NAME

gdtclft − render images in various bitmap formats (GD, GD2, GIF, JPEG, PNG, WBMP, XBM)

DESCRIPTION

TCL GD EXTENSION

Thomas Boutell's Gd package provides a convenient way to generate bitmap images with a C program. If you, like me, prefer Tcl for CGI applications, you'll want my TCL GD extension. You can get it by anonymous FTP from ftp://guraldi.hgp.med.umich.edu/pub/gdtcl.shar.

Here's a quick overview of the package.

* Overview
* Installation
* Reference
* Examples
  + gdsample -- sample program written in Tcl.
  + Gddemo -- demo program written in Tcl.
  + gdshow -- procedure to display an image.

A TCL INTERFACE TO THE GD PACKAGE

Spencer W. Thomas
Human Genome Center
University of Michigan
Ann Arbor, MI 48109

spencer.thomas@med.umich.edu

TrueType font support using the FreeType library was added by John Ellson (ellson@graphviz.org)

Latest sources available from:

http://www.graphviz.org/pub/

Overview

This package provides a simple Tcl interface to the gd (bitmap drawing) package, version 1.1. It includes an interface to all the gd functions and data structures from Tcl commands.

Installation

```
./configure
make
make install
```

Reference

One Tcl command, 'gd', is added. All gd package actions are sub-commands (or "options" in Tcl terminology) of this command.

Each active gd image is referred to with a "handle". The handle is a name of the form gd# (e.g., gd0) returned by the gd create options.

Almost all the gd commands take a handle as the first argument (after the option). All the drawing commands take a color_idx as the next argument.

gd create <width> <height>
Return a handle to a new gdImage that is width X height.

gd createTrueColor <width> <height>
Return a handle to a new trueColor gdImage that is width X height.

gd createFromGD <filehandle>
gd createFromGD2 <filehandle>
gd createFromGIF <filehandle>
gd createFromJPEG <filehandle>
gd createFromPNG <filehandle>
gd createFromWBMP <filehandle>
gd createFromXBM <filehandle>
Return a handle to a new gdImage created by reading an image in the indicate format from the file open on filehandle.

gd destroy <gdhandle>
Destroy the gdImage referred to by gdhandle.

gd writeGD <gdhandle> <filehandle>
gd writeGD2 <gdhandle> <filehandle>
gd writeGIF <gdhandle> <filehandle>
gd writeJPEG <gdhandle> <filehandle>
gd writePNG <gdhandle> <filehandle>
gd writeWBMP <gdhandle> <filehandle>
gd writeXBM <gdhandle> <filehandle>
Write the image in gdhandle to filehandle in the format indicated.

gd writePNGvar <gdhandle> <varname>
Write the image in gdhandle to Tcl variable "varname" as a binary coded PNG object.

gd interlace <gdhandle> <on-off>
Make the output image interlaced (if on-off is true) or not (if on-off is false).

gd color new <gdhandle> <red> <green> <blue>
Allocate a new color with the given RGB values. Returns the color_idx, or −1 on failure (256 colors already allocated).

gd color exact <gdhandle> <red> <green> <blue>
Find a color_idx in the image that exactly matches the given RGB color. Returns the color_idx, or −1 if no exact match.

gd color closest <gdhandle> <red> <green> <blue>
Find a color in the image that is closest to the given RGB color.

Guaranteed to return a color idx.

gd color resolve <gdhandle> <red> <green> <blue>
Return the index of the best possible effort to get a color.
Guaranteed to return a color idx. Equivalent to:

```
if {[set idx [gd color exact $gd $r $g $b]] == -1} {
    if {[set idx [gd color neW $Gd $r $g $b]] == -1} {
        set idx [gd color closest $gd $r $g $b]
    }
}
```

gd color free <gdhandle> <color_idx>
Free the color at the given color_idx for reuse.

gd color transparent <gdhandle> [<color_idx>]
Mark the color at <color_idx> as the transparent background color. Or, return the transparent color_idx if no color_idx specified.

gd color get <gdhandle> [<color_idx>]
Return the RGB value at <color_idx>, or {} if it is not allocated. If <color_idx> is not specified, return a list of {color_idx R G B} values for all allocated colors.

gd brush <gdhandle> <brushhandle>
Set the brush image to be used for brushed lines. Transparent pixels in the brush will not change the image when the brush is applied.

gd style <gdhandle> <color_idx> ...
Set the line style to the list of color indices. This is interpreted in one of two ways. For a simple styled line, each color is applied to points along the line in turn. The transparent color_idx value may be used to leave gaps in the line. For a styled, brushed line, a 0 (or the transparent color_idx) means not to fill the pixel, and a non-zero value means to apply the brush.

gd tile <gdhandle> <tilehandle>
Set the tile image to be used for tiled fills. Transparent pixels in the tile will not change the underlying image during tiling.

In all drawing functions, the color_idx is a number, or may be one of the strings styled, brushed, tiled, "styled brushed" or "brushed styled". The style, brush, or tile currently in effect will be used. Brushing and styling apply to lines, tiling to filled areas.

gd set <gdhandle> <color_idx> <x> <y>
Set the pixel at (x,y) to color <color_idx>.

gd line <gdhandle> <color_idx> <x1> <y1> <x2> <y2>

gd rectangle <gdhandle> <color_idx> <x1> <y1> <x2> <y2>

gd fillrectangle <gdhandle> <color_idx> <x1> <y1> <x2> <y2>
Draw the outline of (resp. fill) a rectangle in color <color_idx>
with corners at (x1,y1) and (x2,y2).

gd arc <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd fillarc <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd openarc <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd chord <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd fillchord <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd openchord <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd pie <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd fillpie <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
gd openpie <gdhandle> <color_idx> <cx> <cy> <width> <height> <start> <end>
All describe an arc based shape in color <color_idx>, centered at (cx,cy)
in a rectangle width x height, starting at start degrees and ending
at end degrees.

arc - Just the curved line.
fillarc - (Intented to be a fill between the curve and chord,
but gd doesn't do that) - Same as pie.
openarc - Outline shape with curve and chord.
chord - Straight line chord between the ends of the curve,
but without showing the curve.
fillchord - Filled triangle between chord and center.
openchord - Outline triangle between chord and center.
pie - Filled pie segment between curve and center.
fillpie - Same as pie.
openpie - Outline pie segment between curve and center.

gd polygon <gdhandle> <color_idx> <x1> <y1> ...

gd fillpolygon <gdhandle> <color_idx> <x1> <y1> ...
Draw the outline of, or fill, a polygon specified by the x, y
coordinate list. There must be at least 3 points specified.

gd fill <gdhandle> <color_idx> <x> <y>

gd fill <gdhandle> <color_idx> <x> <y> <borderindex>
Fill with color <color_idx>, starting from (x,y) within a region of
pixels all the color of the pixel at (x,y) (resp., within a
border colored borderindex).

gd size <gdhandle>
Returns a list {width height} of the image.

gd text <gdhandle> <color_idx> <fontlist> <size> <angle> <x> <y> <string>
Draw text using <fontlist> in color <color_idx>,
with pointsize <size>, rotation in radians <angle>, with lower left
corner at (x,y). String may contain UTF8 sequences like: "À"

Returns 4 corner coords of bounding rectangle.
Use gdhandle = {} to get boundary without rendering.
Use negative of color_idx to disable antialiasing.

<fontlist> may contain either a full pathname of a font, including ".ttf" extension, or it may contain a space-separated list of alternate names for a font, without the ".ttf". e.g.

"Times-Roman times"

The file <name>.ttf corresponding to one of the alternate names must be found in the built-in DEFAULT_FONTPATH, or in the fontpath specified in a GDFONTPATH environment variable.

gd copy <desthandle> <srchandle> <destx> <desty> <srcx> <srcy> <w> <h>

gd copy <desthandle> <srchandle> <destx> <desty> <srcx> <srcy> <destw> <desth> <srcw> <srch> Copy a subimag

srchandle(srcx, srcy) to desthandle(destx, desty), size w x h. Or, resize the subimage in copying from srcw x srch to destw x desth.

Examples

The sample program from the gd documentation can be written thusly:

```
#!/bin/sh
# next line is a comment in tcl exec tclsh "$0" ${1+"$@"}

package require Gdtclft

######################################################################
# Sample gdtcl program  - from gdtclft man page
#
# Create a 64 x 64 image
set im [gd create 64 64]

# Get black and white as colors.  Black is the background color because
# it is allocated first from a new image.

set black [gd color new $im 0 0 0]
set white [gd color new $im 255 255 255]

# Draw a line from upper left to lower right
gd line $im $white 0 0 63 63

# Open a file for writing (Tcl on Unix, at least, doesn't support 'wb' mode)
set out [open test.png w]

# Output the image to the disk file
gd writePNG $im $out

# Close the file
close $out

# Destroy the image in memory
gd destroy $im
```

GDDEMO

Here’s the gddemo.c program translated to tcl.

```
#!/bin/sh
# next line is a comment in tcl exec tclsh "$0" ${1+"$@"}

package require Gdtclft

######################################################################
#
# gddemo in tcl
#

# open demoin.png or die
if {[catch {open demoin.png r} in]} {
  puts stderr "Can’t load source image; this demo is much";
  puts stderr "more impressive if demoin.png is available";
  exit
}

# Create output image 128 x 128
set im_out [gd create 128 128]

# First color is background
set white [gd color new $im_out 255 255 255]

# Set transparent
gd color transparent $im_out $white

# Load demoin.png and paste part of it into the output image.
set im_in [gd createFromPNG $in]
close $in

# Copy and shrink
gd copy $im_out $im_in 16 16 0 0 96 96 128 128

# Get some colors
set red [gd color new $im_out 255 0 0]
set green [gd color new $im_out 0 255 0]
set blue [gd color new $im_out 0 0 255]

# Draw a rectangle
gd line $im_out $green 8 8 120 8
gd line $im_out $green 120 8 120 120
gd line $im_out $green 120 120 8 120
gd line $im_out $green 8 120 8 8

# Text
gd text $im_out $red arial 20 0 16 16 hi
gd text $im_out $red arial 20 90 23 23 hi

# Circle
gd arc $im_out $blue 64 64 30 10 0 360
```

```
# Arc
gd arc $im_out $blue 64 64 20 20 45 135

# Flood fill
gd fill $im_out $blue 4 4

# Polygon
gd fillpolygon $im_out $green 32 0 0 64 64 64

# Brush. A fairly wild example also involving a line style!
if {$im_in != ""} {
  set brush [gd create 8 8];
  eval [concat gd copy $brush $im_in 0 0 0 0 [gd size $brush] [gd size $im_in]]
  gd brush $im_out $brush
  # Style so they won't overprint each other.
  gd style $im_out "0 0 0 0 0 0 0 1"
  gd line $im_out "styled brushed" 0 0 128 128
}

# Interlace the result for "fade in" in viewers that support it
gd interlace $im_out true

# Write PNG
set out [open demoout.png w]
gd writePNG $im_out $out
close $out
gd destroy $im_out
```

GDSHOW

A quick Tcl procedure to display a GD image using the xv program.

```
######################################################################
# gdshow -- use xv to display an image.
#
# Waits until xv quits to return.
#
proc gdshow {gd} {
  set f [open "|xv -" w]
  catch {gd writePNG $gd $f}
  catch {close $f} xx
  if {$xx != {}} {
    error "XV error: $xx"
  }
}
```