Pica Graphics

Tom Gibson, January 21, 2018

USAGE

I've been using tcc to compile...

$ tcc \*.c -lm -o plot

Line, circle, and star objects are produced, plus an executable 'plot', and a few test functions.

Run it with ./plot, which produces a usage report. Notice the -gNAME option is required, and only four names (listed) are recognized. Simplest case is

$ ./plot -gstar

which produces a 17 point star. Experiment with the options to see what happens.

DESIGN

Two layers. The bottom Basic layer, pg, uses only integers on its API, and is hardwired to an 80x80 character matrix. It does the actual plotting and showing.

The upper later, I call it the palette layer, has floating point arguments where appropriate, and offers a user defined problem oriented palette. It maps problem oriented coordinates to pg coordinates, letting pg do all the hard work.

Basic Layer, pg...

The bottom layer, Pica Graphics, has prefix pg on its 3 files. Plots use only 4 ascii characters, space for background, and prime/period/semicolon for one or two dots in each pica cell. This is ALL integer. Plotting is done in a hardwired 80x80 matrix. Available routines are:

typedef struct { int x; int y; } pgPoint;

Declare points, e.g. pgPoint={30,50};

void blank();

initialize the space with all spaces.

void pgLine(pgPoint a, pgPoint b);

Draws a line from a to b inclusive.

void show();

Display the results so far. The array is printed to stdout, so the

drawing shows up in your command window.

void pgPointCircle(pgPoint center, int radius, pgPoint pts[], int npts);

Defines, but does not draw a circle of npts Points, returning the

results in the array pts. User must supply this array with

sufficent dimension >= npts. If npts is 20 or larger it looks like a

circle, hence the name. Used by pgCircle and pgStar.

void pgCircle(pgPoint center, int radius, int npts);

Draws a circle.

void pgStar(pgPoint center, int radius, int npts);

Draws a star with npts points.

void pgPlot(int x, int y);

Mainly used internally. Puts a mark at position x,y. Does nothing if

mark is out of bounds. In bounds is 0 <= x < 80, 0 <= y < 80. Out of

bounds is not an error.

Basic Usage

First call blank() to initialize the fixed 80x80 drawing matrix. Then define pgPoints. Note its syntax is declarative, not function. Then call objects using the points. pgPlot itself is not really an object, but to put down a single mark it can be used as though it were. Finally call show(). Save and compile and run, and if all is well your picture shows up in the command window.

You can add printf's but they do not show in the picture. Scroll up to see them. Use -D to see debug prints. Scroll up to see them, too, or better redirect to an err file. My frequent use is...

./plot -gtest -D 2>err

test is where I put experimental drawings. Clone pgPlot's main function to put together your own main.

Palette Layer

Define a palette with

void palette(float minx, float miny, float maxx, float maxy )

Define points with

typedef struct { float x; float y; } Point;

The object routines are:

void point(float x, float y);

void line(Point, Point);

void circle(Point center, float radius, int npts);

void star(Point center, float radius, int npts);

Palette Usage

First call: palette. You can use ints or floats in the four arguments. Minx/y defines the lower left corner, maxx/y the upper right. For example...

palette( -1, -1, 1, 1 ) ;

gives you a palette of size 2x2, four quadrants, point at 0,0 will plot right in the middle. Palette calls blank() for you.

Points are defined like this…

Point a = { -0.3, +0.7 };

Draw using the object routines and defined points. Finally call show();

Testing

A main program in pgTest.c selectively calls one of four test routines (-g option, required). I wrote test cases in that file.

Sample Drawings

For now pgTest.c is the best example.