# Kernel Integral Images: A Framework for Fast Non-Uniform Filtering

Mohamed Hussein
Dept. of Computer Science
University of Maryland
mhussein@cs.umd.edu

Fatih Porikli
Mitsubishi Electric Research Labs
Cambridge, MA 02139
fatih@merl.com

Larry Davis
Dept. of Computer Science
University of Maryland
lsd@cs.umd.edu

## Abstract

*Integral images are commonly used in computer vision and computer graphics applications. Evaluation of box filters via integral images can be performed in constant time, regardless of the filter size. Although Heckbert [6] extended the integral image approach for more complex filters, its usage has been very limited, in practice. In this paper, we present an extension to integral images that allows for application of a wide class of non-uniform filters. Our approach is superior to Heckbert's in terms of precision requirements and suitability for parallelization. We explain the theoretical basis of the approach and instantiate two concrete examples: filtering with bilinear interpolation, and filtering with approximated Gaussian weighting. Our experiments show the significant speedups we achieve, and the higher accuracy of our approach compared to Heckbert's.*

## 1. Introduction

Filtering is a fundamental image processing operation. The computational complexity of image filtering depends on the complexity and size of the filter. For separable filters, for example, efficient computation is possible by applying two consecutive one-dimensional filters instead of the original two-dimensional filter. However, even when taking advantage of the filter's separability, the computational time increases with the filter's size, which is unfavorable for large filters. In some cases, we do not even know the filter size in advance, *e.g*. when the filter size is determined dynamically based on feature values. In such cases, the separability of the filter does not help. For box filters, which are used to compute averages and summations over rectangular image regions, there is an elegant technique that can overcome these difficulties. Given an integral of image features (Figure 1), filtering with a box filter at any point can be performed in constant time regardless of the filter size. Unfortunately, using pre-computed integrals is limited, in practice, to box filters. In this paper, we present a novel extension that makes pre-computed integrals usable for more complex filters.

The idea of using pre-computed integrals was first introduced, with the name *summed-area tables*, by Crow [3] to be used for texture mapping in computer graphics. Recently, it was popularized in the field of computer vision, with the name *integral images*, by Viola and Johns [11], who used it for fast computation of Haar wavelet features. Later on, integral images were generalized by Porikli [9] to *integral histograms*, which allow for fast construction of feature histograms. More recently, integral images and integral histograms were used to speed construction of Histograms of Oriented Gradient descriptors by Zhu *et al*. [13], Region Covariance descriptors by Tuzel *et al*. [10], and the SURF descriptors by Bay *et al*. [1].

To the best of our knowledge, usage of integral images in computer vision applications has been limited to the special case of box filtering although some of these applications can perform better when using non-uniform filters. For example, Dalal and Triggs [4] use bilinear interpolation between neighboring cells and Gaussian weighting of pixels within a block of pixels in constructing their histograms of oriented gradients features for human detection. They show how these weighting schemes enhance the detector's accuracy. To develop a fast version of Dalal and Triggs' detector, Zhu *et al*. [13] sacrifice the benefits of these weighting schemes to enable usage of integral images. Another example is in the work of Bay *et al*. [1], where Gaussian derivative filters are approximated by box filters so that integral images can be used. Perhaps, a better approximation would be possible if integral images were able to handle non-uniform weighting filters. A third example is in building appearance models for tracking, where pixels closer to the center of the tracked region are given higher weights than pixels closer to the borders, *e.g*. Elgammal *et al*. [5]. Consider a particle filter tracker, *e.g*. Zhou *et al*. [12], where appearance models for hundreds of overlapping regions need to be constructed, possibly for many tracked targets, on every frame. Applying non-uniform weighting of pixels in such a situation without the aid of a fast technique similar to integral images can be impractical for real-time application.

978-1-4244-2243-2/08/$25.00 ©2008 IEEE

Heckbert [6] introduced the theoretical foundation of the summed-area tables (integral images) technique and extended the theory to allow for more complex filters. However, his extension required a very high precession numerical representation even for moderate image sizes [7]. Similar to Heckbert, we present an approach to extend the integral images technique to allow for non-uniform filters. However, our approach has lower precision requirement than Heckbert's and is more suitable for parallel implementation. We call our approach *kernel integral images*. A kernel integral image is a group of integral images such that a linear combination of box filters applied to them is equivalent to applying a more complex filter. We instantiate two examples of applying our approach that are relevant to computer vision applications: feature filtering with bilinear interpolation, and approximation of filtering with Gaussian weighting. Our experimental analysis shows the significant speedups we achieve, and the superiority of our approach to Heckbert's in terms of accuracy.

The rest of the paper is organized as follows: section 2 introduces notation and explains integral images in an abstract form. Section 3 employs filtering with bilinear interpolation as an example to introduce our extension, which is afterwards formalized in section 4. Then, the example of filtering with approximate Gaussian weighting is described in section 5. In section 6, we compare our approach to Heckbert's. Empirical analysis of speedups and numerical errors are presented in section 7, followed by conclusions in section 8.

For clarity of presentation, we focus on one and two dimensional signals. The extension to higher dimensions is straight forward.

## 2. Fast Filtering via Integral Images

### 2.1. Preliminaries

Let $f : \mathbf{x} \rightarrow \mathcal{R}$ be a function that maps a point $\mathbf{x} = (x_1, x_2)$ to a real value, where $0 \leq x_i \leq N_i, N_i > 0, i = 1, 2$. Therefore, the domain of $f$, $\mathbf{D}_f$, is a rectangle bounded by the lines $x_i = 0$ and $x_i = N_i, i = 1, 2$. A rectangular region (referred to as a region from now on) $\mathbf{R} \subseteq \mathbf{D}_f$ is defined by a pair of points $\mathbf{x}^b$ and $\mathbf{x}^e$ such that $\mathbf{x}^b, \mathbf{x}^e \in \mathbf{D}_f$, and $x_i^b < x_i^e, i = 1, 2$. The two points $\mathbf{x}^b$ and $\mathbf{x}^e$ represent the two extreme points of the region $\mathbf{R}$. We refer to the ordered pair $\mathbf{r} = (\mathbf{x}^b, \mathbf{x}^e)$ as the region definition. Figure 1 illustrates some of these definitions. In practice, the function $f$ represents the raw intensity value or some other feature at each point in an image. Its domain, $\mathbf{D}_f$, is the set of all pixel coordinates in the image. $N_1 \times N_2$ is the image size.

A filtering of the values of $f$ over a region $\mathbf{R}$ can be defined as a function $\mathcal{A}_f : \mathbf{R} \rightarrow \mathcal{R}$ that maps the region to a real value. The form of the *filtering function* we consider

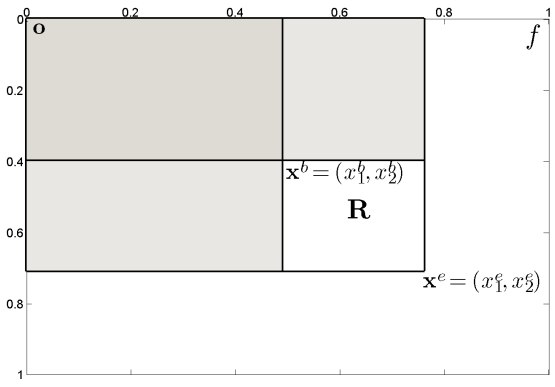


Figure 1. An integral of image features. The value of the integral at a point is the sum of the values of image features in the rectangular area from the origin to the point. The sum of feature values over any axis-aligned rectangular region (*e.g.* the small white rectangle) is determined by the value of the integral at the four corners of the region.

can be expressed as

$$\mathcal{A}_f(\mathbf{R}) = \sum_{\mathbf{x} \in \mathbf{R}} a_f^{\mathbf{r}}(\mathbf{x}), \tag{1}$$

where the *contribution function* $a_f^{\mathbf{r}}(\mathbf{x})$ defines the contribution of the point $\mathbf{x}$ to the filtering of the function $f$ over the region $\mathbf{R}$. In general, as the superscript of $a_f^{\mathbf{r}}$ indicates, the contribution of a point $\mathbf{x}$ depends not only on the point coordinates and the function $f$, but also on the definition of the region, *i.e.* its two extreme points. In this section we first consider the simpler case, where the contribution of a point is *independent* of the region's definition . We handle the general case in sections 3 and 4. Thus, for now, we denote the contribution function by $a_f$ instead of $a_f^{\mathbf{r}}$. Therefore, the filtering function is redefined as

$$\mathcal{A}_f(\mathbf{R}) = \sum_{\mathbf{x} \in \mathbf{R}} a_f(\mathbf{x}). \tag{2}$$

We call such a filtering function and its associated contribution function *region-independent* functions.

In its simplest form, the contribution function can be equal to the function $f$. That is

$$a_f(\mathbf{x}) = f(\mathbf{x}) \ . \tag{3}$$

But, in fact, we can use any function that can be evaluated independently from the filtering region's definition. For example, we can define the contribution function as

$$a_f(\mathbf{x}) = \|\mathbf{x}\| f^2(\mathbf{x}) \ . \tag{4}$$

Therefore, filtering with region-independent contributions is much more general than just summing feature values over a rectangular region.

### 2.2. Integral Images

When filtering is computed over many regions that overlap, using equation 2 is not efficient. This is because the computations performed in areas that are shared among more than one overlapping region will be repeated for each region. Luckily, the filtering equation has a sub structure that allows for a dynamic programming solution. This dynamic programming solution is what we refer to as integral images.

Define the *integral image* of a function $f$, $I_f$, as a function with the same domain and codomain as $f$, and of the form

$$I_f(\mathbf{x}) = \sum_{\mathbf{y}\in\mathbf{D}_f, y_i \le x_i, i=1,2} a_f(\mathbf{y}). \tag{5}$$

The value of the integral image of a function $f$ at a point $\mathbf{x}$ is the sum of the contributions of all points in the region defined by $(\mathbf{o}, \mathbf{x})$, where $\mathbf{o}$ is the origin or the coordinate system.

Given this formulation of integral images, it becomes much simpler to evaluate the filtering function over any region $\mathbf{R}$. A filtering function can be written in terms of an integral image as

$$\mathcal{A}_f(\mathbf{R}) = I_f(x_1^e, x_2^e) - I_f(x_1^b, x_2^e) - I_f(x_1^e, x_2^b) + I_f(x_1^b, x_2^b), \tag{6}$$

where $(\mathbf{x}^b, \mathbf{x}^e)$ defines the filtering region $\mathbf{R}$ (Figure 1).

In general, having the integral image, filtering over a region $\mathbf{R}$ is reduced to $O(1)$ computations compared to $O((x_1^e - x_1^b) \times (x_2^e - x_2^b))$ computations using the original filtering function formulation, equation 2. However, the cost of constructing the integral image itself is $O(N_1 \times N_2)$. Therefore, the utility of using integral images is realized only when we filter over many overlapping regions. In the case of exhaustively filtering over the entire domain of regions, the speedups obtained when using integral images were reported in [9] to be several orders of magnitude for a broad range of parameter choices.

## 3. Extending Integral Images for Filtering with Region-Dependent Contributions

Before discussing the formal treatment of the general case, where the contribution functions are dependent on the filtering region's definition, we start with a concrete example. Consider filtering with bilinear interpolation. A practical example is constructing the SIFT descriptor [8], where filtering is performed over adjacent regions in a $4 \times 4$ grid of cells of pixels, such that each pixel contributes to more than one cell via bilinear interpolation.

We want to define the contribution function in this case. A region $\mathbf{R}$ is defined by $\mathbf{r} = (\mathbf{x}^b, \mathbf{x}^e)$, where $\mathbf{x}^b = (x_1^b, x_2^b)$ and $\mathbf{x}^e = (x_1^e, x_2^e)$. The center of the region is $\mathbf{x}^c = (x_1^c, x_2^c) = (\mathbf{x}^b + \mathbf{x}^e)/2$, half the width of the region is $hw = (x_1^e - x_1^b)/2$, and half the height of the region is $hh = (x_2^e - x_2^b)/2$. The contribution function at a point $\mathbf{x} = (x_1, x_2) \in \mathbf{R}$ is defined as

$$a_f^{\mathbf{r}}(\mathbf{x}) = \left(\frac{hw - |x_1 - x_1^c|}{hw}\right)\left(\frac{hh - |x_2 - x_2^c|}{hh}\right) f(\mathbf{x}). \tag{7}$$

Apparently, the contribution of a point is region-dependent. Hence, the simple integral image approach presented in section 2 is not directly applicable here.

For simplicity of presentation, we consider only the case when $x_1 \ge x_1^c$ and $x_2 \ge x_2^c$. The other cases can be handled similarly. By manipulating equation 7, we obtain

$$\begin{aligned}
a_f^{\mathbf{r}}(\mathbf{x}) &= \left(\frac{hw - x_1 + x_1^c}{hw}\right) \times \\
&\quad \left(\frac{hh - x_2 + x_2^c}{hh}\right) f(\mathbf{x}) && (8)\\
&= \left(\frac{x_1^e - x_1}{hw}\right)\left(\frac{x_2^e - x_2}{hh}\right) f(\mathbf{x}) && (9)\\
&= \left(\frac{x_1^e x_2^e}{hw \times hh}\right) f(\mathbf{x}) - \\
&\quad \left(\frac{x_1^e}{hw \times hh}\right)(x_2 f(\mathbf{x})) - \\
&\quad \left(\frac{x_2^e}{hw \times hh}\right)(x_1 f(\mathbf{x})) + \\
&\quad \left(\frac{1}{hw \times hh}\right)(x_1 x_2 f(\mathbf{x})) && (10)\\
&= g_1^{\mathbf{r}} h_{1f}(\mathbf{x}) + g_2^{\mathbf{r}} h_{2f}(\mathbf{x}) + \\
&\quad g_3^{\mathbf{r}} h_{3f}(\mathbf{x}) + g_4^{\mathbf{r}} h_{4f}(\mathbf{x}), && (11)
\end{aligned}$$

where $h_{1f} = f(\mathbf{x})$, $h_{2f} = x_2 f(\mathbf{x})$, $h_{3f} = x_1 f(\mathbf{x})$, $h_{4f} = x_1 x_2 f(\mathbf{x})$, and $g_1^{\mathbf{r}}$ through $g_4^{\mathbf{r}}$ are the corresponding coefficients from expression 10.

Now, we have expressed the original contribution function as a linear combination of simpler functions, $h_{1f}$ through $h_{4f}$, with weighting coefficients $g_1^{\mathbf{r}}$ through $g_4^{\mathbf{r}}$. The interesting observation here is that all the $h$ functions are region-independent, and none of the $g$ coefficients depends on the point $\mathbf{x}$ or the function $f$, they only depend on the region's definition. We call functions such as the $g$ coefficients *point-independent*. Substituting equation 11 into the filtering function, equation 1, yields

$$\begin{aligned}
\mathcal{A}_f(\mathbf{r}) &= g_1^{\mathbf{r}} \sum_{\mathbf{x}\in\mathbf{R}} h_{1f}(\mathbf{x}) + g_2^{\mathbf{r}} \sum_{\mathbf{x}\in\mathbf{R}} h_{2f}(\mathbf{x}) + \\
&\quad g_3^{\mathbf{r}} \sum_{\mathbf{x}\in\mathbf{R}} h_{3f}(\mathbf{x}) + g_4^{\mathbf{r}} \sum_{\mathbf{x}\in\mathbf{R}} h_{4f}(\mathbf{x}) \ . && (12)
\end{aligned}$$

Equation 12 expresses the original filtering function as a linear combination of other filtering functions. Moreover,

all of the component filtering functions in this linear combination are region-independent. In fact, the linear combination obtained for the filtering function is exactly the same as the linear combination for the contribution function itself. Since each of the component filtering functions in equation 12 is region-independent, each can be computed efficiently using an integral image for its own contribution function. Then, by substituting the resulting values in equation 12, we obtain the desired filtering.

In summary, to use integral images in this example we express the desired region-dependent contribution function as a linear combination of several region-independent contribution functions. Then, the desired region-dependent filtering is easily computed as a linear combination of the corresponding region-independent filtering functions, which can be efficiently computed via integral images.

## 4. Kernel Integral Images

In this section, we treat the case of region-dependent filtering functions in a more formal way. Recall from the example of bilinear interpolation that the mechanism used to enable usage of integral images is expressing the filtering function as a linear combination of other region-independent filtering functions. To understand why this works, we rewrite the final form of the contribution function, equation 11, in a more compact form as

$$a_f^{\mathbf{r}}(\mathbf{x}) = < \mathbf{g}^{\mathbf{r}}, \mathbf{h}_f(\mathbf{x}) >, \quad (13)$$

where

$$\mathbf{g}^{\mathbf{r}} = \begin{bmatrix} g_1^{\mathbf{r}} \\ g_2^{\mathbf{r}} \\ g_3^{\mathbf{r}} \\ g_4^{\mathbf{r}} \end{bmatrix}, \quad (14)$$

and

$$\mathbf{h}_f(\mathbf{x}) = \begin{bmatrix} h_{1f}(\mathbf{x}) \\ h_{2f}(\mathbf{x}) \\ h_{3f}(\mathbf{x}) \\ h_{4f}(\mathbf{x}) \end{bmatrix}, \quad (15)$$

In other words, we can express the contribution function as a dot product of two vector functions: one of them is region-independent and the other is point-independent. This is actually a necessary and sufficient condition to express the filtering function as a linear combination of region-independent filtering functions. We outline the proof of this fact rather informally here. The sufficiency direction is straight forward following the same argument as in the bilinear interpolation example. Basically, by distributing the summation of the filtering function over terms of the dot product, as we did to obtain equation 12, sufficiency immediately follows. The necessity direction is derived as follows. Starting from the linear combination of filtering functions, as in equation 12, we can express the linear combination as a dot product. Then, by pulling the summation out, we obtain an expression of the contribution function that is a dot product of two parts, one of them is region-independent, and the other one is point-independent.

The dot product immediately reminds us of the kernel trick that is frequently used in machine learning, where feature vectors are implicitly transformed into a – typically – higher dimensional space by replacing a dot product by a kernel function that is equivalent to a dot product in the transformed space [2]. Since applying any transformation to the vectors $\mathbf{g}^{\mathbf{r}}$ and $\mathbf{h}_f(\mathbf{x})$, in equation 13, will not change their region-independence or point-independence natures, the condition we stated above still holds on the transformed vectors. Therefore, we can generalize the form of the contribution functions we consider to

$$a_f^{\mathbf{r}}(\mathbf{x}) = \mathbf{H}(\mathbf{g}^{\mathbf{r}}, \mathbf{h}_f(\mathbf{x})), \quad (16)$$

where $H$ is a kernel function, *i.e*. a function that computes a dot product between its two arguments possibly after mapping them to another dimensional space. We call this generalization of integral images *kernel integral images*. In our case, even if the kernel performs a dot product implicitly, to compute our filtering function we have to perform it explicitly. Sometimes, the kernel computes the dot product in an infinite dimensional space. In these cases, approximation of the dot product with a small number of terms may be sufficient for the application in hand. This point will be clarified when we use it in an example in section 5.

## 5. Filtering with Gaussian Weighting

In many applications of image feature filtering in computer vision, higher weights are given to pixels closer to the center of the filtering region and lower weights to pixels closer to the borders of the filtering region. That is applied, for example, in object tracking, *e.g*. [5], where higher weights are given to pixels that more likely belong to the object than the background. The same idea was shown to improve human detection performance in [4]. In both cases, the weighting function used is a Gaussian weighting function.

To simplify the mathematical treatment, we consider the one dimensional case. Consider a region $\mathbf{R}$ defined by the two limiting points $x^b$ and $x^e$. The center of $\mathbf{R}$ is defined as $x^c = (x^b + x^e)/2$. Denote the standard deviation of the Gaussian weighting function by $\sigma^{\mathbf{r}}$. The contribution function in this case can be defined as

$$a_f^{\mathbf{r}}(x) = e^{-\left(\frac{x-x^c}{\sigma^{\mathbf{r}}}\right)^2} f(x). \quad (17)$$

Clearly, the contribution function is region-dependent. Con-

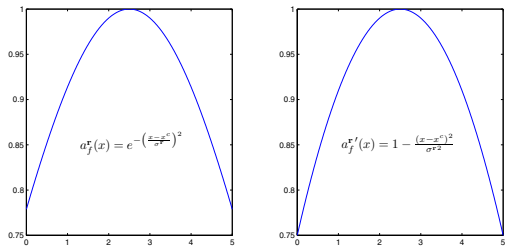


Figure 2. Comparison of the Gaussian weighting function and its approximation, equations 17 and 19, when the filtering region is between 0 and 5 and $\sigma^{\mathbf{r}}$ is 5.

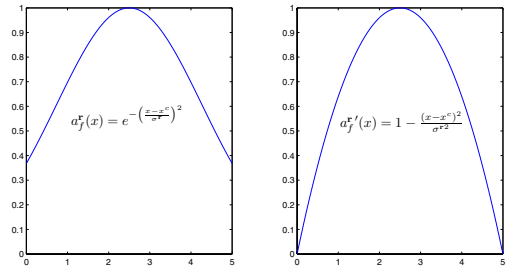


Figure 3. Comparison of the Gaussian weighting function and its approximation, equations 17 and 19, when the filtering region is between 0 and 5 and $\sigma^{\mathbf{r}}$ is 2.5.

sider the Euler expansion of equation 17

$$a_f^{\mathbf{r}}(x) = \sum_{i=0}^{\infty} \frac{(-1)^i (x - x^c)^{2i}}{\sigma^{\mathbf{r}\,2i} i!} f(x). \tag{18}$$

Equation 18 can be viewed as a dot product in an infinite dimensional space between two vector functions one of them is region-independent and the other one is point-independent. (To see this, consider expanding the expression $(x-x^c)^{2i}$ in each term of the power series.) Hence, the kernel integral image method applies. But, it requires computation of an infinite number of integrals. However, we can approximate the contribution function by taking a few of the initial terms of the expansion. For example, taking the first two terms only, we obtain the contribution function

$$a_f^{\mathbf{r}\prime}(x) = \left(1 - \frac{(x - x^c)^2}{\sigma^{\mathbf{r}\,2}}\right) f(x). \tag{19}$$

This approximation is valid, *i.e.* does not give negative weights, as long as $\sigma^{\mathbf{r}}$ is selected so that $\frac{(x-x^c)^2}{\sigma^{\mathbf{r}\,2}} \leq 1$. Figures 2 and 3 show plots of the original Gaussian weighting function, equation 17, and its approximation, equation 19, when $x^b = 0$, $x^e = 5$, and $\sigma^{\mathbf{r}} = 5$ and 2.5, respectively. In the case of $\sigma^{\mathbf{r}} = 5$ plots are very similar. However, for the case of $\sigma^{\mathbf{r}} = 2.5$, the difference is quite large. For applications that need weighting of pixels with respect to one another so that pixels closer to the center get more importance, the difference between the two functions – in case the selected value of $\sigma^{\mathbf{r}}$ makes a difference – is not expected to be important. In general, whether the approximation is accurate enough or not, and whether it is worth using more terms of the expansion to achieve higher accuracy or not, depends on the value of $\sigma^{\mathbf{r}}$ and on the application itself.

## 6. Kernel Integral Images vs. Repeated Integration

Heckbert [6] presented an elegant method, called *filtering by repeated integration*, to extend usage of precomputed integrals to more complex filters. For completeness of presentation, we briefly compare our method to his method. For details, please refer to [6].

Heckbert's approach is based on the fact that more complex filters can be constructed by convolving a box-filter with itself. For example, if we convolve a box filter with itself once, we obtain a triangular filter, which is very similar to filtering with bilinear interpolation in two dimensions. If we convolve a box filter with itself twice, we obtain a quadratic filter, which is similar to the approximation we use for Gaussian filters. In fact, convolution of a box filter with itself an infinite number of times produces the Gaussian filter. Suppose that we want to use a filter that is generated by convolving a box filter with itself $n$ times. Heckbert's approach is based on the fact that convolution with such a filter is equivalent to integrating the image $n$ times and then convolving the $n^{th}$ integral with the $n^{th}$ derivative of the filter. The $n^{th}$ derivative of such a filter turns out to be a simple sparse filter, which is very efficient to convolve with.

The main drawback of the repeated integration approach is integrating the image several times. The required precision to represent the integration values grow linearly with the number of integrations [7]. In our approach, we compute integrals of several functions. But, each is integrated only once. For example, in approximating a Gaussian filter by a quadratic filter, the repeated integration method requires integrating the image three times consecutively, while kernel integral images requires computing nine independent integrals. Experimentally, kernel integral images in this case produces smaller numerical errors using the standard double-precision floating point number representation, as we show in section 7.3.

Another advantage of our approach is that the integrals computed are independent of one another. That allows for parallel computation of the integrals.

## 7. Experimental Results

### 7.1. Implementation Details

We evaluated our approach in terms of speedup by comparing to the conventional filtering approach (equation 1). We implemented filtering with bilinear interpolation, and filtering with approximate Gaussian weighting. Both are implemented in two dimensions.

For bilinear interpolation, equation 11 in section 3 considers only the case where $x_1 \geq x_1^c$ and $x_2 \geq x_2^c$. If we consider the origin at the lower left corner of the filtering domain, then equation 11 considers only the case of the top right quadrant of the filtering region. Figure 4 lists coefficients of different terms for the four quadrants.

In order to perform fast filtering in this case, we compute four different integral images, one for each of the contribution functions. The integration itself is conducted in four steps, since each region's quadrant has a different coefficient for each of the integrals, as shown in figure 4.

For the case of approximating Gaussian weighting in two dimensions, by expanding equation 19 and extending the notation to two dimensions, we obtain

$$a_f^{\mathbf{r}\prime}(\mathbf{x}) = \left[\left(1 - \frac{{x_1^c}^2}{{\sigma^{\mathbf{r}}}^2}\right) + \frac{2x_1^c}{{\sigma^{\mathbf{r}}}^2}x_1 - \frac{1}{{\sigma^{\mathbf{r}}}^2}x_1^2\right] \times \left[\left(1 - \frac{{x_2^c}^2}{{\sigma^{\mathbf{r}}}^2}\right) + \frac{2x_2^c}{{\sigma^{\mathbf{r}}}^2}x_2 - \frac{1}{{\sigma^{\mathbf{r}}}^2}x_2^2\right] f(\mathbf{x}). \quad (20)$$

Hence, to perform fast filtering, we compute nine integral images. These are integral images for the contribution functions: $f(\mathbf{x})$, $x_1 f(\mathbf{x})$, $x_2 f(\mathbf{x})$, $x_1 x_2 f(\mathbf{x})$, $x_1^2 f(\mathbf{x})$, $x_2^2 f(\mathbf{x})$, $x_1 x_2^2 f(\mathbf{x})$, $x_1^2 x_2 f(\mathbf{x})$, and $x_1^2 x_2^2 f(\mathbf{x})$. The coefficient of each region-independent filtering function can easily be obtained from equation 20. Unlike the case of bilinear interpolation, there is no need to handle each region quadrant separately since they all have the same coefficients.

### 7.2. Running Time Analysis

In the two filtering examples, the function filtered on, $f(\mathbf{x})$, is the intensity at point $\mathbf{x}$. Since intensity values do not affect the computation time, we generate images with a constant intensity value. Generated images are squares that differ in the number of pixels, *i.e.* area. Generated image areas range from 10000 to 200000 pixels, with an increment of 10000 pixels.

Each image is scanned with sampled region sizes and locations. The minimum region side length was set to 5 pixels, with side length increment of 5 pixels. Images are scanned with each region size in all possible locations with increments of 5 pixels in both directions. For each region, the two filtering types are computed using integral images and using conventional filtering. For each image, two time periods are measured: 1) the time to set up necessary structures, that is integral images or just type conversion when

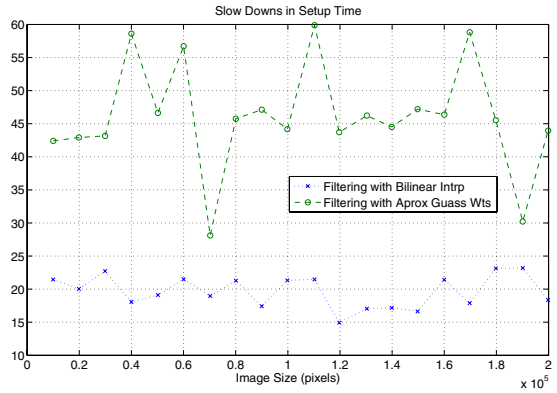


Figure 5. The slow down in setting up integrals vs the naive set up of conventional approaches.

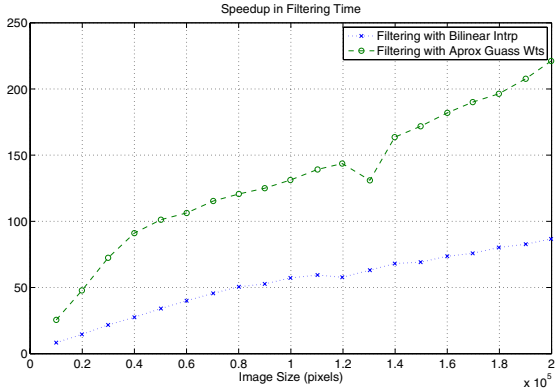


Figure 6. Speedups of using integral images compared to conventional method. These plots consider speedups in filtering time only.

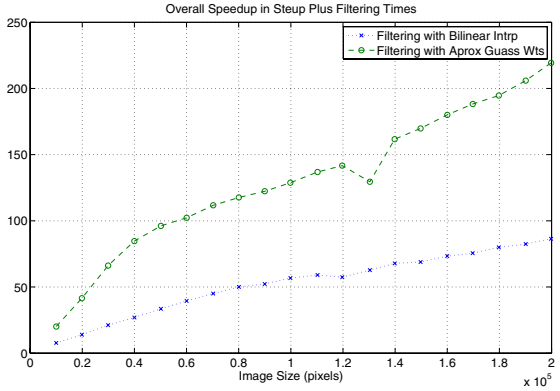


Figure 7. Speedups of using integral images compared to conventional method. These plots consider speedups when adding construction time to filtering time.

| | $f(\mathbf{x})$ | $x_2 f(\mathbf{x})$ | $x_1 f(\mathbf{x})$ | $x_1 x_2 f(\mathbf{x})$ |
|---|---|---|---|---|
| Top Right Quadrant | $x_1^e x_2^e$ | $-x_1^e$ | $-x_2^e$ | $1$ |
| Top Left Quadrant | $-x_1^b x_2^e$ | $x_1^b$ | $x_2^e$ | $-1$ |
| Lower Right Quadrant | $-x_1^e x_2^b$ | $x_1^e$ | $x_2^b$ | $-1$ |
| Lower Left Quadrant | $x_1^b x_2^b$ | $-x_1^b$ | $-x_2^b$ | $1$ |

Figure 4. Coefficients of different contribution functions in the case of bilinear interpolation, equation 7, for the four region quadrants. All coefficients in the table have to be normalized by dividing by $hw \times hh$

the conventional filtering is used, 2) and the time to scan the image and compute filtering over all scanned regions.

The plots in figure 5 show the slow-downs in the setup time. In the case of bilinear interpolation, the slow down is around 20x, and in the case of approximate Gaussian weighting, it is around 45x. On the other hand, figure 6 shows the speedups obtained when considering only the time to scan the image and evaluate the filtering function at all probed regions. The speedups are monotonically increasing with the image size. For an image size of 200000 pixels, we achieve a speedup of around 90x in the case of bilinear interpolation, and 220x in the case of approximate Gaussian weighting. This shows the significant benefit of using our approach, especially in the case of Gaussian weighting. Therefore, despite the complexity of computing more integral images during setup, filtering with Gaussian weighting benefits more from using integral images. Finally, figure 7 shows speedups when adding the setup and filtering times together. The curves in this figure look very similar to the curves in figure 7, which consider speedups on filtering time only. This shows that in the two weighting schemes evaluated, the setup time is almost negligible with respect to the filtering time.

### 7.3. Relative Error Analysis

In this set of experiments, we evaluate the two fast filtering methods, kernel integral images and repeated integration, in terms of their relative error. The error we measure here is the difference between the value computed by a fast filtering method and the value computed by conventional filtering (equation 1). The relative error is the ratio between this difference and the value computed by conventional filtering.

We generate 10 random images of size $1024 \times 1024$. We evaluate the filtering function on a region of size $31 \times 31$ at all possible locations in the image. For each location we compute the relative error and plot relative error values against the distance from the region's top-left corner to the image's top-left corner. The distance measure we use is the area of the rectangle bounded by these two corners. This distance measure is equivalent to the number of feature points that are added to produce the integral value(s) associated with the region's top left corner. The error is expected to increase with this distance measure.

In the case of bilinear interpolation, relative errors are always zeros, but not so for approximate Gaussian weighting. The problem with the approximate Gaussian weighting is the integration of higher order contribution functions, such as $x_1^2 x_2^2 f(x)$. These contribution functions require higher precision to represent. Their integrals require even higher precision that is outside the range the double-precision floating point representation. Figure 8 shows a third-degree polynomial fit of the relative errors in the case of approximate Gaussian weighting using kernel integral images. The figure compares two methods of computing integrals in terms of the error they produce. The one-pass method scans the image once and computes the value of the integral at a pixel as a function of its three preceding pixels. The two-pass approach scans the image twice: once integrating horizontally and once vertically. The error generally increases with the distance from the origin. The two-pass method produces around an order of magnitude lower error than the one-pass method. That is expected since in the one-pass method, numbers grow more rapidly allowing for larger errors when adding two numbers that differ by many orders of magnitude.

Figure 8 also shows the relative errors, using two-pass integration, of the repeated integration method when used to approximate Gaussian filters with a quadratic filter. The error of our approach, even when using one-pass integration, is lower than the error of the repeated integration method. Similar to our approach, the repeated integration method produces no errors when applied to bilinear interpolation filtering.

In these experiments we use non-negative numbers to represent intensity and pixel coordinate values. These values can be linearly mapped to allow for both negative and positive numbers. In this way, the effective precision used can be increased by utilizing the sign bit in the binary representation, and therefore the accuracy can be enhanced, as shown in [7].

## 8. Conclusion

We presented an extension to the integral image framework that allows for fast filtering under non-unform region-dependent weighting of feature values. We refer to the extended framework as kernel integral images. To show the utility of the extension, we provided two examples of

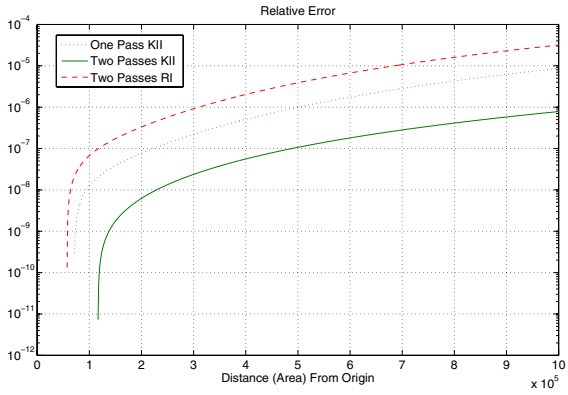


Figure 8. Relative errors of computing Gaussian weighted filtering as a function of distance (area) to the origin. KII stands for Kernel Integral Images. RI stands for Repeated Integration.

widely used non-uniform filtering: one that can be implemented exactly via our framework, that is filtering with bilinear interpolation, and one that can be approximated, which is filtering with Gaussian weighting. Our experiments show that using our approach, significant speedups can be achieved. The presented technique provides a higher precision and more suitability for parallel implementation than the repeated integration approach [6], which also extended the integral images framework for complex filters.

## Acknowledgment

This work was funded, in part, by Army Research Laboratory's Robotics Collaborative Technology Alliance program; contract number DAAD 19-012-0012 ARL-CTA-DJH.

## References

[1] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, 2006.

[2] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley, 1998.

[3] F. Crow. Summed-area tables for texture mapping. *Computer Graphics (ACM SIGGRAPH)*, 18(3):207–212, 1984.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[5] A. Elgammal, R. Duraiswami, and L. Davis. Efficient computation of kernel density estimation using fast gauss transform with applications for segmentation and tracking. In *Second International Workshop on Statistical and Computational Theories of Vision*, 2001.

[6] P. Heckbert. Filtering by repeated integration. *ACM SIGGRAPH*, 20(4):315–321, 1986.

[7] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. *EUROGRAPHICS*, 24(3):547–555, 2005.

[8] D. Lowe. Distinctive image features from scale-invariant keypoints. *nternational Journal of Computer Vision*, 60:91–110, 2004.

[9] F. Porikli. Integral histogram: A fast way to extract histogram features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[10] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *European Conference on Computer Vision (ECCV)*, 2006.

[11] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.

[12] S. K. Zhou, R. Chellappa, and B. Moghaddam. Visual tracking and recognition using appearance-adaptive model in particle filters. *IEEE Transactions on Image Processing*, 13(11):1491–1506, November 2004.

[13] Q. Zhu, S. Avidan, M.-C. Yeh, and K.-T. Cheng. Fast human detection using a cascade of histograms of oriented gradients. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, New York, June 2006.