

# EZ Draw Users Manual

## version 1.0

Donald H. House  
October 3, 2019  
Clemson University

EZ Draw is an interactive graphics Application Program Interface (API) that is designed to be easy to use for programming students in their first courses, enabling them to complete significant interactive graphics programming exercises as they develop their programming skills. EZ Draw is built on top of the SDL game programming API. All of the underlying graphics are done via calls to SDL routines, which in turn utilize the OpenGL API.

On the next page is a first example program written using the EZ Draw API. The main routine does only three things. It first creates an 800x600 pixel window on the screen with a title bar saying: "Hello World". It then enters a loop that calls the `updateDisplay` routine every 1/30 of a second. This loop exits when either the user presses the ESC key on the keyboard or clicks the kill box on the window. The main routine finally finishes by calling the `EZ_Quit()` quit routine, which shuts down the EZ Draw and its display window.

The while loop in the main routine calls the `EZ_HandleEvents()` routine over and over again, until it returns true (i.e. a nonzero value). What `EZ_HandleEvents()` does is to query the operating system for external events, like key presses on the keyboard, mouse button clicks, or mouse motion. It also responds to a timer event that is internally generated by the EZ Draw system every 1/30 of a second. `EZ_HandleEvents()` takes four parameters. The first is the name of a routine to be called whenever the 1/30 of a second timer event occurs. The second, third, and fourth parameters are the names of routines to be called respectively for keyboard events, mouse button events, and mouse motion events. In this example, the last three parameters are NULL, meaning that no routine is to be called for any of the external events. Therefore, all that this loop will do is to call the `updateDisplay()` routine every 1/30 of a second until either the user presses ESC or clicks the window kill box.

The job of the `updateDisplay()` routine is to make a new drawing and display it on the screen in the EZ Draw window. The steps that this routine takes are to first clear the drawing to a background color, then to draw a filled in rectangle in another color, and finally to display this new drawing in the window. It may seem strange that this is being done every 1/30 of a second, since the drawing never changes. However, EZ Draw is designed to allow the construction of interactive animated displays, so the fact that the drawing routine is called repetitively is very useful in more advanced programs.

Notice that this example program makes extensive use of `#define` statements so that the program code itself has no "magic" numbers in it. All of these numbers are specified at the start of the program, so that changing them automatically changes the values throughout.

A word needs to also be said about colors. All colors are specified in EZ Draw using Red, Green, Blue (RGB) color specifications. These three numbers are specified as unsigned chars so they are on the scale 0 to 255. For example if no red is wanted the red value would be 0, if full intensity red is wanted the red value would be 255, and if 1/2 intensity red is wanted the value would be 127. So, the rectangle color (200, 100, 50) is saying that the resulting color should be 78% full red, 39% full green, and 20% full blue.



On the other hand, the background color (70, 100, 140) is 27% full red, 39% full green, and 55% full blue.



```

//
// helloworld.c
// Basic EZ Draw program to draw a rectangle on the screen
//
#include "ezdraw.h"    // the ezdraw interface must be included
#include <stdio.h>

// dimensions of the display window
#define WINDOWWIDTH    800
#define WINDOWHEIGHT   600

// coordinates of the rectangle are relative to the window
#define RECTWIDTH      (WINDOWWIDTH / 2)
#define RECTHEIGHT     (WINDOWHEIGHT / 2)
#define RECTX0         ((WINDOWWIDTH - RECTWIDTH) / 2)
#define RECTY0         ((WINDOWHEIGHT - RECTHEIGHT) / 2)

// RGB colors for the rectangle and screen background
// note: R, G, and B are on a scale from 0 to 255
#define RECTCOLOR       200, 100, 50
#define BACKCOLOR       70, 100, 140

typedef enum _bool{FALSE, TRUE} bool;    // handy boolean constants

// display routine, called from EZ_HandleEvents every 1/30 of a second
void updateDisplay(){
    EZ_SetBackColor(BACKCOLOR);    // specify drawing background color
    EZ_ClearDrawing();             // clear drawing to background color

    // draw the rectangle
    EZ_SetColor(RECTCOLOR);        // first specify drawing color
    EZ_FillRect(RECTX0, RECTY0, RECTWIDTH, RECTHEIGHT);

    EZ_DisplayDrawing();           // send the new drawing to the screen
}

int main() {
    bool quit = FALSE;

    EZ_Init(WINDOWWIDTH, WINDOWHEIGHT, "Hello World");

    // loop that will keep going until EZ_HandleEvents() sees that
    // the ESC key was pressed or the window kill box was clicked
    while(!quit){
        quit = EZ_HandleEvents(updateDisplay, NULL, NULL, NULL);
    }

    EZ_Quit();
}

```

If the main event loop were updated as shown below and the following routines were added to the `helloworld.c` program, the program would respond to all events, not just the timer events. In this example, the event handling routines simply print out their arguments. In a useful program, they would do whatever was necessary to respond to each type of event.

```
// keyboard routine, print out the key that was pressed
void handleKey(unsigned char key){
    printf("k %c\n", key);
}

// mouse button routine, print button status and mouse position
void handleButton(int updown, int mousex, int mousey){
    printf("b %d (%d, %d)\n", updown, mousex, mousey);
}

// mouse motion routine, print mouse position, and change of position
void handleMouseMotion(int mousex, int mousey,
                       int dmousex, int dmousey){
    printf("m (%d, %d) (%d, %d)\n", mousex, mousey, dmousex, dmousey);
}

. . .
// event loop that calls all four event processing routines
while(!quit){
    quit = EZ_HandleEvents(updateDisplay, handleKey,
                           handleButton, handleMouseMotion);
}
. . .
```

## Compiling EZ Draw Programs

A program using EZ Draw can be conveniently compiled using the following Makefile. Note, that the Makefile should be in the same directory as the program being compiled. The `ezdraw.h` include file should be in the directory specified by `IDIR` and the `libezdraw.a` library should be in the directory specified by `LDIR`. Change the `NAME =` line in the Makefile to match the name of your `.c` file. Change the `CC =` line if you want to use a compiler other than `gcc`, for example to compile under the C++ compiler change `gcc` to `g++`.

```
CC = gcc
NAME = helloworld

# path to directory holding the ezdraw.h file
IDIR = ~/ezdraw/
# path to the directory holding the libezdraw.a files
LDIR = ~/ezdraw/

# loader flags showing where to find the SDL libraries
LDFLAGS = `sdl2-config --libs`

# non SDL libraries to load: ezdraw and math library
LIBS = -lezdraw -lm

# compiler flags for maximum warnings, debugger information,
# and where to find SDL include files
CFLAGS = `sdl2-config --cflags` -g -W -Wall -Wextra -pedantic -O0 -I
`sdl2-config --prefix`/include/

OBJS = $(NAME).o

$(NAME): $(NAME).o $(LDIR)libezdraw.a
    $(CC) -o $(NAME) $(NAME).o $(CFLAGS) -L $(LDIR) $(LIBS) $(LDFLAGS)

$(NAME).o: $(NAME).c $(IDIR)ezdraw.h
    $(CC) $(CFLAGS) -I $(IDIR) -c $< -o $@

clean:
    rm -f *.o
    rm -f *~
    rm -f $(NAME)
```

To compile your program, type: `make`

Or to clean up the directory and make sure that everything is recompiled from scratch the next time you make, type: `make clean`

## EZ Draw Application Program Interface

### C Struct Types

**EZ\_Point:** Gives the position of a point relative to the bottom lefthand corner of the window. Used by the `EZ_DrawLineStrip()`, `EZ_DrawLineLoop()`, `EZ_FillTriangle()`, and `EZ_OutlineTriangle()` routines.

int	x	horizontal coordinate
int	y	vertical coordinate

**EZ\_Rect:** Gives the location and size of a rectangle. The position is relative to the bottom lefthand corner of the window. Used by the `EZ_FillRects()`, `EZ_OutlineRects()`, and `EZ_DrawTexture()` routines.

int	x	horizontal coordinate of the bottom lefthand corner
int	y	vertical coordinate of the bottom lefthand corner
int	w	width in pixels
int	h	height in pixels

**EZ\_Color:** Gives the red, green, blue, and alpha components of a color. Since they are unsigned char's all values are on a scale of 0 to 255. For the RGB components 0 means none of the component, and 255 means the full brightness of the component. The alpha component signifies color opacity, where 0 means the color is fully transparent, and 255 means it is fully opaque.

unsigned char	r	red component
unsigned char	g	green component
unsigned char	b	blue component
unsigned char	a	alpha component

**EZ\_Image:** Gives all of the information supporting the handling of an image. Images always are stored with 32 bits per pixel, with each pixel consisting of 4 bytes giving the red, green, blue, and alpha values for the pixel.

int	w	width in pixels
int	h	height in pixels
void *	pixels	pointer to the block of memory holding the image pixels

## Color Setting Routines

All colors are specified as RGBA quadruples, with minimum value 0 and maximum value 255. r, g, and b specify the red, green and blue components, while a specifies the alpha component.

```
void EZ_SetColor(unsigned char r, unsigned char g, unsigned char b);
```

Set the red, green, and blue components of the fully opaque color (a = 255) that will be used for drawing. Note: this color stays current until a subsequent call to EZ\_SetColor() or EZ\_SetColorRGBA().

```
void EZ_SetColorRGBA(unsigned char r, unsigned char g,  
                     unsigned char b, unsigned char a);
```

Set the red, green, blue, and alpha components of the color that will be used for drawing. Note: this color stays current until a subsequent call to EZ\_SetColor() or EZ\_SetColorRGBA().

```
void EZ_SetBackColor(unsigned char r, unsigned char g,  
                     unsigned char b);
```

Set the red, green, and blue components of the fully opaque color (a = 255) that will be used to fill the background of the drawing when EZ\_ClearDrawing() is called, or the background of an image when EZ\_CreateBlankImage() is called. Note: this color stays current until a subsequent call to EZ\_SetBackColor() or EZ\_SetBackColorRGBA().

```
void EZ_SetBackColorRGBA(unsigned char r, unsigned char g,  
                        unsigned char b, unsigned char a);
```

Set the red, green, blue, and alpha components of the color that will be used to fill the background of the drawing when EZ\_ClearDrawing() is called, or the background of an image when EZ\_CreateBlankImage() is called. Note: this background color stays current until a subsequent call to EZ\_SetBackColor() or EZ\_SetBackColorRGBA().

## Drawing Routines

### *Drawing management*

**void** EZ\_ClearDrawing();

Clear the drawing to the most recently set background color. See EZ\_SetBackColor()

**void** EZ\_DisplayDrawing();

Display the current drawing in the display window on the screen.

### *Drawing objects*

All of the following routines draw using the most recently set drawing color. See EZ\_SetColor() and EZ\_SetColorRGBA. All (x, y) positions are pixel positions relative to the lower lefthand corner of the window, with x indicating the horizontal pixel coordinate, and y the vertical pixel coordinate.

**void** EZ\_DrawPoint(**int** x, **int** y);

Draw a single pixel at position (x, y).

**void** EZ\_DrawLine(**int** x0, **int** y0, **int** x1, **int** y1);

Draw a line from (x0, y0) to (x1, y1).

**void** EZ\_DrawLineStrip(EZ\_Point \*points, **int** npoints);

Draw a connected set of lines using the point positions in the array points. npoints-1 lines will be drawn between consecutive npoints points in the array.

**void** EZ\_DrawLineLoop(EZ\_Point \*points, **int** npoints);

Draw a closed loop of connected lines using the point positions in the array points. npoints lines will be drawn between consecutive npoints points in the array, with the final line connecting the last point in the array to the first.

**void** EZ\_FillRect(**int** xll, **int** yll, **int** w, **int** h);

Draw a filled in rectangle whose lower lefthand corner is (xll, yll) and whose width is w and height is h.

**void** EZ\_FillRects(EZ\_Rect \*rects, **int** nrects);

Draw a set of filled in rectangles using the rectangle specifications in the array rects. nrects rectangles will be drawn. See the EZ\_Rect definition above.

```
void EZ_OutlineRect(int xll, int yll, int w, int h);
```

Draw the outline of a rectangle whose lower lefthand corner is (xll, yll), whose width is w and height is h.

```
void EZ_OutlineRects(EZ_Rect *rects, int nrects);
```

Draw the outlines of a set of rectangles using the rectangle specifications in the array rects. nrects rectangles will be drawn. See the EZ\_Rect definition above.

```
void EZ_FillTriangle(EZ_Point p0, EZ_Point p1, EZ_Point p2);
```

Draw a filled in triangle whose three vertices are specified by the points p0, p1, and p2. See the EZ\_Point definition above.

```
void EZ_OutlineTriangle(EZ_Point p0, EZ_Point p1, EZ_Point p2);
```

Draw the outline of a triangle whose three vertices are specified by the points p0, p1, and p2. See the EZ\_Point definition above.

```
void EZ_FillCircle(int cx, int cy, int radius);
```

Draw a filled in circle whose center is at position (cx, cy) and whose radius is given by radius.

```
void EZ_OutlineCircle(int cx, int cy, int radius);
```

Draw the outline of a circle whose center is at position (cx, cy) and whose radius is given by radius.

## Image Management

```
EZ_Image *EZ_CreateBlankImage(int width, int height);
```

Create a blank image of the given width and height. The background of the image will match the most recently set background color.

```
EZ_Image *EZ_LoadBMPImage(const char *bmpfilename);
```

Load an image from a BMP file (with filename suffix .bmp). The width and height of the image will be determined from the file when the image is read. These can be obtained from the EZ\_Image data structure.



## Texture Management

```
int EZ_CreateTexture(EZ_Image *image);
```

Create a texture from an image, and return its unique integer ID number. The texture will be loaded onto the graphics card, but will not be displayed until a call to `EZ_DrawTexture()` is made using its texture ID.

```
void EZ_DestroyTexture(int texture);
```

Destroy a texture by removing it from the graphics card. Its texture ID number will be made invalid after this call, and cannot be reused.

```
void EZ_DrawTexture(int texture, EZ_Rect *texture_rect, EZ_Rect  
                    *drawing_rect);
```

Draw the texture indicated by the given texture ID. Parameter `texture_rect` determines the rectangle on the texture that will be drawn. Parameter `drawing_rect` determines the rectangle in the window that the texture image will be drawn to. If necessary, the portion of the texture determined by `texture_rect` will be resized to fit `drawing_rect`. If `texture_rect` is NULL, the entire texture will be drawn. If `drawing_rect` is NULL, the texture will be drawn to the entire window.

## System Control Routines

### *System management*

```
int EZ_Init(int width, int height, char *title);
```

Initialize the EZ Draw system, and create a display window on the screen whose dimensions are given by `width` and `height`. The window's title is given by the string pointed to by `title`. Note, none of the other EZ Draw routines should be called until an `EZ_Init()` call has been made.

```
void EZ_Quit();
```

Terminate the EZ Draw system, by shutting down the display window and releasing any graphics resources being used by the program.

### *Event handling*

```
void EZ_WaitForQuit();
```

This is an event loop that runs indefinitely until the user either presses the ESC key on the keyboard or clicks the window kill box. This can be used in place of `EZ_HandleEvents()` when EZ Draw is being used to simply display a single drawing, and no interaction or animation is required. In this case, the program must draw the scene to be displayed before calling `EZ_WaitForQuit()`. Note, the routine contains an event loop within it, so it should not be placed inside a while loop.

```
int EZ_HandleEvents(void (*updateDisplay)(),  
                    void (*handleKey)(unsigned char),  
                    void (*handleButton)(int, int, int),  
                    void (*handleMouseMotion)(int, int, int, int));
```

This is the event handling routine that programs must use if they want to do any user interaction or animation. It would normally be placed within a while loop that loops until the return value of `EZ_HandleEvents()` is true (i.e. non zero). Each time it is called it checks for either the 1/30 of a second timer event, a keyboard key press, a left mouse button change (press or release), or mouse motion when the left mouse button is depressed. If the timer event has occurred, the display routine whose name has been passed in as the first parameter (`updateDisplay`) is called with no arguments. If a keyboard key has been pressed, the key processing routine whose name has been passed in as the second parameter (`handleKey`) is called with a single argument giving the ASCII code of the key that was pressed. If the left mouse button has been either pressed or released, the mouse button processing routine whose name has been passed in as the third parameter (`handleButton`) is called with three arguments: button status (1 = button pressed, 0 = button released), and the x and y coordinates of the mouse. If the left mouse button is pressed and the mouse has moved, the mouse motion processing routine whose name has been passed in as the fourth parameter (`handleMouseMotion`) is called with four arguments: the first two are the x and y coordinates of the mouse, and the second two are the change in mouse x and y coordinates since the last mouse motion or left button press event was processed.