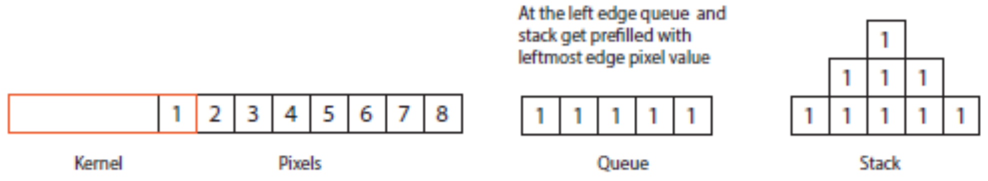
# Quasimondo

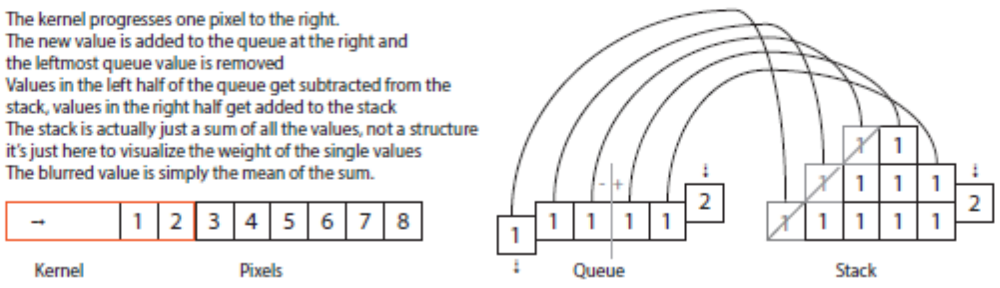
Mario Klingemann, Artist

Stack Blur Algorithm by Mario Klingemann

At the left edge queue and stack get prefilled with leftmost edge pixel value

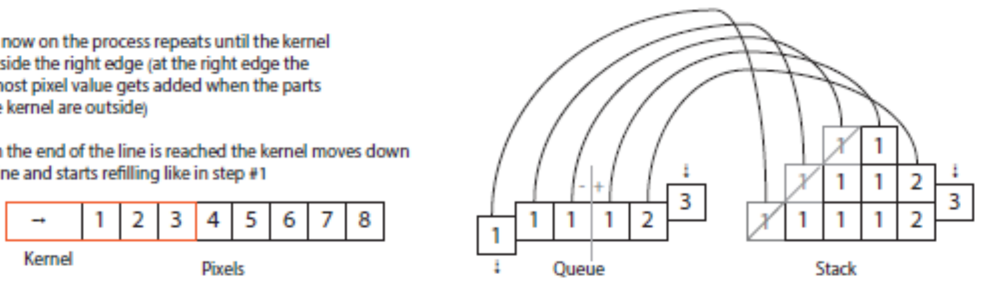


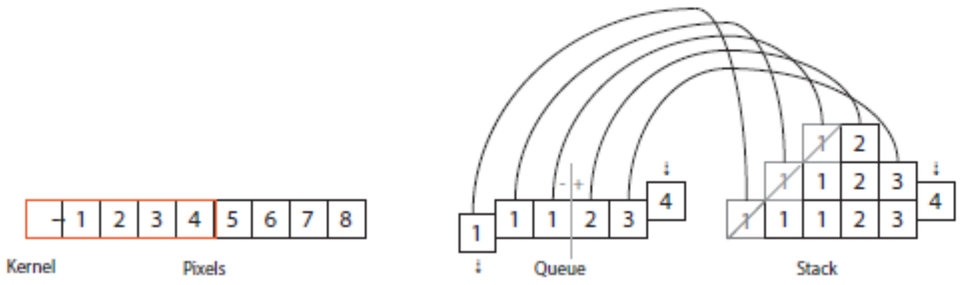
The kernel progresses one pixel to the right.
The new value is added to the queue at the right and the leftmost queue value is removed
Values in the left half of the queue get subtracted from the stack, values in the right half get added to the stack
The stack is actually just a sum of all the values, not a structure it's just here to visualize the weight of the single values
The blurred value is simply the mean of the sum.

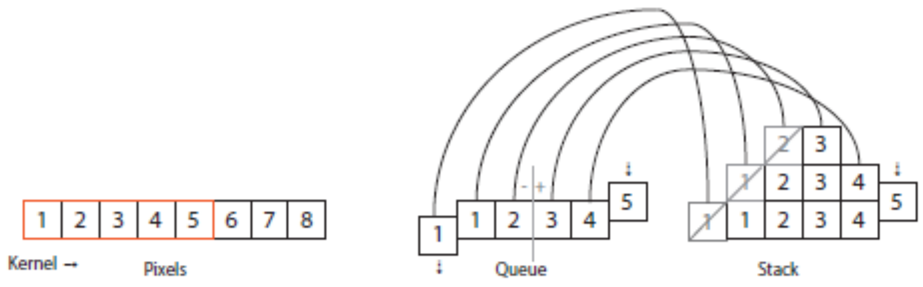


From now on the process repeats until the kernel is outside the right edge (at the right edge the righmost pixel value gets added when the parts of the kernel are outside)

When the end of the line is reached the kernel moves down one line and starts refilling like in step #1







When the horizontal pass is finished, the process repeats in vertical direction using the results of the horizontal pass.

In 2004 I required a fast but still good looking image blur and did not find any existing solutions that fit both requirements. Either they were classical Gaussian blurs that resulted in perfectly smooth blurs at the price of slow processing time or they were simple box blurs that were reasonably fast but resulted in a blocky look. So I wrote my own blur algorithm which tries to be a compromise between the two:

I called it Stack Blur because this describes best how this filter works internally: it creates a kind of moving stack (or maybe a “Tower of Hanoi” kind of structure) of colors whilst scanning through the image. This “tower” controls the weights of the single pixels within the convolution kernel and gives the pixel in the center the highest weight. The secret of the speed is that the algorithm just has to add one new pixel to the right side of the stack and at the same time remove the leftmost pixel. The remaining colors on the topmost layer of the stack are either added on or reduced by one, depending on if they are on the right or on the left side of the stack.

The original version was written in Java for Processing, but over the years I and other people have ported it to many different languages and platforms: C, C++, JavaScript, ActionScript, CUDA, iOS, Go etc. It has become part of some major libraries, frameworks and applications.