

## Re: Selective Gaussian -> Early test app posted

---

*Source:*

<http://coding.derkeiler.com/Archive/Delphi/borland.public.delphi.language.basm/2005-07/msg00460.html>

---

- *From:* "Mattias Andersson" <[mattias@xxxxxxxxxxxxxx](mailto:mattias@xxxxxxxxxxxxxx)>
  - *Date:* Mon, 18 Jul 2005 21:23:42 +0200
- 

Lord Crc wrote:

> On Thu, 14 Jul 2005 02:36:08 +0200, "Mattias Andersson"  
> <[mattias@xxxxxxxxxxxxxx](mailto:mattias@xxxxxxxxxxxxxx)> wrote:  
>  
>> Eric Grange wrote:  
>>> - should the matrix/kernel be passed to the function (and thus  
>>> assumed arbitrary) or should the function compute its own?  
>>  
>> Well, the problem with an arbitrary kernel, is that it imposes a  
>> constraint on the kind of optimizations you can perform. Since the  
>> gaussian kernel is separable, you can take advantage of this fact in  
>> order to speed up the implementation. If we assume an arbitrary  
>> kernel, then we can no longer assume that it is still separable.  
>  
> I believe the Selective Gaussian is not separable, because you do not  
> use all pixels when computing it (hence the name of the algorithm :).  
> Since you've lost this, you might aswell allow arbitrary kernels.

Well, I only said that the gaussian kernel was separable, right? So, the question is, can we take advantage of this for selective gaussian blur?

First of all, let us see how many operations we perform for each pixel with the standard (GIMP) implementation. If we have a kernel of width  $2*r+1$ , then the number of operations will be  $(2*r+1)^2$ .

Okay, now let us do some modelling.

Suppose that  $p[x, y]$  is the value of some color component of a pixel at coordinate  $(x, y)$ . Furthermore, let us consider a single reference pixel at coordinate  $(x\_ref, y\_ref)$ .

Also let us introduce a variable  $a[x, y]$  that is zero if  $Abs(p\_ref - p[x, y]) > \Delta$ , and one otherwise, i.e.

$a[x, y] = 0$  if  $Abs(p\_ref - p[x, y]) > \Delta$   
1 otherwise.

This means that we can express the weight of the kernel at coordinate  $(x, y)$

## Re: Selective Gaussian → Early test app posted

as follows:

$$w[x, y] = a[x, y] \exp(-x^2/c) \exp(-y^2/c)$$

In our summation loops, we will iterate from  $x_1$  to  $x_2$  and from  $y_1$  to  $y_2$ , where

$$\begin{aligned} x_1 &= x_{\text{ref}} - r & y_1 &= y_{\text{ref}} - r \\ x_2 &= x_{\text{ref}} + r & y_2 &= y_{\text{ref}} + r. \end{aligned}$$

Now, consider the following two expressions:

$$\text{Sum}[ y = y_1..y_2 ] \text{Sum}[ x = x_1..x_2 ] ( p[x, y] a[x, y] \exp(-x^2/c) \exp(-y^2/c) )$$

⇔

$$\text{Sum}[ y = y_1..y_2 ] ( \exp(-y^2/c) \text{Sum}[ x = x_1..x_2 ] ( p[x, y] a[x, y] \exp(-x^2/c) ) )$$

The second expression is in fact equal to the first one (this should be obvious). Also, this is how I suggest that one can take advantage of the fact that the gaussian is separable. However, in what way would this be useful? We still have the same number of operations. If we were optimizing ordinary gaussian blur, then we could have taken advantage of the fact that the horizontal summation ( $x = x_1..x_2$ ), would give the same result for any  $y$ -coordinate (independently of the reference coordinate). We can't use the same technique in selective blur, because for each  $y$ -coordinate we will have a different reference color.

What one could do instead, which I implemented in my optimized algorithm, is to cache the convolved values (the horizontal sums) for each reference color at a certain coordinate. This does not correspond to the same kind of optimization as in ordinary gaussian blur, where you would get complexity in the order  $\Theta(\text{pixels} * r)$ . Actually, if you have a completely solid image (with only a single color), then you would get this complexity too. However, as a worst case scenario one would still get  $O(\text{pixels} * r^2)$ .

Anyhow, hope I managed to convince you that separability is useful for selective gaussian blur too. :)

>>> – is the current reference implementation numerically correct?

>>

>> Well, as I pointed out in one of my previous posts — the gaussian  
>> has infinite extent. Hence, if you wanted 100% accuracy, then you  
>> should sum up each pixel in the entire bitmap for each pixel  
>> coordinate. However, at a certain radius the contribution of a pixel  
>> value becomes insignificant and in this case you can safely assume  
>> the weight is zero without any impact on the final result.

>

> Which you can easily find out when you fill in the kernel. Simply keep

Re: Selective Gaussian -> Early test app posted

- > on going until the weights are less than  $1 / \text{NumLevels}$ . However you
- > might want to be slightly less correct for the sake of speed (as it
- > would have a drastic impact on the kernel size).

Well, not really true. Even if the weight is less than  $1 / \text{NumLevels}$ , it will still have an impact on the result. I should have said "without any considerable impact" in my post, because although a single pixel might not make a difference, there will certainly be a difference if a larger number of pixels are convolved with a weight less than  $1 / \text{NumLevels}$ . I think the best approach is really to plot the curve and decide on threshold value that seems reasonable. Alternatively one could test different threshold values and look at how the output image is affected.

Mattias

---

• *Follow-Ups:*

- ◆ *Re: Selective Gaussian -> Early test app posted*  
◇ From: Lord Crc
- ◆ *Re: Selective Gaussian -> Early test app posted*  
◇ From: Lord Crc
- ◆ *Re: Selective Gaussian -> Early test app posted*  
◇ From: Mattias Andersson

• *References:*

- ◆ *Selective Gaussian -> Early test app posted*  
◇ From: Eric Grange
- ◆ *Re: Selective Gaussian -> Early test app posted*  
◇ From: Mattias Andersson
- ◆ *Re: Selective Gaussian -> Early test app posted*  
◇ From: Lord Crc

- Prev by Date: *Re: Fastcode Ceil & Floor*
- Next by Date: *Re: QC voting*
- Previous by thread: *Re: Selective Gaussian -> Early test app posted*
- Next by thread: *Re: Selective Gaussian -> Early test app posted*
- Index(es):
  - ◆ *Date*
  - ◆ *Thread*