APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi How to Draw Triangles and Polygons

DOCUMENT DATE: **13th April 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 triangles and polygons 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 / µUSB-PA5/µUSB-PA5-II

for non-gen4 displays (uLCD-xxx)

- 4D Programming Cable & gen4-IB / gen4-PA / 4D-UPA,

for gen-4 displays (gen4-uLCD-xxx)

- micro-SD (µSD) memory card
- 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

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 triangles and polygons, 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 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

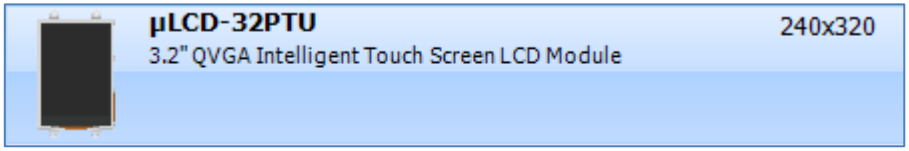
ViSi Getting Started - First Project for Picaso and Diablo16

Design the Project

Display Resolution and Coordinate System

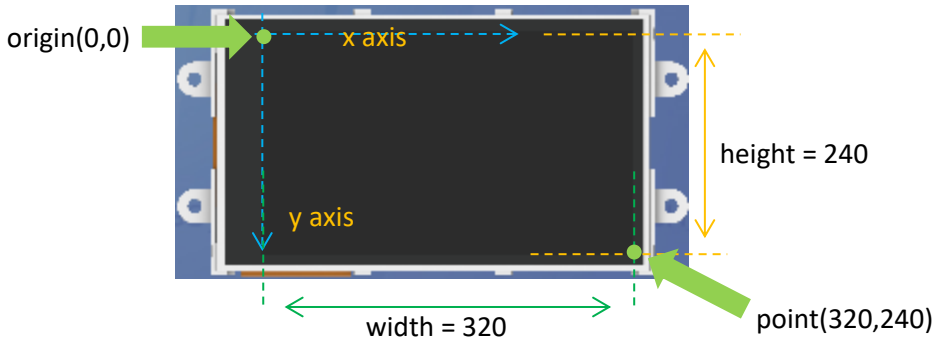
Before starting to draw, it is necessary to know how positions of points on the display screen are determined. 4D Systems offers display screens of various resolutions and sizes. When the user opens a new project, the Choose Your Product window shows the screen resolutions and sizes of different display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

For the uLCD-32PTU screen:

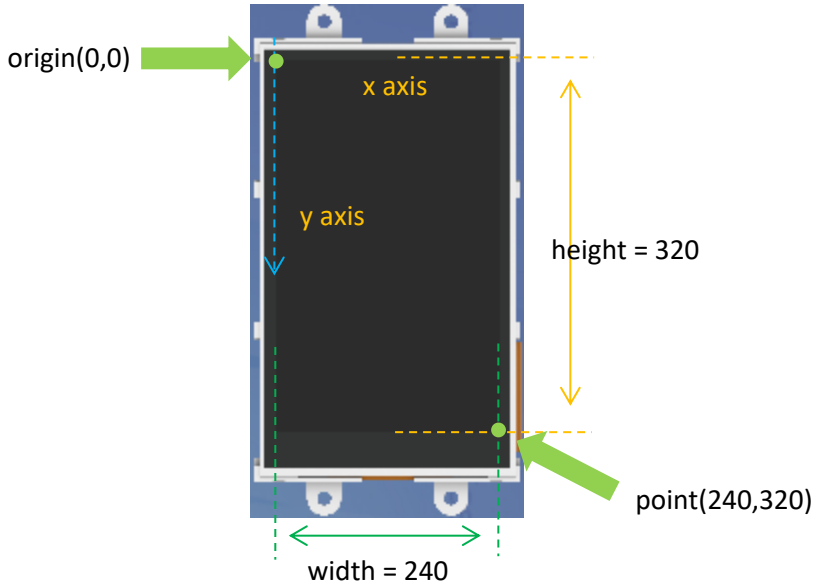


For all modules, positions of points are determined starting from the origin, which is always located at the left-top corner. The location of this reference point is common to all orientations – the portrait, landscape, portrait rotated, and landscape rotated. Two most commonly used orientations are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

```
1  #platform "uLCD-32DT"
2
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
```

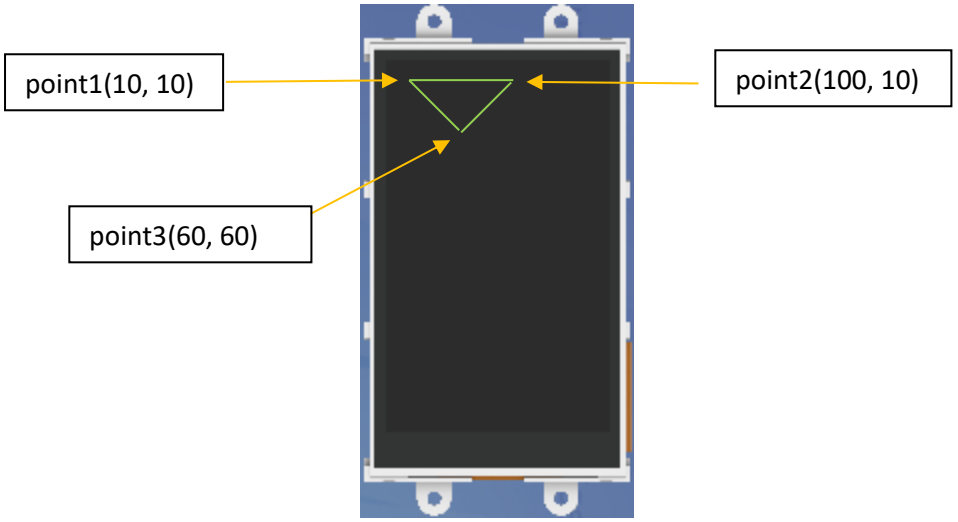
Draw a Triangle

gfx_Triangle(x1, y1, x2, y2, x3,y3, colour)

This function draws a triangle outline between vertices **point1**(x1, y1), **point2**(x2, y2), and **point3**(x3, y3) using the specified **colour**.
Example:

```
13      gfx_Triangle(10, 10, 100, 10, 60, 60, GREEN);
```

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



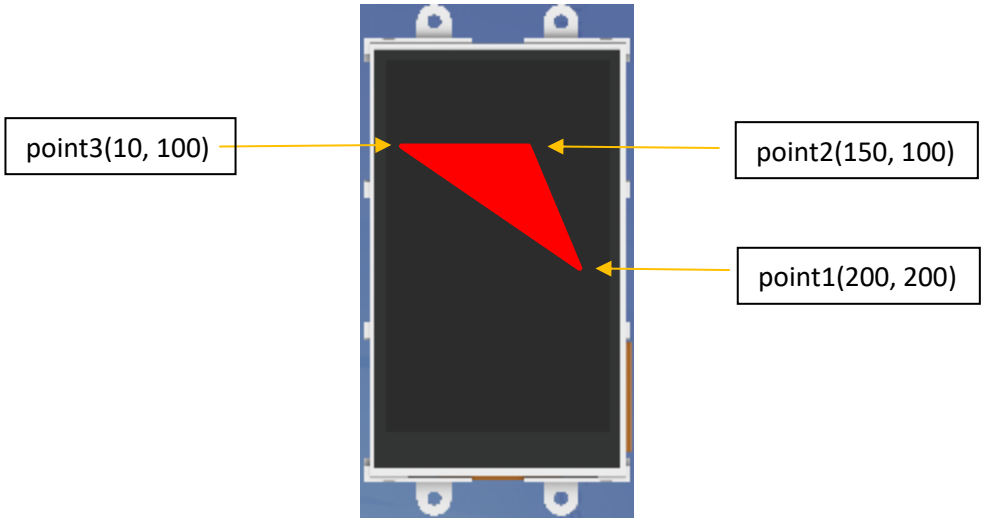
gfx_TriangleFilled(x1, y1, x2, y2, x3,y3, colour)

This function draws a **solid** triangle outline between vertices **point1**(x1, y1), **point2**(x2, y2), and **point3**(x3, y3) using the specified **colour**.

Example:

```
15      gfx_TriangleFilled(200, 200, 150, 100, 10, 100,  RED);
```

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

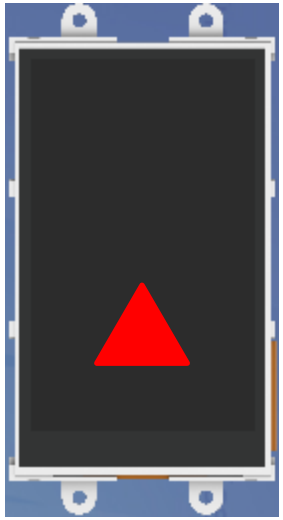


Clear the Screen

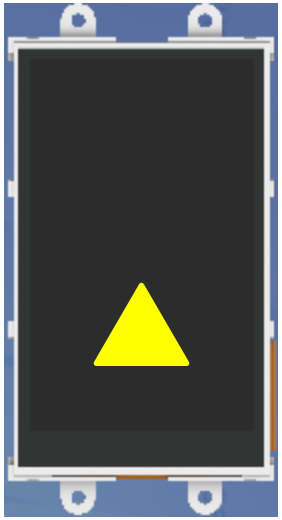
Often times it is necessary to clear the screen to remove unwanted graphics, create a flashing effect, or do animation. Clear the screen using the function **gfx_Cls()**. To illustrate:

```
13  repeat
14      gfx_TriangleFilled(120, 150, 85, 235, 155, 235,  YELLOW);
15      pause(500);     //add a 500-millisecond delay
16      gfx_Cls();
17
18      gfx_TriangleFilled(120, 150, 85, 235, 155, 235,  RED);
19      pause(500);     //add a 500-millisecond delay
20      gfx_Cls();
21  forever
```

The code above repetitively draws a yellow triangle and a red triangle.



Yellow triangle



Red triangle

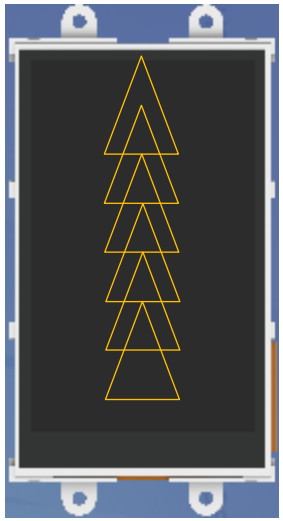
Note that in the code a new function is introduced – **pause(**time**)**. This function adds a time delay, the unit of which is in milliseconds. Adding a delay is necessary for the observer to see the triangles since instructions are executed so fast. Try, for example, removing line 19 of the code.

Using the For/Next 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 **for/next loop** to draw multiple triangles. To illustrate:

```
32 var y;
33
34 for( y := 0; y <= 220; y := y + 50)
35   gfx_Triangle(120, y, 85, y + 100, 155, y + 100, ORANGE);
36   pause(500);
37   //gfx_Cls();
38 next
```

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



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

In line 32, **y** is declared as a **variable**.

```
8   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 triangles. Also, the programmer **must** declare a variable first before using it. To learn more about variables in 4DGL, consult the 4DGL Programmers Reference Manual.

The statements from lines 34 to 38 make up the **for/next loop**.

```
34 for( y := 0; y <= 220; y := y + 50)
35   gfx_Triangle(120, y, 85, y + 100, 155, y + 100, ORANGE);
36   pause(500);
37   //gfx_Cls();
38 next
```

The instructions inside the loop are executed repetitively while a certain condition is true. Below is the syntax or format for declaring a **for/next loop**.

for (variable initialisation; condition; variable update)
    [statements to execute]
next

Applying the syntax above, we analyse line 34.

```
34 for( y := 0; y <= 220; y := y + 50)
```

The statement **y := 0** is the variable initialisation. Here the variable **y** is initialised to its value at the start of the loop (zero). The variable **y** is used for controlling the loop and drawing the triangles.

The statement **y <= 220** is the condition. The statements inside the **for loop** are executed over and over again as long as the value of the variable y is **less than or equal** to 220.

The statement **y := y + 50** is the variable update. This sets the amount by which the variable is changed each time through the loop. Here the value of y is increased by 50 after each iteration or cycle of the loop.

The line below now draws an orange triangle.

```
35   gfx_Triangle(120, y, 85, y + 100, 155, y + 100, ORANGE);
```

Note that for each vertex, the y coordinate uses the variable **y**. Further analysis will show that as the loop iterates, a series of triangles are drawn from top to bottom, with a vertical spacing of 50 pixels.

There other kinds of loops besides the **for loop**. These are discussed in the 4DGL Programmers Reference Manual.

Line 36 adds a delay so that the observer can see the triangle. Line 37 is a comment and is ignored by the compiler.

Line 38 terminates the for/next loop.

Animation

To make the triangle appear to move from top to bottom of the screen, uncomment (remove the double forward slash symbol) the statement in line

37. Compile and download the program. The user can also try creating a triangle with a changing size and colour.

Arrays

It is necessary that the user has an understanding of the concept of arrays prior to drawing polygons. An array can be thought of as a variable containing many values. In the previous section the user was introduced to the use of variables. Variables are simply names used to refer to some location in memory – a location that holds a value. In the for loop code presented in the previous section, the variable y has a value that changes as the loop repeats.

It is also possible for the user to store many values under a variable by making it an array. For example, to create an array **y** that contains five values or that has five elements, declare it first as:

```
8   var y[5];
```

Then assign a value to each element of the array using the format:

```
14  y[n] := value;
```

Here the symbol **n** inside the brackets is called the index. Note that declaring an array (line 8) is different from assigning values to array elements (line 14). The programmer cannot assign values to elements of an array without having declared it first. In other words, the array must be declared before its elements are initialised. It is also possible to declare an array and initialise its elements in a single line, as will be shown later. Now to assign values to each element:

```
16  y[0] := 56;
17  y[1] := 67;
18  y[2] := 45;
19  y[3] := 23;
20  y[4] := 12;
```

Note that there are a total of five elements and that the index starts at zero and ends at four. To print the value of the third element:

```
22  print(y[2]);
```

To print the value of the fifth element:

```
23  print(y[4]);
```

Observe that the first index number, which corresponds to the first element, is zero. The last index number is the number of elements minus one. A five-element array, therefore, has the index numbers 0 to 4.

To declare an array and initialise its elements in a single line:

```
12  var y[5] := [56, 67, 45, 23, 12];
```

Draw a Polygon

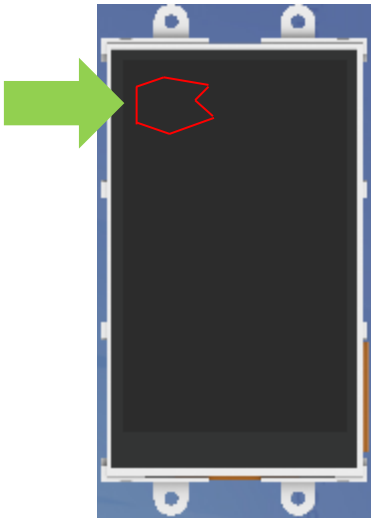
gfx_Polygon(n, vx , vy, colour)

This function plots lines between points specified by a pair of arrays using the specified colour. The last point is drawn back to the first point, completing the polygon.

Example:

```
43 var vx[7], vy[7];    //array declaration
44
45 //initialisation
46 vx[0] := 10;     vy[0] := 10;
47 vx[1] := 35;     vy[1] :=  5;
48 vx[2] := 80;     vy[2] := 10;
49 vx[3] := 60;     vy[3] := 25;
50 vx[4] := 80;     vy[4] := 40;
51 vx[5] := 35;     vy[5] := 50;
52 vx[6] := 10;     vy[6] := 40;
53
54 gfx_Polygon(7, vx, vy, RED);
55
56 repeat                        // maybe replace
57 forever                       // this as well
```

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

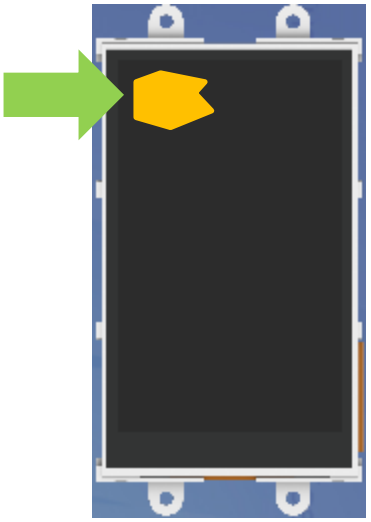


gfx_PolygonFilled(n, vx , vy, colour)

This function draws a solid polygon by plotting lines between points specified by a pair of arrays and using the specified colour. The last point is drawn back to the first point, completing the polygon. Replace the command in line 54 of the code with

```
54     gfx_PolygonFilled(7, vx, vy, ORANGE);
```

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



More Functions

There are other functions for drawing graphics 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 triangles and polygons are drawn. Users who want 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

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

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.
4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.
All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either expresses or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.
Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.
In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.
4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage (‘High Risk Activities’). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.
Use of 4D Systems’ products and devices in 'High Risk Activities' and in any other application is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.