2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Stacked Integral Image

Amit Bhatia, Wesley E. Snyder and Griff Bilbro

Abstract— **Filtering in digital images via Integral Image yields fast computation times for uniform filtering. The extension by Heckbert [1] to perform filtering via repeated integration provides a way for non-uniform filtering, but has its own limitations. A recent method by Hussein et al. [2] provides non-uniform filtering by Euler expansion of filtering kernels, and is called kernel integral method. We propose a simplification for non-uniform filtering by stacking of box filters from a single integral image.** [1] **Results show speedups of as much as 40:1, similar to run time performance gains in kernel integral method, when comparing to naïve non-uniform filtering, while at the same time reducing the setup time drastically.**

Keywords: Integral Image, Kernel, Box Filter

I. Introduction

The process of filtering in image processing applications is a significant sink for computer time, especially in real-time visual control of robotic systems and target tracking.

For uniform filtering the concept of Summed Area Tables, as introduced by Crow [3], provides constant time results. This idea of filtering was generalized as Repeated Integration by Heckbert [1] which shows that convolution of a signal with any piecewise polynomial kernel of degree $n-1$ can be computed by integrating the signal n times and point sampling it several times for each output sample. But there is a cost associated with integrating the image n times.

In computer vision, the idea of summed area tables was introduced as Integral Images by Viola et al. [4] to compute rectangular features. For rectangular regions, the filtering is achieved in constant time using box filters. This idea was used for computing integral histograms by Porikli in [5] and for covariance matrices in [6]. It was used by Zhu et al. [7] for fast computation of histogram of oriented gradients.

Most of the usage of Integral Images listed above pertains to uniform filtering using box filters. But some applications perform better using non-uniform filtering. For example Dalal et al. [8] use bilinear interpolation to enhance their feature detector. In kernel based object tracking [9], Comaniciu et al. use kernel based non uniform filtering for mean shift tracking. For such applications, the non-uniform filter needs to be applied recursively at overlapping regions of the image to match against a given model. This tends to become very costly computationally as the naïve method performs direct convolution at every pixel in the image. Hence, Zhu et al. [7] bypass the non-uniform (Gaussian) weighting to speed up their detection algorithm. But, as pointed out by Dalal et al. [8], using weighting schemes gives much better results.

[1]This work was partly supported by AFOSR under grant no. FA9550-07-1-0176 and ARO under grant no. W911NF-05-1-0330.

A recent method proposed by Hussein et al. [2] provides non-uniform filtering by splitting the filtering function into region-independent and point-independent functions. They resolve the filtering function for bilinear interpolation and Gaussian weighting kernels, using a linear combination of integral images. This method is called kernel integral image. They show that this method provides exact weights for bilinear interpolation and approximate weights for Gaussian filter. There is a performance penalty in the setup time while a performance gain in the actual filtering time, when the kernel integral image method is compared with the direct convolution method.

In this paper we propose to reduce the setup time by stacking multiple box filters from a single integral image. The idea closest to this can be seen in [10], where Bay et al. use integral images to approximate Gaussian derivatives, for feature detection and description. Our paper is organized as follows. In section 2, the basic idea of integral image and filtering by repeated integration is revisited. In section 3, the kernel integral method is summarized. In section 4, the proposed stacked integral image method is described followed by box stack calculation in section 5 and experiment results in section 6. Section 7 describes using the proposed method for multi-stage detection.

II. Integral Image and Repeated Integration

The idea of Integral Image [4] is the same as that of Summed Area Tables [3]. Given a feature of interest f, e.g. intensity, at every point in an image, the feature of any arbitrary rectangular region in the image can be computed using four operations on the integral of an image.

The integral of an image at any point (x, y) in a sampled image represents the sum of feature values from the origin up to and including the point (x, y). Once this integral image is built, the sum of values within any region is computed as follows:

As shown in figure 1, the rectangle of interest is bounded by $(x1, y1), (x2, y2), (x3, y3), (x4, y4)$. If $I(x, y)$ represents the value of the feature in the image at point (x, y) then the corresponding Integral Image value $II(x, y)$ is given as:

$$II(x,y) = \sum_{x' \leq x, y' \leq y} I(x', y') \tag{1}$$

Using the integral image, any rectangular sum can be computed in four array references, as shown in (2) below.

$$\begin{aligned} I(Rect) &= I[(x1,y1),(x2,y2),(x3,y3),(x4,y4)] \\ &= II(x4,y4) - II(x3,y3) \\ &\quad -II(x2,y2) + II(x1,y1) \end{aligned} \tag{2}$$

Fig. 1. Integral Image

Using the above concept, Heckbert [1] proposed to build more complex filters by repeated integration. The main idea being that repeated convolution of a box filter by itself generates higher order filters: triangular filter, parabolic/quadratic filter, parzen/cubic filter. As the number of convolutions increase, the box filter approaches a Gaussian filter. In Heckbert's approach, using a filter which is produced by convolving a box filter n times, is equivalent to first integrating the image n times and then convolving the n^{th} integral image with the n^{th} derivative of the filter. Since n^{th} derivative of n^{th} order box filter is a train of impulses, the final convolution amounts to evaluation of the n^{th} integral image at the selected points. Although this approach is attractive, it involves the repeated integration of the image before the final simple convolution takes place. Also, the precision required to represent the integration values grows with n.

III. Kernel Integral Image

In [2] Hussein et al. proposed an extension to Integral Images, by separating the filtering function into *region-independent* part and *point-independent* part.

A filtering of values of a feature function f (e.g. intensity) over a region r is defined as a mapping $A_f : r \rightarrow \Re$ which maps the region to a real value. This can be expressed as:

$$A_f(r) = \sum_{\mathbf{x} \in r} a_f^r(\mathbf{x}) \tag{3}$$

where the contribution function $a_f^r(\mathbf{x})$ defines contribution of point $\mathbf{x}$ to filtering of f over region r, and hence depends on three things: point coordinates, function f and region r (i.e. extreme points of the region).

When the contribution function is region-independent, such as $a_f(\mathbf{x}) = f(\mathbf{x})$ or $a_f(\mathbf{x}) = \|\mathbf{x}\| f^2(\mathbf{x})$, then the integral image is calculated easily as $II_f(\mathbf{x}) = \sum_{y_i < x_i} a_f(\mathbf{y})$.

For region dependent contribution function, such as for bilinear interpolation, denote a rectangular region r bounded by points $\mathbf{x}^b = (x_1^b, x_2^b)$ and $\mathbf{x}^e = (x_1^e, x_2^e)$ on the top left and bottom right corner of the rectangle, with center $\mathbf{x}^c = (\mathbf{x}^b + \mathbf{x}^e)/2$, and half width and half height as $hw = ((x_1^e - x_1^b)/2)$ and $hh = ((x_2^e - x_2^b)/2)$, where b implies begin point, e implies end point, and c implies center point. The filtering contribution of f at a point $\mathbf{x} = (x_1, x_2)$ is given as:

$$a_f^r(\mathbf{x}) = \left(\frac{hw - |x_1 - x_1^c|}{hw}\right)\left(\frac{hh - |x_2 - x_2^c|}{hh}\right) f(\mathbf{x}) \tag{4}$$

By separating variables, this simplifies to:

$$a_f^r(\mathbf{x}) = g_1^r h_1 + g_2^r h_2 + g_3^r h_3 + g_4^r h_4 \tag{5}$$

where, $h_1 = f(\mathbf{x})$, $h_2 = x_2 f(\mathbf{x})$, $h_3 = x_1 f(\mathbf{x})$, $h_4 = x_1 x_2 f(\mathbf{x})$ are region independent functions and $g's$ are the respective point independent coefficients (see [2]): $g_1^r = ((x_1^e x_2^e)/(hw \times hh))$, $g_2^r = (x_1^e/(hw \times hh))$, $g_3^r = (x_2^e/(hw \times hh))$, $g_4^r = (1/(hw \times hh))$. Hence, the original filtering function from (4), is replaced by a linear combination of other filtering functions. The h functions are region-independent while the g coefficients are point-independent. For the case of bilinear interpolation, this combination of other functions is an exact replica of the original filter.

Similarly, for the case of Gaussian filtering, the filter function contribution in 1-D at point x is given as:

$$a_f^r(x) = e^{-(\frac{x - x^c}{\sigma})^2} f(x) \tag{6}$$

Expanding the above (6) using Euler expansion, and choosing only the first two terms gives the approximate contribution function as:

$$a_f'^r(x) = \left(1 - (\frac{x - x^c}{\sigma})^2\right) f(x) \tag{7}$$

which in 2-D becomes the approximation:

$$\begin{aligned} a_f'^r(\mathbf{x}) &= \left[\left(1 - (\frac{x_1^c}{\sigma})^2\right) + 2\frac{x_1^c}{\sigma^2} x_1 - \frac{x_1^2}{\sigma^2}\right] \times \\ &\quad \left[\left(1 - (\frac{x_2^c}{\sigma})^2\right) + 2\frac{x_2^c}{\sigma^2} x_2 - \frac{x_2^2}{\sigma^2}\right] f(\mathbf{x}) \end{aligned} \tag{8}$$

This requires generation of the following nine integral images: $f(\mathbf{x})$, $x_1 f(\mathbf{x})$, $x_2 f(\mathbf{x})$, $x_1 x_2 f(\mathbf{x})$, $x_1^2 f(\mathbf{x})$, $x_2^2 f(\mathbf{x})$, $x_1 x_2^2 f(\mathbf{x})$, $x_1^2 x_2 f(\mathbf{x})$ and $x_1^2 x_2^2 f(\mathbf{x})$.

Hence, using (5) for bilinear interpolation with four integral images and using (8) for Gaussian filter with nine integral images, the authors provide a fast method of non-uniform filtering, using integral images.

IV. Stacked Integral Image

Although by using kernel integral images, there is substantial improvement in the run-time of computing the convolution, as compared to the run-time of naïve direct convolution, there is a constant cost in generating the multiple integral images. Although this setup cost (four integral images for bilinear interpolation and nine integral images for Gaussian filter) starts diminishing with increasing size of the image, it would be even better if this setup cost is reduced too.

To reduce this constant setup cost in computing multiple integral images, we propose that a set of appropriately chosen box filters with corresponding weights would closely

approximate the desired non-uniform filter. By using integral images, the value of the each box can computed using just four operations as shown in equation (2). A single non-uniform filter is then approximated as a stack of N boxes of different sizes (length, width of box) and weights (height of box). For each non-uniform filter, the number of boxes and their sizes and weights can be precomputed. When the filter needs to be applied, only a *single* integral image needs to be computed and each box value can be computed using at most four integral image lookups per box. The number of boxes in the stack, N, can vary depending on the accuracy of the approximation desired. A larger number of boxes produces greater accuracy, but four more lookups would be required to compute each additional box weight.

For the case of Gaussian 1-D filter, figure 2 shows filter approximation using 3 boxes. The left side shows the value of the Gaussian function at the box boundaries. The right side show the value of the box weights, defined to be the difference between the Gaussian value of the current box and the boxes below it, i.e. incremental weight.

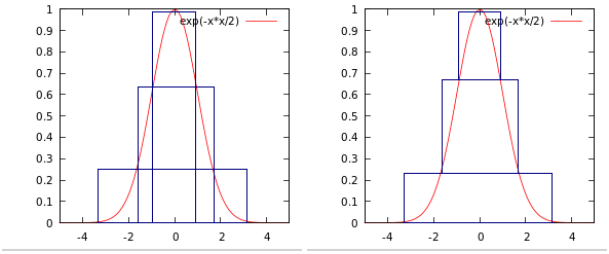


Fig. 2. Gaussian 1D filter: 3 box approximation

Figure 3 shows filter approximation of Gaussian 1-D filter using 4 boxes. As is evident from the figures 2 and 3, the number of parameters for box sizes and weights increase with N, the number of boxes parameter.

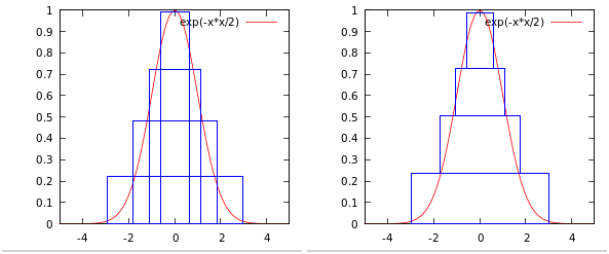


Fig. 3. Gaussian 1D filter: 4 box approximation

In 1-D, the parameters to be decided for each box are its width and height. In 2-D, the parameters to be decided for each box are its length, width and height. Hence, each n-D box has $n+1$ parameters that need to be chosen.

By using a precalculated stack of N boxes, their sizes and their weights, a non-linear filter can be approximated quickly by summing over the corresponding boxes in the *single* integral image. We call this method *Stacked Integral Image* since it builds a filter using a stack of box filters evaluated on a *single* integral image. Compared to the kernel integral image approach, the stacked integral image uses only one integral image, and hence saves on the setup cost. When the kernel integral image approximates a Gaussian filter by two terms of an Euler expansion, it leads to computation of nine integral images. The stacked approach on the other hand increases its accuracy by having more box filters, all using the same integral image. The main cost of the stacked integral image approach for Gaussian filter is the calculation of the box sizes and weights, which can be precomputed and is explained in the next section.

V. Box Stack Calculation

A. Function Minimization

The parameter N decides how many boxes are to be used to approximate a kernel filter. Denote the half width, half height and weight of a box i by w_i, h_i and a_i respectively. Given an origin centered region R with half width and half height as w and h, then the function to minimize is:

$$J = \sum_{x=0}^{w}\sum_{y=0}^{h}\left[K(x,y) - \sum_{i=0}^{N-1}[H(w_i - x)H(h_i - y)a_i]\right]^2 \tag{9}$$

where, $K(x,y)$ is the value of the kernel filter (typically Gaussian) at (x,y). The Heaviside step function H in (9) ensures that only those boxes that cover the given pixel (x,y), have their weight added, where $H(y) = 0$ for $y < 0$, and $H(y) = 1$ for $y >= 0$. Minimization of J by any appropriate numerical method produces the desired parameter values.

B. Computational Complexity

Given a filter size (F_w, F_h) to be applied on an image of size (I_w, I_h), the box stack can be precomputed once and for all, for the size (F_w, F_h). Having N boxes with given sizes and weights, the convolution can be done very quickly. First the single integral image is computed on the input image. Then a weighted sum is computed by simply adding the uniform rectangle filter weights computed for each box size, using equation (2). The computational cost of the kernel and stacked integral methods for the case of a Gaussian filter are compared below. Similar comparisons can be done for other kernel filters.

As done in [4], computing the integral image $ii(x,y)$ of an image $i(x,y)$, using cumulative sum $s(x,y)$, is:

$$s(x,y) = s(x,y-1) + i(x,y) \tag{10}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{11}$$

Hence, using cost of addition/subtraction and multiplication as A and M respectively, the cost for computing equations (10) and (11) is $I_w I_h A$ each. So, total cost to compute an integral image is $2I_w I_h A$.

Since the integral images in kernel integral method involve multiplication of image values too, their respective costs are given in table I. So, total setup cost, for kernel integral

Integral	Cost	Integral	Cost
$f(\mathbf{x})$	$I_w I_h(2A)$	$x_1 f(\mathbf{x})$	$I_w I_h(2A+M)$
$x_2 f(\mathbf{x})$	$I_w I_h(2A+M)$	$x_1 x_2 f(\mathbf{x})$	$I_w I_h(2A+2M)$
$x_1^2 f(\mathbf{x})$	$I_w I_h(2A+2M)$	$x_2^2 f(\mathbf{x})$	$I_w I_h(2A+2M)$
$x_1 x_2^2 f(\mathbf{x})$	$I_w I_h(2A+3M)$	$x_1^2 x_2 f(\mathbf{x})$	$I_w I_h(2A+3M)$
$x_1^2 x_2^2 f(\mathbf{x})$	$I_w I_h(2A+4M)$		

TABLE I

SETUP COST: (GAUSSIAN FILTER) KERNEL INTEGRAL IMAGE

method, is $KII = 18 I_w I_h (A+M)$. Filtering cost, using kernel integral method, at every pixel in an image involves a single rectangle lookup, with 3 additions/subtractions, whose cost is $3AI_wI_h$.

For a stacked integral image, the single integral image cost is $SII = 2I_wI_hA$. Filtering cost, for stacked integral method, at every pixel in an image involves N rectangle lookups, with 3 additions/subtractions each, whose cost is $3ANI_wI_h$.

Hence, total computation cost for kernel integral image is $KII_{tot} = [18I_wI_h(A+M) + (3AI_wI_h)]$, while that for stacked integral image is $SII_{tot} = [(2I_wI_hA) + (3ANI_wI_h)]$. Since, typically a small number of boxes, N, can *approximate* a given filter, the cost of stacked integral image SII_{tot} would be lower than kernel integral image cost KII_{tot}.

The setup demonstrated here is a 2D Gaussian approximation of a filter of size 7x5, with the number of boxes ranging from 3 to 6. A numerical search for the box sizes and weights yields the results as given in the next section.

VI. RESULTS

A. Finding optimal box sizes and weights

Numerical methods were used to solve the parameter values (length, width, height) of each box in the stack. The problem is considered in a 2-D setting. The choice of number of boxes N per stack is varied between 3 and 5. The results for filter dimension 7x5 are given below.

```
* Filter size: 7x5, Num. boxes N=3
Box 3: 7 x 5 Weight : 0.004409
Box 2: 5 x 1 Weight : 0.075215
Box 1: 3 x 3 Weight : 0.033750
Error=0.256872
* Filter size: 7x5, Num. boxes N=4
Box 4: 7 x 5 Weight : 0.005482
Box 3: 5 x 1 Weight : 0.057108
Box 2: 3 x 3 Weight : 0.049716
Box 1: 1 x 3 Weight : 0.025052
Error=0.187201
* Filter size: 7x5, Num. boxes N=5
Box 5: 7 x 5 Weight : 0.006245
Box 4: 5 x 3 Weight : 0.012192
Box 3: 3 x 3 Weight : 0.036766
Box 2: 3 x 1 Weight : 0.058444
Box 1: 1 x 3 Weight : 0.030773
Error=0.131658
```

A plot of how the error changes for filter size 7x5, as the size of box 2 and box 1 are changed for the case of N=3 boxes, is shown in figure 4. The axis values are compressed so that a value e.g. 35 on the axis implies a box size of 3x5.

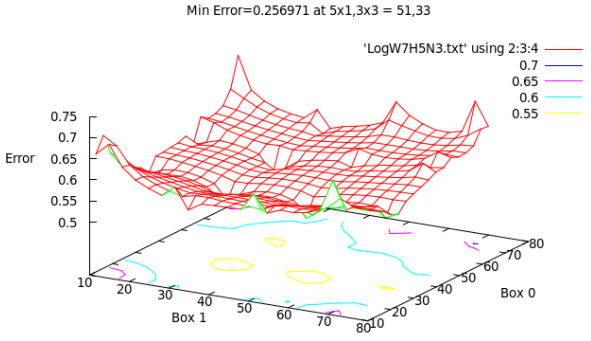


Fig. 4. Error surface for N=3 for different sizes of box 2 and box 1. Box 3 at size 7x5.

The results for filter dimension 55x115 are given below.

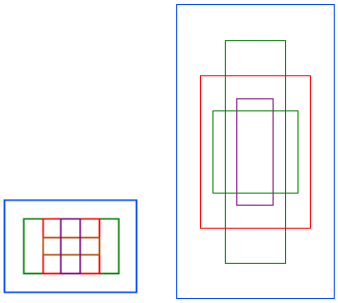
```
* Filter size: 55x115, Num. boxes N=3
Box 3: 55 x 115 Weight : 0.000032
Box 2: 33 x 41 Weight : 0.000299
Box 1: 19 x 65 Weight : 0.000320
Error=0.344631
* Filter size: 55x115, Num. boxes N=4
Box 4: 55 x 115 Weight : 0.000017
Box 3: 17 x 33 Weight : 0.000306
Box 2: 23 x 79 Weight : 0.000195
Box 1: 37 x 47 Weight : 0.000210
Error=0.275832
* Filter size: 55x115, Num. boxes N=5
Box 5: 55 x 115 Weight : 0.000019
Box 4: 21 x 35 Weight : 0.000273
Box 3: 13 x 81 Weight : 0.000171
Box 2: 31 x 59 Weight : 0.000174
Box 1: 43 x 47 Weight : 0.000091
Error=0.233528
```

The results for filter dimension 13x25 are given below.

```
* Filter size: 13x25, Num. boxes N=3
Box 3: 13 x 25 Weight : 0.000656
Box 2: 5 x 15 Weight : 0.005521
Box 1: 7 x 9 Weight : 0.005915
Error=0.334226
* Filter size: 13x25, Num. boxes N=4
Box 4: 13 x 25 Weight : 0.000475
Box 3: 7 x 15 Weight : 0.003557
Box 2: 5 x 11 Weight : 0.005673
Box 1: 9 x 7 Weight : 0.002543
Error=0.262754
* Filter size: 13x25, Num. boxes N=5
Box 5: 13 x 25 Weight : 0.000233
Box 4: 5 x 19 Weight : 0.002917
Box 3: 3 x 9 Weight : 0.005120
Box 2: 7 x 7 Weight : 0.003945
```

```
Box 1: 9 x 13 Weight : 0.002697
Error=0.200619
```

A visual layout for 7x5 and 13x25 filter sizes, using N=5 is shown in figure 5.



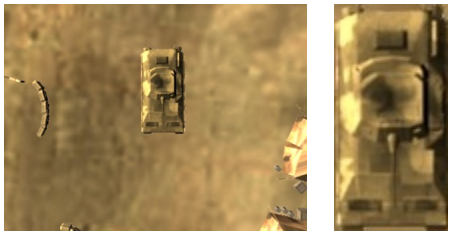
(a) 7x5 SII stack (b) 13x25 SII stack

Fig. 5. Stack layout for 7x5 and 13x25 filter sizes

B. Filtering results

1) Experiment 1: The stacked integral image procedure was applied to perform Gaussian filtering using filters and images of various sizes. The application of this method demonstrated here is the case of matching a model image in a scene image using filter weighted histograms. The color histogram was used for matching and filtering was done to match a model in a scene. Using precalculated stack of box sizes and weights, the scene is searched at every pixel, for the best matching model location. Thus this method can be used for numerous other applications by replacing the filtering function with a stack of box filters.

A sample of one such experiment is shown in figure 6 which shows the model (55x115) and scene (400x300). A single integral image is used to build the stack and the non-uniform filtering is achieved using the precomputed box sizes and weights.



(a) Scene (b) Model

Fig. 6. Sample scene and model

2) Comparison with Direct Convolution: The timing for the Stacked Integral Image (SII) method was compared with direct convolution, to calculate the speed improvement achieved. Using different filter (model) sizes and scene image sizes gave different improvements. Table II lists the timings for some experiments conducted with stacked integral image and direct convolution method. In the table, "Mw" and "Mh" represent width and height of model image, i.e. size of filter, "Sw" and "Sh" represent width and height of scene image, "Direct" shows computation time in seconds using the direct convolution method, while "Stack" represents time using the stacked integral method, and "Gain" shows the speed improvement of stacked integral method over the direct convolution method.

Mw	Mh	Sw	Sh	Direct conv.	Stacked conv.	Gain
17	27	400	300	9.797	1.859	5.270
19	38	600	450	46.80	5.656	8.270
25	46	400	300	29.11	2.634	11.05
50	44	800	600	245.4	11.91	20.59
57	116	400	300	112.5	4.080	27.57
89	83	800	618	729.9	18.46	39.52

TABLE II

SPEED IMPROVEMENT OVER DIRECT CONVOLUTION.

As a function of the filter size, the speed improvement in the convolution time is shown in figure 7. To get a feel of order of speed improvement, for a filter size of 89x83 applied on an image of size of 800x618, the speed improvement of the stacked integral method is 40 times over naïve direct convolution.

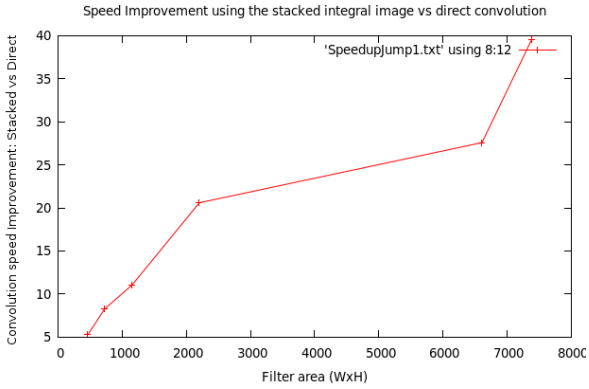


Fig. 7. Speed Improvement using the stacked integral image vs. direct convolution

3) Comparison with Kernel Integral Image method and Direct Convolution: To compare the performance gain of the proposed Stacked Integral Image (SII) method over Kernel Integral Image (KII), the matching for model/scene from 6(a) was done using all the three methods. The matching was done at decreasing density, i.e. first the convolution is done at every pixel, then in the next run at every second pixel, then in the next run at every third pixel and so on. The results of running this set of experiments are shown in figure 8. The figure shows that when convolution is done at every pixel, both the SII and KII methods have many fold improvement over Direct convolution. But when the density of convolution decreases, i.e. not done at every pixel, then the setup cost of computing the integral images becomes significant. This is where SII scores better over KII, since only a single integral image needs to be computed.

The results from a similar experiment with scene size 400x300 and model size 13x25 is show in figure 9. Again, SII performs better than KII or Direct Convolution.

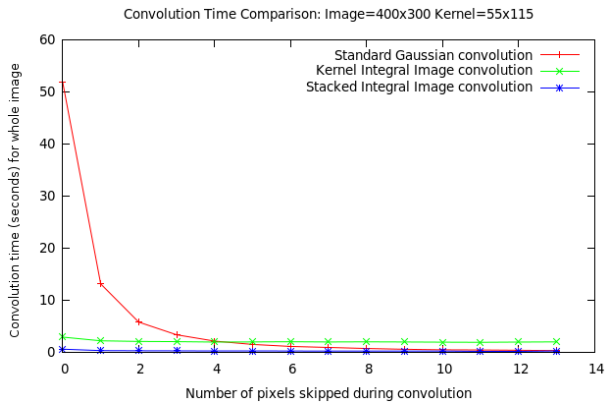


Fig. 8. Speed Improvement using the Stacked Integral Image vs. Kernel Integral Image and Direct Convolution Scene=400x300, Model=55x115

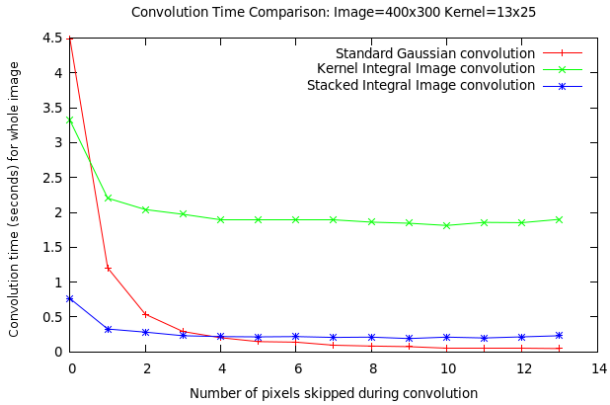
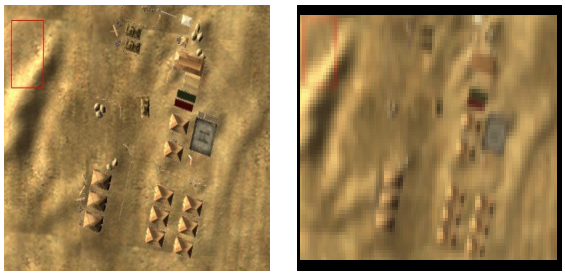


Fig. 9. Speed Improvement using the Stacked Integral Image vs. Kernel Integral Image and Direct Convolution Scene=400x300, Model=13x25

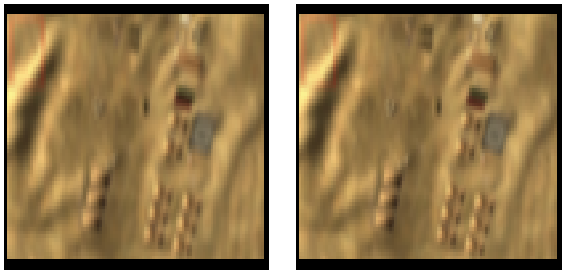
4) Accuracy comparison: To compare the accuracy of the filter output from SII and KII methods over Direct convolution, an image was convolved using all the three methods as shown in figures 10 and 11.



(a) Original Image (b) Direct Convolution

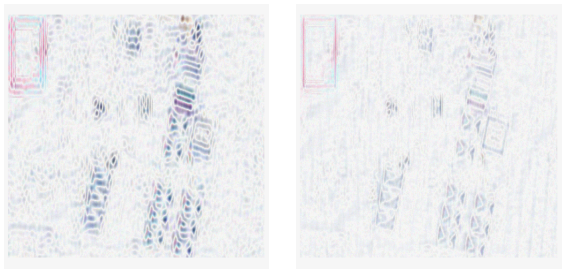
Fig. 10. Image and Direct convolution

The difference between the KII and Direct convolved image and that between SII and Direct convolved image is shown in figure 12, where the differences have been increased 10 times for visual clarity. The difference images show that the accuracy of SII method using only N=4 boxes, is higher than KII method.



(a) KII convolved image (b) SII convolved image

Fig. 11. KII convolution and SII convolution



(a) KII difference image (b) SII difference image

Fig. 12. KII and SII convolution difference images (increased 10 times) w.r.t. Direct method

VII. Conclusion

The Stacked Integral Image (SII) approach is presented as an improvement over Kernel Integral Image (KII) approach, by reducing the setup cost of generating multiple integral images in the latter approach. Using a single integral image, and a stack of box filters, the non-uniform filter can be precomputed. In both speed and accuracy, SII scores better over KII method.

References

[1] P. S. Heckbert, "Filtering by repeated integration," in *SIGGRAPH Comput. Graph.*, vol. 20, (NY, USA), pp. 315–321, ACM, 1986.

[2] M. Hussein, F. Porikli, and L. Davis, "Kernel integral images: A framework for fast non-uniform filtering," pp. 1–8, 2008.

[3] F. C. Crow, "Summed-area tables for texture mapping," in *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, (NY, USA), pp. 207–212, ACM, 1984.

[4] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, pp. 137–154, May 2004.

[5] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 829–836, 2005.

[6] F. Porikli and O. Tuzel, "Fast construction of covariance matrices for arbitrary size image," in *Proc. of IEEE International Conference on Image Processing*, pp. 1581–1584, 2006.

[7] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, (Wash., DC, USA), pp. 1491–1498, IEEE Computer Society, 2006.

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 886–893, 2005.

[9] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003.

[10] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded-up robust features," in *9th European Conference on Computer Vision*, (Graz, Austria).