International Journal of Image and Graphics

Real-time continuous image processing

LEONARDO SACHT
UFSC,
Santa Catarina, Brazil

DIEGO NEHAB
IMPA,
Rio de Janeiro, Brazil

RODOLFO SCHULZ DE LIMA
NVIDIA,
Munich, Germany

In this work, we propose a framework that performs a number of popular image-processing operations in the continuous domain. This is in contrast to the standard practice of defining them as operations over discrete sequences of sampled values. The guiding principle is that, in order to prevent aliasing, non-linear image-processing operations should ideally be performed *prior* to prefiltering and sampling. This is of course impractical, as we may not have access to the continuous input. Even so, we show that it is best to apply image-processing operations over the continuous reconstruction of the input. This transformed continuous representation is then prefiltered and sampled to produce the output. The use of high-quality reconstruction strategies brings this alternative much closer to the ideal than directly operating over discrete values. We illustrate the advantages of our framework with several popular effects. In each case, we demonstrate the quality difference between continuous image-processing, their discrete counterparts and previous anti-aliasing alternatives. Finally, our GPU implementation shows that current graphics hardware has enough computational power to perform continuous image processing in real-time.

Keywords: Image processing; parallelization; sampling; reconstruction; interpolation.

1. Introduction

Figure 1a shows an input image to which we want to apply an image-processing operation. Directly applying the operation to discrete samples can lead to objectionable artifacts, such as *jagged-edges*, that are clearly visible in figure 1b. Although the ideal solution to this problem would require access to the continuous scene from which the image was produced (see figure 1c for an approximation), figure 1d shows that our method can produce results that are closer to the ideal, even though it uses the same amount of information as the discrete method.

The generation of a properly antialiased image—be it captured by a real camera or synthesized by computer—is akin to evaluating a convolution integral between a function f (i.e., the scene), and a prefilter kernel ξ (i.e., a low-pass filter), at uniformly spaced positions.

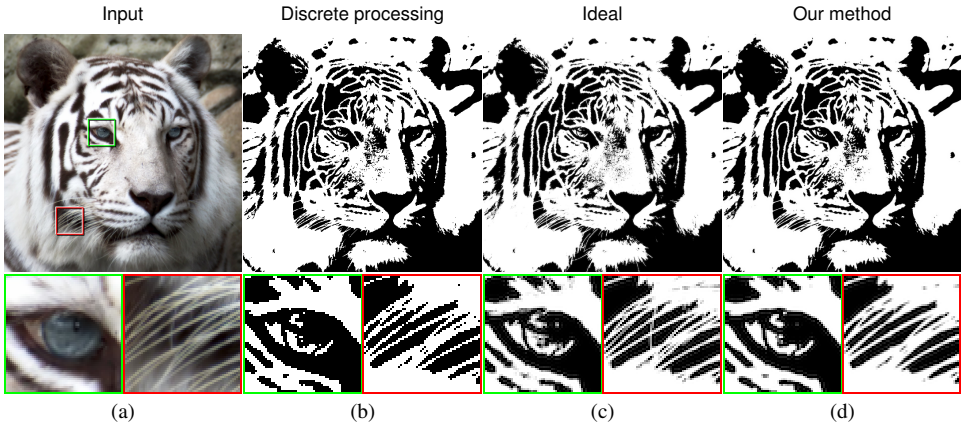


Figure 1. A continuous function f (a scene) is prefiltered and sampled, resulting in an image shown in (a), on which we wish to apply, for example, a threholding operation T. (b) When the transformation is applied directly to image samples, aliasing artifacts are visible. (c) Ideally, we should have prefiltered $T(f)$, but the continuous f may no longer be available. (d) Continuous image-processing reduces artifacts by first reconstructing the image, then transforming it in the continuous domain, before finally prefiltering the result to generate the output.

Working in 1D for simplicity, this process generates a discrete image with the following entries:

$$\boldsymbol{i}[k] = (f * \xi)(k) = \int_{-\infty}^{\infty} f(x)\xi(x-k)\,dx, \quad k \in \mathbb{Z}. \tag{1}$$

This is how the input image in figure 1a was generated.

Image reconstruction happens naturally when images are reproduced in computer monitors or printed on paper, and we often need to reconstruct them for further processing by computer. Whichever the case may be, the process can be modeled by mixed convolution between the samples and a continuous reconstruction kernel φ:

$$\tilde{f}(x) = (\boldsymbol{i} * \varphi)(x) \stackrel{\text{def}}{=} \sum_{k\in\mathbb{Z}} \boldsymbol{i}[k]\,\varphi(x-k). \tag{2}$$

It is important to select prefilter ξ carefully in order to minimize pre- and post-aliasing in the results of reconstruction with φ.

Unfortunately, since the discrete representation is so convenient to manipulate, a large number of image processing operations have traditionally been defined to operate directly over it. For example, a typical *thresholding* operation may be defined as:

$$(T_{a,b}(\boldsymbol{i}))[k] = t_{a,b}(i[k]), \quad k \in \mathbb{Z}, \quad \text{where} \tag{3}$$

$$t_{a,b}(x) = \begin{cases} 0, & x < a \quad \text{or} \quad x > b, \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

Convenience aside, this definition does not correspond to the image formation model described by (1). This is not a technicality: the transformation breaks the "contract" between the prefilter ξ and the reconstruction kernel φ, so that the reconstruction of the samples

in $T_{a,b}(\boldsymbol{i})$ with φ exhibits a number of post-aliasing artifacts. This is the source of the jagged-edges in figure 1b.

To produce antialiased results, the thresholding operation should instead be applied to the original continuous *f*, *prior* to prefiltering and sampling:

$$\boldsymbol{ti}[k] = \int_{-\infty}^{\infty} (t_{a,b} \circ f)(x)\psi(x-k)\,dx, \quad k \in \mathbb{Z}. \tag{5}$$

This formulation matches equation (1) perfectly, and consequently produces the ideal results shown in figure 1c. (We allow for a different prefilter ψ since we may not have access to the original prefilter ξ.)

There are two difficulties in realizing this ideal formulation. One is that we may not have access to the continuous *f*, i.e., when presented with a digital image captured by camera, or previously rendered by computer. The other is that formulating the desired image-processing operation in the continuous domain may not be a trivial matter. This is the case of gradient domain operations, for example.

In this work, we address the first problem by reconstructing $\tilde{f}$ from the image samples in $\boldsymbol{i}$ prior to image processing. Except in extreme conditions, the approximations produced by modern reconstruction strategies [1,2,3,4,5] capture the essence of the continuous f so well that the most objectionable artifacts are gone. The second problem, that of reformulating standard image-processing operations to the continuous domain, is addressed in a case-by-case basis.

We believe that our real-time realization of this continuous image processing framework will encourage practitioners that seek high-quality results to think of their algorithms directly on the continuous domain, instead of adotping the discrete framework. Making an analogy with numerical solution of differential equations, the higher accuracy and flexibility of finite elements makes it more useful than finite differences. This is likely to be the case of continuos image processing versus discrete image processing and we are taking a step towards this direction with our paper.

The remaining text is outlined as follows. Section 2 discusses related work. In section 3 we present the method and analyze different possibilities for reconstruction strategies, prefilters and sampling patterns. Continuous image processing operations are described in section 4. Details of our real-time implementation are presented in section 5, and finally results and comparisons appear in section 6.

2. Related Work

The work most related to ours can be categorized according to the amount and type of information that is assumed to be available for image processing. As a general rule, the more information used, the higher the quality of results. Our method fits nicely into an underexplored niche, as can be seen from the discussion that follows.

Complete access to the input function f and operator T: In this ideal scenario, both the input function f and the image-processing operator T are available in the continuous domain. The rendering task simply samples the convolution $T(f) * \psi$ between the transformed input

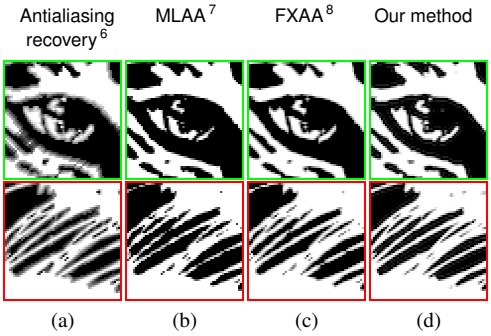


Figure 2. Comparison between our method and approaches that try to correct artifacts that appear in figure 1(b) (post-processing). All such methods can become confused when edges are too close together. Our method robustly handles such cases.

and the prefilter kernel at each pixel, as if rendering from the function $g = T(f)$. This method can be simulated when rendering from procedural scenes, or when the input image is available at much higher resolution than the desired output resolution (e.g., the ideal result in figure 1c).

Access to the sampled image i and operator T: In our work, we assume the continuous input f is unavailable. We are left with input values sampled at the same resolution as the desired output. From this, we reconstruct a high-quality approximation $\tilde{f}$ to the continuous input f. Since we assume access to the continuous operator T, we sample the convolution integral $T(\tilde{f}) * \psi$ between the continuous transformed reconstruction and the prefilter kernel at each pixel.

Access to the sampled image i and sampled processed image $T(i)$: The assumption in *antialiasing recovery* [6] is that the operator T has been lost as well. Only the original sampled input $\boldsymbol{i}$ and the transformed sampled input $T(\boldsymbol{i})$ are available. The goal is to find a sampled output with the overall aspect of the transformed sampled input, but with edges that are as nicely antialiased as those in the sampled input.

One of the main advantages is that antialiasing recovery is agnostic to the non-linear effect in T, whereas our method requires a continuous definition of the same effect. This is not always easy to formulate. The other main advantage is speed. Note, however, that this tells more about the simplicity of their method than about the complexity of ours, which *also* runs in real-time. Additionally, our method excels in a number of scenarios where antialiasing recovery fails (see figure 2a). First, antialiasing recovery assumes a pixel-to-pixel correspondences between the original and transformed inputs. This imposes a difficulty if there is any kind of warp in the transformation. Second, it employs edge detection to select the pixels for recovery. This detection can become confused when edges are close to each other. Third, it assumes that edge pixels can be modeled as a linear combination of the two colors across the edge. This assumption breaks down at corners and other high-frequency regions.

Access only to the sampled processed image $T(i)$: This scenario is treated by a series of algorithms collectively known as *post-processing antialiasing*[7,8,9]. These assume that not even the sampled input is available anymore. Only the sampled transformed input remains. Such methods have recently received much attention from the real-time rendering community for their ability to virtually eliminate the jagged edges associated to point-sampling. Besides bringing performance and quality improvements to older hardware (i.e., gaming consoles), post-processing is the only practical antialiasing alternative for deferred-shading effects. (Both AMD and NVIDIA now offer these solutions.) Nevertheless, results show that the restricted amount of input available to these methods prevents them from matching the quality of our method (see figures 2b,c).

Reconstruction and prefiltering strategies: The naïve threholding operation described by (4) can be interpreted within our framework as first reconstructing the input $\tilde{f}$ with a box kernel, and then using the same kernel as a prefilter. Although the box kernel is not a particularly good prefilter, it is not responsible for the artifacts visible in figure 1b. These artifacts are instead associated to box reconstruction.

High-quality reconstruction is therefore fundamental to our work. Here we benefit from recent progress on different fronts. In particular, contributions from approximation theory have guided the field toward increased *approximation order* (for quality), and narrow support (for reconstruction efficiency).[2] Most recently, Sacht and Nehab[5] showed that approximation order is not the key criterium for approximation quality in image-processing and minimized the approximation error over the full Nyquist range. Their experiments showed improved reconstruction quality over all previous methods that fit into the sampling and reconstruction pipeline. In our work, we make a series of experiments in section 3.1 that corroborate the higher quality of their quasi-interpolators in our context. To implement them, we first transform the input image with the associated digital recursive filter during a preprocessing step. The resulting "image" is then reconstructed as usual, with the narrowly-supported continuous kernel. Both these operations can be efficiently performed on modern GPUs. For the digital filtering stage, we use the parallel recursive filter algorithm of Nehab et al.[10]

Although there is significant freedom in the choice of prefilter, we favor the B-spline family. In particular, we follow the suggestion of McCool[11] and Nehab and Hoppe[12] to prefilter with fundamental splines (a.k.a. *cardinal* splines). These are efficient to evaluate and produce high quality results (see section 3.2 for comparisons to other alternatives). The same factorization applies: the integration is performed against the basic B-Spline continuous kernel, after which the result undergoes a postprocessing stage that applies the associated digital filter.

3. Method and choices

The continuous image processing operations described in the next section can be all performed by following the same steps. Given an input sampled image $\boldsymbol{i}$, a reconstruction kernel φ with associated digital filter $\boldsymbol{r}$, an image processing operator T, and a prefilter ψ

with associated digital filter $\boldsymbol{p}$, we perform the following steps:

(1) Preprocess the input sampled image with the reconstruction digital filter:

$$\boldsymbol{c} = \boldsymbol{i} * \boldsymbol{r}. \tag{6}$$

In the case when the digital filter $\boldsymbol{r}$ has finite impulse response (FIR) this computation corresponds to a discrete convolution. When it is the inverse of a FIR filter, the result of equation (6) is obtained by solving a linear system.

(2) Prefilter the transformed reconstruction:

$$d_k = (T(\tilde{f}) * \psi)(k). \tag{7}$$

I.e., entry d_k of the vector $\boldsymbol{d}$ is given by the integral:

$$d_k = \int_{-\infty}^{\infty} \big(T(\tilde{f})\big)(x)\, \psi(x-k)\, \mathrm{d}x, \tag{8}$$

where the reconstructed input is given by

$$\tilde{f}(x) = \boldsymbol{c} * \varphi \stackrel{\text{def}}{=} \sum_{k \in \mathbb{Z}} c_k\, \varphi(x-k). \tag{9}$$

(3) Postprocess with the prefilter digital-filter to obtain the output:

$$\boldsymbol{ti} = \boldsymbol{d} * \boldsymbol{p}. \tag{10}$$

As in equation (6), this computation depends on the type of the digital filter $\boldsymbol{p}$.

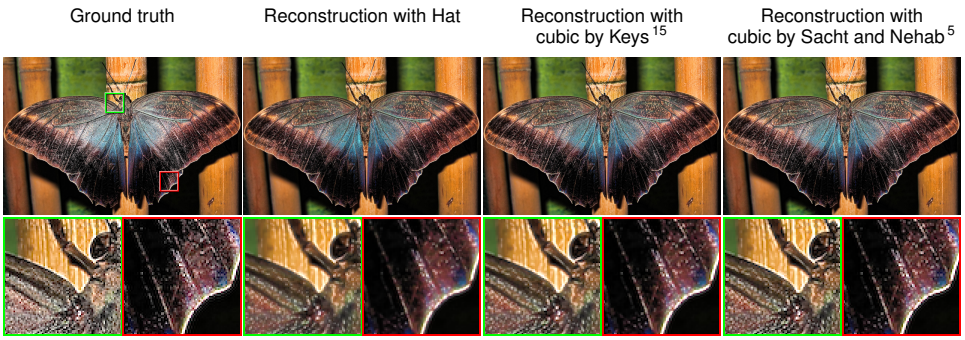
Since the integral in (8) can rarely be computed in closed form, we use Monte-Carlo integration (i.e., *supersampling*) to compute the value of each d_k in all our experiments.

Different choices of reconstruction kernel φ, prefilter ψ and sampling pattern to approximate (8) lead to different results $\boldsymbol{ti}$ in (10). We now analyze these different choices and select the best options. Implementation details are provided in Section 5.

3.1. *Reconstruction*

The choice of the reconstruction kernel φ is the one that has the greatest impact on the quality of the output image. The literature on interpolators for image-processing has evolved a lot in the last decades and offers a huge variety of possibilities for the reconstruction kernel φ: B-splines [13], Windowed-sinc approximations [14], MOMS [2], quasi-interpolators [3,4,5] and the kernels proposed by Keys [15], Mitchell and Netravali [16], Schaum [17] and Dodgson [18].

We compare all these possibilities with an experiment. First, a ground-truth result is generated from a high-resolution input in the follwing manner: starting from an image $\mathbf{I}$ sampled at a very high resolution ($4\times$ to $8\times$ denser than the final output over each dimension), we reconstruct it with the best kernels in the literature [2,3,5] to obtain a reconstruction $\tilde{F}$. At this point, the choice of the kernel does not make much difference, since the best ones generate a reconstruction $\tilde{F}$ very close to the continuous scene. We then sample the convolution between $T(\tilde{F})$ and a prefilter ξ at a lower resolution just as in (8), using a dense



(a) SSIM = 1.0 (b) SSIM = 0.968 (c) SSIM = 0.977 (d) SSIM = 0.981

Figure 3. The choice of the generator φ in our method affects the quality of the final result, as shown in this unsharp masking example. Subpixel oscilations in (a) are more faithfully reproduced by high-quality interpolators such as in (d), as corroborated by the SSIM values (the higher the better).

sampling pattern to approximate the integral. This result is post-processed by the digital filter $\boldsymbol{p}$ in equation (10) to generate the output $\boldsymbol{G}$. This process was used to generate the ideal result in Figure 1c.

We then generate a low resolution discrete image $\boldsymbol{i}$ in the same way, with the only difference of replacing the operator T by identity, to be the input for our method. This low resolution image $\boldsymbol{i}$ is processed by equations (6) to (10) to generate the output $\boldsymbol{g} = \boldsymbol{ti}$, using the different possibilities for the reconstruction kernel φ. The prefilter ψ and the sampling pattern are the same used to generate the output discrete images $\boldsymbol{G}$ and $\boldsymbol{i}$. Finally, we compare $\boldsymbol{g}$ to the ground-truth $\boldsymbol{G}$ using the SSIM metric [19].

Figure 3 shows the result of this experiment for the unsharp masking (sharpening) operation, defining the prefilter ξ to be the cardinal cubic B-spline for all images. While subpixel details are lost by lower quality interpolators—see linear reconstruction (b), and the widely used cubic by Keys [15] (c)—the best ones in the literature [5] are able to reconstruct more details. This is corroborated by the SSIM results comparing the output images to the ground truth.

We repeated the experiment above for five different high-resolution images, using five different image-processing operations (thresholding, contrast and brightness change, quantization,unsharp masking and embossing). We favored highly non-linear operations to test the robustness of the different reconstruction kernels φ. We also tested five different prefilters ξ varying from smooth to sharp: quintic B-spline, cubic B-Spline, hat, cardinal cubic B-Spline and cardinal quintic B-spline. The goal is investigating different settings that could appear in real scenes, for which we have no control over the prefilter.

Averaging the SSIM values over all these tests revealed that the best results are achieved setting the reconstruction kernel φ to be the cubic quasi-interpolator proposed by Sacht and Nehab [5], and we adopt it as the default of our method. Higher speed and good quality can be achieved with the linear quasi-interpolator proposed by Sacht and Nehab [5] and the cardinal version of the linear interpolator proposed by Blu et al. [20]

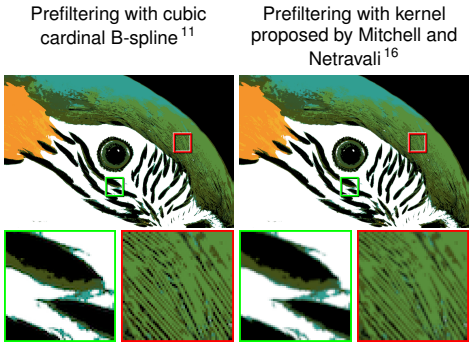


Figure 4. Different choices of prefilter ψ lead to different results, as illustrated in this quantization example. For sharper results (with mild ringing) we recommend to prefilter with cubic cardinal B-splines (left). The user that prefers smoother results can opt for prefiltering with the kernel proposed by Mitchell and Netravali (right).

We opted for a practical experiment rather than a theoretical argumentation for two reasons: first, in the real setting, we do not have control over the prefilter ξ used to generate the discrete input $\boldsymbol{i}$ in equation (1). Second, even if we did know the prefilter ξ, the literature is focused on producing a high quality approximation $\tilde{f}$ for the original function f, and this does not guarantee that $T(\tilde{f})$ will be a good approximation for $T(f)$. Nonetheless, our experiments show that $T(\tilde{f})$ is, in fact, closer to $T(f)$ than the results obtained by other methods.

3.2. *Prefiltering*

The fact that we do not know the original prefilter ξ makes it impossible to replicate the image generation process employed to obtain the input image. On the other hand, we can make different choices to fulfill different goals, such as balance between ringing and aliasing.

We give the user the possibility of choosing between two prefilters: cardinal cubic B-splines (as suggested by McCool [11]) and the kernel proposed by Mitchell and Netravali [16]. Both have a good balance between efficiency and quality. As shown in figure 4, the former produces sharper results, at the cost of mild ringing, and the latter produces blurrier results with no visible ringing. Both results were generated using the reconstruction method defined in the previous subsection and sampling pattern chosen in the next one. We set as default prefilter in our method the cardinal cubic B-spline.

3.3. *Sampling pattern*

To approximate the integral in (8), we tested different sampling patterns with varying densities inside a pixel region. Fixing the reconstruction kernel and prefilter as the ones obtained in sections 3.1 and 3.2, we generated results in a low resolution changing the sampling patterns and compared to a ground truth image, generated with a much higher resolution.

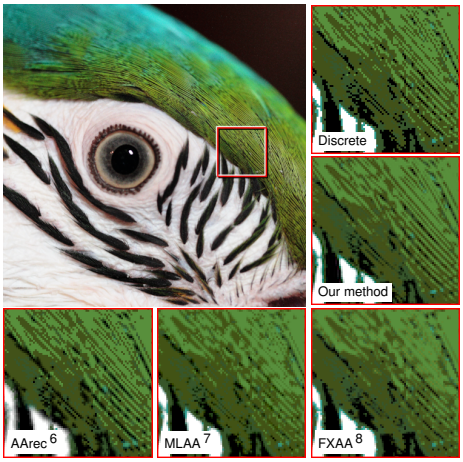


Figure 5. Quantization.

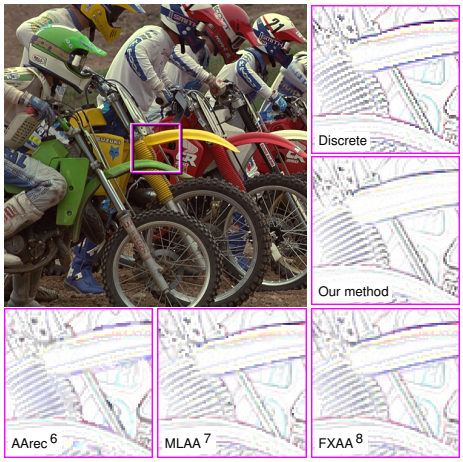


Figure 6. Laplacian.

The distribution of samples tested were: regularly distributed, uniformly distributed, stratified sampling, blue noise[21], and two jittered sampling strategies proposed by Halton[22] and Kensler[23]. For each of these patterns, we obtained results with 4, 8, 16, 32, 64, 128 and 256 samples per pixel.

As expected, the worst results were obtained by regular and uniform sampling. In some cases, stratified sampling needed more samples to achieve the same quality of the best patterns, which were blue noise[21], and the ones proposed by Halton[22] and[23]. These three strategies produced results with equal SSIM up to the fourth decimal and visually identical. We also observed small increase of quality varying the number of samples per pixel from 4 to 32, and no difference from 32 to 256 samples. Thus, we define as default of our method blue noise sampling with 32 samples per pixel.

4. Image processing operations

We now describe basic transformation categories. The classification serves an organization purpose only: complex image-processing operations will in general combine many of these categories.

4.1. *Pointwise operations*

Pointwise operations modify the *value* of the input at a single point:

$$T : f \longrightarrow g \circ f. \tag{11}$$

Dozens of common image processing operations are pointwise in this sense. Examples include typical image adjustment tools that modify brightness and contrast, hue and saturation, levels and curves, gamma, white balance etc. The category also encompasses less common effects, such as posterization, thresholding, color replacement, histogram equalization, and many others.

Pointwise operations can be highly non-linear (discontinuous, even) and significantly change the frequency content of the input. It is in such cases that the advantages of our method are most clearly visible: the prefiltering stage prevents aliasing artifacts from appearing in the results. See section 6 for the results of pointwise operations rendered with our method.

4.2. *Warping operations*

Warping operations modify *where* the input is evaluated:

$$T : f \longrightarrow f \circ g. \tag{12}$$

The obvious examples are projective transformations that encompass scaling, rotation, translation, as well as perspective warps. Image editing programs often provide creative warps in a variety of *distortion filters*. Warps are also very common in rendering tasks that project texture-mapped geometry to the screen.

Many techniques have been designed to enable efficient prefiltered access to warped functions, most notably mipmaps[24] and its many anisotropic variants. In a sense, these methods are shortcuts for the computation of the integral (8). Therefore, they are all applicable to our framework as long as the image-processing operation does not involve a significant non-linear function value transformation. Otherwise, we simply resort to supersampling.

We emphasize that the idea of continuos processing applied to warpings has been long discussed, such as in the book by Wolberg[25]. Our contribution is to fit it into a more general image processing framework and to show that it can produce high quality results in real-time (see sections 5 and 6).

4.3. *Integral operations*

Integral operations involve entire neighborhoods of function values. The quintessential example of integral operation is the convolution. However, in order to capture a wider range of operations, we include a normalization term:

$$T : f \longrightarrow \frac{\int_{-\infty}^{\infty} g\big(f(t-\cdot)\big)dt}{\int_{-\infty}^{\infty} w\big(f(t-\cdot)\big)dt} \tag{13}$$

The most popular integral operator is the Gaussian blur. Since this is a low-pass filter, there is no benefit from continuous image processing. Nevertheless, Gaussian blur is often used as a component of more sophisticated techniques that *do* benefit from our work.

One example is *unsharp masking*. Here, we lose nothing in quality when computing the Gaussian blur in the discrete domain. However, the remaining operations are non-linear and must be performed in the continuous domain. Fortunately, these are all pointwise operations and therefore we do not face any performance impact. The same is true for the *embossing* effect.

An example where this optimization does not apply is the *bilateral filter*[26], which often produces jagged-edges characteristic of non-linear image processing. The brute-force

Figure 7. Unsharp masking.

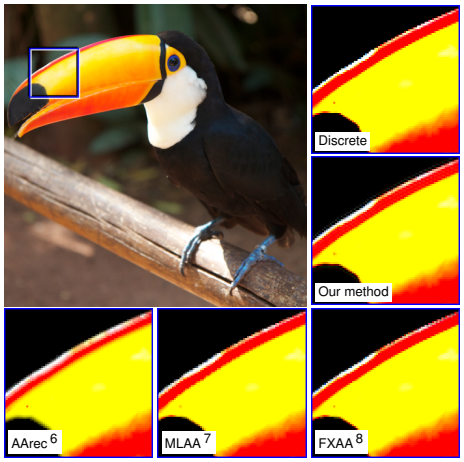


Figure 8. Brightness and contrast.

approach is to nest the integral operator with the convolution integral in (8), leading to a double integral. Although this is significantly more expensive to compute than other effects, our implementation can still run interactively.

Naturally, not all non-linear neighborhood operations can be expressed as in (13). A popular example that is particularly challenging to define in the continuous domain is the *median filter*. This is an interesting area for future research.

4.4. *Derivatives*

In the continuos setting, derivatives (of any order) can be seen as pointwise operations instead of neighborhood ones. We calculate derivatives by taking the derivatives of the reconstruction in (9), which is equivalent to recontructing with the derivative of the generator φ. This requires the generator to be differentiable, which is not the case for the one proposed by Sacht and Nehab[5].

Thus, for operations such as edge detection and Laplacian sharpening (figures 6 and 11), we replace φ to be the interpolating cubic B-spline, which is twice differentiable, and reconstruct with its derivative. We refer the reader to the work by Nehab and Hoppe[12] for details about the approximation properties of this approach.

5. Implementation details

A few details that make our implementation very efficient are worth mentioning. Among them is our approach to evaluating the convolution integrals in (8). It scales to any number of samples per unit area, since the amount of memory required depends only on the prefilter kernel support. The input function is evaluated only once per sample, which is important since input function evaluation can be arbitrarily costly. Yet, each evaluated function value contributes to all neighboring pixels whose kernels overlap with the sample, which is

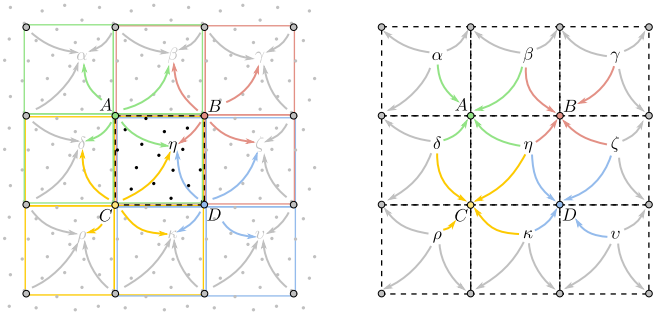


Figure 9. Integration scheme. The 2×2 support of prefilter kernels centered at each of pixels A, B, C, and D overlap the same *unit area* η. In the first phase, we loop over these unit areas, evaluate the function at each sample position in the area, and accumulate a weighted sum for each of the 2×2 kernels overlapping the area. In the second phase, we loop over the pixels, gathering the corresponding partial sums from each unit area its kernel overlaps.

important for variance reduction. Finally, it is massively parallel, with no read- or write-hazards.

Without loss of generality, figure 9 illustrates the integration process with a prefilter kernel of support 2×2 (e.g. the hat kernel). In this case, every unit area is covered by exactly 2×2 neighboring kernels. In the figure, unit area η is covered by kernels centered at pixels A, B, C, and D. We split integration into two stages. In the first stage, we loop over unit areas. Each unit area keeps one partial sum for each of the 2×2 kernels that overlap with it. For each sample in the area, we evaluate the function once, and then for each partial sum we compute the appropriate kernel value and accumulate its product with the sampled function value. When the partial sums in every unit area are finished computing, we start the second stage. Here, we loop over the output pixels. For each pixel, we simply gather the partial sums from the 2×2 unit areas its kernel overlaps and add them together to produce the final pixel value. In the first stage, all unit areas are independent. In the second stage, all pixels are independent. The process is therefore embarrassingly parallel. As discussed in subsections 3.2 and 3.3, we use the 4×4 cubic B-spline prefilter and blue-noise patterns[21]. This technique allows us to quickly compute all required convolution integrals.

Pre and post-digital filtering in equations (6) and (10) are efficiently performed using the method of Nehab et al.[10] We also leverage the GPU's bilinear filtering hardware during reconstruction (9), as per Sigg and Hadwiger[27] and Ruijters et al.[28]

6. Results

Figures 5, 6, 7, 8, 10, and 11 compare the quality of the methods discussed in section 2, on different images and under different transformations. These results are all available in supplemental material for easier comparison. Although we believe the results speak for themselves, some comments are warranted.

Naturally, the shortcomings of any image-processing operation become the most evident when "strong" effects are applied. Results in figures 8 and 11 show that, even in those cases,

Figure 10. Embossing.

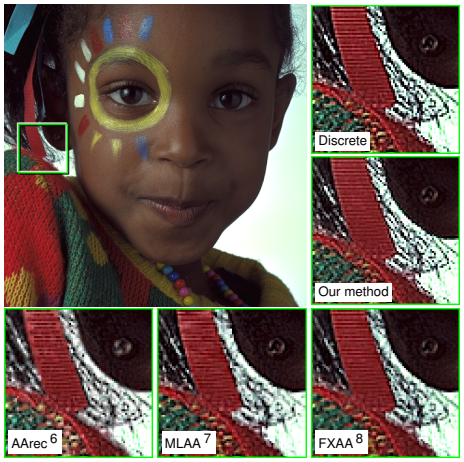


Figure 11. Excessive sharpening.

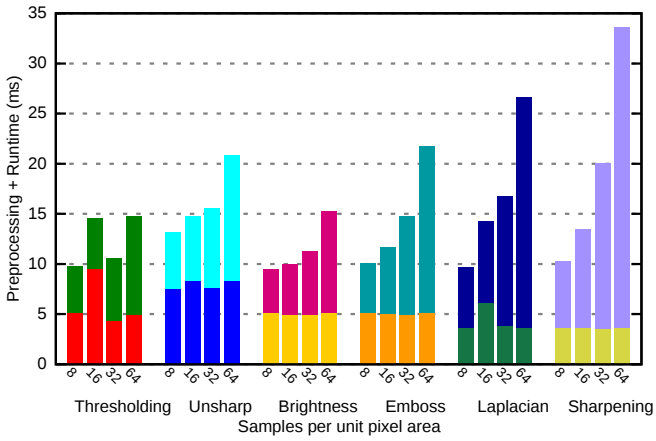


Figure 12. Timings for six different effects, running on 1920×1080 (full HD) input images. These effects run in real-time with up to 64 samples per unit area on a GeForce TITAN X (Pascal). This is more than enough for excellent quality. “Thresholding” is the effect in figure 1, unsharp in figure 7, “brightness” (& contrast) in 8, “emboss” in 10, “laplacian” in 6, and “sharpening” is the effect in figure 11.

our method robustly prevents objectionable artifacts from appearing in the output images. All examples show that our method is better at dealing with small features. This is because we do not rely on the edge-detection stage used by the other methods. This is particularly evident in figures 5 and 6. Finally, our method prevents the appearance of jagged edges throughout all tests, without undoing the work of the image-processing operation. This is most visible in figures 7 and 10. These operations enhance edges by design, and methods that attempt to attenuate these edges end up destroying them along with other relevant fine detail.

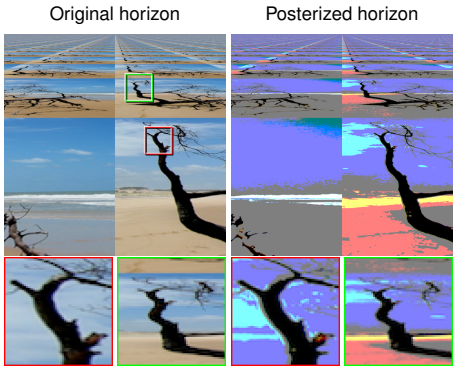


Figure 13. Warp combined with the posterization effect.

The graph in figure 12 shows performance numbers for our GPU implementation. These numbers refer to the processing of 1920×1080 images. (Our method scales linearly with the number of output pixels.) For each effect, preprocessing time is shown in the bottom (e.g., digital filtering or blurring of input images), while interaction times are shown above them (i.e., time taken to provide feedback to a user changing a slider). Although in section 3.3 we defined the defalut resolution of the sampling pattern as 32 samples per pixel, we show peformance numbers up to 64 samples per pixel for completeness. At these rates, our implementation running on a GeForce TITAN X (Pascal) can provide users with real-time feedback (at least 30 frames per second) for all effects in figure 12.

Finally, figure 13 shows a continuous image processing example involving spatial warp. We show an infinite plane in perspective, where we have tiled copies of an image that was subjected to posterization with 3 levels. This is a trivial application that presents no problems to the image-processing pipeline we propose.

7. Conclusions

In this paper, we identified the common practice of performing image-processing operations in the discrete domain as a potential source of undesirable artifacts. Instead, our continuous image processing approach reformulates image-processing operations in the continuous domain, and apply them over the continuous reconstruction of the input. This transformed continuous representation is then prefiltered and sampled as if in a rendering system. Continuous image processing increases the creative power of artists by making image-processing tasks more robust to extreme settings.

References

1. T. Blu and M. Unser. Quantitative Fourier analysis of approximation techniques: Part I—Interpolators and projectors. *IEEE Transactions on Signal Processing*, 47(10):2783–2795, 1999.
2. T. Blu, P. Thévenaz, and M. Unser. MOMS: Maximal-order interpolation of minimal support. *IEEE Transactions on Image Processing*, 10(7):1069–1080, 2001. doi: 10.1109/83.931101.

3. L. Condat, T. Blu, and M. Unser. Beyond interpolation: optimal reconstruction by quasi-interpolation. In *Proceedings of the IEEE International Conference on Image Processing*, volume 1, pages 33–36, 2005. doi: 10.1109/ICIP.2005.1529680.
4. M. Dalai, R. Leonardi, and P. Migliorati. Efficient digital pre-filtering for least-squares linear approximation. In *Visual Content Processing and Representation*, volume 3893 of *Lecture Notes in Computer Science*, pages 161–169, 2006.
5. L. Sacht and D. Nehab. Optimized quasi-interpolators for image reconstruction. *IEEE Transactions on Image Processing*, 24(12):5249–5259, 2015. doi: 10.1109/TIP.2015.2478385.
6. L. Yang, P. V. Sander, J. Lawrence, and H. Hoppe. Antialiasing recovery. *ACM Transactions on Graphics*, 30(3):22, 2011.
7. A. Reshetov. Morphological antialiasing. In *High Performance Graphics*, pages 109–116, 2009. doi: 10.1145/1572769.1572787.
8. T. Lottes. Fast approximation antialiasing (FXAA). NVIDIA whitepaper, 2011.
9. Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. SMAA: Enhanced subpixel morphological antialiasing. *Computer Graphics Forum*, 31(2pt1):355–364, 2012.
10. D. Nehab, A. Maximo, R. S. Lima, and H. Hoppe. GPU-efficient recursive filtering and summed-area tables. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2011)*, 30(6): 176, 2011. doi: 10.1145/2024156.2024210.
11. M. D. McCool. *Analytic Signal Processing for Computer Graphics using Multivariate Polyhedral Splines*. PhD thesis, University of Toronto, 1995.
12. D. Nehab and H. Hoppe. A fresh look at generalized sampling. *Foundations and Trends in Computer Graphics and Vision*, 8(1):1–84, 2014. doi: 10.1561/0600000053.
13. M. Unser, A. Aldroubi, and M. Eden. B-spline signal processing: Part II—efficient design and applications. *IEEE Transactions on Signal Processing*, 41(2):834–848, 1993.
14. E. H. W. Meijering, W. J. Niessen, J. P. W. Pluim, and M. A. Viergever. Quantitative comparison of sinc-approximating kernels for medical image interpolation. In *Medical Image Computing and Computer-Assisted Intervention*, volume 1679 of *Lecture Notes in Computer Science*, pages 210–217. 1999.
15. R. G. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981.
16. D. P. Mitchell and A. N. Netravali. Reconstruction filters in computer graphics. *Computer Graphics (Proceedings of ACM SIGGRAPH 1988)*, 22(4):221–228, 1988. doi: 10.1145/54852.378514.
17. A. Schaum. Theory and design of local interpolators. *Computer Vision, Graphics & Image Processing*, 55(6):464–481, 1993.
18. N. A. Dodgson. Quadratic interpolation for image resampling. *IEEE Transactions on Image Processing*, 6(9):1322–1326, 1997.
19. Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
20. T. Blu, P. Thévenaz, and M. Unser. Linear interpolation revitalized. *IEEE Transactions on Image Processing*, 13(5):710–719, 2004.
21. M. Balzer, T. Schlomer, and O. Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2009)*, 28(3): 86, 2009.
22. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.
23. A. Kensler. Correlated multi-jittered sampling, 2013. Pixar Technical Memo 13-01.
24. L. Williams. Pyramidal parametrics. *Computer Graphics (Proceedings of ACM SIGGRAPH 1983)*, 17(3):1–11, 1983.
25. G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1990.

26. Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision*, pages 839–846, 1998.
27. C. Sigg and M. Hadwiger. Fast third-order texture filtering. In M. Pharr, editor, *GPU Gems 2*, chapter 20, pages 313–329. Addison Wesley Professional, 2005.
28. D. Ruijters, B. M. ter Haar Romeny, and P. Suetens. Efficient GPU-based texture interpolation. *Journal of Graphics, GPU & Game Tools*, 13(4):61–69, 2008.

Photo and Biography

Leonardo Sacht is an adjunct professor at Federal University of Santa Catarina (UFSC) in Florianopolis, Brazil. He received a bachelor degree in Mathematics and Scientific Computing from UFSC in 2008 and MSc and DSc degrees in Mathematics from the Institute for Pure and Applied Mathematics (IMPA) in 2010 and 2014, respectively. He also spent one year between 2012 and 2013 as a visiting student at ETH Zurich, in Switzerland. His main areas of research are image processing, geometry processing and numerical analysis.

Diego Nehab is an associate professor at the National Institute for Pure and Applied Mathematics (IMPA) in Rio de Janeiro, Brazil. He received BEng and MSc degrees in Computer Science from PUC-Rio in 2000 and 2002, respectively, and a PhD degree also in Computer Science from Princeton University in 2007. Before joining IMPA in 2010, he worked as a post-doctoral researcher at Microsoft Research in Redmond. He focuses on parallelism, real-time rendering, and image processing.

Rodolfo S. Lima is graduated from Rio de Janeiro's Federal University (UFRJ) with a degree in Electronics Engineering. He is a senior software engineer with 20 years of experience in C/C++ programming, 15 professionally. A multi-skilled IT professional, with proven experience on mobile, web-based, GPU, server and desktop development, software analysis, design and implementation of small to large projects, along with team management.