BiPOM Graphics Library
User Manual

Date: 14 May 2007

Document Revision: 1.01

BiPOM Electronics

16301 Blue Ridge Road, Missouri City , Texas 77489
Telephone: 1-713-283-9970. Fax: Fax: 1-281-416-2806
E-mail: info@bipom.com
Web: www.bipom.com

TABLE OF CONTENTS

1. OVERVIEW

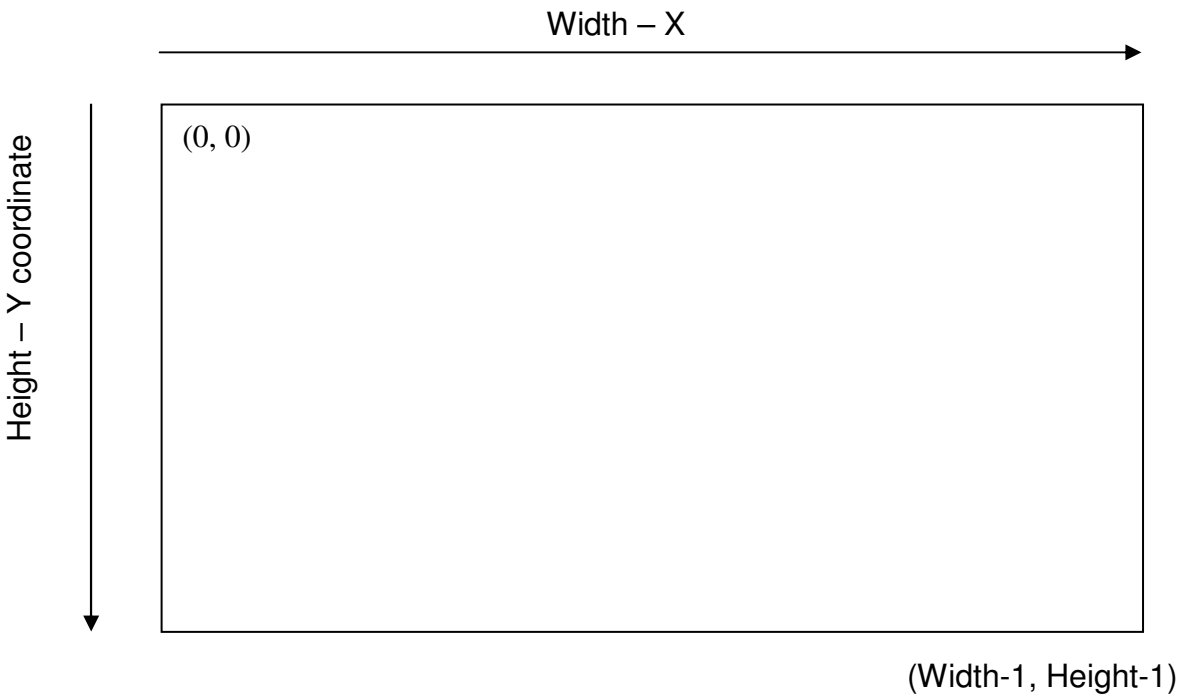
BiPOM Graphics Library (GL) is an open source graphics library written in GNUARM C Compiler for ARM microcontroller. This library allows drawing simple graphics primitives on a display which is connected to an ARM microcontroller board.

GL features:

- Driver interface that allows adding support for new displays without recompiling GL.
- Support up to 4 different displays simultaneously.
- Support follows primitives:
    - Pixels
    - Lines
    - Polylines
    - Rectangles
    - Circles
    - Ellipses
    - Arcs
    - 3 bitmap fonts
- Set line width
- Clear full screen or part of screen
- Set foreground and background colors

2. SCREEN STRUCTURE

GL screen has the following structure:



GL supports 2 functions to determine the size of the screen.

Function	Description
`int GetWidth();`	Returns width of current display in pixels.
`int GetHeight();`	Returns height of current display in pixels.

NOTE: *Depending on selected driver, the size of the screen may be different.*

3. CONSTANTS

GL and driver header file provide several constants to simplify using the library.

GL text constants:

Constant	Description
TEXT_SMALL	This constant is used to set small bitmap font as the current font.
TEXT_MEDIUM	This constant is used to set medium bitmap font as the current font.
TEXT_LARGE	This constant is used to set large bitmap font as the current font.

These constants are used to select the current font to draw the text. See functions SetTextSize and GetTextSize.

GL provides several color constants for 4-bit grayscale OLED display. This display has only 16 indexed colors.GL color constants (4-bit grayscale OLED):

Constant	Value
OLED_4BIT_GRAYSCALE_0	0x0
OLED_4BIT_GRAYSCALE_1	0x1
OLED_4BIT_GRAYSCALE_2	0x2
OLED_4BIT_GRAYSCALE_3	0x3
OLED_4BIT_GRAYSCALE_4	0x4
OLED_4BIT_GRAYSCALE_5	0x5
OLED_4BIT_GRAYSCALE_6	0x6
OLED_4BIT_GRAYSCALE_7	0x7
OLED_4BIT_GRAYSCALE_8	0x8
OLED_4BIT_GRAYSCALE_9	0x9
OLED_4BIT_GRAYSCALE_10	0xA
OLED_4BIT_GRAYSCALE_11	0xB
OLED_4BIT_GRAYSCALE_12	0xC
OLED_4BIT_GRAYSCALE_13	0xD
OLED_4BIT_GRAYSCALE_14	0xE
OLED_4BIT_GRAYSCALE_15	0xF

The OLED_4BIT_GRAYSCALE_0 is the darkest color.
The OLED_4BIT_GRAYSCALE_15 is the lightest color.

Driver header file also provides constants for allowed display interfaces:

Constant	Description
OLED_128_64_4_INTERFACE_1	Driver should use 1st physical interface to OLED display
OLED_128_64_4_INTERFACE_2	Driver should use 2nd physical interface to OLED display

Microcontroller board can have 2 OLED displays at the same time. When you want to use the display you should initialize one of these two interfaces. Please see InitializeGL function.

4. TYPES

GL library provides the following types:

Type	Description
`POINT`	This is the structure that represents X and Y coordinate of point
`P_POINT`	This is the C pointer to POINT
`COLORREF`	Color of pixel

`POINT` has 2 fields:

`POINT.x` - X coordinate of the point
`POINT.y` - Y coordinate of the point

COLORREF is an unsigned long value that represents the color value (Indexed color or RGB value depending on the display driver)

6. ERROR HANDLING

Each function in BiPOM Graphics Library returns GL_ERROR constant if an error occurs in this function. BiPOM Graphics Library defines the following constants:

Constant	Description
GL_OK	This value is returned from the function if no error occurs.
GL_ERROR	This value is returned from the function if an error occurs.

Example:

```
// try to set new color
int err = SetColor(OLED_COLOR_15);

if(err == GL_ERROR)
{
        // function failed
        // Read extended error code
        int ext_err = GetLastError();

        // now ext_err content error code
}
```

List of extended error codes:

Constant	Description
GL_NO_ERROR	No error occurred. The operation completed successfully.

7. FUNCTIONS

GL has the following functions:

Function	Description
Initializing functions:	
InitializeGL	Initializes internal variables of GL. Also initializes driver and specified physical interface of display.
ShutdownGL	Destroys current selected graphics context. Also destroys all resources allocated in driver for this graphics context.
SwitchToContext	Switches current graphics context to specified driver interface.
Functions that work with video RAM buffer:	
GetWidth	Returns width of the screen in pixels.
GetHeight	Returns height of the screen in pixels.
RenderBuffer	Forces copy of all drawn primitives from video buffer to real hardware.
SetPixel	Sets specified pixel to specified color
GetPixel	Gets color of specified pixel.
Functions to draw pixels:	
SetPixelWidth	Sets pixel with specified width and color to specified pixel
Functions to clear display or rectangle:	
ClearDisplay	Clears screen with background color.
ClearRectangle	Clears specified rectangle with background color.
Function to set/get settings of graphics context:	
SetBkColor	Sets background color.
GetBkColor	Returns current background color.
SetColor	Sets foreground color.
GetColor	Returns current foreground color.
SetTextSize	Sets one of predefined fonts as current font.
GetTextSize	Gets current font constant.
GetTextWidth	Returns width of text string in pixels.
SetLineWidth	Sets width of line.
GetLineWidth	Gets current width of line.

‘Draw graphics primitives’ functions:	
DrawLine	Draws line.
DrawRectangle	Draws rectangle.
DrawPolyline	Draws polyline (closed or not closed).
DrawCircle	Draws circle.
DrawEllipse	Draws ellipse.
DrawArc	Draws arc of circle.
DrawText	Draws text string.

InitializeGL

Initializes internal variables of GL. Also initializes driver and specified physical interface of display.

```
int InitializeGL(
        unsigned long Driver,           // Predefined Driver constant from driver header file
        unsigned char Interface         // Predefined display interface constant
);
```

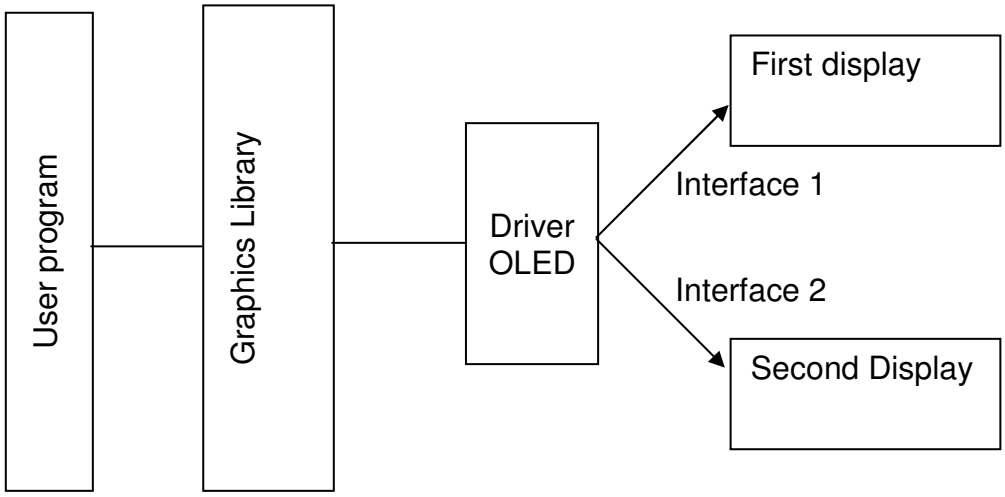
Parameters:

Driver
[in] Predefined Driver constant from the driver header file
Interface
[in] Predefined display interface constant

Return values:
If the function succeeds, the return value is the handle to graphics context
If the function fails, the return value is **GL_ERROR**

Notes:
You should save the returned handle to graphics context. You should pass this handle to **ShutdownGL** function when you are finished using the GL.

You can initialize 4 graphics contexts at the same time. These can be different displays connected to different physical interfaces as shown below:



In this example, the user program uses 2 OLED displays. Each of these displays is connected to its own IO ports. GL Driver knows which ports are used for interface #1 and #2. So, when you pass one of predefined interface constants to **InitializeGL** function, the driver initializes the specified interface.

Graphics library remembers the first initialized context as active, so, if you start drawing, all the graphics output will point to this graphics context.

Example:

```
// initialize 2 OLED displays
int hContext1 = InitializeGL(OLED_128_64_4, OLED_128_64_4_INTERFACE_1);
int hContext2 = InitializeGL(OLED_128_64_4, OLED_128_64_4_INTERFACE_2);

/ /check if context initialized successfully
if (hContext == GL_ERROR || hContext2 == GL_ERROR)
    return;

// set foreground color
SetColor(OLED_4BIT_GRAYSCALE_15);

// by default now active is hContext1, so circle will be drawn on 1 display
DrawCircle(10, 10, 10);
RenderBuffer();

// switch context to second display
SwitchToContext(hContext2);

// draw this circle on second display
DrawCircle(20, 20, 10);
RenderBuffer();

// destroy created contexts
ShutdownGL(hContext1);
ShutdownGL(hContext2);
```

ShutdownGL

Destroys current selected graphics context. Also destroys all resources allocated in the driver for this graphics context.

```
int ShutdownGL(
        int hContext      // handle to context returned from InitializeGL
);
```

Parameters:

hContext
[in] Handle to graphics context returned from **InitializeGL** function

Return values:
GL_OK on success.
GL_ERROR on error.

Note: You should call this function to all initialized contexts.

Examples:

Please see **InitializeGL** example.

SwitchToContext

Switches the current graphics context to specified graphics context.

```
int SwitchToContext(
        int hContext      // handle to context returned from InitializeGL
);
```

Parameters:

hContext
[in] Handle to graphics context returned from **InitializeGL** function

Return values:
GL_OK on success.
GL_ERROR on error.

Remarks:
You should call this function if you use several displays at the same time.

Examples:
Please see **InitializeGL** example.

GetWidth

Returns the width of the screen in pixels.

```
int GetWidth(void);
```

Return values:

The width of the screen in pixels or **GL_ERROR** on error.

Examples:

```
// initialize OLED displays
int hContext1 = InitializeGL(OLED_128_64_4, OLED_128_64_4_INTERFACE_1);

int width = GetWidth();
int height = GetHeight();

// draw line from top left corner to bottom right corner of the screen
DrawlLine(0, 0, width-1, height-1);
RenderBuffer();

// destroy created contexts
ShutdownGL(hContext1);
```

GetHeight

Returns the height of the screen in pixels.

```
int GetHeight(void);
```

Return values:

The height of the screen, in pixels or **GL_ERROR** on error.

Examples:

See **GetWidth** example.

SetPixel

Sets the specified pixel on the screen to the specified color.

```
int  SetPixel(
        int x,                      // X coordinate of pixel
        int y,                      // Y coordinate of pixel
        unsigned long Color     // color of pixel
);
```

Parameters:

x
[in] X coordinate of pixel
y
[in] Y coordinate of pixel
Color
[in] Color of pixel. Should be one of the indexed colors or RGB value, if display supports RGB

Return values:
GL_OK on success.
GL_ERROR on error.

Examples:

```
// initialize OLED displays
int hContext1 = InitializeGL(OLED_128_64_4, OLED_128_64_4_INTERFACE_1);

// Set pixel in position (10, 20)  to the lightest state
SetPixel(10, 20, OLED_4BIT_GRAYSCALE_15);
RenderBuffer();

// destroy created contexts
ShutdownGL(hContext1);
```

GetPixel

Gets color of the specified pixel.

```
COLORREF  GetPixel(
        int x,                  // X coordinate of pixel
        int y,                  // Y coordinate of pixel
);
```

Parameters:

x
        [in] X coordinate of pixel
y
        [in] Y coordinate of pixel

Return values:
        The color of the specified pixel in current color scheme. If the display supports indexed color, then the function returns the index value. If display supports RGB, then function returns RGB value.
        If some error occurs, the function returns **GL_ERROR**.

Examples:

```
        // initialize OLED displays
        int hContext1 = InitializeGL(OLED_128_64_4, OLED_128_64_4_INTERFACE_1);

        // Set pixel in position (10, 20)  to the lightest state
        SetPixel(10, 20, OLED_4BIT_GRAYSCALE_15);
        // Variable 'color' will have value OLED_4BIT_GRAYSCALE_15
        COLORREF color = GetPixel(10, 20);

        RenderBuffer();

        // destroy created contexts
        ShutdownGL(hContext1);
```

RenderBuffer

Forces driver to copy all the drawn primitives from video RAM buffer to real hardware.

```
int  RenderBuffer(void);
```

Parameters:

Nothing

Return values:

GL_OK on success.
GL_ERROR on error.

Notes:

When you call draw functions like **DrawLine**, **DrawCircle** etc., the driver draws all primitives in driver RAM memory buffer. Only when you call **RenderBuffer** function, the memory buffer will copied to the hardware. You don't see any changes on the physical display until you call **RenderBuffer**.

Examples:

See **InitializeGL** example.

SetPixelWidth

Draws pixel with specified width.

```
int  SetPixelWidth(
        int x,                  // X coordinate of pixel
        int y,                  // Y coordinate of pixel
        char width,             // Width of pixel
        COLORREF color          // Color of pixel
);
```

Parameters:

x
        [in] X coordinate of pixel
y
        [in] Y coordinate of pixel
width
        [in] Width of pixel
color
        [in] Color of pixel

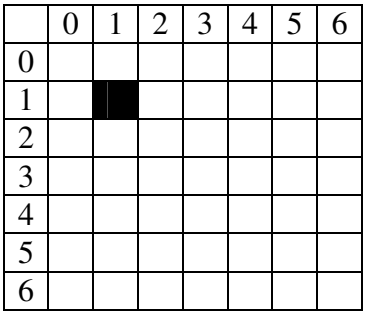
Return values:
        GL_OK on success.
        GL_ERROR on error.

Examples:

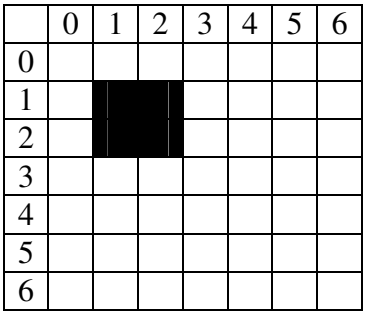
```
        // Initialize graphics context
        // …

        // Draw light pixel in point (10, 10)
        SetPixelWidth(10, 10, 1, OLED_4BIT_GRAYSCALE_15);

        // Draw light pixel in point (20, 20)
        SetPixelWidth(20, 20, 2, OLED_4BIT_GRAYSCALE_15);

        // Draw light pixel in point (30, 30)
        SetPixelWidth(30, 30, 3, OLED_4BIT_GRAYSCALE_15);

        // Flush to real hardware
        RenderBuffer();
```

The pictures below illustrate how the pixels will be drawn:

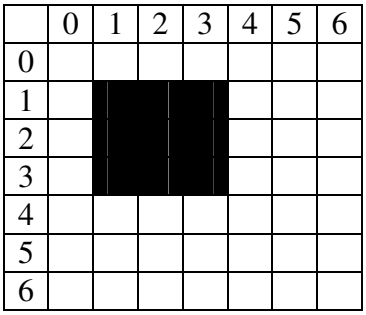
SetPixelWidth(1, 1, 1, OLED_4BIT_GRAYSCALE_0);



SetPixelWidth(1, 1, 2, OLED_4BIT_GRAYSCALE_0);



SetPixelWidth(2, 2, 3, OLED_4BIT_GRAYSCALE_0);



ClearDisplay

Fills the screen with background color.

```
int  ClearDisplay(void);
```

Parameters:

Nothing

Return values:

GL_OK on success.
GL_ERROR on error.

Examples:

```
// Initialize graphics context
// …

// Set background color
SetBkColor(OLED_4BIT_GRAYSCALE_0);

// Clear the screen with color OLED_4BIT_GRAYSCALE_0
ClearDisplay();

// Flush to real hardware
RenderBuffer();

// Set background color
SetBkColor(OLED_4BIT_GRAYSCALE_15);

// Clear the screen with color OLED_4BIT_GRAYSCALE_15
ClearDisplay();

// Flush to real hardware
RenderBuffer();
```

ClearRectangle

Fills the specified rectangle with background color.

```
int  ClearDisplay(
        int x1,             // X coordinate of the left top point
        int y1,             // Y coordinate of the left top point
        int x2,             // X coordinate of the right bottom point
        int y2              // Y coordinate of the right bottom point
);
```

Parameters:

x1
        [in] X coordinate of the left top point
y1
        [in] Y coordinate of the left top point
x2
        [in] X coordinate of the right bottom point
y2
        [in] Y coordinate of the right bottom point

Return values:
        GL_OK on success.
        GL_ERROR on error.

Examples:

```
        // Initialize graphics context
        // …

        // Set background color
        SetBkColor(OLED_4BIT_GRAYSCALE_0);

        // Clear the rectangle with top left corner in point (20,10) and bottom right corner in point (40, 50)
        // with color OLED_4BIT_GRAYSCALE_0
        ClearRectangle(20, 10, 40, 50);

        // Flush to real hardware
        RenderBuffer();

        // Set background color
        SetBkColor(OLED_4BIT_GRAYSCALE_15);

        // Clear the rectangle with top left corner in point (0,0) and bottom right corner in point (10, 20)
        // with color OLED_4BIT_GRAYSCALE_15
        ClearRectangle(0, 0, 10, 20);

        // Flush to real hardware
        RenderBuffer();
```

SetBkColor

Sets the background color in current graphics context.

```
int  SetBkColor(
        COLORREF color, // new background color
);
```

Parameters:

color
[in] New background color

Return values:
GL_OK on success.
GL_ERROR on error.

Examples:
See **ClearDisplay** example.

GetBkColor

Gets the background color of the current graphics context.

```
COLORREF  GetBkColor(void);
```

Parameters:

Nothing

Return values:

Background color value of the current graphics context or **GL_ERROR** if some error occurs.

Examples:

```
// Initialize graphics context
// …

// Get current background color
COLORREF oldBkColor = GetBkColor();

// Set new background color
SetBkColor(OLED_4BIT_GRAYSCALE_15);

// Clear the screen
// with color OLED_4BIT_GRAYSCALE_15
ClearDisplay();

// Flush to real hardware
RenderBuffer();

// Restore old background color
SetBkColor(oldBkColor);
```

SetColor

Sets the foreground color in current graphics context.

```
int  SetColor(
        COLORREF color, // new background color
);
```

Parameters:

color
        [in] New foreground color

Return values:
        GL_OK on success.
        GL_ERROR on error.

Examples:

```
        // Initialize graphics context
        // …

        // Get current foreground color
        COLORREF oldColor = GetColor();

        // Set new foreground color
        SetColor(OLED_4BIT_GRAYSCALE_15);

        // Draw circle
        DrawCircle(10, 10, 5);

        // Flush to real hardware
        RenderBuffer();

        // Restore old foreground color
        SetColor(oldColor);
```

GetColor

Gets the foreground color of the current graphics context.

```
COLORREF  GetColor(void);
```

Parameters:
Nothing

Return values:
Foreground color value of the current graphics context or **GL_ERROR** if some error occurs.

Examples:
See **SetColor** example.

SetTextSize

Sets the new text size in the current graphics context.

```
int  SetTextSize(
        int Size,          // new text size
);
```

Parameters:

Size
[in] New text size. This is one of predefined constants TEXT_SMALL, TEXT_MEDIUM, TEXT_LARGE

Return values:
GL_OK on success.
GL_ERROR on error.

Remarks:
This version of library has only three text sizes. You cannot change font or set any other size. Please refer to BiPOM Professional Graphics Library (GPL) for more sophisticated font support.

Examples:

```
char Msg[32] = "Hello";
int text_width    = 0;
int xp            = 0;
int yp            = 0;

// Get current text size
int oldTextSize = GetTextSize();

// Get screen sizes
int width         = GetWidth();
int height        = GetHeight();

// Set new text size
SetTextSize(TEXT_LARGE);

// calculate coordinates of start point of text
// to center the text on screen
text_width        = GetTextWidth(Msg);
xp                = width/2 - text_width/2;
yp                = 0;

// draw Msg string on the screen
DrawText(xp, yp, Msg);

// Flush to real hardware
RenderBuffer();

// Restore old text size
SetTextSize(oldTextSize);
```

GetTextSize

Gets a text size in the current graphics context.

```
int  GetTextSize(void);
```

Parameters:
Nothing

Return values:
The text size in the current graphics context or GL_ERROR if some error occurs.

Examples:
See **SetTextSize** example.

GetTextWidth

Gets a text string width in pixels depending on the current text size.

```
int  GetTextWidth(const char *pText);
```

Parameters:

pText
[in] Pointer to null terminated string

Return values:
The text width in the pixels or **GL_ERROR** if some error occurs.

Examples:
See **SetTextSize** example.

SetLineWidth

Sets new width of lines in the current graphics context.

```
int  SetLineWidth(
        int Width,          // new line width
);
```

Parameters:

Width
[in] New line width in pixels.

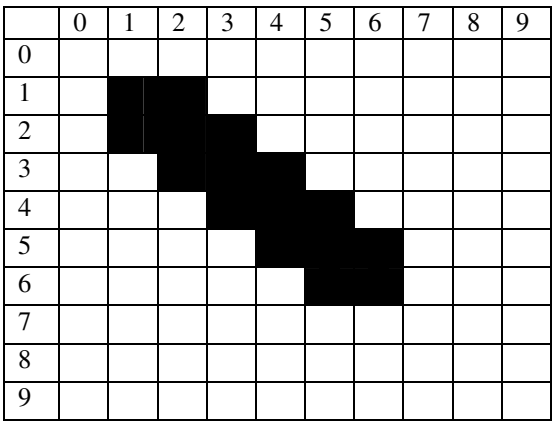
Return values:
GL_OK on success.
GL_ERROR on error.

Examples:

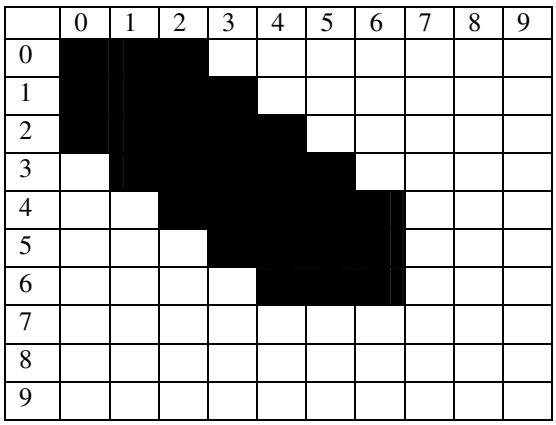
```
// Get current line width
int oldLineWidth = GetLineWidth();

// Set new line width
SetLineWidth(3);

// Draw line from point (5, 5) to point (10, 10)
DrawLine(5, 5, 10, 10);

// Flush to real hardware
RenderBuffer();

// Restore old line width
SetLineWidth(oldLineWidth);
```

The pictures below illustrate how the lines appear when drawn with line width greater then 1:

SetLineWidth(2);
DrawLine(1, 1, 5, 5);



SetLineWidth(3);
DrawLine(1, 1, 5, 5);



GetLineWidth

Gets a width of lines in the current graphics context.

```
int  GetLineWidth(void);
```

Parameters:
Nothing

Return values:
A width of lines in the current graphics context or **GL_ERROR** if some error occurs.

Examples:
See **SetLineWidth** example.

DrawLine

Draws a straight line between 2 points

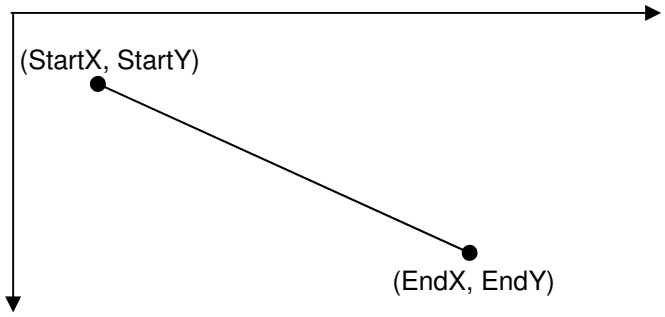
```
int  DrawLine(
        int StartX,        // X coordinate of first point
        int StartY,        // Y coordinate of first point
        int EndX,          // X coordinate of second point
        int EndY,          // Y coordinate of second point
);
```

Parameters:

StartX
[in] X coordinate of the first point
StartY
[in] Y coordinate of the first point
EndX
[in] X coordinate of the second point
EndY
[in] Y coordinate of the second point

Return values:
GL_OK on success.
GL_ERROR on error.

Notes:



Examples:

```
// Get current line width
int oldLineWidth = GetLineWidth();

// Set new line width
SetLineWidth(3);

// Draw line from point (5, 5) to point (10, 10)
DrawLine(5, 5, 10, 10);

// Flush to real hardware
RenderBuffer();

// Restore old line width
SetLineWidth(oldLineWidth);
```

DrawRectangle

Draws a rectangle on 2 points

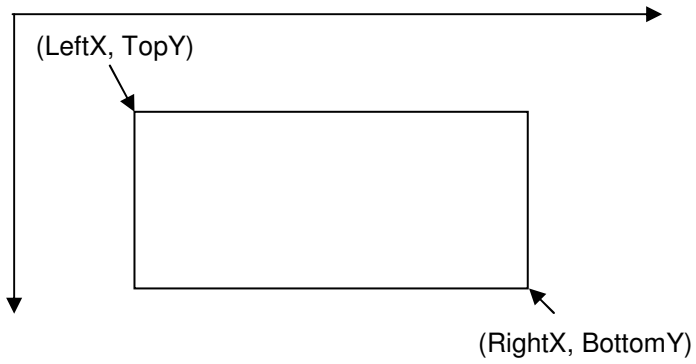
```
int  DrawRectangle(
        int LeftX,        // X coordinate of the first point
        int TopY,         // Y coordinate of the first point
        int RightX,       // X coordinate of the second point
        int BottomY       // Y coordinate of the second point
);
```

Parameters:

LeftX
[in] Left X coordinate of rectangle
TopY
[in] Top Y coordinate of rectangle
RightX
[in] Right X coordinate of rectangle
BottomY
[in] Bottom Y coordinate of rectangle

Return values:
GL_OK on success.
GL_ERROR on error.

Remarks:



Examples:

```
// Get current line width
int oldLineWidth = GetLineWidth();

// Set new line width
SetLineWidth(3);

// Draw rectangle with left top point (5, 5) and right bottom point (10, 10)
Drawrectangle(5, 5, 50, 50);

// Flush to real hardware
RenderBuffer();

// Restore old line width
SetLineWidth(oldLineWidth);
```

DrawPolyline

Draws a polyline

```
int  DrawPolyline(
        int numPoints,   // Number of points in array
        POINT *points,  // Array of points
        char Closed      // Should be drawn the line between the first and last the  points
);
```

Parameters:

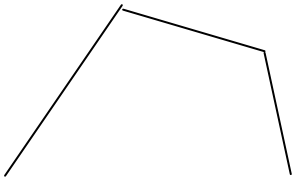
numPoints
[in] Number of points in array
points
[in] Array of points
Closed
[in] Should be drawn line between first and last points

Return values:
GL_OK on success.
GL_ERROR on error.

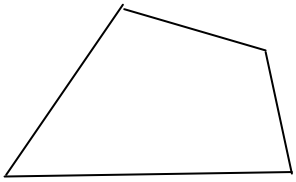
Remarks:

If the third function argument "**Closed**" is zero, then **DrawPolyline** draws **numPoints-1** lines. The first point and the last point are the beginning and the end of polyline. If third function argument is not zero, then **DrawPolyline** draws a polygon. The picture below shows how the third argument affects the result:

Closed == 0



Closed != 0



Examples:

```
// Array of points for polyline
POINT  pArray[4];
pArray[0].x = 0; pArray[0].y = 0;
pArray[1].x = 10; pArray[1].y = 0;
pArray[2].x = 10; pArray[1].y = 10;
pArray[3].x = 0; pArray[3].y = 10;

// Draw polyline with 3 segments
DrawPolyline(4, pArray. 0);

// Draw polyline with connected first and last point
DrawPolyline(4, pArray. 1);

// Flush to real hardware
RenderBuffer();
```

DrawCircle

Draws a circle with the center in point (x, y) and specified radius

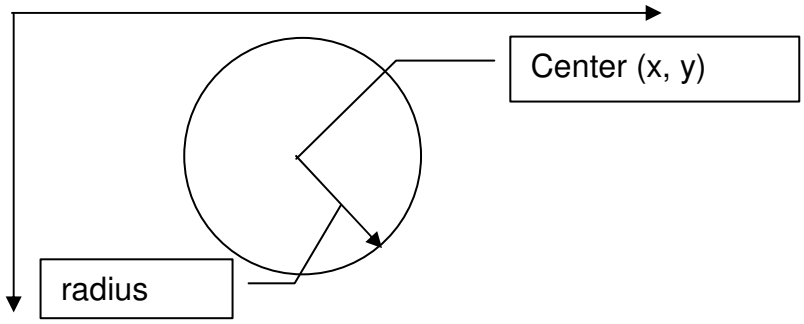
```
int  DrawCircle(
        int x,              // X coordinate of  the center point
        int y,              // Y coordinate of the  center point
        int radius          // Radius of the circle
);
```

Parameters:

x
[in] X coordinate of the center point
y
[in] Y coordinate of the center point
radius
[in] Radius of the circle

Return values:
GL_OK on success.
GL_ERROR on error.

Note:



Examples:

```
// Draw circle with the center in the point (20, 20) and radius = 10 pixels
DrawCircle(20, 20. 10);

// Flush to real hardware
RenderBuffer();
```

DrawEllipse

Draws an ellipse with the center in the point (x, y) , width = xRadius and height = yRadius

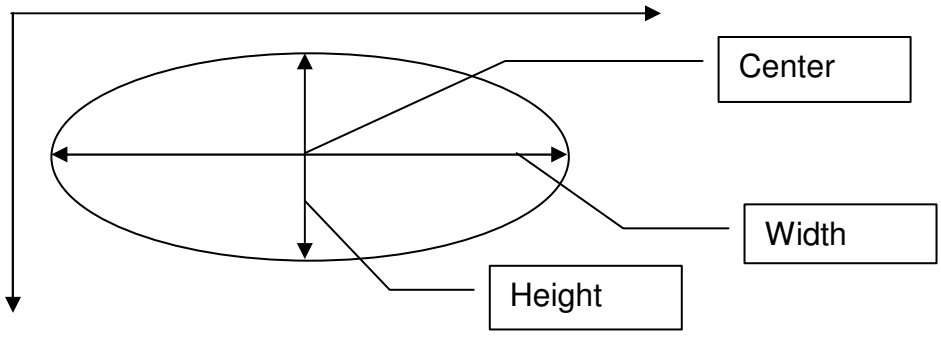
```
int  DrawCircle(
        int x,              // X coordinate of the center point
        int y,              // Y coordinate of the center point
        int xRadius,        // width of the ellipse
        int yRadius         // height of the ellipse

);
```

Parameters:

x
[in] X coordinate of the center point
y
[in] Y coordinate of the center point
xRadius
[in] Width of the ellipse
yRadius
[in] Height of the ellipse

Return values:
GL_OK on success.
GL_ERROR on error.

Notes:



Examples:

```
// Draw ellipse with the center in the point (20, 20);  width is 20 pixels height is 10 pixels
DrawEllipse(20, 20. 20, 10);

// Flush to real hardware
RenderBuffer();
```

DrawArc

Draws an arc of circle

```
int  DrawArc(
        int xCenter,            // X coordinate of the center point
        int yCenter,            // Y coordinate of the center point
        int Radius,             // Radius
        float StartAngle,       // Start angle of the arc
        float ArcAngle          // Arc angle

);
```

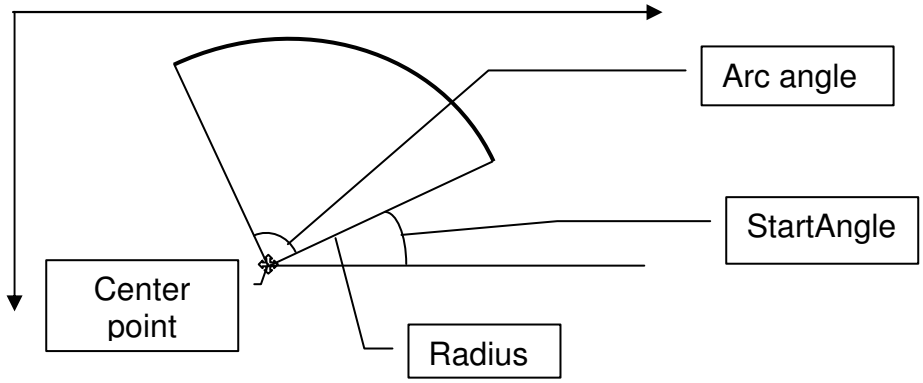
Parameters:

xCenter
[in] X coordinate of the center point
yCenter
[in] Y coordinate of the center point
Radius
[in] Radius of the arc
StartAngle
[in] Start angle of the arc in radians
ArcAngle
[in] Arc angle in radians

Return values:
GL_OK on success.
GL_ERROR on error.

Note:

Arc is always drawn in counterclockwise direction



Examples:

```
// Draw arc
DrawArc(50, 50. 0, PI);

// Flush to real hardware
RenderBuffer();
```

DrawText

Draws a text string

```
int  DrawText(
        int x,                  // X coordinate of the top left point
        int y,                  // Y coordinate of the top left point
        const char *pText       // Null terminated string
);
```

Parameters:

x
[in] X coordinate of the top left point

y
[in] Y coordinate of the top left point

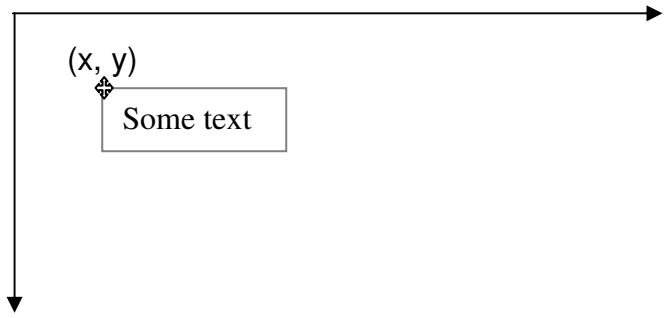
pText
[in] Null terminated string

Return values:
GL_OK on success.
GL_ERROR on error.

Remarks:

Point (x, y) specifies the top left point from wich the text is being drawn. You can set the text size by calling **SetTextSize** function.

NOTE: Text is drawn in foreground color. Backgound of font rectangle is drawn in background color.



Examples:

```
// Draw arc
DrawText(1, 1. “Hello World!”);

// Flush to real hardware
RenderBuffer();
```

8. OLED DEMO

BiPOM Graphics Library provides a DEMO program which demonstrates all the features of the library for an OLED display (128x64, 4 bit grayscale color). DEMO is a C program written in GNUARM C compiler using BiPOM's Micro-IDE Integrated Development Environment.

You can find this DEMO program under Examples in BiPOM ARM7 Development System release.
This program uses Graphics Library and OLED_124x64x4 display driver.

Demo program contains two functions:

```
int main(void);                 // the entry point of program
void OLED_GL_Demo(void);        // the function which makes graphics output
```

Let's look at these functions:

```
int main (void)
{
        /* Initialize the system */
        Initialize();
        delayMs(2000);
        uart0Puts("\n\rBIPOM MINI-MAX/ARM");
        uart0Puts("\n\rOLED-1 DEMO");
        Initialize_SPI();
        DATAFLASH_SPI(SPI_7MHZ_BUS);

        // Initialize Graphics Library
        int hContext1 = InitializeGL(OLED_128_64_4, OLED_128_64_4_INTERFACE_1);


        if(hContext1 == GL_ERROR)
        {
                uart0Puts("\n\rInvalid handle to context");
                uart0Puts("\n\rInitializedGL failed");
                return -1;
        }

        uart0Puts("\n\rGraphics Library initialized");

        OLED_GL_Demo();

        ShutdownGL(hContext1);
        uart0Puts("\n\rGraphics Library closed");

        return 0;
}
```

As first step, the program initializes hardware of ARM7 board (controller, UART0, SPI interface).

Next step is to call **InitializeGL** function in order to obtain graphics context for specified display.

In our case we use OLED display with a resolution of 128x64 pixels and 4-bit grayscale. This display is connected to the board through the first interface. We specified this using the argument to **InitializeGL** function. First parameter is a constant **OLED_128_64_4** which identifies the type of display; the second parameter is a constant **OLED_128_64_4_INTERFACE_1** which identifies the number of hardware interface.

In order to use these constants, you need to include the header file of the driver for your display. Also you need to include the header file of the library.

```
#include <GL\Drivers\oled_128x64x4\oled_128x64x4.h>
#include <GL\InterfaceGL\1.01\GL.h>
```

After calling **InitializeGL,** we should check if the function returns a valid handle to the graphics context. If **InitializeGL** fails, it returns GL_ERROR. Any other value is a handle to the graphics context. If the return value is GL_ERROR you should not call any drawing function. Drawing functions always return GL_ERROR, if a graphics context has not been created.

After initialization of graphics context program, **OLED_GL_Demo()** function is called. This function draws display output (lines, rectangles, circles and others).

When this function is finished, the program frees the graphics context. It calls **ShutdownGL** with handle to the graphics context as argument. This function frees all internal resources. You should call **ShutdownGL** when the graphics context is no more needed.

Let's look at the **OLED_GL_Demo()** function:

```
void OLED_GL_Demo()
{
        // run demo
        GreetingsDemo();
        LinesDemo();
        RectanglesDemo();
        CirclesDemo();
        EllipsesDemo();
        ArcsDemo();
        FontsDemo();
        GoodbyeDemo();
}
```

This function calls several functions. Each of them shows one part of demo.

1. GreetingsDemo

This function shows the greeting text. See the picture below.

Let's look at this function. Most codes in this function are also used in other ones.

```
void GreetingsDemo()
{
        int xp, yp;
        int width, height;
        int text_width;

        width = GetWidth();
        height = GetHeight();

        SetBkColor(OLED_4BIT_GRAYSCALE_0);
        SetColor(OLED_4BIT_GRAYSCALE_15);
        ClearDisplay();

        // draw 1st line of text
        SetTextSize(TEXT_LARGE);

        // calculate coordinates of start point of text
        // to center the text on screen
        text_width = GetTextWidth(GreetingsMsg1);
        xp = width/2 - text_width/2;
        yp = 0;

        DrawText(xp, yp, GreetingsMsg1);

        // draw 2nd line of text

        // calculate coordinates of start point of text
        // to center the text on screen
        text_width = GetTextWidth(GreetingsMsg2);
        xp = width/2 - text_width/2;
        yp = height/3;

        DrawText(xp, yp, GreetingsMsg2);

        // draw 3rd line of text

        // calculate coordinates of start point of text
        // to center the text on screen
        text_width = GetTextWidth(GreetingsMsg3);
        xp = width/2 - text_width/2;
        yp = height/3 * 2;

        DrawText(xp, yp, GreetingsMsg3);

        // flush memory buffer to hardware
        RenderBuffer();

        delayMs(2000);
}
```

First, this function determines the width and the height of the screen. It calls two functions: **GetWidth** and **GetHeight**. The screen resolutions are saved into two variables **width** and **height**. The function use these to place text on the screen.

Next step is to set foreground and background colors. This is done with **SetColor** and **SetBkColor** functions.

Then, the function clears the screen by calling **ClearDisplay** function. Also it changes the size of font to **TEXT_LARGE**. The function **SetTextSize** does this.

Then, the function calculates position of the top left point of text.

```
text_width = GetTextWidth(GreetingsMsg1);
xp = width/2 - text_width/2;
yp = 0;
```

GetTextWidth function returns a width of the string parameter in pixels. The current text size used to determine a width of each symbol.

Then, the function calls **DrawText** and draws text on the screen at the specified position. First line is drawn at the top of the screen (**yp** set to zero).

The second and the third lines are drawn in the same manner. At first, the function calculates the text width. Then, depending on text width, it calculates the **X** coordinate. Also, it sets the **Y** coordinate to draw next line below the previous and calls **DrawText** function.

At the end of the function **GreetingsDemo,** the function **RenderBuffer** is called. This function copies memory buffer to the real hardware (OLED display). At this point, you can see the changes on the display.

Finally, **delayMs** function is called. This is a standard function from BiPOM library. It forces microcontroller to wait the specified number of milliseconds before it continues with the execution of the program.

2. LinesDemo

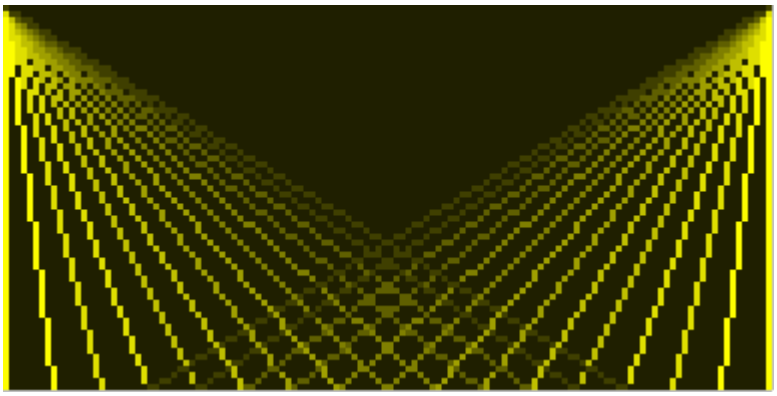
LinesDemo shows different types of lines.

First, it shows the name of the demo. Let's look at the code.

```
// get width and height of screen in pixels
width = GetWidth();
height = GetHeight();

// set back color to black and clear display
SetBkColor(OLED_4BIT_GRAYSCALE_0);
ClearDisplay();

// calculate coordinates of start point of text
// to center the text on screen
text_width = GetTextWidth(LinesDemoMsg);
xp = width/2 - text_width/2;
yp = height/3;

// draw text and flush memory buffer to hardware
DrawText(xp, yp, LinesDemoMsg);
RenderBuffer();
delayMs(2000);
```

As you can see, this code is very similar to the previous function **GreetingsDemo**. **LinesDemo** sets the background color, clears the display, calculates the width of text and starting point of text and draws the text on the screen. Then, **LinesDemo** waits 2 seconds and continues with the execution of the program.

As next step, **LinesDemo** draws two fans from the top left and the top right corners of the screen as shown below:

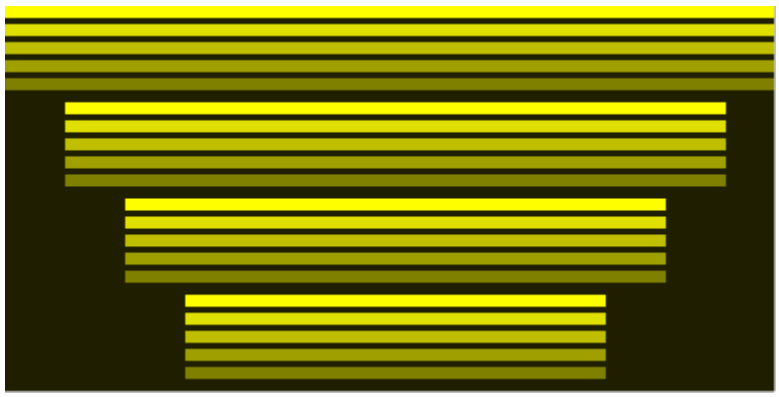


Let's look at the code:

```
ClearDisplay();

color    = OLED_4BIT_GRAYSCALE_15;
xpos1    = 0;
xpos2    = width-1;
step     = width / 15;
num      = 0;
while(num < 14)
{
        // set new color of line
        SetColor(color);

        // draw 2nd lines
        DrawLine(0, 0, xpos1, height);
        DrawLine(width-1, 0, xpos2, height);

        // flush memory buffer to hardware
        RenderBuffer();
        delayMs(200);

        // move positions of second point
        xpos1 += step;
        xpos2 -= step;
        // make color of line more dark
        color--;
        num++;
}

delayMs(2000);
```

First, the function clears the display. Then, it sets the initial values of variables: color of line, **X** coordinate of the bottom point and step between the lines. Then, the program goes into a loop and draws 14 lines from the top left and right corners to the bottom side of the screen. On each step, the function sets a new value for the foreground color and moves the **X** coordinate for the bottom point. Next line is drawn after a small delay (200 milliseconds). When all lines are drawn, the function waits 2 seconds before continuing execution.

After the delay this function draws next portion of lines:



Let’s look at the code.

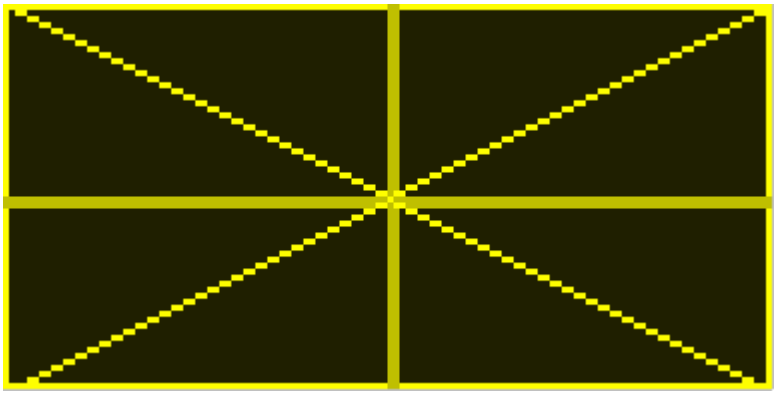
```
// clear display, set line width to 2 points
// set foreground color
ClearDisplay();
SetLineWidth(2);
SetColor(OLED_4BIT_GRAYSCALE_15);

// draw horizontal lines with different color and width
// ystep - y coordinate step between lines
// xwidth - width of each line
ystep = height / 4;
xwidth = width;
for(num=0; num<5; num++)
{
        // make color of each next series of line is more dark
        SetColor(OLED_4BIT_GRAYSCALE_15 - num*2);

        ypos   = 3*num; // start draw next lines on 3 pixel lower then previous
        xpos   = 0;        // left x start coordinate
        xwidth = width; // start width
        while(ypos < height)
        {
                // draw line and flush memory buffer to hardware
                DrawLine(xpos, ypos, xpos+xwidth, ypos);
                RenderBuffer();
                delayMs(400);

                // calculate start point coordinate and width of next line
                ypos   += ystep;
                xpos   += 10;
                xwidth  -= 20;
        }
}


delayMs(2000);
```

First, the function clears the display and sets the foreground. The function **SetLineWidth** sets the line width to 2 pixels.

Then, the function calculates the distance between lines (**ystep**) and width of the first line (**xwidth**). Then, the function starts two loops. First loop will draw four series of lines. Lines of each series have its own color. The second loop draws lines of one series. Before starting the second loop, the program sets the new start **Y** coordinate and restores the width of the line and starting **X** coordinate.

In the second loop, the program calls **DrawLine** to draw the next line. After **DrawLine,** the program calls **RenderBuffer** in order to apply all changes immediately. Lines are drawn with a little delay (200 milliseconds). Then the function calculates **X** and **Y** coordinates of the next line and its width.

After this, the function waits again 2 seconds and draws the last demo screen:



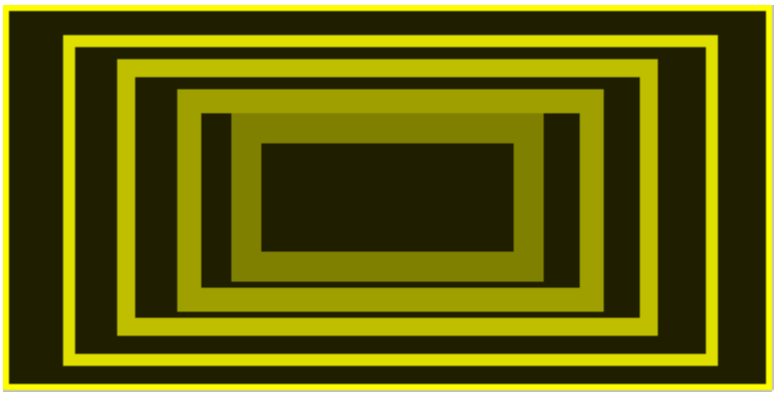
This is a very simple code. The program makes several calls to **DrawLine** function to draw this shape. Lines are drawn with a little delay. Let's look at the code.

```
// pass 3
ClearDisplay();
SetColor(OLED_4BIT_GRAYSCALE_15);
SetLineWidth(1);

// draw borders along all four sides
DrawLine(0, 0, width-1, 0);
DrawLine(0, 0, 0, height-1);
DrawLine(width-1, 0, width-1, height-1);
DrawLine(0, height-1, width-1, height-1);
RenderBuffer();

delayMs(1000);

// draw cross in the center of display
SetColor(OLED_4BIT_GRAYSCALE_10);
SetLineWidth(2);

DrawLine(width/2, 0, width/2, height-1);
DrawLine(0, height/2, width, height/2);
RenderBuffer();

delayMs(1000);

// draw diagonal cross in the center of display
SetColor(OLED_4BIT_GRAYSCALE_15);
SetLineWidth(1);

DrawLine(width/2, height/2, 0, 0);
DrawLine(width/2, height/2, 0, height);
DrawLine(width/2, height/2, width, 0);
DrawLine(width/2, height/2, width, height);
RenderBuffer();

delayMs(2000);
```

As you can see, this function uses **width** and **height** variables to determine coordinates of points.

3. RectanglesDemo

This demo function draws rectangles. The first screen shows rectangles drawn with different line width. The program draws rectangles of different sizes, from large to small. Each next rectangle is drawn with thicker line.

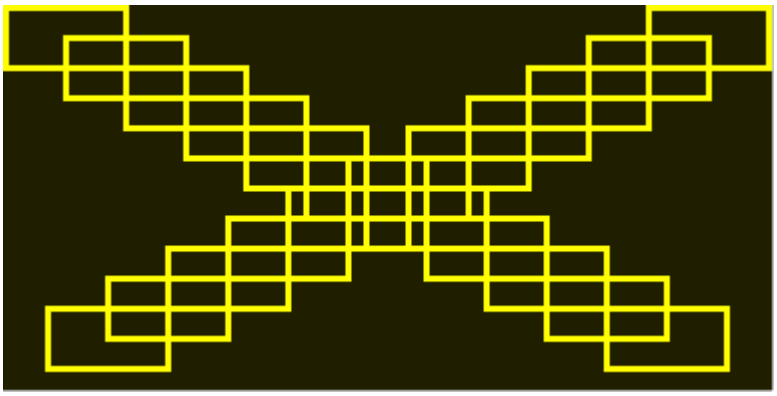


Let's look at the code:

```
// calculate 2 points of the largest rectangle
sx = sy = 0;
ex = width-1;
ey = height-1;
line_width = 1;

// draw 5 rectangles
for(num=0; num<5; num++)
{
        // Change line width and color
        SetLineWidth(line_width);
        SetColor(OLED_4BIT_GRAYSCALE_15 - num*2);

        // draw rectangle and flush memory buffer to hardware
        DrawRectangle(sx, sy, ex, ey);
        RenderBuffer();
        delayMs(200);

        // calculate cordinates of next rectangle
        sx += 10;
        sy += 5;
        ex -= 10;
        ey -= 5;

        // change line width
        line_width++;
}
```

First, the function calculates the coordinates of the largest rectangle. Then, it draws four rectangles in the loop. Before drawing a rectangle, the function sets the new foreground color (darker) and line width (thicker). Then, the function calls **RenderBuffer** to draw rectangles on the screen. After this, the function calculates new coordinates of the next smaller rectangle.

On the next screen, the function draws two sequences of rectangles from the top left and the top right corners to the bottom right and the bottom left corners of the screen.



Let's look at the code:

```
ClearDisplay();
SetLineWidth(1);
SetColor(OLED_4BIT_GRAYSCALE_15);

// calculate coordinates of first rectangle in the top left corner
sx = sy = 0;
ex = sx+20;
ey = sy+10;

// draw 11 rectangles
for(num=0; num<11; num++)
{
        // draw rectangle and flush memory buffer to hardware
        DrawRectangle(sx, sy, ex, ey);
        RenderBuffer();
        delayMs(200);

        // calculate coordinates of the next rectangle
        sx += 10;
        sy += 5;
        ex = sx+20;
        ey = sy+10;
}

// draw the same sequence of rectangles but
// from the top right corner to down left corner

// calculate coordinates of the first rectangle
ex = width-1;
ey = 10;
sx = ex - 20;
sy = ey-10;

// draw 11 rectangles
for(num=0; num<11; num++)
{
        // draw rectangle and flush memory buffer to hardware
        DrawRectangle(sx, sy, ex, ey);
        RenderBuffer();
        delayMs(200);
```

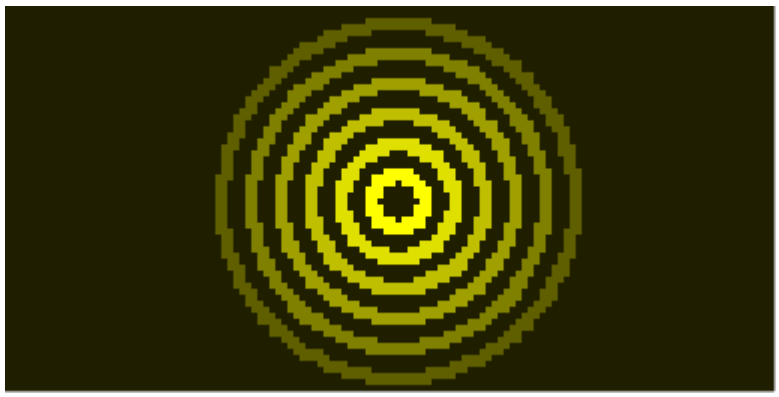
```
            // calculate coordinates of  next rectangle
            sx -= 10;
            sy += 5;
            ex = sx+20;
            ey = sy+10;
    }
```

First, the function draws rectangles from the top left to the bottom right corners of the screen. It calculates the starting point and the ending point coordinates of the first rectangle.

Then, the function draws 11 rectangles in a loop. Each new rectangle is moved 10 pixels to the left and 5 pixels to the bottom. Rectangles are drawn with a little delay. Each new rectangle is drawn on the display immediately because directly after **DrawRectangle**, the function calls **RenderBuffer**.

4. CirclesDemo

This demo function draws several circles with different foreground colors and line width set to 2 pixels on the first screen.



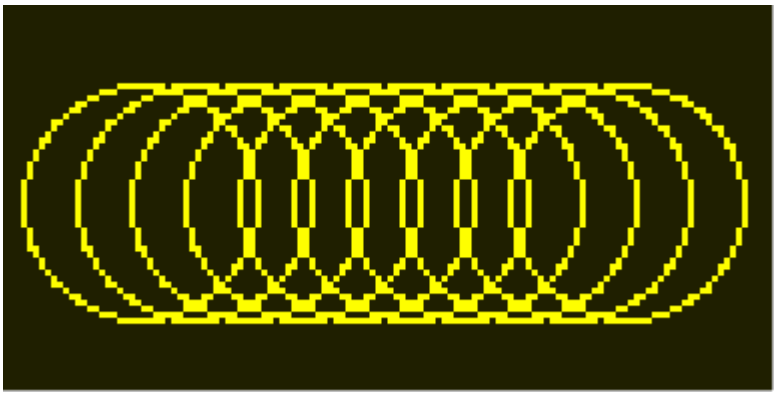
Let's look at the code:

```
// set line width to 2 pixels
SetLineWidth(2);

num = 0;
radius = 5;
while(num < 6)
{
        // change color. make next circle darker then previous
        SetColor(OLED_4BIT_GRAYSCALE_15 - num*2);

        // draw circle and flush memory buffer to hardware
        DrawCircle(width/2, height/2-1, radius);
        RenderBuffer();
        delayMs(200);

        // add 5 pixels to radius to draw larger circle
        radius += 5;
        num++;
}
```

The code is very simple. First, the function sets the starting radius to 5 pixels. Then, it draws five circles in a loop. Before calling **DrawCircle**, the function sets new foreground color (makes each new circle darker).

Then, it calls **DrawCircle** to draw a circle in the video buffer and **RenderBuffer** to show the circle on the display immediately. After this, the function increases radius by 5 pixels and continues loop.

On the next screen, the function draws several circles from the left side of the screen to the right side of the screen. The center point of all the circles lies on the same horizontal axis in the middle of the screen.

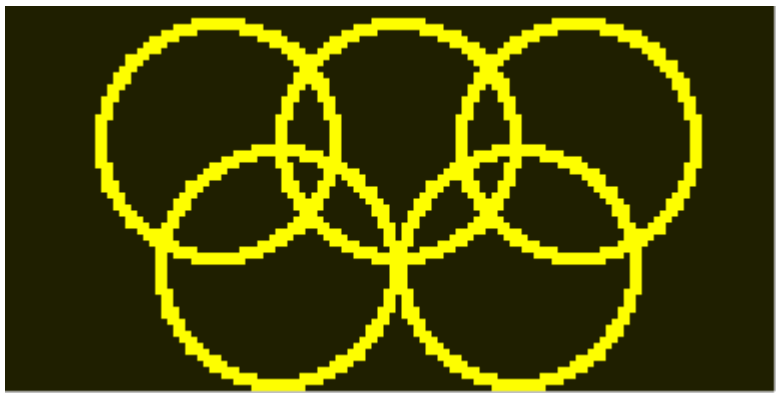


Let's look at the code:

```
// all circles has radius 20 pixels
radius = 20;
// draw 10 circles
for(num=0; num<10; num++)
{
        // draw circle and flush memory buffer to hardware
        DrawCircle(num*9 + 22, height/2, radius);
        RenderBuffer();
        delayMs(200);
}
```

Each circle has a radius of 20 pixels. **Y** coordinate of the center point of the circle is half of the height of the screen. **X** coordinate is moved from left to right. The function calls **DrawCircle** in a loop to draw nine circles. In the loop, only the **X** coordinate of the center point is changed.

On the next screen, this function draws Olympic circles:



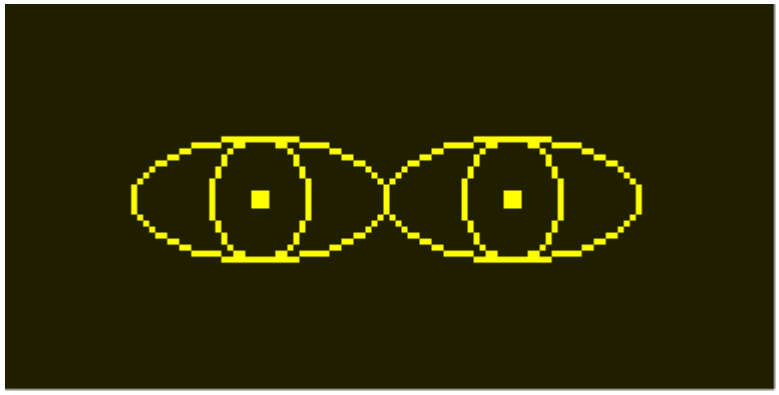
Let's look at the code:

```
ClearDisplay();
SetLineWidth(2);
SetColor(OLED_4BIT_GRAYSCALE_15);

radius = 20;
DrawCircle(width/2 - radius*1.5, height/3, radius);
DrawCircle(width/2, height/3, radius);
DrawCircle(width/2 + radius*1.5, height/3, radius);
DrawCircle(width/2 + radius, height/3*2, radius);
DrawCircle(width/2 - radius, height/3*2, radius);

// flush memory buffer to hardware
RenderBuffer();
```

As you can see, all the circles have a radius of 20 pixels. The function calls **DrawCircle** function five times to draw five circles on the screen.

5. EllipsesDemo

This function shows different ellipses. On the first screen, this function draws animated eyes with ellipses.



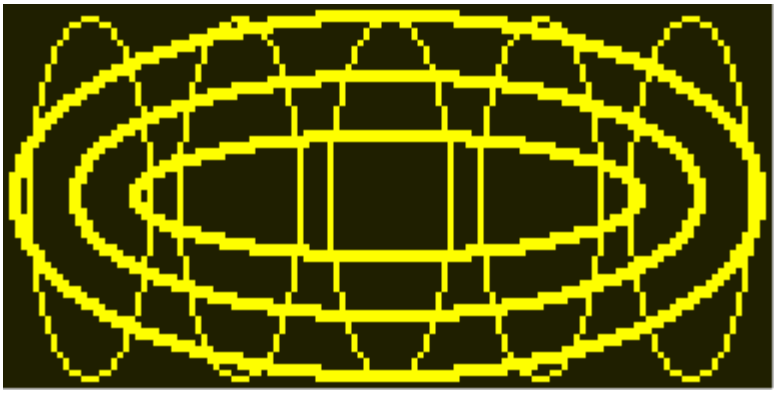
Let’s look at the code:

```
// draw 2 "eye"
DrawEllipse(width/3, height/2, width/6, height/6);
DrawEllipse(width/3*2, height/2, width/6, height/6);
RenderBuffer();

delayMs(200);

// draw 2 draw "pupils"
DrawEllipse(width/3, height/2, height/8, height/6);
DrawEllipse(width/3*2, height/2, height/8, height/6);

SetPixelWidth(width/3, height/2, 3, OLED_4BIT_GRAYSCALE_15);
SetPixelWidth(width/3*2, height/2, 3, OLED_4BIT_GRAYSCALE_15);

RenderBuffer();
delayMs(500);

// clear "pupils" pixels in old position
SetPixelWidth(width/3, height/2, 3, OLED_4BIT_GRAYSCALE_0);
SetPixelWidth(width/3*2, height/2, 3, OLED_4BIT_GRAYSCALE_0);

// draw new "pupils" [pixels in new position
SetPixelWidth(width/3+5, height/2, 3, OLED_4BIT_GRAYSCALE_15);
SetPixelWidth(width/3*2+5, height/2, 3, OLED_4BIT_GRAYSCALE_15);

// flush memory buffer to hardware
RenderBuffer();
delayMs(500);

// clear "pupils" pixels in old position
SetPixelWidth(width/3+5, height/2, 3, OLED_4BIT_GRAYSCALE_0);
SetPixelWidth(width/3*2+5, height/2, 3, OLED_4BIT_GRAYSCALE_0);

// draw new "pupils" [pixels in new position
SetPixelWidth(width/3-5, height/2, 3, OLED_4BIT_GRAYSCALE_15);
SetPixelWidth(width/3*2-5, height/2, 3, OLED_4BIT_GRAYSCALE_15);

// flush memory buffer to hardware
RenderBuffer();
delayMs(500);

// clear "pupils" pixels in old position
SetPixelWidth(width/3-5, height/2, 3, OLED_4BIT_GRAYSCALE_0);
SetPixelWidth(width/3*2-5, height/2, 3, OLED_4BIT_GRAYSCALE_0);

// draw new "pupils" [pixels in new position
SetPixelWidth(width/3, height/2, 3, OLED_4BIT_GRAYSCALE_15);
SetPixelWidth(width/3*2, height/2, 3, OLED_4BIT_GRAYSCALE_15);

// flush memory buffer to hardware
RenderBuffer();
```

The function makes several calls to **DrawEllipse** to draw “eyes” and **SetPixelWidth** to draw “pupils” of eyes. To determine the position and the sizes of ellipses, the function uses width and height of the screen. When the function draws “eyes”, it waits 0.5 seconds and animates the “pupils”.

First, it sets the foreground color to the same value as background and draws “pupils” at the old place. This clears pixels at the old position. Then, the function moves “pupils” coordinates and sets new foreground color. Then, **SetPixelWidth** is called to draw new “pupils” in the new position and **RenderBuffer** is called to draw all the changes on the display. There is a delay of 0.5 seconds before the next animation step. This is a very simple technique of animation.

On the next screen, this function draws several horizontal and vertical ellipses.



Let’s look at the code:

```
ClearDisplay();
SetLineWidth(1);
step = width/5;

// draw 5 vertical ellipses
for(num=0; num<5; num++)
{
        // draw ellipse and flush memory buffer to hardware
        // each next ellipse moved to the right on step pixels
        DrawEllipse(step*num + 14, height/2, 10, height/2-2);
        RenderBuffer();
        delayMs(200);
}

SetLineWidth(2);
for(num=0; num<3; num++)
{
        // draw ellipse and flush memory buffer to hardware
        // each next ellipse has smaller X and Y radiuses on 10 pixels
        DrawEllipse(width/2-1, height/2-1, width/2-10*num-2, height/2-10*num-2);
        RenderBuffer();
        delayMs(200);
}
```

There are 2 loops. First loop draws five vertical ellipses (this is the ellipse which has height greater then width). All ellipses have the same width and height but each next ellipse is drawn to the left of the previous position. The function calls **DrawEllipse** function to draw an ellipse with the specified center point and width and height. Then, it calls **RenderBuffer** to draw the changes immediately on the display.

The second loop draws three horizontal ellipses (width greater than height). Also these ellipses are drawn with line width of 2 pixels.

6. ArcsDemo

This function shows how different arcs can be drawn. On the first screen, you see six arcs.

Let's look at the code:

```
// calculate center point of first arc
step = width/6;
xc = step;
yc = height/3*2;

// all arcs has radius in 15 pixels
radius = 15;
s_angle = 0.0f;             // start angel = 0
a_angle = 3.1415f;          // length of arc is PI

// draw 6 arcs
for(num=0; num<3; num++)
{
        DrawArc(xc, yc, radius, s_angle, a_angle);
        DrawArc(xc, height/3, radius, a_angle, a_angle);
        RenderBuffer();
        delayMs(200);

        // calculate x coordinate of center of next arc
        xc += step*2;
}
```

In order to draw an arc, we should have the following values: center point coordinates, radius, start angle and arc angle (See **DrawArc** function description for more information).

All the arcs on the first screen have a radius of 15 pixels. Also, all the arcs have an arc angle equal to PI (3.1415 radians: half of the circle).

The function draws two arcs in the loop, the first from the start angle of 0 radians, the second from the start angle of PI (3.1415) radians. As soon as the arcs are drawn, the function calls **RenderBuffer** to copy video buffer to real hardware. Then, it moves the X coordinate by 1/6 of the width and continues the loop.

On the second screen, this function draws an animated wave.

Let's look at the code:

```
// set start angle and angle of arc
s_angle = (float)(3.1415 * 1.5); // 1.5*PI
a_angle = 3.1415f;                  // length of arc is PI

// repeat 3 times
for(i=0; i<3; i++)
{
        // set initial line width, color, center point coordinates and radius
        step = width/12;
        line_width = 1;
        xc = 0;
        yc = height/2;
        radius = 5;

        // draw 7 "waves"
        for(num=0; num<7; num++)
        {
                // change line width and color
                // make next "wave" more dark and more thick
                SetLineWidth(line_width);
                SetColor(OLED_4BIT_GRAYSCALE_15 - num*2);

                // draw arc and flush memory buffer to hardware
                DrawArc(xc, yc, radius, s_angle, a_angle);
                RenderBuffer();
                delayMs(200);

                // clear previous drawn arc to make animation effect
                SetColor(OLED_4BIT_GRAYSCALE_0);
                DrawArc(xc, yc, radius, s_angle, a_angle);


                // calculate coordinates of center point of next arc
                xc += step + num*1;
                // increase line width and radius of next arc
                line_width++;
                radius += 5;
        }
}
```

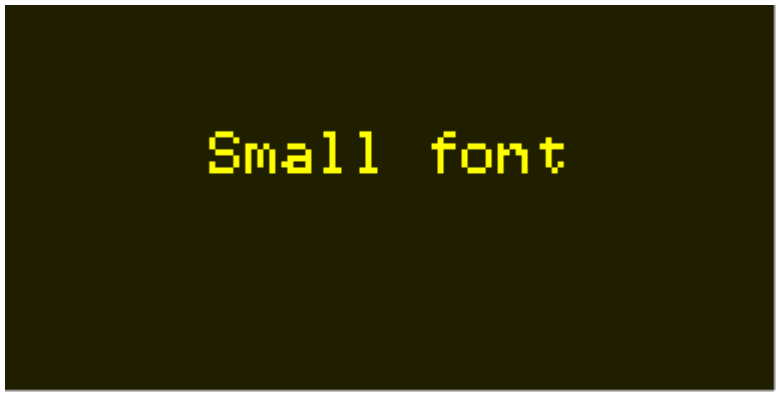
This function draws the same animation three times. Each time the function draws several arcs from the left side of the screen to the right side of the screen. Each next arc is bigger (has bigger radius), thicker and darker. Each time, the function draws seven waves.

Before drawing the next arc, the function sets new foreground color and line width. Then, it calls **DrawArc** to draw next arc. After this, it calls **RenderBuffer** to the draw arc on the physical display. After small delay this function clears this arc from the memory buffer in the following way: At first, it sets foreground color to the same value as a background color; then draws the same arc as previous. This clears the arc. Then, the function increments the radius of the arc and line width. Also, it moves **X** coordinate of the center point.

7. TextDemo

This function shows how to draw text on the screen. This function shows three screens. Each of the screens shows one of available fonts: small, medium, large.

On the first screen, you see text **"Small font"** drawn with the small font:

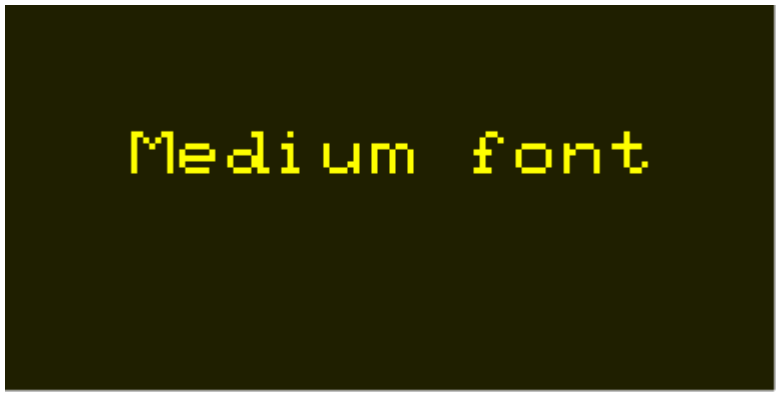


Let's look at the code:

```
// pass 1 - test small font
// clear display
SetColor(OLED_4BIT_GRAYSCALE_15);
SetBkColor(OLED_4BIT_GRAYSCALE_0);
ClearDisplay();

// set small font as current
SetTextSize(TEXT_SMALL);

// calculate coordinates of start point of text
// to center the text on screen
text_width = GetTextWidth("Small font");
xp = width/2 - text_width/2;
yp = height/3;

// draw text and flush memory buffer to hardware
DrawText(xp, yp, "Small font");
RenderBuffer();
```

First, the function sets the foreground and background colors and clears the display. Then, it sets small font as current font. Then it calculates coordinates of the start point of the text to center the text on the screen and calls **DrawText** function. This function draws text to the memory buffer. Then, the function calls **RenderBuffer** to draw text on the display.

The next screen shows medium font size text. The function follows the same steps as small font. The only difference is in **SetTextSize** function. Now it has **TEXT_MEDIUM** as its parameter.



Let's look the code:

```
// pass 1 - test small font
// clear display
SetColor(OLED_4BIT_GRAYSCALE_15);
SetBkColor(OLED_4BIT_GRAYSCALE_0);
ClearDisplay();

// set small font as current
SetTextSize(TEXT_MEDIUM);

// calculate coordinates of start point of text
// to center the text on screen
text_width = GetTextWidth("Medium font");
xp = width/2 - text_width/2;
yp = height/3;

// draw text and flush memory buffer to hardware
DrawText(xp, yp, "Medium font");
RenderBuffer();
```

The third screen shows the large font size text. The function follows the same steps as small font and medium font. The only difference is in **SetTextSize** function. Now it has **TEXT_LARGE** as its parameter.

Let's look at the code:

```
// pass 1 - test small font
// clear display
SetColor(OLED_4BIT_GRAYSCALE_15);
SetBkColor(OLED_4BIT_GRAYSCALE_0);
ClearDisplay();

// set small font as current
SetTextSize(TEXT_LARGE);

// calculate coordinates of start point of text
// to center the text on screen
text_width = GetTextWidth("Large font");
xp = width/2 - text_width/2;
yp = height/3;

// draw text and flush memory buffer to hardware
DrawText(xp, yp, "Large font");
RenderBuffer();
```

8. GoodbyeDemo

This function is very similar to **GreetingsDemo**. It shows two lines of text. Let's look at the code:

```
// get width and height of screen in pixels
width = GetWidth();
height = GetHeight();

// clear screen
SetBkColor(OLED_4BIT_GRAYSCALE_0);
SetColor(OLED_4BIT_GRAYSCALE_15);
ClearDisplay();

// set large font
SetTextSize(TEXT_LARGE);

// calculate coordinates of start point of text
// to center the text on screen
text_width = GetTextWidth(GoodbyeDemoMsg1);
xp = width/2 - text_width/2;
yp = height/3;

// draw text
DrawText(xp, yp, GoodbyeDemoMsg1);

// set small font
SetTextSize(TEXT_SMALL);

// calculate coordinates of start point of text
// to center the text on screen
text_width = GetTextWidth(GoodbyeDemoMsg2);
xp = width/2 - text_width/2;
yp = height/3*2;

// draw text
DrawText(xp, yp, GoodbyeDemoMsg2);

// flush memory buffer to hardware
RenderBuffer();
delayMs(5000);
```