PlotterWriter Application

The PlotterWriter Application (PlotterWriterConsoleUI) is a console application for Windows written in C# for creating images using the HP 7475 pen plotter or another plotter that supports is command set. In this case, it was developed using a Houston Instrument ImageMaker plotter, which supports almost the entire command set, which are defined in HPGL, Hewlett-Packard Graphics Language. HPGL is defined in great detail in "HP 7475A Graphics Plotter - Interfacing and Programming Manual" (part no. 07475-90001), which can be found on eBay.

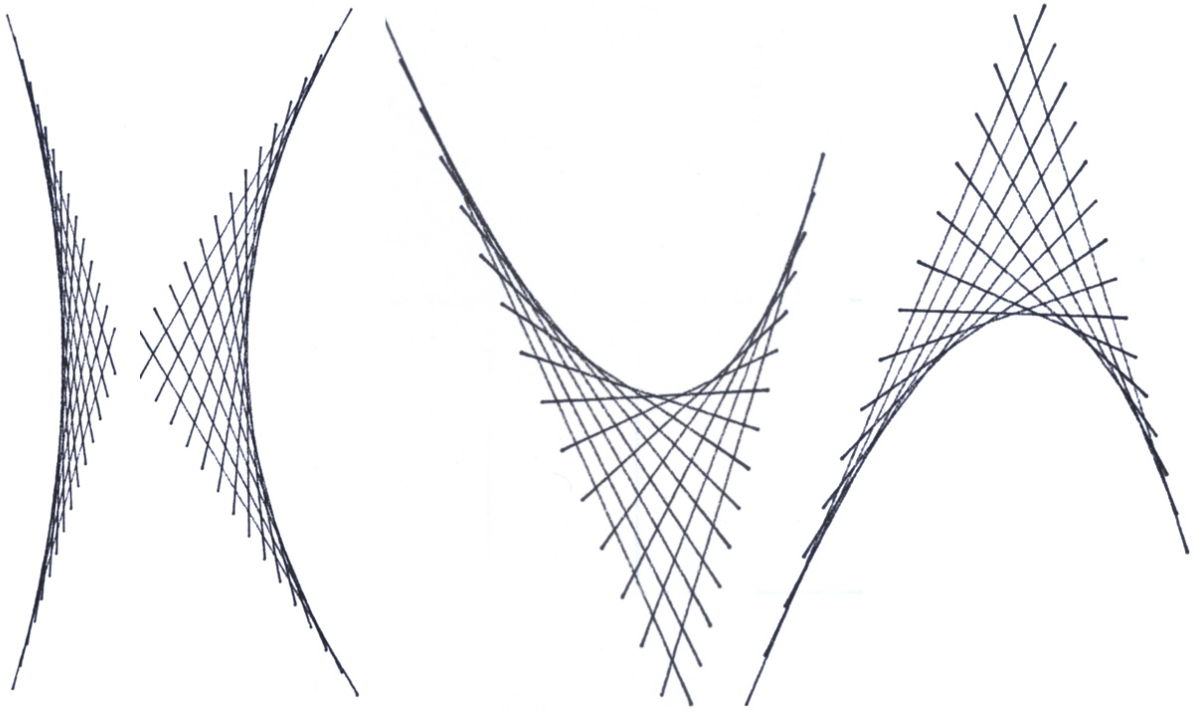
The solution is composed of several components:

- The entire solution contains 2 projects, only one of which builds to create the PlotterWriterConsoleUI executable. The non-building project is a container for the code that does all the real work:
    - **HPGL.cs** contains definitions for wrapper methods that generate HPGL strings for all the HPGL commands supported by the ImageMaker as well as generic methods for math and other essential operations.
    - **PlotterDriver.cs** is an entirely new code file that handles all I/O operations for both the serial and parallel ports, written entirely in C# using the .NET framework exclusively (the older version had a C++ DLL to handle I/O using Win32 API function calls. HPIB is not currently supported. Output to the serial port is multi-threaded to enable the main UI thread to operate without blocking while a plot is in progress. The development machine, which has no physical serial or parallel ports, uses a pair of USB cables, one for serial port communications to a DB9 connector, and the other for parallel port communications to a 50-pin Centronics connector.
    - **PlotterEngine.cs** is a code file that handles calling the methods in PlotterDriver.cs as well as sorting the plot shapes first by pen color and then by start positions. It also contains definitions for classes representing all the shapes defined in HPGL, e.g. circles, arcs, wedges, text strings, line collections, etc.
- The **PlotterDriver** component is written in native C++ to handle all communications over the serial and parallel ports. The development machine, which has no physical serial or parallel ports, uses a pair of USB cables, one for serial port communications to a DB9 connector, and the other for parallel port communications to a 50-pin Centronics connector. This project also contains the C++/CLI wrapper classes that connect the native C++ to the .NET code.
- The other project, **PlotterWriterConsoleUI**, also written entirely in C#, is the entry point for the solution, and drives the rest of the code. It contains all the necessary commands to allow the user to select output port, the number of pens (maximum of 8 as supported by the ImageMaker), and the shapes to be drawn. It also supports writing all the HPGL to output files with the .HPGL extension. These files can be viewed by applications such as HpglViewCern. The app also supports tracking progress as each segment of HPGL code is sent to the plotter and the ability to abort a plot in progress, clearing both the plotter buffer and any queued up plots.

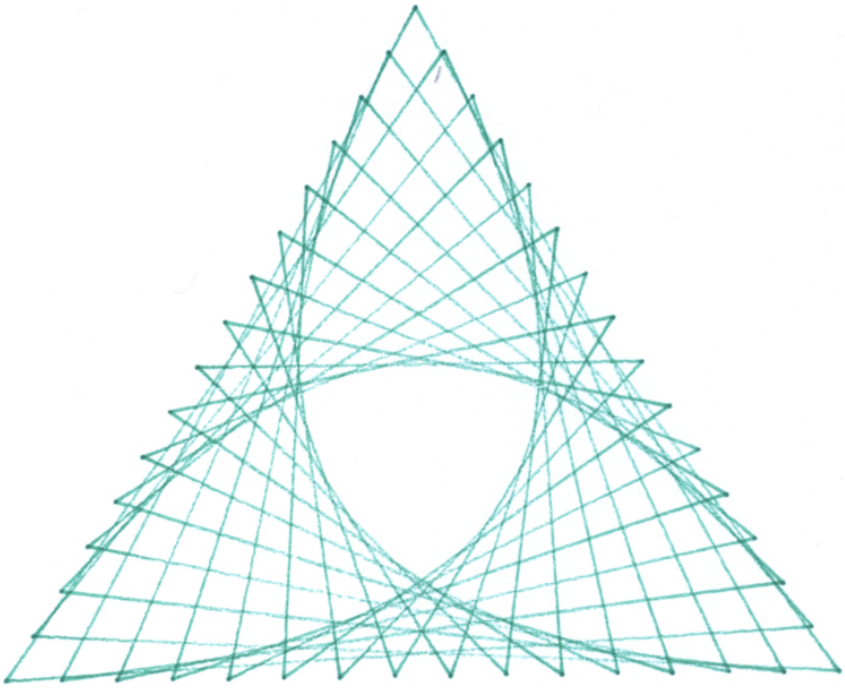
A video of the new version at work with my Houston Instrument ImageMaker plotter is available on YouTube.

Some examples of the defined shapes follow:

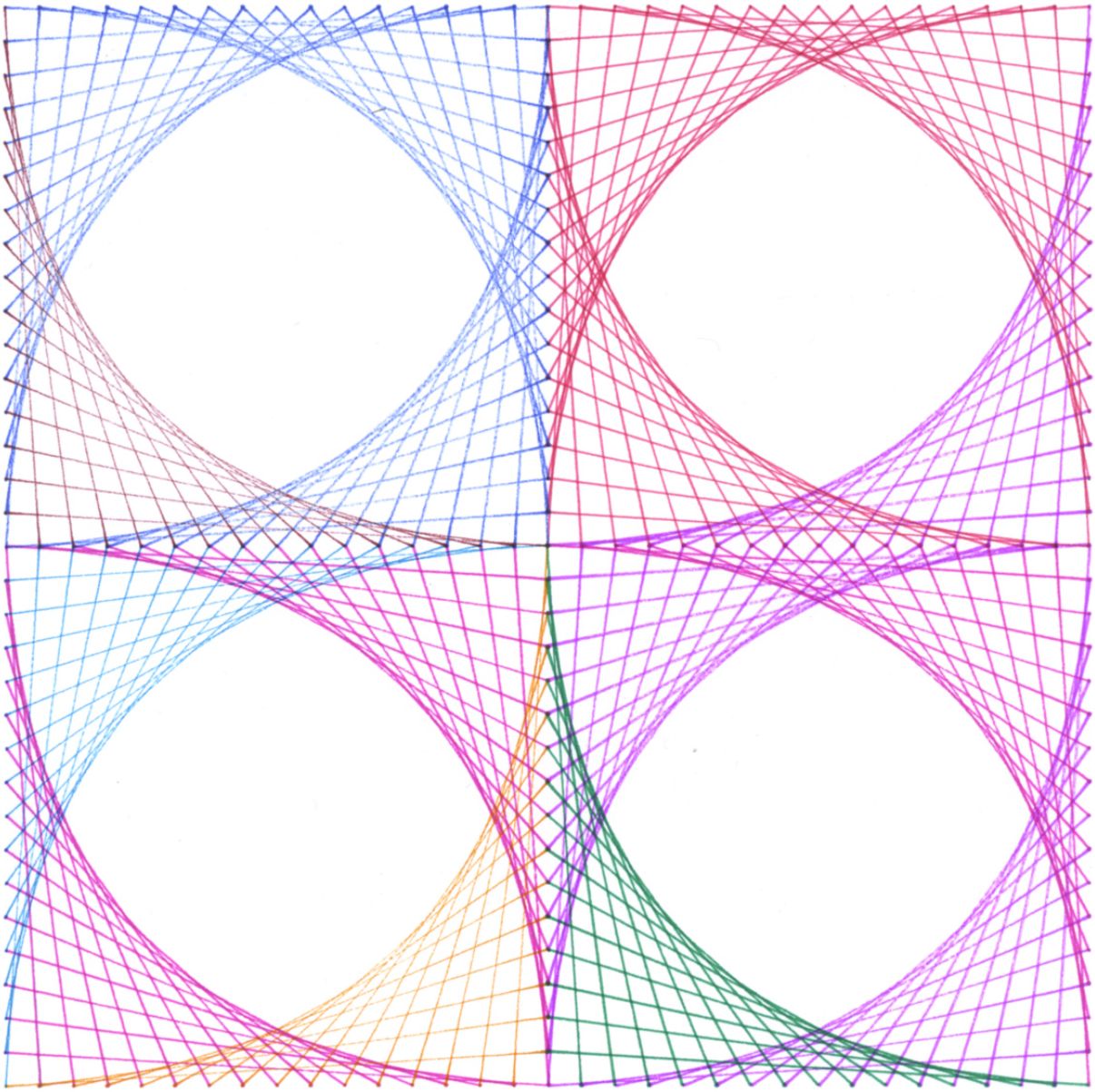
The first image shows 4 string-art curved-grid images, created by “Simple Triangle 3” in “String Art Presets”.



The green triangle was drawn by “Complex Triangle” in “String Art Presets” which drew 3 connected curved-grid images to form the equilateral triangle, in green ink.



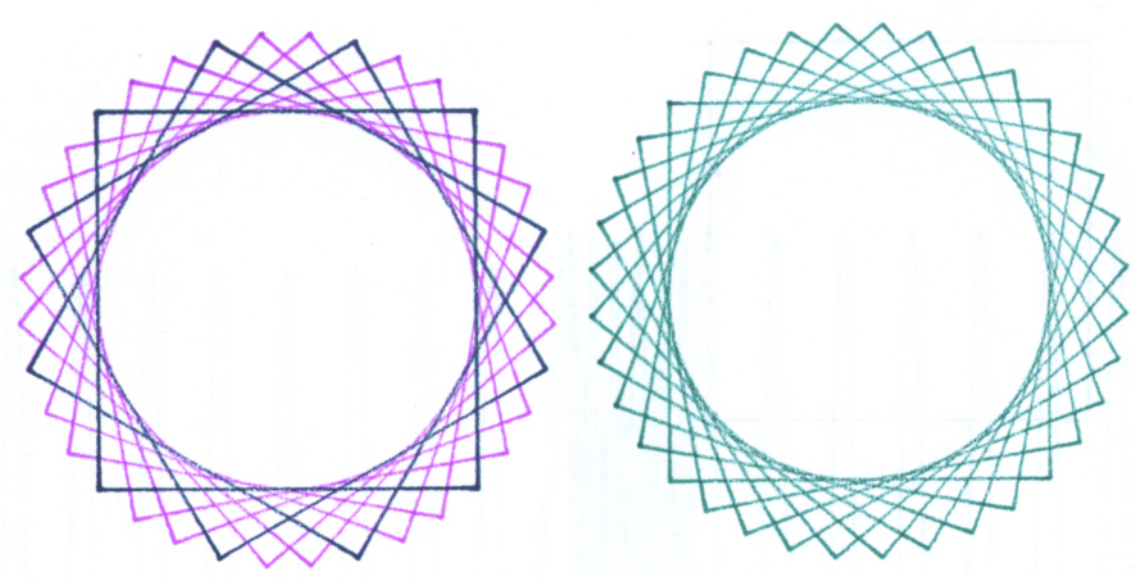
“Four Quadrants “ in “String Art Presets” used 8 different color pens randomly selected to draw this series of 16 connected curved-grid images. The image is based on a hand-drawn picture found on the Internet, having been drawn by hand by a very talented seventh-grade student.



The next 2 images were drawn by the “Rotated Square” option in “String Art Presets”. The image on the left is actually 2 images from 2 separate tests. The black ink shows the original square with 2 copies of it, rotated 30 degrees and 60 degrees. The purple ink shows the same square rotated 10 degrees per rotation.

The image on the right shows the same square rotated 10 degrees per step from 0 degrees to 80 degrees. Since the shape is a square, 9 steps are all that are needed to complete the image.

This code made use of the PolarToCartesian and CartesianToPolor methods to rotate the images.



The last 2 images are Lissajous curves, named after the French mathematician, Jules Antoine Lissajous, who studied them in depth long before computers or oscilloscopes made the process easier. A Lissajous curve is drawn with sine values in both x- and y-axis. By plotting sine values of differing wavelengths, these plots become more interesting.

The next image was drawn by the “Lissajous Pattern 1” option in “Lissajous Patterns Presets, using the method CPlotterShapes.PlotLissajousCurve. The test case uses 3 different pens for each of the separate wavelength combinations.

The last image was created while experimenting with other wavelength combinations, also using the CPlotterShapes.PlotLissajousCurve method.