Reflection

Computing optimal borders for a print

Optimal for myself!

Serge Émond

23th August 2014

1 Introduction

I want to print some of my photos, and to frame them using a simple mat (no framing).

To do that, I need to choose a border size, and to center it "optically" so the image doesn't give the impression of sinking into the frame. I also require the top border to be at least as big as the side borders.

Some people[1] use the *golden ratio* (ϕ) as a multiplicator to the width and height of the size of the mat's window. I personnally find it produces borders that are way too big.

Others[2] use ϕ as a multiplicator to the *area* of the print surface to compute the surface of the mat.

It gives a size that I like, however it also means the shortest side has smaller borders, which is visually ugly if this happens to be a landscape.

I could simply use the longest side for all sides, but then I would have other problems:

- "visually centering" the window would still give a smaller top border than the edges,
- and on an image with a strong difference between the two dimensions, this could produces borders too big.

So I decided to:

- use the golden ratio to compute the area of the mat size;
- keep the resulting border surface constant;
- compute the 4 borders so that the top equals the side borders, and the bottom is bigger accorging to the most common vertical centering rules.

[1] `http://kombat.org/FrameMaking/step1.html` uses ϕ to multiply the longest side and use that border for both sides

[2] `http://www.phimatrix.com/matte-frame-golden-ratio/`

2 Optical centering

To center the print window on a mat, the most common technique (actually the only one I found aside "move the picture a big higher") is:

1. align the top left corner of the print with the top left corner of the mat,
2. draw a vertical line at the center of the remaining space on the right to produce the left and right borders,
3. draw an horizontal line at the center of the remaining space on the bottom to produce the top and left borders before correction,
4. draw a line from the bottom left corner of the print to a point placed on the absolute right side of the mat, and the lower left line,
5. move the picture so the bottom right corner alligns with the diagonal line and the rightmost vertical line.

Visually, you get something like the figure 2.1.

On that figure, b_l, b_r, b'_t and b'_b represent the borders around a perfectly centered print. Normally we want the left and right borders to be the same size, and the top and bottom borders too.

So we can define $b_x = b_l = b_r$ as the size for the borders along the X axis, and $b_y = b'_t = b'_b$ for the top and bottom borders.

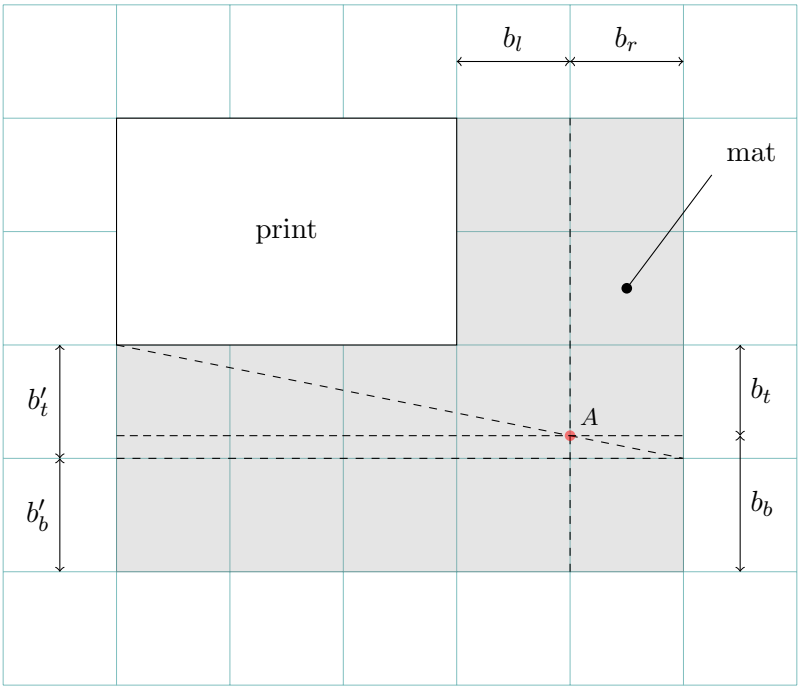
b_t and b_b represent the top and bottom border of an "optically centered", obtained by sliding the print so its bottom right corner is set on point A. Their value is:

$$b_t = b'_t - A$$
$$b_b = b'_b + A$$

Let say that (w_x, w_y) are the width and height of the mat's window (or print size), and (w_x, w_y) the dimensions of the mat itself.

It can be shown that the amount of upward shift, A, is:

Figure 2.1: Centering a print on a mat



$$A = \frac{b_x b_y}{m_x} = \frac{b_x b_y}{w_x + 2b_x}$$

Personnally, what I want is to find the proper border sizes to that $b_x = b_y = b$, and $b_b = b + A$.

2.1 Golden Ratio

Let the golden ratio be ϕ, because I don't intend on typing "golden ratio" more than I have to.

I want the mat's surface to be ϕ the mat window's surface:

$$m_x m_y = \phi w_x w_y$$

3 Finding the border sizes

From a bit of trigonometry, we have

$$m_x = w_x + 2b \tag{3.1}$$

$$m_y = w_y + 2b + A = w_y + 2b + \frac{b^2}{mx - b} \tag{3.2}$$

$$w_x = m_x - 2b \tag{3.3}$$

$$w_y = m_y - 2b - A = m_y - 2b - \frac{b^2}{mx + b} \tag{3.4}$$

The golden ratio of the area gives us:

$$0 = \phi w_x w_y - m_x m_y \tag{3.5}$$

So if we mix those to compute b from the window's dimensions:

$$0 = 6b^3 + (7w_x + 2w_y)b^2 + (2w_x^2 + (3 - \phi)w_x w_y)b + (1 - \phi)w_x^2 w_y \tag{3.6}$$

And to compute b from the mat's dimensions:

$$0 = -2\phi b^3 + (5m_x + 2m_y)\phi b^2 + (m_y - 2m_x\phi + 3m_y\phi)m_x b + (\phi - 1)m_x^2 m_y \tag{3.7}$$

4 Implementing

Solving symbolically is not really practical, producing way too much complexity for not much.

In the methods presented in table 4.1, the methods producing the best precision should be `mpmath`, because it's an arbitrary precision library, and `sympy` because it keeps everything symbolically, and finally evaluates using `mpmath`.

The least precise method should be `scipy` because `fsolve` is very general and doesn't even guarantee a proper result.

The fastest should be `numpy` because it uses the CPU's floating point.

Table 4.1 shows the processing times for a 9x6 print.

Practically, the precision has no impact since any of these methods should produce way more than what I can achieve while cutting a mat sheet. Thus only speed has a meaning in this case.

Pure processing time should point to `numpy`, however if we account for python's startup time, `numpy` takes on average 0.11s on my computer to run, while `mpmath` (with and without `gmpy`) takes 0.06s.

Thus my final choice is `mpmath`. It also happens to be much simpler (at least in virtual environments) and faster to install than `numpy`.

Table 4.1: Solving times, based on 1000 iterations (except sympy, which is based on 10)

Library	time / iteration	b	b_b
numpy	0.000171954s	0.956913776241981	1.04887841465701
scipy	0.000365011s	0.956913776241981	1.04887841465701
mpmath (gmpy)	0.003560504s	0.956913776241981	1.04887841465701
mpmath	0.005906222s	0.956913776241981	1.04887841465701
sympy (gmpy)	1.134872603s	0.956913776241981	1.04887841465701
sympy	1.259345698s	0.956913776241981	1.04887841465701

5 Examples

Using `mpmath` with the python code from figure 5, for a 9x6 print size, we obtain:

$$b = 0.95691$$
$$b_{bot} = 1.0489$$

Assuming the lengths are in centimeters, you can find representation of this at scale in figure 5.2.

If we want to compute the border area so we have the *lengths* (width and height) as $m_{(x|h)} = \phi \cdot w_{(x|h)}$, we use $\phi' = \phi^2$ and obtain the following borders, represented in figure 5.3:

$$b = 2.1285$$
$$b_{bot} = 2.5356$$

Figure 5.1: A simple example of numerically solving b from equation 3.6

```
from mpmath import mp

mp.dps = 5

wx = mp.mpf('9'); wy = mp.mpf('6')
phi = (1 + mp.sqrt(5)) / 2
coeffs = [
    6,
    7 * wx + 2 * wy,
    wx * ((3 - phi)*wy + 2*wx),
    (1 - phi) * mp.power(wx, 2) * wy
]

roots = mp.polyroots(coeffs, 15)
b = [root for root in roots if mp.im(root) == mp.mpf(0)][0]
bottom = b + mp.power(b, 2) / (wx + b)

print("Print dimensions: {wx!s} x {wy!s}\n"
      "Resulting border: {b!s}\n"
      "Resulting bottom border: {bottom!s}"
      .format(wx=wx, wy=wy, b=b, bottom=bottom))
```

Figure 5.2: Result for a 9x6cm print using $m_a = \phi w_a$

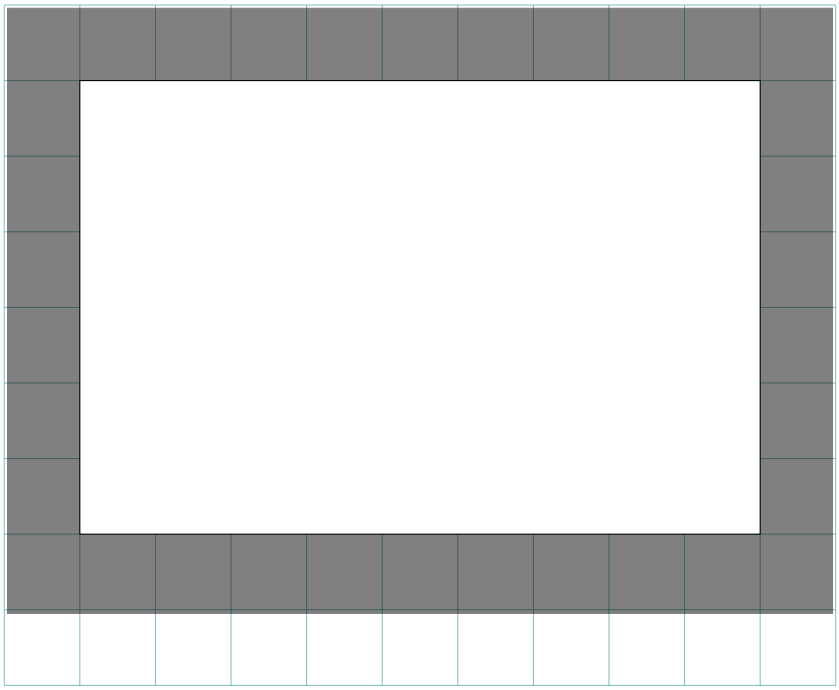


Figure 5.3: Result for a 9x6cm print using $m_a = \phi^2 w_a$