Journal of the Optical Society of America A
OPTICS, IMAGE SCIENCE, AND VISION

Algorithm for exact area-weighted antialiasing of discrete circular apertures

Scott D. Will* and James R. Fienup

The Institute of Optics, University of Rochester, Rochester, New York 14627, USA
*Corresponding author: scott.will@rochester.edu

Received 28 October 2019; revised 14 February 2020; accepted 5 March 2020; posted 5 March 2020 (Doc. ID 381672); published 30 March 2020

Accurately calculating diffraction from geometrical shapes such as circular apertures is important in computational Fourier optics. In this paper, we present an algorithm for exactly generating a discrete representation of a circular aperture whose pixel values are given by the integral of the true aperture within each pixel. We characterize the accuracy and runtime of the presented algorithm in comparison to approximate techniques such as binning high-resolution arrays, relative to the analytical Airy pattern. © 2020 Optical Society of America

https://doi.org/10.1364/JOSAA.381672

1. INTRODUCTION

In physical optics modeling, predicting the diffraction pattern or point-spread function (PSF) from an optical system with a circular aperture is a ubiquitous problem. For instance, in image-based wavefront sensing, where the objective is to obtain an accurate estimate of the phase in the exit pupil of the optical system using only measurements of the PSF at a plane at or near the paraxial focus, the well-known Fourier transform relationship between the exit pupil and the paraxial focal plane is used to construct a forward model [1], which is then inverted to estimate the unknown phase function [2]. In high-precision applications such as high-contrast imaging of exoplanets, one is motivated to characterize the performance of an optical instrument with realistic aberrations and defects as accurately as possible using numerical physical optics models, in order to place practical tolerances on the fabrication and alignment of various internal optical components [3].

In these example scenarios, and many others, the Fourier transformation forms the foundation for the forward model of the imaging system. Therefore, when constructing computer models, one faces the problem of discretizing the system in a way that is as physically consistent with the true, continuous-domain physics as possible. The most common strategy, based on Nyquist–Shannon sampling theory, is to sample the continuous-domain input function using a regular grid of Dirac δ functions [1]; for an input that is bandlimited (i.e., whose Fourier transform has a finite extent) and with sufficiently fine sampling, Shannon showed that the discretized output is equivalent to the continuous input in the sense that the input can be perfectly reconstructed from its samples [4].

However, many input functions in physical optics modeling, such as fields transmitted by circular apertures, are not bandlimited. In this case, the discrete Fourier transform (DFT) of the δ-sampled input becomes aliased, meaning that sampling-induced periodically shifted copies overlap within the fundamental period, corrupting the values in the fundamental period, particularly near the edges of the computational window. This means that the predicted PSF differs from the true PSF and causes errors in the forward model and inverse-problem solution.

Though aliasing is unavoidable for non-bandlimited functions, there are approaches to discretizing the aperture that can reduce its effects. One approach is to δ-sample the aperture with much finer sample spacing than would otherwise be needed to capture the spatial frequency information of interest, and then rather than employ the entirety of the fundamental period, truncate the PSF array to use only the values near the center, where the aliased energy is minimal. This comes at the cost of much greater computing and memory requirements. If we instead restrict ourselves to sampling the aperture only as finely as needed, another natural approach is to form a Riemann sum approximation to the Fourier transform integral; doing so requires that one calculate the integral of the aperture function within the rectangular regions representing each of the pixels of the computational grid. This is equivalent to computing the convolution between the continuous-domain aperture and a continuous-domain rectangle function in two dimensions before δ-sampling; in the Fourier domain, this corresponds to multiplying the continuous Fourier transform of the aperture with a sinc window, which reduces the energy in the sidelobes and consequently reduces aliasing in the DFT after sampling. This is also analogous to the Lanczos sigma factors in time-domain signal processing, which form a sinc window on the

Fourier series coefficients of a continuous-time, periodic signal [5].

Approximating the pixel integrals for circular apertures and other shapes can be done straightforwardly by generating a high-resolution discrete representation using δ-sampling, and then rebinning with boxcar averaging by an integer factor K to the desired resolution, which we shall refer to as the "bin-K" algorithm. Another simple approximation is obtained by using a linear ramp between zero and one, with the value of the ramp given by the distance between the edge of the continuous-domain aperture and the center of each pixel, as a fraction of pixel width. We will refer to this algorithm as the "ramp" algorithm, and describe it in greater detail in the following section. We have utilized the ramp algorithm for many years; however, an algorithm for exactly evaluating the pixel integrals, which we will henceforth refer to as "area-weighted sampling" or "area-weighted antialiasing," has, as of yet, not been described in the literature, to our knowledge. In principle, having such an algorithm enables both improved accuracy in diffraction models and the ability to characterize the error associated with the bin-K and ramp algorithms.

In the past, much work was devoted to graphical antialiasing of circles and filled apertures in the computer graphics community with the goal of improving visual quality [6–14]. Recently, Sheppard [15] studied methods for generating optimal binary representations of circular apertures, but the Fourier transforms even of these optimal binary apertures poorly match the analytical Airy pattern, which is the continuous Fourier transform of a continuous-domain circular aperture when illuminated by a plane wave.

In this paper, we describe an algorithm for exactly evaluating the pixel integrals for a discrete circular aperture, providing an exact solution to the Riemann sum approximation of the Fourier transform of a continuous circular aperture. Our algorithm utilizes simple geometric calculations in a recursive manner, making it computationally efficient. As a consequence of the exact evaluation of pixel integrals, the DFT of the resulting aperture achieves the minimum-achievable error relative to the analytical Airy pattern under the Riemann sum approximation, which is nonzero due to aliasing.

A discrete circular aperture with zero error relative to the analytical Airy pattern can be obtained by δ-sampling the Airy pattern and computing an inverse DFT of the result. However, the resulting aperture will have ringing artifacts due to the truncation of the original Airy pattern, which cause the aperture to have nonzero transmittance outside of the nominal radius. We argue that this does not satisfy the requirement that the discretization be consistent with the underlying continuous-domain physics. For a specific example, one can consider wavefront aberrations described by Zernike polynomials, which are undefined beyond the nominal radius of the aperture. Extrapolating the aberrations leads to unphysical wavefront aberration components being transmitted. A full analysis of this particular strategy is beyond the scope of this paper.

This paper is structured as follows: in Section 2, we briefly describe the ramp algorithm. In Section 3, we derive the exact algorithm for area-weighted sampling. Finally, in Section 4, we benchmark the performance of the exact algorithm against binary apertures, as well as apertures generated by the ramp and bin-K algorithms, in runtime and in Fourier-domain accuracy.

2. RAMP ALGORITHM

The ramp algorithm is a simple method for approximating the pixel integrals over a discrete circular aperture. In one dimension, it yields the exact pixel integrals for a rectangular window, but does not produce exact integrals in two or higher dimensions. For the $[m, n]$th pixel, compute the quantity

$$r_{mn} \triangleq \frac{1}{2} + \frac{1}{\Delta}\left(R - \sqrt{(x_m - x_c)^2 + (y_n - y_c)^2}\right), \quad \textbf{(1)}$$

where R is the radius of the desired aperture, (x_m, y_n) is the center of the (m, n)th square pixel, Δ is the pixel width (step size), and (x_c, y_c) is the center of the aperture, which can be at a fractional-pixel location. Then the value of each pixel is assigned as

$$C[m, n] = \min\{\max\{r_{mn}, 0\}, 1\}. \quad \textbf{(2)}$$

This equation is applied to all pixels in the array. The result is that all pixels whose centers fall more than $\Delta/2$ inside or outside the radius of the aperture are labeled as interior or exterior pixels with a value of one or zero, respectively. For all other pixels, whose centers are within $\Delta/2$ of the aperture edge, the pixel integral is approximated as a linear ramp between zero and one. This is highly accurate for pixels along a cardinal direction from the center but is less accurate everywhere else.

3. EXACT ALGORITHM

In this section, we describe an algorithm that analytically computes the area of the shape formed by the intersection between the desired continuous-domain aperture and each pixel in the discrete computational grid. These intersection areas will be referred to as pixel integrals in the remainder of this paper. The approach is as follows: first, the pixels that lie along the edge of the aperture, henceforth referred to as edge pixels, are identified by counting the number of intersections between the boundary of the aperture and the sides of each pixel. Next, we use simple geometry to derive analytical expressions for the areas of the four partitions formed when vertical and horizontal chords intersect within the aperture. This result then is used to recursively find the pixel integral for each edge pixel.

Consider a circular aperture $C(x, y)$ with radius R, centered at $(x, y) = (x_c, y_c)$, which we would like to discretize using area-weighted sampling. Also consider a discrete coordinate grid $[m, n]$, where m and n are integers corresponding to the locations $x_m = m\Delta x$ and $y_n = n\Delta y$, respectively, for step sizes Δx and Δy.

For simplicity, let $\Delta x = \Delta y \triangleq \Delta$ and assume that the grid size N with $-N/2 \leq m \leq N/2 - 1$ and $-N/2 \leq n \leq N/2 - 1$ (for even N) is chosen so that C is contained entirely inside the region bounded by the discrete coordinate grid. If this is not the case, the aperture can be sampled using an expanded grid and cropped to the desired size.

Let $C[m, n]$ denote the discrete representation of $C(x, y)$. We interpret each sample $C[m, n]$ as corresponding to a finite,

$\Delta \times \Delta$ square region of space centered at $(x, y) = (x_m, y_n)$, which we denote $S_{m,n}(x, y)$. The objective is to calculate the fractional area of the aperture $C[m, n]$ within each square $S_{m,n}(x, y)$ for all $[m, n]$.

A. Identifying Edge Pixels

To simplify the problem, we restrict the problem to all edge pixels, i.e., squares that contain the edge $E(x, y)$ of C, because any square strictly inside or outside C will be assigned a value of one or zero, respectively. Mathematically, the set of edge pixels is all $S_{m,n}(x, y)$ that intersect $E(x, y)$ at two or more locations; if only one intersection point exists, then the corresponding side is tangent to E and the square is entirely outside the aperture. If the edge of the aperture passes directly through the corner of a square, it will be identified as an edge pixel because two separate intersection points will be counted, one with the vertical side and one with the horizontal side that form the corner.

First, we construct an $N \times N$ array $I[m, n]$ to store the number of intersections between the edge of the aperture and the boundary of each square, initialized with $I[m, n] = 0$ everywhere. As illustrated in Fig. 1(a), for each column m, consider the vertical line $x'_m = x_m - \Delta/2$, which contains the left edge of $S_{m,n}$ and the right edge of $S_{m-1,n}$. The point(s) where $E(x, y)$ intersects this line are

$$y_\pm = y_c \pm \sqrt{R^2 - (x'_m - x_c)^2}. \quad (3)$$

We then identify the rows $n_\pm$ that contain the intersection point(s) by finding $n_\pm \le y_\pm/\Delta < n_\pm + 1$. This yields one or two pairs of squares $S_{m,n+}$ and $S_{m-1,n+}$, and $S_{m,n-}$ and $S_{m-1,n-}$, whose shared vertical edges contain the intersection point(s). Increment by one the corresponding values of $I[m-1, n_+]$, $I[m, n_+]$, $I[m-1, n_-]$, and $I[m, n_-]$.

Similarly, as illustrated in Fig. 1(b), for each row n, consider the horizontal line $y'_n = y_n - \Delta/2$, which contains the bottom edge of $S_{m,n}$ and the top edge of $S_{m,n-1}$. The x locations where $E(x, y)$ intersects this line are given by

$$x_\pm = x_c \pm \sqrt{R^2 - (y'_n - y_c)^2}. \quad (4)$$

We then find the two rows $m_\pm$ that contain the intersection points using $m_\pm \le x_\pm/\Delta < m_\pm + 1$, and increment $I[m_+, n-1]$, $I[m_+, n]$, $I[m_-, n-1]$, and $I[m_-, n]$.

After iterating over the set of horizontal and vertical pixel edges, arriving at the intersections shown in Fig. 1(c), $I[m, n]$ will contain the number of intersections, shown in Fig. 1(d), between the four sides of each square $S_{m,n}(x, y)$ and the aperture edge $E(x, y)$. As described above, any square with zero or one intersection points is either fully inside or outside the aperture, or tangent to it, so the edge set $\mathcal{E}$ is identified by selecting all $[m, n]$ such that $I[m, n] \ge 2$:

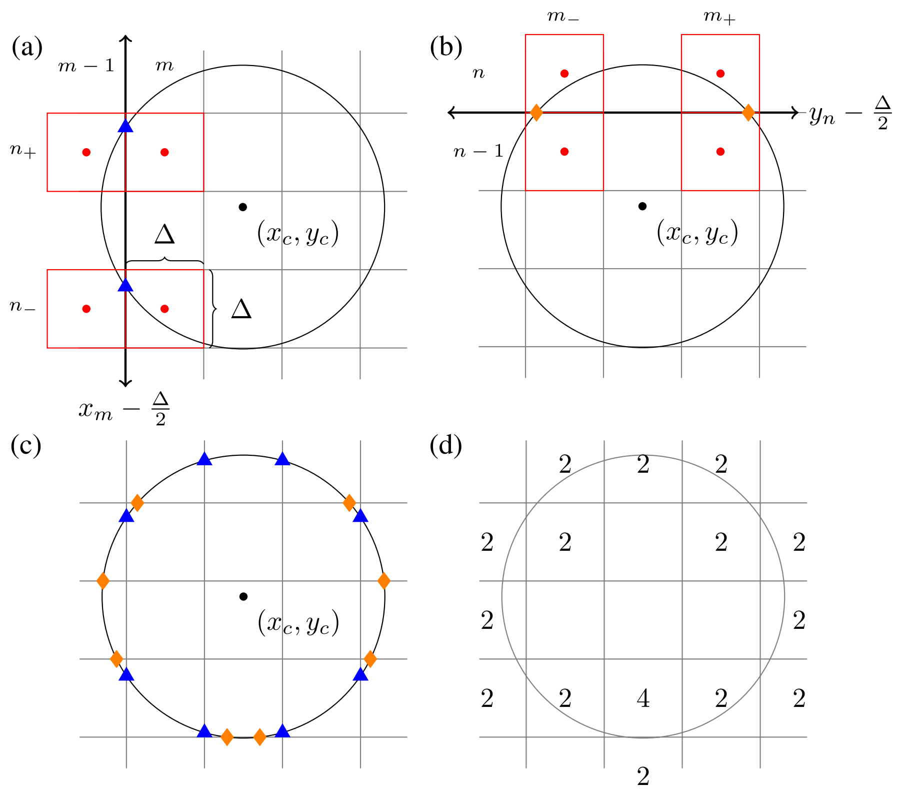

Fig. 1. Edge-finding algorithm described in Section 3.A. (a) For each column of pixels, the point(s) (blue triangles) where the edge of the aperture intersects the line along the left edge of column m are calculated using Eqs. (3) and (4). The four pixels whose edges contain these points are indicated in red, with row coordinates n_+ and n_-. (b) The procedure in (a) is repeated to find the points of intersection (orange diamonds), with column coordinates m_+ and m_-, between the edge of the aperture and the bottom edge of row n. (c) Intersection points after iterating over all horizontal and vertical gridlines. (d) Array $I[m, n]$ containing the number of intersection points between the edges of each pixel and the edge of the aperture. All pixels with two or more intersection points are identified as edge pixels.

$$\mathcal{E} \triangleq \{[m, n] : I[m, n] \geq 2\} . \tag{5}$$

B. Areas Formed by Intersecting Chords

In this section, we derive analytical expressions for the areas of four partition regions formed when a horizontal chord and vertical chord intersect within the aperture. When these chords are chosen to coincide with the edges of a row and column of pixels, respectively, each partition area can be identified as the sum of pixel integrals for a collection of pixels. As described in further detail in Section 3.C below, by making sequential partition-area calculations with circular and horizontal chords placed along the edges of each edge pixel, one can obtain the individual contributions of each pixel to the total pixel integral, yielding the desired discrete aperture.

Consider two orthogonal chords displaced from the aperture center by the signed distances d_h and d_v, respectively, with the intersection point $\mathcal{I} \triangleq (x_c + d_v, y_c + d_h)$ inside C so that the aperture is partitioned into four regions, indexed by $k \in \{1, 2, 3, 4\}$, starting from the upper-left region and increasing clockwise, as shown in Fig. 2. These chords intersect $E(x, y)$ at points $\mathcal{L}$ and $\mathcal{R}$ (horizontal chord) and $\mathcal{T}$ and $\mathcal{B}$ (vertical chord).

The lengths $\overline{\mathcal{LR}}$ and $\overline{\mathcal{TB}}$ are computed using d_h and d_v as

$$\overline{\mathcal{LR}} = 2\sqrt{R^2 - d_h^2}, \tag{6}$$

$$\overline{\mathcal{TB}} = 2\sqrt{R^2 - d_v^2}. \tag{7}$$

These are simply the differences of the intersection coordinates $x_\pm$ and $y_\pm$ found using Eqs. (3) and (4). The coordinates of points $\mathcal{L}$, $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{B}$, along with the coordinate of the intersection point $\mathcal{I}$, are

$$\mathcal{L} = \left(x_c - \frac{1}{2}\overline{\mathcal{LR}}, y_c + d_h \right), \tag{8}$$

$$\mathcal{R} = \left(x_c + \frac{1}{2}\overline{\mathcal{LR}}, y_c + d_h \right), \tag{9}$$

$$\mathcal{T} = \left(x_c + d_v, y_c + \frac{1}{2}\overline{\mathcal{TB}} \right), \tag{10}$$

$$\mathcal{B} = \left(x_c + d_v, y_c - \frac{1}{2}\overline{\mathcal{TB}} \right), \tag{11}$$

$$\mathcal{I} = (x_c + d_v, y_c + d_h) , \tag{12}$$

and the lengths of the line segments $\mathcal{IL}$, $\mathcal{IT}$, $\mathcal{IR}$, and $\mathcal{IB}$ are

$$\overline{\mathcal{IL}} = d_v + \frac{1}{2}\overline{\mathcal{LR}}, \tag{13}$$

$$\overline{\mathcal{IR}} = \frac{1}{2}\overline{\mathcal{LR}} - d_v, \tag{14}$$

$$\overline{\mathcal{IT}} = \frac{1}{2}\overline{\mathcal{TB}} - d_h, \tag{15}$$

$$\overline{\mathcal{IB}} = d_h + \frac{1}{2}\overline{\mathcal{TB}}. \tag{16}$$

Each of the four regions is composed of a right triangle and a circular segment between the edge of the aperture and the hypotenuse of the triangle, with areas $A_{T,k}$ and $A_{C,k}$, respectively, as shown in Fig. 2. Both of these areas have simple geometric formulas that can be evaluated using the quantities given above.

The triangle areas $A_{T,k}$ are

$$A_{T,1} = \frac{1}{2}\left(\overline{\mathcal{IL}}\right)\left(\overline{\mathcal{IT}}\right), \tag{17}$$

$$A_{T,2} = \frac{1}{2}\left(\overline{\mathcal{IR}}\right)\left(\overline{\mathcal{IT}}\right), \tag{18}$$

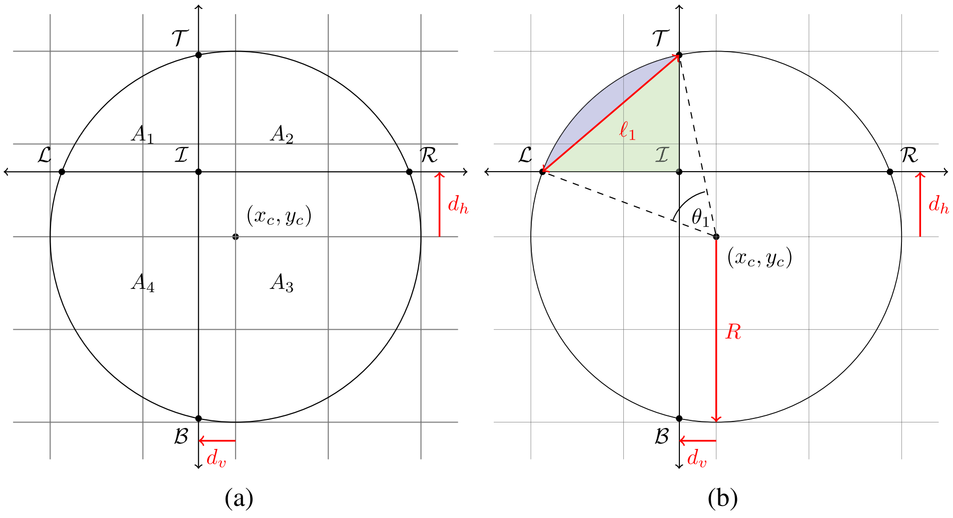

Fig. 2. (a) Regions A_1, A_2, A_3, and A_4 formed when orthogonal chords intersect inside an aperture with radius R. (b) The area of the upper-left region A_1 (shaded) is the sum of the areas of a circular segment $A_{C,1}$ (blue) and a right triangle $A_{T,1}$ (green) whose hypotenuse is $\mathcal{LT}$. The quantities d_h and d_v denote the signed displacement of the horizontal and vertical chords, respectively, away from the aperture center (x_c, y_c). The quantities ℓ_1 and θ_1 denote, respectively, the length of, and central angle subtended by, the hypotenuse $\mathcal{LT}$.

$$A_{T,3} = \frac{1}{2}\left(\overline{\mathcal{IR}}\right)\left(\overline{\mathcal{IB}}\right), \tag{19}$$

$$A_{T,4} = \frac{1}{2}\left(\overline{\mathcal{IL}}\right)\left(\overline{\mathcal{IB}}\right). \tag{20}$$

The areas of the circular segments are given by [16]

$$A_{C,k} = \frac{1}{2}R^2\left(\theta_k - \sin\theta_k\right), \tag{21}$$

where θ_k is the central angle subtended by the chord of segment k, given by [16]

$$\theta_k = 2\sin^{-1}\left(\frac{\ell_k}{2R}\right), \tag{22}$$

where ℓ_k is the length of the hypotenuse of the corresponding right triangle. Finally, the total area A_k of each region is $A_k = A_{T,k} + A_{C,k}$.

C. Computing Edge Pixel Integrals

We now apply this result iteratively to obtain the values $C[m, n]$ of the edge pixels of the sampled aperture. We break the problem into separate subproblems for each of the four quadrants of the aperture, relative to the aperture center. In each subproblem, we work from the outside in to compute the pixel integral between each square $S_{m,n}(x, y)$ and $C(x, y)$. As we will see shortly, this allows each area to be found using only the simple geometric calculations described in the previous section.

To illustrate the basic principle, we first consider the case for the upper-left quadrant of the aperture, shown in Fig. 3 below. We begin at the leftmost pixel in the topmost row of edge set $\mathcal{E}$, i.e.,

$$[m_0, n_0] = \min_m \left\{\max_n \mathcal{E}\right\}, \tag{23}$$

whose center is at coordinate $(x, y) = (m_0\Delta, n_0\Delta)$. By construction, there are no edge pixels above or to the left of this pixel.

Place a vertical chord at $x = x_{m_0} + \Delta/2$, and a horizontal chord at $y = y_{n_0} - \Delta/2$, which are, respectively, the right and bottom edges of the square $S_{m_0,n_0}(x, y)$. For shorthand, let $A^1_{m,n}$ denote the area A_1 of the upper-left region, computed when the horizontal and vertical chords are placed at the right and bottom edges, respectively, of square $S_{m,n}(x, y)$. Using the procedure in Section 3.B, the pixel integral $C[m_0, n_0]$ is given directly as the upper-left region area $A^1_{m,n}$.

Moving rightwards one pixel to location $[m_0 + 1, n_0]$, shown in Fig. 3(b), we move the vertical chord one pixel width to the right, and compute the area $A^1_{m_0+1,n_0}$. We can see in Fig. 3(b) that $A^1_{m_0+1,n_0}$ is simply the sum of the known area $C[m_0, n_0]$ and the unknown area $C[m_0 + 1, n_0]$, so by subtraction, we have

$$C[m_0 + 1, n_0] = A^1_{m_0+1,n_0} - C[m_0, n_0]. \tag{24}$$

We continue moving to the right toward the center of the aperture, in each step computing A^1_{m,n_0}, and subtracting the areas of all pixels to the left, which have already been evaluated. Once we have reached the center of the aperture, we move to the leftmost edge pixel of the next (lower) row and repeat. As seen in Fig. 3(c), in this case, the partition area A_{m_0,n_0-1} contains the pixel integral for the current square as well as the pixel immediately above it, and so

$$C[m_0, n_0 - 1] = A^1_{m_0,n_0-1} - C[m_0, n_0]. \tag{25}$$

In general, in the upper-left quadrant, the area $A^1_{m,n}$ for any given $[m, n]$ contains the sum of the pixel integrals of square $S_{m,n}$ and all squares above or to the left of $S_{m,n}$. Defining $\mathcal{E}_1 \triangleq \{[j, k] \in \mathcal{E} : j \leq m, k \geq n\}$ as the set of edge pixels above and to the left of pixel $[m, n]$, the pixel integral $C[m, n]$ can be written mathematically as

$$C[m, n] = A^1_{m,n} - \sum_{[j,k]\in\mathcal{E}_1} C[j, k]. \tag{26}$$

By iterating from left to right and top to bottom in the upper-left quadrant of the aperture, the sum in the right-hand side of (26) always contains known quantities, allowing the next pixel integral to be computed analytically by computing the area $A^1_{m,n}$ followed by subtraction of the known pixel integrals for all pixels above and to the left.

We proceed similarly for the other three quadrants, in each case adjusting the order of iteration so that the pixel integrals for each edge pixel are computed from the outside in. For the upper-right quadrant, we begin at the rightmost pixel of the top row of edge pixels and iterate to the left and downwards, in each iteration computing the area $A^2_{m,n}$, defined as the area of the upper-right region when the horizontal and vertical chords are placed at the bottom and left edges of $S_{m,n}(x, y)$, respectively.

For the lower-right quadrant, we begin at the rightmost pixel of the bottom row of edge pixels and iterate to the left and upwards, in each iteration computing the area $A^3_{m,n}$, defined as the area of the lower-right region when the horizontal and vertical chords are placed at the top and left edges of $S_{m,n}(x, y)$, respectively.

Finally, for the lower-left quadrant, we begin at the leftmost pixel of the bottom row of edge pixels and iterate to the right and upwards, in each iteration computing the area $A^4_{m,n}$, defined as the area of the lower-left region when the horizontal and vertical chords are placed at the top and right edges of $S_{m,n}(x, y)$, respectively.

In a small number of cases, it is possible for $E(x, y)$ to traverse a pixel without enclosing any corner of the pixel, forming a region enclosed by a line segment along the inner edge of the pixel and the edge of the aperture, as shown in Fig. 4. This pixel integral cannot be computed by the recursive algorithm in this section because it is composed of only a circular segment, rather than a circular segment and a triangle. This case can be detected easily during the edge-finding stage by checking if the two intersection points $x_\pm$ or $y_\pm$ occur inside the same pixel. The pixel integral for this case is then given by Eqs. (21) and (22), where ℓ_k in Eq. (22) is given by $x_+ - x_-$ or $y_+ - y_-$, depending on the orientation of the overlap.

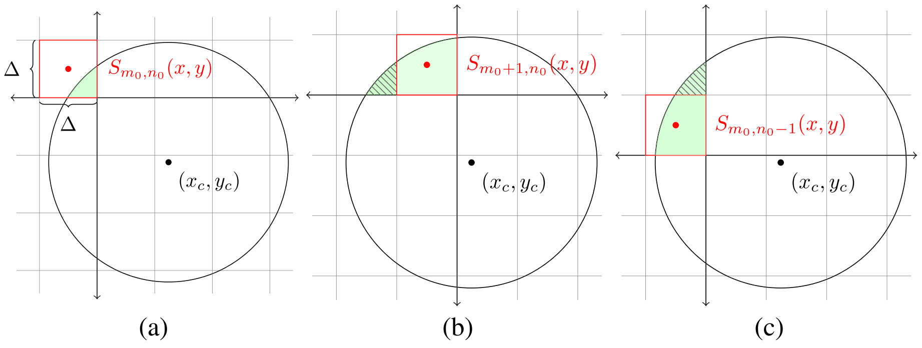

Fig. 3. Calculation of pixel integrals for edge pixels in the upper-left quadrant of the aperture. (a) Beginning at the upper-leftmost square $S_{m_0 n_0}$, the pixel integral $C[m_0, n_0]$ is found using area A_1 (see Section 3.B). (b) Moving right one pixel, the pixel integral $C[m_0 + 1, n_0]$ is given by A_1 (green) minus the known $C[m_0, n_0]$ (crosshatched). (c) Moving down one row, the pixel integral $C[m_0, n_0 - 1]$ is given by A_1 (green) minus $C[m_0, n_0]$ (crosshatched).

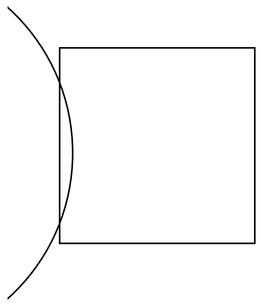

Fig. 4. Illustration of special case described in Section 3, in which the edge of the aperture intersects a single edge of a given pixel twice. In this case, the pixel integral is determined directly using Eqs. (21) and (22), with the chord length $\ell = y_+ - y_-$, where y_+ and y_- are given by Eq. (3).

4. RESULTS

The exact algorithm was benchmarked on a desktop workstation with a 3.4 GHz, 8-core processor and 32 GB of RAM against a simple binary aperture generated using δ-sampling, the ramp algorithm, and bin-K algorithms for $K \in \{2, 8, 16\}$, as a function of array width N. The exact algorithm was implemented using Cython, a superset of the Python programming language with support for integration with C. This implementation is publicly available in a GitHub repository located at https://github.com/sdwill/circleshade. All other algorithms were implemented in a vectorized fashion using the NumPy and SciPy Python libraries.

In each trial, the x and y axes were normalized to have a unity total width and sampled over the range $[-0.5, 0.5)$ with N points (i.e., with step size $\Delta = 1/N$). Apertures were randomly generated with radius R chosen randomly from a uniform distribution over the interval $[0, 0.5)$. The center location (x_c, y_c) was also randomly chosen, with each coordinate drawn from a uniform distribution over the interval $[-(0.5 - R), (0.5 - R)]$. These two measures admit a robust variety of test cases, while ensuring that in each case the aperture is contained entirely within the computational window to avoid truncation artifacts when performing a DFT.

For each aperture, the Fourier-domain error of DFT$\{C[m, n]\}$ was computed relative to the analytical Fourier transform of the circular aperture, given by

$$\mathcal{F}\{C(x, y)\} = R\frac{J_1(2\pi R\rho)}{\rho} \exp\left\{-i2\pi(x_c f_x + y_c f_y)\right\}, \tag{27}$$

where $J_1(\cdot)$ is the first-order Bessel function of the first kind, and $\rho = \sqrt{f_x^2 + f_y^2}$ is the radial Fourier-domain coordinate, using the normalized sum-of-squared differences (NSSD) error metric. The NSSD metric is defined for complex-valued functions $g[m, n]$ and $h[m, n]$, normalized to $h[m, n]$, as

$$\frac{\sum_m \sum_n |g[m, n] - h[m, n]|^2}{\sum_m \sum_n |h[m, n]|^2}. \tag{28}$$

The DFT was computed using the matrix Fourier transform (MFT) algorithm [17], expressed by

$$\text{DFT}\{C[m, n]\} = (\Delta x)^2 \exp\left\{-i2\pi \mathbf{f}_x \mathbf{x}^T\right\} C[m, n] \exp\left\{-i2\pi \mathbf{y}\mathbf{f}_y^T\right\}, \tag{29}$$

where exponentiation is performed elementwise, and the products are understood as matrix multiplications. This transformation is carried out in two dimensions from length-N discrete coordinate axes $\mathbf{x}$ and $\mathbf{y}$ to the spatial frequency axes $\mathbf{f}_x$ and $\mathbf{f}_y$, which may be sampled with arbitrary sample spacing and range. If the fast Fourier transform (FFT) is used instead, one must take care to ensure that the chosen implementation of the bin-K algorithm correctly places the location $(x, y) = (0, 0)$ in accordance with the conventions of the FFT algorithm (many do not). Incorrect coordinate centering will manifest itself as an extra linear phase in FFT$\{C[m, n]\}$ not accounted for by the analytical description in Eq. (27) derived from continuous-domain Fourier mathematics, and will distort comparisons between the bin-K algorithm and others.

In the figures and discussion that follow, we will restrict our attention to one particular randomly generated test case with

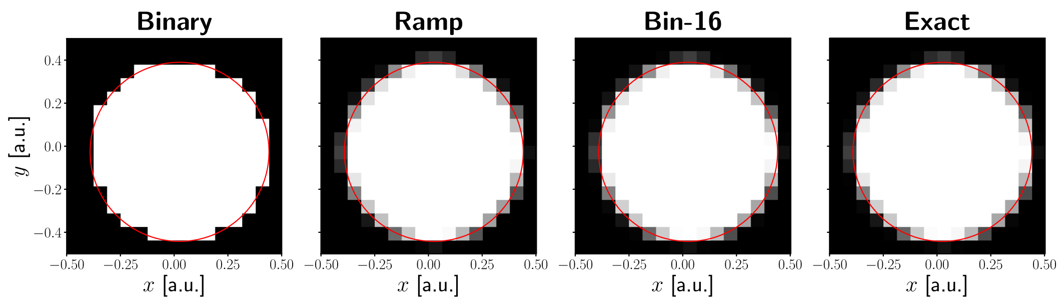

Fig. 5. Circular apertures sampled on 16×16 pixel grids for $R = 0.416$ and $(x_c, y_c) = (0.025, -0.026)$, generated by, from left to right, δ-sampling, the ramp algorithm, bin-K for $K = 16$, and the exact algorithm. The edge of the true aperture is shown in each figure in red.

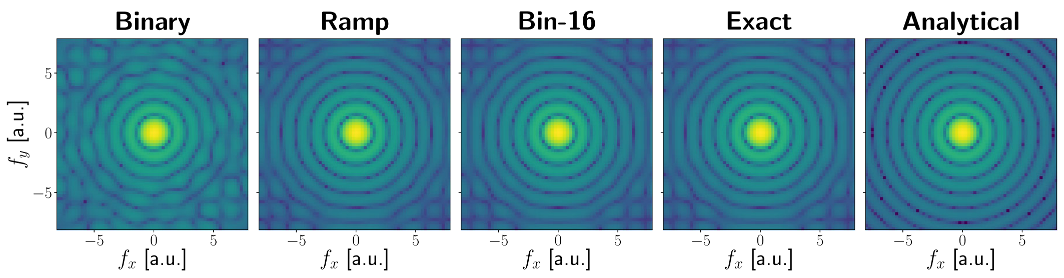

Fig. 6. Discrete Fourier transform amplitudes of each of the apertures in Fig. 5, along with an analytical Airy pattern given by Eq. (27). The DFT of the binary aperture suffers from aliasing artifacts within a small number of Airy rings, while the antialiased sampling methods well approximate the analytical Airy pattern (right) over a much greater extent of the Fourier domain.

radius $R = 0.416$, center location $(x_c, y_c) = (0.025, -0.026)$, and array width $N = 16$. The spatial frequency axes $\mathbf{f}_x$ and $\mathbf{f}_y$ were sampled at twice the Nyquist frequency, i.e., $\Delta f_x = \Delta f_y = 1/4$, and contained the full fundamental period of the DFT, with total width $W_{f_x} = W_{f_y} = 4$. We stress that although the results presented here are specific to this geometry, the broad patterns and conclusions were identical for all random test cases analyzed. The apertures generated by δ-sampling and the ramp, bin-16, and exact algorithms are shown in Fig. 5, while their DFT amplitudes are shown alongside the analytical Fourier transform of the continuous-domain aperture in Fig. 6.

In the spatial domain, where a closed-form reference does not exist, the NSSD error of the apertures from the δ-sampling, ramp, and bin-K algorithms was computed relative to the aperture from the exact algorithm. Average runtime was estimated using 50 executions of each algorithm to enable better estimation for small array sizes.

Spatial-domain error, Fourier-domain error, and runtime as a function of array width N for all algorithms are shown in Fig. 7. As N increased and each aperture became more finely sampled, the Fourier-domain error decreased for all algorithms; as discussed in Section 1, this error cannot be nonzero for a Riemann sum approximation to the Fourier transform of a non-bandlimited function such as a circular aperture. The Fourier-domain error of the exact algorithm was tracked remarkably closely by the ramp algorithm and the bin-K algorithm for $K > 8$. However, at $N = 1024$, the exact method executes in approximately 0.2 s, twice as fast as the bin-8 and an order of magnitude faster than the bin-16 algorithm.

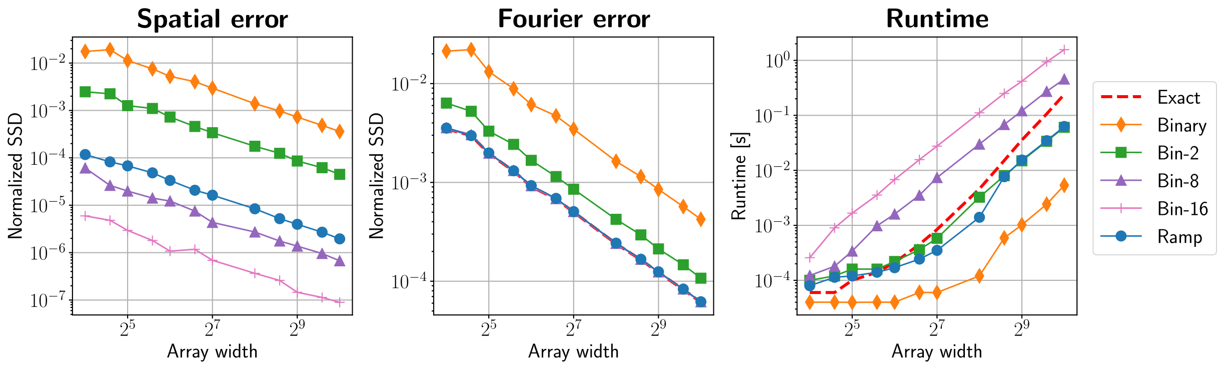

Fig. 7. Spatial-domain error, Fourier-domain error, and runtime for five approximate aperture-generating algorithms (see Section 4) and the exact algorithm as a function of array width N. The Fourier-domain error shown here is the NSSD, Eq. (28), of the DFT of each aperture relative to $\mathcal{F}\{C(x, y)\}$. The spatial-domain error is the NSSD error for each aperture relative to the exact aperture.

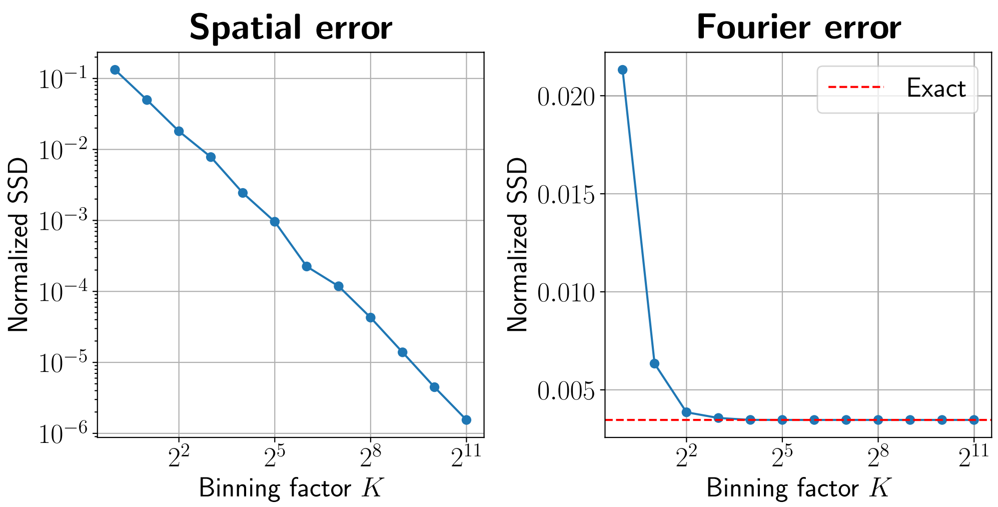

Fig. 8. Spatial-domain error between bin-K apertures and exact aperture, and Fourier-domain error between bin-K apertures and analytical Fourier transform in Eq. (27), as a function of K and for fixed array size $N = 16$. Both are measured by the NSSD, Eq. (28). The Fourier-domain error of the exact aperture provides a lower bound to the Fourier error of the bin-K algorithm, and is shown by the red dashed line in (b). As K increases, the bin-K aperture converges to the exact aperture in both the spatial and Fourier domains, providing strong evidence of the correctness of the exact algorithm.

To demonstrate the correctness of the proposed algorithm, we chose to take a numerical approach. The pixel integrals approximated by the bin-K algorithm converge to the true pixel integrals as K tends toward infinity. We generated a collection of bin-K apertures with an array width fixed at $N = 16$, and varied K between two and 2048. For large K, generating a full, high-resolution binary aperture with $MK \times MK$ samples required impractically large amounts of memory, and so for this validation stage only, we utilized a slower, yet more memory efficient, implementation of the bin-K algorithm in which each $K \times K$ block of the high-resolution binary aperture was evaluated sequentially. For each aperture, we then computed spatial-domain and Fourier-domain errors as before.

Figure 8 shows the result of this analysis. The bin-K aperture does indeed converge monotonically to the exact aperture in the spatial domain as K increases. In the Fourier domain, convergence is also rapid: for $K > 8$, the NSSD error of the bin-K algorithm is within 5×10^{-6} of the error of the exact algorithm. Though not a formal proof, this offers strong evidence that the algorithm presented here behaves as a limiting case of the bin-K algorithm, and consequently that it produces exact pixel integrals down to the floating-point noise floor.

5. CONCLUSION

In this paper, we have presented a novel algorithm for discretizing a circular aperture by exactly computing the integrals over each pixel in a discrete grid. We have shown that the apertures generated by the algorithm behave as the limiting case of the bin-K algorithm as K tends toward infinity. The presented algorithm is more computationally efficient than generating a highly binned aperture, and because it computes pixel integrals exactly, it achieves the minimum error with respect to an analytical Airy pattern possible when approximating the Fourier transform integral as a Riemann sum. The ramp algorithm described in Section 2 is a simple and efficient alternative whose Fourier-domain error is, surprisingly, only very slightly worse than that of the proposed algorithm; we believe that this is because the Fourier error of the ramp algorithm is dominated by the inherent aliasing due to sampling, which is much greater than the error due to its approximation to the Riemann sum.

Because there is no closed-form expression for the diffraction pattern from a circular aperture illuminated by any field other than a plane wave, we have limited our analysis to on-axis plane wave illumination. Evaluating the pixel integrals for a circular aperture under general illumination is a considerably more challenging problem. To circumvent this, one can separately discretize the aperture function and the illuminating field, and multiply the two to form the illuminated aperture. This is not strictly identical to discretizing the illuminated aperture; however, we believe that for smoothly varying aberrations, the error from this approximation is small. A more rigorous analysis of sampling effects in circular apertures with general wavefront aberrations will be explored in future work.

The proposed approach may additionally be extended to include sampling of ellipses with arbitrary eccentricity and angle of rotation by replacing the calculation of the area of a circular segment with that of an elliptical segment.

Funding. Goddard Space Flight Center (80NSSC18K0213).

Disclosures. The authors declare no conflicts of interest.

REFERENCES

1. J. W. Goodman, *Introduction to Fourier Optics*, 4th ed. (W. H. Freeman, 2017).
2. J. R. Fienup, “Phase-retrieval algorithms for a complicated optical system,” Appl. Opt. **32**, 1737–1746 (1993).

3. J. E. Krist, B. Nemati, and B. Mennesson, “Numerical modeling of the proposed WFIRST-AFTA coronagraphs and their predicted performances,” J. Astron. Telesc. Instrum. Syst. **2**, 011003 (2016).
4. C. E. Shannon, “Communication in the presence of noise,” Proc. IRE **37**, 10–21 (1949).
5. C. Lanczos, *Applied Analysis* (Van Nostrand, 1956).
6. J. E. Bresenham, “Algorithm for computer control of a digital plotter,” IBM Syst. J. **4**, 25–30 (1965).
7. B. K. Horn, “Circle generators for display devices,” Comput. Graph. Image Process. **5**, 280–288 (1976).
8. M. Doros, “Algorithms for generation of discrete circles, rings, and disks,” Comput. Graph. Image Process. **10**, 366–371 (1979).
9. Z. Kulpa, “On the properties of discrete circles, rings, and disks,” Comput. Graph. Image Process. **10**, 348–365 (1979).
10. M. McIlroy, “Best approximate circles on integer grids,” ACM Trans. Graph. **2**, 237–263 (1983).
11. D. Field, “Algorithms for drawing anti-aliased circles and ellipses,” Comput. Vis. Graph. Image Process. **33**, 1–15 (1986).
12. X. Wu, “An efficient antialiasing technique,” SIGGRAPH Comput. Graph. **25**, 143–152 (1991).
13. Y. K. Liu and X. N. Li, “Double-step circle drawing algorithm with and without grey scale,” in *4th International Conference on Image and Graphics (ICIG)* (2007), pp. 886–891.
14. E. Andres, “Discrete circles, rings and spheres,” Comput. Graph. **18**, 695–706 (1994).
15. C. J. R. Sheppard, “Pixellated circle,” Appl. Opt. **57**, 7878–7882 (2018).
16. E. W. Weisstein, “Circular segment,” in *Wolfram MathWorld* (2018).
17. R. Soummer, L. Pueyo, A. Sivaramakrishnan, and R. J. Vanderbei, “Fast computation of Lyot-style coronagraph propagation,” Opt. Express **15**, 15935–15951 (2007).