

API documentation for visualization library

API documentation for visualization library

Table of Contents

Overview	v
1. Directories	v
2. Project statistics	v
I. API	1
1. <i>color/</i>	5
2. <i>directgraphics/</i>	15
3. <i>flow/</i>	19
4. <i>googlechart/</i>	23
5. <i>graphs/</i>	29
6. <i>images/</i>	31
7. <i>lineplots/</i>	35
8. <i>objectgraphics/</i>	39
9. <i>povray/</i>	41
10. <i>text/</i>	47
11. <i>util/</i>	49

Overview

Library of routines to facilitate creating better visualizations. Both direct graphics and object graphics systems are addressed.

1. Directories

Directory	Description
<i>color/</i>	routines for dealing with specifying colors and color tables
<i>directgraphics/</i>	helper routines for direct graphics
<i>flow/</i>	routines for visualization of vector fields
<i>googlechart/</i>	IDL interface to Google Charts API
<i>graphs/</i>	routines for visualization of trees and graphs
<i>images/</i>	routines for display of images
<i>lineplots/</i>	routines for creating various types of line plots
<i>objectgraphics/</i>	helper routines and classes for the object graphics systems
<i>povray/</i>	routines to create POV-Ray renderings of data in IDL
<i>text/</i>	routines for handling text in graphics
<i>util/</i>	utility routines for other routines

2. Project statistics

Attribute	Value
Directories:	11
.pro files:	32
.sav files:	0
Routines:	61
Lines:	3,213

Part I. API

Table of Contents

1. <i>color/</i>	5
1.1. Overview	5
1.2. <i>vis_color.pro</i>	5
1.3. <i>vis_create_ctfile.pro</i>	7
1.4. <i>vis_index2rgb.pro</i>	8
1.5. <i>vis_loadct.pro</i>	8
1.6. <i>vis_rgb2index.pro</i>	10
1.7. <i>vis_xloadct.pro</i>	11
1.8. <i>visgrpalette_define.pro</i>	12
2. <i>directgraphics/</i>	15
2.1. Overview	15
2.2. <i>vis_contour.pro</i>	15
2.3. <i>vis_psbegin.pro</i>	16
2.4. <i>vis_psend.pro</i>	17
2.5. <i>visdgvars_define.pro</i>	17
3. <i>flow/</i>	19
3.1. Overview	19
3.2. <i>vis_build_flow.pro</i>	19
3.3. <i>vis_lic.pro</i>	19
3.4. <i>vis_vel.pro</i>	20
4. <i>googlechart/</i>	23
4.1. Overview	23
4.2. <i>vis_gc_base.pro</i>	23
4.3. <i>vis_gc_piechart.pro</i>	25
4.4. <i>vis_gc_scatter.pro</i>	26
4.5. <i>vis_gc_venn.pro</i>	27
5. <i>graphs/</i>	29
5.1. Overview	29
5.2. <i>vis_graph_layout.pro</i>	29
5.3. <i>vis_tree_layout.pro</i>	29
6. <i>images/</i>	31
6.1. Overview	31
6.2. <i>vis_image.pro</i>	31
6.3. <i>vis_image_getsize.pro</i>	32
6.4. <i>vis_image_resize.pro</i>	32
7. <i>lineplots/</i>	35
7.1. Overview	35
7.2. <i>vis_plot.pro</i>	35
7.3. <i>vis_scatter3d.pro</i>	36
7.4. <i>vis_themeriver.pro</i>	36
8. <i>objectgraphics/</i>	39
8.1. Overview	39
8.2. <i>visgrcube_define.pro</i>	39

9. <i>povray/</i>	41
9.1. Overview	41
9.2. <i>visgrpovray_define.pro</i>	41
10. <i>text/</i>	47
10.1. Overview	47
10.2. <i>vis_strwrap.pro</i>	47
11. <i>util/</i>	49
11.1. Overview	49
11.2. <i>vis_convert.pro</i>	49
11.3. <i>vis_make_dll.pro</i>	50
11.4. <i>vis_src_root.pro</i>	50

Chapter 1. *color/*

1.1. Overview

The *.pro* files in this directory are listed below.

vis_color.pro

Get a RGB color value for the specified color name.

vis_create_ctfile.pro

Create a new color table file suitable for use with MODIFYCT, LOADCT, XLOADCT, and IDLgrPalette::loadCT.

vis_index2rgb.pro

Converts color indices to RGB coordinates.

vis_loadct.pro

Load a Brewer color table by index.

vis_rgb2index.pro

Convert RGB coordinates of colors to the decomposed color indices of the colors.

vis_xloadct.pro

Load a Brewer color table by index using a GUI interface.

visgrpalette__define.pro

Subclass of IDLgrPalette which loads Brewer color tables instead of the default IDL color tables.

1.2. *vis_color.pro*

VIS_COLOR

```
result = vis_color(colorname [, /names] [, /index])
```

Get a RGB color value for the specified color name. The available colors are:

AliceBlue	AntiqueWhite	Aqua	Cyan
Aquamarine	Azure	Beige	Gold
Bisque	Black	BlanchedAlmond	Goldenrod
Blue	BlueViolet	Brown	Gray
Burlywood	CadetBlue	Chartreuse	Green
Chocolate	Coral	CornflowerBlue	GreenYellow
Cornsilk	Crimson	Cyan	IndianRed
DarkBlue	DarkCyan	DarkGoldenrod	Ivory
DarkGray	DarkGrey	DarkGreen	Khaki
DarkKhaki	DarkMagenta	DarkOliveGreen	LavenderBlush
DarkOrange	DarkOrchid	DarkRed	LightBlue
DarkSalmon	DarkSeaGreen	DarkSlateBlue	LightGoldrenrodYellow
DarkSlateGray	DarkSlateGrey	DarkTurquoise	LightGray
DarkViolet	DeepPink	DeepSkyBlue	LightGreen
DimGray	DimGrey	DodgerBlue	LightPink
FireGray	FloralWhite	ForestGreen	LightSalmon
Fuchsia	Gainsboro	GhostWhite	LightSlateGray
Gold	Goldenrod	Gray	Lime
Grey	Green	GreenYellow	LimeGreen
HoneyDew	HotPink	IndianRed	Maroon
Indigo	Ivory	Khaki	MediumOrchid
Lavender	LavenderBlush	LawnGreen	MediumSeaGreen
LemonChiffon	LightBlue	LightCoral	MediumSlateBlue
LightCyan	LightGoldrenrodYellow	LightGray	MediumTurquoise
LightGrey	LightGreen	LightPink	MediumVioletRed
LightSalmon	LightSeaGreen	LightSkyBlue	MintCream
LightSlateGray	LightSlateGrey	LightSteelBlue	MistyRose
LightYellow	Lime	LimeGreen	Navy
Linen	Magenta	Maroon	OliveDrab
MediumAquamarine	MediumBlue	MediumOrchid	Orchid
MediumPurple	MediumSeaGreen	MediumSlateBlue	PaleTurquoise
MediumSpringGreen	MediumTurquoise	MediumVioletRed	PeachPuff
MidnightBlue	MintCream	MistyRose	Plum
Moccasin	NavajoWhite	Navy	Red
OldLace	Olive	OliveDrab	SaddleBrown
Orange	OrangeRed	Orchid	SeaGreen
PaleGoldenrod	PaleGreen	PaleTurquoise	Silver
PaleVioletRed	PapayaWhip	PeachPuff	SlateGray
Peru	Pink	Plum	SpringGreen
PowderBlue	Purple	Red	Teal
RosyBrown	RoyalBlue	SaddleBrown	Tan
Salmon	SandyBrown	SeaGreen	Tomato
Seashell	Sienna	Silver	Wheat
SkyBlue	SlateBlue	SlateGray	White
SlateGrey	Snow	SpringGreen	WhiteSmoke
SteelBlue	Tan	Teal	Yellow
Thistle	Tomato	Turquoise	YellowGreen
Violet	Wheat	White	
WhiteSmoke	Yellow	YellowGreen	

Return value

either a triple as a bytarr(3), a decomposed color index as a long, or strarr for the names

Parameters

colormame — in required type=string
case-insensitive name of the color

Keywords

NAMES — in optional type=boolean
set to return a string of color names

INDEX — in optional type=boolean
set to return a long integer with the RGB decomposed into it

Examples

- For example:

```
IDL> print, vis_color('black')
    0  0  0
IDL> print, vis_color('slateblue')
   106  90 205
IDL> c = vis_color('slateblue', /index)
IDL> print, c, c, format='(I, Z)'
   13458026      CD5A6A
IDL> print, vis_color(/names)
```

These commands are in the main-level example program:

```
IDL> .run vis_color
```

1.3. vis_create_ctfile.pro

VIS_CREATE_CTFILE

vis_create_ctfile, filename

Create a new color table file suitable for use with MODIFYCT, LOADCT, XLOADCT, and IDLgrPalette::loadCT.

Parameters

filename — in required type=string
filename for new color table file

Examples

- To create a new color table file, use VIS_CREATE_CTFILE to create the new file and MODIFYCT to add color tables to it. For example:

```
IDL> vis_create_ctfile, 'test.tbl'
IDL> modifyct, 0, 'CT 0', r0, g0, b0, file='test.tbl'
IDL> modifyct, 1, 'CT 1', r1, g1, b1, file='test.tbl'
...etc...
```

1.4. vis_index2rgb.pro

VIS_INDEX2RGB

```
result = vis_index2rgb(indices)
```

Converts color indices to RGB coordinates. Color indices are long integers used in decomposed color in direct graphics where the lowest order byte value is the red value, the next byte is the green value, the next byte is the blue value, and the highest order byte value is unused.

Return value

bytarr(3) or bytarr(n, 3)

Parameters

indices — in required type=long or lonarr(n)
indices representing either a color or n colors

Examples

- For example:

```
IDL> print, vis_index2rgb('ffff00'x)
0 255 255
```

Multiple colors can be converted at once:

```
IDL> colors = ['ffff00'x, 'ffffff'x, '0000ff'x, 'ff00ff'x]
IDL> rgbColors = vis_index2rgb(colors)
IDL> print, rgbColors
0 255 255 255
255 255 0 0
255 255 0 255
IDL> tvlct, rgbColors
```

1.5. vis_loadct.pro

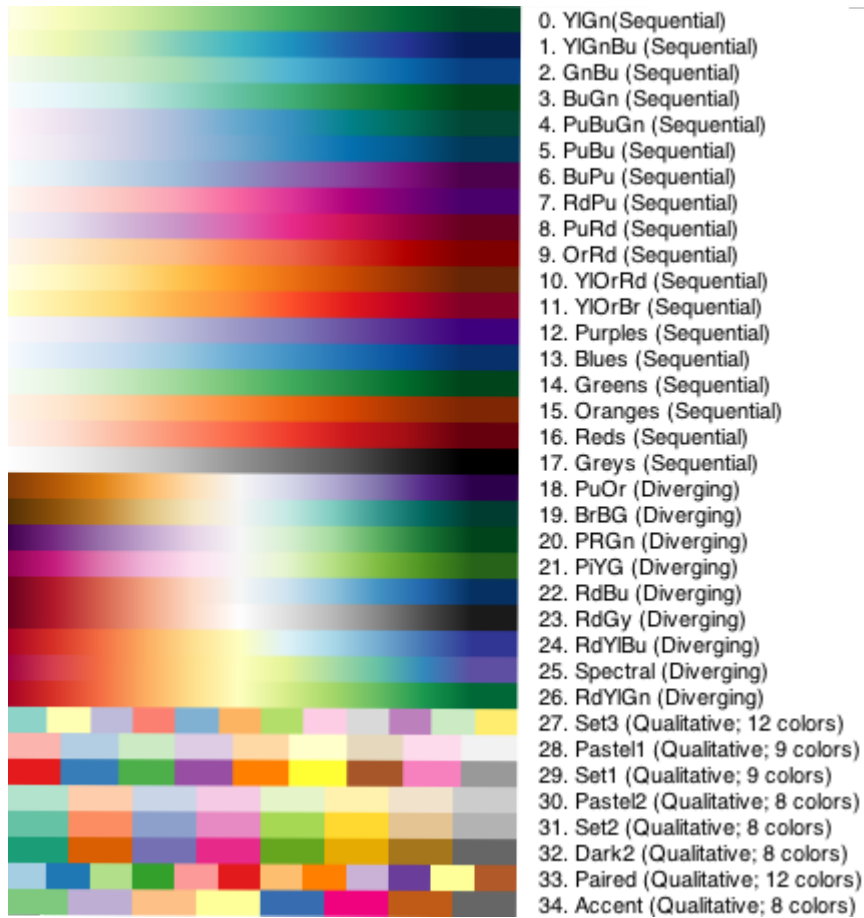
VIS_LOADCT

```
vis_loadct [, table] [, file=string] [, _ref_extra=keyword]
```

Load a Brewer color table by index. This routine is directly analogous to LOADCT, but with the Brewer color tables.

The Brewer color tables are split into three types: sequential, diverging, and qualitative. Sequential color tables are simple sequences from white to a given color. The diverging color tables have white in the middle of the color table and progress in each direction towards two different colors. The qualitative color tables contain only

a few colors for labeling purposes. The qualitative color tables are expanded to take up the same space of the other color tables in the graphic below:



Parameters

table — in optional type=long
 table number, 0-34 if using default color table file

Keywords

FILE — in optional type=string default=brewer.tbl
 filename of color table file; this is present to make VIS_LOADCT completely implement LOADCT's interface, it would normally not be used

_REF_EXTRA — in out optional type=keyword
 keywords to LOADCT

Author information

Copyright:
 Color tables accessed with VIS_LOADCT and VIS_XLOADCT are provided courtesy of Brewer, Cynthia A., 2007. <http://www.ColorBrewer.org>, accessed 20 October 2007.

Apache-Style Software License for ColorBrewer software and ColorBrewer Color Schemes

Copyright (c) 2002 Cynthia Brewer, Mark Harrower, and The Pennsylvania State University.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1.6. vis_rgb2index.pro

VIS_RGB2INDEX

```
result = vis_rgb2index(rgb)
```

Convert RGB coordinates of colors to the decomposed color indices of the colors. Color indices are long integers used in decomposed color in direct graphics where the lowest order byte value is the red value, the next byte is the green value, the next byte is the blue value, and the highest order byte value is unused.

Return value

long or lonarr(n)

Parameters

rgb — in required type=bytarr
either bytarr(3) or bytarr(n, 3) array of RGB coordinates of colors

Examples

- For example:

```
IDL> print, vis_rgb2index([255, 255, 255]), format='(Z06)' ; white
FFFFFF
IDL> print, vis_rgb2index([255, 255, 0]), format='(Z06)' ; yellow
00FFFF
IDL> print, vis_rgb2index([0, 0, 255]), format='(Z06)' ; blue
FF0000
```

Multiple RGB triplets can also be passed to VIS_RGB2INDEX in an n by 3 byte array:

```
IDL> vis_loadct, 5
% LOADCT: Loading table PuBu (Sequential)
IDL> tvlct, rgb, /get
IDL> print, vis_rgb2index(rgb), format='(8Z)'
```

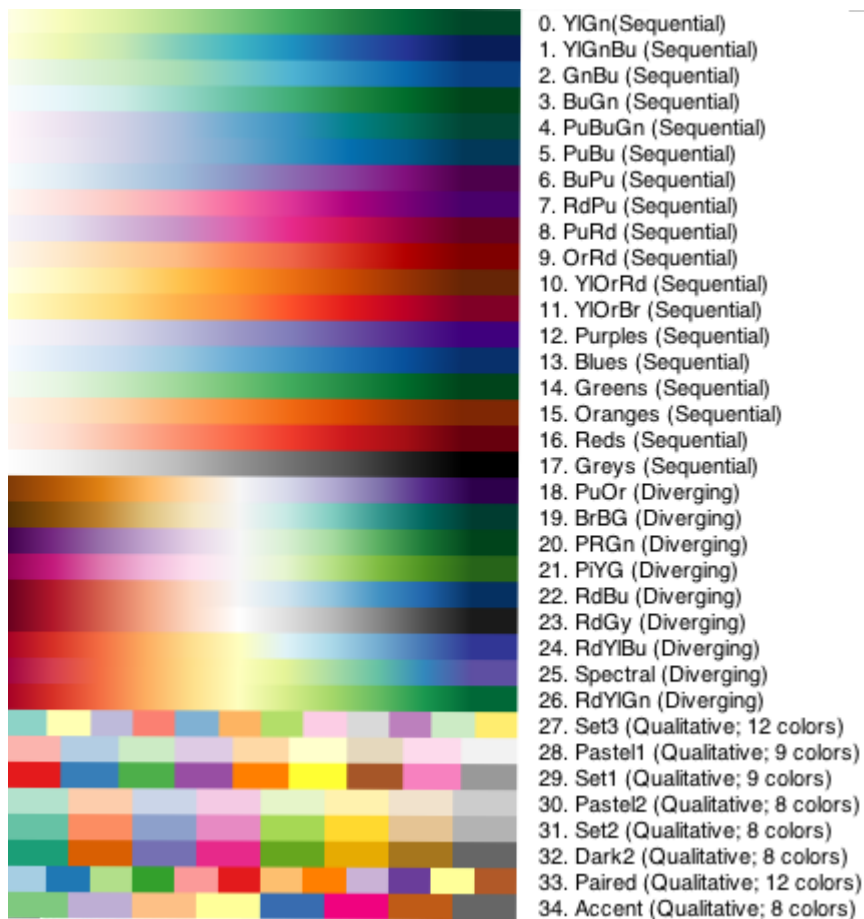
1.7. vis_xloadct.pro

VIS_XLOADCT

```
vis_xloadct [, file=string] [, _extra=keyword]
```

Load a Brewer color table by index using a GUI interface. This routine is directly analogous to XLOADCT, but with different color tables.

The Brewer color tables are split into three types: sequential, diverging, and qualitative. Sequential color tables are simple sequences from white to a given color. The diverging color tables have white in the middle of the color table and progress in each direction towards two different colors. The qualitative color tables contain only a few colors for labeling purposes. The qualitative color tables are expanded to take up the same space of the other color tables in the graphic below:



Keywords

FILE — in optional type=string default=brewer.tbl

filename of color table file; this is present to make VIS_XLOADCT completely implement XLOADCT's interface, it would normally not be used

_EXTRA — in out optional type=keyword
keywords to LOADCT

Author information

Copyright:

Color tables accessed with VIS_LOADCT and VIS_XLOADCT are provided courtesy of Brewer, Cynthia A., 2007. <http://www.ColorBrewer.org>, accessed 20 October 2007.

Apache-Style Software License for ColorBrewer software and ColorBrewer Color Schemes

Copyright (c) 2002 Cynthia Brewer, Mark Harrower, and The Pennsylvania State University.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1.8. *visgrpalette__define.pro*

Author information

Copyright

Color tables accessed with VIS_LOADCT and VIS_XLOADCT are provided courtesy of Brewer, Cynthia A., 2007. <http://www.ColorBrewer.org>, accessed 20 October 2007.

Apache-Style Software License for ColorBrewer software and ColorBrewer Color Schemes

Copyright (c) 2002 Cynthia Brewer, Mark Harrower, and The Pennsylvania State University.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Routines

`visgrpalette::loadCT [, tableNum] [, file=string]`

Load a Brewer color table by index.

`visgrpalette__define`
Define instance variables.

Routine details

VISGRPALETTE::LOADCT

`visgrpalette::loadCT [, tableNum] [, file=string]`
Load a Brewer color table by index.

Parameters

tableNum — in optional type=long
table number, 0-34 if using default color table file

Keywords

FILE — in optional type=string default=brewer.tbl
filename of color table file; this is present to make VISgrPalette completely implement IDLgrPalette's interface, it would normally not be used

VISGRPALETTE__DEFINE

`visgrpalette__define`
Define instance variables.

Chapter 2. *directgraphics/*

2.1. Overview

The *.pro* files in this directory are listed below.

vis_contour.pro

Wrapper for CONTOUR that handles NLEVELS keyword better.

vis_psbegin.pro

Set IDL direct graphics system to PostScript plotting.

vis_psend.pro

Used in conjunction with VIS_PSBEGIN to end PostScript output.

visdgvars__define.pro

Object to save/restore direct graphics system variables.

2.2. *vis_contour.pro*

VIS_CONTOUR

vis_contour, *z* [, *x*] [, *y*] [, *nlevels*=integer] [, *levels*=fltarr] [, *_extra*=keywords]

Wrapper for CONTOUR that handles NLEVELS keyword better.

Parameters

z — in required type=fltarr(m, n)

2D array to be plotted

x — in optional type=fltarr(m)

values for x-axis

y — in optional type=fltarr(n)

values for y-axis

Keywords

NLEVELS — in optional type=integer

number of contour levels

LEVELS — in out optional type=fltarr

values for isocline levels; output if nlevels is set

_EXTRA — in optional type=keywords

keywords to CONTOUR

2.3. vis_psbegin.pro

VIS_PSBEGIN

`vis_psbegin [, /image], charsize=charsize, thick=thick, symsize=symsize [, _extra=keywords]`
Set IDL direct graphics system to PostScript plotting.

Keywords

IMAGE — in optional type=boolean
set to configure PostScript with a few defaults specific to converting the PostScript output to an image format later

CHARSIZE —

THICK —

SYMSIZE —

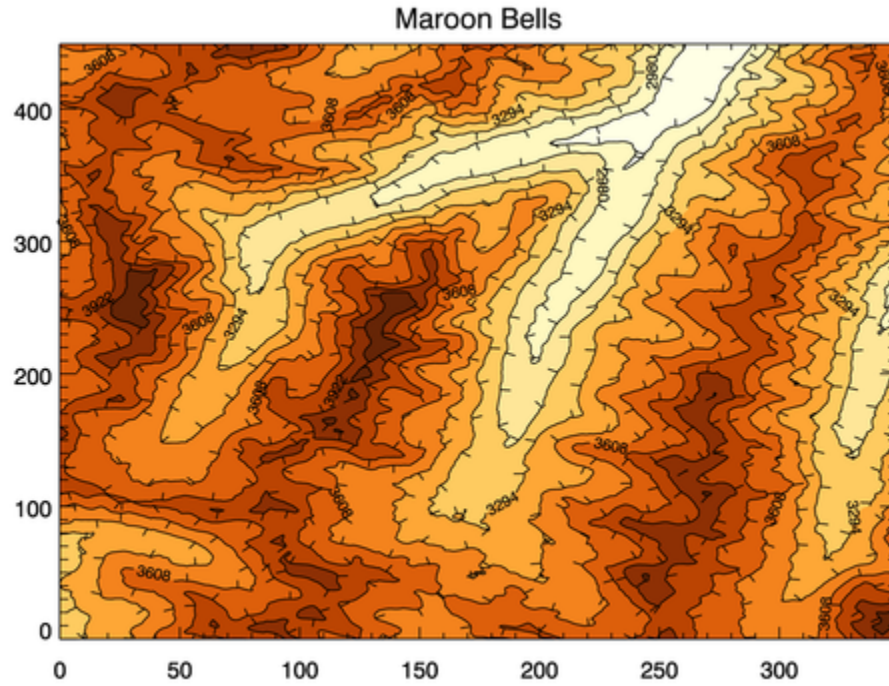
_EXTRA — in optional type=keywords
keywords to DEVICE to configure the PostScript device

Examples

- Running the main-level program attached to this program:

```
IDL> .run vis_psbegin
```

Should produce the following image:



2.4. *vis_psend.pro*

VIS_PSEND

`vis_psend`

Used in conjunction with VIS_PSBEGIN to end PostScript output.

2.5. *visdvars__define.pro*

Routines

`visdvars::save`

Save direct graphics system variables.

`visdvars::restore`

Restore direct graphics system variables.

`visdvars::cleanup`

Free resources.

`result = visdvars::init()`

Create an visdvars object.

`visdvars__define`

Define member variables.

Routine details

VISDGVARS::SAVE

`visdgvvars::save`
Save direct graphics system variables.

VISDGVARS::RESTORE

`visdgvvars::restore`
Restore direct graphics system variables.

VISDGVARS::CLEANUP

`visdgvvars::cleanup`
Free resources.

VISDGVARS::INIT

`result = visdgvvars::init()`
Create an `visdgvvars` object.

Return value

1B for success, 0B otherwise

VISDGVARS__DEFINE

`visdgvvars__define`
Define member variables.

Chapter 3. *flow/*

3.1. Overview

The *.pro* files in this directory are listed below.

vis_build_flow.pro

Build the flow DLM.

vis_lic.pro

Example program demonstrating the use of VIS_LIC.

vis_vel.pro

Make streamline plots of 2D vector fields.

3.2. *vis_build_flow.pro*

VIS_BUILD_FLOW

vis_build_flow

Build the flow DLM.

3.3. *vis_lic.pro*

VIS_LIC

vis_lic, u, v [, texture=bytarr(m, n)]

Compute the line integral convolution for a vector field.

Parameters

u — in required type=fltarr(m, n)

x-coordinates of vector field

v — in required type=fltarr(m, n)

y-coordinates of vector field

Keywords

TEXTURE — in optional type=bytarr(m, n)

random texture map; it is useful to use the same texture map for generating frames of a movie

3.4. vis_vel.pro

Routines

`result = vis_vel_interpolate(a, x, y)`

Bilinear interpolation.

`vis_vel_arrowhead, s`

Add the heads to the arrows.

`result = vis_vel_streamlines(u, v, nvecs=long, length=fltarr, nsteps=long [, /grid] [, stride=long] [, jitter=float])`

Compute the streamlines from each starting point.

`vis_vel, u, v [, x] [, y], overplot=overplot [, nvecs=long] [, length=float] [, nsteps=long] [, xmax=float] [, /grid] [, stride=long] [, jitter=float] [, thick=float] [, max_thick=float] [, color=color] [, axes_color=color] [, _extra=keywords]`

Draw a velocity (flow) field with streamlines following the field proportional in length to the vector field magnitude.

Routine details

VIS_VEL_INTERPOLATE

`result = vis_vel_interpolate(a, x, y)`

Bilinear interpolation.

Parameters

`a` — in required type=fltarr(m, n)
vector

`x` — in required type=fltarr(m * n)
x coords

`y` — in required type=fltarr(m * n)
y coords

VIS_VEL_ARROWHEAD

`vis_vel_arrowhead, s`

Add the heads to the arrows.

Parameters

`s` — in out required type=fltarr
array of streamlines

VIS_VEL_STREAMLINES

```
result = vis_vel_streamlines(u, v, nvecs=long, length=fltarr, nsteps=long [, /grid] [,
stride=long] [, jitter=float])
```

Compute the streamlines from each starting point.

Return value

```
fltarr(mvecs, nsteps + 3, 2)
```

Parameters

u — in required type=fltarr(m, n)
x component at each point of the vector field; must be a 2D array

v — in required type=fltarr(m, n)
y component at each point of the vector field; must be a 2D array

Keywords

NVECS — in out required type=long
number of steps in the streamline

LENGTH — in required type=fltarr
scaling factor for the length of the streamlines

NSTEPS — in required type=long
number of steps in each streamline

GRID — in optional type=boolean
set to jitter a regular grid of starting points instead of choosing completely random starting points

STRIDE — in optional type=long default=1L
stride amount through grid; only used if GRID is set

JITTER — in optional type=float default=0.5
amount to jitter elements in the grid; as a fraction of the distance between grid elements

VIS_VEL

```
vis_vel, u, v [, x] [, y], overplot=overplot [, nvecs=long] [, length=float] [, nsteps=long] [,
xmax=float] [, /grid] [, stride=long] [, jitter=float] [, thick=float] [, max_thick=float] [,
color=color] [, axes_color=color] [, _extra=keywords]
```

Draw a velocity (flow) field with streamlines following the field proportional in length to the vector field magnitude.

A random number of starting points can be picked (with NVECS=n) or a grid of starting points jittered slightly to eliminate linear patterns (with /GRID, STRIDE=3, and JITTER=jit).

NVECS random points within the (u,v) arrays are selected. For each "shot" the field (as bilinearly interpolated) at each point is followed using a vector of LENGTH length, tracing a line with NSTEPS segments. An arrow head is drawn at the end.

Parameters

- u* — in required type=fltarr(m, n)
x component at each point of the vector field; must be a 2D array
- v* — in required type=fltarr(m, n)
y component at each point of the vector field; must be a 2D array
- x* — in optional type=fltarr(m)
x axis values
- y* — in optional type=fltarr(n)
y axis values

Keywords

- OVERPLOT* —
- NVECS* — in optional type=long default=200L
number of vectors (arrows) to draw
- LENGTH* — in optional type=float default=0.1
the length of each arrow line segment expressed as a fraction of the longest vector divided by the number of steps
- NSTEPS* — in optional type=long default=10L
number of shoots or line segments for each arrow
- XMAX* — in optional type=float default=1.0
ignored; only present to implement the interface of VEL
- GRID* — in optional type=boolean
set to jitter a regular grid of starting points instead of choosing completely random starting points
- STRIDE* — in optional type=long default=1L
stride amount through grid; only used if GRID is set
- JITTER* — in optional type=float default=0.5
amount to jitter elements in the grid; as a fraction of the distance between grid elements
- THICK* — in optional type=float default=1.0
set to a constant to use that thickness for streamlines instead of thicknesses set to values proportional to the magnitude of the vector field at the point of the beginning of the streamline
- MAX_THICK* — in optional type=float default=3.0
maximum thickness to use for streamlines; ignored if THICK keyword is present
- COLOR* — in optional type=color
color of streamlines
- AXES_COLOR* — in optional type=color
color of axes
- _EXTRA* — in optional type=keywords
keywords to PLOT and PLOTS routines that plot the streamlines

Chapter 4. *googlechart/*

4.1. Overview

The *.pro* files in this directory are listed below.

vis_gc_base.pro

Interface to Google Charts API.

vis_gc_piechart.pro

Create an image of a pie chart using the Google Charts API.

vis_gc_scatter.pro

Scatter plot using Google Charts API.

vis_gc_venn.pro

Create a Venn Diagram using the Google Charts API.

4.2. *vis_gc_base.pro*

Routines

```
result = vis_gc_base_processstr( [s] [, param])
```

Process a string for inclusion in the URL: replace spaces with + signs, join multiple array elements with |'s, and add the param=.

```
result = vis_gc_base(type=string, data=numeric, range=range [, dimensions=lonarr(2)] [,  
title=string or strarr] [, label=strarr] [, legend_labels=strarr] [, legend_position=string] [,  
color=lonarr] [, background=long] [, alpha_channel=float] [, url=string])
```

Interface to Google Charts API.

Routine details

VIS_GC_BASE_PROCESSSTR

```
result = vis_gc_base_processstr( [s] [, param])
```

Process a string for inclusion in the URL: replace spaces with + signs, join multiple array elements with |'s, and add the param=. If no string is specified, the empty string will be returned.

Return value

string

Parameters

s — in optional type=string
string to process

param — in optional type=string
parameter name

VIS_GC_BASE

```
result = vis_gc_base(type=string, data=numeric, range=range [, dimensions=lonarr(2)] [,  
title=string or strarr] [, label=strarr] [, legend_labels=strarr] [, legend_position=string] [,  
color=lonarr] [, background=long] [, alpha_channel=float] [, url=string])
```

Interface to Google Charts API. Returns an image to display. The Google Charts API is documented at:

<http://code.google.com/apis/chart/>

Return value

bytarr(3, xsize, ysize)

Keywords

TYPE — in required type=string

type of chart required, options are: lc (line chart), lxy (xy points), ls (sparkline), bhs, bvs, bhg, bvg, p (pie chart), p3 (3D pie chart), v (Venn diagram), s (scatter plot), r (radar), t (map), gom (Google-o-meter)

DATA — in required type=numeric

array of data to displayed

RANGE —

DIMENSIONS — in optional type=lonarr(2) default=[200, 100]

size of returned image

TITLE — in optional type=string or strarr

title of the chart

LABEL — in optional type=strarr

chart labels (depending on type)

LEGEND_LABELS — in optional type=strarr

string array of labels for sets

LEGEND_POSITION — in optional type=string

position of legend: t (top), b (bottom), r (right), or l (left)

COLOR — in optional type=lonarr

colors of the chart

BACKGROUND — in optional type=long

color of background

ALPHA_CHANNEL — in optional type=float

transparency of chart: 0.0 for completely transparent, 1.0 for completely opaque

URL — out optional type=string

URL used by Google Charts API

Examples

- An example of using the routine is given in a main-level program at the end of this file. Run it using:

```
IDL> .run vis_gc_base
```

It produces:



4.3. vis_gc_piechart.pro

VIS_GC_PIECHART

```
result = vis_gc_piechart(slices [, dimensions=lonarr] [, /threed], title=title [, label=strarr]
[, color=lonarr] [, url=string])
```

Create an image of a pie chart using the Google Charts API.

Return value

bytarr(3, xsize, ysize)

Parameters

slices — in required type=fltarr
vector of values of slices

Keywords

DIMENSIONS — in optional type=lonarr default=[200, 100]
size of output image

THREED — in optional type=boolean
set to create a 3D pie chart; default is a 2D pie chart

TITLE —

LABEL — in optional type=strarr
labels for pie slices

COLOR — in optional type=lonarr
colors of the slices

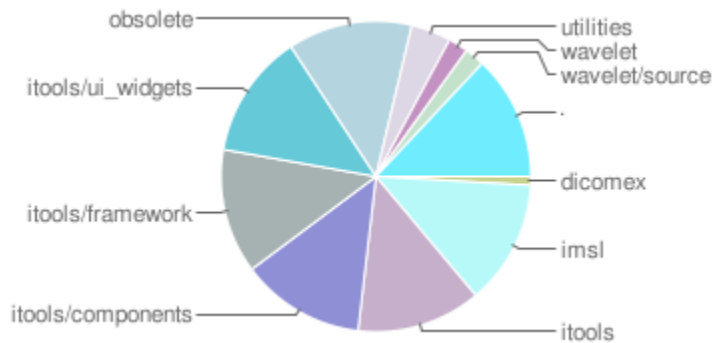
URL — out optional type=string
URL used by Google Charts API

Examples

- Running the main-level example at the end of this file:

```
IDL> .run vis_gc_piechart
```

produces:



4.4. vis_gc_scatter.pro

VIS_GC_SCATTER

```
result = vis_gc_scatter(x, y, xrange=xrange, yrange=yrange, sym_size=sym_size,  
dimensions=dimensions, url=url)
```

Scatter plot using Google Charts API.

Parameters

x —

y —

Keywords

XRANGE —

YRANGE —

SYM_SIZE —

DIMENSIONS —

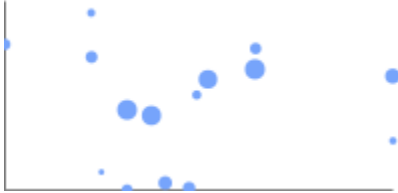
URL —

Examples

- Run the main-level example program:

```
IDL> .run vis_gc_scatter
```

It should generate:



4.5. vis_gc_venn.pro

VIS_GC_VENN

```
result = vis_gc_venn(sizes, ab, ac, bc, abc [, dimensions=lonarr] [, title=string or strarr]  
[, legend_labels=strarr] [, legend_position=string] [, color=lonarr] [, background=long] [,  
alpha_channel=float] [, url=string])
```

Create a Venn Diagram using the Google Charts API.

Return value

bytarr(3, xsize, ysize)

Parameters

sizes — in required type=fltarr(3)
relative sizes of A, B, and C

ab — in required type=float
area of intersection of A and B

ac — in required type=float
area of intersection of A and C

bc — in required type=float
area of B and C

abc — in required type=float
area of intersection of A, B, and C

Keywords

DIMENSIONS — in optional type=lonarr default=[200, 100]
size of output image

TITLE — in optional type=string or strarr
string or string array representing title

LEGEND_LABELS — in optional type=strarr
string array of labels for sets

LEGEND_POSITION — in optional type=string
position of legend: t (top), b (bottom), r (right), or l (left)

COLOR — in optional type=lonarr
colors of the slices

BACKGROUND — in optional type=long
background color of chart

ALPHA_CHANNEL — in optional type=float
transparency of chart: 0.0 for completely transparent, 1.0 for completely opaque

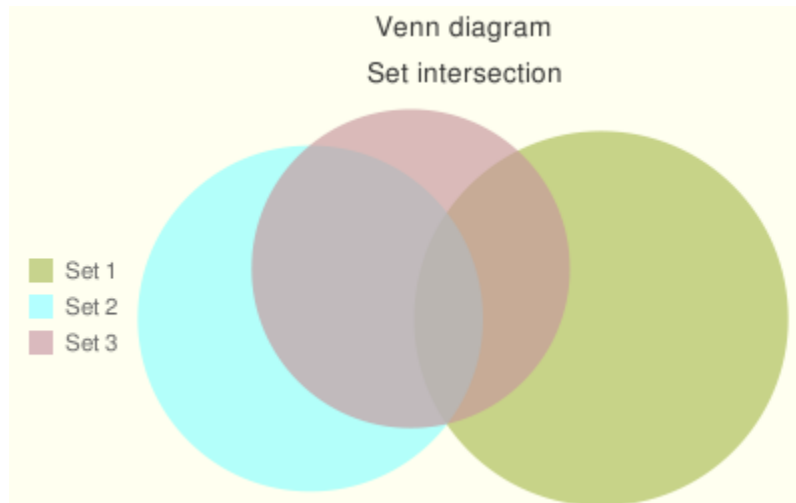
URL — out optional type=string
URL used by Google Charts API

Examples

- See the main-level program at the end of this file:

```
IDL> .run vis_gc_venn
```

This should produce:



Chapter 5. *graphs/*

5.1. Overview

The *.pro* files in this directory are listed below.

vis_graph_layout.pro
Create a [graphs graph].

vis_tree_layout.pro
Create a tree.

The *.idl* files in this directory are listed below.

graphs.idl

5.2. *vis_graph_layout.pro*

VIS_GRAPH_LAYOUT

result = *vis_graph_layout*(*graph*)
Create a [graphs graph].

Return value

IDLgrView

Parameters

graph — in required type=object
IDL_Container containing the nodes of the graph; each node should have two properties: VIS_NODE_NAME (string) and VIS_NODE_CHILDREN (array object references to other nodes or -1L if no children)

5.3. *vis_tree_layout.pro*

VIS_TREE_LAYOUT

vis_tree_layout
Create a tree.

Chapter 6. *images/*

6.1. Overview

The *.pro* files in this directory are listed below.

vis_image.pro

Displays an image scaled to a "reasonable" size with x- and y-axes.

vis_image_getsize.pro

Return the x and y size of the given image array.

vis_image_resize.pro

Resize an image similarly to CONGRID.

6.2. *vis_image.pro*

Author information

Author

Michael Galloy

Other file information

Bugs:

This routine is not set to work with PS output yet.

VIS_IMAGE

vis_image, *im* [*x*] [*y*] [*true=long*] [*_extra=keywords*]

Displays an image scaled to a "reasonable" size with x- and y-axes.

Parameters

im — in required type=image array
image array

x — in optional type=fltarr default=bindgen(xsize)
x-axis values

y — in optional type=fltarr default=bindgen(ysize)
y-axis values

Keywords

TRUE — in optional type=long

Set to 0 for (m, n) array images, 1 for (3, m, n), 2 for (m, 3, n), and 3 for (m, n, 3).

If TRUE is not present, VIS_IMAGE_GETSIZE will attempt to guess the size. 2D images will automatically be set to TRUE=0; 3D images' dimensions will be searched for a size 3 dimension.

_EXTRA — in optional type=keywords
keywords to PLOT or CONGRID routines

6.3. *vis_image_getsize.pro*

VIS_IMAGE_GETSIZE

```
result = vis_image_getsize(im [, true=long])
```

Returns the size of the image array as a two element array, [xsize, ysize]. The TRUE keyword can be set to indicate the interleave or it can be guessed if the TRUE keyword is not present.

Return value

lonarr(2)

Parameters

im — in required type=image array
image array of the form (m, n), (3, m, n), (m, 3, n), or (m, n, 3)

Keywords

TRUE — in out optional type=long
Set to 0 for (m, n) array images, 1 for (3, m, n), 2 for (m, 3, n), and 3 for (m, n, 3).

If TRUE is not present, VIS_IMAGE_GETSIZE will attempt to guess the size. 2D images will automatically be set to TRUE=0; 3D images' dimensions will be searched for a size 3 dimension.

The TRUE value used will be returned through the variable.

6.4. *vis_image_resize.pro*

VIS_IMAGE_RESIZE

```
result = vis_image_resize(im, xsize, ysize [, true=long] [, _extra=keywords])
```

Resize an image similarly to CONGRID. Advantage over congrid is that nearest neighbor interpolation is used even for multiple band images.

Return value

the resized image

Parameters

im — in required type=image array
input image array; if the image has multiple bands, use the TRUE keyword to specify which dimension contains the channels

xsize — in required type=long
xsize of the output image

ysize — in required type=long
ysize of the output image

Keywords

TRUE — in optional type=long default=0
set to specify which dimensions contains the channels; TRUE=0 is for m by n images, TRUE=1 is for 3 by m by n images, TRUE=2 is for m by 3 by n images, TRUE=3 is for m by n by 3 images

_EXTRA — in optional type=keywords
keywords to CONGRID

Chapter 7. *lineplots/*

7.1. Overview

The *.pro* files in this directory are listed below.

vis_plot.pro

Wrapper for PLOT routine which has several differences: # removes top and right axis frames by default, but may be changed through the XSTYLE or YSTYLE keywords # contracts limits of plot to exact x range

vis_scatter3d.pro

Display a 3D scatter plot.

vis_themeriver.pro

Create a theme river style plot.

7.2. *vis_plot.pro*

Routines

vis_plot_setdims

Make sure that !d is set correctly.

vis_plot [, x], y [, /slope_aspect] [, xstyle=integer] [, ystyle=integer] [, _extra=keywords]

Wrapper for PLOT routine which has several differences: # removes top and right axis frames by default, but may be changed through the XSTYLE or YSTYLE keywords # contracts limits of plot to exact x range

Routine details

VIS_PLOT_SETDIMS

vis_plot_setdims

Make sure that !d is set correctly.

VIS_PLOT

vis_plot [, x], y [, /slope_aspect] [, xstyle=integer] [, ystyle=integer] [, _extra=keywords]

Wrapper for PLOT routine which has several differences: # removes top and right axis frames by default, but may be changed through the XSTYLE or YSTYLE keywords # contracts limits of plot to exact x range

Parameters

x — in optional type=1D numeric array
x values for plot, defaults just to findgen(n)

y — in required type=1D numeric array
y values for plot

Keywords

SLOPE_ASPECT — in optional type=boolean
set to make the average slope of line segments (on display) +/- 1

XSTYLE — in optional type=integer default=9
XSTYLE keyword from PLOT

YSTYLE — in optional type=integer default=9
YSTYLE keyword from PLOT

_EXTRA — in optional type=keywords
keywords to PLOT

7.3. vis_scatter3d.pro

VIS_SCATTER3D

vis_scatter3d, *x*, *y*, *z* [, *_extra*=keywords]
Display a 3D scatter plot.

Parameters

x — in required type=fltarr
y — in required type=fltarr
z — in required type=fltarr

Keywords

_EXTRA — in optional type=keywords
graphics keywords to SURFACE and PLOTS

7.4. vis_themeriver.pro

VIS_THEMERIVER

vis_themeriver, *x*, *data*, *colors* [, *show_lines*=lonarr] [, *axis_color*=color] [, *color*=color] [, *_extra*=keywords]
Create a theme river style plot.

Parameters

x — in required type=fltarr(n)
x-coordinates of data

data — in required type=fltarr(nlines, n)
multiple y-coordinates of data values (nlines number of datasets)

colors — in required type=bytarr(nlines - 1)
colors of shaded regions between datasets (starting from the bottom)

Keywords

SHOW_LINES — in optional type=lonarr
indices of dataset lines in data to overplot

AXIS_COLOR — in optional type=color
color of axis

COLOR — in optional type=color
color of lines

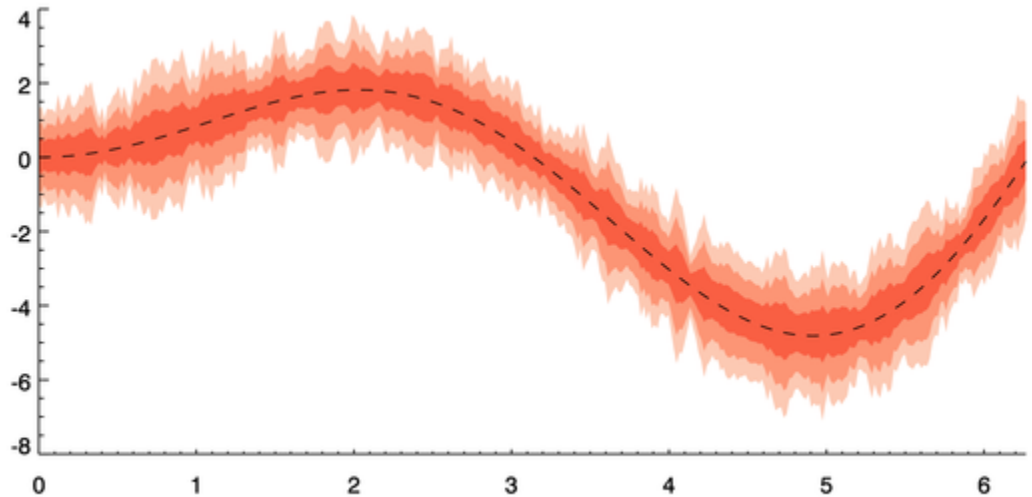
_EXTRA — in optional type=keywords
keywords to plot (for axis) and oplot (for dataset lines overplotted)

Examples

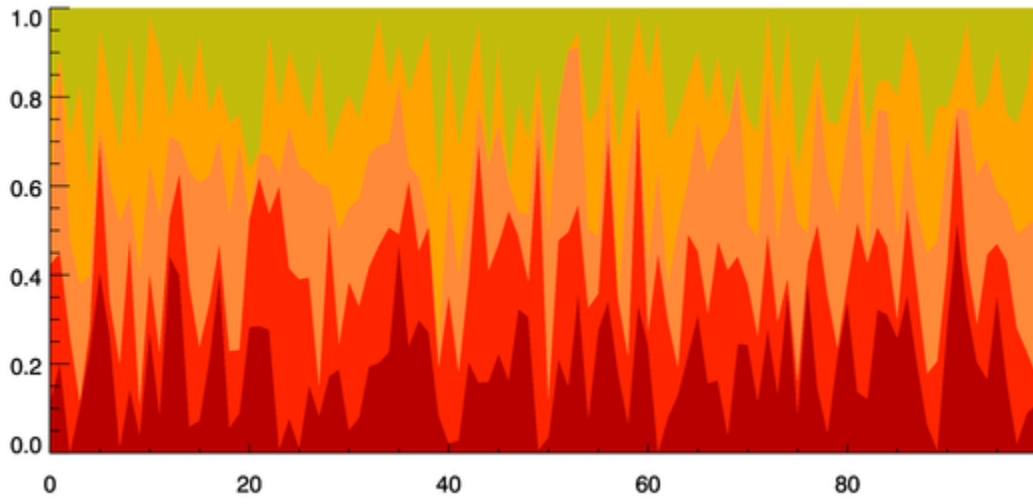
- See the main-level program at the end of this file:

```
IDL> .run vis_themeriver
```

The first example is similar to an error plot:



The second example produces a stacked plot like:



Chapter 8. *objectgraphics/*

8.1. Overview

The *.pro* files in this directory are listed below.

visgrcube__define.pro

Unit cube polygon (maybe scaled and translated).

8.2. *visgrcube__define.pro*

Routines

visgrcube::_computeVertices [, x] [, y] [, z]

Compute the new vertices of the cube.

visgrcube::setProperty, scale=scale, translate=translate, _extra=_extra

Set properties.

visgrcube::getProperty, scale=scale, translate=translate, _ref_extra=_ref_extra

Get properties.

result = visgrcube::init([scale=float/fltarr(3)] [, translate=fltarr(3)] [, _extra=keywords])

Create a cube polygon.

visgrcube__define

Define instance variables.

Routine details

VISGRCUBE::_COMPUTEVERTICES

visgrcube::_computeVertices [, x] [, y] [, z]

Compute the new vertices of the cube.

Parameters

x — out optional type=fltarr(8)

x-coordinates of vertices

y — out optional type=fltarr(8)

y-coordinates of vertices

z — out optional type=fltarr(8)

z-coordinates of vertices

VISGRCUBE::SETPROPERTY

visgrcube::setProperty, scale=scale, translate=translate, _extra=_extra

Set properties.

Keywords

SCALE —

TRANSLATE —

_EXTRA —

VISGRUCUBE::GETPROPERTY

`visgrcube::getProperty, scale=scale, translate=translate, _ref_extra=_ref_extra`
Get properties.

Keywords

SCALE —

TRANSLATE —

_REF_EXTRA —

VISGRUCUBE::INIT

`result = visgrcube::init([scale=float/fltarr(3)] [, translate=fltarr(3)] [, _extra=keywords])`
Create a cube polygon.

Return value

1 for success, 0 for failure

Keywords

SCALE — in optional type=float/fltarr(3)
scale cube either equally or separately in each direction

TRANSLATE — in optional type=fltarr(3)
translate the cube to another location

_EXTRA — in optional type=keywords
keywords to IDLgrPolygon::init

VISGRUCUBE__DEFINE

`visgrcube__define`
Define instance variables.

Chapter 9. *povray/*

9.1. Overview

The *.pro* files in this directory are listed below.

visgrpovray__define.pro
POV-Ray object graphics destination.

9.2. *visgrpovray__define.pro*

Other file information

Todo:

This implementation is not complete; many object graphics classes are not handled at all and no classes' properties are fully implemented. Also, there are difficulties because the POV-Ray coordinate system is left-handed while the object graphics system is right-handed. No attempt to reconcile these is made right now.

Routines

result = visgrpovray::_getRgb(color)

Convert an RGB 3-element byte array to a POV-Ray string specifying the color, like [255, 0, 0] to '<1.0, 0.0, 0.0>'.
</p></div>
<div data-bbox="144 560 584 575" data-label="Text"><p><i>visgrpovray::_writeVertices, lun, vertices, name=string</i></p></div>
<div data-bbox="174 577 478 593" data-label="Text"><p>Write a list of vertices to the open output file.</p></div>
<div data-bbox="144 602 498 617" data-label="Text"><p><i>visgrpovray::_writeTransform, lun, transform</i></p></div>
<div data-bbox="174 619 880 653" data-label="Text"><p>Write the matrix property of the POV-Ray object (the current transformation matrix of the object graphics object).</p></div>
<div data-bbox="144 662 544 677" data-label="Text"><p><i>result = visgrpovray::_getFilename(object, prefix)</i></p></div>
<div data-bbox="174 679 521 695" data-label="Text"><p>Create a unique name for the given object's .inc file.</p></div>
<div data-bbox="144 704 623 719" data-label="Text"><p><i>visgrpovray::_writeSurface, surface, prefix, includes=strarr</i></p></div>
<div data-bbox="174 721 480 737" data-label="Text"><p>Writes output for a subclass of IDLgrSurface.</p></div>
<div data-bbox="144 746 623 761" data-label="Text"><p><i>visgrpovray::_writePolygon, polygon, prefix, includes=strarr</i></p></div>
<div data-bbox="174 763 484 779" data-label="Text"><p>Writes output for a subclass of IDLgrPolygon.</p></div>
<div data-bbox="144 788 593 803" data-label="Text"><p><i>visgrpovray::_writeLight, light, prefix, includes=strarr</i></p></div>
<div data-bbox="174 805 466 821" data-label="Text"><p>Writes output for a subclass of IDLgrLight.</p></div>
<div data-bbox="144 830 506 845" data-label="Text"><p><i>visgrpovray::_writePov, tree, includes=strarr</i></p></div>
<div data-bbox="174 847 304 863" data-label="Text"><p>Write the .pov file.</p></div>
<div data-bbox="144 872 324 887" data-label="Text"><p><i>visgrpovray::_writeIni</i></p></div>
<div data-bbox="174 889 297 904" data-label="Text"><p>Write the .ini file.</p></div>
<div data-bbox="892 951 920 967" data-label="Page-Footer"><p>41</p></div>

`visgrpovray::_traverse, tree, prefix, includes=strarr`

Traverse the object graphics hierarchy rooted from the given node.

`visgrpovray::draw [, tree]`

Write the object graphics rooted at the specified scene or view.

`visgrpovray::setProperty, file_prefix=file_prefix, dimensions=dimensions`

Set properties.

`visgrpovray::getProperty, file_prefix=file_prefix, dimensions=dimensions`

Get properties.

`visgrpovray::cleanup`

Free resources.

`result = visgrpovray::init([file_prefix=string] [, dimensions=lonarr(2)] [, graphics_tree=object])`

Create POV-Ray output destination for object graphics.

`visgrpovray__define`

Define instance variables.

Routine details

VISGRPOVRAY::_GETRGB

`result = visgrpovray::_getRgb(color)`

Convert an RGB 3-element byte array to a POV-Ray string specifying the color, like [255, 0, 0] to '<1.0, 0.0, 0.0>'.
</p></div><div data-bbox="84 554 174 568" data-label="Section-Header"><h4>Return value</h4></div><div data-bbox="144 583 424 599" data-label="Text"><p>POV-Ray string representation of a color</p></div><div data-bbox="84 613 163 628" data-label="Section-Header"><h4>Parameters</h4></div><div data-bbox="115 644 384 660" data-label="Text"><p><code>color</code> — in required type=bytarr(3)</p></div><div data-bbox="144 661 324 677" data-label="Text"><p>object graphics style color</p></div><div data-bbox="84 693 373 709" data-label="Section-Header"><h3>VISGRPOVRAY::_WRITEVERTICES</h3></div><div data-bbox="115 726 555 741" data-label="Text"><p><code>visgrpovray::_writeVertices, lun, vertices, name=string</code></p></div><div data-bbox="144 742 449 759" data-label="Text"><p>Write a list of vertices to the open output file.</p></div><div data-bbox="84 786 163 801" data-label="Section-Header"><h4>Parameters</h4></div><div data-bbox="115 816 333 832" data-label="Text"><p><code>lun</code> — in required type=long</p></div><div data-bbox="144 833 268 850" data-label="Text"><p>lun for output file</p></div><div data-bbox="115 858 424 875" data-label="Text"><p><code>vertices</code> — in required type=fltarr(n, 3)</p></div><div data-bbox="144 875 234 891" data-label="Text"><p>vertices data</p></div><div data-bbox="84 951 108 967" data-label="Page-Footer"><p>42</p></div>

Keywords

NAME — in required type=string
name of section

VISGRPOVRAY::_WRITETRANSFORM

visgrpovray::_writeTransform, *lun*, *transform*
Write the matrix property of the POV-Ray object (the current transformation matrix of the object graphics object).

Parameters

lun — in required type=long
logical unit number to write to

transform — in required type=fltarr(4, 4)
object graphics transformation matrix

VISGRPOVRAY::_GETFILENAME

result = *visgrpovray::_getFilename(object, prefix)*
Create a unique name for the given object's .inc file.

Return value

string filename for object

Parameters

object — in required type=object
object graphics atom object

prefix — in required type=string
string prefix for the name

VISGRPOVRAY::_WRITESURFACE

visgrpovray::_writeSurface, *surface*, *prefix*, *includes=strarr*
Writes output for a subclass of IDLgrSurface.

Parameters

surface — in required type=object
subclass of IDLgrSurface

prefix — in required type=string
path to polygon object

Keywords

INCLUDES — in out required type=strarr
tacks on the filename of the .inc file to this variable; undefined to represent the empty list

VISGRPOVRAY::_WRITEPOLYGON

visgrpovray::_writePolygon, *polygon*, *prefix*, *includes*=strarr
Writes output for a subclass of IDLgrPolygon.

Parameters

polygon — in required type=object
subclass of IDLgrPolygon
prefix — in required type=string
path to polygon object

Keywords

INCLUDES — in out required type=strarr
tacks on the filename of the .inc file to this variable; undefined to represent the empty list

VISGRPOVRAY::_WRITELIGHT

visgrpovray::_writelight, *light*, *prefix*, *includes*=strarr
Writes output for a subclass of IDLgrLight.

Parameters

light — in required type=object
subclass of IDLgrLight
prefix — in required type=string
path to polygon object

Keywords

INCLUDES — in out required type=strarr
tacks on the filename of the .inc file to this variable; undefined to represent the empty list

VISGRPOVRAY::_WRITEPOV

visgrpovray::_writePov, *tree*, *includes*=strarr
Write the .pov file.

Parameters

tree — in required type=object
object graphics tree to traverse

Keywords

INCLUDES — in out required type=strarr
tacks on the filename of the .inc file to this variable; undefined to represent the empty list

VISGRPOVRAY::_WRITEINI

visgrpovray::_writeIni
Write the .ini file.

VISGRPOVRAY::_TRAVERSE

visgrpovray::_traverse, *tree*, *prefix*, *includes*=strarr
Traverse the object graphics hierarchy rooted from the given node.

Parameters

tree — in required type=object
any node (including children) in the object graphics hierarchy to process
prefix — in required type=string
filename prefix to add to generated files

Keywords

INCLUDES — in out required type=strarr
.inc files that have already been written, starts off as undefined

VISGRPOVRAY::DRAW

visgrpovray::draw [*tree*]
Write the object graphics rooted at the specified scene or view.

Parameters

tree — in optional type=object
scene or view object

VISGRPOVRAY::SETPROPERTY

visgrpovray::setProperty, *file_prefix*=file_prefix, *dimensions*=dimensions
Set properties.

Keywords

FILE_PREFIX —
DIMENSIONS —

VISGRPOVRAY::GETPROPERTY

```
visgrpovray::getProperty, file_prefix=file_prefix, dimensions=dimensions  
Get properties.
```

Keywords

FILE_PREFIX —

DIMENSIONS —

VISGRPOVRAY::CLEANUP

```
visgrpovray::cleanup  
Free resources.
```

VISGRPOVRAY::INIT

```
result = visgrpovray::init( [file_prefix=string] [, dimensions=lonarr(2)] [,  
graphics_tree=object])  
Create POV-Ray output destination for object graphics.
```

Keywords

FILE_PREFIX — in optional type=string default='idlgr'
prefix to add to all output files; final result will be:

```
file_prefix + '.png'
```

DIMENSIONS — in optional type=lonarr(2) default=[400, 400]
default image size of the POV-Ray result

GRAPHICS_TREE — in optional type=object
graphics tree to tree if none is provided to draw method

VISGRPOVRAY__DEFINE

```
visgrpovray__define  
Define instance variables.
```

Chapter 10. *text/*

10.1. Overview

The *.pro* files in this directory are listed below.

vis_strwrap.pro
Wrap a string to a given width.

10.2. *vis_strwrap.pro*

VIS_STRWRAP

```
result = vis_strwrap(text, width [, charsize=float] [, charthick=float] [, font=integer])
```

Wrap a string to a given width.

Return value

string array

Parameters

text — in required type=string
scalar string to wrap

width — in required type=long
width in pixels of the text area

Keywords

CHARSIZE — in optional type=float
CHARSIZE keyword to XYOUTS

CHARTHICK — in optional type=float
CHARTHICK keyword to XYOUTS

FONT — in optional type=integer
FONT keyword to XYOUTS

Examples

- To run a simple example:

```
IDL> .run vis_strwrap
```


Chapter 11. *util/*

11.1. Overview

The *.pro* files in this directory are listed below.

vis_convert.pro

Use ImageMagick to convert a file between formats.

vis_make_dll.pro

Wrapper for MAKE_DLL that handles input and output directories more intelligently.

vis_src_root.pro

Returns the directory name (with a trailing slash) of the location of the source code for the routine that called this function.

11.2. *vis_convert.pro*

VIS_CONVERT

```
vis_convert [, basename] [, density=long] [, max_dimensions=lonarr(2)], scale=scale [, /from_ps]
[, /to_png] [, command=string]
```

Use ImageMagick to convert a file between formats.

Parameters

basename — in optional type=string

basename of file to convert (used for output name as well)

Keywords

DENSITY — in optional type=long default=300

density of output image in dots per inch

MAX_DIMENSIONS — in optional type=lonarr(2)

maximum dimensions for the output image in pixels

SCALE —

FROM_PS — in optional type=boolean

if set, indicates the input is a PostScript file

TO_PNG — in optional type=boolean

if set, indicates the output should a PNG image file

COMMAND — out optional type=string

convert command

11.3. *vis_make_dll.pro*

VIS_MAKE_DLL

vis_make_dll, *cfile* [, *_extra*=keywords]

Wrapper for MAKE_DLL that handles input and output directories more intelligently.

Parameters

cfile — in required type=string
C filename to create DLL from

Keywords

_EXTRA — in optional type=keywords
keywords to MAKE_DLL

11.4. *vis_src_root.pro*

VIS_SRC_ROOT

result = *vis_src_root*()

Returns the directory name (with a trailing slash) of the location of the source code for the routine that called this function.

Return value

string