Subject: IDL Christmas present for the newsgroup
Posted by ronn on Fri, 22 Dec 2000 14:52:50 GMT
View Forum Message <> Reply to Message

All,

I wanted to give everyone on the newsgroup a little Christmas present, so I wrote a small routine that simulates a snowflake growing using object graphics. (For the curious I am using a variation of the parity rule for cellular automata to make this.) Just cut out the code below and save it to a file called snowflakeObjectDemo.pro. I tried to keep the lines short so that they won't wrap, but it may still happen on your browser so be careful. Compile it and type snowflakeObjectDemo at the IDL prompt and watch it grow. It will grow until a certain point and then stop, but the snowflake will keep rotating. If you want to change the growth rate then set the interval keyword to the interval you want in seconds (snowflakeObjectDemo, interval=5 for 5 seconds).

Thanks again for the help over the year and buying my book. On the first of the year I will have a new book on calling C from IDL that may be of interest to many of you. Look for an announcement here when it is available.

Merry Christmas!

-Ronn

--
Ronn Kling
Ronn Kling Consulting
email: ronn@rlkling.com
"Application Development with IDL"ï¿½ programming book updated for IDL5.4!
Shareware and Freeware at: http://www.rlkling.com/

---------- cut here ------------------------------

```
pro snowflake_event,event
;event that triggers the growth of the snowflake
widget_control,event.top,get_uvalue=object
if obj_valid(object) then object->grow
return
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

pro snowflake::grow
;method to grow snowflake
;uses a variation of the parity rule for cellular automata
```

```
fourneighbor = [[0,1,0], $ ;neighborhood for convolution
                [1,1,1], $
                [0,1,0]]

ind = where(*self.image eq 1) ;find the "frozen" ones
*self.image = convol(*self.image,fourneighbor)
*self.image = *self.image mod 2B ;odd freezes even doesn't
(*self.image)[ind] = 1B ;add the original frozen ones in
self.oImage ->setProperty,data=bytscl(*self.image)
widget_control,self.base,timer=self.interval ;set timer

return
end
;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

pro snowflake::cleanup
;clean everything up
self->IDLgrModel::cleanup
if obj_valid(self.oImage) then obj_destroy,self.oImage
if ptr_valid(self.image) then ptr_free,self.image

return
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

function snowflake::init, interval=interval

if not keyword_set(interval) then interval=1
self.interval = interval

;initialize the super class. THIS HAS TO BE DONE!
IF (self->IDLgrModel::Init(_EXTRA=extra) NE 1) THEN RETURN, 0

image = bytarr(256,256)
image[128,128] = 1B
self.image = ptr_new(image,/no_copy)
;have to bytscl image since it is only 0 and 1's
self.oImage = obj_new('IDLgrImage',bytscl(*self.image) )
;put it in the center
x = [0,1,1,0]-.5
y = [0,0,1,1] -.5
z = [0,0,0,0]
oPoly =
 obj_new('IDLgrPolygon',x,y,z,color=[255,255,255],texture_map =self.oImage, $
           texture_coord = [[0,0], [1,0], [1,1], [0,1]], /texture_interp)
self->add, oPoly
```

```
;create an invisible base that generates timer events
self.base = widget_base(map=0)
widget_control,self.base,/realize
widget_control,self.base,timer=self.interval
widget_control,self.base,set_uvalue=self
xmanager,'snowflake',self.base,/no_block

return,1
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

pro snowflake__define
;definition of the snowflake object
void = { snowflake, $
        inherits IDLgrmodel, $
        oImage : obj_new(), $ ;image object
        image : ptr_new(), $ ;original data
        base : 0L, $ ;widget base
        interval : 0 } ;interval for growth

return
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|
; Widget code starts here
;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

pro snowflakeObjectDemo_event,event

widget_control,event.top,get_uvalue=state

if tag_names(event,/structure_name) eq 'WIDGET_TIMER' then begin
  state.oRotateModel->rotate,[1,1,1],5 ;rotate object
  state.oWindow->Draw, state.oView
  widget_control, event.id, timer=.1 ;set new event
  return
endif

if (event.type EQ 4) then begin ;expose events here
                state.oWindow->Draw, state.oView
endif

return
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|
```

```
pro snowflakeObjectExit,event
;called from pull down
widget_control,event.top,/destroy

return
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

pro snowflakeObjectDemoCleanup, base
;cleans up all the objects
widget_control,base,get_uvalue=state
obj_destroy, state.oContainer
tvlct,state.colorTable ;reload orig color table

return
end

;{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|{{:|

pro snowflakeObjectDemo,  interval=interval

;  Get the current color table.
;  It will be restored when exiting.
;
tvlct, savedR, savedG, savedB, /GET
colorTable=[[savedR],[savedG],[savedB]]

;  Get the screen size.
;
device, get_screen_size = screenSize
xdim = screenSize[0]*.50 ;about 50 percent of the screen size
ydim=xdim  ;keep isotropic

wTopBase=widget_base(TITLE='Snowflake', $
           XPAD=0, YPAD=0, $
          /COLUMN,/tlb_size_events,MBAR=barBase)

fileId = widget_button(barBase, value='File', /menu)
quitter = widget_button(fileId, /separator, value='Exit', $
         event_pro='snowflakeObjectExit')

drawId=widget_draw(wTopBase, xsize=xdim, $
                ysize=ydim, $  ;keep isotropic
                graphics_level=2, retain=0, /expose_events)

widget_control, wTopBase, /realize
widget_control, hourglass=1
```

```
widget_control, drawId, get_value=oWindow
;  Create the objects.
mView = [-1,-1,2,2]
maxDim = max([xdim,ydim])
maxDim = 1.
;  Create view object.
oView = obj_new('IDLgrView', projection=1, eye=maxDim + 0.1, $
       color=[0,0,0], $
       view=myView, zclip=[maxDim,-maxDim])
oRotateModel = obj_new('IDLgrModel') ;need a model to rotate
oSnow = obj_new('snowflake' ,interval=interval ) ;snowflake object
oRotateModel->add, oSnow
oView -> add, oRotateModel
owindow->Draw, oView

oContainer = OBJ_NEW('IDL_Container');store objects for cleanup
oContainer->add, oView
oContainer -> add, oRotateModel
oContainer -> add, oSnow

;  Build state structure used in event handlers.
;
state={ drawId:drawId, $             ; Widget draw ID
       oContainer: oContainer, $  ; Container object
       oWindow:oWindow, $         ; Window object
       oView:oView, $             ; View object
       oRotateModel : oRotateModel, $ ;rotating model
       colorTable:colorTable  } ;orig color table

widget_control, wTopBase, set_uvalue=state, /no_copy
widget_control, hourglass=0
widget_control, drawId, timer=.1 ;start timer
xmanager, 'snowflakeObjectDemo', wTopBase, /no_block, $
         cleanup='snowflakeObjectDemoCleanup'

return
end
```