APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi How to Draw Circles and Rectangles

DOCUMENT DATE: **20th MAY 2019**
DOCUMENT REVISION: **1.1**

Description

This application note shows how to program a 4D display module in the Designer environment to make it draw circles and rectangles on the screen. The 4DGL code of the Designer project can be copied and pasted to an empty ViSi project and it will compile normally. The code can also be integrated to that of an existing ViSi project.

Before getting started, the following are required:

- Any Picaso, Diablo16, or Goldelox display module. Visit www.4dsystems.com.au to see the latest products using any of these graphics processors.
- 4D Programming Cable or µUSB-PA5
- Workshop 4 IDE (installed according to the installation document)
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

Note: *The attached Designer project and the discussions in this application note make use of a uLCD-32PTU, which is a 320-pixel-by-240-pixel display. Goldelox displays, however, are usually smaller. The uOLED-96-G2, for instance, is a 96-pixel-by-64-pixel display. For Goldelox display users, the discussions are still relevant and there should be no difficulty in editing the simple project.*

Content

Description 2
Content 2
Application Overview 3
Setup Procedure 3
Create a New Project 3
Design the Project 4
Display Resolution and Coordinate System 4
Start Drawing 5
Draw a Circle 5
gfx_Circle(x, y, radius, colour) 5
gfx_CircleFilled(x, y, radius, colour) 6
Draw a Rectangle 6
gfx_Rectangle(x1, y1, x2, y2, colour) 6
gfx_RectangleFilled(x1, y1, x2, y2, colour) 7
More Functions 7
Clearing the Screen 8
Using the While Loop 8
Animation 10
Run the Program 10
Proprietary Information 11
Disclaimer of Warranties & Limitation of Liability 11

Application Overview

The Designer environment enables the user to write 4DGL code in its natural form to program the display module. 4DGL is a graphics oriented language allowing rapid application development, and the syntax structure was designed using elements of popular languages such as C, Basic, Pascal and others. Programmers familiar with these languages will feel right at home with 4DGL.

The purpose of this application note is, besides showing the user how to draw circles and rectangles, to introduce the basics of 4DGL through examples.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **Designer** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note
Designer Getting Started - First Project

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note
ViSi Getting Started - First Project for Goldelox

or

ViSi Getting Started - First Project for Picaso and Diablo16

Create a New Project

For instructions on how to create a new **Designer** project, please refer to the section "**Create a New Project**" of the application note
Designer Getting Started - First Project

For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note
ViSi Getting Started - First Project for Goldelox

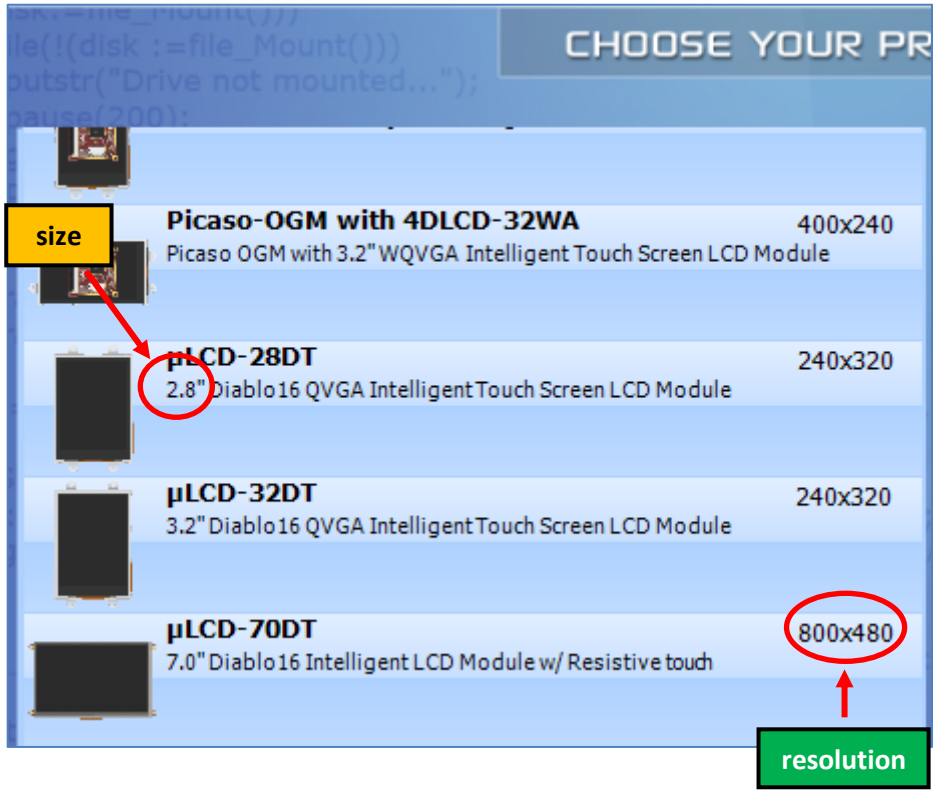
or

ViSi Getting Started - First Project for Picaso and Diablo16

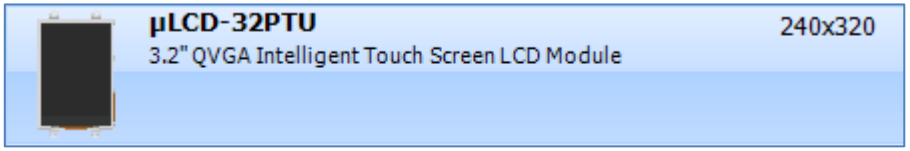
Design the Project

Display Resolution and Coordinate System

Before starting to draw, it is necessary to know how positions of points on the display screen are determined. 4D Systems offers display screens of various resolutions and sizes. When the user opens a new project, the Choose Your Product window shows the screen resolutions and sizes of different display modules.

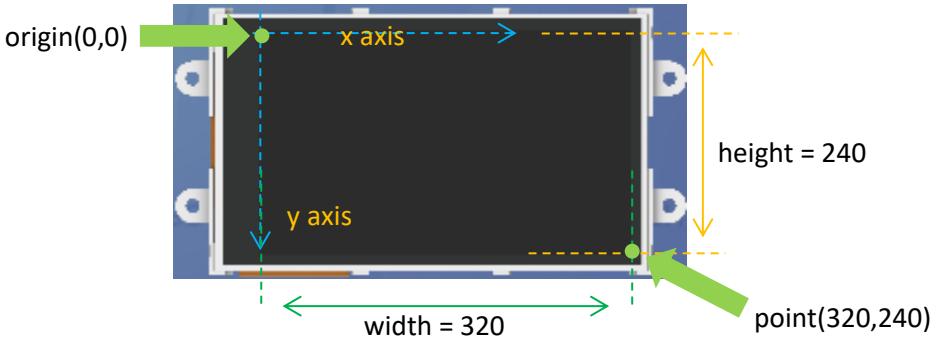


For the uLCD-32PTU screen:

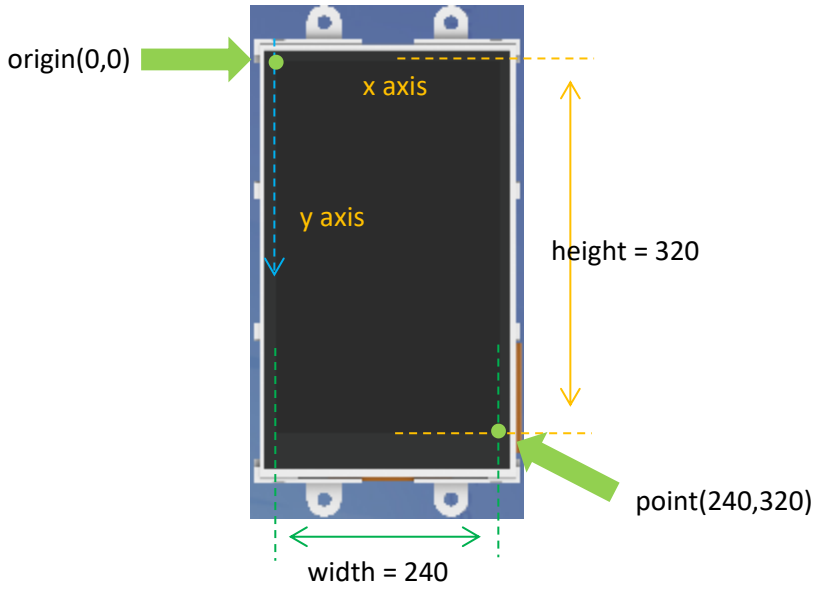


For all modules, positions of points are determined starting from the origin, which is always located at the left-top corner. The location of this reference point is common to all orientations – the portrait, landscape, portrait rotated, and landscape rotated. Two most commonly used orientations are portrait and landscape.

Landscape orientation:



Portrait orientation:



Observe the similarity of the above to the Cartesian coordinate system. The y axis points downward only here, however. This system applies to all display modules and all orientations.

Start Drawing

The user is advised to use the Designer program skeleton provided by Workshop as a starting code. Just place additional codes before line 11. If repetitive operation is desired, the programmer can also insert additional codes between lines 11 and 12 instead.

```
3   #inherit "4DGL_16bitColours.fnc"
4
5   func main()
6
7       gfx_ScreenMode(PORTRAIT) ; // change manually if orientation
8
9       print("Hello World") ;      // replace with your code
10
11      repeat                     // maybe replace
12      forever                    // this as well
13
14  endfunc
```

Insert additional codes here

Draw a Circle

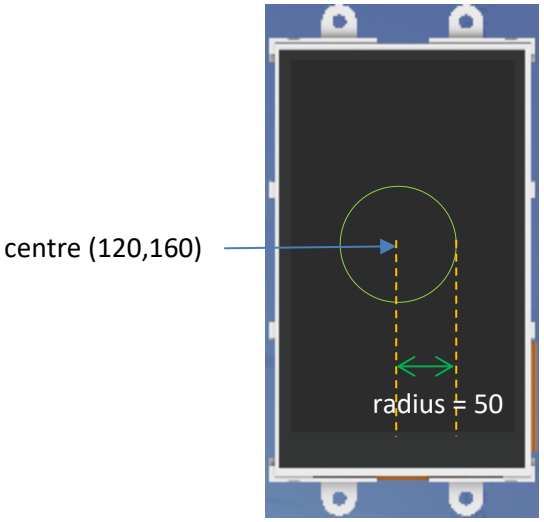
gfx_Circle(x, y, radius, colour)

This function draws a circle with the centre at point **x,y** using the specified **radius** and **colour**.

Example:

```
18      gfx_Circle(120, 160, 50, GREEN);
```

The screen (uLCD-32PTU, portrait orientation in this example) will look like as shown below excluding the labels.



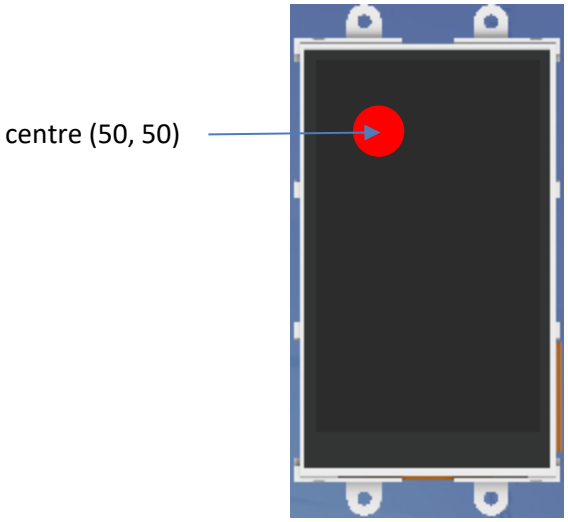
gfx_CircleFilled(x, y, radius, colour)

This function draws a **solid** circle, with the centre at point **x, y** using the specified **radius** and **colour**.

Example:

```
19      gfx_CircleFilled(50, 50, 20, RED);
```

The screen (uLCD-32PTU, portrait orientation in this example) will look like as shown below.



Draw a Rectangle

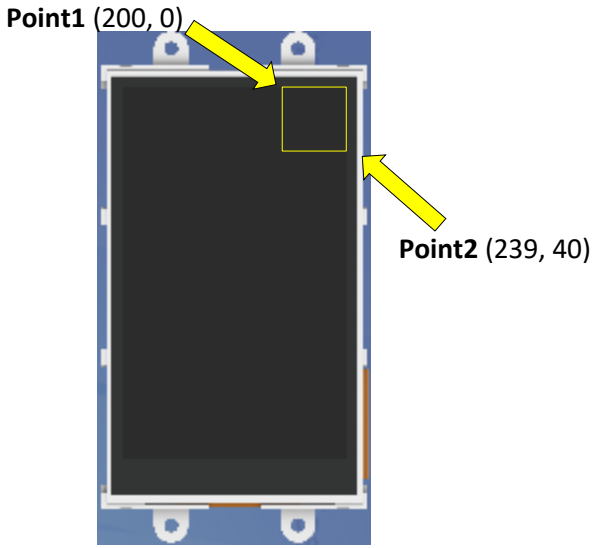
gfx_Rectangle(x1, y1, x2, y2, colour)

This function draws a rectangle from **point1 (x1, y1)** to **point2 (x2, y2)** using the specified **colour**. **Point1** specifies the **top** corner and **point2** specifies the **bottom** opposite corner of the rectangle.

Example:

```
20      gfx_Rectangle(200, 0, 239, 40, YELLOW);
```

The screen will look like as shown below.



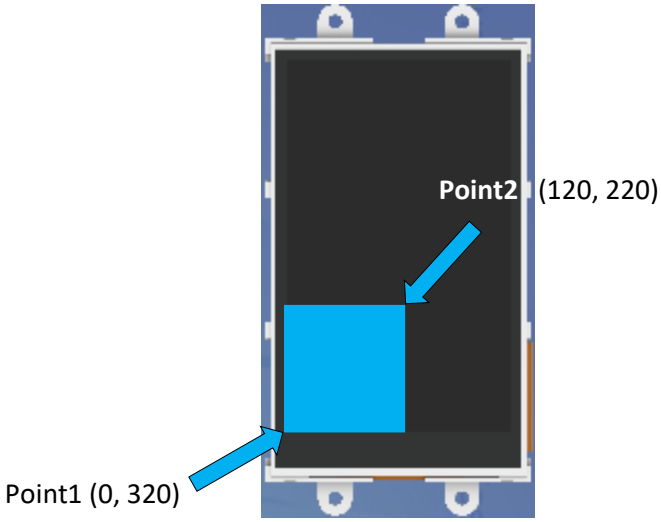
gfx_RectangleFilled(x1, y1, x2, y2, colour)

This function draws a solid rectangle from **point1 (x1, y1)** to **point2 (x2, y2)** using the specified **colour**. **Point1** specifies the **top** corner and **point2** specifies the **bottom** corner of the rectangle.

Example:

```
21    gfx_RectangleFilled(0, 320, 120, 220, BLUE);
```

The screen (uLCD-32DPT, portrait orientation in this example) will look like as shown below.



More Functions

There are other graphics functions for drawing pixels, lines, and shapes and for setting colours and patterns. Learning how to use these functions is relatively easy after having been acquainted with how points are addressed in 4D display screens and how circles and rectangles are drawn. Users who want to experiment with these functions may refer to experiment with these functions may refer to **section 2.6 Graphics Functions** of any of the following documents:

Goldelox Internal Functions Manual
Picaso Internal Functions Manual
Diablo16 Internal Functions Manual

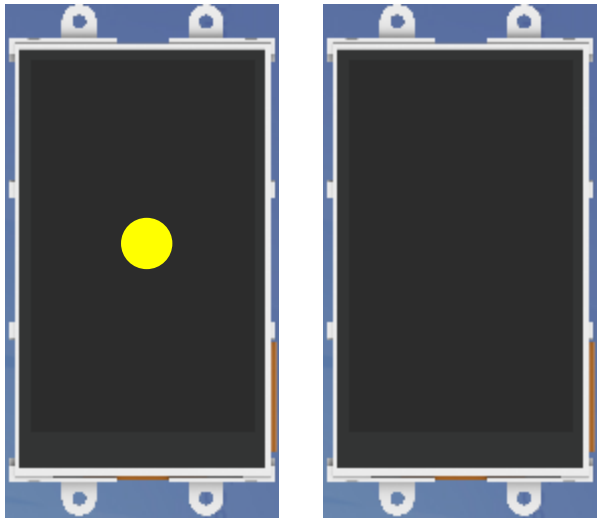
The following sections of this application note discuss how to clear the screen, how to add a delay, and how to use the while loop.

Clearing the Screen

Often times it is necessary to clear the screen to remove unwanted graphics, create a flashing effect, or do animation. Clear the screen using the function **gfx_Cls( )**. To illustrate:

```
27      repeat
28          gfx_CircleFilled(120, 160, 20, YELLOW);
29          pause(500);      //add a 500-millisecond delay
30          gfx_Cls();
31          pause(500);      //add a 500-millisecond delay
32      forever
```

The code above repetitively draws a yellow circle and clears the screen.



Yellow circle Blank

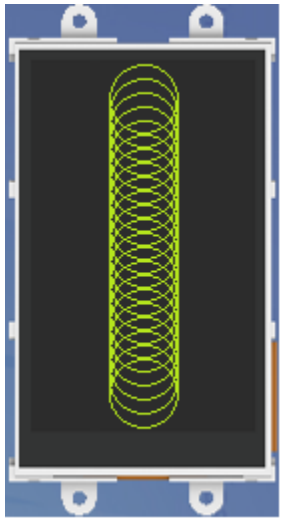
Note that in the code a new function is introduced – **pause (**time**)**. This function adds a time delay, the unit of which is in milliseconds. Adding a delay is necessary for the observer to see the circle since instructions are executed so fast. Try, for example, removing line 29 of the code.

Using the While Loop

Using loops is one way to shorten a code. Again, a loop is a part of a program used to perform repetitive operations. For example, the programmer can use a **while** loop to draw multiple circles. To illustrate:

```
37      var y;
38      y := 20;
39
40      while( y != 300)
41          gfx_Circle(120, y, 20, GREEN);
42          pause(100);
43          //gfx_Cls();
44          y := y + 5;      //can also be y += 5
45      wend
```

The figure above shows a code for drawing circles from top to bottom of the screen. The figure below shows the outcome.



A line-by-line discussion of the code now follows. Lines previously discussed are bypassed.

In line 37, **y** is declared as a **variable**.

```
37      var y;
```

Values of **variables**, as opposed to constants, may **change** during the course of program execution. Here the incrementing value of **y** is used in drawing the circles. Also, the programmer **must** declare a variable first before using it. To learn more about variables in 4DGL, consult 4DGL Programmers Reference Manual.

In line 38, the value of **y** is initially set to 20.

```
38      y := 20;
```

The statements from lines 40 to 45 make up the **while** loop. The instructions inside the loop are executed repetitively while a certain condition is true. Below is the syntax or format for declaring a while loop.

```
Syntax:
        while(condition)
            [statements]
        wend
```

The condition, in this case, is that the increasing value of the variable **y**, which was initially set to 20, is not equal to 300.

```
40      while( y != 300)
```

The statement in line 40 can be interpreted as, “while the value of y is **not equal** to 300”. There other kinds of loops besides the **while** loop. These are discussed in the 4DGL Programmers Reference Manual.

The statement in line 41 now makes use of the value of the variable **y** to draw a green circle.

```
41          gfx_Circle(120, y, 20, GREEN);
```

After a circle is drawn, the value of y is increased by five.

```
44          y := y + 5;      //can also be y += 5
```

Note that the expression **y := y + 5** can also be written as **y += 5**. When the next iteration of the loop occurs, the new circle will be drawn five pixels below the previous one. This repeats until the value of y reaches 300.

Animation

To make the circle appear to move from top to bottom of the screen, uncomment (i.e., remove the double forward slash symbol) the statement in line 43. Compile and download the program. The user can also try creating a circle with a changing radius or colour. A moving rectangle or one whose size is changing is also possible.

Run the Program

For instructions on how to save a **Designer** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

Designer Getting Started - First Project

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

ViSi Getting Started - First Project for Goldelox

or

ViSi Getting Started - First Project for Picaso and Diablo16

The uLCD-32PTU, uLCD-35DT, uOLED-96-G2, and/or uOLED-160-G2 display modules are commonly used as examples, but the procedure is the same for other displays.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability