MIDTERM EXAM
- Take Home Exam (1 Week)
- 6 Sections
  - UNIX Commands
  - Those related to imaging
  - RCS
  - Number Representations
  - PBMPLUS Image Formats
  - Shell Programming
  - IDL Statistic and Simple Widget Question
  - Mystery Image

Pull Down Menu Example

Pulldown Menus in IDL
- Widget_Droplist
- CW_PDMENU

CW_PDMENU
- Offers a more flexible alternative to WIDGET_DROPLIST
- Allows submenus to be defined

CW_PDMENU
- Statistics
  - Minimum
  - Maximum
  - Moments
    - Mean
    - Variance
    - Skewness
    - Kurtosis

CW_PDMENU
- Processing
  - Global
    - Threshold
    - Equalization
  - Local
    - Smoothing
    - Sharpening
Creation and Arrangement of Submenus in CW_PDMENU

- Based on tag numbers specified below
  1 - Root of new pulldown submenu (next button starts submenu)
  2 - Last button of current pulldown level
  3 - Root of new pulldown submenu, but also the last button of current pulldown level
  0 - Regular menu entry (none of the above)

Menu Definition

```plaintext
menu=['1\Statistics', '0\Minimum', '0\Maximum', '3\Moments', '0\Mean', '0\Variance', '0\Skewness', '2\Kurtosis', '3\Processing', '1\Global', '0\Threshold', '2\Equalize', '1\Local', '0\Smooth', '2\Sharpen']
```

CW_PDMENU Widget Definition

```plaintext
pro pd_menu_example
  base=Widget_Base(Title='Pull Down Example')
  ; menu variable defined in previous slide
  pd_menu=CW_PDMENU( base, menu, /return_index)
  Widget_Control, base, /realize
  Xmanager,'pd_menu_example',base
end
```

CW_PDMENU Event Handler

```plaintext
pro pd_menu_example_event, event
  help,/structure,event
end
```

Menu Definition

```plaintext
menu = [ '1\Statistics', '0\Minimum', '0\Maximum', '3\Moments', '0\Mean', '0\Variance', '0\Skewness', '2\Kurtosis', '3\Processing', '1\Global', '0\Threshold', '2\Equalize', '1\Local', '0\Smooth', '2\Sharpen' ]
```

Sample outputs of pd_menu_example

- The following outputs are variations on the different ways in which CW_PDMENU returns widget information
- In this case, we have chosen the following
  - Statistics->Minimum
  - Processing->Global->Threshold
Sample output of
pd_menu_example
using /return_id

Structure <1a355fc>, 4 tags, length=16, refs=1:
ID       LONG             273
TOP      LONG             272
HANDLER  LONG             272
VALUE    LONG             275

Structure <1a355fc>, 4 tags, length=16, refs=1:
ID       LONG             273
TOP      LONG             272
HANDLER  LONG             272
VALUE    LONG             284

Sample output of
pd_menu_example
using /return_index

Structure <1a3561c>, 4 tags, length=16, refs=1:
ID       LONG             307
TOP      LONG             306
HANDLER  LONG             306
VALUE    INT              1

Structure <1a3561c>, 4 tags, length=16, refs=1:
ID       LONG             307
TOP      LONG             306
HANDLER  LONG             306
VALUE    INT              10

Sample output of
pd_menu_example
using /return_name

Structure <1a35cd4>, 4 tags, length=20, refs=1:
ID       LONG             324
TOP      LONG             323
HANDLER  LONG             323
VALUE    STRING  'Mininum'

Structure <1a35cd4>, 4 tags, length=20, refs=1:
ID       LONG             324
TOP      LONG             323
HANDLER  LONG             323
VALUE    STRING  'Threshold'

Sample output of
pd_menu_example
using /return_full_name

Structure <1a35d7c>, 4 tags, length=20, refs=1:
ID       LONG             341
TOP      LONG             340
HANDLER  LONG             340
VALUE    STRING  'Statistics.Mininum'

Structure <1a35d7c>, 4 tags, length=20, refs=1:
ID       LONG             341
TOP      LONG             340
HANDLER  LONG             340
VALUE    STRING  'Processing.Global.Threshold'

CW_DEFROI Example

Defining a region of interest (ROI)

• An example of a pop-up compound widget is the CW_DEFROI
• It is a widget designed to provide a means to define a region of interest for selective processing.
CW_DEFROI Example

- Problem:
  - Display an image
  - Define a region of interest
  - Make the negative of that region

- CW_DEFROI usage specifics
  - Works on a draw_widget
  - draw_widget must be button and motion event enabled
  - Returns indices of pixels defining region of interest

CW_DEFROI Widget View

CW_DEFROI Widget Definition

```pro
pro cw_defroi_example
  base = Widget_Base(/column)
  draw = Widget_Draw( base, xsize=256, ysize=256, /button_events,/motion_events )
  roi_button = Widget_Button( base, value='Define ROI', event_pro='roi_event')
  Widget_Control, base, /realize
  Widget_Control, draw, get_value=window
  image=bindgen(256,256)
  global_data={ draw_id: draw, window_id:window, image_data:image, roi_data: 0 }
  Widget_Control, base, set_uvalue=global_data
  Xmanager,'cw_defroi_example',base
end
```

CW_DEFROI Event Handler

```pro
pro cw_defroi_example_event, event
end
pro roi_event, event
  Widget_Control,event.top,get_uvalue=global_data
  window=global_data.window_id & wset, window
  image=global_data.image_data & tvscl,image
  draw_id=global_data.draw_id
  roi=CW_DEFROI( draw_id )
  image(roi) = 1-image(roi) & tvscl, image
  new_data={draw_id:global_data.draw_id, window_id:global_data.window_id, image_data:image, roi_data:roi}
  Widget_Control,event.top,set_uvalue=new_data
end
```

Mouse Events

- To detect mouse input from the user the following keywords need to be set
  - /MOTION_EVENTS
    - Generates an event whenever the mouse is moved inside a Draw Widget
  - /BUTTON_EVENTS
    - Generates an event whenever the mouse buttons are pushed/released in a Draw Widget
**Widget_Draw Event Structure**

(WIDGET_DRAW, ID:0L, TOP:0L, HANDLER:0L, TYPE:0, X:0, Y:0, PRESS:0B, RELEASE:0B, CLICKS:0)

---

**TYPE**

Event Structure Record

0  Button Press  
1  Button Release  
2  Motion  
3  Viewport Moved(Scrollbars)  
4  Visibility Changed(Exposed)

---

**X and Y**

Event Structure Records

- Gives the location where the event occurred
- Note that the location are in DEVICE COORDINATE with respect to the Draw Widget
- Measured from the LOWER LEFT CORNER of Draw Widget

---

**PRESS and RELEASE**

Event Structure Record

- Gives a BIT MASK value of which buttons on the mouse were pressed/released
- Left-most mouse button corresponds to the least significant bit
- Middle mouse button corresponds to the next significant bit
- Right-most mouse button corresponds to the next significant bit

---

**PRESS and RELEASE**

- Both values are zero (0) when motion events are generated

---

**TRACK MOUSE EXAMPLE**

- We want to create a Draw Widget which prints out the location of the mouse to the screen
track_mouse.pro

pro report_location, event
    print('[event.x, event.y]')
end

pro track_mouse
    base = Widget_Base( Title='Track Mouse' )
    window = Widget_Draw( base, xsize=512, ysize=512, /motion_events, event_pro='report_location' )
    Widget_Control, base, /realize
    Xmanager,'track_mouse',base
end

DRAW MOUSE EXAMPLE

• The following example will draw line segments as the mouse is moved across the Draw Widget

DRAW CLICK MOUSE EXAMPLE

• Similar to the DRAW MOUSE example
• Line segment will be drawn only when a button is pressed
• Color of line segment has been modified to RED

draw_mouse.pro

pro plot_location, event
    plots, event.x, event.y, /device, color=128, /continue
end

pro draw_mouse
    base = Widget_Base( Title='Draw Mouse' )
    window = Widget_Draw( base, xsize=512, ysize=512, /motion_events, /button_events, event_pro='plot_location' )
    Widget_Control, base, /realize
    Xmanager,'draw_mouse',base
end

draw_click_mouse.pro

EVENT HANDLER

pro plot_location, event
    case event.type of
        0: begin
            plots, event.x, event.y, /device, color=128, /continue
        end
        1: ; Button Release
        2: ; Motion
        3: ; Viewport Moved
        4: ; Visibility Changes
        endcase
end

WIDGET DEFINITION

pro draw_click_mouse
    base = Widget_Base( Title='Draw Mouse' )
    window = Widget_Draw( base, xsize=512, ysize=512, /motion_events, /button_events, event_pro='plot_location' )
    Widget_Control, base, /realize
    tvlct, 255,0,0,128
    Xmanager,'draw_click_mouse',base
end
Modal Widgets

Following slides are obsolete.
Please refer to Chapter 13
(Widget Dialog Programs)
for updated information

Modal Widgets

- Modal Widgets also known as pop-up widgets are a means to simplify a GUI interface by allowing an input window to be presented to the user at an appropriate time
- Because a new window is created and destroyed, there needs to be a mechanism to pass the values specified to the used back to the calling widget.

Variables inside a Widget

- Remember that variables defined in a widget only exist as long as that widget exists
- We have to somehow pass out any values defined to a calling function

pop_fslider Case Study

- Problem: We want to create a simple slider that a user can call from the IDL prompt and return to the calling routine a floating point value reflecting the position of the floating point slider (CW_FSLIDER)

IDL> a = pop_fslider()

pop_fslider.pro (initial version - Widget Definition)

function pop_fslider
  base = Widget_Base( /column )
  fslider = CW_Fslider( base, /edit )
  quit_button = Widget_Button( base, value="Quit", event_pro='quit_event' )
  Widget_Control, base, /realize
  global_data={ fslider_id: fslider, fslider_value: 0.0 }
  Widget_Control, base, set_uvalue=global_data
  Xmanager, 'pop_fslider', base, event_handler='pop_fslider_event'
  Widget_Control, base, get_uvalue=global_data
  print,'In Widget Definition=', global_data.fslider_value
  return, global_data.fslider_value
end

pop_fslider.pro (initial version - Event Handler Definition)

pro pop_fslider_event, event
  Widget_Control, event.top, get_uvalue=global_data
  Widget_Control, global_data.fslider_id, get_value=fslider_value
  global_data.fslider_value=fslider_value
  print,'In Event Handler =', global_data.fslider_value
  Widget_Control, event.top, set_uvalue=global_data
end
pro quit_event, event
  Widget_Control, event.top, /destroy
end
pop_fslider.pro
(initial version output)

IDL> a=pop_fslider()
In Event Handler = 44.0000
In Event Handler = 39.0000
In Event Handler = 39.0000
In Event Handler = 42.0000
% WIDGET_CONTROL: Invalid widget identifier: 1.
% Execution halted at: POP_FSLIDER 28
pop_fslider.pro
% SMAINS

pop_fslider.pro
(function pop_fslider)

function pop_fslider
    base = Widget_Base( /column )
    fslider = CW_Fslider( base, /edit )
    quit_button = Widget_Button( base, value="Quit", event_pro='quit_event' )
    Widget_Control, base, /realize
    global_data={ fslider_id: fslider, fslider_value: 0.0 } 
    Widget_Control, base, set_uvalue=global_data
    Xmanager, 'pop_fslider', base, event_handler='pop_fslider_event'
    Widget_Control, base, get_uvalue=global_data
    print,'In Widget Definition=', global_data.fslider_value
    return, global_data.fslider_value
end

pop_fslider.pro
(initial version - Event Definition)

• This does not work because the widget has already been destroyed and along with it the uvalue that was being stored
• To make this work, we need to use what IDL calls handles (or in C/C++ the concept of pointers)

What are Handles?
• Handles (pointers) are special variables that point to memory locations that are explicitly allocated by the user.
• It is different from other variables in that the memory location pointed to by a handle are persistent (i.e., they exist from one routine to another)
• It is also the user’s responsibility to destroy the allocated memory

Uses of Handles
• Management of persistent data
• Minimizes passing around large data structures
• Linked lists data structures
• Tree data structures

IDL Handles (Pointers)

What are Handles?

Uses of Handles
Routines for managing Handles

- Handle_Create (Create a pointer)
  IDL> image_pointer = Handle_Create()
- Handle_Value (Point a handle to data)
  IDL> image = bindgen(5,5)
  IDL> handle_value, pointer, image, /set
- Handle_Value (Copy data pointed to by the handle to a new variable)
  IDL> handle_value, pointer, a

Sample use of Handle Routines

IDL> image_pointer=image_Create()
IDL> image=bindgen(5,5)
IDL> handle_value, image_pointer, image, /set
IDL> help, image
IMAGE    BYTE      = Array[5, 5]
IDL> help, image_pointer
IMAGE_POINTER  LONG      =            1
IDL> help, image_pointer,a
IDL> help,a
A       BYTE      = Array[5, 5]

pop_fslider.pro
(new version - Event Handler Definition)

pro pop_fslider_event, event
  Widget_Control, event.top, get_uvalue=global_data
  Widget_Control, global_data.fslider_id, get_value=fslider_value
  pointer = global_data.handle
  print, 'In Event Handler =', fslider_value
  handle_value, pointer, fslider_value, /set
end
pro quit_event, event
  Widget_Control, event.top, /destroy
end

pop_fslider.pro
(new version - Widget Definition)

function pop_fslider
  ...
  Widget_Control, base, /realize
  pointer = Handle_Create()
  global_data={ fslider_id: fslider, handle: pointer }
  Widget_Control, base, set_uvalue=global_data
  Xmanager, 'pop_fslider', base,
  event_handler='pop_fslider_event'
  Widget_Control, base, get_uvalue=global_data
  print, 'In Event Handler =', global_data.fslider_value
handle_value, pointer, fslider_value
return, fslider_value
end

Result of pop_fslider.pro

IDL> a=pop_fslider()
In Event Handler =       40.0000
In Event Handler =       43.0000
In Event Handler =       42.0000
IDL> print,a
42.0000