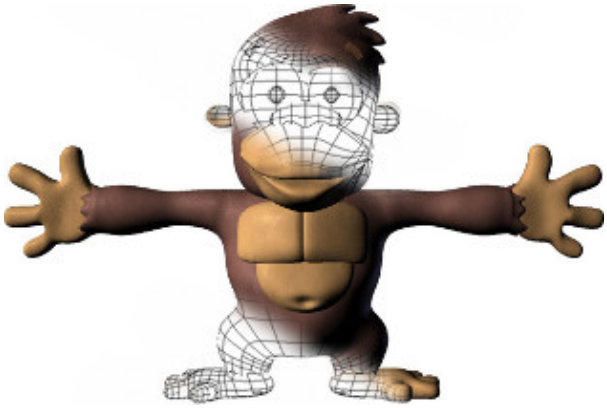
B1 - C Graphical Programming

B-MUL-100

Bootstrap My Screensaver

Introduction to shapes drawing and graphical effects



1.1

Preambule

In order to be able to do that bootstrap correctly you should have entirely finished the first initiation bootstrap.
The exercices asked here assume that you already have managed to put a pixel in a window.

Drawing a line

A line, a segment in fact, seems to be the easiest mathematical entity after a point. In computer graphics, it is not.

A line is a link between two points that we can name *A* and *B*.
Remember that for any point *P* on the line, we have `P.y = a * P.x + b` with `P.x` and `P.y` respectively the the x-coordinate and y-coordinate of the point *P*.

We can calculate *a* and *b*:

- `a = (A.y - B.y)/ (A.x - B.x)`
- `b = A.y - a * A.x`.

Now, implement a function that draws a line between two points given as parameters. It should have the following prototype.

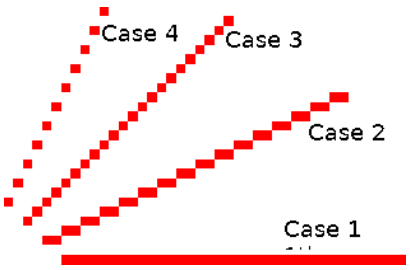
```
int draw_line(framebuffer_t *framebuffer, sfVector2i point_a, sfVector2i point_b);
```

Other properties of the line (e.g. color or thickness) might be given as parameters.

It ***must*** work with any pair of points

Intermediate result

At this point, you might end up with something like this :



We have 4 cases:

- case 1: the line is horizontal and the y-coordinate is constant.
- case 2: because we are working with integers, pixels might have the same y-coordinate.
- case 3: all the pixels have different x- and y-coordinates.
- case 4: because we are working with integers, pixels might have the same x-coordinate.

Is there any other case?

How to fix the case n°4?

So far, we assumed that any point *P* on the line such that `P.y = a * P.x + b`, meaning that we calculate *y* for each *x*.
We can also consider *P* such that `P.x = (P.y - b)/ a`, meaning that we calculate *x* for each *y*.

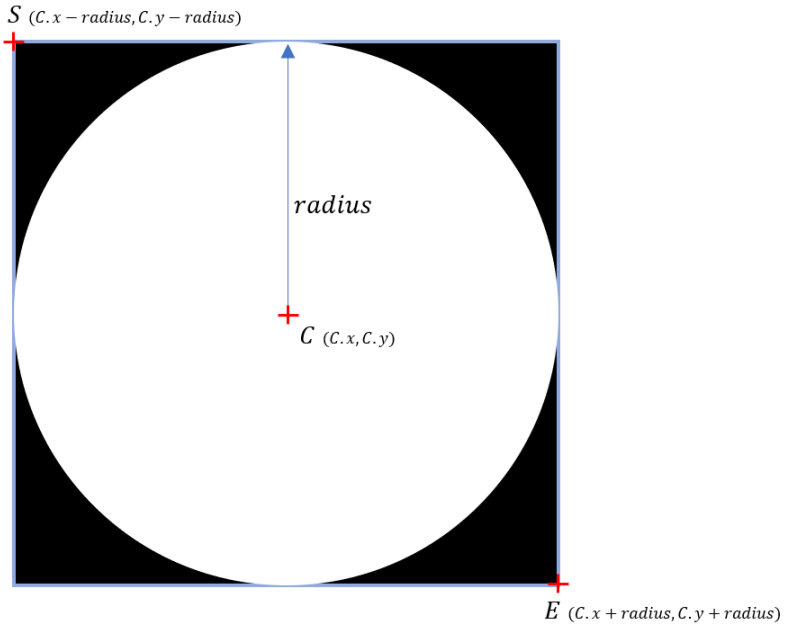
Now fix your `my_draw_line` function to handle the case n°4.

It's a good start, but the most vertical ones are rather cruddy, don't you think? Do you know why? Can you fix the problem?

DRAWING A CIRCLE

Drawing a circle is not as complicated as drawing a line.

We can assume that any circle with center *C (C.x, C.y)* and radius *r* is inscribed in a rectangle as shown in the following picture.



Any point *P* with the coordinates *(x, y)*, such that `((P.x - C.x)^2)+ ((P.y - C.y)^2)<= r^2` belongs to the circle.
Now, implement a function that draws a filled circle given its radius the coordinates of its center. It should have the following prototype.

```
int draw_circle(framebuffer_t *framebuffer, sfVector2i center, int radius);
```

What would you change if you only wanted to draw the outline of the circle?

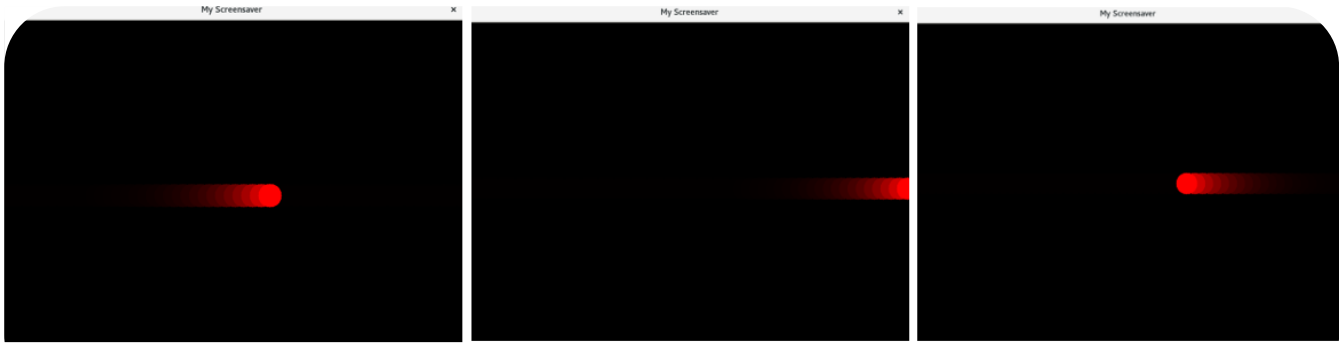
Make it move

Start moving a circle horizontally. It must change its direction as soon as it touches the border of the screen.

To do so, you can declare and initialize two variables related to the circle to be drawn:

- a `sfVector2i` for the coordinates of the center.
- an integer for the radius.

Then, you can change the value of the x-coordinate of the center before drawing your circle at each iteration of the game loop.
Remember to clear your framebuffer before drawing any new circle. Clearing the framebuffer means filling it with black pixels.



The circle moves at a constant speed

How can you control the speed of your circle ?

Move it smoothlier

Using the *sin* function, you can slow down the circle as it approaches the border of the screen.

You can do something like this:

```
// The variables you need
sfVector2i  center;
int     radius;
float       a;

// Their affectations in the game loop
radius = 20;
center.x = WIDTH / 2 + WIDTH / 2 * sin(a)
center.y = HEIGHT / 2;
a += 0.03;
```

$sin(a)$ is always between 1 and -1. We use *WIDTH / 2 + WIDTH / 2* sin(a)* to stretch $[-1; 1]$ to $[0; WIDTH]$

You can clear your framebuffer with partially-transparent black pixels to get a speed effect like in the following images.

The circle slows down as it approaches a border

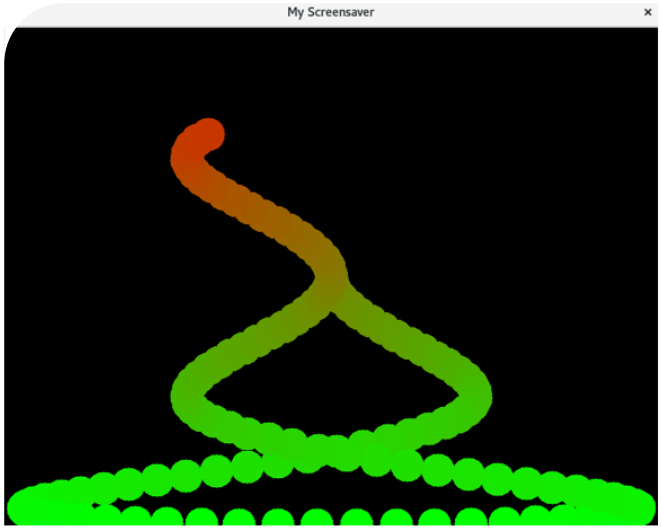
Now, you can apply the same formula to `center.y`.

MATHEMATICS ARE COOL

Using mathematical functions like *sin* or *cos* can create beautiful curves.

Instead of doing `WIDTH / 2 + WIDTH / 2 * sin(a)`, try with `WIDTH / 2 + WIDTH / 2 * sin(a)* sin(a*2)`.
You can apply it either to `center.x` or to `center.y`, or to both.
Try other formulas using *sin* and *cos* and see the results.

Now, you can complete your first animation by varying the color of the circle depending on `center.y`.



Without clearing the framebuffer

Clearing the framebuffer

You can add several circles.