) = (',Outstr1, ', ', Outstr2, ')');
Drawstring(Outstr);

Moveto(30, 90);
Lineto(30, 250);
Moveto(30, 90);
Lineto(280, 90);
Moveto(27, 90+80);
Lineto(33, 90+80);
Moveto(27, 90+160);
Lineto(33, 90+160);
Moveto(280, 87);
Lineto(280,93);

FOR J := 0 TO 1 DO
   BEGIN
      Forecolor(409 - J*204);
      Moveto(30 + J*129, 430);
      Lineto(157 + J*129,430);
      FOR I := 0 TO 3 DO
         BEGIN
            Moveto(30 + I*32 + J*160, 425);
            Lineto(30 + I*32 + J*160, 435);
         END;
      FOR I := 0 TO 15 DO
         BEGIN
            Moveto(30 + I*8 + J*136, 428);
            Lineto(30 + I*8 + J*136, 432);
         END;
   END;
Forecolor(blackColor);
Moveto(158, 425);
Lineto(158, 435);

IF NOT Firstdraw THEN          {wipe the old plots if not the first time}
   BEGIN
      DrawMyCross(white, 30, 90, 1);
      Moveto(30, Round(160*A2/4.605) + 90);
      Lineto(250 + 30, Round(160*(A2 + B2*KMax)/4.605) + 90);
      I := 0;
      J := 0;
      REPEAT
         y1 := Round(QSpectrum[I] / 1500);
         Moveto(J + 30, 420 - y1);
         Line(0, 0);
         I := I + Inc;
         J := J + 1;
      UNTIL (I >= FFTNo - 1);
   END;

DynamicData(Place);                    {draw LR plot}
DoubleLR;
FOR I := 1 TO z DO
   BEGIN
      CX[I] := Round(250*KStep[I]/KMax);
      CY[I] := Round(160*NormSig[I]/4.605);
   END;
DrawMyCross(black, 30, 90, 1);
Moveto(30, Round(160*A2/4.605) + 90);
Lineto(250+30, Round(160*(A2 + B2*KMax)/4.605) + 90);
```

```
                    TheValue := Round(B2*Power/100)*100;
                    Numtostring(TheValue, Outstr1);
                    Numtostring(Power1, Outstr2);
                    Outstr := Concat('Diffusion*1E', Outstr2, ' = ', Outstr1);
                    Moveto(10, 62);
                    Drawstring(Outstr);

                    FFT(Place);                          {draw FFT plot}
                    Searching;

                    I := 0;
                    J := 0;
                    REPEAT
                       y1 := Round(QSpectrum[I] / 1500);
                       Moveto(J + 30, 420 - y1);
                       Line(0, 0);
                       I := I + Inc;
                       J := J + 1;
                    UNTIL (I >= FFTNo - 1);

                    Numtostring(FWHM, Outstr1);
                    Numtostring(Vel, Outstr2);
                    Outstr := Concat('Vel.(k) = ', Outstr2);
                    Moveto(10, 300);
                    Drawstring(Outstr);
                    Outstr := Concat('FWHM = ', Outstr1);
                    Moveto(10, 315);
                    Drawstring(Outstr);

                    Firstdraw := False;
                 END
              ELSE    {Cursor is out of image}
                 Drawstring('Move towards image!');
           END
        ELSE DoNotice(155);
     END;

{-------------------------------------------------------------------------------}
   {$S Part2}
   { This is the routine which is used in the 'one-shot' velocity microscopy. }

   PROCEDURE SingleQ(Dur, Sep, Grad, Thresh: Longint);

      VAR
         Num                                  : Integer;
         Place, SinP, CosP, CalP, Factor      : Longint;
         Myarray1, Myarray2, Myarray3, Myarray : Intptr;
         Temp1, Temp2, ArSin, ArCos, Final    : Extended;
         Gradient, RDur, RSep, Proton, K      : Extended;
         Qwrecord                             : Windowrecord;
         Windowsize                           : Rect;
         MyTwindow                            : Windowptr;
         Outstr, Outstr1                      : Str255;
         Ourcursorh                           : Curshandle;
      LABEL 21;

      BEGIN
         Setrect(Windowsize, 150, 356, 498, 370);
         NoOpen := False;

         MyTwindow := Newcwindow(@Qwrecord, Windowsize, '', True, Dboxproc, Pointer(-1),
                                                                        True, 0);
         Setport(MyTwindow);
```

```
ForeColor(redColor);
Moveto(0, 10);
DrawString('Open the calibration image under zero-flow');
Doarray;
Disposewindow(MyTwindow);
IF NoOpen THEN GOTO 21;

MyTwindow := Newcwindow(@Qwrecord, Windowsize, '', True, Dboxproc, Pointer(-1),
                                                                    True, 0);
Setport(MyTwindow);
ForeColor(redColor);
Moveto(0, 10);
DrawString('Open the velocity coding image under flow [Sin Image]');
Doarray;
Disposewindow(MyTwindow);
IF NoOpen THEN GOTO 21;

MyTwindow := Newcwindow(@Qwrecord, Windowsize, '', True, Dboxproc, Pointer(-1),
                                                                    True, 0);
Setport(MyTwindow);
ForeColor(redColor);
Moveto(0, 10);
DrawString('Open the calibration image under flow [Cos Image]');
Doarray;
Disposewindow(MyTwindow);
IF NoOpen THEN GOTO 21;

Ourcursorh := Getcursor(4);
Setcursor(Ourcursorh^^);

RSep := Sep/1000000.0;
RDur := Dur/1000000.0;
Gradient := Grad/1000.0;
Proton := 2.68E8;
K := Proton*RDur*RSep*Gradient;
Num := Getwrefcon(Frontwindow);

Factor := Trunc(1000*1000*2*Pi/K);
IF (Factor < 10) THEN    Factor := 1000*1000
 ELSE IF ((Factor >= 10) AND (Factor < 100)) THEN     Factor := 100*1000
 ELSE IF ((Factor >= 100) AND (Factor < 1000)) THEN  Factor := 10*1000
 ELSE IF ((Factor >= 1000) AND (Factor < 10000)) THEN   Factor := 1000
 ELSE IF ((Factor >= 10000) AND (Factor < 100000)) THEN    Factor := 100
 ELSE IF ((Factor >= 100000) AND (Factor < 1000000)) THEN      Factor := 10
 ELSE IF (Factor >= 1000000) THEN     Factor := 1;

NumToString(Factor, Outstr1);
Outstr := Concat(Outstr1, '*V [mm/s]');
Newpiximage(Outstr, Ourimages[Num].Imagesize);

Place := 0;
REPEAT
   Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
   CosP := Myarray1^;
   Myarray2 := Intptr(Ord(Ourimages[Num-1].Data^) + Place);
   SinP := Myarray2^;
   Myarray3 := Intptr(Ord(Ourimages[Num-2].Data^) + Place);
   CalP := Myarray3^;

   IF (CalP >= Thresh) THEN
      BEGIN
         Temp1 := SinP/CalP;
         (*Temp2 := CosP/CalP;*)
```

```
                IF (Temp1 > 1)       THEN Temp1 := 1
                   ELSE IF (Temp1 < -1) THEN Temp1 := -1
                   (*ELSE IF (Temp2 > 1)  THEN Temp2 := 1
                   ELSE IF (Temp2 < -1) THEN Temp2 := -1*);

                ArSin := Arctan(Temp1/Sqrt(1-Temp1*Temp1));     {Arc Sin}

                IF (SinP >= 0) AND (CosP >= 0) THEN          Final := ArSin
                   ELSE IF (SinP >= 0) AND (CosP < 0) THEN   Final := Pi - ArSin
                   ELSE IF (SinP < 0) AND (CosP < 0) THEN    Final := Pi - ArSin
                   ELSE IF (SinP < 0) AND (CosP >= 0) THEN   Final := 2*Pi + ArSin;

             (* ArCos := Arctan(Sqrt(1-Temp2*Temp2)/Temp2);    {Arc Cos}

                IF (SinP >= 0) AND (CosP >= 0) AND (ArSin > 0) AND (ArSin <= Pi/4)
                                          THEN   Final := ArSin
                ELSE IF (SinP >= 0) AND (CosP >= 0) AND (ArSin > Pi/4) AND
                                          (ArSin <= Pi/2)  THEN Final := ArCos
                ELSE IF (SinP >= 0) AND (CosP < 0) AND (ArSin > 0) AND
                                          (ArSin <= Pi/4)  THEN Final := Pi + ArCos
                ELSE IF (SinP >= 0) AND (CosP < 0) AND (ArSin > Pi/4) AND
                                          (ArSin <= Pi/2)  THEN Final := Pi - ArSin
                ELSE IF (SinP < 0) AND (CosP < 0) AND (ArSin < 0) AND
                                          (ArSin >= -Pi/4) THEN Final := Pi - ArSin
                ELSE IF (SinP < 0) AND (CosP < 0) AND (ArSin <= -Pi/4) AND
                                          (ArSin >= -Pi/2) THEN Final := Pi - ArCos
                ELSE IF (SinP < 0) AND (CosP >= 0) AND (ArSin < 0) AND
                                          (ArSin >= -Pi/4) THEN Final := 2*Pi - ArCos
                ELSE IF (SinP < 0) AND (CosP >= 0) AND (ArSin <= -Pi/4) AND
                                          (ArSin >= -Pi/2) THEN Final := 2*Pi + ArSin;*)
                END
             ELSE   Final := 0;

             Myarray := Intptr(Ord(Ourimages[Images].Data^) + Place);
             Myarray^ := Round(1000*Factor*Final/K);
             Place := Place + 2;
          UNTIL Place >= Ourimages[Num].Imagesize;

          FindLimits(Images);
          Setourpalette(Images);
          Adjust(Images);
21:       NoOpen := False;
          Initcursor;
       END;

{-------------------------------------------------------------------------------------------------}
   {$S Part2}
   { This is the calling routine in the 'one-shot' velocity microscopy. }

   PROCEDURE DoSingleQ;

      VAR
         Sometext                : Str255;
         Box                     : Rect;
         Item                    : Handle;
         Itemhit, Itemtype       : Integer;
         Thresh, Sep, Dur, Grad  : Longint;

      BEGIN
         Mydialog := Getnewdialog(158, nil, Pointer( - 1));
         SetOKButton(MyDialog);
         ModalDialog(nil, Itemhit);
```

```
      CASE Itemhit OF
         1:
            BEGIN
               Getditem(Mydialog, 5, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, Thresh);

               Getditem(Mydialog, 7, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, Dur);

               Getditem(Mydialog, 9, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, Sep);

               Getditem(Mydialog, 11, Itemtype, Item, Box);
               Getitext(Item, Sometext);
               Stringtonum(Sometext, Grad);

               Disposdialog(Mydialog);

               SingleQ(Dur, Sep, Grad, Thresh);
            END;
         2: Disposdialog(Mydialog);
      END;
   END;

{-------------------------------------------------------------------------}
{-------------------------------------------------------------------------}
   {$S Part3}
   { This is part of the printing procedure. }

   PROCEDURE Prdrawingproc(Prwindow: Integer);

      VAR
         V, H, Pixvalue   : Integer;
         Incr, last       : Longint;
         Dummyptr         : Ptr;
         R                : Rect;

      BEGIN
         WITH Ourimages[Prwindow].Pixmap^^ DO
            BEGIN
            Incr := 0;
            last := Ourimages[Prwindow].Imagesize DIV 2;
            IF Ourimages[Prwindow].Imagesize = GoodArray3 THEN
                  PenSize(4, 4)
               ELSE IF Ourimages[Prwindow].Imagesize = GoodArray2 THEN
                  PenSize(2, 2)
               ELSE IF Ourimages[Prwindow].Imagesize = GoodArray1 THEN
                  PenSize(1, 1);

            REPEAT
               Dummyptr := Ptr(Ord(Baseaddr) + Incr);
               Pixvalue := Dummyptr^;
               IF (Pixvalue > - 1) AND (Pixvalue < 4) THEN Forecolor(Blackcolor);
               IF (Pixvalue > 3) AND (Pixvalue < 8) THEN Forecolor(Redcolor);
               IF (Pixvalue > 7) AND (Pixvalue < 12) THEN Forecolor(Yellowcolor);
               IF (Pixvalue > 11) AND (Pixvalue < 16) THEN Forecolor(Greencolor);
               IF (Pixvalue > 15) AND (Pixvalue < 20) THEN Forecolor(Cyancolor);
               IF (Pixvalue > 19) AND (Pixvalue < 24) THEN Forecolor(Bluecolor);
               IF (Pixvalue > 23) AND (Pixvalue < 28) THEN Forecolor(Bluecolor);
               IF (Pixvalue > 27) AND (Pixvalue < 32) THEN Forecolor(Whitecolor);
```

```
            IF Ourimages[Prwindow].Imagesize = GoodArray3 THEN
               BEGIN
                  V := Incr DIV 64;
                  H := Incr MOD 64;
               END
            ELSE IF Ourimages[Prwindow].Imagesize = GoodArray2 THEN
               BEGIN
                  V := Incr DIV 128;
                  H := Incr MOD 128;
               END
            ELSE IF Ourimages[Prwindow].Imagesize = GoodArray1 THEN
               BEGIN
                  V := Incr DIV 256;
                  H := Incr MOD 256;
               END;
            Moveto(H, V);
            Line(0, 0);

            Incr := Incr + 1;
         UNTIL (Incr = last);

         PenNormal;
         Forecolor(Blackcolor);
         IF Ourimages[Prwindow].Imagesize = GoodArray3 THEN
               Setrect(R, 0, 0, 64, 64)
            ELSE IF Ourimages[Prwindow].Imagesize = GoodArray2 THEN
               Setrect(R, 0, 0, 128, 128)
            ELSE IF Ourimages[Prwindow].Imagesize = GoodArray1 THEN
               Setrect(R, 0, 0, 256, 256);
         Framerect(R);
      END;
   END;

{-------------------------------------------------------------------------}
   {$S Part3}
   { This is the routine printing the slice plot. }

   PROCEDURE PrPlot(X, Y, Prwindow, scbase: Integer;Scfac,Yshift: Longint; vector: Boolean);

      VAR
         Ampl, Place, Scaledampl, Temp, PreviousAmpl, Yscaled, Dim, Max, Min: Longint;
         i, Space          : Integer;
         Datptr            : Intptr;
         R                 : Rect;
         Outstr1, Outstr2  : Str255;
         Done              : Boolean;

      BEGIN
         Dim := Ourimages[Prwindow].ImageWidth;
         Yscaled := Round(Dim*Y/(Ourimages[Prwindow].Dispdimension)) + Yshift;
         Numtostring(Yscaled, Outstr1);
         GetWTitle(Ourimages[Prwindow].window, Outstr2);

         Textfont(21);
         Textsize(14);
         MoveTo(20, 30);
         Drawstring('A slice across the Image " ');
         Drawstring(Outstr2);
         Drawstring(' " at Y = ');
         Drawstring(Outstr1);
         Textfont(0);
         Textsize(0);
```

```
PenNormal;

Max := 0;
Min := 0;
Place := 2*Dim*Yscaled;
IF (Ourimages[Prwindow].Imagesize = GoodArray1) THEN                Space := 2
   ELSE IF (Ourimages[Prwindow].Imagesize = GoodArray2) THEN        Space := 4
   ELSE IF (Ourimages[Prwindow].Imagesize = GoodArray3) THEN
      BEGIN
         Space := 8;
         IF (vector) THEN
            BEGIN
               Pensize(3, 3);
               ForeColor(redColor);
            END;
      END;

FOR i := 0 TO Dim-1 DO
   BEGIN
      Temp := 2*i;      {offset}
      datptr := intptr(ord(ourImages[Prwindow].data^) + place + Temp);
      Ampl := datptr^;
      IF (Ampl > Max) THEN Max := Ampl
         ELSE IF (Ampl < Min) THEN Min := Ampl;
      ScaledAmpl:=(Ampl*Scfac)DIV(10000);
      Temp := scbase*3 + 25 - ScaledAmpl;     {Y}
      MoveTo(30 + i*Space, Temp);
      Line(0, 0);

      IF ((Dim = 256) AND (vector) AND (i <> 0)) OR
                  ((Dim = 128) AND (vector) AND (i <> 0)) THEN
         BEGIN
            MoveTo(30 + (i - 1)*Space, scbase*3 + 25 - PreviousAmpl);
            LineTo(30 + i*Space,Temp);
         END
      ELSE IF (Dim = 64) AND (vector) AND (i <> 0) THEN
         BEGIN
            Pensize(1, 1);
            ForeColor(blackColor);
            MoveTo(31 + Space*(i - 1), scbase*3 + 26 - PreviousAmpl);
            LineTo(31 + Space*i, Temp + 1);
            Pensize(3, 3);
            ForeColor(redColor);
         END;

      PreviousAmpl := ScaledAmpl;
   END;

PenNormal;
Forecolor(blackcolor);
Setrect(R, 0, 0, 560, 300);
Framerect(R);

MoveTo(0, scbase*3 + 25);
LineTo(3, scbase*3 + 25);
MoveTo(5, scbase*3 + 29);
Drawstring('0');

Ampl := Trunc(Max);
ScaledAmpl := (Ampl*Scfac) DIV (10000);
Done := False;
REPEAT
   Temp := scbase*3 + 25 - ScaledAmpl;
```

```
            IF (Temp > 50) THEN
               BEGIN
                  MoveTo(0, Temp);
                  LineTo(3, Temp);
                  MoveTo(5, Temp + 5);
                  NumToString(Ampl, OutStr1);
                  Drawstring(OutStr1);
                  Done := True;
               END;
            Ampl := Ampl - 1;
            ScaledAmpl:=(Ampl * Scfac) DIV (10000);
         UNTIL Done = True;

         Ampl := Trunc(Min);
         ScaledAmpl := (Ampl * Scfac) DIV (10000);
         Temp := scbase*3 + 25 - ScaledAmpl;
         IF (Temp < 295) THEN
            BEGIN
               MoveTo(0, Temp);
               LineTo(3, Temp);
               MoveTo(5, Temp + 5);
               NumToString(Ampl, OutStr1);
               Drawstring(OutStr1);
            END;
      END;

{-------------------------------------------------------------------------------}
   {$S Part3}
   { This is the routine printing the FFT and linear regression plots. }

   PROCEDURE PrQPlot(X, Y, Prwindow: Integer; Xshift, Yshift: Longint);

      VAR
         Outstr1, Outstr2, Outstr              : Str255;
         Xscaled, Yscaled, Place, Dim          : Longint;
         DataPtr                               : Intptr;
         TheValue, I, J, Power1, y1, IncI, IncJ : Integer;

      BEGIN
         Pennormal;
         Textsize(0);
         GetWTitle(Ourimages[Prwindow].window, Outstr1);

         Dim := Ourimages[Prwindow].ImageWidth;
         Xscaled := Round(Dim * X / (Ourimages[Prwindow].Dispdimension)) + Xshift;
         Yscaled := Round(Dim * Y / (Ourimages[Prwindow].Dispdimension)) + Yshift;
         Power1 := Abs(DFacter) + 3;
         TheValue := Round(B2*Power/100) * 100;

         IF (FFTNo = 256) THEN
            BEGIN
               IncI := 1;
               IncJ := 2;
            END
         ELSE IF (FFTNo = 512) THEN
            BEGIN
               IncI := 1;
               IncJ := 1;
            END
         ELSE IF (FFTNo = 1024) THEN
            BEGIN
               IncI := 2;
               IncJ := 1;
```

```
    END;

TextFace([underline]);
MoveTo(5, 10);
Drawstring('Image Name:');
MoveTo(56, 25);
Drawstring('Pixel:');
MoveTo(10, 50);
Drawstring('Stejskal-Tanner plot:');
MoveTo(10, 460);
Numtostring(FFTNo, Outstr2);
Outstr := Concat('Fourier Transform (', Outstr2, ') plot:');
Drawstring(Outstr);
TextFace([]);
MoveTo(105, 10);
Drawstring(Outstr1);
MoveTo(105, 25);
Numtostring(Xscaled, Outstr1);
Numtostring(Yscaled, Outstr2);
Outstr := Concat('(x,y) = (', Outstr1, ',', Outstr2,')');
Drawstring(Outstr);

Moveto(30, 80);
Lineto(30, 400);
Moveto(30, 80);
Lineto(530, 80);
Moveto(27,240);
Lineto(33, 240);
Moveto(27,400);
Lineto(33, 400);
Moveto(530, 77);
Lineto(530, 83);
MoveTo(13, 83);
Drawstring('0');
MoveTo(6, 244);
Drawstring('0.1');
MoveTo(1, 404);
Drawstring('0.01');
MoveTo(455, 72);
Drawstring('(');
Textfont(23);
Drawstring('gδ');
Textfont(0);
Drawstring('g)^2(Δ-');
Textfont(23);
Drawstring('δ');
Textfont(0);
Drawstring('/3)');
MoveTo(0,420);
Drawstring('ln[Sig(i)/Sig(1)]');

Moveto(20, 650);
Lineto(532, 650);
FOR I := 0 TO 8 DO
    BEGIN
        Moveto(20 + I*64, 645);
        Lineto(20 + I*64, 655);
    END;
FOR I := 0 TO 32 DO
    BEGIN
        Moveto(20 + I*16, 648);
        Lineto(20 + I*16, 652);
    END;
```

```
        MoveTo(0, 670);
        Numtostring(-FFTNo DIV 2, Outstr1);
        Numtostring((FFTNo - 2) DIV 2, Outstr2);
        Drawstring(Outstr1);
        MoveTo(272, 670);
        Drawstring('0');
        MoveTo(521, 670);
        Drawstring(Outstr2);

        MoveTo(160, 50);
        Numtostring(TheValue, Outstr1);
        Numtostring(Power1, Outstr2);
        Outstr := Concat(' Diffusion*1E', Outstr2, ' = ', Outstr1);
        Drawstring(Outstr);
        Moveto(30, 2*Round(160*A2/4.605) + 80);
        Lineto(2*250 + 30, 2*Round(160*(A2 + B2*KMax)/4.605) + 80);
        DrawMyCross(black, 30, 80, 2);

        MoveTo(230, 460);
        Numtostring(Vel, Outstr2);
        Numtostring(FWHM, Outstr1);
        Outstr := Concat('Velocity = ', Outstr2, ' ;  FWHM = ', Outstr1);
        Drawstring(Outstr);
        I := 0;
        J := 0;
        REPEAT
            y1 := Round(QSpectrum[I] / 1400);
            Moveto(J + 21, 630 - y1);
            Line(0,0);
            I := I + IncI;
            J := J + IncJ;
        UNTIL (I >= FFTNo - 1);
    END;

{-------------------------------------------------------------------------------}
    {$S Part3}
    { This is the routine which prints the statistical information. }

    PROCEDURE PrCFPlot(X, Y, Prwindow: Integer);

        VAR
            OutStr1, OutStr2, OutStr   : Str255;
            i, j, k, m                 : Integer;

        BEGIN
            Pennormal;
            Textsize(0);
            GetWTitle(Ourimages[Prwindow].window, OutStr1);

            Moveto(2, 10);
            TextFace([underline]);
            Drawstring('Statistical Information about Image:');
            TextFace([]);
            OutStr := Concat(' ', OutStr1);
            Drawstring(OutStr);
            Moveto(10, 25);
            NumToString(ZNum, OutStr1);
            OutStr := Concat('Zero value pixels = ', OutStr1);
            Drawstring(OutStr);
            Moveto(10, 40);
            NumToString(TNum, OutStr1);
            OutStr := Concat('Total non-zero pixels = ', OutStr1);
            Drawstring(OutStr);
```

```
Moveto(10, 55);
NumToString(Mean1, OutStr1);
NumToString(Mean2, OutStr2);
OutStr := Concat('Arithmetic mean of non-zero = ', OutStr1, '.', OutStr2);
Drawstring(OutStr);
Moveto(10, 70);
NumToString(NMean1, OutStr1);
NumToString(NMean2, OutStr2);
OutStr := Concat('Arithmetic mean of 95% non-zero = ', OutStr1, '.', OutStr2);
Drawstring(OutStr);
Moveto(100, 85);
NumToString(Large, OutStr1);
NumToString(Small, OutStr2);
OutStr := Concat('( in range: [ ', OutStr2, ' , ', OutStr1, ' ] )');
Drawstring(OutStr);
TextSize(9);

Moveto(8, 110);
Drawstring('95% distribution');
Moveto(5, 125);
Drawstring('Value  Frequency ');
IF C2 = -1 THEN              {Count Frequency case}
   BEGIN
      k := 2;                {Assume the zero value pixel is the most frequent one}

      REPEAT
         NumToString(FinalV[k], OutStr1);
         NumToString(FinalC[k], OutStr2);
         Moveto(5, 125 + k*9);
         Drawstring(OutStr1);
         Moveto(75, 125 + k*9);
         Drawstring(OutStr2);
         k := k + 1;
      UNTIL (k = C1 + 1) OR (k = 60);
   END
ELSE                         {Count Magnitude case}
   BEGIN
      k := C1;
      REPEAT
         NumToString(FinalV[k], OutStr1);
         NumToString(FinalC[k], OutStr2);
         Moveto(5, 125 + (k - C1 + 2)*9);
         Drawstring(OutStr1);
         Moveto(75, 125 + (k - C1 + 2)*9);
         Drawstring(OutStr2);
         k := k + 1;
      UNTIL (k = C2 + 1) OR (k - C1 = 60);
   END;

Moveto(148, 110);
Drawstring(' 5% distribution');
Moveto(145, 125);
Drawstring('Value  Frequency ');
IF C2 = -1 THEN              {Count Frequency case}
   BEGIN
      k := C1 + 1;
      REPEAT
         NumToString(FinalV[k], OutStr1);
         NumToString(FinalC[k], OutStr2);
         Moveto(145, 125 + (k - C1 + 1) * 9);
         Drawstring(OutStr1);
         Moveto(218, 125 + (k - C1 + 1) * 9);
         Drawstring(OutStr2);
```

```
            k := k + 1;
          UNTIL (k = MaxNum + 1) OR (k - C1 = 60);
        END
      ELSE                          {Count Magnitude case}
        BEGIN
          k := 1;
          REPEAT
            NumToString(FinalV[k], OutStr1);
            NumToString(FinalC[k], OutStr2);
            Moveto(145, 125 + (k + 1)*9);
            Drawstring(OutStr1);
            Moveto(218, 125 + (k + 1)*9);
            Drawstring(OutStr2);
            k := k + 1;
          UNTIL (k=C1) OR (k=60);
          m := 0;
          k := C2 + 1;
          REPEAT
            IF (FinalV[k] <> 0) THEN
              BEGIN
                NumToString(FinalV[k], OutStr1);
                NumToString(FinalC[k], OutStr2);
                Moveto(145, 125 + (k - C2 + C1 - m)*9);
                Drawstring(OutStr1);
                Moveto(218, 125 + (k - C2 + C1 - m)*9);
                Drawstring(OutStr2);
              END
            ELSE m := 1;
            k := k + 1;
          UNTIL (k = MaxNum + 1) OR (k - C2 = 60);
        END;

      Moveto(5, 700);
      IF (C2 = -1) THEN
        IF (C1 > 60) THEN
          Drawstring('The list is too long to print fully, omit some data in 95% range.')
        ELSE IF (MaxNum - C1 > 60) THEN
          Drawstring('The list is too long to print fully, omit some data in 5% range.')
      ELSE IF (C2 <> -1) THEN
          Drawstring('The list is too long to print fully, omit some data in 5% range.')
        ELSE IF (C2 - C1 > 60) THEN
          Drawstring('The list is too long to print fully, omit some data in 95% range.');
    END;

{-------------------------------------------------------------------------}
  {$S Part3}
  { This is the routine used in the histogram. }

  PROCEDURE HisSorting;

    VAR
      IncX, ZeroC, Zerok, k, LeftP, RightP, TopP    : Integer;
      OutStr1, OutStr2, OutStr                      : Str255;
      IncY                                          : Extended;
      HisRect                                       : rect;

    BEGIN
      k := 1;
      IncY := 1;
      REPEAT
        IF FinalV[k] = 0 THEN
          BEGIN
            ZeroC := FinalC[k];
```

```
            Zerok := k;
        END
    ELSE IF FinalC[k] > IncY THEN IncY := FinalC[k];
    k := k + 1;
UNTIL k = MaxNum + 1;

IncX := Round(500/MaxNum);
IncY := 1.2*IncY;

Moveto(40, 20);
Lineto(40, 135);
Moveto(40, 135);
Lineto(600, 135);

Moveto((MaxNum-C1)*IncX+IncX DIV 2+50,135);
Lineto((MaxNum - C1)*IncX + IncX DIV 2 + 50, 138);
Moveto((MaxNum - C2)*IncX + IncX DIV 2 + 50, 135);
Lineto((MaxNum - C2)*IncX + IncX DIV 2 + 50, 138);
Moveto(35, 22);
Lineto(40, 22);

Numtostring(ZeroC, Outstr1);
Moveto((MaxNum - Zerok + 1)*IncX + 53, 20);
Drawstring(Outstr1);
Numtostring(Round(IncY/1.2), Outstr2);
Moveto(10, 26);
Drawstring(Outstr2);

Numtostring(FinalV[C1], Outstr1);
Moveto((MaxNum - C1)*IncX + IncX DIV 2 + 40, 150);
Drawstring(Outstr1);
Numtostring(FinalV[C2], Outstr2);
Moveto((MaxNum - C2)*IncX + IncX DIV 2 + 40, 150);
Drawstring(Outstr2);

k := MaxNum;
REPEAT
    LeftP := (MaxNum - k)*IncX + 50;
    TopP := 130 - Round(130*(FinalC[k]/IncY));
    RightP := (MaxNum - k + 1)*IncX + 50;

    IF (k <= MaxNum) AND (k > C2) THEN
        IF (FinalV[k] = 0) THEN
            BEGIN
                Forecolor(blackColor);
                SetRect(HisRect, LeftP, 5, RightP, 130);
            END
        ELSE
            BEGIN
                Forecolor(blueColor);
                SetRect(HisRect, LeftP, TopP, RightP, 130);
            END
    ELSE IF (k <= C2) AND (k >= C1) THEN
        BEGIN
            Forecolor(redColor);
            SetRect(HisRect, LeftP, TopP, RightP, 130);
        END
    ELSE IF (k < C1) THEN
        BEGIN
            Forecolor(blueColor);
            SetRect(HisRect, LeftP, TopP, RightP, 130);
        END;
    PaintRect(HisRect);
```

```
            k := k - 1;
        UNTIL (k = 0);
    END;

{-------------------------------------------------------------------------}
    {$S Part3}
    { Print the histogram. }

    PROCEDURE PrHisPlot(X, Y, Prwindow: Integer);

        VAR
            OutStr1, OutStr2, OutStr   : Str255;
            Ourcursorh                 : Cursshandle;

        BEGIN
            Ourcursorh := Getcursor(4);
            Setcursor(Ourcursorh^^);

            Pennormal;
            Textsize(0);
            GetWTitle(Ourimages[Prwindow].window, OutStr1);

            Moveto(2, 160);
            TextFace([underline]);
            Drawstring('Histogram of Image:');
            TextFace([]);
            OutStr := Concat(' ', OutStr1);
            Drawstring(OutStr);

            HisSorting;
        END;

{-------------------------------------------------------------------------}
    {$S Part3}
    { Main printing routine. }

    PROCEDURE Doprint(X, Y, Prwindow, scbase: Integer; Scfac, Xshift, Yshift: Longint);

        VAR
            Prrechdl          : Thprint;
            Myprport          : Tpprport;
            Mystrec           : Tprstatus;
            R                 : Rect;
            Aprocptr          : Procptr;
            Prrect            : Tprect;
            Ourcursorh        : Cursshandle;
            vector, dontdoit  :Boolean;
            itemhit           :Integer;

        BEGIN
            IF NOT Cursmode AND NOT QMode AND NOT CFmode AND NOT Hismode
                THEN
                IF Ourimages[Prwindow].Imagesize = GoodArray3 THEN
                    Setrect(R, 0, 0, 100, 100)
                ELSE
                    Setrect(R, 0, 0, 400, 400);

            dontdoit := false;

            Prrect := @R;
            Propen;
            Prrechdl := Thprint(Newhandle(Sizeof(Tprint)));
            Printdefault(Prrechdl);
```

```
IF Prstldialog(Prrechdl) THEN
   BEGIN
      IF Prjobdialog(Prrechdl) THEN
         BEGIN
            Myprport := Propendoc(Prrechdl, nil, nil);
            IF Prerror = Noerr THEN
               BEGIN
                  IF Cursmode THEN
                     BEGIN
                        Mydialog := Getnewdialog(141, nil, Pointer( - 1));
                        SetOKButton(MyDialog);

                        Modaldialog(nil, Itemhit);
                        CASE Itemhit OF
                        1:
                        BEGIN
                           vector := false;
                           Disposdialog(Mydialog);
                        END;
                        2:
                        BEGIN
                           vector := true;
                           Disposdialog(Mydialog);
                        END;
                        5:
                        BEGIN
                           dontdoit := true;
                           Disposdialog(Mydialog);
                        END;
                     END;
               END;
            IF NOT(dontdoit) THEN
               BEGIN
                  Updatewindow;
                  Ourcursorh := Getcursor(watchCursor);
                  Setcursor(Ourcursorh^^);

                  IF NOT Cursmode AND NOT QMode AND NOT CFmode
                                                AND NOT Hismode THEN
                     Propenpage(Myprport, Prrect)
                  ELSE
                     Propenpage(Myprport, nil);

                  IF Prerror = Noerr THEN
                     IF Cursmode THEN
                        PrPlot(X, Y, Prwindow, scbase, Scfac, Yshift, vector)
                     ELSE IF QMode THEN
                        PrQPlot(X, Y, Prwindow, Xshift, Yshift)
                     ELSE IF CFMode THEN
                        PrCFPlot(X, Y, Prwindow)
                     ELSE IF HisMode THEN
                        PrHisPlot(X, Y, Prwindow)
                     ELSE
                        Prdrawingproc(Prwindow);

                  Prclosepage(Myprport);

                  Ourcursorh := Getcursor(crossCursor);
                  Setcursor(Ourcursorh^^);
               END;
            Prclosedoc(Myprport);
            IF (Prrechdl^^.Prjob.Bjdocloop = Bspoolloop) AND (Prerror = Noerr) THEN
               BEGIN
```

```
                        Ourcursorh := Getcursor(watchCursor);
                        Setcursor(Ourcursorh^^);

                        Aprocptr := @Donotice;
                        Unloadseg(Aprocptr);
                        Aprocptr := @Pictsave;
                        Unloadseg(Aprocptr);
                        Prpicfile(Prrechdl, nil, nil, nil, Mystrec);

                        Ourcursorh := Getcursor(crossCursor);
                        Setcursor(Ourcursorh^^);
                    END;
                END;
            END;
        END;
    Prclose;
END;

{-------------------------------------------------------------------------------}
  {$S Part3}
  { Print calling routine. }

  PROCEDURE Printing(X, Y, Prwindow, scbase: Integer; Scfac, Xshift, Yshift: Longint);

    VAR
      Itemhit                  : Integer;
      Itemstring1, Itemstring2 : Str255;

    BEGIN
      Getitem(Myhandle[Cursormenu], 1, Itemstring1);
      Getitem(Myhandle[Multislicesmenu], 1, Itemstring2);
      IF Cursmode AND (Itemstring1 = 'Hide Cursor') THEN
                Doprint(X, Y, Prwindow, scbase, Scfac, Xshift, Yshift)
      ELSE IF   QMode AND (Itemstring2 = 'Hide q-Space Plots') THEN
                Doprint(X, Y, Prwindow, scbase, Scfac, Xshift, Yshift)
      ELSE IF   CFmode OR Hismode THEN
                Doprint(X, Y, Prwindow, scbase, Scfac, Xshift, Yshift)
      ELSE   {print image}
        BEGIN
          IF (Ourimages[Prwindow].Imagesize = GoodArray3) THEN
                Doprint(X, Y, Prwindow, scbase, Scfac, Xshift, Yshift);

          IF (Ourimages[Prwindow].Imagesize <> GoodArray3) THEN
            BEGIN
              Mydialog := Getnewdialog(140, nil, Pointer( - 1));
              SetOKButton(MyDialog);

              Modaldialog(nil, Itemhit);
              CASE Itemhit OF
                2:
                  BEGIN
                    Disposdialog(Mydialog);
                    Doprint(X, Y, Prwindow, scbase, Scfac, Xshift,Yshift);
                  END;
                3: Disposdialog(Mydialog);
              END;
            END;
        END;
    END;

{===============================================================================}
{-------------------------------------------------------------------------------}
  {$S Statis}
```

```
{ This the main routine which counts the entire image pixel array. }

PROCEDURE FirstCount(Num: Integer);

   VAR
      i, Ball, BallC     : Integer;
      Place, Plate1      : Longint;
      Done               : Boolean;
      Myarray1           : Intptr;
      Ourcursorh         : Curshandle;
   LABEL 222;

   BEGIN
      Ball := 128;
      Ourcursorh := Getcursor(Ball);
      Setcursor(Ourcursorh^^);
      Place := 0;

      FOR i := 1 TO MaxCount DO
         BEGIN
            FinalC[i] := 0;
            FinalV[i] := 0;
         END;

      {Count the frequency of each different value}

      BallC := 1;
      REPEAT
         Myarray1 := Intptr(Ord(Ourimages[Num].Data^) + Place);
         Plate1 := Myarray1^;

         i := 1;
         Done := False;
         REPEAT
            IF (FinalC[i] = 0) THEN
               BEGIN
                  FinalV[i] := Plate1;
                  FinalC[i] := FinalC[i] + 1;
                  Done := True;
               END
            ELSE IF (FinalV[i] = Plate1)  THEN
               BEGIN
                  FinalC[i] := FinalC[i] + 1;
                  Done := True;
               END;
            i := i + 1;

            IF i >= MaxCount - 1 THEN
               BEGIN
                  Over := True;
                  GOTO 222;
               END;
         UNTIL Done = True;

         BallC := BallC + 1;
         IF (BallC = 60) THEN
            BEGIN
               BallC := 1;
               Ball := Ball + 1;
               IF (Ball=132) THEN Ball := 128;
            END;
         Ourcursorh := Getcursor(Ball);
         Setcursor(Ourcursorh^^);
```

```
        Place := Place + 2;
      UNTIL Place >= Ourimages[Num].Imagesize;

      i := 1;
      REPEAT
        IF (FinalC[i] >0) THEN i := i +1;
      UNTIL FinalC[i] = 0;
      MaxNum := i - 1;

222: IF i >= MaxCount - 1 THEN DoNotice(130);
    END;

{-------------------------------------------------------------------------------}
  {$S Statis}
  { Set up a window for displaying the statistical information. }

  PROCEDURE SetCFWindow;

    VAR
      OutStr1, OutStr2, OutStr  : Str255;
      windowsize                : rect;

    BEGIN
      Setrect(Windowsize, 330, 30, 630, 470);
      Mycurwindow := Newcwindow(@Curwrecord, Windowsize, 'Cursor', True,
                                                dBoxProc, Pointer( - 1), True, 0);
      Setport(MyCurwindow);
      TextSize(10);
      TextFace([bold]);

      Moveto(5, 10);
      NumToString(TNum,OutStr1);
      OutStr := Concat('Total non-zero pixels = ', OutStr1);
      Drawstring(OutStr);
      Moveto(5, 20);
      NumToString(Mean1,OutStr1);
      NumToString(Mean2,OutStr2);
      OutStr := Concat('Arithmetic mean of non-zero = ', OutStr1, '.', OutStr2);
      Drawstring(OutStr);
      Moveto(5, 31);
      NumToString(NMean1,OutStr1);
      NumToString(NMean2,OutStr2);
      OutStr := Concat('Arithmetic mean of 95% non-zero = ', OutStr1, '.', OutStr2);
      Drawstring(OutStr);
      Moveto(45, 42);
      NumToString(Large,OutStr1);
      NumToString(Small,OutStr2);
      OutStr := Concat('( in range: [ ', OutStr2, ' , ', OutStr1, ' ] )');
      Drawstring(OutStr);

      Moveto(5, 55);
      Drawstring('Value    Frequency ');
      Moveto(160, 55);
      Drawstring('Value    Frequency ');
      TextFace([]);
    END;

{-------------------------------------------------------------------------------}
  {$S Statis}
  { Re-arrange some of least frequent data according to deviation. }

  PROCEDURE ReArrange_Freq;
```

```
    VAR
       ValueF              : Array [0..MaxCount] OF Integer;
       CountF              : Array [0..MaxCount] OF Longint;
       i, j, k, m, n, Ball : Integer;
       TempC, TempV        : Longint;
       Ourcursorh          : Cursshandle;

    BEGIN
       Ball := 131;
       Ourcursorh := Getcursor(Ball);
       Setcursor(Ourcursorh^^);

       i := MaxNum + 1;
       FOR n := 1 TO 10 DO                {do frequency from 1 to 10}
          BEGIN
             m := i;
             REPEAT
                i := i-1;
             UNTIL FinalC[i] > n;
             i := i+1;
             FOR j := i TO m-1 DO
                BEGIN
                   ValueF[j] := FinalV[j];
                   CountF[j] := FinalC[j];
                END;

             FOR k := i TO m-1 DO
                BEGIN
                   TempV := 32766;
                   TempC := k;
                   FOR j := i TO m-1 DO
                      IF ((CountF[j] <> -1) AND (Abs(ValueF[j] - Mean1) < TempV)) THEN
                         BEGIN
                            TempV := Abs(ValueF[j] - Mean1);
                            TempC := j;
                            FinalV[k] := ValueF[j];
                         END;
                   CountF[TempC] := -1;
                END;

             Ball := Ball + 1;
             IF (Ball=132) THEN Ball := 128;
             Ourcursorh := Getcursor(Ball);
             Setcursor(Ourcursorh^^);
          END;
    END;

{-------------------------------------------------------------------------}
    {$S Statis}
    { The routine does the counting according to the frequencies. }

    PROCEDURE Freq_Stats(X, Y, Prwindow, scbase: Integer; Scfac, Yshift: Longint);

       VAR
          ValueF                    : Array [0..MaxCount] OF Integer;
          CountF                    : Array [0..MaxCount] OF Longint;
          Num, i, j, k, m, n, Ball  : Integer;
          OutStr1, OutStr2, OutStr  : Str255;
          TValue, NTValue           : Extended;
          windowsize                : rect;
          TempC, TempV              : Longint;
          Ourcursorh                : Cursshandle;
       LABEL 20;
```

```
BEGIN
   Ball := 129;
   Ourcursorh := Getcursor(Ball);
   Setcursor(Ourcursorh^^);
   Num := Getwrefcon(Frontwindow);

   {Count the frequency of each different value}

   FirstCount(Num);

   IF (NOT Over) THEN
      BEGIN
         CFmode := True;
         FOR i := 1 TO MaxNum DO
            BEGIN
               ValueF[i] := FinalV[i];
               CountF[i] := FinalC[i];
            END;

         {Sort the data in the order of frequency magnitude}

         FinalV[0] := 60000;
         TempC := 0;
         TempV := 0;

         FOR j := 1 TO MaxNum DO
            BEGIN
               FOR i := 1 TO MaxNum DO
                  BEGIN
                     k := 0;
                     REPEAT
                        IF (ValueF[i] = FinalV[k]) THEN GOTO 20;
                        k := k + 1;
                     UNTIL k = j;

                     IF (CountF[i] > TempC) THEN
                        BEGIN
                           TempC := CountF[i];
                           TempV := ValueF[i];
                        END;
20:                  END;
               FinalV[j] := TempV;
               FinalC[j] := TempC;
               TempC := 0;

               Ball := Ball + 1;
               IF (Ball=132) THEN Ball := 128;
               Ourcursorh := Getcursor(Ball);
               Setcursor(Ourcursorh^^);
            END;

         {Calculate arithmetic mean value of the total non-zero data and etc}

         TNum := 0;
         TValue := 0.0;
         FOR j := 1 TO MaxNum DO
            BEGIN
               IF (FinalV[j] <> 0) THEN
                  BEGIN
                     TNum := TNum + FinalC[j];
                     TValue := TValue + FinalC[j]*FinalV[j];
                  END
```

```
        ELSE ZNum := FinalC[j];
    END;
Mean1 := Trunc(TValue/TNum);
Mean2 := Abs(Round(100*(TValue/TNum)) MOD 100);

{Re-arrange some of least frequent data according to deviation}

ReArrange_Freq;
Ourcursorh := Getcursor(4);
Setcursor(Ourcursorh^^);

{Calculate arithmetic mean value of 95% total non-zero and etc}

j := MaxNum;
NTNum := TNum;
NTValue := TValue;
REPEAT
    NTNum := NTNum - FinalC[j];
    NTValue := NTValue - FinalC[j]*FinalV[j];
    j:=j-1;
UNTIL NTNum < Round(0.95*TNum);
NMean1 := Trunc(NTValue/NTNum);
NMean2 := Abs(Round(100*(NTValue/NTNum)) MOD 100);

C1 := j;
C2 := -1;

Large := NMean1;
Small := NMean1;
FOR i := 2 TO j DO          {Assume zero value is the most frequent one}
        IF (FinalV[i] >= Large) THEN    Large := FinalV[i]
    ELSE IF (FinalV[i] < Small) THEN Small := FinalV[i];

{Print on the screen}

SetCFWindow;
TextSize(9);

i := 2;                     {Assume that zero value is the most frequent value}
REPEAT
    NumToString(FinalV[i], OutStr1);
    NumToString(FinalC[i], OutStr2);

    Moveto(5, 60 + (i - 1)*9);
    Drawstring(OutStr1);
    Moveto(70, 60 + (i - 1)*9);
    Drawstring(OutStr2);
    i := i + 1;
UNTIL (i = j+1) OR (i=41);

i := j + 1;
REPEAT
    NumToString(FinalV[i], OutStr1);
    NumToString(FinalC[i], OutStr2);

    Moveto(160, 60 + (i - j)*9);
    Drawstring(OutStr1);
    Moveto(225, 60 + (i - j)*9);
    Drawstring(OutStr2);
    i := i + 1;
UNTIL (i=MaxNum + 1) OR (i - j = 40);

Moveto(1, 438);
```

```
            TextFace([bold]);
            IF ( j > 40 ) OR ( MaxNum-j > 40) THEN
                Drawstring('Click mouse button to print more details or exit.')
              ELSE
                Drawstring('Click mouse button to print or exit.');
            TextFace([]);

            REPEAT UNTIL Button;
            Initcursor;
            Printing(X, Y, Num, scbase, Scfac, Xshift, Yshift);
            CFmode := False;
            Disposewindow(Mycurwindow);
          END;
        Over := False;
        Initcursor;
    END;

{-------------------------------------------------------------------------------}
  {$S Statis}
  { The routine sorts out the sequence according to their magnitudes. }

  PROCEDURE MagnitudeSorting;

    VAR
      ValueF                          : Array [0..MaxCount] OF Integer;
      CountF                          : Array [0..MaxCount] OF Longint;
      i, j, TempC, TempV, Ball, BallC : Integer;
      TValue, NTValue                 : Extended;
      Ourcursorh                      : Curshandle;
      Tempx                           : Longint;
    LABEL 50;

    BEGIN
      Ball := 128;
      BallC := 1;
      Ourcursorh := Getcursor(Ball);
      Setcursor(Ourcursorh^^);
      FOR i := 1 TO MaxNum DO
        BEGIN
          ValueF[i] := FinalV[i];
          CountF[i] := FinalC[i];
        END;

      {Sort the data in the order of value decrement}

      FinalV[0] := 32766;
      TempV := -32766;
      FOR j := 1 TO MaxNum DO
        BEGIN
          FOR i := 1 TO MaxNum DO
            IF (ValueF[i] > TempV) AND (ValueF[i] < FinalV[j-1]) THEN
              BEGIN
                TempC := CountF[i];
                TempV := ValueF[i];
              END;
          FinalV[j] := TempV;
          FinalC[j] := TempC;
          TempV := -32000;

          Ball := Ball + 1;
          IF (Ball=132) THEN Ball := 128;
          Ourcursorh := Getcursor(Ball);
          Setcursor(Ourcursorh^^);
```

```
        END;

      {Calculate arithmetic mean value and etc}

      TNum := 0;
      TValue := 0.0;
      FOR j := 1 TO MaxNum DO
        IF (FinalV[j] <> 0) THEN
          BEGIN
            TNum := TNum + FinalC[j];
            TValue := TValue + FinalC[j]*FinalV[j];
          END
        ELSE ZNum := FinalC[j];
      Mean1 := Trunc(TValue/TNum);
      Mean2 := Abs(Round(100*(TValue/TNum)) MOD 100);

      {Calculate 95% arithmetic mean value and etc}

      i := 1;
      j := MaxNum;
      NTNum := TNum;
      NTValue := TValue;
      Tempx := Mean1;

      REPEAT
        IF ((FinalV[i] <> 0) AND (FinalV[j] <> 0)) THEN
          BEGIN
            IF (Abs(FinalV[j] - Tempx) > Abs(FinalV[i] - Tempx)) THEN
              BEGIN
                NTNum := NTNum - FinalC[j];
                NTValue := NTValue - FinalC[j]*FinalV[j];
                j:=j-1;
                IF (NTNum < Round(0.95*TNum)) THEN GOTO 50;
              END
            ELSE
              BEGIN
                NTNum := NTNum - FinalC[i];
                NTValue := NTValue - FinalC[i]*FinalV[i];
                i:=i+1;
              END;
            Tempx := Trunc(NTValue/NTNum);
          END
        ELSE   IF (FinalV[i] = 0) THEN i := i + 1
        ELSE   IF (FinalV[j] = 0) THEN j := j - 1;

      UNTIL NTNum < Round(0.95*TNum);

50:   NMean1 := Trunc(NTValue/NTNum);
      NMean2 := Abs(Round(100*(NTValue/NTNum)) MOD 100);

      Large := FinalV[i];
      Small := FinalV[j];
      C1 := i;
      C2 := j;
    END;

{-------------------------------------------------------------------------------}
  {$S Statis}
  { The routine which does the counting according to the magnitudes. }

  PROCEDURE Magn_Stats(X, Y, Prwindow, scbase: I nteger; Scfac, Yshift: Longint);

    VAR
```

```
    Num, i, j, k, m            : Integer;
    OutStr1, OutStr2, OutStr   : Str255;
    windowsize                 : rect;
    Ourcursorh                 : Curshandle;
LABEL 11;

BEGIN
    CFmode := True;
    Num := Getwrefcon(Frontwindow);
    FirstCount(Num);                    {Count the frequency of each different value}

    IF (NOT Over) THEN
       BEGIN
          MagnitudeSorting;             {Sorting the data}

          SetCFWindow;                  {Print on the screen}
          Ourcursorh := Getcursor(4);
          Setcursor(Ourcursorh^^);
          TextSize(9);

          k := C1;
          REPEAT
             NumToString(FinalV[k], OutStr1);
             NumToString(FinalC[k], OutStr2);
             Moveto(5, 60 + (k - C1 + 1)*9);
             Drawstring(OutStr1);
             Moveto(70, 60 + (k - C1 + 1)*9);
             Drawstring(OutStr2);
             k := k + 1;
          UNTIL (k = C2 + 1) OR (k - C1 = 40);

          k := 1;
          REPEAT
             NumToString(FinalV[k], OutStr1);
             NumToString(FinalC[k], OutStr2);
             Moveto(160, 60 + k*9);
             Drawstring(OutStr1);
             Moveto(225, 60 + k*9);
             Drawstring(OutStr2);
             k := k + 1;
          UNTIL (k = C1) OR (k = 40);
          IF (k >= 40) THEN GOTO 11;

          k := C2 + 1;
          m := 0;
          REPEAT
             IF (FinalV[k] <> 0) THEN
                BEGIN
                   NumToString(FinalV[k],OutStr1);
                   NumToString(FinalC[k],OutStr2);
                   Moveto(160, 60 + (k - C2 + C1 - 1 - m)*9);
                   Drawstring(OutStr1);
                   Moveto(225, 60 + (k - C2 + C1 - 1 - m)*9);
                   Drawstring(OutStr2);
                END
             ELSE m := 1;
             k := k + 1;
          UNTIL (k = MaxNum + 1) OR (k - C2 + C1 - 1 - m = 40);

11:       Moveto(1, 438);
          TextFace([bold]);
          IF ( C2 - C1 > 40 ) OR ( MaxNum - C2 + i > 40) THEN
                Drawstring('Click mouse button TO print more details.')
```

```
            ELSE
                  Drawstring('Click mouse button TO print or exit.');
            TextSize(0);
            TextFace([]);

            REPEAT UNTIL Button;
            Printing(X, Y, Num, scbase, Scfac, Xshift, Yshift);
            Disposewindow(Mycurwindow);
            CFmode := False;
         END;
      Over := False;
      Initcursor;
   END;

{-------------------------------------------------------------------------------}
   {$S Statis}
   { The main routine which does the histogram. }

   PROCEDURE Histogram(X, Y, Prwindow,s cbase: Integer; Scfac, Yshift: Longint);

      VAR
         Num          : Integer;
         windowsize   : rect;
         Ourcursorh   : Curshandle;

      BEGIN
         Num := Getwrefcon(Frontwindow);
         Ourcursorh := Getcursor(4);
         Setcursor(Ourcursorh^^);

         FirstCount(Num);          {Count the frequency of each different value}
         IF (NOT Over) THEN
            BEGIN
               Hismode := True;
               MagnitudeSorting;

               Setrect(Windowsize, 10, 320, 630, 470);
               Mycurwindow := Newcwindow(@Curwrecord, Windowsize, '', True,
                                                   Dboxproc, Pointer(-1), True, 0);
               Setport(MyCurwindow);
               PenPat(black);
               Pennormal;

               HisSorting;

               Initcursor;
               REPEAT UNTIL Button;
               Printing(X, Y, Num, scbase, Scfac, Xshift, Yshift);
               Hismode := False;
               Disposewindow(Mycurwindow);
            END;
         Over := False;
         Initcursor;
      END;

{-------------------------------------------------------------------------------}
{-------------------------------------------------------------------------------}
   {$S main}
   { Mousedown event handling routine. }

   PROCEDURE Domousedown;

      VAR
```

```
    R                                   : Rect;
    Factor, Count, Num, Refnum, Temp, i : Integer;
    Endpt, Mymouseloc                   : Point;
    Close                               : Boolean;
    Itemstring, Title, Name             : Str255;
    Templ, Menusel, Windowmeasure       : Longint;
    Whichwindow                         : Windowptr;
LABEL 10;

BEGIN
   CASE Findwindow(Myevent.Where, Whichwindow) OF

   Inmenubar:
      BEGIN
         Menusel := Menuselect(Myevent.Where);
         Num := Getwrefcon(Frontwindow);

         CASE Hiword(Menusel) OF
         Applemenu:
            BEGIN
               IF Loword(Menusel) = 1 THEN  Donotice(136)
               ELSE
                  BEGIN
                     Getitem(Myhandle[Applemenu], Loword(Menusel), Name);
                     Refnum := Opendeskacc(Name);
                  END;
            END;

         Imagemenu:
            BEGIN
               CASE Loword(Menusel) OF
               1:
                  BEGIN
                     IF Cursmode OR QMode THEN
                        BEGIN
                           Initcursor;
                           Cursmode := False;
                           Setcurmenu;
                           Qmode := False;
                           SetQMenu;
                           Disposewindow(Mycurwindow);
                        END;
                     Doarray;
                  END;
               3:
                  BEGIN
                     Getitem(Myhandle[Cursormenu], 1, Itemstring);
                     IF Cursmode AND (Itemstring = 'Hide Cursor') THEN
                                 SaveAsCG(X,Y,Num,XShift,YShift)
                        ELSE Savearray;
                  END;
               4: Pictsave(False);
               5: Pictsave(True);
               7: Printing(X,Y,Num,scbase,Scfac,Xshift,Yshift);
               9: Quit := True;
               END;    {of CASE}
            END;

         Editmenu:
            BEGIN
               IF NOT Systemedit(Loword(Menusel) - 1) THEN
                  CASE Loword(Menusel) OF
                  3:
```

```
                BEGIN
                    Tecut(Texth);
                    Templ := Zeroscrap;
                    Temp := Tetoscrap;
                END;
            4:
                BEGIN
                    Tecopy(Texth);
                    Templ := Zeroscrap;
                    Temp := Tetoscrap;
                END;
            5:
                BEGIN
                    Temp := Tefromscrap;
                    Tepaste(Texth);
                END;
            6: Tedelete(Texth);
            END; {of LoWord(menusel) CASE}
        END; {of 130}

Scalemenu:
    BEGIN
        IF Cursmode THEN
            BEGIN
                Initcursor;
                Cursmode := False;
                Setcurmenu;
                Disposewindow(Mycurwindow);
            END;

        CASE Loword(Menusel) OF
        1:
            BEGIN
                Getitem(Myhandle[Scalemenu], 1, Itemstring);
                IF Ourimages[Num].Showcscale AND (Itemstring = 'Hide Scale')
                        THEN    Closecscale(Num);
                IF NOT (Ourimages[Num].Showcscale) AND
                        (Itemstring = 'Show Scale') THEN    Createcscale(Num);
            END;
        3:
            BEGIN
                Ourimages[Num].Cscaletype := True;
                Changeourpalette(Num, Clut4);
            END;
        4:
            BEGIN
                Ourimages[Num].Cscaletype := True;
                Changeourpalette(Num, Clut8);
            END;
        5:
            BEGIN
                Ourimages[Num].Cscaletype := True;
                Changeourpalette(Num, Clut16);
            END;
        6:
            BEGIN
                Ourimages[Num].Cscaletype := True;
                Changeourpalette(Num, Clut32);
            END;
        7:
            BEGIN
                Changeourpalette(Num, Greyclut);
                Ourimages[Num].Cscaletype := True;
```

```
            END;
        8:
            BEGIN
                Ourimages[Num].Cscaletype := False;
                Changeourpalette(Num, Monoclut);
            END;
        9:
            BEGIN
                Ourimages[Num].Cscaletype := False;
                Changeourpalette(Num, Flow15clut);
            END;
        11: Showcontrols;
        END;    {of CASE}
    END;

Cursormenu:
    BEGIN
        CASE Loword(Menusel) OF
        1:
            BEGIN
                IF QMode THEN
                    BEGIN
                        Initcursor;
                        Qmode := False;
                        SetQMenu;
                        Disposewindow(Mycurwindow);
                    END;
                Getitem(Myhandle[Cursormenu], 1, Itemstring);
                IF Cursmode AND (Itemstring = 'Hide Cursor') THEN
                    BEGIN
                        Initcursor;
                        Cursmode := False;
                        Setcurmenu;
                        Disposewindow(Mycurwindow);
                    END;
                IF NOT (Cursmode) AND (Itemstring = 'Show Cursor') THEN
                    BEGIN
                        Cursmode := True;
                        Setcurmenu;
                        Coordcursor(Num);
                    END;
            END;
        2:
            BEGIN
                IF Cursmode THEN
                    BEGIN
                        Scalemode := True;
                        Dispcoord(Num, Scbase, X, Y, Scfac, Xshift, Yshift);
                    END;
            END;
        4:
            BEGIN
                IF Cursmode OR QMode THEN
                    BEGIN
                        Initcursor;
                        Cursmode := False;
                        Setcurmenu;
                        Qmode := False;
                        SetQMenu;
                        Disposewindow(Mycurwindow);
                    END;
                DoRetouch;
            END;
```

```
        END;    {of CASE}
    END;

Displaymenu:
    BEGIN
        CASE Loword(Menusel) OF
        1:
            BEGIN
                IF Cursmode OR QMode THEN
                    BEGIN
                        Initcursor;
                        Cursmode := False;
                        Setcurmenu;
                        Qmode := False;
                        SetQMenu;
                        Disposewindow(Mycurwindow);
                    END;
                DoStackplot;
            END;
        2,3,4:
            BEGIN
                IF Cursmode OR QMode THEN
                    BEGIN
                        Initcursor;
                        Cursmode := False;
                        Setcurmenu;
                        Qmode := False;
                        SetQMenu;
                        Disposewindow(Mycurwindow);
                    END;
                CASE Loword(Menusel) OF
                    2: Freq_Stats(X, Y, Num, scbase, Scfac, Yshift);
                    3: Magn_Stats(X, Y, Num, scbase, Scfac, Yshift);
                    4: Histogram(X, Y, Num, scbase, Scfac, Yshift);
                END;
            END;
        6:
            BEGIN
                IF Ourimages[Num].Square = True THEN
                    BEGIN
                        Ourimages[Num].Square := False;
                        Setitem(Myhandle[Displaymenu], 6, 'Square');
                    END
                ELSE
                    BEGIN
                        Ourimages[Num].Square := True;
                        Setitem(Myhandle[Displaymenu], 6, 'Non-Square');
                    END;
            END;
        8:
            BEGIN
                IF Cursmode THEN
                    BEGIN
                        Initcursor;
                        Cursmode := False;
                        Setcurmenu;
                        Disposewindow(Mycurwindow);
                    END;
                Getitem(Myhandle[Displaymenu], 8, Itemstring);
                IF Qmode AND (Itemstring = 'Hide q Plots ') THEN
                    BEGIN
                        Initcursor;
                        QMode := False;
```

```
                SetQMenu;
                Disposewindow(Mycurwindow);
            END;
        IF NOT (Qmode) AND (Itemstring = 'Show q Plots ') THEN
            BEGIN
                QMode := True;
                SetQMenu;
                Coordcursor(Num);
            END;
        END;
    END;    {of CASE}
END;

Operationmenu:
BEGIN
    IF Cursmode OR QMode THEN
        BEGIN
            Initcursor;
            Cursmode := False;
            Setcurmenu;
            Qmode := False;
            SetQMenu;
            Disposewindow(Mycurwindow);
        END;
    CASE Loword(Menusel) OF
        1: Doconvol;
        2: Doedgedet;
        3: DoArith;
        4: DoMath;
        5: DoFilter;
        6: DoShifting;
        13: DoRadialAve;
        14: DoAnti_Sloping;
        8,9,10,11:
            BEGIN
                IF (Num < Maxwindows) THEN
                IF (Num >= 1) THEN
                IF(Ourimages[Num].Imagesize =
                                Ourimages[Num-1].Imagesize) THEN
                    BEGIN
                        CASE Loword(Menusel) OF
                            8:  Dolinearcom;
                            9:  DoModulus;
                            10: DoRatio;
                            11: Do2D_FT;
                        END;
                    END
                ELSE
                        Donotice(149)
                ELSE
                        Donotice(149)
                ELSE
                        Donotice(134);
            END;
    END;    {of CASE}
END;

Multislicesmenu:
BEGIN
    IF Cursmode OR QMode THEN
        BEGIN
            Initcursor;
            Cursmode := False;
```

```
                    Setcurmenu;
                    Qmode := False;
                    SetQMenu;
                    Disposewindow(Mycurwindow);
                END;
            CASE Loword(Menusel) OF
                1: DoDynamicAnalysis(Loword(Menusel));
                2: DoDynamicAnalysis(Loword(Menusel));
                3: DoSingleQ;
            END;
        END;

    END;    {of CASE Hiword(Menusel)}
    Hilitemenu(0);
END;

Insyswindow: Systemclick(Myevent, Whichwindow);

Indrag:
    BEGIN
        Dragwindow(Whichwindow, Myevent.Where, Dragrect);
        Setthescale;
    END;

Incontent:
    BEGIN
        Selectwindow(Whichwindow);
        Setport(Whichwindow);
        Num := Getwrefcon(Frontwindow);
        Setthescale;

        IF Cursmode AND (Whichwindow <> Ourimages[Num].Colscalew) THEN
            BEGIN
                Xshift := 0;
                Yshift := 0;
                Getmouse(Mymouseloc);
                X := Mymouseloc.H;
                Y := Mymouseloc.V;
                Dispcoord(Num, Scbase, X, Y, Scfac, 0, 0);
            END;

        IF QMode AND (Whichwindow <> Ourimages[Num].Colscalew) THEN
            BEGIN
                Xshift := 0;
                Yshift := 0;
                Getmouse(Mymouseloc);
                X := Mymouseloc.H;
                Y := Mymouseloc.V;
                ShowQPlots(Num, X, Y, Xshift, Yshift);
            END;
    END; {of Incontent}

Ingrow:
    BEGIN
        IF Cursmode THEN GOTO 10;
        Setport(Whichwindow);
        Setrect(R, 20, 100, 650, 500);
        Num := Getwrefcon(Frontwindow);
        Windowmeasure := Growwindow(Whichwindow, Myevent.Where, R);
        Getwtitle(Whichwindow, Title);

        IF (Title = 'S') OR (Title = 'D') THEN
            BEGIN
```

```
                    Factor := (Hiword(Windowmeasure) DIV 32) * 32;
                    Sizewindow(Whichwindow, Loword(Windowmeasure), Factor, True);
                END
            ELSE IF Ourimages[Num].Square = False THEN
                Sizewindow(Whichwindow, Loword(Windowmeasure), Hiword(Windowmeasure),
                                                                        True)
            ELSE
                BEGIN
                    IF Hiword(Windowmeasure) > Loword(Windowmeasure) THEN
                        BEGIN
                            Sizewindow(Whichwindow, Loword(Windowmeasure),
                                                    Loword(Windowmeasure), True);
                            Ourimages[Num].Dispdimension := Loword(Windowmeasure);
                        END
                    ELSE
                        BEGIN
                            Sizewindow(Whichwindow, Hiword(Windowmeasure),
                                                    Hiword(Windowmeasure), True);
                            Ourimages[Num].Dispdimension := Hiword(Windowmeasure);
                        END;
                END;
        10:
            Invalrect(Whichwindow^.Portrect);
        END;

    Ingoaway:
        BEGIN
            Close := Trackgoaway(Whichwindow, Myevent.Where);
            IF Close and (Not(CursMode)) THEN
                BEGIN
                    Num := GetWRefCon(whichWindow);
                    HideWindow(WindowPtr(whichWindow));
                    CloseCScale(Num);
                END;
        END;

    END;
END;

{-------------------------------------------------------------------------}
  {$S main}
  { Keydown event handling routine. }

  PROCEDURE Dokey;

    VAR
      Num          : Integer;
      Itemstring   : Str255;
      Thekeys      : Keymap;
      Thechar      : Char;

    BEGIN
      Thechar := Chr(Band(Myevent.Message, Charcodemask));
      Getitem(Myhandle[Cursormenu], 1, Itemstring);
      Num := Getwrefcon(Frontwindow);

      IF Band(Myevent.Modifiers, Cmdkey) <> 0 THEN
        BEGIN
          CASE Hiword(Menukey(Thechar)) OF
          129:
            BEGIN
              IF (Loword(Menukey(Thechar)) = 9) THEN Quit := True;
```

```
                IF (Loword(Menukey(Thechar)) = 7) THEN
                                    Printing(X, Y, Num, scbase, Scfac, Xshift, Yshift);
                IF (Loword(Menukey(Thechar)) = 3) THEN
                   BEGIN
                      IF Cursmode AND (Itemstring = 'Hide Cursor') THEN
                               SaveAsCG(X, Y, Num, XShift, YShift)
                      ELSE Savearray;
                   END;
                IF (Loword(Menukey(Thechar)) = 1) THEN Doarray;
             END;
          135:
             BEGIN
                IF Cursmode OR QMode THEN
                   BEGIN
                      Initcursor;
                      Cursmode := False;
                      Setcurmenu;
                      Qmode := False;
                      SetQMenu;
                      Disposewindow(Mycurwindow);
                   END;
                CASE Loword(Menukey(Thechar)) OF
                   1: DoDynamicAnalysis(Loword(Menukey(Thechar)));
                   2: DoDynamicAnalysis(Loword(Menukey(Thechar)));
                   3: DoSingleQ;
                END;
             END;
          END; {of CASE}
          Hilitemenu(0);
       END
    ELSE IF Cursmode OR Qmode THEN {IF not cmdKey}
       BEGIN
          Obscurecursor;
          Getkeys(Thekeys);
          IF Thekeys[Leftkey] THEN
             BEGIN
                Xshift := Xshift - 1;
                IF Cursmode THEN   Dispcoord(Num, Scbase, X, Y, Scfac, Xshift, Yshift)
                   ELSE IF Qmode THEN  ShowQPlots(Num, X, Y, Xshift, Yshift);
             END
          ELSE IF Thekeys[Rightkey] THEN
             BEGIN
                Xshift := Xshift + 1;
                IF Cursmode THEN   Dispcoord(Num, Scbase, X, Y, Scfac, Xshift, Yshift)
                   ELSE IF Qmode THEN  ShowQPlots(Num, X, Y, Xshift, Yshift);
             END
          ELSE IF Thekeys[Upkey] THEN
             BEGIN
                Yshift := Yshift - 1;
                IF Cursmode THEN   Dispcoord(Num, Scbase, X, Y, Scfac, Xshift, Yshift)
                   ELSE IF Qmode THEN  ShowQPlots(Num, X, Y, Xshift, Yshift);
             END
          ELSE IF Thekeys[Downkey] THEN
             BEGIN
                Yshift := Yshift + 1;
                IF Cursmode THEN   Dispcoord(Num, Scbase, X, Y, Scfac, Xshift, Yshift)
                   ELSE IF Qmode THEN  ShowQPlots(Num, X, Y, Xshift, Yshift);
             END;
       END;
 END;

{-------------------------------------------------------------------------------}
  {$S main}
```

```
{ Executed before quit. }

PROCEDURE CleanUpMess;

   VAR
      onetoclose    : WindowPtr;
      datadispose   : Integer;
      Ourcursorh    : Curshandle;

   BEGIN
      Ourcursorh := Getcursor(4);
      Setcursor(Ourcursorh^^);

      REPEAT
         onetoclose := FrontWindow;
         IF (onetoclose <> nil) THEN    CloseWindow(onetoclose);
      UNTIL (onetoclose = nil);

      FOR datadispose := 0 TO Images DO
         BEGIN
            DisposHandle(OurImages[datadispose].data);
            DisposPixMap(OurImages[datadispose].pixMap);
         END;
      Initcursor;
   END;

{-------------------------------------------------------------------------------}
{-------------------------------------------------------------------------------}
{-------------------------------------------------------------------------------}
{ Main program - initialization and eventloop. }

BEGIN
   MaxApplZone;
   Initgraf(@Theport);                  {initialize QuickDraw}
   Initfonts;                           {initialize Font Manager}
   Flushevents(Everyevent, 0);
   Initwindows;                         {initialize Window Manager}
   Initcursor;                          {call QuickDraw to make cursor (pointer) an arrow}
   Teinit;
   Initdialogs(nil);

   Menuset;
   WITH Screenbits.Bounds DO Setrect(Dragrect, 4, 24, Right - 4, Bottom - 4);

   Quit := False;
   Sorry := False;
   Mymask := Everyevent;
   Images := - 1;                       {This is the index number giving total images opened}
   Cursmode := False;
   Scalemode := False;
   Qmode := False;
   CFmode := False;
   Hismode := False;
   Over := False;
   STImage := False;
   NoOpen := False;
   Type2RData := False;
   Type2IData := False;
   Scfac := 80;
   Scbase := 75;
   Xshift := 0;
   Yshift := 0;
   NI := 0;
```

```
    z := 0;
    DFacter := -9;
    Firstdraw := True;
    FFTNo := 256;
    CurrentArray := 8192;

    Donotice(135);

    WHILE Quit = False DO
        BEGIN
        Systemtask;
        IF Getnextevent(Mymask, Myevent) THEN
            CASE Myevent.What OF
                Mousedown: Domousedown;
                Keydown, Autokey: Dokey;
                Updateevt: Updatewindow;
            END;
        END;

    CleanUpMess;
END.
```

A3.2 ImageShow.r

```
# include "Types.r"


resource 'MENU' (128, "Apple", preload) {
        128,
        textMenuProc,
        0x7FFFFFFD,
        enabled,
        apple,
        {       /* array: 2 elements */
                /* [1] */
                "About ImageShow...", noIcon, "", "", plain,
                /* [2] */
                "-", noIcon, "", "", plain
        }
};

resource 'MENU' (129, "Image", preload) {
        129,
        textMenuProc,
        0x7FFFF101,
        enabled,
        "Image",
        {       /* array: 9 elements */
                /* [1] */
                "Open", noIcon, "O", "", plain,
                /* [2] */
                "-", noIcon, "", "", plain,
                /* [3] */
                "Save as ..", noIcon, "S", "", plain,
                /* [4] */
                "Window PICT Save", noIcon, "", "", plain,
                /* [5] */
                "Screen PICT Save", noIcon, "", "", plain,
                /* [6] */
                "-", noIcon, "", "", plain,
                /* [7] */
                "Print..", noIcon, "P", "", plain,
                /* [8] */
                "-", noIcon, "", "", plain,
                /* [9] */
                "Quit", noIcon, "Q", "", plain
        }
};

resource 'MENU' (130, "Edit", preload) {
        130,
        textMenuProc,
        0x0,
        disabled,
        "Edit",
        {       /* array: 6 elements */
                /* [1] */
                "Undo", noIcon, "Z", "", plain,
                /* [2] */
                "-", noIcon, "", "", plain,
                /* [3] */
                "Cut", noIcon, "X", "", plain,
                /* [4] */
```

```
                "Copy", noIcon, "C", "", plain,
                /* [5] */
                "Paste", noIcon, "V", "", plain,
                /* [6] */
                "Clear", noIcon, "", "", plain
        }
};

resource 'MENU' (131, "Scale", preload) {
        131,
        textMenuProc,
        0x7FFFFDFD,
        disabled,
        "Scale",
        {       /* array: 11 elements */
                /* [1] */
                "Hide Scale", noIcon, "", "", plain,
                /* [2] */
                "-", noIcon, "", "", plain,
                /* [3] */
                "4 Colour", noIcon, "", "", plain,
                /* [4] */
                "8 Colour", noIcon, "", "", plain,
                /* [5] */
                "16 Colour", noIcon, "", "", plain,
                /* [6] */
                "32 Colour", noIcon, "", "", plain,
                /* [7] */
                "Grey Scale", noIcon, "", "", plain,
                /* [8] */
                "Flow Mono", noIcon, "", "", plain,
                /* [9] */
                "Flow 15", noIcon, "", "", plain,
                /* [10] */
                "-", noIcon, "", "", plain,
                /* [11] */
                "Show Controls", noIcon, "", "", plain
        }
};

resource 'MENU' (132, "Cursor", preload) {
        132,
        textMenuProc,
        0x7FFFFFFF,
        disabled,
        "Cursor",
        {       /* array: 4 elements */
                /* [1] */
                "Show Cursor", noIcon, "", "", plain,
                /* [2] */
                "Adjust Slice Scale", noIcon, "", "", plain,
                /* [3] */
                "-", noIcon, "", "", plain,
                /* [4] */
                "Retouch", noIcon, "", "", plain
        }
};

resource 'MENU' (133, "Display", preload) {
        133,
        textMenuProc,
        0x7FFFFFFF,
        disabled,
```

```
        "Display",
        {       /* array: 8 elements */
                /* [1] */
                "Stackplot", noIcon, "", "", plain,
                /* [2] */
                "Freq. Stats.", noIcon, "", "", plain,
                /* [3] */
                "Magn. Stats.", noIcon, "", "", plain,
                /* [4] */
                "Histogram", noIcon, "", "", plain,
                /* [5] */
                "-", noIcon, "", "", plain,
                /* [6] */
                "Non-Square", noIcon, "", "", plain,
                /* [7] */
                "-", noIcon, "", "", plain,
                /* [8] */
                "Show q Plots ", noIcon, "", "", plain
        }
};

resource 'MENU' (134, "Operation", preload) {
        134,
        textMenuProc,
        0x7FFFFFFF,
        disabled,
        "Operation",
        {       /* array: 14 elements */
                /* [1] */
                "Convolution", noIcon, "", "", plain,
                /* [2] */
                "Edge Detection", noIcon, "", "", plain,
                /* [3] */
                "Arithmetic", noIcon, "", "", plain,
                /* [4] */
                "Mathmatical", noIcon, "", "", plain,
                /* [5] */
                "Filters", noIcon, "", "", plain,
                /* [6] */
                "Shifting", noIcon, "", "", plain,
                /* [7] */
                "-", noIcon, "", "", plain,
                /* [8] */
                "Linear Combination", noIcon, "", "", plain,
                /* [9] */
                "Modulus", noIcon, "", "", plain,
                /* [10] */
                "Ratio", noIcon, "", "", plain,
                /* [11] */
                "2D-FT", noIcon, "", "", plain,
                /* [12] */
                "-", noIcon, "", "", plain,
                /* [13] */
                "Radial Averaging", noIcon, "", "", plain,
                /* [14] */
                "Apply_Ramp", noIcon, "", "", plain
        }
};

resource 'MENU' (135, "Multi-slices", preload) {
        135,
        textMenuProc,
        0x7FFFFFF7,
```

```
        enabled,
        "Multi-slices",
        {       /* array: 3 elements */
                /* [1] */
                "Stejskal-Tanner", noIcon, "1", "", plain,
                /* [2] */
                "Fourier Transform", noIcon, "2", "", plain,
                /* [3] */
                "Single-Q Velocity", noIcon, "3", "", plain
        }
};

resource 'clut' (128, "32 colour", purgeable) {
        0x80,
        0,
        {       /* array ColorSpec: 32 elements */
                /* [1] */
                0, 65535, 65535, 65535,
                /* [2] */
                1, 65535, 0, 0,
                /* [3] */
                2, 65535, 13107, 0,
                /* [4] */
                3, 65535, 26214, 0,
                /* [5] */
                4, 65535, 39321, 0,
                /* [6] */
                5, 65535, 52428, 0,
                /* [7] */
                6, 65535, 65535, 0,
                /* [8] */
                7, 52428, 65535, 0,
                /* [9] */
                8, 39321, 65535, 0,
                /* [10] */
                9, 26214, 65535, 0,
                /* [11] */
                10, 13107, 65535, 0,
                /* [12] */
                11, 0, 65535, 0,
                /* [13] */
                12, 0, 65535, 13107,
                /* [14] */
                13, 0, 65535, 26214,
                /* [15] */
                14, 0, 65535, 39321,
                /* [16] */
                15, 0, 65535, 52428,
                /* [17] */
                16, 0, 65535, 65535,
                /* [18] */
                17, 0, 52548, 65535,
                /* [19] */
                18, 0, 39321, 65535,
                /* [20] */
                19, 0, 26214, 65535,
                /* [21] */
                20, 0, 13107, 65535,
                /* [22] */
                21, 0, 0, 65535,
                /* [23] */
                22, 13107, 0, 65535,
                /* [24] */
```

```
                23, 26214, 0, 65535,
                /* [25] */
                24, 39321, 0, 65535,
                /* [26] */
                25, 52428, 0, 65535,
                /* [27] */
                26, 65535, 0, 65535,
                /* [28] */
                27, 52428, 0, 52428,
                /* [29] */
                28, 39321, 0, 39321,
                /* [30] */
                29, 26214, 0, 26214,
                /* [31] */
                30, 13107, 0, 13107,
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'clut' (129, "16 colour", purgeable) {
        0x81,
        0,
        {       /* array ColorSpec: 32 elements */
                /* [1] */
                0, 65535, 65535, 65535,
                /* [2] */
                1, 65535, 65535, 65535,
                /* [3] */
                2, 65535, 0, 0,
                /* [4] */
                3, 65535, 0, 0,
                /* [5] */
                4, 65535, 13107, 0,
                /* [6] */
                5, 65535, 13107, 0,
                /* [7] */
                6, 65535, 26214, 0,
                /* [8] */
                7, 65535, 26214, 0,
                /* [9] */
                8, 65535, 39321, 0,
                /* [10] */
                9, 65535, 39321, 0,
                /* [11] */
                10, 65535, 52428, 0,
                /* [12] */
                11, 65535, 52428, 0,
                /* [13] */
                12, 65535, 65535, 0,
                /* [14] */
                13, 65535, 65535, 0,
                /* [15] */
                14, 0, 65535, 0,
                /* [16] */
                15, 0, 65535, 0,
                /* [17] */
                16, 0, 65535, 32768,
                /* [18] */
                17, 0, 65535, 32768,
                /* [19] */
                18, 0, 65535, 65535,
                /* [20] */
```

```
                19, 0, 65535, 65535,
                /* [21] */
                20, 0, 43690, 65535,
                /* [22] */
                21, 0, 43690, 65535,
                /* [23] */
                22, 0, 21845, 65535,
                /* [24] */
                23, 0, 21845, 65535,
                /* [25] */
                24, 0, 0, 65535,
                /* [26] */
                25, 0, 0, 65535,
                /* [27] */
                26, 0, 0, 43690,
                /* [28] */
                27, 0, 0, 43690,
                /* [29] */
                28, 0, 0, 21845,
                /* [30] */
                29, 0, 0, 21845,
                /* [31] */
                30, 0, 0, 0,
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'clut' (130, "8 colour", purgeable) {
        0x82,
        0,
        {       /* array ColorSpec: 32 elements */
                /* [1] */
                0, 65535, 65535, 65535,
                /* [2] */
                1, 65535, 65535, 65535,
                /* [3] */
                2, 65535, 65535, 65535,
                /* [4] */
                3, 65535, 65535, 65535,
                /* [5] */
                4, 65535, 0, 0,
                /* [6] */
                5, 65535, 0, 0,
                /* [7] */
                6, 65535, 0, 0,
                /* [8] */
                7, 65535, 0, 0,
                /* [9] */
                8, 65535, 65535, 0,
                /* [10] */
                9, 65535, 65535, 0,
                /* [11] */
                10, 65535, 65535, 0,
                /* [12] */
                11, 65535, 65535, 0,
                /* [13] */
                12, 0, 65535, 0,
                /* [14] */
                13, 0, 65535, 0,
                /* [15] */
                14, 0, 65535, 0,
                /* [16] */
```

```
                15, 0, 65535, 0,
                /* [17] */
                16, 0, 32768, 65535,
                /* [18] */
                17, 0, 32768, 65535,
                /* [19] */
                18, 0, 32768, 65535,
                /* [20] */
                19, 0, 32768, 65535,
                /* [21] */
                20, 0, 0, 65535,
                /* [22] */
                21, 0, 0, 65535,
                /* [23] */
                22, 0, 0, 65535,
                /* [24] */
                23, 0, 0, 65535,
                /* [25] */
                24, 13107, 0, 26214,
                /* [26] */
                25, 13107, 0, 26214,
                /* [27] */
                26, 13107, 0, 26214,
                /* [28] */
                27, 13107, 0, 26214,
                /* [29] */
                28, 0, 0, 0,
                /* [30] */
                29, 0, 0, 0,
                /* [31] */
                30, 0, 0, 0,
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'clut' (131, "4 colour", purgeable) {
        0x83,
        0,
        {       /* array ColorSpec: 32 elements */
                /* [1] */
                0, 65535, 65535, 65535,
                /* [2] */
                1, 65535, 65535, 65535,
                /* [3] */
                2, 65535, 65535, 65535,
                /* [4] */
                3, 65535, 65535, 65535,
                /* [5] */
                4, 65535, 65535, 65535,
                /* [6] */
                5, 65535, 65535, 65535,
                /* [7] */
                6, 65535, 65535, 65535,
                /* [8] */
                7, 65535, 65535, 65535,
                /* [9] */
                8, 0, 65535, 0,
                /* [10] */
                9, 0, 65535, 0,
                /* [11] */
                10, 0, 65535, 0,
                /* [12] */
```

```
                11, 0, 65535, 0,
                /* [13] */
                12, 0, 65535, 0,
                /* [14] */
                13, 0, 65535, 0,
                /* [15] */
                14, 0, 65535, 0,
                /* [16] */
                15, 0, 65535, 0,
                /* [17] */
                16, 0, 0, 65535,
                /* [18] */
                17, 0, 0, 65535,
                /* [19] */
                18, 0, 0, 65535,
                /* [20] */
                19, 0, 0, 65535,
                /* [21] */
                20, 0, 0, 65535,
                /* [22] */
                21, 0, 0, 65535,
                /* [23] */
                22, 0, 0, 65535,
                /* [24] */
                23, 0, 0, 65535,
                /* [25] */
                24, 0, 0, 0,
                /* [26] */
                25, 0, 0, 0,
                /* [27] */
                26, 0, 0, 0,
                /* [28] */
                27, 0, 0, 0,
                /* [29] */
                28, 0, 0, 0,
                /* [30] */
                29, 0, 0, 0,
                /* [31] */
                30, 0, 0, 0,
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'clut' (132, "Grey colour", purgeable) {
        0x84,
        0,
        {       /* array ColorSpec: 32 elements */
                /* [1] */
                0, 65535, 65535, 65535,
                /* [2] */
                1, 63000, 63000, 63000,
                /* [3] */
                2, 61000, 61000, 61000,
                /* [4] */
                3, 59000, 59000, 59000,
                /* [5] */
                4, 57000, 57000, 57000,
                /* [6] */
                5, 55000, 55000, 55000,
                /* [7] */
                6, 52000, 52000, 52000,
                /* [8] */
```

```
                7, 50000, 50000, 50000,
                /* [9] */
                8, 48000, 48000, 48000,
                /* [10] */
                9, 46000, 46000, 46000,
                /* [11] */
                10, 44000, 44000, 44000,
                /* [12] */
                11, 42000, 42000, 42000,
                /* [13] */
                12, 40000, 40000, 40000,
                /* [14] */
                13, 38000, 38000, 38000,
                /* [15] */
                14, 36000, 36000, 36000,
                /* [16] */
                15, 34000, 34000, 34000,
                /* [17] */
                16, 32000, 32000, 32000,
                /* [18] */
                17, 30000, 30000, 30000,
                /* [19] */
                18, 28000, 28000, 28000,
                /* [20] */
                19, 26000, 26000, 26000,
                /* [21] */
                20, 24000, 24000, 24000,
                /* [22] */
                21, 22000, 22000, 22000,
                /* [23] */
                22, 20000, 20000, 20000,
                /* [24] */
                23, 18000, 18000, 18000,
                /* [25] */
                24, 16000, 16000,16000,
                /* [26] */
                25, 14000, 14000, 14000,
                /* [27] */
                26, 12000, 12000,12000,
                /* [28] */
                27, 10000, 10000, 10000,
                /* [29] */
                28, 6000, 6000,6000,
                /* [30] */
                29, 4000,4000, 4000,
                /* [31] */
                30, 2000, 2000, 2000,
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'clut' (133, "Flow Mono", purgeable) {
        0x85,
        0,
        {       /* array ColorSpec: 32 elements */
                /* [1] */
                0, 65535, 0, 0,
                /* [2] */
                1, 61235, 0, 0,
                /* [3] */
                2, 56935, 0, 0,
                /* [4] */
```

```
                3, 52635, 0, 0,
                /* [5] */
                4, 48335, 0, 0,
                /* [6] */
                5, 44035, 0, 0,
                /* [7] */
                6, 39735, 0, 0,
                /* [8] */
                7, 35435, 0, 0,
                /* [9] */
                8, 31135, 0, 0,
                /* [10] */
                9, 26835, 0, 0,
                /* [11] */
                10, 22535, 0, 0,
                /* [12] */
                11, 18235, 0, 0,
                /* [13] */
                12, 13935, 0, 0,
                /* [14] */
                13, 9635, 0, 0,
                /* [15] */
                14, 5335, 0, 0,
                /* [16] */
                15, 0, 0, 0,
                /* [17] */
                16, 0, 0, 5300,
                /* [18] */
                17, 0, 0, 9600,
                /* [19] */
                18, 0, 0, 13900,
                /* [20] */
                19, 0, 0, 18200,
                /* [21] */
                20, 0, 0, 22500,
                /* [22] */
                21, 0, 0, 26800,
                /* [23] */
                22, 0, 0, 31100,
                /* [24] */
                23, 0, 0, 35400,
                /* [25] */
                24, 0, 0,39700,
                /* [26] */
                25, 0, 0, 44000,
                /* [27] */
                26, 0, 0,48300,
                /* [28] */
                27, 0, 0, 52600,
                /* [29] */
                28, 0, 0,56900,
                /* [30] */
                29, 0,0, 61200,
                /* [31] */
                30, 0, 0, 65500,
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'clut' (134, "Flow15", purgeable) {
        0x86,
        0,
```

```
{       /* array ColorSpec: 32 elements */
        /* [1] */
        0, 65535, 65535, 65535,
        /* [2] */
        1, 65535, 65535, 65535,
        /* [3] */
        2, 65535, 0, 0,
        /* [4] */
        3, 65535, 0, 0,
        /* [5] */
        4, 65535, 21845, 0,
        /* [6] */
        5, 65535, 21845, 0,
        /* [7] */
        6, 65535, 43690, 0,
        /* [8] */
        7, 65535, 43690, 0,
        /* [9] */
        8, 65535, 65535, 0,
        /* [10] */
        9, 65535, 65535, 0,
        /* [11] */
        10, 0, 65535, 0,
        /* [12] */
        11, 0, 65535, 0,
        /* [13] */
        12, 0, 26214, 0,
        /* [14] */
        13, 0, 26214, 0,
        /* [15] */
        14, 0, 0, 0,
        /* [16] */
        15, 0, 0, 0,
        /* [17] */
        16, 0, 0, 32768,
        /* [18] */
        17, 0, 0, 32768,
        /* [19] */
        18, 0, 0, 65535,
        /* [20] */
        19, 0, 0, 65535,
        /* [21] */
        20, 0, 21845, 65535,
        /* [22] */
        21, 0, 21845, 65535,
        /* [23] */
        22, 0, 43960, 65535,
        /* [24] */
        23, 0, 43960, 65535,
        /* [25] */
        24, 0, 65535,65535,
        /* [26] */
        25, 0, 65535, 65535,
        /* [27] */
        26, 65535, 36121,65535,
        /* [28] */
        27, 65535, 36121, 65535,
        /* [29] */
        28, 43690, 0,43960,
        /* [30] */
        29, 43690,0, 43960,
        /* [31] */
        30, 0, 0, 0,
```

```
                /* [32] */
                31, 0, 0, 0
        }
};

resource 'pltt' (128) {
        {       /* array ColorInfo: 91 elements */
                /* [1] */
                65535, 65535, 65535, pmTolerant, 0,
                /* [2] */
                0, 0, 0, pmTolerant, 0,
                /* [3] */
                65535, 0, 0, pmTolerant, 0,
                /* [4] */
                65535, 13107, 0, pmTolerant, 0,
                /* [5] */
                65535, 26214, 0, pmTolerant, 0,
                /* [6] */
                65535, 39321, 0, pmTolerant, 0,
                /* [7] */
                65535, 52428, 0, pmTolerant, 0,
                /* [8] */
                65535, 65535, 0, pmTolerant, 0,
                /* [9] */
                52428, 65535, 0, pmTolerant, 0,
                /* [10] */
                39321, 65535, 0, pmTolerant, 0,
                /* [11] */
                26214, 65535, 0, pmTolerant, 0,
                /* [12] */
                13107, 65535, 0, pmTolerant, 0,
                /* [13] */
                0, 65535, 0, pmTolerant, 0,
                /* [14] */
                0, 65535, 13107, pmTolerant, 0,
                /* [15] */
                0, 65535, 26214, pmTolerant, 0,
                /* [16] */
                0, 65535, 39321, pmTolerant, 0,
                /* [17] */
                0, 65535, 52428, pmTolerant, 0,
                /* [18] */
                0, 65535, 65535, pmTolerant, 0,
                /* [19] */
                0, 52548, 65535, pmTolerant, 0,
                /* [20] */
                0, 39321, 65535, pmTolerant, 0,
                /* [21] */
                0, 26214, 65535, pmTolerant, 0,
                /* [22] */
                0, 13107, 65535, pmTolerant, 0,
                /* [23] */
                13107, 0, 65535, pmTolerant, 0,
                /* [24] */
                26214, 0, 65535, pmTolerant, 0,
                /* [25] */
                39321, 0, 65535, pmTolerant, 0,
                /* [26] */
                52428, 0, 65535, pmTolerant, 0,
                /* [27] */
                65535, 0, 65535, pmTolerant, 0,
                /* [28] */
                52428, 0, 52428, pmTolerant, 0,
```

```
/* [29] */
39321, 0, 39321, pmTolerant, 0,
/* [30] */
26214, 0, 26214, pmTolerant, 0,
/* [31] */
13107, 0, 13107, pmTolerant, 0,
/* [32] */
65535, 65535, 65535, pmTolerant, 0,
/* [33] */
63000, 63000, 63000, pmTolerant, 0,
/* [34] */
61000, 61000, 61000, pmTolerant, 0,
/* [35] */
59000, 59000, 59000, pmTolerant, 0,
/* [36] */
57000, 57000, 57000, pmTolerant, 0,
/* [37] */
55000, 55000, 55000, pmTolerant, 0,
/* [38] */
52000, 52000, 52000, pmTolerant, 0,
/* [39] */
50000, 50000, 50000, pmTolerant, 0,
/* [40] */
48000, 48000, 48000, pmTolerant, 0,
/* [41] */
46000, 46000, 46000, pmTolerant, 0,
/* [42] */
44000, 44000, 44000, pmTolerant, 0,
/* [43] */
42000, 42000, 42000, pmTolerant, 0,
/* [44] */
40000, 40000, 40000, pmTolerant, 0,
/* [45] */
38000, 38000, 38000, pmTolerant, 0,
/* [46] */
36000, 36000, 36000, pmTolerant, 0,
/* [47] */
34000, 34000, 34000, pmTolerant, 0,
/* [48] */
32000, 32000, 32000, pmTolerant, 0,
/* [49] */
30000, 30000, 30000, pmTolerant, 0,
/* [50] */
28000, 28000, 28000, pmTolerant, 0,
/* [51] */
26000, 26000, 26000, pmTolerant, 0,
/* [52] */
24000, 24000, 24000, pmTolerant, 0,
/* [53] */
22000, 22000, 22000, pmTolerant, 0,
/* [54] */
20000, 20000, 20000, pmTolerant, 0,
/* [55] */
18000, 18000, 18000, pmTolerant, 0,
/* [56] */
16000, 16000, 16000, pmTolerant, 0,
/* [57] */
14000, 14000, 14000, pmTolerant, 0,
/* [58] */
12000, 12000, 12000, pmTolerant, 0,
/* [59] */
10000, 10000, 10000, pmTolerant, 0,
/* [60] */
```

```
6000, 6000, 6000, pmTolerant, 0,
/* [61] */
4000, 4000, 4000, pmTolerant, 0,
/* [62] */
2000, 2000, 2000, pmTolerant, 0,
/* [63] */
61235, 0, 0, pmTolerant, 0,
/* [64] */
56935, 0, 0, pmTolerant, 0,
/* [65] */
52635, 0, 0, pmTolerant, 0,
/* [66] */
48335, 0, 0, pmTolerant, 0,
/* [67] */
44035, 0, 0, pmTolerant, 0,
/* [68] */
39735, 0, 0, pmTolerant, 0,
/* [69] */
35435, 0, 0, pmTolerant, 0,
/* [70] */
31135, 0, 0, pmTolerant, 0,
/* [71] */
26835, 0, 0, pmTolerant, 0,
/* [72] */
22535, 0, 0, pmTolerant, 0,
/* [73] */
18235, 0, 0, pmTolerant, 0,
/* [74] */
13935, 0, 0, pmTolerant, 0,
/* [75] */
9635, 0, 0, pmTolerant, 0,
/* [76] */
5335, 0, 0, pmTolerant, 0,
/* [77] */
0, 0, 5300, pmTolerant, 0,
/* [78] */
0, 0, 9600, pmTolerant, 0,
/* [79] */
0, 0, 13900, pmTolerant, 0,
/* [80] */
0, 0, 18200, pmTolerant, 0,
/* [81] */
0, 0, 22500, pmTolerant, 0,
/* [82] */
0, 0, 26800, pmTolerant, 0,
/* [83] */
0, 0, 31100, pmTolerant, 0,
/* [84] */
0, 0, 35400, pmTolerant, 0,
/* [85] */
0, 0, 39700, pmTolerant, 0,
/* [86] */
0, 0, 44000, pmTolerant, 0,
/* [87] */
0, 0, 48300, pmTolerant, 0,
/* [88] */
0, 0, 52600, pmTolerant, 0,
/* [89] */
0, 0, 56900, pmTolerant, 0,
/* [90] */
0, 0, 61200, pmTolerant, 0,
/* [91] */
0, 0, 65500, pmTolerant, 0
```

```
        }
};

resource 'DLOG' (129, "Controls") {
        {416, 36, 470, 342},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        129,
        "Controls"
};

resource 'DLOG' (130, "(Counter??)") {
        {122, 110, 286, 536},
        dBoxProc,
        visible,
        goAway,
        0x0,
        130,
        "(Counter??)"
};

resource 'DLOG' (131, "Linear Comb") {
        {348, 12, 468, 230},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        131,
        "Linear Combination"
};

resource 'DLOG' (132, "Adjust Slice") {
        {382, 488, 468, 630},
        plainDBox,
        visible,
        noGoAway,
        0x0,
        132,
        "Adjust Slice"
};

resource 'DLOG' (133, "(array size)") {
        {78, 98, 320, 534},
        dBoxProc,
        visible,
        goAway,
        0x0,
        133,
        "N(array size)"
};

resource 'DLOG' (134, "(too many)") {
        {110, 178, 258, 402},
        dBoxProc,
        visible,
        goAway,
        0x0,
        134,
        "(too many)"
};
```

```
resource 'DLOG' (135, "(copyright)") {
        {72, 144, 350, 468},
        dBoxProc,
        visible,
        goAway,
        0x0,
        135,
        "(copyright)"
};

resource 'DLOG' (136, "(About Me..)") {
        {76, 110, 272, 524},
        altDBoxProc,
        visible,
        noGoAway,
        0x0,
        136,
        "(About Me..)"
};

resource 'DLOG' (137, "overflow") {
        {88, 118, 304, 522},
        dBoxProc,
        visible,
        goAway,
        0x0,
        137,
        "overflow"
};

resource 'DLOG' (138, "Convolution") {
        {264, 456, 464, 620},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        138,
        "Convolve"
};

resource 'DLOG' (139, "Edge Detection") {
        {386, 436, 468, 626},
        dBoxProc,
        visible,
        goAway,
        0x0,
        139,
        "Edge_Detection"
};

resource 'DLOG' (140, "printing") {
        {122, 142, 250, 470},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        140,
        "printing"
};

resource 'DLOG' (141, "PrOption") {
        {126, 188, 304, 422},
        dboxProc,
```

```
        visible,
        noGoAway,
        0x0,
        141,
        "PrOption"
};

resource 'DLOG' (142, "ST") {
        {340, 14, 466, 212},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        142,
        "ST"
};

resource 'DLOG' (143, "Retouch") {
        {330, 336, 464, 624},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        143,
        "Retouch"
};

resource 'DLOG' (144, "Radial Average") {
        {122, 184, 270, 438},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        144,
        "Radial Average"
};

resource 'DLOG' (145, "Modulus") {
        {108, 146, 266, 478},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        145,
        "Modulus"
};

resource 'DLOG' (146, "DataArith") {
        {96, 184, 348, 448},
        dBoxProc,
        visible,
        nogoAway,
        0x0,
        146,
        "DataArith"
};

resource 'DLOG' (147, "Over18slice") {
        {98, 158, 284, 476},
        dBoxProc,
        visible,
        goAway,
        0x0,
```

```
        147,
        "Over18slice"
};

resource 'DLOG' (149, "(Requires 2)") {
        {98, 164, 240, 462},
        dBoxProc,
        visible,
        goAway,
        0x0,
        149,
        "(Requires 2)"
};

resource 'DLOG' (150, "ReadDynamic") {
        {52, 78, 422, 562},
        dBoxProc,
        visible,
        goAway,
        0x0,
        150,
        "ReadDynamic"
};


resource 'DLOG' (151, "(ReadD)") {
        {106, 154, 262, 474},
        dBoxProc,
        visible,
        goAway,
        0x0,
        151,
        "(ReadD)"
};

resource 'DLOG' (152, "Ratio") {
        {62, 58, 358, 586},
        dBoxProc,
        visible,
        goAway,
        0x0,
        152,
        "Ratio"
};

resource 'DLOG' (153, "Filter") {
        {70, 144, 318, 496},
        dBoxProc,
        visible,
        goAway,
        0x0,
        153,
        "Filter"
};

resource 'DLOG' (154, "(Not in folder)") {
        {98, 158, 284, 476},
        dBoxProc,
        visible,
        goAway,
        0x0,
        154,
        "(Not in folder)"
```

```
};

resource 'DLOG' (155, "(Not64128)") {
        {104, 126, 266, 512},
        dBoxProc,
        visible,
        goAway,
        0x0,
        155,
        "(Not64128)"
};

resource 'DLOG' (156, "Shifting") {
        {116, 154, 324, 490},
        dBoxProc,
        visible,
        goAway,
        0x0,
        156,
        "Shifting"
};

resource 'DLOG' (157, "Mathmatical") {
        {90, 132, 306, 520},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        157,
        "Mathmatical"
};

resource 'DLOG' (158, "SingleQ") {
        {72, 70, 356, 574},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        158,
        "SingleQ"
};

resource 'DLOG' (159, "Ramping") {
        {86, 110, 362, 534},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        159,
        "Ramping"
};

resource 'DLOG' (160, "Input2DFT") {
        {86, 146, 280, 484},
        dBoxProc,
        visible,
        goAway,
        0x0,
        160,
        "Input2DFT"
};

resource 'DITL' (129) {
```

```
        {       /* array DITLarray: 10 elements */
                /* [1] */
                {3, 236, 23, 296},
                Button {
                        enabled,
                        "Apply.."
                },
                /* [2] */
                {31, 236, 51, 296},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {0, 0, 32, 32},
                Icon {
                        enabled,
                        129
                },
                /* [4] */
                {0, 109, 32, 141},
                Icon {
                        enabled,
                        129
                },
                /* [5] */
                {30, 0, 62, 32},
                Icon {
                        enabled,
                        130
                },
                /* [6] */
                {30, 109, 62, 141},
                Icon {
                        enabled,
                        130
                },
                /* [7] */
                {2, 115, 18, 195},
                StaticText {
                        disabled,
                        "Upper limit:"
                },
                /* [8] */
                {2, 2, 18, 86},
                StaticText {
                        disabled,
                        "Lower limit:"
                },
                /* [9] */
                {23, 35, 39, 83},
                EditText {
                        disabled,
                        "0"
                },
                /* [10] */
                {23, 144, 39, 192},
                EditText {
                        disabled,
                        "32767"
                }
        }
};
```

```
resource 'DITL' (130) {
	{	/* array DITLarray: 3 elements */
		/* [1] */
		{111, 308, 137, 381},
		Button {
			enabled,
			"OK"
		},
		/* [2] */
		{22, 25, 80, 70},
		Icon {
			disabled,
			1
		},
		/* [3] */
		{29, 88, 78, 401},
		StaticText {
			disabled,
			"Counter is overflowing.....\n(In this ver"
			"sion of ImageShow™, the maximum counter "
			"size in statistical rountines is 1000)"
		}
	}
};

resource 'DITL' (131) {
	{	/* array DITLarray: 7 elements */
		/* [1] */
		{86, 125, 106, 185},
		Button {
			enabled,
			"Apply"
		},
		/* [2] */
		{86, 41, 106, 101},
		Button {
			enabled,
			"Cancel"
		},
		/* [3] */
		{7, 20, 23, 52},
		EditText {
			disabled,
			" 100"
		},
		/* [4] */
		{7, 55, 23, 190},
		StaticText {
			disabled,
			"%  * Current Image"
		},
		/* [5] */
		{28, 120, 45, 140},
		StaticText {
			disabled,
			"+"
		},
		/* [6] */
		{50, 20, 66, 52},
		EditText {
			disabled,
			"-100"
```

```
                },
                /* [7] */
                {50, 55, 66, 190},
                StaticText {
                        disabled,
                        "%  * Previous Image"
                }
        }
};

resource 'DITL' (132) {
        {       /* array DITLarray: 10 elements */
                /* [1] */
                {0, 45, 32, 93},
                Icon {
                        enabled,
                        129
                },
                /* [2] */
                {0, 0, 32, 48},
                Icon {
                        enabled,
                        129
                },
                /* [3] */
                {2, 3, 18, 46},
                StaticText {
                        disabled,
                        "Y Gain"
                },
                /* [4] */
                {2, 54, 18, 90},
                StaticText {
                        disabled,
                        "Base"
                },
                /* [5] */
                {30, 0, 64, 48},
                Icon {
                        enabled,
                        130
                },
                /* [6] */
                {30, 45, 64, 93},
                Icon {
                        enabled,
                        130
                },
                /* [7] */
                {0, 92, 33, 140},
                Icon {
                        enabled,
                        129
                },
                /* [8] */
                {33, 92, 65, 140},
                Icon {
                        enabled,
                        130
                },
                /* [9] */
                {2, 98, 18, 141},
                StaticText {
```

```
                        disabled,
                        "X Gain"
                },
                /* [10] */
                {63, 49, 80, 91},
                Button {
                        enabled,
                        "Exit"
                }
        }
};

resource 'DITL' (133) {
        {       /* array DITLarray: 5 elements */
                /* [1] */
                {199, 333, 226, 404},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {94, 36, 134, 407},
                StaticText {
                        disabled,
                        "This version of ImageShow™ supports only"
                        " square data arrays of size 64x64, 128x1"
                        "28 or 256x256. "
                },
                /* [3] */
                {146, 36, 180, 407},
                StaticText {
                        disabled,
                        "The requirement of displaying other data"
                        " dimensions is possible by writing to th"
                        "e authors."
                },
                /* [4] */
                {32, 79, 50, 408},
                StaticText {
                        disabled,
                        "SORRY!! YOUR DATA ARRAY SIZE IS NOT SUPP"
                        "ORTED.."
                },
                /* [5] */
                {17, 27, 65, 68},
                Icon {
                        disabled,
                        1
                }
        }
};

resource 'DITL' (134, "Too many") {
        {       /* array DITLarray: 2 elements */
                /* [1] */
                {106, 146, 124, 194},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {14, 20, 82, 206},
                StaticText {
```

```
                        disabled,
                        "You have opened too many windows! Would "
                        "you like\na big rest? To be safe, quit\nan"
                        "d start again."
                }
        }
};

resource 'DITL' (135) {
        {       /* array DITLarray: 10 elements */
                /* [1] */
                {205, 263, 256, 303},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {22, 102, 39, 220},
                StaticText {
                        disabled,
                        "ImageShow™ 4.0"
                },
                /* [3] */
                {248, 92, 268, 235},
                StaticText {
                        disabled,
                        "Demonstration Copy!"
                },
                /* [4] */
                {60, 77, 78, 251},
                StaticText {
                        disabled,
                        "August 1991, Copyright!"
                },
                /* [5] */
                {96, 108, 112, 212},
                StaticText {
                        disabled,
                        "NMR Research :\n "
                },
                /* [6] */
                {115, 33, 132, 286},
                StaticText {
                        disabled,
                        "Department of Physics and Biophysics"
                },
                /* [7] */
                {142, 35, 158, 284},
                StaticText {
                        disabled,
                        "Massey University, Palmerston North"
                },
                /* [8] */
                {165, 115, 180, 204},
                StaticText {
                        disabled,
                        "NEW ZEALAND\n"
                },
                /* [9] */
                {194, 93, 208, 228},
                StaticText {
                        disabled,
                        "Fax : 64-6-3505613"
```

```
                },
                /* [10] */
                {213, 98, 229, 220},
                StaticText {
                        disabled,
                        "Tel: 64-6-3569099"
                }
        }
};

resource 'DITL' (136, "About Me..") {
        {       /* array DITLarray: 4 elements */
                /* [1] */
                {159, 272, 184, 375},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {12, 22, 62, 398},
                StaticText {
                        disabled,
                        "ImageShow™ is written in MPW Pascal for "
                        "Macintosh II. It can be used to display "
                        "and analyze images with the file type of"
                        " 'IMGE'."
                },
                /* [3] */
                {66, 22, 101, 399},
                StaticText {
                        disabled,
                        "More information can be obtained from th"
                        "e address in the copyright statement. \n"
                },
                /* [4] */
                {119, 149, 138, 400},
                StaticText {
                        disabled,
                        "Yang Xia, Andrew Coy, Paul Callaghan"
                }
        }
};

resource 'DITL' (137, "Overflow") {
        {       /* array DITLarray: 8 elements */
                /* [1] */
                {166, 283, 208, 377},
                Button {
                        enabled,
                        "Do it!"
                },
                /* [2] */
                {177, 156, 196, 260},
                Button {
                        enabled,
                        "Don't worry,"
                },
                /* [3] */
                {12, 13, 123, 55},
                Icon {
                        disabled,
                        2
                },
```

```
                /* [4] */
                {27, 81, 45, 367},
                StaticText {
                        disabled,
                        "Sorry, your data is over the ±32767 rang"
                        "e,"
                },
                /* [5] */
                {77, 82, 95, 343},
                StaticText {
                        disabled,
                        "Do you want to scale them down using "
                },
                /* [6] */
                {46, 101, 62, 329},
                StaticText {
                        disabled,
                        "[Max = ?]"
                },
                /* [7] */
                {96, 102, 112, 296},
                StaticText {
                        disabled,
                        "2**NG?"
                },
                /* [8] */
                {114, 81, 147, 387},
                StaticText {
                        disabled,
                        "[the scaling exponent would be saved as "
                        "the first pixel of both real and imagina"
                        "ry images] "
                }
        }
};

resource 'DITL' (138) {
        {       /* array DITLarray: 12 elements */
                /* [1] */
                {166, 98, 186, 158},
                Button {
                        enabled,
                        "Apply"
                },
                /* [2] */
                {166, 9, 186, 69},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {87, 18, 107, 48},
                EditText {
                        disabled,
                        "5"
                },
                /* [4] */
                {117, 18, 137, 48},
                EditText {
                        disabled,
                        "5"
                },
                /* [5] */
```

```
                {57, 68, 77, 98},
                EditText {
                        disabled,
                        "5"
                },
                /* [6] */
                {87, 68, 107, 98},
                EditText {
                        disabled,
                        "60"
                },
                /* [7] */
                {117, 68, 137, 98},
                EditText {
                        disabled,
                        "5"
                },
                /* [8] */
                {57, 118, 77, 148},
                EditText {
                        disabled,
                        "5"
                },
                /* [9] */
                {87, 118, 107, 148},
                EditText {
                        disabled,
                        "5"
                },
                /* [10] */
                {117, 118, 137, 148},
                EditText {
                        disabled,
                        "5"
                },
                /* [11] */
                {57, 18, 77, 48},
                EditText {
                        disabled,
                        "5"
                },
                /* [12] */
                {13, 15, 31, 150},
                StaticText {
                        disabled,
                        "Convolution Kernel"
                }
        }
};

resource 'DITL' (139) {
        {       /* array DITLarray: 6 elements */
                /* [1] */
                {56, 113, 76, 173},
                Button {
                        enabled,
                        "Apply"
                },
                /* [2] */
                {56, 21, 76, 81},
                Button {
                        enabled,
                        "Cancel"
```

```
                },
                /* [3] */
                {7, 135, 22, 178},
                EditText {
                        disabled,
                        "5000"
                },
                /* [4] */
                {7, 12, 23, 113},
                StaticText {
                        disabled,
                        "Edge size"
                },
                /* [5] */
                {29, 12, 45, 127},
                StaticText {
                        disabled,
                        "Neighbour range"
                },
                /* [6] */
                {30, 135, 44, 178},
                EditText {
                        disabled,
                        "2"
                }
        }
};

resource 'DITL' (140) {
        {       /* array DITLarray: 4 elements */
                /* [1] */
                {30, 78, 47, 301},
                StaticText {
                        disabled,
                        "Printing will take ~ 20 minutes!"
                },
                /* [2] */
                {70, 208, 89, 283},
                Button {
                        enabled,
                        "Continue"
                },
                /* [3] */
                {70, 104, 89, 179},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [4] */
                {17, 18, 65, 65},
                Icon {
                        enabled,
                        2
                }
        }
};

resource 'DITL' (141) {
        {       /* array DITLarray: 5 elements */
                /* [1] */
                {125, 129, 145, 189},
                Button {
                        enabled,
```

```
                "Normal"
            },
            /* [2] */
            {85, 129, 105, 189},
            Button {
                enabled,
                "Vector"
            },
            /* [3] */
            {36, 96, 55, 215},
            StaticText {
                disabled,
                "Print slice option:"
            },
            /* [4] */
            {18, 22, 73, 76},
            Icon {
                disabled,
                1
            },
            /* [5] */
            {125, 39, 145, 99},
            Button {
                enabled,
                "Cancel"
            }
        }
};

resource 'DITL' (142, "ST") {
        {   /* array DITLarray: 12 elements */
            /* [1] */
            {103, 116, 121, 175},
            Button {
                enabled,
                "Plot"
            },
            /* [2] */
            {103, 26, 121, 86},
            Button {
                enabled,
                "Cancel"
            },
            /* [3] */
            {3, 3, 21, 141},
            StaticText {
                disabled,
                "Maximum Plot Scale:"
            },
            /* [4] */
            {3, 145, 21, 190},
            EditText {
                enabled,
                "32767"
            },
            /* [5] */
            {31, 3, 47, 41},
            StaticText {
                disabled,
                "Data:"
            },
            /* [6] */
            {31, 45, 47, 118},
```

```
                RadioButton {
                        enabled,
                        "Direct"
                },
                /* [7] */
                {31, 122, 47, 194},
                RadioButton {
                        enabled,
                        "Squared"
                },
                /* [8] */
                {67, 3, 84, 74},
                StaticText {
                        disabled,
                        "Plot from:"
                },
                /* [9] */
                {53, 82, 75, 139},
                RadioButton {
                        enabled,
                        "Front"
                },
                /* [10] */
                {53, 142, 75, 194},
                RadioButton {
                        enabled,
                        "Back"
                },
                /* [11] */
                {78, 82, 99, 132},
                RadioButton {
                        enabled,
                        "Left"
                },
                /* [12] */
                {78, 142, 99, 198},
                RadioButton {
                        enabled,
                        "Right"
                }
        }
};

resource 'DITL' (143, "Retouch") {
        {       /* array DITLarray: 12 elements */
                /* [1] */
                {99, 97, 121, 184},
                Button {
                        enabled,
                        "Check value"
                },
                /* [2] */
                {99, 13, 121, 68},
                Button {
                        enabled,
                        "Exit"
                },
                /* [3] */
                {30, 13, 47, 34},
                StaticText {
                        disabled,
                        "X="
                },
```

```
                /* [4] */
                {30, 43, 47, 70},
                EditText {
                        disabled,
                        "0"
                },
                /* [5] */
                {59, 13, 75, 36},
                StaticText {
                        disabled,
                        "Y="
                },
                /* [6] */
                {59, 43, 75, 70},
                EditText {
                        disabled,
                        "0"
                },
                /* [7] */
                {56, 97, 72, 210},
                StaticText {
                        disabled,
                        "Change Value to:"
                },
                /* [8] */
                {57, 217, 72, 268},
                EditText {
                        disabled,
                        "?"
                },
                /* [9] */
                {3, 64, 21, 220},
                StaticText {
                        disabled,
                        "Modifying image pixels"
                },
                /* [10] */
                {33, 97, 50, 196},
                StaticText {
                        disabled,
                        "Original value:"
                },
                /* [11] */
                {33, 217, 49, 268},
                EditText {
                        disabled,
                        "?"
                },
                /* [12] */
                {99, 192, 121, 283},
                Button {
                        enabled,
                        "Make change"
                }
        }
};

resource 'DITL' (144, "Radial Average") {
        {       /* array DITLarray: 11 elements */
                /* [1] */
                {99, 185, 117, 237},
                Button {
                        enabled,
```

```
            "Apply"
        },
        /* [2] */
        {45, 185, 64, 237},
        Button {
            enabled,
            "Cancel"
        },
        /* [3] */
        {4, 34, 22, 225},
        StaticText {
            disabled,
            "Enter Image Pixel Boudaries"
        },
        /* [4] */
        {35, 16, 50, 58},
        StaticText {
            disabled,
            "Top = "
        },
        /* [5] */
        {60, 16, 75, 61},
        StaticText {
            disabled,
            "Left ="
        },
        /* [6] */
        {85, 16, 100, 80},
        StaticText {
            disabled,
            "Bottom ="
        },
        /* [7] */
        {110, 16, 125, 67},
        StaticText {
            disabled,
            "Right ="
        },
        /* [8] */
        {35, 85, 50, 113},
        EditText {
            disabled,
            "35"
        },
        /* [9] */
        {60, 85, 75, 113},
        EditText {
            disabled,
            "35"
        },
        /* [10] */
        {85, 85, 100, 113},
        EditText {
            disabled,
            "90"
        },
        /* [11] */
        {110, 85, 125, 113},
        EditText {
            disabled,
            "90"
        }
}
```

```
};

resource 'DITL' (145, "Modulus") {
	{	/* array DITLarray: 3 elements */
		/* [1] */
		{107, 208, 127, 268},
		Button {
			enabled,
			"Apply"
		},
		/* [2] */
		{107, 63, 127, 123},
		Button {
			enabled,
			"Cancel"
		},
		/* [3] */
		{8, 10, 85, 324},
		StaticText {
			disabled,
			"This routine will produce an image using"
			"\n                              the fol"
			"lowing operation:\n    \n(Current Image**2"
			" + Previous Image**2)**1/2"
		}
	}
};

resource 'DITL' (146, "DataArith") {
	{	/* array DITLarray: 15 elements */
		/* [1] */
		{203, 188, 223, 248},
		Button {
			enabled,
			"Apply"
		},
		/* [2] */
		{151, 188, 171, 248},
		Button {
			enabled,
			"Cancel"
		},
		/* [3] */
		{7, 41, 24, 223},
		StaticText {
			disabled,
			"Data Arithmetic Operations"
		},
		/* [4] */
		{40, 20, 60, 98},
		RadioButton {
			enabled,
			"Absolute"
		},
		/* [5] */
		{65, 20, 85, 95},
		RadioButton {
			enabled,
			"Square"
		},
		/* [6] */
		{90, 20, 110, 120},
		RadioButton {
```

```
                enabled,
                "Square Root"
            },
            /* [7] */
            {115, 20, 135, 160},
            RadioButton {
                enabled,
                "Reverse"
            },
            /* [8] */
            {140, 20, 160, 71},
            RadioButton {
                enabled,
                "Plus"
            },
            /* [9] */
            {165, 20, 185, 80},
            RadioButton {
                enabled,
                "Minus"
            },
            /* [10] */
            {190, 20, 210, 94},
            RadioButton {
                enabled,
                "Multiply"
            },
            /* [11] */
            {215, 20, 235, 80},
            RadioButton {
                enabled,
                "Divide"
            },
            /* [12] */
            {142, 100, 158, 150},
            EditText {
                disabled,
                "0"
            },
            /* [13] */
            {167, 100, 183, 150},
            EditText {
                disabled,
                "0"
            },
            /* [14] */
            {192, 100, 208, 150},
            EditText {
                disabled,
                "1"
            },
            /* [15] */
            {217, 100, 233, 150},
            EditText {
                enabled,
                "1"
            }
        }
};

resource 'DITL' (147) {
        {       /* array DITLarray: 3 elements */
                /* [1] */
```

```
                {143, 185, 169, 261},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {28, 36, 96, 85},
                Icon {
                        disabled,
                        1
                },
                /* [3] */
                {39, 119, 89, 297},
                StaticText {
                        disabled,
                        "Maximum q slices are limited to 18 in "
                        " this version of ImageShow™."
                }
        }
};

resource 'DITL' (149, "Requires Two") {
        {       /* array DITLarray: 2 elements */
                /* [1] */
                {99, 189, 122, 251},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {21, 38, 74, 262},
                StaticText {
                        disabled,
                        "Sorry, this routine requires two images"
                        " with the same data sizes."
                }
        }
};

resource 'DITL' (150) {
        {       /* array DITLarray: 28 elements */
                /* [1] */
                {255, 383, 315, 438},
                Button {
                        enabled,
                        "Ready"
                },
                /* [2] */
                {189, 379, 209, 439},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {35, 21, 70, 471},
                StaticText {
                        disabled,
                        "Please leave the data images in the same"
                        " folder as the ImageShow application, an"
                        "d prepare the data image names in the fo"
                        "rmat as:"
                },
                /* [4] */
```

```
{71, 126, 106, 469},
StaticText {
        disabled,
        "Real ones:          Name.1.R, Name.2.R,"
        " Name.3.R, ...\nImaginary ones: Name.1.I,"
        " Name.2.I, Name.3.I, ..."
},
/* [5] */
{8, 23, 27, 467},
StaticText {
        disabled,
        "***** This routine analyzes dynamic imag"
        "ing data in q-space*****\n"
},
/* [6] */
{253, 41, 272, 181},
StaticText {
        disabled,
        "No. of q image pairs:"
},
/* [7] */
{255, 188, 270, 210},
EditText {
        disabled,
        "18"
},
/* [8] */
{280, 41, 298, 168},
StaticText {
        disabled,
        "Analyze Threshold:"
},
/* [9] */
{280, 188, 296, 224},
EditText {
        disabled,
        "5000"
},
/* [10] */
{203, 42, 219, 152},
StaticText {
        disabled,
        "PGSE Duration ∂:"
},
/* [11] */
{205, 188, 221, 242},
EditText {
        disabled,
        "2000"
},
/* [12] */
{179, 42, 195, 168},
StaticText {
        disabled,
        "PGSE Separation Δ:"
},
/* [13] */
{181, 188, 196, 243},
EditText {
        disabled,
        "5000"
},
/* [14] */
```

```
{154, 42, 172, 172},
StaticText {
        disabled,
        "Max PGSE Gradient:"
},
/* [15] */
{155, 188, 170, 227},
EditText {
        disabled,
        "936"
},
/* [16] */
{155, 231, 172, 333},
StaticText {
        disabled,
        "(0.1 Gauss/cm)"
},
/* [17] */
{181, 246, 198, 279},
StaticText {
        disabled,
        "(μs)"
},
/* [18] */
{205, 246, 221, 279},
StaticText {
        disabled,
        "(μs)"
},
/* [19] */
{127, 42, 145, 179},
StaticText {
        disabled,
        "q Data Image Name:"
},
/* [20] */
{127, 188, 142, 288},
EditText {
        disabled,
        "Sample"
},
/* [21] */
{228, 42, 245, 163},
StaticText {
        disabled,
        "No. of PGSE pairs:"
},
/* [22] */
{230, 188, 245, 211},
EditText {
        disabled,
        "1"
},
/* [23] */
{308, 41, 327, 172},
StaticText {
        disabled,
        "Estimated D range:"
},
/* [24] */
{311, 188, 327, 215},
EditText {
        disabled,
```

```
                "-9"
            },
            /* [25] */
            {311, 220, 329, 333},
            StaticText {
                disabled,
                "(th power of 10)"
            },
            /* [26] */
            {336, 41, 354, 164},
            StaticText {
                disabled,
                "Array size of FFT:"
            },
            /* [27] */
            {338, 188, 355, 221},
            EditText {
                disabled,
                "256"
            },
            /* [28] */
            {338, 227, 355, 328},
            StaticText {
                disabled,
                "(256,512,1024)"
            }
        }
};

resource 'DITL' (151, "(ReadD)") {
        {       /* array DITLarray: 2 elements */
            /* [1] */
            {105, 200, 126, 261},
            Button {
                enabled,
                "OK"
            },
            /* [2] */
            {28, 44, 65, 281},
            StaticText {
                disabled,
                "Sorry, there is something wrong in your "
                "data images. Please check."
            }
        }
};

resource 'DITL' (152) {
        {       /* array DITLarray: 7 elements */
            /* [1] */
            {239, 389, 273, 473},
            Button {
                enabled,
                "Apply"
            },
            /* [2] */
            {205, 400, 225, 460},
            Button {
                enabled,
                "Cancel"
            },
            /* [3] */
            {32, 52, 68, 465},
```

```
                StaticText {
                        disabled,
                        "This routine divides the current image b"
                        "y the previous image:\n                Ne"
                        "w Image = 1000*(current/previous)."
                },
                /* [4] */
                {101, 67, 155, 454},
                StaticText {
                        disabled,
                        "\nIf any pixel in either image is below t"
                        "hreshold then the corresponding pixel in"
                        " the new image is assigned to zero."
                },
                /* [5] */
                {78, 54, 98, 114},
                StaticText {
                        disabled,
                        "Note:"
                },
                /* [6] */
                {219, 64, 237, 136},
                StaticText {
                        disabled,
                        "Threshold:"
                },
                /* [7] */
                {220, 140, 236, 193},
                EditText {
                        disabled,
                        "1000"
                }
        }
};

resource 'DITL' (153, "Filter") {
        {       /* array DITLarray: 21 elements */
                /* [1] */
                {204, 246, 224, 306},
                Button {
                        enabled,
                        "Apply"
                },
                /* [2] */
                {205, 68, 225, 128},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {80, 13, 99, 98},
                RadioButton {
                        enabled,
                        "Low Pass:"
                },
                /* [4] */
                {37, 13, 55, 101},
                RadioButton {
                        enabled,
                        "High Pass:"
                },
                /* [5] */
                {125, 13, 147, 103},
```

```
RadioButton {
        enabled,
        "Band Pass:"
},
/* [6] */
{38, 105, 54, 207},
StaticText {
        disabled,
        "Set lower than"
},
/* [7] */
{38, 215, 53, 266},
EditText {
        disabled,
        "5000"
},
/* [8] */
{37, 272, 54, 293},
StaticText {
        disabled,
        "to"
},
/* [9] */
{38, 300, 53, 335},
EditText {
        disabled,
        "0"
},
/* [10] */
{6, 122, 23, 219},
StaticText {
        disabled,
        "*** Filters ***"
},
/* [11] */
{81, 105, 97, 210},
StaticText {
        disabled,
        "Set higher than"
},
/* [12] */
{81, 215, 96, 266},
EditText {
        disabled,
        "1000"
},
/* [13] */
{81, 272, 98, 294},
StaticText {
        disabled,
        "to"
},
/* [14] */
{81, 300, 96, 335},
EditText {
        disabled,
        "0"
},
/* [15] */
{119, 130, 136, 210},
StaticText {
        disabled,
        "higher than"
```

```
                },
                /* [16] */
                {121, 215, 136, 266},
                EditText {
                        disabled,
                        "5000"
                },
                /* [17] */
                {141, 133, 159, 208},
                StaticText {
                        disabled,
                        "lower than"
                },
                /* [18] */
                {143, 215, 159, 266},
                EditText {
                        disabled,
                        "1000"
                },
                /* [19] */
                {132, 272, 149, 293},
                StaticText {
                        disabled,
                        "to"
                },
                /* [20] */
                {132, 300, 148, 335},
                EditText {
                        disabled,
                        "0"
                },
                /* [21] */
                {126, 105, 144, 129},
                StaticText {
                        disabled,
                        "Set"
                }
        }
};

resource 'DITL' (154) {
        {       /* array DITLarray: 3 elements */
                /* [1] */
                {143, 185, 169, 261},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {28, 36, 96, 85},
                Icon {
                        disabled,
                        1
                },
                /* [3] */
                {39, 119, 89, 297},
                StaticText {
                        disabled,
                        "This routine requires data images in the"
                        " same folder as ImageShow™."
                }
        }
};
```

```
resource 'DITL' (155, "(Not64128)") {
        {       /* array DITLarray: 3 elements */
                /* [1] */
                {111, 255, 134, 321},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {12, 33, 119, 71},
                Icon {
                        disabled,
                        1
                },
                /* [3] */
                {22, 101, 75, 354},
                StaticText {
                        disabled,
                        "Sorry....\nonly 64x64 and 128x128 images"
                        " are supported by this routine."
                }
        }
};

resource 'DITL' (156, "Shifting") {
        {       /* array DITLarray: 11 elements */
                /* [1] */
                {117, 253, 173, 301},
                Button {
                        enabled,
                        "OK"
                },
                /* [2] */
                {72, 249, 92, 302},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {16, 86, 34, 278},
                StaticText {
                        disabled,
                        "** Shifting Image Around **"
                },
                /* [4] */
                {59, 23, 79, 100},
                RadioButton {
                        enabled,
                        "Shift Up:"
                },
                /* [5] */
                {89, 23, 109, 119},
                RadioButton {
                        enabled,
                        "Shift down:"
                },
                /* [6] */
                {119, 23, 139, 125},
                RadioButton {
                        enabled,
                        "Shift to left:"
                },
```

```
                /* [7] */
                {149, 23, 169, 133},
                RadioButton {
                        enabled,
                        "Shift to right:"
                },
                /* [8] */
                {62, 143, 77, 170},
                EditText {
                        disabled,
                        "0"
                },
                /* [9] */
                {92, 143, 107, 170},
                EditText {
                        disabled,
                        "0"
                },
                /* [10] */
                {122, 143, 137, 170},
                EditText {
                        disabled,
                        "0"
                },
                /* [11] */
                {152, 143, 167, 170},
                EditText {
                        disabled,
                        "0"
                }
        }
};

resource 'DITL' (157, "Mathmatical") {
        {       /* array DITLarray: 9 elements */
                /* [1] */
                {156, 245, 197, 317},
                Button {
                        enabled,
                        "Apply"
                },
                /* [2] */
                {169, 82, 189, 142},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {3, 67, 20, 326},
                StaticText {
                        disabled,
                        "*** Image Mathmatical Operations ***"
                },
                /* [4] */
                {40, 20, 60, 128},
                RadioButton {
                        enabled,
                        "Arc Sin(pixel)"
                },
                /* [5] */
                {65, 20, 85, 131},
                RadioButton {
                        enabled,
```

```
                    "Arc Cos(pixel)"
                },
                /* [6] */
                {90, 20, 110, 130},
                RadioButton {
                    enabled,
                    "Arc Tan(pixel)"
                },
                /* [7] */
                {95, 277, 110, 325},
                EditText {
                    disabled,
                    "1000"
                },
                /* [8] */
                {40, 163, 88, 374},
                StaticText {
                    disabled,
                    "Notes: \n 1: Pixel = 1000*(in radians) fo"
                    "r \n      ArcSin and ArcCos."
                },
                /* [9] */
                {94, 167, 112, 271},
                StaticText {
                    disabled,
                    "2: Final image *"
                }
        }
};

resource 'DITL' (158, "SingleQ") {
        {       /* array DITLarray: 16 elements */
                /* [1] */
                {208, 392, 249, 464},
                Button {
                    enabled,
                    "Ready"
                },
                /* [2] */
                {151, 397, 175, 456},
                Button {
                    enabled,
                    "Cancel"
                },
                /* [3] */
                {16, 118, 36, 404},
                StaticText {
                    disabled,
                    "*** Single Q Velocity Image Processing  "
                    "***"
                },
                /* [4] */
                {228, 41, 246, 168},
                StaticText {
                    disabled,
                    "Analyze Threshold:"
                },
                /* [5] */
                {231, 188, 247, 224},
                EditText {
                    disabled,
                    "1000"
                },
```

```
/* [6] */
{203, 42, 219, 152},
StaticText {
        disabled,
        "PGSE Duration ∂:"
},
/* [7] */
{205, 188, 221, 242},
EditText {
        disabled,
        "200"
},
/* [8] */
{179, 42, 195, 168},
StaticText {
        disabled,
        "PGSE Separation Δ:"
},
/* [9] */
{180, 188, 195, 243},
EditText {
        disabled,
        "80000"
},
/* [10] */
{154, 42, 172, 172},
StaticText {
        disabled,
        "Max PGSE Gradient:"
},
/* [11] */
{155, 188, 170, 227},
EditText {
        disabled,
        "99"
},
/* [12] */
{155, 231, 172, 333},
StaticText {
        disabled,
        "(0.1 Gauss/cm)"
},
/* [13] */
{179, 247, 196, 280},
StaticText {
        disabled,
        "(μs)"
},
/* [14] */
{205, 248, 221, 281},
StaticText {
        disabled,
        "(μs)"
},
/* [15] */
{45, 28, 83, 485},
StaticText {
        disabled,
        "This routine requires three images: a ca"
        "libration image at zero flow; a calibrat"
        "ion image under flow; and a velocity cod"
        "ing image."
},
```

```
                /* [16] */
                {87, 28, 107, 452},
                StaticText {
                        disabled,
                        "They are required to be centered to the "
                        "first calibration image."
                }
        }
};

resource 'DITL' (159, "Ramping") {
        {       /* array DITLarray: 15 elements */
                /* [1] */
                {230, 320, 257, 401},
                Button {
                        enabled,
                        "Apply"
                },
                /* [2] */
                {234, 235, 254, 295},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {17, 44, 36, 383},
                StaticText {
                        disabled,
                        "*** This routine adjusts the 'slope' of "
                        "an image ***"
                },
                /* [4] */
                {135, 77, 155, 168},
                RadioButton {
                        enabled,
                        "Horizontal"
                },
                /* [5] */
                {160, 77, 180, 151},
                RadioButton {
                        enabled,
                        "Vertical"
                },
                /* [6] */
                {111, 62, 130, 188},
                StaticText {
                        disabled,
                        "Select a direction,"
                },
                /* [7] */
                {138, 243, 152, 276},
                EditText {
                        disabled,
                        "10"
                },
                /* [8] */
                {163, 243, 178, 276},
                EditText {
                        disabled,
                        "20"
                },
                /* [9] */
                {137, 216, 154, 238},
```

```
                StaticText {
                        disabled,
                        "X="
                },
                /* [10] */
                {162, 216, 179, 239},
                StaticText {
                        disabled,
                        "Y="
                },
                /* [11] */
                {111, 190, 128, 336},
                StaticText {
                        disabled,
                        "input a balance point"
                },
                /* [12] */
                {190, 68, 208, 208},
                StaticText {
                        disabled,
                        "and apply a ramp of:"
                },
                /* [13] */
                {191, 212, 206, 272},
                EditText {
                        disabled,
                        "100"
                },
                /* [14] */
                {190, 277, 207, 348},
                StaticText {
                        disabled,
                        "(per pixel)"
                },
                /* [15] */
                {49, 28, 82, 405},
                StaticText {
                        disabled,
                        "For pixels > the balance point, Amp=Amp+"
                        "ramp*distance;\nfor pixels < the balance "
                        "point, Amp=Amp-ramp*distance."
                }
        }
};

resource 'DITL' (160) {
        {       /* array DITLarray: 14 elements */
                /* [1] */
                {138, 253, 184, 320},
                Button {
                        enabled,
                        "Apply"
                },
                /* [2] */
                {154, 189, 174, 239},
                Button {
                        enabled,
                        "Cancel"
                },
                /* [3] */
                {70, 23, 86, 110},
                StaticText {
                        disabled,
```

```
        "Hori. Center:"
},
/* [4] */
{70, 114, 86, 149},
EditText {
        disabled,
        "128"
},
/* [5] */
{70, 165, 86, 270},
StaticText {
        disabled,
        "Hori. Exponent:"
},
/* [6] */
{70, 275, 86, 308},
EditText {
        disabled,
        "82"
},
/* [7] */
{6, 83, 22, 256},
StaticText {
        disabled,
        "* 2-D Fourier Transform *"
},
/* [8] */
{37, 12, 53, 332},
StaticText {
        disabled,
        "Enter the double exponential apodization"
        " filter:"
},
/* [9] */
{139, 19, 156, 119},
RadioButton {
        enabled,
        "Forward FFT"
},
/* [10] */
{164, 19, 180, 116},
RadioButton {
        enabled,
        "Inverse FFT"
},
/* [11] */
{100, 23, 116, 110},
StaticText {
        disabled,
        "Vert. Center:"
},
/* [12] */
{100, 114, 116, 149},
EditText {
        disabled,
        "128"
},
/* [13] */
{100, 165, 116, 270},
StaticText {
        disabled,
        "Vert. Exponent:"
},
```

```
                /* [14] */
                {100, 275, 116, 308},
                EditText {
                        disabled,
                        "82"
                }
        }
};

data 'MASY' (0) {
        $"1A 49 6D 61 67 65 53 68 6F 77 2D 34 2E 30 20 4D"    /* .ImageShow-4.0 M */
        $"61 73 73 65 79 20 55 6E 69 76 2E 20 2D 20 4E 5A"    /* assey Univ. - NZ */
        $"28 34 2F 31 39 39 31 29 20 20"                      /* (4/1991)   */
};

resource 'BNDL' (128) {
        'MASY',
        0,
        {       /* array TypeArray: 2 elements */
                /* [1] */
                'ICN#',
                {       /* array IDArray: 2 element */
                        /* [1] */
                        0, 128,
                        /* [2] */
                        1, 129
                },
                /* [2] */
                'FREF',
                {       /* array IDArray: 2 element */
                        /* [1] */
                        0, 128,
                        /* [2] */
                        1, 129
                }
        }
};

resource 'ICN#' (128) {
        {       /* array: 2 elements */
                /* [1] */
                $"0000 0000 0000 0000 0000 0000 0000 0000"
                $"0007 FFFC 0008 0002 0008 0002 0009 FFF2"
                $"0009 0012 0009 1C12 0009 1F12 0009 3F12"
                $"0009 3F12 0009 0C12 0009 0012 0009 FFF2"
                $"0008 0002 0008 0002 0007 FFFC 0002 0008"
                $"0003 FFF8 0002 0008 0007 FFFC 0008 0002"
                $"7FFF FFFF 4000 0001 5800 3E7D 4000 0001"
                $"4000 0001 7FFF FFFF 2AAA AAAA 3FFF FFFE",
                /* [2] */
                $"0000 0000 0000 0000 0000 0000 0000 0000"
                $"0007 FFFC 000F FFFE 000F FFFE 000F FFFE"
                $"000F FFFE 000F FFFE 000F FFFE 000F FFFE"
                $"000F FFFE 000F FFFE 000F FFFE 000F FFFE"
                $"000F FFFE 000F FFFE 0007 FFFC 0003 FFF8"
                $"0003 FFF8 0003 FFF8 0007 FFFC 000F FFFE"
                $"7FFF FFFF 7FFF FFFF 7FFF FFFF 7FFF FFFF"
                $"7FFF FFFF 7FFF FFFF 3FFF FFFE 3FFF FFFE"
        }
};

resource 'ICN#' (129) {
        {       /* array: 2 elements */
```

```
                /* [1] */
                $"0000 0000 744C 6700 26D2 8400 255E B600"
                $"2452 9400 7452 6700 0000 0000 0000 0000"
                $"0000 0000 FFFF FE00 8000 0200 8000 0200"
                $"8000 0200 8020 0200 8058 0200 8186 0200"
                $"8102 0200 8201 0200 8C00 8200 8800 8200"
                $"8400 8200 8401 0200 8201 0200 81E2 0200"
                $"801C 0200 8000 0200 8000 0200 FFFF FE",
                /* [2] */
                $"FFFF FF80 FFFF FF80 FFFF FF80 FFFF FF80"
                $"FFFF FF80 FFFF FF80 FFFF FF80 0000 0000"
                $"0000 0000 FFFF FE00 FFFF FE00 FFFF FE00"
                $"FFFF FE00 FFFF FE00 FFFF FE00 FFFF FE00"
                $"FFFF FE00 FFFF FE00 FFFF FE00 FFFF FE00"
                $"FFFF FE00 FFFF FE00 FFFF FE00 FFFF FE00"
                $"FFFF FE00 FFFF FE00 FFFF FE00 FFFF FE"
        }
};

resource 'FREF' (128) {
        'APPL',
        0,
        ""
};

resource 'FREF' (129) {
        'IMGE',
        1,
        ""
};

resource 'ICON' (129) {
        $"0000 0000 0000 0000 0000 0000 0000 0000"
        $"0000 0000 0000 0000 0000 0000 0000 0000"
        $"0000 0000 0000 0000 0000 0000 0000 0000"
        $"0000 0000 0000 0000 0000 0000 0000 0000"
        $"0000 0000 0000 0000 0000 0000 0000 0000"
        $"0007 F000 0008 0800 0010 8400 0021 C200"
        $"0023 E200 0027 F200 002F FA00 0021 C200"
        $"0021 C200 0021 C200 0020 0200 0020 02"
};

resource 'ICON' (130) {
        $"0020 0200 0020 0200 0021 C200 0021 C200"
        $"0021 C200 002F FA00 0027 F200 0023 E200"
        $"0021 C200 0010 8400 0008 0800 0007 F0"
};

resource 'ICON' (131) {
        $"00FF FFFF 01FF FFFF 0380 0000 0700 0000"
        $"0E00 C000 1C01 C000 3803 C000 7007 C000"
        $"E00F C000 C01F C000 C03F C000 C07F C000"
        $"C0FF C000 C1FF FFFE C3FF FFFE C7FF FFFE"
        $"C7FF FFFE C3FF FFFE C1FF FFFE C0FF C000"
        $"C07F C000 C03F C000 C01F C000 E00F C000"
        $"7007 C000 3803 C000 1C01 C000 0E00 C000"
        $"0700 0000 0380 0000 01FF FFFF 00FF FFFF"
};

resource 'ICON' (132) {
        $"FFFF FF00 FFFF FF80 0000 01C0 0000 00E0"
        $"0003 0070 0003 8038 0003 C01C 0003 E00E"
        $"0003 F007 0003 F803 0003 FC03 0003 FE03"
```

```
        $"0003 FF03 7FFF FF83 7FFF FFC3 7FFF FFE3"
        $"7FFF FFE3 7FFF FFC3 7FFF FF83 0003 FF03"
        $"0003 FE03 0003 FC03 0003 F803 0003 F007"
        $"0003 E00E 0003 C01C 0003 8038 0003 0070"
        $"0000 00E0 0000 01C0 FFFF FF80 FFFF FF"
};

resource 'ICON' (133) {
        $"00FF FF00 01FF FF80 0380 01C0 0700 00E0"
        $"0E01 8070 1C03 C038 3807 E01C 700F F00E"
        $"E01F F807 C03F FC03 C07F FE03 C0FF FF03"
        $"C1FF FF83 C3FF FFC3 C7FF FFE3 CFFF FFF3"
        $"CFFF FFF3 C007 E003 C007 E003 C007 E003"
        $"C007 E003 C007 E003 C007 E003 C007 E003"
        $"C007 E003 C007 E003 C007 E003 C007 E003"
        $"C007 E003 C007 E003 C007 E003 C000 0003"
};

resource 'ICON' (134) {
        $"C000 0003 C007 E003 C007 E003 C007 E003"
        $"C007 E003 C007 E003 C007 E003 C007 E003"
        $"C007 E003 C007 E003 C007 E003 C007 E003"
        $"C007 E003 C007 E003 C007 E003 CFFF FFF3"
        $"CFFF FFF3 C7FF FFE3 C3FF FFC3 C1FF FF83"
        $"C0FF FF03 C07F FE03 C03F FC03 E01F F807"
        $"700F F00E 3807 E01C 1C03 C038 0E01 8070"
        $"0700 00E0 0380 01C0 01FF FF80 00FF FF"
};

resource 'CURS' (128, preload) {
        $"07 C0 08 20 0A A0 19 30 08 20 09 20 04 44 03 88"
        $"01 10 3F E0 01 00 02 80 04 48 38 30 7F FC",
        $"3F F8 7F FC FF FE FF FE FF FE FF FE FF FE FF FE"
        $"FF FE FF FE FF FE FF FE FF FE FF FE 7F FC 3F F8",
        {7, 7}
};

resource 'CURS' (129, preload) {
        $"07 C0 08 20 0A A0 19 30 08 20 09 20 44 44 23 88"
        $"11 10 0F E0 01 00 02 80 24 48 18 30 7F FC",
        $"3F F8 3F FC FF FE FF FE FF FE FF FE FF FE FF FE"
        $"FF FE FF FE FF FE FF FE FF FE FF FE 7F FC 3F F8",
        {7, 7}
};

resource 'CURS' (130, preload) {
        $"07 C0 08 20 0A A0 19 30 08 20 09 20 44 40 23 80"
        $"11 00 0F E0 01 10 02 88 04 40 38 38 7F FC",
        $"3F F8 7F FC FF FE FF FE FF FE FF FE FF FE FF FE"
        $"FF FE FF FE FF FE FF FE FF FE FF FE 7F FC 3F F8",
        {7, 7}
};

resource 'CURS' (131, preload) {
        $"07 C0 08 20 0A A0 19 30 08 20 09 20 04 40 03 80"
        $"01 00 0F E0 11 10 22 88 04 40 38 38 7F FC",
        $"3F F8 7F FC FF FE FF FE FF FE FF FE FF FE FF FE"
        $"FF FE FF FE FF FE FF FE FF FE FF FE 7F FE 3F FC",
        {7, 7}
};
```

A3.3 ImageShow.make

```
# File:       ImageShow.make
# Target:     ImageShow
# Sources:    ImageShow.p ImageShow.r
# Created:    Saturday, 28 October 1989 3:22:24 PM

ImageShow.p.o ƒ ImageShow.make ImageShow.p
        Pascal -m -MC68020 -MC68881 ImageShow.p
ImageShow ƒƒ ImageShow.make ImageShow.r
        Rez ImageShow.r -append -o ImageShow

SOURCES = ImageShow.p ImageShow.r
OBJECTS = ImageShow.p.o

ImageShow ƒƒ ImageShow.make {OBJECTS}
        Link -w -t APPL -c MASY ∂
                {OBJECTS} ∂
                "{Libraries}"Runtime.o ∂
                "{Libraries}"Interface.o ∂
                "{PLibraries}"PasLib.o ∂
                "{PLibraries}"SANELib881.o ∂
                -o ImageShow
```