CPlotter Public Interface / API

v1.5, Feb. 2016

Horst-W. Radners

1 General

The *CPlotter* library provides a basic, **unified** interface to different graphics formats for simple (passive, file-based) 2D-drawings in C on unix-like operating systems. For an overview of implemented graphic primitives, see Fig. 6 on page 7.

1.1 Graphics File Formats

Each of the different backends/implementations generates a file of one distinct graphics format, currently are implemented:

file suffix	**graphics format**
eps	Encapsulated PostScript vector graphics (PS-Adobe-3.0 EPSF-3.0)
png	Portable Network Graphics true-color raster image (PNG 1.2)
svg	Scalable Vector Graphics (SVG 1.1)

Table 1: Graphics file formats

The graphics format to use is determined by the file-suffix given in the `plotfilename` to **`CPLT_init_graphics`**`()`, see below.

Please note, since *CPlotter* utilizes the GD-library (see https://libgd.github.io/) to create PNG files, it has to be linked with *libgd* (at least version 2.0 with *FreeType*) and the resp. include files have to be available too if re-compiling is desired (on most Linux distributions this requires installation of *libgd* and *libgd-devel* or similar).

The EPS and SVG formats are stand-alone text files, their generation by *CPlotter* is self-contained, hence independent of any external libraries. As vector-graphics, EPS and SVG formats are resolution-independent, i.e. they may be scaled without loss of quality. Because of this free scalability these vector formats are recommended for including *CPlotter*-generated graphics in other (word processing) documents.

1.2 Abstract Data Type

Since the graphics context is stored in a client-variable and all backend functions are reentrant, *CPlotter* serves as a *first-class ADT*, hence multiple instances may coexist. Therefore a client program may hold open various plots (of may be different graphics formats) at the same time. The client must not exploit any knowledge about the internal structure of the graphics context, but interact with *CPlotter* solely by calls to the interface/API functions declared below.

1.3 Coordinate System

The unified coordinate system used by *CPlotter* has `(0,0)` at its lower left, `x` increases to the right and `y` increases upwards. All coordinates are in native units of the resp. backend format (pix, pt). Although *CPlotter* uses **`float`** for the coordinates, they may be rounded to **`int`** dependent on the backend format.

1.4 Introductory Example

As a minimal usage example, the following C program generates a SVG graphics file `myplotfile.svg` as shown in Fig. 1, which can be viewed with any modern browser:

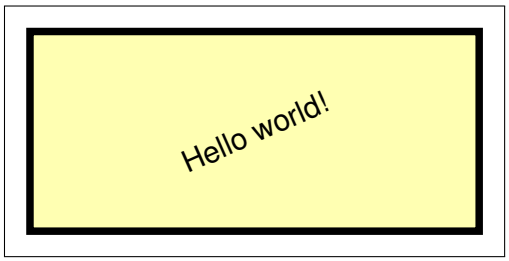

Figure 1: Output of following example program

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "CPlotter.h"

int main() {

   /* plot area width, height and margin [pix] */
   enum { pwd = 200, pht = 100, mrg = 10 };
   const float RAD2DEG = 57.29578;    /* factor: radiants ==> degrees */

   CPLT_gc_t gc;                /* stores graphics context */
   CPLT_point_t points[] = {    /* rectangle coords inside margin */
      {mrg, mrg}, {pwd - mrg, mrg}, {pwd - mrg, pht - mrg}, {mrg, pht - mrg} };

   /* initialize graphics context, open plotfile */
   if ((gc = CPLT_init_graphics(pwd, pht, "myplotfile.svg")) == NULL) {
      fprintf(stderr, " *** ERR: Can't initialize graphics context!\n");
      return 1;
   }

   CPLT_set_color(gc, 1., 1., 0.7);         /* fill with light-yellow */
   CPLT_draw_filledPolygon(gc, 4, points);  /* draw a colored rectangle */

   CPLT_set_linewidth(gc, 3.);              /* thicker ... */
   CPLT_set_color(gc, 0., 0., 0.);          /* black ... */
   CPLT_draw_polygon(gc, 4, points);        /* border */

   /* draw centered text rotated to diagonal */
   CPLT_draw_text(gc, 0.5 * pwd, 0.5 * pht, "c", atan2f(pht - 2 * mrg,
                  pwd - 2 * mrg) * RAD2DEG, "Hello world!");

   /* finish graphics, close plotfile, destroy graphics context */
   CPLT_finish_graphics(gc);

   return 0;
}
```

Example drawing program using *CPlotter* API

2 API

The *CPlotter* public API consists of 3 data type definitions and 14 functions.

2.1 Data Types

CPLT_gc_t

CPLT_gc_t is the type clients use as graphics context, i.e. the ADT object.
See **CPLT_init_graphics()**.

```
typedef struct CPLT_gctx* CPLT_gc_t;
```

CPLT_point_t

The type for 2D-points with x/y-coords used for array-parameters.

```
typedef struct {
   float x, y;
} CPLT_point_t;
```

CPLT_lnstyle_t

Enumerated dash patterns for linestyle, see **CPLT_set_linestyle()** and Fig. 5 on page 6.

```
typedef enum {
   CPLT_SolidLine,
   CPLT_DashLine,
   CPLT_DotLine,
   CPLT_DashDotLine,
   CPLT_DashDotDotLine
} CPLT_lnstyle_t;
```

2.2 Functions

2.2.1 Initializing and Finalizing

To starting a drawing **CPLT_init_graphics()** must be called first, each other function needs as its first parameter the graphics context pointer returned by this initialization function. Respectively, to end a drawing, the last function called must be **CPLT_finish_graphics()**.

CPLT_init_graphics

Initializes graphics of pwidth x pheight [pix] in graphics file plotfilename, the graphics-format specific suffix (.eps, .svg, .png, see Table 1 on page 1) must be included and determines the graphics format/backend used. Returns graphics context pointer or NULL on error.

```
CPLT_gc_t CPLT_init_graphics(const unsigned int pwidth,
                             const unsigned int pheight,
                             char *plotfilename);
```

CPLT_finish_graphics

Finishes graphics, closes plotfile, destroys graphics context. The plotfile is not usable by other programs until this function is finally called, since not till then all internal buffers are flushed.

```
void CPLT_finish_graphics(CPLT_gc_t gc);
```

2.2.2 Drawing

CPLT_draw_polyline

Plots line through numpts 2D-points at given x/y-pairs in array points. Draws line with current color and linewidth/style.

```
void CPLT_draw_polyline(CPLT_gc_t gc,
                        const int numpts, CPLT_point_t points[]);
```

CPLT_draw_polygon

Plots (automatically closed) 2D-polygon with `numpts` points at given x/y-pairs in array `points`. Draws outline of the polygon with current color and linewidth/style.

```
void CPLT_draw_polygon(CPLT_gc_t gc,
                       const int numpts, CPLT_point_t points[]);
```

CPLT_draw_filledPolygon

Plots (automatically closed) 2D-polygon with `numpts` points at given x/y-pairs in array `points`. Fills and strokes the polygon with current color.

```
void CPLT_draw_filledPolygon(CPLT_gc_t gc,
                             const int numpts, CPLT_point_t points[]);
```

CPLT_draw_arc

Plots partial circle at center given by `cx`/`cy` with the specified `radius`. The arc begins at the position in degrees specified by angle `start` and ends at the position specified by angle `end`. A full circle can be drawn by beginning from `start=0` degrees and ending at `end=360` degrees. Both angles turn counterclockwise, i.e. mathematically positive. Draws outline of the arc with current color and linewidth/style.

```
void CPLT_draw_arc(CPLT_gc_t gc,
                   const float cx, const float cy, const float radius,
                   const float start, const float end);
```

CPLT_draw_filledArc

Plots partial circle at center given by `cx`/`cy` with the specified `radius`. The arc begins at the position in degrees specified by angle `start` and ends at the position specified by angle `end`. A full circle can be drawn by beginning from `start=0` degrees and ending at `end=360` degrees. Both angles turn counterclockwise, i.e. mathematically positive. Fills and strokes the arc/"pie slice" with current color.

```
void CPLT_draw_filledArc(CPLT_gc_t gc,
                         const float cx, const float cy, const float radius,
                         const float start, const float end);
```

CPLT_draw_curve

Plots cubic Bézier curve segment defined by (always!) four 2D-points given as x/y-pairs in array `points`. The four points act as control points for the shape of the curve, see Fig. 2 and the following quotation.

> "The four points define the shape of the curve geometrically. The curve starts at `points[0]`, it is tangent to the line from `points[0]` to `points[1]` at that point, and it leaves the point in that direction. The curve ends at `points[3]`, it is tangent to the line from `points[2]` to `points[3]` at that point, and it approaches the point from that direction. The lengths of the lines `points[0]` to `points[1]` and `points[2]` to `points[3]` represent, in a sense, the *velocity* of the path at the endpoints. The curve is always entirely enclosed by the convex quadrilateral defined by the four points."
>
> (modified) from:
> *Adobe Systems Inc., PostScript Language Reference Manual, Addison-Wesley*

Draws line with current color and linewidth/style.

```
void CPLT_draw_curve(CPLT_gc_t gc, CPLT_point_t points[]);
```

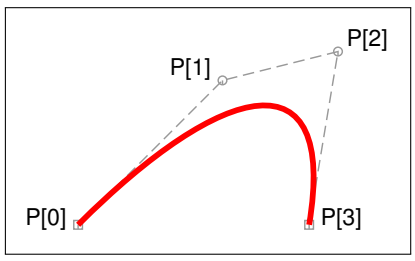

Figure 2: Bézier curve segment with control points

CPLT_draw_marker

Plots marker centered at cx/cy of specified width and height wd [pix], with current linewidth and color. symbol [0-7] enumerates the marker's form, see Fig. 3.

```
void CPLT_draw_marker(CPLT_gc_t gc,
                      const float cx, const float cy,
                      const int wd, const int symbol);
```

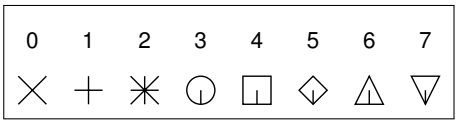

Figure 3: Enumerated marker symbols

CPLT_draw_text

Plots Latin-1/ISO-8859-1 encoded text of current fontsize at angle degrees with current color (although the current linewidth is ignored, the actual strokewidth used is determined by the current fontsize, see **CPLT_set_fontsize()**). angle=0 degrees results in horizontal text, angle turns counterclockwise, i.e. mathematically positive. The rotation occurs about the anchored reference point of the text.

anchor ["sw", "s", "se", "w", "c", "e", "nw", "n", "ne"] sets the reference point of the text enclosing rectangle positioned at x/y, see Fig. 4. For example, if anchor="sw", the lower-left corner of text is positioned at x/y, if anchor="c", the text is centered at x/y.

```
void CPLT_draw_text(CPLT_gc_t gc,
                    const float x, const float y, char *anchor,
                    float angle, char *text);
```

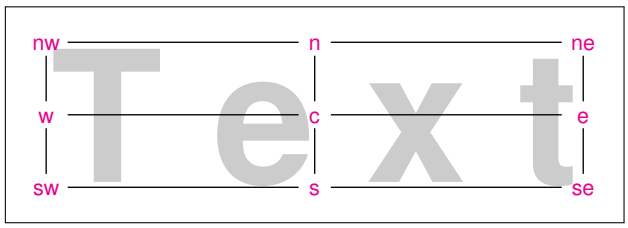

Figure 4: Anchor positions for text

2.2.3 Attributes Setting

All attributes (color, linewidth, linestyle, fontsize) of the drawing items are persistent, i.e. they keep their values until reset by the respective CPLT_set_*() function (again).

CPLT_set_fontsize

Sets current fontsize [pix] for **CPLT_draw_text()**. (Preset: fontsize=12.0)

```
void CPLT_set_fontsize(CPLT_gc_t gc,
                       const float fontsize);
```

CPLT_set_color

Sets current color of RGB values [0,1]. (Preset: r=0., g=0., b=0., i.e. *black*)

```
void CPLT_set_color(CPLT_gc_t gc,
                    float r, float g, float b);
```

CPLT_set_linewidth

Sets current linewidth w [pix]. (Preset: w=1.0)

```
void CPLT_set_linewidth(CPLT_gc_t gc,
                        const float w);
```

CPLT_set_linestyle

Sets current linestyle `s` [enumeration], see Fig. 5. (Preset: `s=CPLT_SolidLine`)

```
void CPLT_set_linestyle(CPLT_gc_t gc,
                        const CPLT_lnstyle_t s);
```

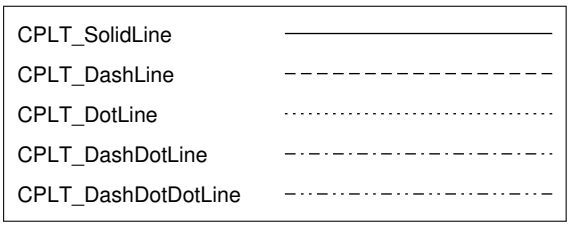

Figure 5: Line styles

3 Miscellaneous

3.1 Known deficiencies

The font-family/-face used on drawing text is backend-specific and constant, hence cannot be changed by the API.
As curves only circular arcs and cubic Bézier curve segments are available, other curves (e.g. ellipses or splines of other orders) are not implemented. Cubic Bézier curve segments can be outlined only, filling of (closed) curves is not provided.
Transparency (alpha channel) is not supported.

3.2 Author and Copyright

3.3 License

4 Demonstration Output

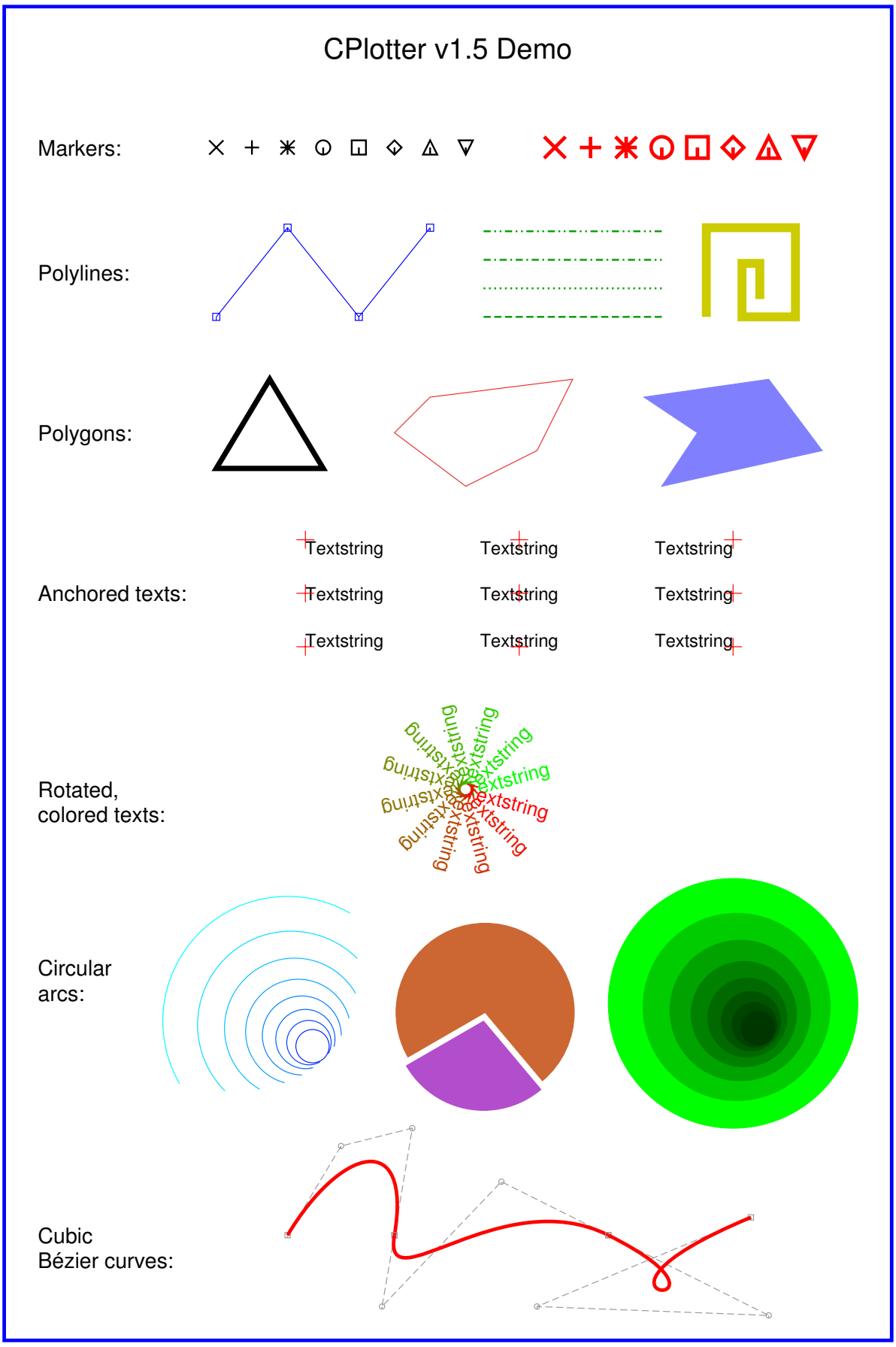

Figure 6: CPlotter demonstration output