ezxdisp API Manual

Last updated on June 20, 2007.

This document was written and is maintained by Morihiko Tamai <morihi-t AT is.naist.jp>. Please contact him, if there are any questions/comments/suggestions regarding this document.

Introduction

The ezxdisp library mainly consists of functions for creating/deleting a window, drawing 2D/3D objects on the window, and handling keyboard/mouse events. This document describes how to use these functions.

Header File and Compilation

All ezxdisp functions are declared in 'ezxdisp.h'. Your program should include the header.

If 'ezxdisp.h' is in './include' directory and 'libezx.a' is in './lib' directory, a typical compile command would look like this:

On a x11 platform

```
gcc -I./include foobar.c -o foobar -L./lib -lezx -lX11 -lm
```

On a win32 platform (with gcc under MinGW)

```
gcc -I./include foobar.c -o foobar -L./lib -lmingw32 -lezx -mwindows
```

Window Creation/Deletion

To create a new window, use ezx_init(). This function takes the width, height, and title of the window as its arguments, and returns a pointer to ezx_t structure. This pointer is used in the first argument of the other ezxdisp functions. If ezx_init() fails, NULL is returned.

To destroy the window, use ezx_quit(). This function releases all of the resources (e.g., allocated memory) associated with the window and closes the window.

Example:

```
#include <stdio.h>
#include "ezxdisp.h"

int main(int argc, char *argv[])
{
  ezx_t *e;

  e = ezx_init(200, 200, "no title");

  getchar();

  ezx_quit(e);

  return 0;
}
```

Drawing 2D Objects

ezxdisp offers a set of functions to draw 2D graphics objects:

- **Point:** ezx_point_2d()
- **Line:** ezx_line_2d() and ezx_lines_2d()
- **Polygon:** ezx_poly_2d()
- **String:** ezx_str_2d()
- **Rectangle:** ezx_rect_2d() and ezx_fillrect_2d()
- **Circle:** ezx_circle_2d() and ezx_fillcircle_2d()
- **Arc:** ezx_arc_2d() and ezx_fillarc_2d()

These functions take a pointer to ezx_color_t structure as one of its arguments. It is used to specify the color of the 2D object. The definition of the ezx_color_t structure is as follows:

```
typedef struct {
  double r, g, b;   // red, green, and blue values of the color, they are in the range of 0.0-1.0
} ezx_color_t;
```

Also, the predefined constants are available: ezx_black, ezx_white, ezx_grey25, ezx_grey50, ezx_grey75, ezx_blue, ezx_red, ezx_green, ezx_yellow, ezx_purple, ezx_pink, ezx_cyan, ezx_brown, and ezx_orange.

When one of the above function is called, a new *2D object data* (type of 2D object and its parameter values) is added to the list of 2D object data managed by the window (without displaying the 2D object). To display all of the 2D objects added to the window, call ezx_redraw().

Example:

```
#include <stdio.h>
#include "ezxdisp.h"
```

```
int main(int argc, char *argv[])
{
  ezx_t *e;

  e = ezx_init(200, 200, "no title");

  ezx_line_2d(e, 0, 0, 200, 200, &ezx_red, 1);
  ezx_circle_2d(e, 50, 100, 20, &ezx_black, 1);
  ezx_fillrect_2d(e, 120, 40, 170, 160, &ezx_blue);

  ezx_redraw(e);

  getchar();

  ezx_quit(e);

  return 0;
}
```

To remove all of the 2D object data added to the window, call ezx_wipe().

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include "ezxdisp.h"

int main(int argc, char *argv[])
{
  ezx_t *e;
  int width=200, height=200;

  e = ezx_init(width, height, "no title");

  for (;;) {
    ezx_wipe(e);
    ezx_fillcircle_2d(e, rand()%width, rand()%height, 10, &ezx_blue);
    ezx_redraw(e);
  }
}
```

Layers

Each window has multiple layers, and each layer manages its own list of 2D object data (Fig. 1).

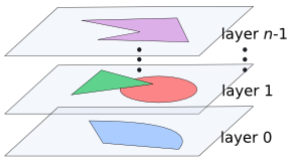

Fig. 1 2D object data managed by layers.

When one of the drawing function (e.g., ezx_line_2d()) called, its 2D object data is added to the list of the 2D object data managed by the *current layer*. To change the current layer, use ezx_select_layer(), which takes the layer number you want to select. Note that when ezx_init() function returns, the layer 0 is implicitly set as the current layer. There are 8 layers available (this number is defined in ezxdisp.h as EZX_NLAYERS).

We can use ezx_wipe_layer() to remove all of the 2D object data managed by a specific layer.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include "ezxdisp.h"

int main(int argc, char *argv[])
{
  ezx_t *e;
  int width=200, height=200;

  e = ezx_init(width, height, "no title");

  ezx_line_2d(e, 0, 0, width, height, &ezx_grey50, 1);
  ezx_line_2d(e, 0, height, width, 0, &ezx_grey50, 1);

  ezx_select_layer(e, 1);

  for (;;) {
    ezx_wipe_layer(e, 1);
    ezx_fillcircle_2d(e, rand()%width, rand()%height, 10, &ezx_blue);
    ezx_redraw(e);
  }
}
```

Event Handling

Each window maintains an event queue which is composed of a series of *event data* (event type and specific information based on the type) which are generated by events occurred in the window. The supported event types are mouse button press/release event, keyboard press/release event, mouse motion event, and window close event. Note that mouse motion events occur if and only if any mouse button is pressed (i.e., while dragging the mouse).

To catch mouse button press events, use ezx_pushbutton(). This function returns the button code pressed and sets the x-y coordinates of the mouse pointer to the variables passed as arguments. The button code is one of EZX_BUTTON_LEFT, EZX_BUTTON_MIDDLE, EZX_BUTTON_RIGHT, EZX_BUTTON_WHEELUP, or EZX_BUTTON_WHEELDOWN. If there is no event data generated by button press event in the event queue, this function blocks until the next button press event occurs.

Example:

```
#include <stdio.h>
#include "ezxdisp.h"
```

```
int main(int argc, char *argv[])
{
  ezx_t *e;
  int b, x, y;

  e = ezx_init(200, 200, "no title");

  for (;;) {
    b = ezx_pushbutton(e, &x, &y);
    if (b == EZX_BUTTON_LEFT) break;
  }

  printf("The left button was pressed at the location (%d,%d).\n", x, y);

  ezx_quit(e);

  return 0;
}
```

More general event handling can be achieved by using ezx_next_event(). This function removes the next event data from the event queue and stores it to the variable of ezx_event_t union which is passed as one of its arguments. If there is no event data in the event queue, this function blocks until the next event occurs. The event type is identified by the *type* field of ezx_event_t union. To access the event specific information, the following fields are available depending on the event type.

- Mouse button press/release event
  - **type:** EZX_BUTTON_PRESS or EZX_BUTTON_RELEASE
  - **button.b:** the mouse button code
  - **button.x, button.y:** the x-y coordinates of the mouse pointer
  - **button.state:** the key or button mask
- Keyboard press/release event
  - **type:** EZX_KEY_PRESS or EZX_KEY_RELEASE
  - **key.k:** the key code
  - **key.x, key.y:** the x-y coordinates of the mouse pointer
  - **key.state:** the key or button mask
- Mouse motion event
  - **type:** EZX_MOTION_NOTIFY
  - **motion.x, motion.y:** the x-y coordinates of the mouse pointer
  - **motion.state:** the key or button mask
- Window close event
  - **type:** EZX_CLOSE

The mouse button code takes the same values as the return value of ezx_pushbutton() (described above). The key code is an ascii code or one of EZX_KEY_HOME, EZX_KEY_LEFT, EZX_KEY_UP, EZX_KEY_RIGHT, or EZX_KEY_DOWN. The button.state, key.state, and motion.state fields indicate the state (pressed or not) of the shift and control keys and the mouse buttons when the event occurred, which are the bitwise inclusive OR of one or more of the key or button masks: EZX_SHIFT_MASK, EZX_CONTROL_MASK, EZX_BUTTON_LMASK, EZX_BUTTON_MMASK, and EZX_BUTTON_RMASK.

Example:

```
#include <stdio.h>
#include "ezxdisp.h"

int main(int argc, char *argv[])
{
  ezx_t *e;
  ezx_event_t event;

  e = ezx_init(200, 200, "no title");

  for (;;) {
    ezx_next_event(e, &event);
    switch (event.type) {
    case EZX_BUTTON_PRESS:
    case EZX_BUTTON_RELEASE:
      printf("button event (b=%d)\n", event.button.b);
      break;
    case EZX_KEY_PRESS:
    case EZX_KEY_RELEASE:
      printf("key event (k=%d)\n", event.key.k);
      break;
    case EZX_MOTION_NOTIFY:
      printf("mouse motion event (x=%d, y=%d)\n", event.motion.x, event.motion.y);
      break;
    case EZX_CLOSE:
      ezx_quit(e);
      return 0;
    }
  }
}
```

Misc. Functions

- To set the background color of the window, use ezx_set_background(), which takes a pointer to ezx_color_t structure.
- To set the title of the window, use ezx_window_name().
- To get the current x-y coordinates of the mouse pointer, use ezx_sensebutton(). This function stores the x-y coordinates of the mouse pointer to the variables passed as arguments. The return value of the function is the key or button mask, which takes the same value as the button.state field of the ezx_event_t union returned by ezx_next_event() (described above).
- To check if the close button of the window was pressed, use ezx_isclosed(), which returns 1 if it was pressed and 0 otherwise.

Drawing 3D Objects

Sorry, not documented yet...

API Reference for Basic Operations

EZX_NLAYER - the number of layers

```
#define EZX_NLAYER 8
```

ezx_t - represents a window managed by ezxdisp library

```
typedef struct ezx_s ezx_t;
```

ezx_t is an opaque data type.

ezx_color_t - represents a color with red, green, and blue components

```
typedef struct {
  double r, g, b;   // red, green, and blue values of the color, they are in the range of 0.0-1.0
} ezx_color_t;
```

The following predefined colors are available: ezx_black, ezx_white, ezx_grey25, ezx_grey50, ezx_grey75, ezx_blue, ezx_red, ezx_green, ezx_yellow, ezx_purple, ezx_pink, ezx_cyan, ezx_brown, and ezx_orange.

ezx_event_t - stores detailed information about an event

```
typedef union {
  enum ezx_event_type {
    EZX_BUTTON_PRESS,
    EZX_BUTTON_RELEASE,
    EZX_KEY_PRESS,
    EZX_KEY_RELEASE,
    EZX_MOTION_NOTIFY,
    EZX_CLOSE
  } type;

  struct ezx_button_event {
    enum ezx_event_type type; /* EZX_BUTTON_PRESS or EZX_BUTTON_RELEASE */
    unsigned int b;
    int x, y;
    unsigned int state;
  } button;

  struct ezx_key_event {
    enum ezx_event_type type; /* EZX_KEY_PRESS or EZX_KEY_RELEASE */
    unsigned int k;
    int x, y;
    unsigned int state;
  } key;

  struct ezx_motion_event {
    enum ezx_event_type type; /* EZX_MOTION_NOTIFY */
    int x, y;
    unsigned int state;
  } motion;
} ezx_event_t;
```

- **button.b:** is one of EZX_BUTTON_LEFT, EZX_BUTTON_MIDDLE, EZX_BUTTON_RIGHT, EZX_BUTTON_WHEELUP, or EZX_BUTTON_WHEELDOWN.
- **key.k:** is an ascii code or one of EZX_KEY_HOME, EZX_KEY_LEFT, EZX_KEY_UP, EZX_KEY_RIGHT, or EZX_KEY_DOWN.
- **button.state, key.state, motion.state:** is a bitwise inclusive OR of any combination of EZX_SHIFT_MASK, EZX_CONTROL_MASK, EZX_BUTTON_LMASK, EZX_BUTTON_MMASK, and EZX_BUTTON_RMASK.

ezx_init() - creates a new window

```
ezx_t *ezx_init(int size_x, int size_y, char *window_name);
```

- **size_x, size_y:** specify the width and height of the window.
- **window_name:** specifies the title of the window.
- **returns:** a pointer to the newly-allocated ezx_t structure, or NULL if the function fails.

ezx_quit() - destroys the specified window

```
void ezx_quit(ezx_t *e);
```

ezx_set_background() - sets the background color

```
void ezx_set_background(ezx_t *e, const ezx_color_t *col);
```

- **col:** specifies the background color.

ezx_redraw() - redraws all of the graphics in the window

```
void ezx_redraw(ezx_t *e);
```

ezx_wipe() - removes all of the graphics in the window

```
void ezx_wipe(ezx_t *e);
```

ezx_select_layer() - sets the current layer

```
void ezx_select_layer(ezx_t *e, int lay);
```

- **lay:** specifies the layer number in the range [0..7].

ezx_wipe_layer() - removes all of the graphics on a layer

```
void ezx_wipe_layer(ezx_t *e, int lay);
```

- **lay:** specifies the layer number in the range [0..7].

ezx_window_name() - sets the title of the window

```
void ezx_window_name(ezx_t *e, char *window_name);
```

- **window_name:** specifies the title of the window.

ezx_isclosed() - tests if the close button of the window was pressed

```
int ezx_isclosed(ezx_t *e);
```

ezx_sensebutton() - gets the current coordinates of the mouse pointer

```
int ezx_sensebutton(ezx_t *e, int *x, int *y);
```

- **x, y:** return the current x-y coordinates of the mouse pointer.
- **returns:** the state of the mouse buttons and shift/control keys, which takes the same value as the button.state field of ezx_event_t.

ezx_pushbutton() - waits for a mouse button to be pressed

```
int ezx_pushbutton(ezx_t *e, int *x, int *y);
```

- **x, y:** return the x-y coordinates at which the mouse button was pressed.
- **returns:** which mouse button was pressed.

ezx_next_event() - waits for an event occurs

```
void ezx_next_event(ezx_t *ezx, ezx_event_t *event);
```

- **event:** returns the data of the next event.

API Reference for 2D Drawing Functions

ezx_point2d_t - represents x-y coordinates of a point

```
typedef struct {
  int x, y;
} ezx_point2d_t;
```

ezx_point_2d() - draws a point

```
void ezx_point_2d(ezx_t *e, int x, int y, const ezx_color_t *col);
```

- **x, y:** specify the x-y coordinates of the point to be drawn.

ezx_line_2d() - draws a line

```
void ezx_line_2d(ezx_t *e, int x0, int y0, int x1, int y1, const ezx_color_t *col, int width);
```

- **x0, y0, x1, y1:** specify the points (x0, y0) and (x1, y1) to be connected.
- **width:** specifies the line width.

ezx_lines_2d() - draws a series of lines connecting the given points

```
void ezx_lines_2d(ezx_t *e, ezx_point2d_t *points, int npoints, const ezx_color_t *col, int width);
```

- **points:** specifies an array of points.
- **npoints:** specifies the number of points.
- **width:** specifies the line width.

ezx_poly_2d() - draws a polygon

```
void ezx_poly_2d(ezx_t *e,ezx_point2d_t *points, int npoints, const ezx_color_t *col);
```

- **points:** specifies an array of points.
- **npoints:** specifies the number of points.

ezx_str_2d() - draws a string

```
void ezx_str_2d(ezx_t *e, int x, int y, char *str, const ezx_color_t *col);
```

- **x, y:** specify the x-y coordinates of the origin of the string.
- **str:** specifies the string drawn.

ezx_rect_2d(), ezx_fillrect_2d() - draws a rectangle (outline or filled)

```
void ezx_rect_2d(ezx_t *e, int x0, int y0, int x1, int y1, const ezx_color_t *col, int width);
void ezx_fillrect_2d(ezx_t *e, int x0, int y0, int x1, int y1, const ezx_color_t *col);
```

- **x0, y0:** specify the x-y coordinates of the upper-left corner of the rectangle.
- **x1, y1:** specify the x-y coordinates of the lower-right corner of the rectangle.

- **width:** specifies the line width.

ezx_circle_2d(), ezx_fillcircle_2d() - draws a circle (outline or filled)

```
void ezx_circle_2d(ezx_t *e, int x, int y, int r, const ezx_color_t *col, int width);
void ezx_fillcircle_2d(ezx_t *e, int x, int y, int r, const ezx_color_t *col);
```

- **x, y:** specify the x-y coordinates of the center of the circle.
- **r:** specifies the radius of the circle.
- **width:** specifies the line width.

ezx_arc_2d(), ezx_fillarc_2d() - draws an arc (outline or filled)

```
void ezx_arc_2d(ezx_t *e, int x, int y, int w, int h, double angle1, double angle2, const ezx_color_t *col, int width);
void ezx_fillarc_2d(ezx_t *e, int x, int y, int w, int h, double angle1, double angle2, const ezx_color_t *col);
```

- **x, y:** specify the x-y coordinates of the center of the arc.
- **w, h:** specify the width and height of the arc.
- **angle1:** specifies the beginning angle of the arc measured in degrees relative to the three o'clock position.
- **angle2:** specifies the angular extent of the arc measured in degrees from the beginning angle, in the counterclockwise direction.
- **width:** specifies the line width.

Home