APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Serial Arduino How to Draw Shapes

DOCUMENT DATE:	**15th April 2019**
DOCUMENT REVISION:	**1.1**

W W W . 4 D S Y S T E M S . C O M . A U

Description

This Application Note explores the possibilities provided by the Serial environment in Workshop for a 4D display module to work with an Arduino host. In this example, the host is an Arduino Uno board. The host can also be an Arduino Mega 2560 or Due. Ideally, the application described in this document should work with any Arduino board that supports software serial or with at least one UART serial port. See specifications of Aduino boards here.

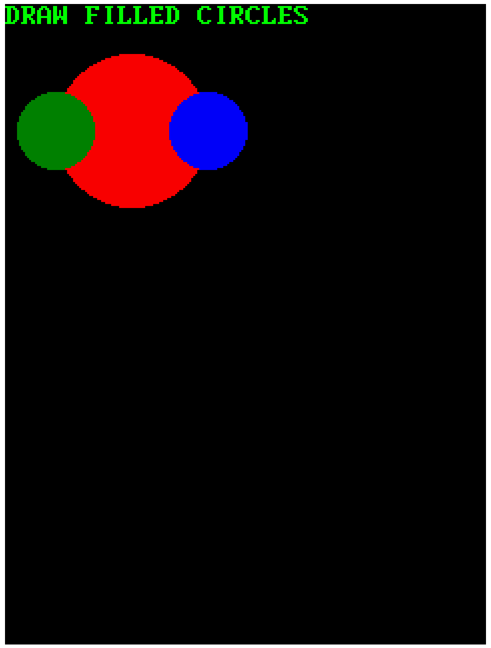
Before getting started, the following are required:

- 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.

Content

Application Overview

In this application note, the Arduino host sends commands to draw pixels, lines, circles, rectangles, triangles, polylines, and polygons on the display.



Setup Procedures

The display must be configured as a slave device first before it can be controlled by a host. For instructions on how to launch Workshop 4, how to connect the display module to the PC, and how to configure the display as a slave device, kindly refer to the section "**Setup Procedure**" of any of the application notes below. Choose according to your display module's processor.

Serial Goldelox Getting Started - The SPE Application

Serial Picaso Getting Started - The SPE Application

Serial Diablo16 Getting Started - The SPE Application

These application notes also introduce the user to the Serial Protocol thru the use of the Serial Commander.

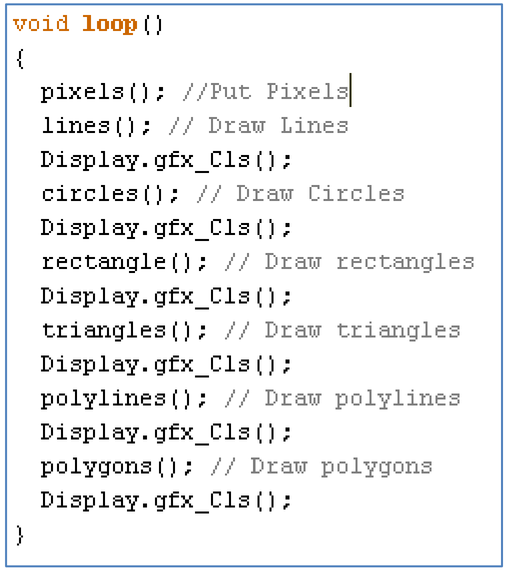
Program the Arduino Host

A thorough understanding of the application note **Serial Connection to an Arduino Host** is required before attempting to proceed further beyond this point. **Serial Connection to an Arduino Host** provides all the basic information that a user needs to be able to get started with the Serial Environment and Arduino. The following is a list of the topics discussed in **Serial Connection to an Arduino Host**.

- How to download and install the Serial-Arduino library (choose a library according to your display module's processor)
- How to modify the library for Arduino Due (due to a Due bug reported by a forum user)
- How to define the serial port to be used for talking to the display
- How to set the baud rate
- The Error Handling Routine
- How to set the Timeout Limit
- How to reset the Arduino Host and the Display
- How to let the Display Start Up
- How to set the Screen Orientation
- How to Clear the Screen
- The uSD Card Mount Routine
- How to enable message logging to the Serial Monitor of the Arduino IDE

Discussion of any of these topics is avoided in other Serial-Arduino application notes unless necessary. Users are encouraged to read **Serial Connection to an Arduino Host** first.

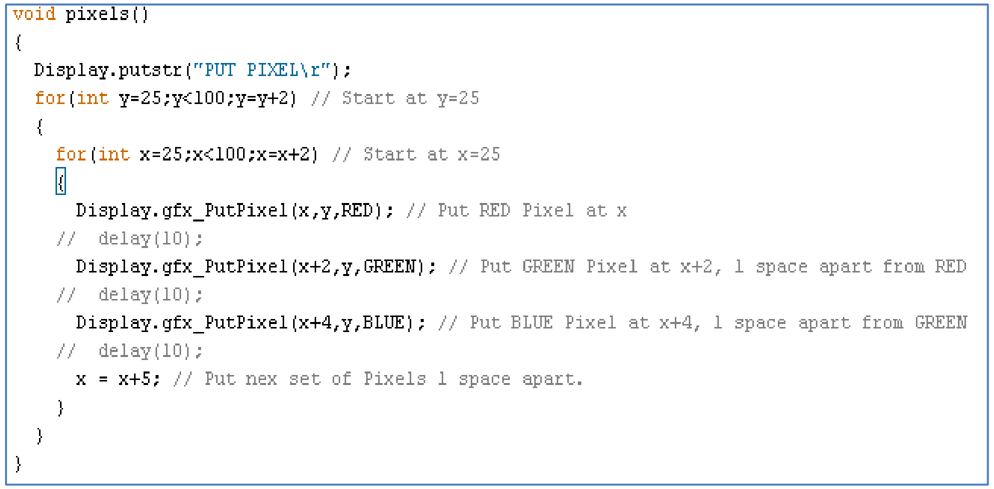
Main Loop



```
void loop()
{
  pixels(); //Put Pixels
  lines(); // Draw Lines
  Display.gfx_Cls();
  circles(); // Draw Circles
  Display.gfx_Cls();
  rectangle(); // Draw rectangles
  Display.gfx_Cls();
  triangles(); // Draw triangles
  Display.gfx_Cls();
  polylines(); // Draw polylines
  Display.gfx_Cls();
  polygons(); // Draw polygons
  Display.gfx_Cls();
}
```

The main loop repeats the entire drawing process on the display. The loop calls the different functions for drawing on the display. The functions are for drawing pixels, lines, circles, rectangles, triangles, polylines, and polygons. After each function is called, the display is cleared.

Put Pixels



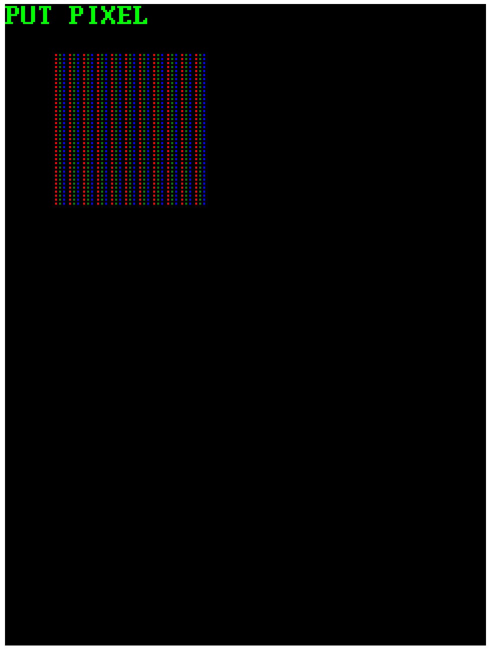
```
void pixels()
{
  Display.putstr("PUT PIXEL\r");
  for(int y=25;y<100;y=y+2) // Start at y=25
  {
    for(int x=25;x<100;x=x+2) // Start at x=25
    {
      Display.gfx_PutPixel(x,y,RED); // Put RED Pixel at x
    //  delay(10);
      Display.gfx_PutPixel(x+2,y,GREEN); // Put GREEN Pixel at x+2, 1 space apart from RED
    //  delay(10);
      Display.gfx_PutPixel(x+4,y,BLUE); // Put BLUE Pixel at x+4, 1 space apart from GREEN
    //  delay(10);
      x = x+5; // Put nex set of Pixels 1 space apart.
    }
  }
}
```

This function shows how to put pixels on the screen. The command

gfx_PutPixel (x, y, colour)

requires three parameters. The 'x' and 'y' parameters are the coordinates where the pixel is going to appear. The 'colour' parameter tells what will be the colour of the pixel. Values can be "RED", "GREEN", "BLUE", "LIME", etc.

Output:



```
PUT PIXEL
```

Draw Lines

Vertical Lines

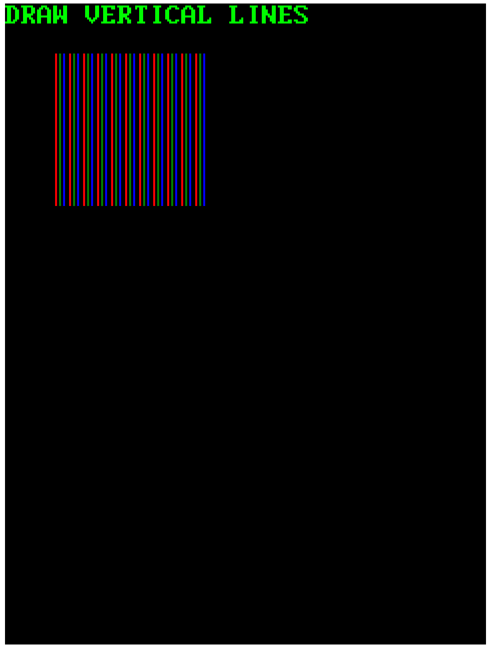
```
Display.putstr("DRAW VERTICAL LINES    \r");
for(int x=25;x<100;x=x+2) // Start at x=25
{
  Display.gfx_Line(x,25,x,100,RED);
  delay(30);
  Display.gfx_Line(x+2,25,x+2,100,GREEN);
  delay(30);
  Display.gfx_Line(x+4,25,x+4,100,BLUE);
  delay(30);
  x = x+5;
}
```

This code draws vertical lines one by one from left to right. The colour of the line changes from red to green to blue. The function

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

requires five parameters. The x1 and y1 parameters specify the starting coordinates of the line. The x2 and y2 parameters specify the ending coordinates of the line. The colour parameter specifies the colour of the line.

Output:

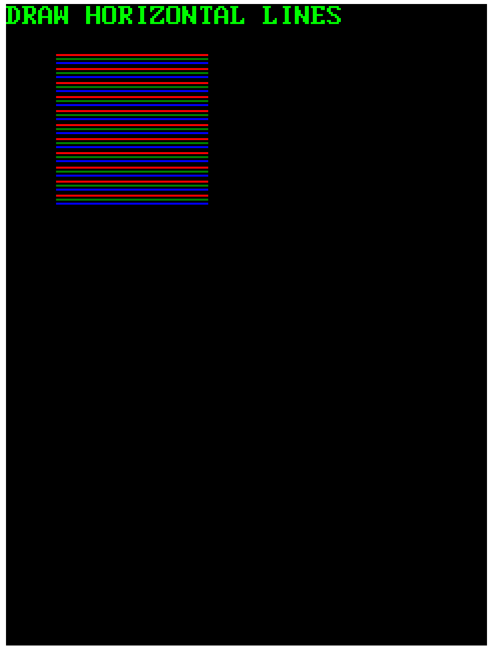


Horizontal Lines

```
Display.putstr("DRAW HORIZONTAL LINES\r");
for(int y=25;y<100;y=y+2) // Start at x=25
{
  Display.gfx_Line(25,y,100,y,RED);
  delay(30);
  Display.gfx_Line(25,y+2,100,y+2,GREEN);
  delay(30);
  Display.gfx_Line(25,y+4,100,y+4,BLUE);
  delay(30);
  y = y+5;
}
Display.gfx_Cls();
```

This code draws horizontal lines one by one from top to bottom.

Output:



```
DRAW HORIZONTAL LINES
```

Draw Circles

Hollow Circles

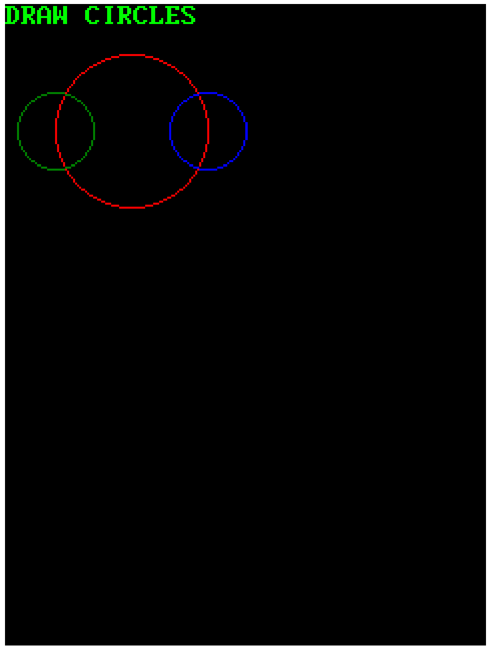
```
Display.putstr("DRAW CIRCLES          \r");
Display.gfx_Circle(63,63,38,RED);
delay(500);
Display.gfx_Circle(25,63,19,GREEN);
delay(500);
Display.gfx_Circle(101,63,19,BLUE);
delay(500);
```

This code draws three circles as shown in the image below. The function

gfx_Circle (x, y, rad, colour)

requires four parameters. The x and y parameters specify the coordinates of the centre of the circle. The rad parameter specifies the radius of the circle. The colour parameter specifies the colour of the outline of the circle.

Output:



Filled Circles

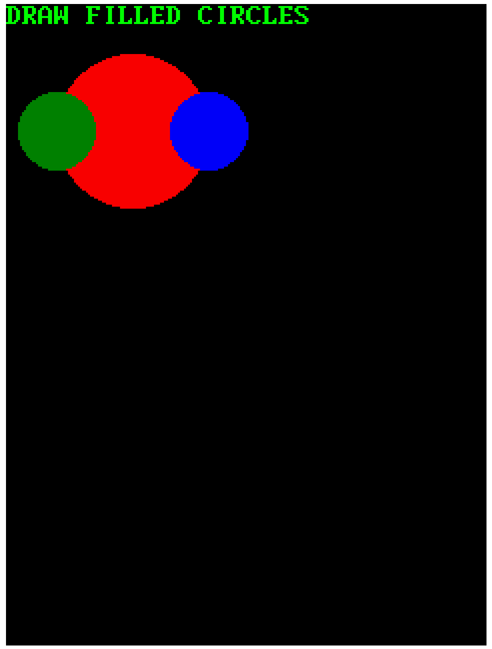
```
Display.putstr("DRAW FILLED CIRCLES   \r");
Display.gfx_CircleFilled(63,63,38,RED);
delay(500);
Display.gfx_CircleFilled(25,63,19,GREEN);
delay(500);
Display.gfx_CircleFilled(101,63,19,BLUE);
delay(500);
```

This code draws three filled circles as shown in the image below. The function

gfx_CircleFilled (x, y, rad, colour)

requires four parameters. The x and y parameters specify the centre of the circle. The rad parameter specifies the radius of the circle. The colour parameter specifies the colour of the filled circle.

Output:



```
DRAW FILLED CIRCLES
```

Draw Rectangles

Hollow Rectangle

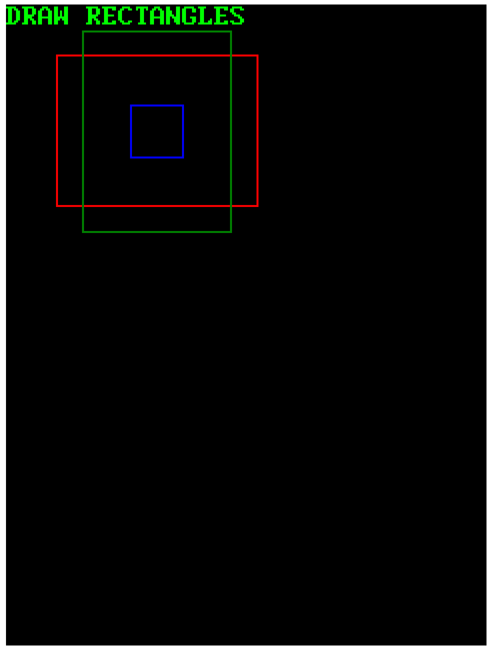
```
Display.putstr("DRAW RECTANGLES        \r");
Display.gfx_Rectangle(25,25,125,100,RED);
delay(500);
Display.gfx_Rectangle(38,13,112,113,GREEN);
delay(500);
Display.gfx_Rectangle(62,50,88,76,BLUE);
delay(500);
```

This code draw three rectangles with different colours as shown in the image below. The function

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

requires five parameters. The x1 and y1 parameters specify the top left corner coordinates of the rectangle. The x2 and y2 parameters specify the bottom right corner coordinates of the rectangle. The colour parameter specifies the outline colour of the rectangle.

Output:



Filled Rectangle

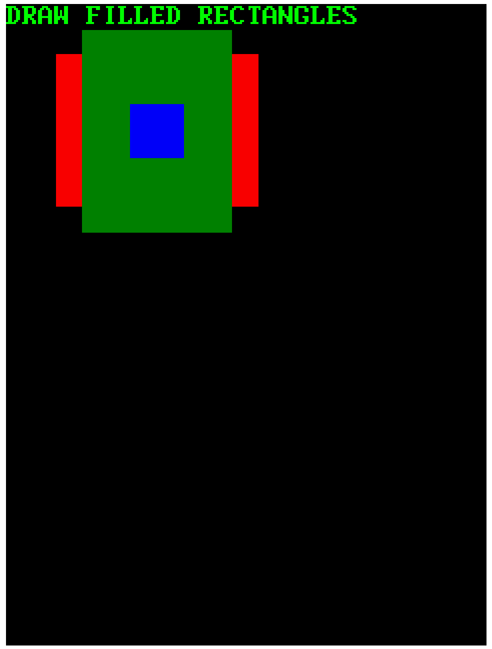
```
Display.putstr("DRAW FILLED RECTANGLES\r");
Display.gfx_RectangleFilled(25,25,125,100,RED);
delay(500);
Display.gfx_RectangleFilled(38,13,112,113,GREEN);
delay(500);
Display.gfx_RectangleFilled(62,50,88,76,BLUE);
delay(500);
```

This code draw three filled rectangles with different colours as shown in the image below. The function

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

requires five parameters. The x1 and y1 parameters specify the top left corner coordinates of the rectangle. The x2 and y2 parameters specify the bottom right corner coordinates of the rectangle. The colour parameter specifies the colour of the filled rectangle

Output:



Draw Triangles

Hollow Triangles

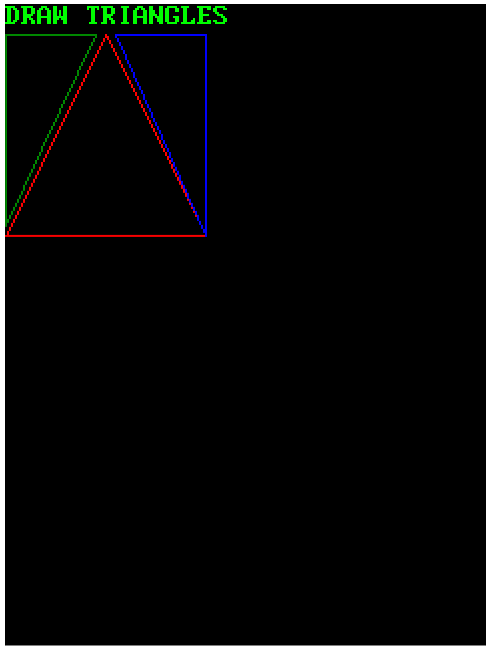
```
Display.putstr("DRAW TRIANGLES        \r");
Display.gfx_Triangle(0,115,50,15,100,115,RED);
delay(500);
Display.gfx_Triangle(0,15,0,110,45,15,GREEN);
delay(500);
Display.gfx_Triangle(55,15,100,15,100,115,BLUE);
delay(500);
```

This code draw three triangles with different colours as shown in the image below. The function

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

requires seven parameters. The x1 and y1 parameters specify the first vertex of the triangle. The x2 and y2 parameters specify the second vertex of the triangle. The x3 and y3 parameters specify the third vertex of the triangle. The colour parameter specifies the outline colour of the triangle.

Output:



Filled Triangles:

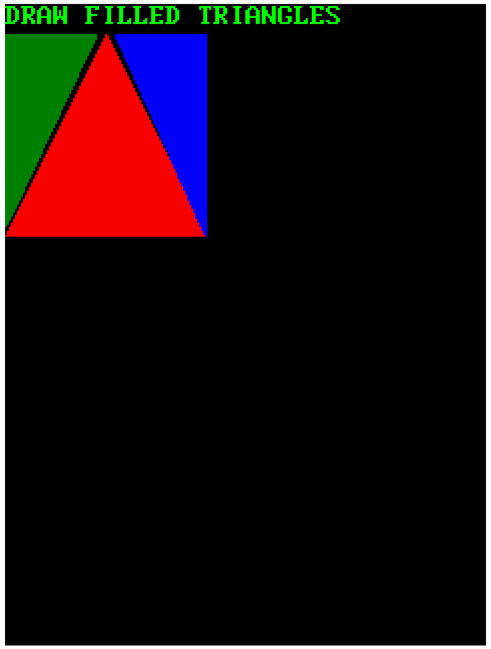
```
Display.putstr("DRAW FILLED TRIANGLES\r");
Display.gfx_TriangleFilled(0,115,50,15,100,115,RED);
delay(500);
Display.gfx_TriangleFilled(0,15,0,110,45,15,GREEN);
delay(500);
Display.gfx_TriangleFilled(55,15,100,15,100,115,BLUE);
delay(500);
```

This code draw three filled triangles with different colours as shown in the image below. The function

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

requires seven parameters. The x1 and y1 parameters specify the first vertex of the triangle. The x2 and y2 parameters specify the second vertex of the triangle. The x3 and y3 parameters specify the third vertex of the triangle. The colour parameter specifies the colour of the filled triangle.

Output:



```
DRAW FILLED TRIANGLES
```

Draw Polylines

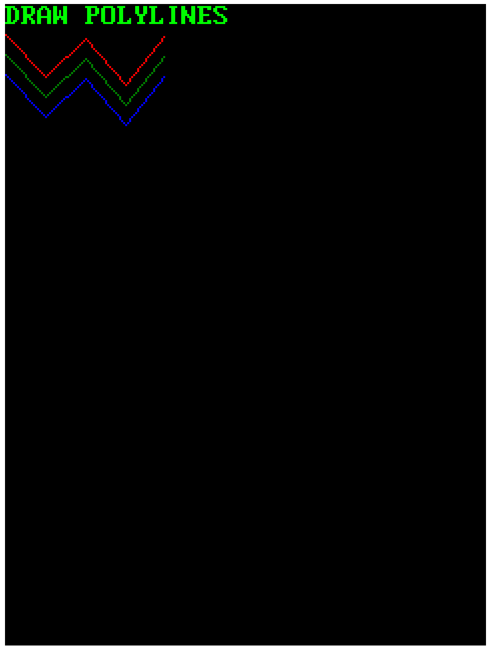
```
Display.putstr("DRAW POLYLINES        \r");
uint16_t RED_x[5] = {0,20,40,60,80};
uint16_t RED_y[5] = {15,36,17,40,15};
Display.gfx_Polyline(5,RED_x,RED_y,RED);
delay(500);
uint16_t GREEN_x[5] = {0,20,40,60,80};
uint16_t GREEN_y[5] = {25,46,27,50,25};
Display.gfx_Polyline(5,GREEN_x,GREEN_y,GREEN);
delay(500);
uint16_t BLUE_x[5] = {0,20,40,60,80};
uint16_t BLUE_y[5] = {35,56,37,60,35};
Display.gfx_Polyline(5,BLUE_x,BLUE_y,BLUE);
delay(500);
```

The “Draw Polyline” command plots lines between points specified by a pair of arrays using the specified colour. The lines may be tessellated with the “Line Pattern” command. The “Draw Polyline” command can be used to create complex raster graphics by loading the arrays from serial input or from MEDIA with very little code requirement. The function

gfx_Polyline (n, vx, vy, colour)

requires five parameters. The n parameter specifies the number of elements in the x and y arrays containing the vertices for the polyline. The vx parameter specifies the starting address of the array of elements for the x coordinates of the vertices. The vy parameter specifies the starting address of the array of elements for the y coordinates of the vertices. The colour parameter specifies the colour for the lines

Output:



Draw Polygons

Hollow Polygon

```
Display.putstr("DRAW POLYGON        \r");
uint16_t RED_x[7] = {50,100,100,50,0,0};
uint16_t RED_y[7] = {15,44,88,117,88,44};
Display.gfx_Polygon(6,RED_x,RED_y,RED);
delay(1000);
```

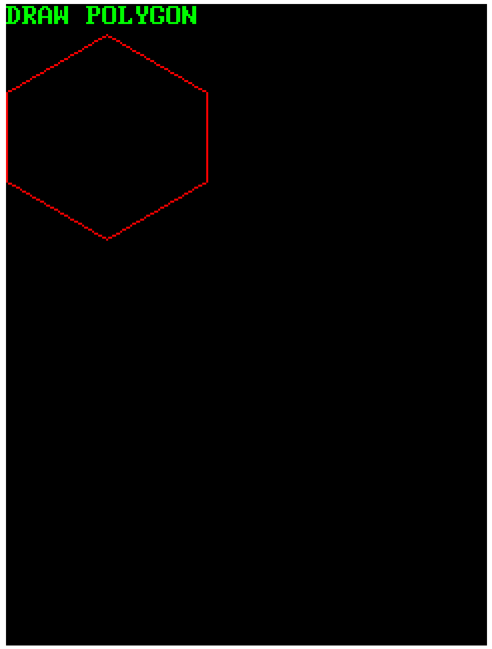
The “Draw Polygon” command plots lines between points specified by a pair of arrays using the specified colour. The last point is drawn back to the first point, completing the polygon. The lines may be tessellated with “Line Pattern” command. The “Draw Polygon” command can be used to create complex raster graphics by loading the arrays from serial input or from MEDIA with very little code requirement.

The function

gfx_Polygon (n, vx, vy, colour)

requires five parameters. The n parameter specifies the number of elements in the x and y arrays specifying the vertices for the polyline. The vx parameter specifies the starting address of the array of elements for the x coordinates of the vertices. The vy parameter specifies the starting address of the array of elements for the y coordinates of the vertices. The colour parameter specifies the colour for the polygons.

Output:



```
DRAW POLYGON
```

Filled Polygon

```
Display.putstr("DRAW FILLED POLYGON \r");
uint16_t BLUE_x[7] = {50,100,100,50,0,0};
uint16_t BLUE_y[7] = {15,44,88,117,88,44};
Display.gfx_PolygonFilled(6,BLUE_x,BLUE_y,BLUE);
delay(1000);
```

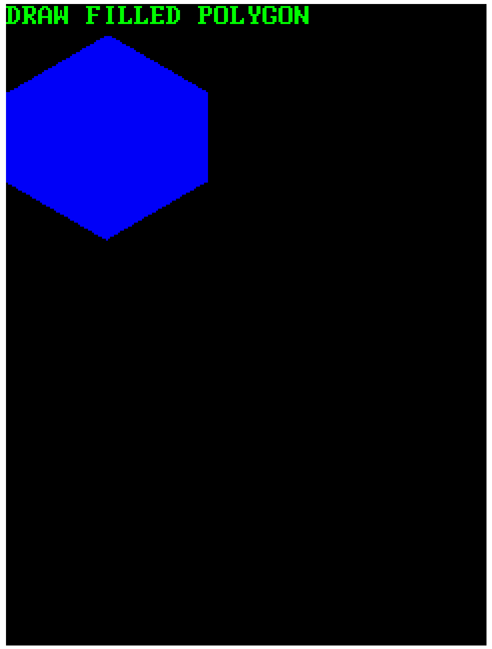
The "Draw Filled Polygon" command draws a solid polygon between specified vertices: x1, y1 x2, y2, ... , xn, yn using the specified colour. The last point is drawn back to the first point, completing the polygon. Vertices must be a minimum of 3 and can be specified in any fashion.

The function

gfx_PolygonFilled (n, vx, vy, colour)

requires five parameters. The n parameter specifies the number of elements in the x and y arrays specifying the vertices for the polyline. The vx parameter specifies the starting address of the array of elements for the x coordinates of the vertices. The vy parameter specifies the starting address of the array of elements for the y coordinates of the vertices. The colour parameter specifies the colour for the polygons.

Output:



Connect the 4D Display Module to the Arduino Host

Refer to the section **“Connect the Display Module to the Arduino Host”** of the application note **Serial Connection to an Arduino Host** for the following topics:

- Using the New 4D Arduino Adaptor Shield (Rev 2.00)
  - Definition of Jumpers and Headers
  - Default Jumper Settings
  - Change the Arduino Host Serial Port
  - Power the Arduino Host and the Display Separately
- Using the Old 4D Arduino Adaptor Shield (Rev 1)
- Connection Using Jumper Wires

Proprietary Information

Disclaimer of Warranties & Limitation of Liability