NAME

akfavatar-graphic − module to display graphics in Lua-AKFAvatar

SYNOPSIS

local graphic = require "akfavatar-graphic"

DESCRIPTION

Module for Lua-AKFAvatar to display graphics.

Notes:

– Coordinates start with 1, 1 in the upper left corner.

– It can also be used in the style of turtle graphic.

– The pen size, position, heading, pen color and background color are set for each graphic.

– The code is not optimized for speed. You can do animations, but they won't be smooth.

Functions / Methods

graphic.new(*[width, height] [, background-color]***)**

Creates a new graphic (a canvas).

When *width* and *height* are not given it uses the whole window/screen.
A thin black pen is chosen and the position is centered, heading to the top.
If the *background-color* is not given, it is imported from AKFAvatar.

Retuns the graphic, the width and the height (ie. three values).

Use it like this:

```
local gr, width, height = graphic.new()
```

graphic.fullsize()

Returns the width and height for a graphic so that it fills the whole window.

graphic.set_resize_key(*key***)**

Sets a key-code (as number) which should be returned when the window gets resized. Returns the previous key-code.

Hint: Use something between 0xE000 and 0xE9FF to avoid conflicting with real key-codes.

graphic.set_pointer_buttons_key(*key***)**

Sets a key-code (as number) which should be returned when any mouse button is pressed. Returns the previous key-code.

Hint: Use something between 0xE000 and 0xE9FF to avoid conflicting with real key-codes.

graphic.set_pointer_motion_key(*key***)**

Sets a key-code (as number) which should be returned when the mouse is moved. Returns the previous key-code.

graphic.get_pointer_position()

Returns the x and y coordinates of the current mouse position, relative to the last shown graphic.

*gr***:show()**

Shows the graphic as image after drawing.

Hint: You can show intermediate images, continue drawing, wait a while and then show the next step and so on. This makes a nice animation. But don't make the steps too small, or it will be painful on slow devices.

*gr***:size()**

Returns the width and the height of the graphic *gr*.

*gr***:width()**

Returns the width of the graphic *gr*.

gr:**height()**
: Returns the height of the graphic *gr*.

gr:**color(***color***)**
: Sets the drawing *color*.
The color can be either a defined color name, or a hexadecimal representation, like for example *0xFFFF00* or *"#FFFF00"*.

gr:**rgb(***red, green, blue***)**
: Sets the drawing color from RGB values.
The values must be in the range from 0 to 255 inclusive.
This method is faster than using **gr:color()**.

gr:**eraser()**
: Sets the drawing color to the background color.

gr:**thickness(***value***)**
: Sets the thickness of the pen.
The *value* 1 is the thinnest.

gr:**clear(***[color]***)**
: Clears the graphic.
If a *color* is given, then that becomes the new background color.
The pen position is not changed.

gr:**border3d(***[pressed]***)**
: Draws 3d border around the graphic. The color is based on the background color. The setting for thickness is ignored, it is always the 3 pixels.

if *pressed* is **true** the border is presented as being pressed.

The pen position is not changed.

gr:**putpixel(***[x, y]***)**
: Puts a pixel at the given coordinates or at the current pen position.
The pen position is not changed.

gr:**getpixel(***[x, y]***)**
: Gets the pixel color at the given coordinates or at the current pen position.
The color is returned as string in the hexadecimal RGB notation.
On error it returns *nil* and an error message.
The pen position is not changed.

gr:**getpixelrgb(***[x, y]***)**
: Gets the pixel color at the given coordinates or at the current pen position.
Returns three integers for red, green and blue, in the range of 0-255.
On error it returns *nil* and an error message.
The pen position is not changed.

gr:**putdot(***[x, y]***)**
: Puts a dot at the given coordinates or at the current pen position.
If the pen is thin, it's the same as **gr:putpixel**.
The pen position is not changed.

gr:**pen_position()**
: Returns the x and y position of the pen (ie. returns two values).

gr:**center()**
gr:**home()**
: Sets the pen to the center of the graphic, heading to the top.

gr:**moveto(***x, y***)**
Move the pen to *x, y* without drawing.

gr:**moverel(***x, y***)**
Move the pen without drawing relative to its current position.
A positive *x* value moves it to the right,
a negative *x* value moves it to the left.
A positive *y* value moves it down,
a negative *y* value moves it up.

gr:**lineto(***x, y***)**
Draws a line from the current pen position to these absolute coordinates.
The pen is moved to the new coordinates.

gr:**linerel(***x, y***)**
Draws a line relative to its current pen position.
A positive *x* value draws to the right,
a negative *x* value draws to the left.
A positive *y* value draws down,
a negative *y* value draws up.
The pen is moved to the new position.

gr:**line(***x1, y1, x2, y2***)**
Draws a line from *x1, y1* to *x2, y2*.
The pen is set to *x2, y2*.

gr:**bar(***x1, y1, x2, y2***)**
Draws a solid bar with *x1, y1* as the upper left corner and *x2, y2* as the lower right corner.
The pen position is not changed.

gr:**rectangle(***x1, y1, x2, y2***)**
Draws rectangle with *x1, y1* as the upper left corner and *x2, y2* as the lower right corner.
The pen position is not changed.

gr:**arc(***radius [, angle1] [, angle2]***)**
gr:**circle(***radius [, angle1] [, angle2]***)**
Draws a circle or an arc with the given *radius*.
The pen position is the center.

You can draw a part of the circle (an arc) by giving one or two angles in degree. If two angles are given, then it draws clockwise from the first angle to the second. If just one angle is given, it uses the heading as start-angle (see below under **turtle graphics**).

gr:**disc(***radius [, x, y]***)**
Draws a disc, ie. a filled circle with the given *radius* with the given coordinates as center. If no coordinates are given, the current pen position is used as center.
The pen position is not changed.

gr:**text(***text [, x, y]***)**
Prints a text aligned to the given position or the pen position.

By default the text is centered to the position. But you can change this with **gr:textalign()**.

The encoding is used from the AKFAvatar settings. However no other of those settings are taken into account. The color is the drawing color for the graphic. There is currently no easy way to make boldface, underlined or inverted text.

You can use all printable characters, but control characters are not supported, not even a newline.
The pen position is not changed.

gr:**textalign(***[horizontal] [, vertical]***)**
Sets the textalignment for **gr:text()**.

The horizontal alignment can be one of "left", "center" or "right". The default is "center".

The vertical alignment can be one of "top", "center" or "bottom". The default is "center".

The alignment means, where the given point is, eg. when you tell it to be "left"-aligned, the fixed point is on the left, but the text runs to the right.

graphic.font_size()
gr:**font_size()**
Returns the width, height and the baseline of the font, ie. one character. It is a fixed-width font, each character has the same width.

gr:**put(***graphic [, x, y]***)**
Puts a graphic onto graphic *gr* at the given position (upper-left corner). If no position is given it puts it at the upper-left corner. The previous content is overwritten (no transparency supported).

Copying a graphic with the same size and no position is highly efficient. The same is true for a graphic with the same width and *x* set to 1.

gr:**put_transparency(***graphic [, x, y]***)**
Puts a graphic onto graphic *gr* at the given position (upper-left corner).
If no position is given it puts it at the upper-left corner.
Pixels with the background color are not copied, they are transparent.
This is much slower than **gr:put()**.

gr:**put_file(** *filename [, x, y]***)**
Puts a graphic from a file onto graphic *gr* at the given position (upper-left corner).
If no position is given it puts it at the upper-left corner.

gr:**put_image(***data [, x, y]***)**
Puts a graphic from *data* onto graphic *gr* at the given position (upper-left corner). The *data* can be a string with image-data or a table with strings from XPM data.
If no position is given it puts it at the upper-left corner.

gr:**get(***x1, y1, x2, y2***)**
Returns an area of the graphic *gr* as a new graphic.
Most settings are copied, except the size and the pen settings.
The pen is put in the center, heading to the top.
All values must be in range.

gr:**duplicate()**
Returns an exact duplicate (a copy) of the graphic *gr*.
The graphic-specific settings are copied, too.
This is faster than using **gr:get()**.

You can use this for example to create a fixed background and then make a duplicate and draw the foreground on it. Then you can **gr:put()** the background graphic back to the duplicate and draw another foreground.

gr:**shift_vertically(***lines***)**
Shifts the graphic vertically.
A positive value for *lines* shifts it down.
A negative value for *lines* shifts it up.
The pen gets also moved.

gr:**shift_horizontally(***columns***)**
Shifts the graphic horizontally.
A positive value for *columns* shifts it right.
A negative value for *columns* shifts it left.
The pen gets also moved.

gr:**export_ppm(** *filename***)**
Exports the graphic as Portable Pixmap (PPM) file.

The PPM format is simple to implement, but not very efficient. You might want to use the "netpbm" tools or "ImageMagick" to convert it to another format.

The following example shows how to do this:

```
function export(graphic, name)
  graphic:export_ppm(name..".ppm")
  if os.execute("pnmtopng "..name..".ppm > "..name..".png")
     or os.execute("convert "..name..".ppm "..name..".png") then
    os.remove(name..".ppm")
  end
end
```

First it exports the graphic in the PPM format. Then it tries to convert it to the PNG format. If that succeeds, it deletes the PPM file. If the user doesn't have "netpbm" or "ImageMagick" installed, he still ends up with the PPM file.

Turtle graphics

To understand turtle graphics think of a turtle that carries a pen. You can control the turtle by telling her in which direction to turn and how far to move.

gr:**heading(***heading***)**
Sets the heading for the turtle. The value must be given in degree and the turtle turns clockwise. The value 0 means, it's heading to the top, 90 means it heads to the right.

gr:**get_heading()**
Returns the heading of the turtle (see **gr:heading()**).

gr:**right(***angle***)**
Turn the turtle clockwise by the specified *angle* in degree.

gr:**left(***angle***)**
Turn the turtle counterclockwise by the specified *angle* in degree.

gr:**draw(***steps***)**
Draw a line in the direction the turtle is heading.

gr:**move(***steps***)**
Move the turtle in the direction it is heading without drawing.

gr:**home()**
Sets the pen to the center of the graphic, heading to the top.

SEE ALSO

lua-akfavatar(1) **lua**(1) **lua-akfavatar-ref**(3) **akfavatar-term**(3) **akfavatar.utf8**(3)