SIAM J. SCI. COMPUT.
Vol. 20, No. 4, pp. 1417–1429

FAST CALCULATION OF MULTIPLE LINE INTEGRALS*

A. BRANDT† AND J. DYM‡

Abstract. A line integral is defined as the integral of two-dimensional data along a (one-dimensional, straight) line of given length and orientation. Line integrals are used in various forms of edge and line detectors in images and in the computation of the Radon transform. We present a recursive algorithm which enables approximation of discretized line integrals at *all* lengths, orientations, and locations to within a prescribed error bound in at most $O(n \log n \log\log n)$ operations, where n is the number of data points. Furthermore, for most applications (in particular, where even small amounts of noise are present in the data) all of these integrals can be computed to the desired accuracy in about $24n \log n$ operations.

Key words. Radon transform, numerical integration

AMS subject classifications. 65D30, 65D05

PII. S1064827595285718

1. Introduction.

Given two-dimensional data $g(x, y)$, define a line integral to be the (one-dimensional) integral of the data along a line of given length and orientation. Numerous applications require computation of line integrals of various lengths at many locations and in many directions. In this paper, we present a recursive algorithm which enables computation of line integrals at *all* lengths, orientations, and locations to within a given error bound in at most $O(n \log n \log\log n)$ operations, where n is the number of data points. Only the base level of the algorithm actually accesses the data to compute a set of short discretized integrals. The recursion step of the algorithm utilizes already computed integrals to compute a new set of integrals at double the length. This can be done with a *constant* number of operations per integral. The main point of this article is to show that, using this quite straightforward idea, approximations of the integrals can be computed to any desired accuracy.

Among the applications are edge and ridge detectors in image processing, which compute differences between neighboring integrals to determine the presence of steep slopes (edges) or peaks (ridges). A multilevel detector requires computation of these differences at various lengths and orientations throughout the image. An example of such a detector, using a precursor of the algorithm presented here, can be found in [2]. Recent research on multilevel computation of completion fields [5] (used to extract smooth curves in an image given raw output from a detector) used line integral computations as presented here to generate input. A similar algorithm was presented in [3, 4], where it was used to evaluate the Hough transform.

Another application is fast computation of the Radon transform. *Inversion* of the Radon transform, required for X-ray tomography and synthetic aperture radar (SAR) image reconstruction, among other applications, has been done using a data structure similar in nature to the one presented here [1]. The transform can also be

*Received by the editors May 4, 1995; accepted for publication (in revised form) September 13, 1997; published electronically March 23, 1999. This research was supported by Israel Ministry of Science grant 4135193 and by the Carl F. Gauss Minerva Center for Scientific Computations.
http://www.siam.org/journals/sisc/20-4/28571.html

†Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel (achi@wisdom.weizmann.ac.il).

‡Department of Mathematics, University of Southern California, Los Angeles, CA. Current address: Unigraphics Solutions Inc., 10824 Hope St., Cypress, CA, 90630 (jdym@cams.usc.edu).

inverted, in principle, by directly inverting the construction algorithm presented in this paper. This is the subject of current research.

In the next section we examine the relationship between the minimal resolution (angular and spatial) and the integration length, concluding that the *total* necessary number of computed integrals at any integration length is independent of the length. The following section describes an algorithm for approximating the integrals, building long integrals out of already-computed short ones using interpolation to evaluate integrals at new locations and orientations. It is also shown that doubling the resolution at which the integrals are computed improves the accuracy of the approximation by 2^p (p is the order of interpolation), while increasing the work by a factor of 4. In the final section, the algorithm is tested on a variety of targets, both establishing the correctness of the algorithm and providing an indication as to the necessary accuracy. In particular, when noise is added to the image, even in minute amounts, the approximation algorithm can be applied at minimal resolution without significant loss of accuracy. The total operation count will then be about $24n \log n$.

2. Integral resolution. Let *continuous* data $g(x, y)$ be given in a two-dimensional domain. Denote by $F(\mathbf{x}, L, \theta)$ the normalized line integral of length L, centered at $\mathbf{x} = (x, y)$ at orientation θ; i.e.,

$$F(\mathbf{x}, L, \theta) = \frac{1}{L} \int_{-\frac{L}{2}}^{\frac{L}{2}} g(x + \gamma \cos\theta, y + \gamma \sin\theta) d\gamma. \tag{1}$$

A typical task is to compute this integral at "all" locations and orientations, either for one (long) length or for a number of lengths. The quotation marks around "all" indicate that, although locations and orientations can take on a continuous range of values, minimal smoothness assumptions on $g(x, y)$ make it necessary to compute only a finite number. Such assumptions are always implied, e.g., by the data being given in some spatially discretized form, say, on a grid. The integral at interim orientation values can be computed to an arbitrary degree of accuracy by interpolation. The desired accuracy of the interpolation in between computed points will determine the minimal angular and spatial resolutions.

Aside from its dependence on the desired accuracy, the necessary resolution is also related to the integration length L. Assume that, given an evaluation at $(\mathbf{x}, L, \theta)$, we want to estimate the integral at a nearby angle $\theta + \xi$. Using (1), an approximation can be obtained by

$$\begin{aligned} F(\mathbf{x}, L, \theta + \xi) &= \frac{1}{L} \int_{-\frac{L}{2}}^{\frac{L}{2}} g(x + \gamma \cos(\theta + \xi), y + \gamma \sin(\theta + \xi)) d\gamma \\ &\approx F(\mathbf{x}, L, \theta) \\ &\quad + \frac{\xi}{L} \int_{-\frac{L}{2}}^{\frac{L}{2}} \gamma \left[-\sin\theta g_x(x + \gamma \cos\theta, y + \gamma \sin\theta) \right. \\ &\qquad \left. + \cos\theta g_y(x + \gamma \cos\theta, y + \gamma \sin\theta) \right] d\gamma \\ &= F(\mathbf{x}, L, \theta) + R \end{aligned}$$

with

$$|R| \le \frac{\xi L}{4} \max |\nabla g|.$$

This equation can also be viewed as evaluating the error incurred by using $F(\mathbf{x}, L, \theta)$ as an approximation for $F(\mathbf{x}, L, \theta + \xi)$. Since the dominant error term is a multiple of $L\xi$, the conclusion is that, for a given accuracy, *the minimal angular resolution is proportional to the integration length.* In particular, if the integration length is doubled, the number of angles computed per site must also be doubled.

Next, assume that given the integral at $(\mathbf{x}, L, \theta)$ we want to estimate the value at a nearby point $\hat{\mathbf{x}} = \mathbf{x} + \epsilon(\cos\theta, \sin\theta)$, a neighboring point along the direction of integration. Then

$$\begin{aligned} F(\hat{\mathbf{x}}, L, \theta) &= F(\mathbf{x}, L, \theta) - \frac{1}{L}\int_{-\frac{L}{2}}^{-\frac{L}{2}+\epsilon} g(x + \gamma\cos\theta, y + \gamma\sin\theta)d\gamma \\ &\quad + \frac{1}{L}\int_{\frac{L}{2}}^{\frac{L}{2}+\epsilon} g(x + \gamma\cos\theta, y + \gamma\sin\theta)d\gamma \\ &= F(\mathbf{x}, L, \theta) + R \end{aligned}$$

with

$$|R| \leq \frac{2\epsilon}{L}\max|g|.$$

Thus, the error incurred by using $F(\mathbf{x}, L, \theta)$ for $F(\hat{\mathbf{x}}, L, \theta)$ is proportional to $\frac{\epsilon}{L}$, indicating that *the minimal spatial resolution in the direction of integration is inversely proportional to the integration length.* In particular, when doubling the integration length, the number of evaluation points may be halved.

The spatial resolution perpendicular to the direction of integration will not depend on the integration length. It will be determined by the bandwidth of the data. (Later we will show that, in order to compute the integrals quickly and accurately, the perpendicular spatial resolution must be linked to the angular resolution at the base level.)

The direct consequence of the necessary angular and spatial resolutions is that the *total* number of integrals at each length L is independent of L. The actual number will be dictated by the desired accuracy. In what follows we will find the rate at which the accuracy (following deterioration by *several* interpolation steps, as will be described) rises with the resolution.

Finally, one might ask how densely the parameter L itself need be evaluated. Well,

$$F(\mathbf{x}, L + \epsilon, \theta) \approx F(\mathbf{x}, L, \theta) + \epsilon F_L(\mathbf{x}, L, \theta). \tag{2}$$

The derivative of F with respect to the length parameter is

$$F_L = -\frac{F}{L} + \frac{1}{L}\left[g\left(\mathbf{x} + \frac{L}{2}(\cos\theta, \sin\theta)\right) + g\left(\mathbf{x} - \frac{L}{2}(\cos\theta, \sin\theta)\right)\right].$$

Thus, the last term in (2) is proportional to $\frac{\epsilon}{L}$, so the sampling distance for L is directly proportional to L itself, meaning that *L may be incremented by a constant factor per level.*

3. Integral construction. In all practical digital applications, what is computed is some discrete approximization of the line integrals. One possible discretization is presented, followed by a description of a recursive algorithm for approximating

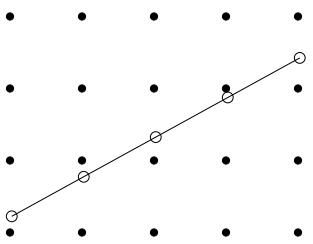

FIG. 1. *Integral discretization. At each of the points marked along the integration line, the value is approximated by interpolation from the given (black) datapoints.*

the discretized integrals. In the base level a set of integrals at an initial (short) length is computed directly from the data. The recursion takes a set of integrals at a given length and uses them to compute a set of integrals at double the length, doubling the angular resolution and halving the spatial resolution in the direction of integration.

Discrete approximation. Figure 1 depicts one possible discretization (there are many others) of $F(\mathbf{x}, L, \theta)$ for $|\theta| \leq \frac{\pi}{4}$. The data value at points where the integration line intersects a vertical grid line will be computed by interpolation from the data points on that line.[1] The integral will then be computed using the trapezoid rule, under which the integral over each subinterval is approximated by the average of its endpoints multiplied by its length. For example, the normalized integral for Figure 1 will be given by the sum of the three interior points plus the average of the endpoints, divided by four. For $\frac{\pi}{4} \leq |\theta| \leq \frac{\pi}{2}$, the same discretization applies, except that the data is interpolated at points where the integration line intersects *horizontal* grid lines.

Algorithm: Base level. The base level of the computation is the only one to actually access the data. Using the above discretization (or any other), a set of integrals is computed at initial spatial and angular resolutions for some initial integration length, all the values of which have yet to be determined. In fact, there are two independent spatial resolutions, one along the direction of integration, the other perpendicular to it. The second will be seen (in this algorithm's case) to be linked to the angular resolution.

Two examples of initial parameter settings appear in Figure 2. In each, the initial integration length is two (measured in L_∞ norm; this way all integration lines start, end, and are centered on grid lines). In the first diagram, eight integrals are computed per point. Shown are the integrals "starting from" a single point. (The integration directions will be limited to $-\frac{\pi}{2} < \theta \leq \frac{\pi}{2}$, thus integrals "start" at the left.) At this resolution, every integral starts and ends at a grid point (computing at equal intervals in $\tan\theta$ (or $\cot\theta$ if $|\theta| > \frac{\pi}{4}$); this is convenient for the length doubling stage).

The second diagram in Figure 2 shows doubled resolution. Now, sixteen directions are computed from each gridpoint.[2] Note that, in order to make the coverage uniform,

[1] Interpolation is mentioned in three different contexts in this paper. The first was implicit when discussing the necessary resolutions for the integral. There, each computed point will usually represent the integral value for its immediate neighborhood, which is equivalent to first-order interpolation. The second mention is here, and the third will be to interpolate already computed integrals to estimate the value of an integral in a new direction. The relation between the second and third interpolations will be discussed later.

[2] They can be efficiently computed by initializing the integrals at "single resolution" (eight per point) on a grid with half the meshsize, then using the length-doubling algorithm (next section). Finer initializations can be handled similarly.

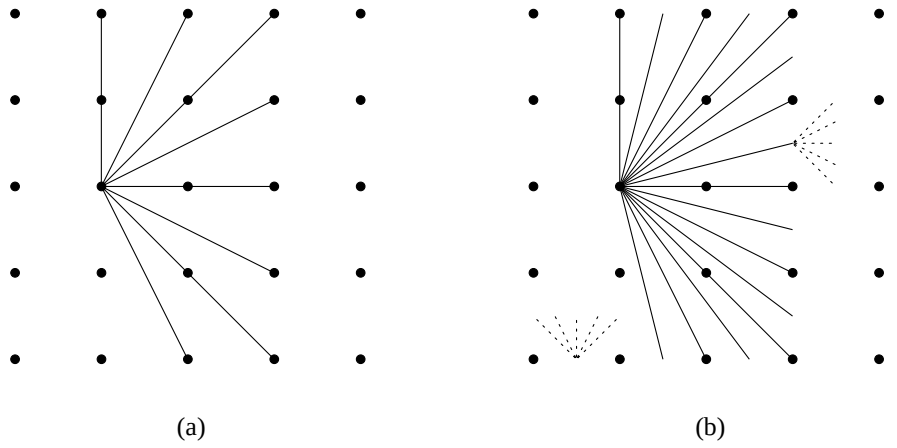

FIG. 2. *Initial resolutions.* (a) *Eight integrals starting from a point.* (b) *Doubled resolution, sixteen integrals starting from a point. At midway points, either all directions with* $|\theta| \leq \frac{\pi}{4}$ *or those with* $|\theta| \geq \frac{\pi}{4}$ *are computed, nine in either case.*

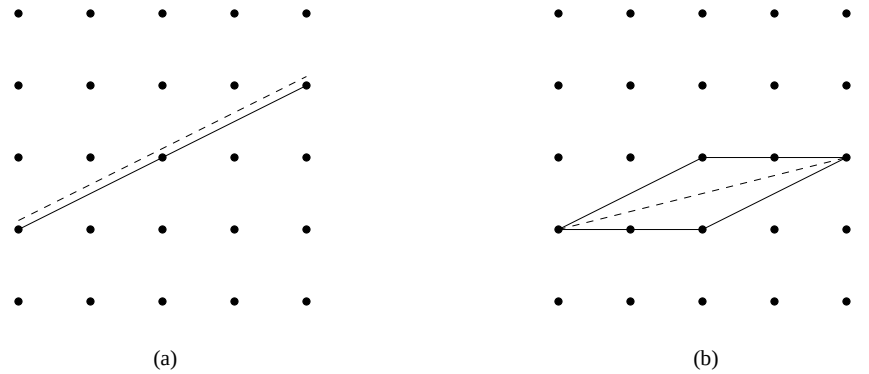

FIG. 3. *Building integrals of length four (dotted lines) from integrals of length two (solid lines).* (a) *For a pre-existing direction, by taking the average of two length-two integrals in that direction.* (b) *For a new direction, by averaging the four nearest integrals.*

all directions with $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$ will be computed starting also from midway points on the vertical grid lines, while directions with $|\theta| \geq \frac{\pi}{4}$ will be computed starting from midway horizontal gridline points. Thus, the number of computed integrals will grow by a factor of approximately four with each doubling of the initial resolution.

Length doubling. The integrals of length $2L$ are constructed using those of length L. As shown earlier, the angular resolution should double when the length does. Thus, only half of the directions for length $2L$ exist at length L. They can be computed by taking the average of two length L integrals with one coinciding endpoint (see Figure 3(a)).

The other length-$2L$ directions have not been computed at length L. However, if the angular resolution is fine enough, a very good approximation can be obtained by interpolating nearby directions. In Figure 3(b) the average of four length-two integrals is used to create a length-four integral. This can be viewed as linearly interpolating the two nearest directions to approximate the new direction at length two, then creating a length-four operator as in Figure 3(a). This is not the only way to

do it—higher orders of interpolation can be used. In this case, more length-L integrals will participate in the approximation of each new length-$2L$ direction. The angular and spatial resolution on the one hand, and the order of interpolation on the other, determine the accuracy of the approximation. Note that the length-four integrals in Figure 3 are *exactly* the discrete integrals of Figure 1 if linear interpolation is used in the latter. This, however, is not true for longer integration lengths.

Accuracy analysis. Denote by H the meshsize on which the data is given and by h the perpendicular integration meshsize (see Figure 4). Note that the latter is *constant* at each level of the algorithm, depending only on the initial sampling resolution. If the order of interpolation is p, the exact (discretized, cf. Figure 1) integral is $\tilde{F}$, and the approximated one is F, then the error caused by a *single* interpolation step can be estimated using

$$F = \tilde{F} + (\delta h)^p \tilde{F}^{(p)}$$

with $\delta < \frac{1}{2}$. The pth derivative of $\tilde{F}$ is along the direction of interpolation,[3] at some intermediate point. The key observation is that since the data is given at intervals of H and assumed to vary smoothly between datapoints, the numerical estimate for $\tilde{F}^{(p)}$ will be proportional to $\frac{\tilde{F}}{(cH)^p}$. The relative error incurred by using F for $\tilde{F}$ is thus bounded by

$$|E_{rel}| < \left(\frac{\delta h}{cH}\right)^p. \tag{3}$$

Thus, halving the integral meshsize h should result in reduction of the error by a factor of 2^p. There is a catch, however. When discretizing $\tilde{F}$, interpolation was used to evaluate points on the integration line from the given data points (Figure 1). The above formula bounding the error holds rigorously only if the *data* interpolation has p continuous derivatives (piecewise polynomial interpolation of any order has *no* continuous derivatives; cubic splines have two). In numerical experimentation we find that this requirement can be relaxed somewhat and fewer continuous derivatives ($p-2$) are needed for the formula to hold approximately.

The error estimate also allows computation of an upper bound for the necessary work to calculate all the integrals. Assuming an identical error at each of l levels, the final result will be

[3]For a given integration length L, $F = F(x, y, \theta)$ is a three-variable function. The derivatives in the direction of the interpolation are along lines in the $x - \theta$ or $y - \theta$ planes. A more rigorous definition is unnecessary, as the main issue is the order of the error.

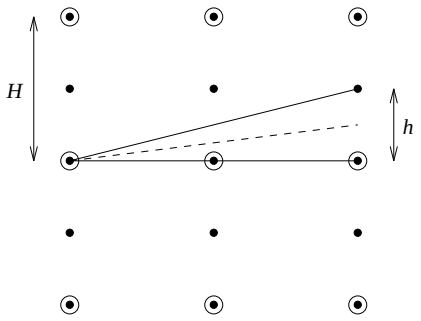

FIG. 4. *Meshsizes for accuracy computation. H is the data meshsize (and the emphasized points are datapoints). h is the integration meshsize, and all the points are integration meshpoints (points at which integrals start and end). Note that the integration mesh is denser than the data mesh in one direction only (this diagram relates to integrals with direction nearer horizontal; for near-vertical integrals the integration mesh will be dense on horizontal lines).*

$$F \approx \tilde{F}\left(1+\left(\frac{\delta h}{cH}\right)^{p}\right)^{l}.$$

If the desired relative accuracy is q, then an acceptable approximation will be obtained if

$$\left(1+\left(\frac{\delta h}{cH}\right)^{p}\right)^{l} \approx 1+q,$$

that is,

$$l\left(\frac{\delta h}{cH}\right)^{p} \approx q.$$

The desired value of q will be discussed later. Now, typically, the maximal length of the integrals is nearly equal to the size of the image. Therefore, $l \approx \log n$ and h should satisfy

$$\frac{H}{h} \propto (\log n)^{\frac{1}{p}}. \tag{4}$$

Thus, for practical values, h is very weakly dependent on n. In practice it appears that the error grows slower than exponentially per level, so the upper bound is not tight (and the dependence of h on n even weaker).

The work (operation count) is quadratic in $\frac{H}{h}$, since the number of integrals computed is quadrupled with each resolution doubling (double the angles at double the points). When $h = H$, the entire set of integrals can be computed in $O(pn \log n)$ operations, $O(pn)$ operations per level times $O(\log n)$ levels. Combining with (4), the total work for the entire computation is bounded by

$$W = O\left(pn(\log n)^{(1+\frac{2}{p})}\right).$$

This figure will be minimized by setting $p = 2 \ln \log n$, arriving at a final upper bound on the work count of

$$W = O(n \log n \log \log n).$$

Desired accuracy. In the above discussion it was shown that, with the proper investment of computational effort, it is possible to attain practically unlimited accuracy. A more difficult question to answer is how much accuracy is actually needed. To properly address this question, we must carefully determine what exactly is being calculated and what it is we *want* to calculate.

When defining the line integral in (1), the underlying function g was assumed to be continuous, defined over the entire domain. The discrete integral depicted in Figure 1, however, uses a discretized data function, defined only at the grid points. The discretization can be done in a variety of ways, for example, sampling (not usually the case in practical applications) or local averaging of g around the datapoints. The interpolation used to define the discrete integral extends the definition of the discrete function to the entire domain. The extended function will be denoted by $\tilde{g}$. *It is this, not g, that is integrated by the discrete integral!*

The approximating algorithm can efficiently calculate the integrals of $\tilde{g}$ to practically arbitrary accuracy. However, since the desired calculation is actually of the integrals of g, the approximation error, $||F - \tilde{F}||$, need not be much smaller than the discretization error, $||\tilde{F} - F_g||$, F_g denoting the "true" integrals of g. Restating, *the accuracy of the approximations should not greatly exceed that of the discretization.*

Variations. Different applications may require slight variations of the algorithm as described. A simple example is the case where it is desired to compute integrals over the entire image (as in a Radon transform). If the image is of size 2^k, this can be most efficiently done if the integral centers (and starting and ending points) are moved to be midway between the pixels, rather than on gridlines. All this does is change the integration rule of Figure 1 from trapezoid to midpoint.

Computing a Radon transform with our algorithm results in different transversal spacings between integrals at different angles. For example, if integrals at $\theta = 0$ are evaluated with distance h between integrals, the distance for $\theta = \frac{\pi}{4}$ will be $\frac{h}{\sqrt{2}}$. Also, at sublevels, the distance between integrals in the same direction along the integration line is likewise angle dependent. An alternate framework can be designed (similar to that of [1]) in which the integrals are uniformly spaced both in the transversal and parallel directions. In this case, integrals in each direction will be evaluated on a grid oriented in that particular direction. In Figure 5, this type of grid is illustrated, using two directions and three integrals along each line. Multilevel construction of longer integrals proceeds as before, interpolating adjacent directions to form new ones. However, the interpolation coefficients will no longer be the same for every new integral—rather, for each, the nearest shorter integrals will have to be determined and interpolation coefficients computed accordingly.

An alternate way to overcome the nonuniform spacing problem (in the Radon transform case, where equispacing is really necessary only at the final stage of the algorithm) is to interpolate the integrals obtained by the algorithm to an equispaced grid.

4. Numerical experiments. The approximation algorithm can be validated by testing it on a variety of synthetic images, namely, smooth data (simulating the background), data consisting of a line (nearly in the direction of integration—simulating the target), and noise (simulating noise). There are two interesting aspects of the test:

1. Demonstrating that the approximating algorithm indeed converges to the discrete integral at the rates predicted by (3).

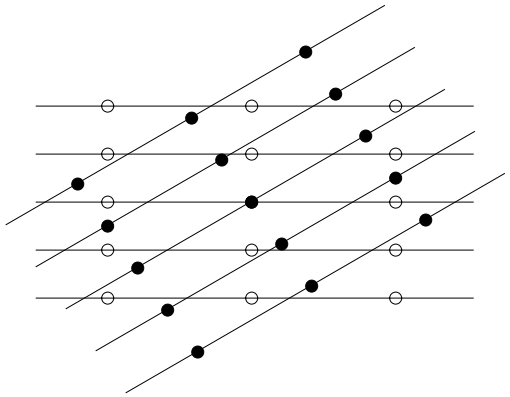

FIG. 5. *Equispaced grid (two directions shown).*

TABLE 1

Average (worst case) relative approximation error for smooth data. The discretized images are sinusoids of the form $\tilde{g}(i,j) = 1+\sin i/w$ *for various* w*. The third column shows the average relative discretization error.*

w	L	Disc. err.	$\frac{H}{h}=1$	$\frac{H}{h}=2$	$\frac{H}{h}=4$
4	8	.092%	.13% (.16%)	.030% (.039%)	.0074% (.010%)
	32	.025%	.16% (.30%)	.038% (.073%)	.0093% (.018%)
	128	.008%	.11% (.35%)	.028% (.094%)	.0070% (.025%)
2	8	.38%	.64% (.85%)	.16% (.21%)	.039% (.050%)
	32	.11%	.66% (1.5%)	.16% (.38%)	.043% (.075%)
	128	.026%	.42% (1.6%)	.11% (.45%)	.029% (.12%)
1	8	.54%	1.0% (1.8%)	.38% (.59%)	.11% (.17%)
	32	.12%	1.0% (2.9%)	.34% (.97%)	.09% (.30%)
	128	.034%	.58% (3.3%)	.21% (1.1%)	.062% (.35%)

2. Determining the minimal necessary resolution (maximal h) for which the approximation error is lower than the discretization error.

Smooth data. For smooth data, the function can be discretized by *sampling*. The underlying function g was chosen as $g(x,y) = 1 + \sin y/w$ for various values of w. Thus, at the datapoints, $\tilde{g}(i,j) = 1 + \sin i/w$ (the y axis aligned with the row index i). At the center of the image, the integral is computed for all directions with $0 \le \theta \le \frac{\pi}{4}$ in three different ways:

1. The true integral, I_t. This is calculated numerically, using the underlying function values $g(x,y) = 1 + \sin y/w$ along the integration line.
2. The discrete integral, I_d. Here, $\tilde{g}$ is integrated (directly from the image), using interpolation along the vertical gridlines and trapezoid integration, as in Figure 1.
3. The approximated integral I_a, using the algorithm described above.

As has been pointed out, (3) is rigorously valid only if the interpolated data function $\tilde{g}$ has p continuous derivatives. Thus, if linear interpolation is to be used for the approximating algorithm, cubic splines should be used to interpolate the data. (Later it will be shown that $p-2$ continuous derivatives are "almost" enough, resulting in near-optimal convergence.)

Table 1 shows the relative approximation error $|\frac{I_a - I_d}{I_d}|$ (average over all directions and worst case) for three values of w with comparison to the average discretization

TABLE 2

Average (worst case) approximation error as a function of interpolation. In these examples, the data was a sinusoid of the form $\tilde{g}(i,j) = 1 + \sin i/2$. *Different interpolations were used for the integral discretization (linear and cubic piecewise polynomial and cubic splines) and the length doubling process (linear and cubic). The comparison is for integrals of length* 128.

Disc. int. and error	Doubling int.	$\frac{H}{h} = 1$	$\frac{H}{h} = 2$	$\frac{H}{h} = 4$
linear .28%	linear	.24% (.86%)	.070% (.27%)	.025% (.14%)
	cubic	.24% (1.0%)	.062% (.25%)	.018% (.13%)
cubic .031%	linear	.41% (1.6%)	.11% (.45%)	.028% (.12%)
	cubic	.0082% (.025%)	.0021% (.0092%)	.0007% (.0052%)
spline .026%	linear	.42% (1.6%)	.11% (.45%)	.029% (.12%)
	cubic	.013% (.048%)	.0009% (.0035%)	.0001% (.0003%)

error, $|\frac{I_t - I_d}{I_t}|$. Cubic splines are used to discretize, linear interpolation to approximate. The quadratic decrease of the error with $\frac{H}{h}$ is readily apparent (for $w = 1$, only for h smaller than $\frac{H}{2}$). For this particular case, the discretization error is very low, so that h may be at most $\frac{H}{2}$, sometimes less, for the approximation error to reach the level of the discretization error. However, this is a rather special case, where it was known that there were no high frequencies or noise in the data, enabling discretization by sampling without fear of degradation due to aliasing phenomena. For more generally used discretizations, and especially for noisy images, extra resolution ($h < H$) will not usually be necessary (see examples below).

The discretization and approximation can be done with a variety of interpolation operators. Table 2 shows a number of these, applied to one of the data sinusoids of the previous table ($w = 2$). The best discretization is obtained using splines, with piecewise cubic interpolation nearly as good. Linear interpolation when discretizing gives a significantly worse result. As for the approximating algorithm's interpolater, the *rate* of decrease in the error is identical (nearly four for each halving of h) unless splines are used—then the cubic interpolater begins to show some of its fourth-order nature with error reduction of more than ten per h-halving (full fourth-order behavior is to be expected only if quintic splines are used for discretization). The low error for cubic discretization coupled with cubic approximation may be due to the smoothness of the underlying function. In the other examples shown below this low error does not occur.

The results seem to indicate that $p - 2$ continuous derivatives suffice to nearly achieve the maximal approximation convergence rate.

Lines. One of the applications of the multiple integral calculation is detection of narrow one-dimensional features (e.g., lines, edges) in a noisy image. Thus it is of importance to check that the approximating method performs well for this type of data.

The lines used as targets are defined (prior to discretization) by a rectangle with some width and length at chosen center and orientation. The strength of the line is an additional parameter. When discretizing, each image point represents an $H \times H$ square with assigned value equal to the fraction of the square covered by the line rectangle. In other words, the data is discretized by averaging the underlying image over the area of one pixel.

For 128 orientations equally spaced between 0 and $\frac{\pi}{4}$, a line was drawn, centered at the middle of the image, with width $\sqrt{2}H$ and "infinitely" long (longer than the

TABLE 3

Average (worst case) approximation error as a function of interpolation. Errors are the average of the nearest integral for 128 *lines centered at the middle of the image in various orientations. The comparison is for integrals of length* 128.

Disc. int. and error	Doubling int.	$\frac{H}{h}=1$	$\frac{H}{h}=2$	$\frac{H}{h}=4$
linear 17%	linear	11% (19%)	2.8% (5.2%)	.72% (1.8%)
	cubic	3.6% (9.7%)	1.8% (3.7%)	.53% (1.2%)
cubic 9%	linear	16% (25%)	4.2% (6.7%)	1.1% (1.9%)
	cubic	3.6% (6.8%)	.50% (1.4%)	.19% (.50%)
spline 4%	linear	19% (29%)	5.3% (8.2%)	1.4% (2.0%)
	cubic	5.9% (10%)	.63% (1.0%)	.044% (.077%)

longest integration length). For each line, the integral in the direction nearest the line is computed, from the underlying image, from the discretized data, and by the approximating method. The results (dependent on the discretization interpolation and the approximation) are shown in Table 3.

The results are similar in nature to those obtained for smooth targets, although the actual numbers are somewhat worse, as was to be expected. Again, discretization using splines is best, piecewise cubic interpolation somewhat worse, and piecewise linear interpolation substantially poorer. And again, cubic length doubling shows some of its fourth-order character only when spline interpolation is used in the discretization, showing improvement by a factor of about 10 with each meshsize halving. For all other cases tested the improvement is by about 4 per halving.

The choice of base interpolation has no effect on the complexity of the total algorithm (as this interpolation is used only in the lowest level, and even computing splines is only $O(n)$), but, at least for these synthetic targets, it may affect the outcome. (If there is noise in the data, as is usually the case, this is less true, as will soon become evident.) For the better discretizations, straightforward use of linear interpolation (for doubling) with $h = H$ may not provide an adequate approximation. However, using cubic interpolation (doubling the work) may be sufficient. The desired accuracy may also be attained by halving the meshsize once (this is a fourfold increase in the work). Doing both (high-order doubling interpolation *and* smaller meshsizes) shouldn't be necessary except in very extreme cases.

Noise. The addition of noise to the target alters the conclusions one might have prematurely drawn from the previous experiments. Noise prevents the discrete integral from reaching the true (noise-free) integral's value. The degradation due to noise can thus be regarded, in this context, as another form of discretization error. The presence of noise, even in very small amounts, blurs the distinction between the different discretization methods.

Table 4 shows some results for an extremely simple target—a flat one (all points assigned value 1.0). Gaussian noise (at varying variances) has been added to each image point. The discretization error is the relative difference between the discrete integral and the *noise-free* true integral. The approximation error is, as usual, the relative difference between the directly computed discrete integral and the integral computed using the doubling algorithm. The discretization error is entirely due to the noise and it is practically independent of the discretization method. As is to be expected, the effect of the noise decreases with increasing integration lengths. For the lengths in Table 4, using linear interpolation for the doubling (with $h = H$) brings the approximation error well under the discretization error level.

TABLE 4

Average discretization and approximation error with noise. $H/h = 1$ and doubling interpolation is linear throughout. Gaussian noise with different standard deviations σ was added to a flat target (all elements 1.0*). The discretization error measures the difference between the discrete integral and the* noise-free *true integral.*

Disc. int.	Int. length	$\sigma = .03$		$\sigma = .1$		$\sigma = .3$	
		Disc.	App.	Disc.	App.	Disc.	App.
linear	8	.70%	.18%	4.3%	.35%	15 %	1.1%
	32	.42%	.09%	1.3%	.32%	4.0%	.99%
	128	.24%	.06%	.68%	.21%	2.1%	.64%
cubic	8	.70%	.17%	4.5%	.39%	16 %	1.3%
	32	.45%	.11%	1.4%	.35%	4.3%	1.1%
	128	.25%	.08%	.73%	.25%	2.2%	.74%
spline	8	.68%	.19%	4.7%	1.1%	16 %	1.8%
	32	.46%	.13%	1.5%	.99%	4.6%	1.3%
	128	.27%	.09%	.78%	.64%	2.4%	.91%

TABLE 5

Same as above, for a sinusoid ($w = 2$). All results are for integration length 128.

Disc. int.	$\sigma = .03$		$\sigma = .05$		$\sigma = .1$	
	Disc.	App.	Disc.	App.	Disc.	App.
linear	.38%	.25%	.50%	.26%	.84%	.31%
cubic	.25%	.41%	.41%	.42%	.83%	.46%
spline	.26%	.42%	.43%	.43%	.87%	.50%

The same holds true if noise is added to a nontrivial target. Table 5 shows results when noise is added to the sinusoid target used in Table 2. Only at the lowest level of noise ($\sigma = .03$) is the use of high-order discretization methods in any way justifiable. For most applications, linear interpolation for doubling at $H/h = 1$ will give a close enough approximation. Higher orders or more integration points can substantially reduce error only for very low noise levels. (Using them *will* reduce approximation error, but not total error.)

5. Summary. An algorithm has been presented, efficiently approximating the discrete line integral in a two-dimensional image at all locations, orientations, and lengths. The algorithm utilizes already-computed integrals to create new ones of double the length. The accuracy of the approximation is a function of the initial density of the base-level integrals and the order of interpolation used in the length doubling. For most applications, the algorithm gives sufficient accuracy using linear interpolation with the integrals computed on the same mesh that the data is given on. Computing all the integrals in this manner requires about $24n \log n$ floating point operations, each either an addition or a constant multiplication. In some extreme cases, higher-order interpolation and/or denser initialization of the integrals may be required. In any case the total work need not be more than $O(n \log n \log\log n)$.

REFERENCES

[1] A. Brandt, J. Mann, M. Brodski, and M. Galun, *A fast and accurate inversion of the Radon transform,* SIAM J. Appl. Math., to appear.

[2] J. Dym, *Multilevel Methods for Early Vision,* Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1994.

[3] W. A. Götz, *Ein Schnelle Diskrete Radon Transformation Basierend auf reckursiv definiertern Digitalen Geraden,* Ph.D. thesis, University of Innsbruck, Innsbruck, Austria, 1993.

[4] W. A. Götz and H. G. Druckmüller, *A fast digital Radon transform—An efficient means for evaluating the Hough transform*, Pattern Recognition, 28 (1996), pp. 711–718.

[5] E. Sharon, A. Brandt, and R. Basri, *Completion energies and scale*, Proceedings of IEEE CVPR, 1997, Puerto Rico, pp. 884–890.