ELIAS: An Accurate and Extensible Lithography Aerial Image Simulator With Improved Numerical Algorithms

Peng Yu and David Z. Pan, *Senior Member, IEEE*

***Abstract*—Lithography simulators have been playing an indispensable role in process optimization and design for manufacturability (DFM). The ever smaller feature sizes demand higher numerical accuracy and faster runtime on these lithography simulators. Aerial image simulation is the first key step in lithography simulation, and the method using transmission cross coefficient (TCC), which is a two-dimensional integral, is the most commonly used technique for full-chip aerial image simulation. In this paper, we present a very accurate, yet efficient and extensible aerial image simulator, ELIAS. We find that the majority of the numerical error during the TCC computation is due to the discontinuous boundaries of the support of the TCC integrand. We reduce the error dramatically by using a recursive integration algorithm. Because TCC is usually computed on uniform grids, we further speed up the algorithm without increasing the errors. Given the same accuracy, our new algorithm can speed up the runtime by $100\times$–$1000\times$. Our algorithm also provides smooth tradeoff between accuracy and runtime. It can be used to benchmark other lithography aerial simulators. In addition, ELIAS provides an open-source, flexible software framework to incorporate different lithography settings.**

***Index Terms*—Accuracy, aerial image simulation, C++, ELIAS, fast Fourier transform (FFT), Hopkins equation, lithography simulation, numerical algorithm, recursive integration, runtime, transmission cross coefficient (TCC).**

I. Introduction

IN MODERN semiconductor industry, simulations of manufacturing processes are required to ensure circuit manufacturability. Fast and accurate lithography simulation is a key enabling technology [1] in the design-to-manufacturing flow, e.g., optical proximity correction (OPC) [2], post-OPC silicon image verification, design rule definition and litho-aware physical design. These computational lithography applications [3], [4] have recently received many interests [5]–[8].

A typical full-chip lithography simulation flow is shown in Fig. 1. The transmission cross coefficient (TCC) matrix includes all of the optics information. It can be decomposed into a set of kernels using Optimal Coherent Approximations [9], [10]. The images can be simulated by convolving masks with the kernels.

Manuscript received September 09, 2008; revised February 02, 2008. Current version published May 06, 2009. This work was supported in part by the National Science Foundation, in part by KLA-Tencor (software donation), and in part by the Intel Corporation (equipment donation).

The authors are with the Department of Electrical and Computer Engineering, the University of Texas, Austin, TX 78712-1528 USA (e-mail: yupeng@cerc.utexas.edu; dpan@ece.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSM.2009.2017652

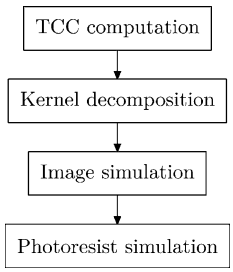


Fig. 1. Typical full-chip lithography simulation flow.

Photoresist can be simulated using, for example, the variable threshold model [11].

As feature sizes reduce, smaller simulation errors are required. For example, a critical dimension (CD) error of 5 nm might be tolerable in the 130 nm technology node, but it is definitely unacceptable in the 22-nm technology node [12]. Therefore, it is important to improve the accuracy of a simulator to keep up with the shrinking of feature sizes. In this paper, we present a very accurate and efficient algorithm for aerial image simulation.

We prove that the jump discontinuity of the integrand of TCC on the boundary of the integrand support is the major source of TCC errors. We improve the computation accuracy by integrating the discontinuous regions using a recursive integration method. The flow in Fig. 1 requires the computation of the function values of TCC on a uniform grid, which form a four-dimensional (4-D) TCC matrix. By taking advantage of the correction between the entries within a TCC matrix, we can speed up its computation without losing accuracy. As the error of kernel decomposition can be reduced by hardware improvement [13], the improvement of the accuracy of TCC directly increases the aerial image simulation accuracy.

Our algorithm can be used to benchmark other aerial image simulators extensively. Closed-form solutions have been used to benchmark lithography simulators [14], [15]. However, a simulator can not be benchmarked for cases where closed-form solutions do not exist. Because our algorithm can compute aerial image very accurately for arbitrary lithography settings, a closed-form solution is not required any more.

We implement the algorithm in a C++ software package ELIAS [16]. It can be extended to support various lithography settings, such as, aberrations, illumination schemes and vectorial imaging. Since ELIAS can compute TCC very accurately, it can be used to benchmark other image simulation tools.

The contributions of this paper are:

1) We prove that the discontinuity of illumination and projection functions is the major source of the numerical errors in TCC.
2) We introduce the recursive integration method to reduce these errors.
3) Without losing accuracy, we further speed up the algorithm by taking advantage of the correlation of the entries in TCC matrices.
4) Our experiments show that the new algorithm can run $100\times$ to $1000\times$ faster than the conventional algorithm achieving the same level of accuracy and ELIAS can be used as a benchmark.

The remaining of this paper is organized as follows. In Section II, we review the lithography image simulation model and the TCC matrix. In Section III, we prove that the discontinuity of TCC integrand results in the majority of the error in the numerical integration. We introduce the recursive integration method to reduce the error due to the discontinuity. We speed up the algorithm in Section IV. Section V shows the runtimes and the numerical errors of ELIAS. Section VI concludes this paper.

II. Lithography Imaging Basics and TCC Matrix

A. Lithography Imaging Basics

The aerial image intensity is given by the Hopkins equation [17]–[20]:

$$\mathcal{I}(f,g)=\iint_{-\infty}^{+\infty}\mathcal{T}(f+f',g+g';f',g')\times\mathcal{F}(f+f',g+g')\mathcal{F}^*(f',g')\,\mathrm{d}f'\mathrm{d}g'. \quad (1)$$

$\mathcal{F}(f,g)$ is the mask transmission function $F(x,y)$ in the frequency domain, where (f,g) denotes a frequency point and (x,y) denotes a spatial point. The superscript * denotes the complex conjugation operation. $\mathcal{I}(f,g)$ is the image in the frequency domain. $\mathcal{T}(f',g';f'',g'')$ is the *transmission cross coefficient* (TCC), given by

$$\mathcal{T}(f',g';f'',g'')=\iint_{-\infty}^{+\infty}\mathcal{J}(f,g)\mathcal{K}(f+f',g+g')\times\mathcal{K}^*(f+f'',g+g'')\,\mathrm{d}f\mathrm{d}g. \quad (2)$$

The meanings of the symbols in (2) are described below.

- $\mathcal{J}(f,g)$ is the illumination function, which satisfies

$$\mathcal{J}(f,g)=0,\quad \text{for } f^2+g^2\geqslant 1. \quad (3)$$

We illustrate some commonly used illumination functions in Fig. 2.

- $\mathcal{K}(f,g)$ is the projection system transfer function. It can be written as

$$\mathcal{K}(f,g)=e^{i\frac{2\pi}{\lambda}z\phi(f,g)}\mathcal{K}_0(f,g) \quad (4)$$

where $\phi(f,g)=\sqrt{1-(f^2+g^2)\sin^2\theta_{\mathrm{obj}}}$, λ is the wavelength, θ_{obj} is the semi-aperture angle at the image plane

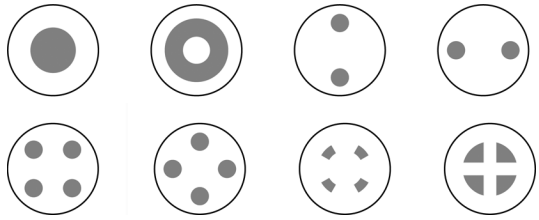


Fig. 2. Some commonly used illumination schemes. The outer circles are references, whose radii are all 1. $\mathcal{J}$ is a constant over the gray regions.

[21] and z denotes the focus error. Assuming a circular pupil, $\mathcal{K}_0(f,g)$ can be written as

$$\mathcal{K}_0(f,g)=\begin{cases}e^{i2\pi\Phi(f,g)}, & k<1\\ 0, & \text{otherwise}\end{cases} \quad (5)$$

where $\Phi(f,g)$ denotes the lens aberration function.

As the feature size shrinks, the process variations become increasingly important. This requires simulation of the effects of the process variations on imaging characteristics. In particular, the image intensity sensitivity with respect to the focus error z in a scalar model can be written as [22]–[24]

$$\left.\frac{\partial^n}{\partial z^n}\mathcal{I}(f,g)\right|_{z=0}=\iint_{-\infty}^{+\infty}\left.\frac{\partial^n}{\partial z^n}\mathcal{T}(f+f',g+g';f',g')\right|_{z=0}\times\mathcal{F}(f+f',g+g')\mathcal{F}^*(f',g')\,\mathrm{d}f'\mathrm{d}g'. \quad (6)$$

In (6), the *variational* TCC $\left.\frac{\partial^n}{\partial z^n}\mathcal{T}(f',g';f'',g'')\right|_{z=0}$ is defined as

$$\left.\frac{\partial^n}{\partial z^n}\mathcal{T}(f',g';f'',g'')\right|_{z=0}=\sum_{m=0}^{n}(-1)^{n-m}(i2\pi)^n\iint_{-\infty}^{+\infty}\mathcal{J}(f,g)\times(\phi(f+f',g+g'))^m\mathcal{K}_0(f+f',g+g')\times(\phi(f+f'',g+g''))^{n-m}\mathcal{K}_0^*(f+f'',g+g'')\times\mathrm{d}f\mathrm{d}g. \quad (7)$$

We only reviewed the scalar model above, which is good for low numerical aperture (NA). Polarized/high NA imaging [21] can be formulated in a similar fashion, which also have associated TCCs and variational TCCs.

Both TCC and variational TCC [see (2) and (7)] are in the form of an integral of the product of three functions. Since they are numerically the same, we do not distinguish them and simply call them TCC in the rest of the paper.

B. TCC Matrix

Aerial images requires the computation of TCC on a uniform grid in the frequency domain [25], [26]. Let us denote the grid size as $\tilde{\Delta}$. Based on (5), we have that

$$\mathcal{K}_0(f,g)=0,\quad \text{if } f \text{ or } g \text{ is not in } [-1,1]. \quad (8)$$

We can find multiple rectangular regions, inside of which $\mathcal{J}(f,g)$ is not always zero, and outside of which is always zero. Assume that the smallest such rectangular region is of the size

$$\sigma_1 \times \sigma_2. \tag{9}$$

Therefore, based on (2), we have that TCC $\mathcal{T}(f_1,g_1,f_2,g_2)$ is always zero outside a 4-D box of size

$$(2+2\sigma_1)\times(2+2\sigma_2)\times(2+2\sigma_1)\times(2+2\sigma_2). \tag{10}$$

This means that we need to compute a 4-D matrix, named TCC matrix, whose entries are

$$\mathcal{T}(i_1\tilde{\Delta}, j_1\tilde{\Delta}, i_2\tilde{\Delta}, j_2\tilde{\Delta}) \tag{11}$$

where i_1, j_1, i_2 and j_2 are integers, and $(i_1\tilde{\Delta}, j_1\tilde{\Delta}, i_2\tilde{\Delta}, j_2\tilde{\Delta})$ are in the box (10). We will take advantage of the fact that the integrals in a TCC matrix are related to reduce the runtime (see Section IV).

III. Error Analysis for TCC Integration

TCC is an integral of a function with jump discontinuity. In Section III-A, we demonstrate that the jump discontinuity can result in large truncation errors using the conventional TCC computation method. We reduce the errors using the recursive integration method in Section III-B.

A. Truncation Error Analysis for TCC

TCC can be written as an integral $I(u)$ over a finite region R

$$I_R(u) = \iint_R u(x,y)\,\mathrm{d}x\mathrm{d}y. \tag{12}$$

The *midpoint*numerical integration rule [27] has been used previously to compute TCC [28], [29]. In this method, the integral is appropriated as a summation of the function values on a grid with grid size Δ:[1]

$$I_R(u) \approx M_{R,\Delta}(u) = \Delta^2 \sum_{ij} u(\boxdot_{ij}). \tag{13}$$

Here, $\boxdot_{ij}$ denotes a grid point, which is the center of a square as shown in Fig. 3. The summation is over all the square centers that are in R.

In the following theorem, we show that this rule can result in a large truncation error when it is used to integrate a function with jump-discontinuity. The proof is shown in Appendix.

Theorem 1: If a function $u(x,y)$ has a bounded support R and is smooth in each connected region of R, and the function and its derivatives to all orders in both arguments are bounded, then the truncation error of $M_{R,\Delta}(u)$ for the approximation of $I_R(u)$ is bounded by

$$|I_R(u) - M_{R,\Delta}(u)| \leqslant C_1\Delta^2 + C_2\Delta \tag{14}$$

[1]Note that Δ is the size of the grid which is used by the midpoint numerical integration, whereas $\tilde{\Delta}$ in (11) is the size of the grid where the integration values shall be computed. Δ' which will be introduced later is the minimal grid size after quadrisections.

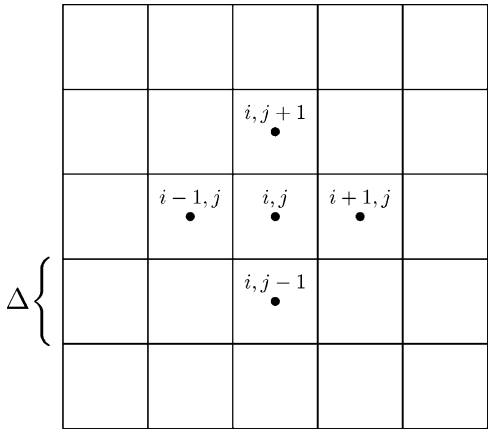


Fig. 3. Midpoint Rule. Each square, denoted as $\square_{ij}$, is centered at $(i\Delta, j\Delta)$, denoted as $\boxdot_{ij}$.

where C_1 and C_2 are two non-negative constants that depend on the function u, but not the grid size Δ.

Here, C_1 is proportional to the area of the support R and is proportional to the average magnitudes of the second order derivatives of the function $u(x,y)$ on R; C_2 is proportional to the length of the boundary of R and is proportional to the average jump of the function $u(x,y)$ on the boundary.

Remark 1: When the function $u(x,y)$ is a linear function in each connected region of the support R, the constant C_1 reduces to zero. In this case, the truncation error is purely bounded by the Δ term, which is originated from the jump of the function $u(x,y)$ along the boundary of the support R. The error is still dominated from the boundary, when the second order derivatives of the function (x,y) are small. Therefore, to improve the numerical integration accuracy, the boundary must be examined separately and is discussed in the next subsection.

B. Improving Accuracy—Recursive Integration

We have shown that the boundaries are the primary contributors to the numerical integration error. To reduce such errors, we use the recursive integration method. We then estimate the runtime of this method.

We divide the domain of integration into smaller subregions recursively until the approximation in each subregion is accurate enough (Fig. 5) [30]. Algorithm 1 shows the details. It concentrates more on the boundaries than the internal regions. When the square size is small enough ($< \Delta'$, Δ' is a parameter) or the integrand is continuous in it, the algorithm does not divided the square further. In this case, the algorithm still uses the midpoint rule as an approximation. We denote the approximation of $I_\square(u)$ on a boundary square as

$$M_{\square,\Delta'}(u) = \mathrm{INTEGRATE}(u,\square,\Delta'). \tag{15}$$

Therefore, the integral $I_R(u)$ can be approximated as

$$I_R(u) \approx M_{R,\Delta,\Delta'}(u) = \Delta^2 \sum_{ij}^{(0)} u(\boxdot_{ij}) + \sum_{ij}^{(1)} M_{\square_{ij},\Delta'}(u) \tag{16}$$

Fig. 4. Domain of the integration $\square$ can be divided into 4 smaller squares $\square_i (i = 1, 2, 3, 4)$.

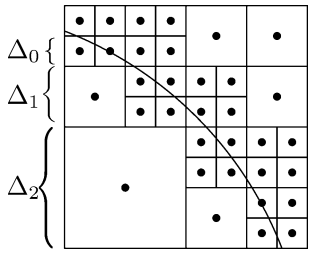


Fig. 5. Recursive integration method. The integrand is discontinuous on the curve. A square is recursively divided into smaller squares, if the integrand is discontinuous in it. The integrand is evaluated at the not-divided square centers (dots). Δ_k denotes the square size ($k = 0, 1, 2$ in this case).

where the first summation is over internal squares and the second is over boundary squares. Compared to the old method (13), the new method (16) integrates the boundary regions using the recursive integration method instead of the midpoint rule.

Algorithm 1 Recursive Integration Algorithm

1: **function** INTEGRATE$(u, \square, \Delta')$

2: **return** $\Delta^2 \times$ AVERAGE$(u, \square, \Delta')$

1: **function** AVERAGE$(u, \square, \Delta')$

2: $\Delta \leftarrow$ the size of $\square$

3: **if** $\Delta \geqslant \Delta'$ **and** u is not continuous on $\square$ **then**

4: Divide the square $\square$ into 4 smaller squares $\square_i (i = 1, 2, 3, 4)$. See **Fig. 4**.

5: **return** $\frac{1}{4} \sum_{i=1}^{4}$ AVERAGE$(u, \square_i, \Delta')$

6: **else**

7: **return** $u(\boxdot)$

The following theorem states the truncation error of this method. The proof is shown in Appendix.

Theorem 2: If $u(x, y)$ satisfies all the requirements of $u(x, y)$ that are stated in Theorem 1, then the truncation error of $M_{R,\Delta,\Delta'}(u)$ for the approximation of $I_R(u)$ is bounded by

$$|I_R(u) - M_{R,\Delta,\Delta'}(u)| \leqslant C_1 \Delta^2 + C_2 \Delta' \tag{17}$$

where C_1 and C_2 here are the same as C_1 and C_2 in (14).

Remark 2: The only difference between $M_{R,\Delta}(u)$ in (13) and $M_{R,\Delta,\Delta'}(u)$ in (16) is how the integration is done on the boundary squares. The second term in the right-hand side of (17) is also from boundary squares. In the recursive integration method, we can control the minimum square size by Δ'. Therefore, that term is related to Δ' instead of Δ as in (14).

Based on the above theorem, we can reduce the error contributed by boundaries arbitrarily smaller by controlling Δ'. The following theorem shows the runtime of the recursive integration algorithm is related with Δ' according to a power law. In practice, we need to choose an appropriate Δ' to balance the error and the runtime. The proof of Theorem 3 is shown in Appendix.

Theorem 3: The time complexity of Algorithm 1 for a square $\square$ where $u(x, y)$ is discontinuous is

$$T_{\square,\Delta'} \propto \left(\frac{\Delta}{\Delta'}\right)^{\gamma}$$

where γ is a constant satisfying $0 < \gamma < 2$.

Remark 3: The constant γ can be inferred experimentally as shown in Section V.

IV. Integration Algorithm for TCC Matrix

Because not just an entry but a whole TCC matrix needs to be computed, the information sharing between neighboring entries can be exploited to speed up the algorithm presented in Section III. In Section IV-A, we derive that a TCC matrix can be decomposed into a triple correlation term which is mainly from the internal region and a correction term which is from the boundary region. We then show how to compute the two terms efficiently in Sections IV-B and IV-C.

A. Numerical Integration Formula

The TCC integral is a continuous triple correlation of the following form

$$\begin{aligned} &h(x_1, y_1, x_2, y_2) \\ &\quad = \iint u(x, y) v(x + x_1, y + y_1) \\ &\qquad \times w(x + x_2, y + y_2)\, \mathrm{d}x\mathrm{d}y. \end{aligned} \tag{18}$$

According to the discussion of the TCC matrix in Section II, we need to compute $h(i_1\tilde{\Delta}, j_1\tilde{\Delta}, i_2\tilde{\Delta}, j_2\tilde{\Delta})$ for integers i_1, j_1, i_2, and j_2. We choose $\Delta = \tilde{\Delta}/n$, where n is a positive integer. For any function $u(x, y)$, we denote the function resulted from shifting the arguments of a function $u(x, y)$ as

$$u^{ij}(x, y) = u(x - i\Delta, y - j\Delta).$$

Therefore, we have

$$\begin{aligned} &h(i_1\tilde{\Delta}, j_1\tilde{\Delta}, i_2\tilde{\Delta}, j_2\tilde{\Delta}) \\ &\quad = \iint u(x, y) v^{-ni_1, -nj_1}(x, y) \\ &\qquad \times w^{-ni_2, -nj_2}(x, y)\, \mathrm{d}x\mathrm{d}y. \end{aligned} \tag{19}$$

We could directly use the recursive integration algorithm to compute the approximations of all the TCC matrix entries. But we do not do so for reasons as follows:

1) For any function $u(x, y)$,

$$M_{\square_{i_1,j_1},\Delta'}(u^{i_2,j_2}) = M_{\square_{i_1-i_2,j_1-j_2},\Delta'}(u) \tag{20}$$

which means that shifting the integrand is the same as shifting the region of integration;

2) We need to compute a whole TCC matrix.

The follows theorem takes advantage of the fact that the integrand is a product of three functions which reduces the runtime without decreasing accuracy. The proof is shown in Appendix.

Theorem 4: If the integrand is a product of a discontinuous function $u(x,y)$ and a continuous function $v(x,y)$ over a square $\square$, we approximate

$$I_{\square}(uv) = \iint_{\square} u(x,y)v(x,y)\,\mathrm{d}x\mathrm{d}y$$

by

$$M_{\square,\Delta'}(u,v) = M_{\square,\Delta'}(u)v(\boxdot). \tag{21}$$

$I_R(uvw)$ can be approximated as (22), found at the bottom of the page. The truncation error of $M_{R,\Delta,\Delta'}(u,v,w)$ is of the same order as that of $M_{R,\Delta,\Delta'}(uvw)$.

To simplify the discussions, we introduce a few short hand notations. For a function $u(x,y)$, we define

$$u_{ij} = \begin{cases} \frac{1}{\Delta^2} M_{\square_{ij}}(u), & \text{if } u(x,y) \text{ is continuous in } \square_{ij} \\ 0, & \text{if } u(x,y) \text{ is discontinuous in } \square_{ij} \end{cases}$$

$$\overline{u}_{ij} = \begin{cases} 0, & \text{if } u(x,y) \text{ is continuous in } \square_{ij} \\ \frac{1}{\Delta^2} M_{\square_{ij},\Delta'}(u), & \text{if } u(x,y) \text{ is discontinuous in } \square_{ij} \end{cases}$$

and

$$\hat{u}_{ij} = u_{ij} + \overline{u}_{ij}. \tag{23}$$

Note that $\hat{u}$ is a matrix, whereas $\hat{u}_{ij}$, with the index ij, is a number. Based on the above definitions, it is obvious that

$$\overline{u}_{ij} = \overline{u^{-i,-j}}_{00} = \overline{u^{-i,-j}} \tag{24}$$

where we omit the subscript 00 for convenience.

According to Theorem 4,

$$h_{i_1 j_1 i_2 j_2} = \frac{1}{\Delta^2} h(i_1\tilde{\Delta}, j_1\tilde{\Delta}, i_2\tilde{\Delta}, j_2\tilde{\Delta})$$

can be approximated as

$$\begin{aligned} h_{i_1 j_1 i_2 j_2} &= \sum_{ij} \hat{u}_{ij}\hat{v}_{i+ni_1,j+nj_1}\hat{w}_{i+ni_2,j+nj_2} \\ &\quad + \Bigg(\sum_{ij}^{(2)} (\overline{v^{-i-ni_1,-j-nj_1} w^{-i-ni_2,-j-nj_2}} \\ &\qquad - \hat{v}_{i+ni_1,j+nj_1}\hat{w}_{i+ni_2,j+nj_2})\hat{u}_{ij} \\ &\quad + \sum_{ij}^{(2)} (\overline{u^{-i,-j} w^{-i-ni_2,-j-nj_2}} \\ &\qquad - \hat{u}_{ij}\hat{w}_{i+ni_2,j+nj_2})\hat{v}_{i+ni_1,j+nj_1} \\ &\quad + \sum_{ij}^{(2)} (\overline{u^{-i,-j} v^{-i-ni_1,-j-nj_1}} \\ &\qquad - \hat{u}_{ij}\hat{v}_{i+ni_1,j+nj_1})\hat{w}_{i+ni_2,j+nj_2} \\ &\quad + \sum_{ij}^{(3)} (\overline{u^{-i-i_1,-j-nj_1} w^{-i-ni_2,-j-nj_2}} \\ &\qquad - \hat{u}_{ij}\hat{v}_{i+ni_1,j+nj_1}\hat{w}_{i+ni_2,j+nj_2}) \Bigg) \\ &= \hat{h}_{i_1 j_1 i_2 j_2} + \tilde{h}_{i_1 j_1 i_2 j_2}. \end{aligned} \tag{25}$$

We call the first term, denoted as $\hat{h}_{i_1 j_1 i_2 j_2}$, the triple correlation term, and the sum of the remaining terms, denoted as $\tilde{h}_{i_1 j_1 i_2 j_2}$, the correction term. In the remaining part of this section, we discuss their computation methods.

B. Triple Correlation Term

$\hat{h}_{i_1 j_1 i_2 j_2}$ in (25) can be rewritten as

$$\begin{aligned} \hat{h}_{i_1 j_1 i_2 j_2} &= \sum_{i'=1}^{n}\sum_{j'=1}^{n}\sum_{ij} \hat{u}_{i'+ni,j'+nj} \\ &\quad \times \hat{v}_{i'+n(i+i_1),j'+n(j+j_1)}\hat{w}_{i'+n(i+i_2),j'+n(j+nj_2)} \end{aligned} \tag{26}$$

$$\begin{aligned} M_{R,\Delta,\Delta'}(u,v,w) =& \Delta^2 \sum_{ij} \frac{M_{\square_{ij},\Delta'}(u)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(v)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(w)}{\Delta^2} \\ &+ \Delta^2 \sum_{ij}^{(2)} \left(\frac{M_{\square_{ij},\Delta'}(vw)}{\Delta^2} - \frac{M_{\square_{ij},\Delta'}(v)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(w)}{\Delta^2} \right) u(\boxdot_{ij}) \\ &+ \Delta^2 \sum_{ij}^{(2)} \left(\frac{M_{\square_{ij},\Delta'}(uw)}{\Delta^2} - \frac{M_{\square_{ij},\Delta'}(u)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(w)}{\Delta^2} \right) v(\boxdot_{ij}) \\ &+ \Delta^2 \sum_{ij}^{(2)} \left(\frac{M_{\square_{ij},\Delta'}(uv)}{\Delta^2} - \frac{M_{\square_{ij},\Delta'}(u)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(v)}{\Delta^2} \right) w(\boxdot_{ij}) \\ &+ \Delta^2 \sum_{ij}^{(3)} \left(\frac{M_{\square_{ij},\Delta'}(uvw)}{\Delta^2} - \frac{M_{\square_{ij},\Delta'}(u)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(v)}{\Delta^2}\frac{M_{\square_{ij},\Delta'}(w)}{\Delta^2} \right) \end{aligned} \tag{22}$$

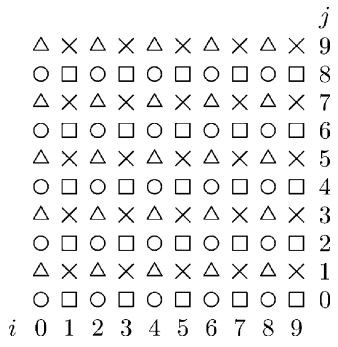


Fig. 6. As an example ($n = 2$), the summation $\sum_i \sum_j$ can be decomposed into for 4 summations on circles, squares, triangles and crosses. They are $\sum_{\text{even}i} \sum_{\text{even}j}$, $\sum_{\text{odd}i} \sum_{\text{even}j}$, $\sum_{\text{even}i} \sum_{\text{odd}j}$ and $\sum_{\text{odd}i} \sum_{\text{odd}j}$.

where we decompose a summation into a number of summations on grids with a bigger grid size as shown in Fig. 6 [29]. We can rewrite (26) as

$$\hat{h}_{i_1 j_1 i_2 j_2} = \sum_{i'=1}^{n} \sum_{j'=1}^{n} \sum_{ij} {}^n\hat{u}_{ij}^{-i',-j'} \times {}^n\hat{v}_{i+i_1,j+j_1}^{-i',-j'} \times {}^n\hat{w}_{i+i_2,j+j_2}^{-i',-j'} \quad (27)$$

where ${}^n\hat{u}_{ij} = \hat{u}_{ni,nj}$, $\hat{u}_{ij}^{i'j'} = \hat{u}_{i-i',j-j'}$ and the matrix ${}^n\hat{u}^{ij}$ is the contracted form of ${}^n(\hat{u}^{ij})$. We can see that the term

$$\sum_{ij} {}^n\hat{u}_{ij}^{-i',-j'} \times {}^n\hat{v}_{i+i_1,j+j_1}^{-i',-j'} \times {}^n\hat{w}_{i+i_2,j+j_2}^{-i',-j'} \quad (28)$$

in (27) is a discrete triple correlation.

We will show below that the discrete triple correlation can be efficiently computed by the fast Fourier transform (FFT). As a simple case, the continuous one-dimensional (1-D) triple correlation can computed by a two-dimensional (2-D) convolution [28]

$$\int_{-\infty}^{+\infty} \mathrm{d}f\, u(f)v(f+f')w(f+f'') = (\delta(f_1 - f_2)u(-f_1)) * (v(f_1)w(f_2)) \quad (29)$$

where $*$ is the convolution operator. Similarly, the discrete 2-D triple correlation can be computed by a 4-D discrete convolution

$$\sum_{ij} \hat{u}_{ij}\hat{v}_{i+i_1,j+j_1}\hat{w}_{i+i_2,j+j_2} = (\delta_{i_1 i_2}\delta_{j_1 j_2}\hat{u}_{-i_1,-j_1}) * (\hat{v}_{i_1 j_1}\hat{w}_{i_2 j_2}) \quad (30)$$

where δ_{ij} is the Kronecker delta. The convolution can be computed efficiently by the FFT.

C. The Correction Term

Using the definition in (25), we have a straightforward algorithm to compute the correction term $\tilde{h}_{i_1,j_1,i_2,j_2}$ (Algorithm 2). But this algorithm is slow because of the redundant computation in Line 5, 7 and 9. It can be seen that there can be multiple sets of i, j, i_1 and j_1 such that

$$\begin{cases} -i - i_1 = \hat{i}_1 \\ -j - j_1 = \hat{j}_1 \\ -i - i_2 = \hat{i}_2 \\ -j - j_2 = \hat{j}_2 \end{cases} \quad (31)$$

for any given $\hat{i}_1, \hat{j}_1, \hat{i}_2$ and $\hat{j}_2$. Therefore, $\overline{v^{-i-i_1,-j-j_1} w^{-i-i_2,-j-j_2}}$ in Line 5 of Algorithm 2 has to be computed multiple times for the same set of superscripts. The same observation is true for $\overline{u^{-i,-j} w^{-i-i_2,-j-j_2}}$ in Line 7 and $\overline{u^{-i,-j} v^{-i-i_1,-j-j_1}}$ in Line 9 as well.

Algorithm 2 Straightforward Correction Term Computation Algorithm

1: **function** CORRECTION(u, v, w, n)

2: **for all** i_1, j_1, i_2 and j_2 that are multiples of n **do**

3: $\tilde{h}_{i_1/n,j_1/n,i_2/n,j_2/n} \leftarrow 0$

4: **for all** i and j, where $u(x,y)$ is continuous in $\square_{ij}$, $v(x,y)$ is discontinuous in $\square_{i+i_1,j+j_1}$ and $w(x,y)$ is discontinuous in $\square_{i+i_2,j+j_2}$ **do**

5: $\tilde{h}_{i_1/n,j_1/n,i_2/n,j_2/n} \leftarrow (\overline{v^{-i-i_1,-j-j_1} w^{-i-i_2,-j-j_2}} - \hat{v}_{i+i_1,j+j_1}\hat{w}_{i+i_2,j+j_2})\hat{u}_{ij}$

6: **for all** i and j, where $u(x,y)$ is discontinuous in $\square_{ij}$, $v(x,y)$ is continuous in $\square_{i+i_1,j+j_1}$ and $w(x,y)$ is discontinuous in $\square_{i+i_2,j+j_2}$ **do**

7: $\tilde{h}_{i_1/n,j_1/n,i_2/n,j_2/n} \leftarrow (\overline{u^{-i,-j} w^{-i-i_2,-j-j_2}} - \hat{u}_{i,j}\hat{w}_{i+i_2,j+j_2})\hat{v}_{i+i_1,j+j_1}$

8: **for all** i and j, where $u(x,y)$ is discontinuous in $\square_{ij}$, $v(x,y)$ is discontinuous in $\square_{i+i_1,j+j_1}$ and $w(x,y)$ is continuous in $\square_{i+i_2,j+j_2}$ **do**

9: $\tilde{h}_{i_1/n,j_1/n,i_2/n,j_2/n} \leftarrow (\overline{u^{-i,-j} v^{-i-i_1,-j-j_1}} - \hat{u}_{i,j}\hat{v}_{i+i_1,j+j_1})\hat{w}_{i+i_2,j+j_2}$

10: **for all** i and j, where $u(x,y)$ is discontinuous in $\square_{ij}$, $v(x,y)$ is discontinuous in $\square_{i+i_1,j+j_1}$ and $w(x,y)$ is discontinuous in $\square_{i+i_2,j+j_2}$ **do**

11: $\tilde{h}_{i_1/n,j_1/n,i_2/n,j_2/n} \leftarrow \overline{u^{-i,-j} v^{-i-i_1,-j-j_1} w^{-i-i_2,-j-j_2}} - \hat{u}_{i,j}\hat{v}_{i+i_1,j+j_1}\hat{w}_{i+i_2,j+j_2}$

In order to reduce the unnecessary computation, we transform the indexes using

$$\begin{cases} i + i_1 \rightarrow i_1 \\ j + j_1 \rightarrow j_1 \\ i + i_2 \rightarrow i_2 \\ j + j_2 \rightarrow j_2. \end{cases} \quad (32)$$

The details are shown in Algorithm 3. Note that the recursive integration is called only once for any set of superscripts in Line

6, 14 and 18 in Algorithm 3, therefore the runtime is improved compared with Algorithm 2.

Algorithm 3 Improved Correction Term Computation Algorithm

1: **function** CORRECTION(u, v, w, n)

2: **for all** i_1, j_1, i_2 and j_2 **do**

3: $\tilde{h}_{i_1 j_1 i_2 j_2} \leftarrow 0$

4: **for all** i_1 and j_1, where $v(x, y)$ is discontinuous in $\square_{i_1 j_1}$ **do**

5: **for all** i_2 and j_2, where $i_1 - i_2$ and $j_1 - j_2$ are multiples of n, and $w(x, y)$ is discontinuous in $\square_{i_2 j_2}$ **do**

6: $t \leftarrow \overline{v^{-i_1,-j_1} w^{-i_2,-j_2}} - \hat{v}_{i_1,j_1} \hat{w}_{i_2 j_2}$

7: **for all** i and j, where $i_1 - i$ and $j_1 - j$ are multiples of n **do**

8: **if** $u(x, y)$ is continuous in $\square_{ij}$ **then**

9: $\tilde{h}_{(i_1-i)/n,(j_1-j)/n,(i_2-i)/n,(j_2-j)/n}$ += $t \times \hat{u}_{ij}$

10: **else if** $u(x, y)$ is discontinuous in $\square_{ij}$ **then**

11: $\tilde{h}_{(i_1-i)/n,(j_1-j)/n,(i_2-i)/n,(j_2-j)/n}$ += $\overline{u^{-i,-j} v^{-i_1,-j_1} w^{-i_2,-j_2}} - \hat{u}_{ij} \hat{v}_{i_1 j_1} \hat{w}_{i_2 j_2}$

12: **for all** i and j, where $u(x, y)$ is discontinuous in $\square_{ij}$ **do**

13: **for all** i_2 and j_2, where $i_2 - i$ and $j_2 - j$ are multiples of n, and $w(x, y)$ is discontinuous in $\square_{i_2 j_2}$ **do**

14: $t \leftarrow \overline{u^{-i,-j} w^{-i_2,-j_2}} - \hat{u}_{ij} \hat{w}_{i_2 j_2}$

15: **for all** i_1 and j_1, where $i_1 - i$ and $j_1 - j$ are multiples of n, and $v(x, y)$ is continuous in $\square_{i_1 j_1}$ **do**

16: $\tilde{h}_{(i_1-i)/n,(j_1-j)/n,(i_2-i)/n,(j_2-j)/n}$ += $t \times \hat{v}_{i_1 j_1}$

17: **for all** i_1 and j_1, where $i_1 - i$ and $j_1 - j$ are multiples of n, and $v(x, y)$ is discontinuous in $\square_{i_1 j_1}$ **do**

18: $t \leftarrow \overline{u^{-i,-j} v^{-i_1,-j_1}} - \hat{u}_{ij} \hat{v}_{i_1 j_1}$

19: **for all** i_2 and j_2, where $i_2 - i$ and $j_2 - j$ are multiples of n, and $w(x, y)$ is continuous in $\square_{i_2 j_2}$ **do**

20: $\tilde{h}_{(i_1-i)/n,(j_1-j)/n,(i_2-i)/n,(j_2-j)/n}$ += $t \times \hat{w}_{i_2 j_2}$

V. Experimental Results

We implement the simulator ELIAS in C++. The simulation platform is a 2.8-GHz Pentium-4 Linux machine. The lithography settings are a normal quadrupole illumination with the parameters $\sigma_{\text{center}} = 0.92$ and $\sigma_{\text{radius}} = 0.15$, and a circular pupil.

We denote the method using the correction term the "new" method, and the method using only the triple correction term the "old" method. We show the accuracy and the runtime of both methods. We demonstrate that the new method is much faster than the old method for the same accuracy requirements.

A. Accuracy Verification

We denote the exact solution and the simulation result of $\mathcal{T}(i_1\tilde{\Delta}, j_1\tilde{\Delta}, i_2\tilde{\Delta}, j_2\tilde{\Delta})$ as $\mathcal{T}_{i_1 j_1 i_2 j_2}$ and $\widetilde{\mathcal{T}}_{i_1 j_1 i_2 j_2}$. We denote the error as

$$E_{i_1 j_1 i_2 j_2} = \left| \widetilde{\mathcal{T}}_{i_1 j_1 i_2 j_2} - \mathcal{T}_{i_1 j_1 i_2 j_2} \right|.$$

The worst case (WC) error is defined as

$$E_{\text{WC}} = \max_{i_1 j_1 i_2 j_2} E_{i_1 j_1 i_2 j_2}.$$

The root mean square (RMS) error is defined as

$$E_{\text{RMS}} = \sqrt{\frac{1}{N} \sum_{i_1 j_1 i_2 j_2} E^2_{i_1 j_1 i_2 j_2}}.$$

where N is the number of nonzero $\mathcal{T}_{i_1 j_1 i_2 j_2}$. In the experiments, we took $\tilde{\Delta} = 0.1$.

As we have shown in Theorem 2, the error of TCC is contributed by the internal regions ($C_1\Delta^2$ terms) and the boundaries ($C_2\Delta'$). However, if the integrand is a linear function over the internal regions, the error is only contributed by the boundaries. To analyze both types of errors, we consider an infocus case, where the integrand is constant, and a defocused case ($z = 100\,\text{nm}$), where the integrand is in general not a linear function. In the infocus case, all the errors come from boundaries. In the defocused case, the errors come from both boundaries and internal regions, but we can reduce errors from boundaries by reducing the minimum recursive integration grid size Δ'. From the infocus case, we can determine how small Δ' should be in order to make the errors from boundaries small enough. With a small enough Δ', all the errors practically come from internal regions in the defocused case. By this way, we separate the two types of errors.

For the infocus case, we use the method from [15] to generate the exact solution. It essentially converts TCC region integrals to line integrals, which can be computed analytically. Therefore, it produces results that do not have truncation errors.

Fig. 7 shows the errors in the new method as functions of Δ'. Obviously, the errors always decrease as Δ' decreases and can be reduced substantially small. Fig. 8 shows the errors in the old method for different n. We can see that the ratio between E_{WC} and E_{RMS} of the old method is a few times bigger (about 5) that the ratio of the new method, which means the TCC matrix errors of the latter case is more evenly distributed than those of the former case. Since $\tilde{\Delta}$ is the same for both methods, when the minimum square size of the old method $\Delta = \tilde{\Delta}/n$ and the minimum square size of the new method Δ' ($\Delta = \tilde{\Delta}$, since $n = 1$ for this case) the same, we should have approximately the same errors. This relation is confirmed by the data replotted in Fig. 9 (a combination of Figs. 7 and 8).

For the defocused case, since there is no analytical solution available in the literature, we chose the results computed with a small enough $\Delta' = 1 \times 10^{-4}$ and a big decimation factor $n = 200$ as a close approximation to an analytical solution. As shown in Fig. 9, $\Delta' = 1 \times 10^{-4}$ is small enough to bound

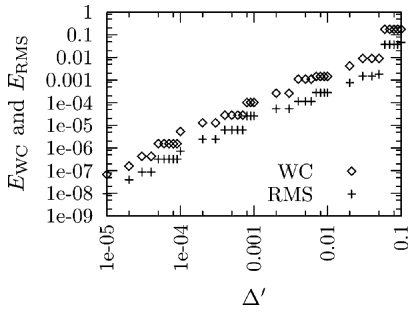


Fig. 7. Errors for different Δ' of the new method ($n = 1$) (the infocus case).

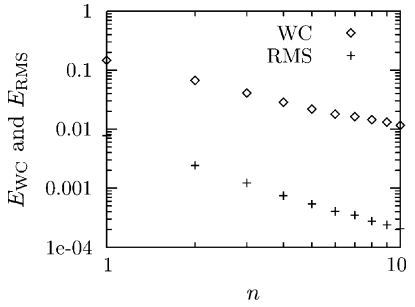


Fig. 8. Errors for different n (ratio of the TCC matrix grid size $\bar{\Delta}$ and the simulation grid size, see (26)) of the old method (the infocus case).

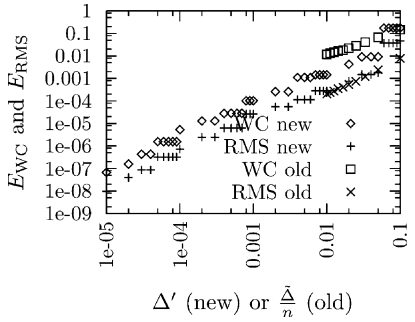


Fig. 9. Errors of the new (with respect to Δ', $n = 1$) and old (with respect to $\bar{\Delta}/n$, see (26) for the definition of n) methods, where $\bar{\Delta} = 0.1$ (the infocus case).

the errors due to boundaries to the order of about 1×10^{-6}, which is practically very small. In this case, when all the errors from internal regions are much larger than 1×10^{-6}, we can ignore the errors from boundaries as if all the errors are from internal regions. According to Theorem 2 (see the $C_1\Delta^2$ term), the errors shall follow power-laws of n. In Fig. 10, the errors of the old method indeed follow power-laws of n, when n is between 1 and 100. The errors of the new method follow power-laws of n up to $n \approx 40$. Beyond $n \approx 40$, the errors of the new method are of the order of 1×10^{-6}, in which case the errors from boundaries (when $\Delta' = 1 \times 10^{-4}$) are longer negligible. It is clear to see in Fig. 9 that the errors of the old method are much greater than the errors of the new method for the same n.

B. Runtime Characteristics

The runtime for the defocusd case shall be the same as that of the infocus case, for the same parameters $\tilde{\Delta}$, n, and Δ', because

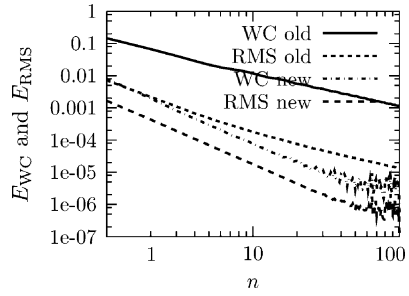


Fig. 10. Errors of the old and new methods with $\Delta' = 1 \times 10^{-4}$, where $z = 100\,\mathrm{nm}$. Compared with the new method with $n = 200$ (see (26) for the definition of n) and $\Delta' = 1 \times 10^{-4}$ (the defocused case).

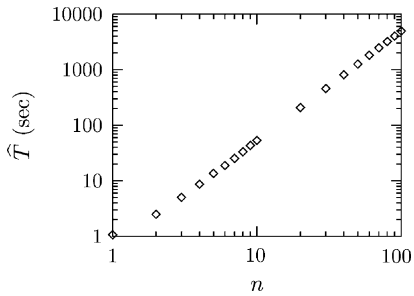


Fig. 11. T_{conv} as a function of n (see (26) for the definition of n), where $\bar{\Delta} = 0.1$. $T_{\mathrm{conv}} \propto n^2$.

the same program can be used for both cases. Therefore, we will only show the runtime for the infocus case.

Let us denote the runtime of the computation of the triple correction term using the convolution as T_{conv} and the runtime of the computation of the correction term using Algorithm 3 as T_{corr}. Obviously the runtimes of both methods are known if T_{conv} and T_{corr} are known. Below, we show how the parameters $\tilde{\Delta}$ and n affect the runtime T_{conv} and how the parameters $\tilde{\Delta}$, n and Δ' affect the runtime T_{corr}.

Fig. 11 shows T_{conv} as a function of n, which demonstrates the relation

$$T_{\mathrm{conv}} \propto n^2. \tag{33}$$

This is because the runtime for the discrete correlation (28) does not depend on n but there are n^2 such terms in (27). Fig. 12 shows T_{conv} as a function of $\tilde{\Delta}$ or equivalently Δ, since $n = 1$. The runtime T_{conv} is dominated by the FFT used in the convolution, which can be written as

$$T_{\mathrm{conv}} \propto \frac{1}{\Delta^4} \log\left(\frac{C}{\Delta}\right)^4 \tag{34}$$

where C is some constant. For small enough Δ compared with C, the change in the log term due to the change in Δ is less important than the term in front of it. Therefore, we can take the log term as a constant and we have

$$T_{\mathrm{conv}} \propto \frac{1}{\Delta^4} \tag{35}$$

which is consistent with the data in Fig. 12.

Since the majority of the runtime T_{corr} is taken by the recursive integration, T_{corr} shall be related to Δ' to a power between

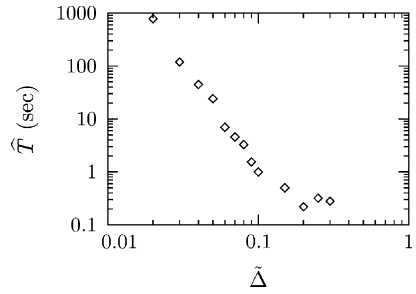


Fig. 12. T_{conv} as a function of $\tilde{\Delta}(n = 1)$.

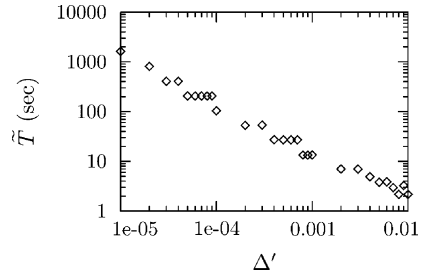


Fig. 13. T_{corr} as a function of Δ' ($\tilde{\Delta} = 0.1$ and $n = 1$).

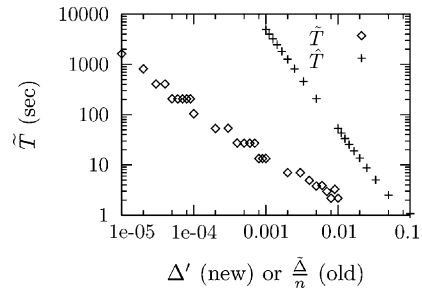


Fig. 14. Runtimes of the new (with respect to Δ', $n = 1$) and old (with respect to $\tilde{\Delta}/n$) methods, where $\tilde{\Delta} = 0.1$.

0 and 2 as discussed in Section III-B. Fig. 13 shows the runtime T_{corr} as a function of Δ' ($n = 1$), which can be approximately written as

$$T_{\text{corr}} \propto \frac{1}{\Delta'} \tag{36}$$

where the power is about 1. Fig. 14 is a combination of Figs. 11 and 13. As we have shown that when Δ' of the new method and $\tilde{\Delta}/n$ of the old method are the same, these two methods generate results of approximately the same accuracy for the infocus case. Fig. 14 shows that the runtime of the new method can be much faster than that of the old method for the same accuracy. Fig. 15 shows the runtime T_{corr} as a function of $\tilde{\Delta}$ or equivalently Δ, since $n = 1$. The power law relation between T_{corr} and Δ is due to the fact that the number of the recursive integrations that are computed is proportional to the number of the boundary squares, which is inversely proportional to the square size Δ.

Fig. 16 shows the runtime for the correction term as a function of n. We can see that there is an optimal n which gives the minimum T_{corr}. This can be explained in Fig. 17. When n is small, Δ can be as big as $\tilde{\Delta}$ and there will be some unnecessary recursive integration function calls (represented by gray dots and

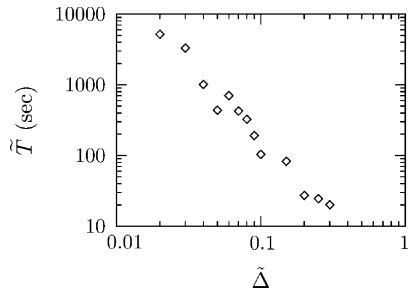


Fig. 15. T_{corr} as a function of $\tilde{\Delta}$ ($n = 1$ and $\Delta' = 1 \times 10^{-4}$).

lines) compared with the case where n is median. When n is big, Δ can be as small as Δ' and the number of the recursive integration function calls reaches the maximum—$(\tilde{\Delta}/\Delta')^2$. Therefore, an optimal runtime is achieved for some median n between 1 and $\tilde{\Delta}/\Delta'$.

Since T_{conv} increases quadratically with the decrease in the minimum square size (related with n, see (33)), while T_{corr} increases linearly with the decrease in the minimum square size (related with Δ', see (36)), T_{conv} will be bigger than T_{corr} for a small minimum square size. The old method is slower than the new method in this case. For example, the new method with $n = 1$ and $\Delta' = 1 \times 10^{-4}$ give results with the same accuracy as the old method with $n = 1000$ for the infocus case. According to Fig. 11, we have $T_{\text{conv}}(n = 1) \approx 1$ sec, and by extrapolation, we have $T_{\text{conv}}(n = 1000) \approx 3 \times 10^5$ sec. According to Fig. 13, we have $T_{\text{corr}}(n = 1, \Delta' = 1 \times 10^{-4}) \approx 100$ sec. Therefore, the new method speeds up the runtime $3000\times$ of times for the infocus case.

If we choose $\Delta' = 1 \times 10^{-4}$, the error introduced by boundaries in the infocus case can be estimated as 1×10^{-6} (see Fig. 9), which is also an estimate of the error introduced by boundaries in the defocused case. We require that the total error is bounded to the same order. Therefore, we need to take n about 1000 in the old method, and to take n at least 40 in the new method (see Fig. 10). According to Fig. 16, $T_{\text{corr}}(n = 40, \Delta' = 1 \times 10^{-4}) \approx 40$ sec; according to Fig. 11, $T_{\text{conv}}(n = 40) \approx 1 \times 10^3$ sec; and we have estimated $T_{\text{conv}}(n = 1000) \approx 3 \times 10^5$ sec. Therefore, the new method speeds up the runtime hundreds of times for the defocused case.

C. Application to Aerial Image Simulation

In Hopkins equation, the TCC matrix can be used directly to simulate aerial images. We use this equation because the errors in aerial images are only due to the errors in the TCC matrix, which is ideal for the quantification of the aerial simulation errors solely introduced by TCC errors. We show below how much aerial image errors are for given amounts of TCC computation time.

Here, we simulate an isolated via of size 105 nm, where the background transmittance is 1 and the feature transmittance is 0. We still use the quadrupole illumination that we mentioned previously. The numerical aperture $\text{NA} = 0.8$ and the wavelength $\lambda = 193$ nm. We choose $\tilde{\Delta} = 0.1$. CD is measured at the threshold of 0.6.

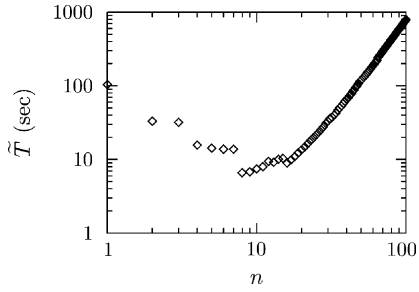


Fig. 16. T_{corr} as a function of n ($\bar{\Delta} = 0.1$ and $\Delta' = 1 \times 10^{-4}$).

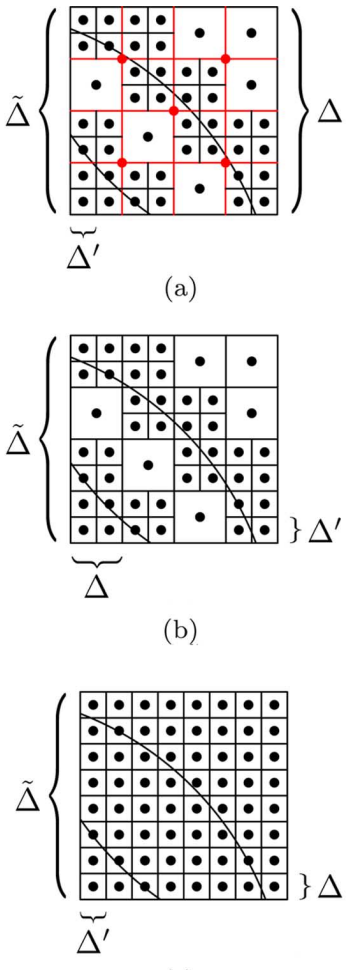


Fig. 17. The recursive integration for different n. (a) n is small ($\Delta = \bar{\Delta}$). (b) n is big ($\Delta = \Delta'$). (c) n is somewhere in between ($\Delta = \frac{1}{2}\bar{\Delta} = 2\Delta'$).

Figs. 18 and 19 show the CD errors as a function of TCC computation runtime using both the old and new methods for the infocus case and the defocused case, respectively. It is easy to see that the CD errors of the new method are much less than the error from the old method with the same amount TCC computation time. The new method can give almost accurate results, for example, 1×10^{-4} nm CD error as shown in Fig. 19, with about an hour TCC computation time. For the same accuracy requirements, the runtime of the old method can be estimated as about a hundred years by extrapolation. Therefore, the new method can be used to benchmark other lithography simulators.

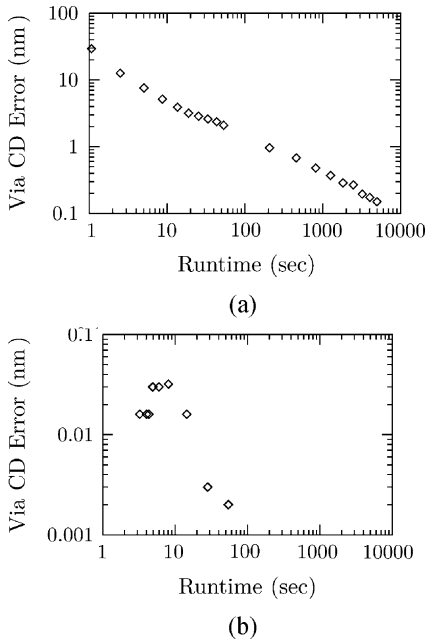


Fig. 18. CD errors versus the TCC computation runtime (the infocus case). (a) The old method. (b) The new method ($n = 1$ and $\Delta' = 1 \times 10^{-4}$). No data points are shown, when Runtime is over 100 s, because the CD errors are almost zero under these conditions.

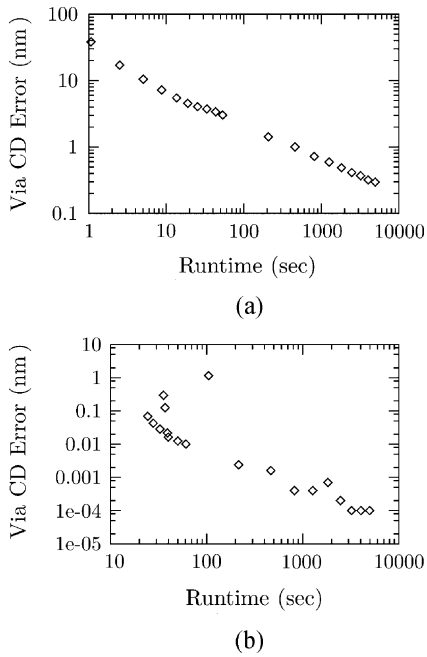


Fig. 19. CD errors versus the TCC computation runtime (the defocused case). (a) The old method. (b) The new method ($n = 1$ and $\Delta' = 1 \times 10^{-4}$).

VI. Conclusion

It is very important to reduce lithography simulator error as technology improves. We find the major error contributor in conventional transmission cross coefficient (TCC) computation method. We improve the accuracy by using a recursive integration method and by using a previously overlooked correction term. We implement the algorithm in an open-source software

package ELIAS. The simulation accuracy and runtime are significantly improved. It is flexible to take arbitrary lithography conditions can be used to benchmark other aerial image simulators, which is essential for nanometer design-for-manufacturability.

APPENDIX
PROOFS OF THEOREMS

Proof 1 (Theorem 1): We use D to denote the support of $u(x,y)$ and ∂D to denote the boundary of the support. And we denote the bounds of $u(x,y)$ and its first and second derivatives in R as

$$|u| \leqslant \zeta \tag{37}$$

$$\left|\frac{\partial u}{\partial x}\right| \leqslant \eta \quad \text{and} \quad \left|\frac{\partial u}{\partial y}\right| \leqslant \eta \tag{38}$$

$$\left|\frac{\partial^2 u}{\partial x^2}\right| \leqslant \theta \quad \text{and} \quad \left|\frac{\partial^2 u}{\partial y^2}\right| \leqslant \theta \tag{39}$$

where ζ, η and θ are all constants.

The truncation error of $M_{R,\Delta}(u)$ (see (12)) can be written as

$$|I_R(u) - M_{R,\Delta}(u)| \leqslant \sum_{ij}^{(0)} |I_{\square_{ij}}(u) - \Delta^2 u(\boxdot_{ij})| + \sum_{ij}^{(1)} |I_{\square_{ij}}(u) - \Delta^2 u(\boxdot_{ij})| \tag{40}$$

where the superscripts of $\sum$'s indicate how many functions in the integrand are discontinuous and $I_\square(u)$ denotes the integration of u over the square $\square$

$$I_\square(u) = \iint_\square u(x,y)\,\mathrm{d}x\mathrm{d}y.$$

The first term on the right-hand side of (40) is summed over all squares where u is smooth, and the second term is summed over all squares where u is discontinuous.

The truncation error for each square can be described as in the following two cases.

1) The function $u(x,y)$ is smooth in the square $\square$. According to the Taylor's theorem,

$$u(x,y) = u(x_0,y_0) + \Big((x-x_0)\frac{\partial}{\partial x} + (y-y_0)\frac{\partial}{\partial y}\Big)u(x_0,y_0) + \frac{1}{2}\Big((x-x_0)\frac{\partial}{\partial x} + (y-y_0)\frac{\partial}{\partial y}\Big)^2 u(x^*,y^*)$$

where (x_0,y_0) is the square center $\boxdot$, and (x^*,y^*) is a point satisfying

$$(x^*-x_0, y^*-y_0) = (\lambda(x-x_0), \lambda(y-y_0)),\ 0<\lambda<1.$$

Therefore, we have the truncation error

$$\begin{aligned}|I_\square(u) - \Delta^2 u(x_0,y_0)| &\leqslant \left|\iint_\square (u(x,y)-u(x_0,y_0))\,\mathrm{d}x\mathrm{d}y\right| \\ &= \iint_\square \frac{1}{2}\Big((x-x_0)^2\frac{\partial^2}{\partial x^2} + (y-y_0)^2\frac{\partial^2}{\partial y^2}\Big) \times u(x^*,y^*)\,\mathrm{d}x\mathrm{d}y \\ &\leqslant \frac{\theta}{2}\iint_\square ((x-x_0)^2+(y-y_0)^2)\,\mathrm{d}x\mathrm{d}y = \frac{\theta}{12}\Delta^4. \end{aligned} \tag{41}$$

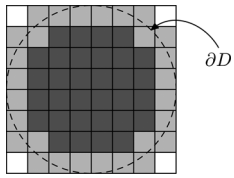


Fig. 20. The support D of the function is the region enclosed by the circle which is denoted as ∂D. The summation $\sum_{ij}^{(0)}$ is over the dark gray squares, and the summation $\sum_{ij}^{(1)}$ is over the light gray squares.

2) The function $u(x,y)$ is discontinuous in the square $\square$. The truncation error can be estimated as

$$\begin{aligned}|I_\square(u) - \Delta^2 u(x_0,y_0)| &\leqslant \iint_\square |u(x,y)-u(x_0,y_0)|\,\mathrm{d}x\mathrm{d}y \\ &\leqslant \iint_\square \zeta\,\mathrm{d}x\mathrm{d}y = \zeta\Delta^2. \end{aligned} \tag{42}$$

By using (40)–(42), we can derive that the truncation error of $M_{R,\Delta}(u)$ is bounded as follows:

$$|I_R(u) - M_{R,\Delta}(u)| \leqslant \frac{\theta}{12}\sum_{ij}^{(0)}\Delta^4 + \zeta\sum_{ij}^{(1)}\Delta^2.$$

As an example, we show the support of a function in Fig. 20. The summations $\sum_{ij}^{(0)}$ and $\sum_{ij}^{(1)}$ are indicated by the dark gray squares and the light gray squares. It is obvious that the number of dark gray squares is bounded by $A(D)/\Delta^2$, where $A(D)$ is the total area of the support D, and the number of light gray squares is bounded by $C_l l(\partial D)/\Delta$, where C_l is a constant and $l(\partial D)$ is the length of the boundary ∂D. Therefore, the truncation error can be estimated as

$$|I_R(u) - M_{R,\Delta}(u)| \leqslant C_1\Delta^2 + C_2\Delta \tag{43}$$

where $C_1 = A(D)\theta/12$ and $C_2 = C_l l(\partial D)\zeta$. ■

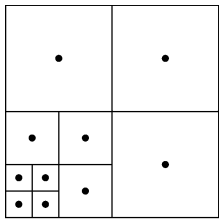
Proof 2 (Theorem 2): According to Algorithm 1, $M_{\square,\Delta'}(u)$ can be written as

$$\begin{aligned}M_{\square,\Delta'}(u) &= \sum_k \Delta_k^2 \sum_i u(\boxdot_{k,i}) \\ &= \sum_{ki}^{(0)} \Delta_k^2 u(\boxdot_{k,i}) + \sum_i^{(1)} \Delta_0^2 u(\boxdot_{0,i})\end{aligned}$$

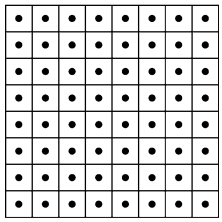
where Δ_k denotes the square size and the subscript $_{k,i}$ is the index of a not-divided square of size Δ_k. The superscript of the summation sign again denotes whether u is smooth or discontinuous. Therefore, we have the truncation error

$$\begin{aligned}|I_\square(u) - M_{\square,\Delta'}(u)| &\leqslant \sum_{ki}^{(0)} |I_{\square_{k,i}}(u) - \Delta_k^2 u(\boxdot_{k,i})| + \sum_i^{(1)} |I_{\square_{0,i}}(u) - \Delta_0^2 u(\boxdot_{0,i})| \\ &\leqslant \sum_{ki}^{(0)} \frac{\theta}{12}\Delta_k^4 + \sum_i^{(1)} \zeta\Delta_0^2.\end{aligned}$$

Using the inequality



(a) $C_d = 1$



(b) $C_d = 4$

Fig. 21. Two extreme cases of the recursive quadrisection of a square.

$$\sum_i x_i^2 \leqslant \left(\sum_i x_i\right)^2, \quad \text{for } x_i \geqslant 0$$

we have

$$\sum_{ki}^{(0)} \Delta_k^4 \leqslant \left(\sum_{ki}^{(0)} \Delta_k^2\right)^2 \leqslant \left(\sum_{ki} \Delta_k^2\right)^2 \leqslant \left(\Delta^2\right)^2 = \Delta^4.$$

Similar to Proof 1, the number of terms in the summation $\sum_i^{(1)}$ is bounded by $C_l l_\square(D)/\Delta_0$, where C_l is the constant that we mentioned in Proof 1 and $l_\square(\partial D)$ is the length of the boundary ∂D in the square $\square$. Therefore, we have

$$\begin{aligned}|I_\square(u) - M_{\square,\Delta'}(u)| &\leqslant \frac{\theta}{12}\Delta^4 + \frac{C_l l_\square(\partial D)}{\Delta_0}\zeta\Delta_0^2 \\ &\leqslant \frac{\theta}{12}\Delta^4 + C_l l_\square(\partial D)\zeta\Delta'. \quad (44)\end{aligned}$$

We can then derive that the truncation error of $M_{R,\Delta,\Delta'}(u)$ is bounded as

$$|I_R(u) - M_{R,\Delta,\Delta'}(u)| \leqslant C_1\Delta^2 + C_2\Delta'. \quad (45)$$

where $C_1 = A(D)\theta/12$ and $C_2 = C_l l(\partial D)\zeta$. ■

Proof 3 (Theorem 3): Let T_i be the runtime of Algorithm 1 for a square of the size Δ_i Because Algorithm 1 is a recursive algorithm, we can approximate T_i by a recursive sequence

$$T_{i+1} = C_\mathrm{d} T_i + (4 - C_\mathrm{d})T_0 = C_\mathrm{d} T_i + b$$

where C_d denote the number of the smaller squares of size Δ_i that needs to be further divided and $b = (4 - C_\mathrm{d})T_0$, where T_0 is the runtime of the midpoint rule for a square that is not divided.

The constant C_d is bounded ($1 \leqslant C_\mathrm{d} < 4$) practically:

1) Fig. 21(a) shows the case for $C_\mathrm{d} = 1$, which rarely happens in practice. The recursion will not be sustained if C_d is smaller than 1.
2) $C_\mathrm{d} = 4$ means that each squares is quadrisected unless it is smaller enough. This is equivalent to use a uniform grid as in Fig. 21(b), which is impossible, because cutting all the squares of a small enough size means that the curve is not simple.

We approximate C_d for different i's by a same constant C_d ($0 < C_\mathrm{d} < 2$), which is some kind of "average" over all i's. We can transform the recursive relation to

$$T_{i+1} - \beta = C_\mathrm{d}(T_i - \beta)$$

where $\beta = b/1 - C_\mathrm{d}$. Therefore,

$$T_n = C_\mathrm{d}^n(T_0 - \beta) + \beta = \alpha C_\mathrm{d}^n + \beta$$

where $\alpha = T_0 - \beta$. The level of recursion n can be approximated as $n = \log \Delta/\Delta'$. Therefore, the runtime of the recursive integration (15) over a square $\square$ of size Δ is

$$T_{\square,\Delta'} = \alpha C_\mathrm{d}^{\log \Delta/\Delta'} = \alpha\left(\frac{\Delta}{\Delta'}\right)^{\log C_\mathrm{d}} \quad (46)$$

where the additive constant β is ignored for large Δ/Δ' and $0 < \log C_\mathrm{d} < 2$. ■

Proof 4 (Theorem 4): By distinguishing whether $u(x,y)$ $v(x,y)$ $w(x,y)$ are discontinuous or not, we can approximate $I_R(uvw)$ as

$$\begin{aligned}&M_{R,\Delta,\Delta'}(u,v,w) \\ &= \Delta^2 \sum_{ij}^{(0)} u(\boxdot_{ij})v(\boxdot_{ij})w(\boxdot_{ij}) + \sum_{ij}^{(1)} M_{\square_{ij},\Delta'}(u, vw) \\ &\quad + \sum_{ij}^{(1)} M_{\square_{ij},\Delta'}(v, uw) + \sum_{ij}^{(1)} M_{\square_{ij},\Delta'}(w, uv) \\ &\quad + \sum_{ij}^{(2)} M_{\square_{ij},\Delta'}(vw, u) + \sum_{ij}^{(2)} M_{\square_{ij},\Delta'}(uw, v) \\ &\quad + \sum_{ij}^{(2)} M_{\square_{ij},\Delta'}(uv, w) + \sum_{ij}^{(3)} M_{\square_{ij},\Delta'}(uvw), \quad (47)\end{aligned}$$

where the superscripts $^{(n)}$ ($n = 0,1,2,3$) of the summation signs denote the number of functions of u, v and w that are discontinuous in $\square_{ij}$ and the discontinuous functions are in the left arguments of $M_{\square_{ij},\Delta'}(\cdot,\cdot)$.

The truncation error of (21) can be written as

$$\begin{aligned}&|I_\square(uv) - M_{\square,\Delta'}(u,v)| \\ &= \left| \iint_\square u(x,y)v(x,y)\,\mathrm{d}x\mathrm{d}y \right. \\ &\qquad \left. - \iint_\square \frac{M_{\square,\Delta'}(u)}{\Delta^2} v(x_0,y_0)\,\mathrm{d}x\mathrm{d}y \right| \\ &\leqslant \underbrace{\left| \iint_\square \left(u(x,y) - \frac{M_{\square,\Delta'}(u)}{\Delta^2}\right) v(x,y)\,\mathrm{d}x\mathrm{d}y \right|}_{\text{see}(48)} \\ &\qquad + \underbrace{\left| \iint_\square \frac{M_{\square,\Delta'}(u)}{\Delta^2}(v(x,y) - v(x_0,y_0))\,\mathrm{d}x\mathrm{d}y \right|}_{\text{see}(49)}.\end{aligned}$$

Using Taylor's theorem, we have

$$\begin{aligned}&\left| \iint_\square \left(u(x,y) - \frac{M_{\square,\Delta'}(u)}{\Delta^2}\right) v(x,y)\,\mathrm{d}x\mathrm{d}y \right| \\ &\leqslant \left| \iint_\square \left(u(x,y) - \frac{M_{\square,\Delta'}(u)}{\Delta^2}\right) v(x_0,y_0)\,\mathrm{d}x\mathrm{d}y \right|\end{aligned}$$

$$+\left|\iint_{\square}\Big(u(x,y)-\frac{M_{\square,\Delta'}(u)}{\Delta^2}\Big)\right.$$
$$\left.\times\Big((x-x_i)\frac{\partial}{\partial x}+(y-y_i)\frac{\partial}{\partial y}\Big)v(x^*,y^*)\,\mathrm{d}x\mathrm{d}y\right|$$
$$\leqslant\zeta_v|I_{\square}(u)-M_{\square,\Delta'}(u)|$$
$$+\zeta_u\eta_v\iint_{\square}(|x-x_i|+|y-y_i|)\,\mathrm{d}x\mathrm{d}y$$
$$\leqslant\zeta_v\Big(\frac{\theta_u}{12}\Delta^4+Cl_{\square}(D_u)\zeta_u\Delta'\Big)+\zeta_u\eta_v\frac{\Delta^3}{2}. \quad (48)$$

where we have used (44). Here, ζ, η and θ are still the bounds of functions and their derivatives, and their subscripts denote what the functions are. We also have

$$\left|\iint_{\square}\frac{M_{\square,\Delta'}(u)}{\Delta^2}(v(x,y)-v(x_0,y_0))\,\mathrm{d}x\mathrm{d}y\right|$$
$$\leqslant\zeta_u|I_{\square}(v)-M_{\square}(v)|=\frac{\zeta_u\theta_v}{12}\Delta^4 \quad (49)$$

where we have used (41). Ignoring (49), which is bounded by a higher order term of Δ, the truncation error of (21) is bounded as

$$|I_{\square}(uv)-M_{\square,\Delta'}(u,v)|\leqslant Cl_{\square}(D_u)\zeta_u\zeta_v\Delta'+\zeta_u\eta_v\frac{\Delta^3}{2}. \quad (50)$$

Let $f(x,y)=u(x,y)v(x,y)w(x,y)$, we have

$$I_R(f)=\sum_{ij}^{(0)}I_{\square_{ij}}(f)+\sum_{ij}^{(1)}I_{\square_{ij}}(f). \quad (51)$$

It is clear that the number of terms in the summations $\sum_{ij}^{(1)}$, $\sum_{ij}^{(2)}$ and $\sum_{ij}^{(3)}$ in (47) is the same as the number of terms in the summation $\sum_{ij}^{(1)}$ of (51), which is bounded by $Cl(\partial D)/\Delta$. Here, D is the support of the function $u(x,y)v(x,y)w(x,y)$.

Using (41), (50) and (44), the truncation error of (47) can be estimated as

$$|I_{\square}(uvw)-M_{R,\Delta,\Delta'}(u,v,w)|$$
$$\leqslant\frac{A(D)}{\Delta^2}C_1\Delta^4+Cl(\partial D)C_2\Delta'+\frac{Cl(\partial D)}{\Delta}C_3\Delta^3$$

where C_1, C_2 and C_3 are constants depending on the bounds on the functions and their first and second order derivatives. Therefore,

$$|I_{\square}(uvw)-M_{R,\Delta,\Delta'}(u,v,w)|$$
$$\leqslant(A(D)C_1+Cl(\partial D)C_3)\Delta^2+Cl_R(D_{uvw})C_2\Delta' \quad (52)$$

which is of the same order as (45).

Noting that $M_{\square,\Delta'}(u)$ is the same as $\Delta^2u(\boxdot)$ if $u(x,y)$ is smooth in the square $\square$, we can easily derive (22) with some simple mathematical manipulations. ■

REFERENCES

[1] G. A. Gomba, "Collaborative innovation: IBM's immersion lithography strategy for 65 nm and 45 nm half-pitch nodes & beyond," in *Proc. SPIE 6521*, 2007.

[2] A. K.-K. Wong, *Resolution Enhancement Techniques in Optical Lithography*. Bellingham, WA: SPIE, 2001.

[3] J. Wiley, "Future challenges in computational lithography," *Solid State Technology*, vol. 49, no. 5, p. 68, May 2006.

[4] C. Albertalli and T. Kingsley, "Computational lithography (cover story)," *Semicond. Int.*, vol. 30, no. 5, pp. 36–42, May 2007.

[5] T. Kingsley, J. Sturtevant, S. McPherson, and M. Sexton, "Advances in compute hardware platforms for computational lithography," in *Proc. SPIE 6520*, Mar. 2007, p. 652018.

[6] C. Tabery, H. Morokuma, R. Matsuoka, L. Page, G. E. Bailey, I. Kusnadi, and T. Do, "SEM image contouring for OPC model calibration and verification," in *Proc. SPIE*, Mar. 2007, vol. 6520, p. 652019.

[7] B. Yenikaya and A. Sezginer, "Model-based assist feature generation," in *Proc. SPIE*, Mar. 2007, vol. 6521, p. 652102.

[8] S. Suh, S. Lee, K.-Y. Back, S. Lee, Y. Kim, S. Kim, and Y.-J. Chun, "Three-dimensional mask effect approximate modeling for sub-50-nm node device OPC," in *Proc. SPIE*, Mar. 2007, vol. 6521, p. 652103.

[9] Y. C. Pati and T. Kailath, "Phase-shifting masks for microlithography: Automated design and mask requirements," *J. Opt. Soc. Amer. A*, vol. 11, pp. 2438–2452, Sep. 1994.

[10] Y. Pati, A. Ghazanfarian, and R. Pease, "Exploiting structure in fast aerial image computation forintegrated circuit patterns," *IEEE Trans. Semicond. Manuf.*, vol. 10, no. 1, pp. 62–74, Feb. 1997.

[11] Y. Granik, N. B. Cobb, and T. Do, "Universal process modeling with VTRE for OPC," in *Proc. SPIE*, Jul. 2002, vol. 4691, pp. 377–394.

[12] The International Technology Roadmap for Semiconductors ITRS, 2007.

[13] Y. Zhang, M. Feng, and H.-Y. Liu, "A focus exposure matrix model for full chip lithography manufacturability check and optical proximity correction," in *Proc. SPIE*, Jun. 2006, vol. 6283, p. 62830W.

[14] M. D. Smith and C. A. Mack, "Methods for benchmarking photolithography simulators," in *Proc. SPIE*, Jun. 2003, vol. 5040, pp. 57–68.

[15] R. L. Gordon, "Exact computation of scalar 2D aerial imagery," in *Proc. SPIE*, Jul. 2002, vol. 4692, pp. 517–528.

[16] P. Yu, ELIAS [Online]. Available: http://www.cerc.utexas.edu/utda/download/download.html

[17] H. H. Hopkins, "On the diffraction theory of optical images," in *Proc. Roy. Soc. Lond., Ser. A, Mathematical and Physical Sci.*, May 1953, vol. 217, pp. 408–432.

[18] M. Born and E. Wolf, *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*, 7 ed. Cambridge, U.K.: Cambridge Univ. Press, 1999.

[19] P. Yu and D. Z. Pan, "A novel intensity based optical proximity correction algorithm with speedup in lithography simulation," in *Proc. Int. Conf. Computer Aided Design*, 2007, pp. 854–859.

[20] P. Yu, W. Qiu, and D. Z. Pan, "Fast lithography image simulation by exploiting symmetries in lithography systems," *IEEE Trans. Semicond. Manuf.*, vol. 21, no. 4, pp. 638–645, Nov. 2008.

[21] A. K.-K. Wong, *Optical Imaging in Projection Microlithography, ser. SPIE Tutorial Texts in Optical Engineering*. Bellingham, WA: SPIE, 2005, vol. TT66.

[22] P. Yu, D. Z. Pan, and C. A. Mack, "Fast lithography simulation under focus variations for OPC and layout optimizations," in *Proc. SPIE*, Apr. 2006, vol. 6156, pp. 397–406.

[23] P. Yu, S. X. Shi, and D. Z. Pan, "Process variation aware OPC with variational lithography modeling," in *Proc. Design Automation Conf.*, 2006, pp. 785–790.

[24] P. Yu, S. X. Shi, and D. Z. Pan, "True process variation aware optical proximity correction with variational lithography modeling and model calibration," *J. Micro/Nanolithography, MEMS and MOEMS*, vol. 6, no. 3, p. 031004, Jul.–Sep. 2007.

[25] P. Flanner , III, Two-Ddimensional Optical Imaging for Photolithography Simulation EECS Dept., Univ. of California, Berkeley, 1986, Tech. Rep. UCB/ERL M86/57.

[26] N. B. Cobb, "Fast Optical and Process Proximity Correction Algorithms for Integrated Circuit Manufacturing," Ph.D. dissertation, Univ. of California, Berkeley, 1998.

[27] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, 2 ed. New York: Academic, 1984, ch. 2.

[28] J. van der Gracht, "Simulation of partially coherent imaging by outerproduct expansion," *Appl. Opt.*, vol. 33, no. 17, pp. 3725–3731, Jun. 1994.

[29] R. Köhle, "Fast TCC algorithm for the model building of high NA lithography simulation," in *Proc. SPIE*, May 2004, vol. 5754, pp. 918–929.

[30] W. M. Pieper, "Recursive multidimensional integration," *Int. J. Numer. Methods Eng.*, vol. 40, no. 10, pp. 1923–1935, 1997.

Peng Yu received the B.S. degree in physics from Peking University, Beijing, China, in 2002 and the MS degree in physics from the University of California San Diego in 2004. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Texas at Austin. He has interned in Synopsys, IBM, and Cadence.

He is interest Design For Manufacturing (DFM), OPC algorithms and lithography modeling. He is also interested in computational biology and biology problems in general. He has several publications in DAC, ICCAD, SPIE Microlithography and IEEE Transactions. He has reviewed papers for the IEEE Transactions on CAD, IEEE Transactions on VLSI Systems and various conferences. He has two patents pending. He is a student member of SPIE.

Mr. Yu has received DAC Young Student Support Program Award, IBM PhD Fellowship nomination, BACUS Photomask Scholarship from SPIE, the University of Texas Graduate School Continuing Fellowship, BioBricks Foundation SB4.0 Travel Award from Synthetic Biology 4.0, and Inventor Recognition Award from Semiconductor Research Corporation (SRC). He is also a member of Phi Kappa Phi.

David Z. Pan (S'97–M'00–SM'06) received the Ph.D. degree in computer science from University of California at Los Angeles (UCLA) in 2000.

From 2000 to 2003, he was a Research Staff Member at IBM T. J. Watson Research Center, Yorktown Heights, NY. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, the University of Texas at Austin. He has published over 100 technical papers and is the holder of six U.S. patents. His research interests include nanometer physical design, design for manufacturing, low-power vertical integration design and technology, and design/automation for emerging technologies. He has served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON VERY LARGE SCALE (VLSI) SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II, and *IEEE CAS Society Newsletter* (since 2007). He is also a Guest Editor of TCAD special section on the International Symposium on Physical Design in 2007 and 2008. He is an elected officer in the IEEE CANDE Committee (Workshop Chair in 2007, Secretary in 2008, and Chair in 2009) and a member of the ACM/SIGDA Technical Committee on Physical Design. He is in the Design Technology Working Group of International Technology Roadmap for Semiconductor. He has served in the Technical Program Committees of major VLSI/CAD conferences, including ASPDAC (Topic Chair), DAC, DATE, ICCAD, ISPD (Program Chair), ISQED (Topic Chair), ISCAS (CAD Track Chair), SLIP, GLSVLSI, ACISC (Program Co-chair), ICICDT, and VLSI-DAT. He is the General Chair of ISPD 2008 and Steering Committee Chair of ISPD 2009. He is a member of the Technical Advisory Board of Pyxis Technology Inc.

Dr. Pan has received a number of awards for his research contributions and professional services, including the ACM/SIGDA Outstanding New Faculty Award (2005), NSF CAREER Award (2007), SRC Inventor Recognition Award (2000 and 2008), IBM Faculty Award (2004–2006), IBM Research Bravo Award (2003), SRC Techcon Best Paper in Session Award (1998 and 2007), Dimitris Chorafas Foundation Research Award (2000), ISPD Routing Contest Awards (2007), eASIC Placement Contest Grand Prize (2009), several Best Paper Award Nominations at DAC/ICCAD/ASPDAC, and ACM Recognition of Service Award (2007 and 2008). He is a Cadence Distinguished Speaker in 2007 and an IEEE CAS Society Distinguished Lecturer for 2008–2009.