An Efficient Shift Invariant Rasterization Algorithm for All-Angle Mask Patterns in ILT

Yixiao Ding
Iowa State University
Ames, IA 50010
yxding@iastate.edu

Chris Chu
Iowa State University
Ames, IA 50010
cnchu@iastate.edu

Xin Zhou
Synopsys Inc
Mountain View, CA 94043
Xin.Zhou@synopsys.com

ABSTRACT

Lithography simulation is an essential technique to guide the design of inverse lithography technology (ILT) masks. To reduce the complexity in modern lithography simulation, a widely used approach is to first rasterize the ILT masks before they are inputted to the simulation tools. Currently, there is no high performance technology to handle the rasterization of all-angle polygons, which are very common in modern ILT masks. Traditional rasterization technology is very expensive in term of runtime and memory usage. In this paper, we propose an efficient rasterization algorithm for all-angle polygons based on a pre-computed lookup table (LUT). We expect that the convolution for a majority of large pixels can be performed in a single lookup table query, which decreases the overall runtime dramatically. The experimental results demonstrate that our proposed algorithm can speed up rasterization process by almost 500× while maintaining small variations in CD. Meanwhile, the time for pre-computing the lookup table and the size of it can be kept within very reasonable limits.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Design, Performance

Keywords

Inverse Lithography Technology (ILT), rasterization, Look-up table (LUT)

1. INTRODUCTION

Rasterization of polygons, also known as scan conversion, is a fundamental technique of geometric data processing widely used in the electronic design automation (EDA) industry in particular, and many other industries in general. A set of polygons in EDA (for example, the shapes defining the physical design of an integrated circuit) is usually represented in GDSII or OASIS format as arrays of vertices, and the rasterization process seeks to represent them by grayscale pixels on a grid.

One important application of polygon rasterization is in the simulation of projection lithography [1]. The image at the wafer level is the foundation of any lithography process modelling at the core of optical proximity correction (OPC). The manufacturing of IC chips using optical lithography involves projecting COG (Chrome on Glass) circuit masks onto coated silicon wafters. The optics that performs the projection inevitably distorts the shapes on reticle, as the imaging process cuts off all the high-frequency content limited by the numerical aperture (NA) of the lens system. Other factors, including chemical effects, cause additional distortions in the final wafer shapes. In order to improve the fidelity of the figures at the wafer level compared to the original intent of circuit designer, we have to apply OPC to the figures at the mask level.

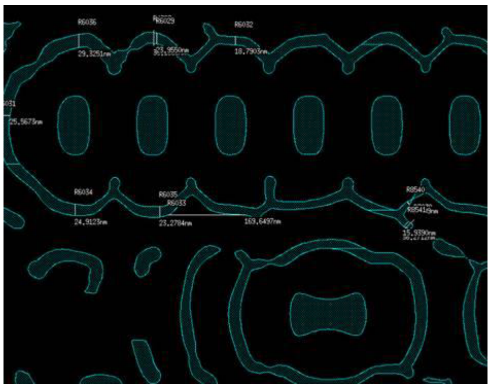


Figure 1: Example of ILT mask.

In the past, mask shapes with OPC consisted mostly of Manhattan polygons, i.e., polygons made up of horizontal and vertical edges (no arbitrary angles). As chip makers push below 20nm node, masks using an aggressive form of OPC called inverse lithography technology (ILT) are becoming more popular. Fig. 1 is an example of what an ILT mask could look like. Note that the shapes contain many curved features, and almost none of the edges are aligned to a multiple of 45 degrees. In principle, the lithography simulation can take any polygon figures as input directly. But when the figures are getting more numerous and more complex, as in the case of ILT masks, taking rasterized mask as input becomes more efficient than the direct method [2].

In order to preserve the fidelity of the original mask polygons, the size of fill pixels in the rasterized representation should be very small (say $1nm \times 1nm$). In other words, the ideal rasterization procedure should be undertaken on a grid of very small fill pixels. On the other hand, lithography simulation takes the rasterized image as input to generate wafer image. In order to keep the input data volume and runtime reasonable for lithography simulation, we would like to use larger(i.e., fewer) pixels to present the rasterized image. In practice, a suitable pixel size for lithography simulation is on the order of 10nm, and the size of image could be around 1024 × 1024 pixels.

The challenge of using large pixels in rasterization is that the result should preserve shift-invariance, which means that changes in the position of input polygons relative to the pixel sampling grid should cause very minimal variations in the corresponding output data. This property is critical in chip manufacturing. For example, in the core of a memory chip, an arrayed pattern of bitcells repeats itself millions or even billions of times, and the wafer image for each repeated cell must be as consistent as possible. If the mask representation shows any significant asymmetry or distortion, the resulting distortions on the wafer will likely be amplified due to the mask error enhancement factor (MEEF) effect at advanced nodes. A common approach for preserving shift-invariance is to preprocess the input mask patterns with an anti-aliasing filter before rasterization in large pixel level. Anti-aliasing removes the frequencies in the input polygons that are too high to be accurately captured at the sampling rate of rasterization, so that they do not appear in the samples improperly at a lower frequency (a phenomenon called aliasing). The software challenge is how to perform filtering and rasterization on a large amount of all-angle polygon data very quickly, while preserving accuracy of lithography simulation.

Traditional EDA rasterization tools deal with large volume of data of mostly Manhattan geometry, and very small percentage of 45-degree-angled polygons. There are established techniques to handle rectangular figures efficiently and accurately, and they have been extended to handle figures with 45/135 degree edges, and even figures with a limited set of predefined angles (e.g., 22.5 degrees). [3] is the state-of-the-art of Manhattan rasterization by look-up tables. For non-Manhattan patterns, [4] explains an area-coverage technique and a significant refinement by cone-filter anti-aliasing, which are not based on look-up tables. Unfortunately, there is no published technique that handles truly all-angle figures with a known performance on par with the best rectangle-processing algorithms. To the best of our knowledge, there is also no mature technology to handle all-angle polygon rasterization in industry.

In this paper, we propose to solve the problem of all-angle polygon rasterization with a lookup table-based approach. In this approach the convolution for a majority of the large pixels can be done in a single lookup table query. The experimental results demonstrate that our proposed algorithm can speed up rasterization by several orders of magnitude over conventional methods. Meanwhile, the time for pre-computing the lookup table and the memory for storing it can be kept within reasonable limits.

The remainder of the paper is organized as follows. In Section 2, we present a description of the problem and overview of our proposed algorithm. In Section 3, we describe our proposed algorithm in detail. In Section 4, we show our experimental results. Finally, Section 5 is the conclusion of this paper.

2. PROBLEM DESCRIPTION

2.1 Overview of traditional rasterization approach

Rasterization is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels) for output. Given mask patterns which have been applied OPC, we want to represent them on a pixel grid. In order to preserve the fidelity of the original patterns during the rasterization process, we ideally should use a grid of very small fill pixels. We denote such small fill pixels as small pixels. Fig. 2(a) shows an all-angle mask pattern (triangle) and a small pixel grid. Then, a scan conversion algorithm [5] is applied upon the mask patterns to obtain a bitmap on the small pixel grid. It is shown in Fig. 2(b). Even though we can take this bitmap directly as the input to lithography simulation, it is enormous and undesirable. In order to preserve shift invariance, the bitmap should goes through a convolution process under a properly-designed anti-alising filter. Since the convolution process is undertaken on the small pixel level, the output image is represented in grayscale on a grid of small pixel. As shown in the Fig. 2(c), the grayscale value for each small pixel is represented by its color lightness. To reduce the computation load of lithography simulation, we would like to increase the size of pixel for the output image while keeping certain degree of resolution. We denote pixels with increased size for the output image as large pixels. The grayscale value of large pixel is computed by averaging out the grayscale values of all small pixels contained inside. Fig. 2(d) shows the grayscale result based on large pixels. This grayscale image is a shift-invariance preserving representation of the original mask in Fig. 2(a) and is the input to lithography simulation tools.

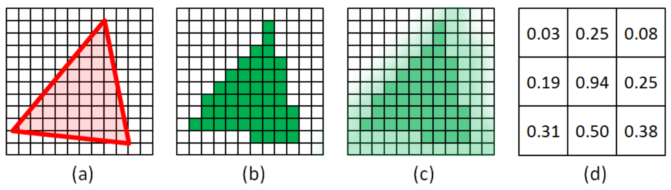


Figure 2: Traditional rasterization approach. (a) Pattern and small pixel grid. (b) Bitmap generation by scan conversion. (c) Pattern after convolution. (d) Grayscale results based on large pixels.

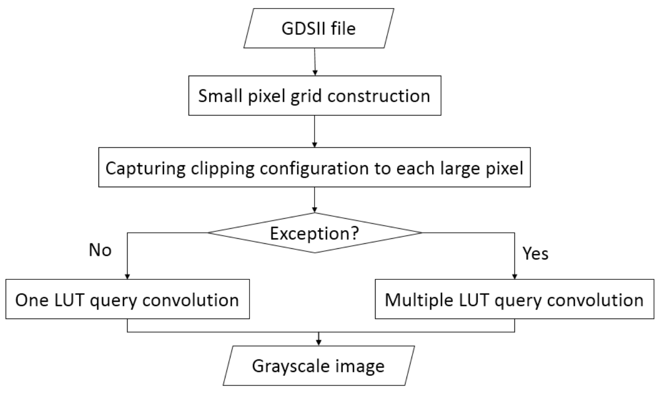
There is a problem associated with the traditional rasterization approach. In order to preserve the fidelity of input mask patterns, the convolution process is undertaken on the small pixel level. Besides, the filter used in the convolution process can be large for industrial applications. Even though the traditional rasterization approach is ideal, it is expensive in term of runtime and memory usage. In this paper, we solve this problem by applying the idea of lookup table. By taking advantage of a pre-computed lookup table, the whole filtering and rasterization process can be undertaken on the large pixel level without loss of accuracy. Meanwhile, the patterns within one large pixel can be handled with just one lookup table query. This dramatically speeds up the rasterization process.

2.2 Problem formulation

Given a set of all-angle polygon patterns represented by arrays of vertices and the size of large pixels according to the requirement of the lithography simulation, the problem is to convert the polygon patterns into a grayscale image at the large pixel level. The result should preserve shift-invariance, which means translation of input data causes very small difference in output data. The objective is to process large amount of these kinds of data very quickly but still accurately.

3. ALGORITHM

3.1 Algorithm overview



GDSII file
Small pixel grid construction
Capturing clipping configuration to each large pixel
Exception?
No
Yes
One LUT query convolution
Multiple LUT query convolution
Grayscale image

Figure 3: The overall algorithm flow.

The algorithm overview is shown in Fig. 3. The program takes a GDSII file as input. Firstly, a small pixel grid is constructed for input polygon patterns on a specific layer. Meanwhile, assume the size of large pixels is a multiple of the size of small pixels. Then, clipping configurations of input polygon patterns to each large pixel are captured. Based on clipping configuration, each large pixel is identified which categories it belongs to:

non-exception or exception. Non-exception refers to the large pixel which is only cut through by at most one polygon edge and the rest of large pixels are exception. The pre-computed and stored grayscale results in the lookup table can be taken advantage for convolution process. If it is a non-exception large pixel, only one lookup table query is required for convolution process. If it is an exception large pixel, multiple lookup table queries are required. A heuristic method is applied here in order to speed up this process. After completion of convolution for all large pixels, the program outputs a grayscale image.

3.2 Definitions of small and large pixel

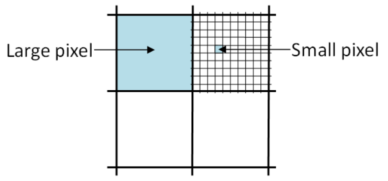


Figure 4: Small pixel and large pixel.

In order to preserve fidelity of mask patterns, the fill pixels in rasterization process should be very small. We define such a small fill pixel as a *small pixel*. In practise, $1nm \times 1nm$ small pixels are small enough to capturing fine fragments of mask patterns. However, if we directly take the grayscale image which is represented by small pixels to the lithography simulation, the input data volume is prohibitive. Thus, we would like to use increased size of pixels to represent the grayscale image. We define the pixel with increased size as a *large pixel*. In practice, $10nm \times 10nm$ large pixel is a appropriate choice such that we can keep a reasonable data volume while having enough resolution for grayscale image. The Fig. 4 shows the small pixel with size of $1nm \times 1nm$ and the large pixel with size of $10nm \times 10nm$.

3.3 Key observation

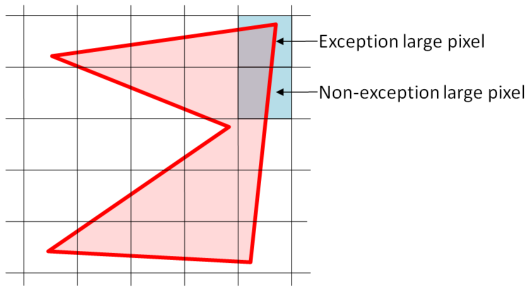


Figure 5: Non-exception large pixel and exception large pixel.

With appropriately chosen size of large pixel, we have a key observation for a given polygon under the large pixel grid: a majority of large pixels are only cut through by zero or one polygon edge. We define those large pixels as *non-exception large pixels*. For the rest of large pixels, we define them as *exception large pixels*. Thus, by acquiring clipping configuration of input polygon patterns to each large pixel, we can identify whether it belongs to non-exception large pixel or exception large pixel. As shown in Fig. 5, the top-right large pixel is an exception large pixel which containing a polygon corner, and the large pixel below it is a non-exception large pixel which is only cut through by a polygon edge. In general case, unless the given set of polygons have many extremely thin or tiny parts, the number of exception large pixels is O(n) where n is the total number of vertices for given set of polygons.

3.4 LUT approach for non-exception large pixels

For each large pixel with certain clipping configuration, we apply the traditional rasterization approach. Fig. 6 illustrates the whole process. Given a large pixel with clipping configuration shown in Fig. 6(a), its bitmap will be firstly computed based on small pixel grid which is shown in Fig. 6(b). Then, the bitmap will go through convolution process under a pre-design filter. The result is shown in Fig. 6(c). After that, grayscale value of each small pixel projects back to the corresponding large pixel. Finally, grayscale results are generated based on large pixels which is shown in Fig. 6(d). Note that the convolution process is conducted on the small pixel level and the size of filter can be very large for industrial applications. Besides, some large pixels with same clipping configuration will probably appear multiply times, thus the computation of grayscale result for each time are repeated. As a result, even though this procedure is ideal, it is relatively expensive in term of runtime and memory usage.

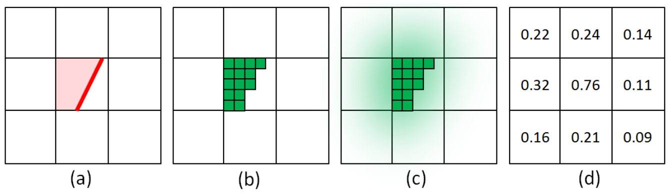


Figure 6: Ideal procedure for large pixel rasterization. (a) Given a large pixel with certain clipping configuration. (b) Bitmap generation for the large pixel based on small pixel grid. (c) Convolution process based on small pixel grid. (d) Grayscale results based on large pixel grid after projection.

However, if we pre-compute and store the grayscale results in the look-up table for all different clipping configurations of non-exception large pixels, time spent on computing grayscale results is saved. The total runtime will decrease dramatically since non-exception large pixels account for a majority of large pixels. Fortunately, the number of all different clipping configurations for non-exception large pixel is limited with given sizes of large and small pixel. Suppose the size of large pixel is $n \times n$ small pixel. If the non-exception large pixel is only cut through by one polygon edge, two out of four boundary edges of square pixel will intersect with the polygon edge. For each boundary edge, there are n different locations for a intersection point. Provided that either side of polygon edge could be within the polygon interior, the total number of clipping configurations for non-exception large pixel is $12n^2$. If considering the symmetric property of all the clipping configuration, the total number can be further reduced to $6n^2$. For industrial applications, we usually have small pixel with size of $1nm \times 1nm$ and large pixel with size of $10nm \times 10nm$. Consequently, the number of entries used to store grayscale results for non-exception large pixels in the lookup table is 600.

3.5 Technique to handle exception large pixels

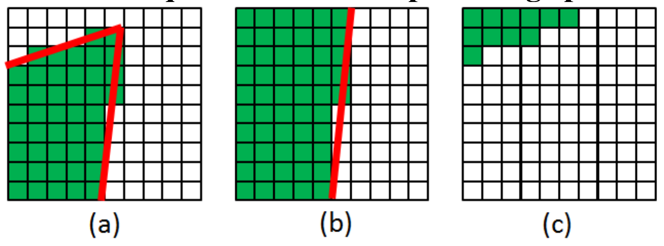


Figure 7: Technique to handle exception large pixels. (a) Given an exception large pixel and compute its bitmap B_{exp}. (b) Find the corresponding non-exception large pixel and compute its bitmap $B_{non-exp}$. (c) Compute the different between two bitmaps and denote as B_{diff}.

For each non-exception large pixel, we can easily obtain its grayscale results from lookup table. Next, we propose our technique to deal with exception large pixels. As shown in Fig. 5, exception large pixels could have various clipping configurations, such as containing a polygon corner or cut through by two polygon edges. Even though exception large pixels only account for a small percentage, we are reluctant to directly apply traditional rasterization approach due to the long runtime. Meanwhile, we want to take full advantage of the lookup table. The idea of our technique is trying to use corresponding non-exception large pixel in the lookup table together with small pixels to reconfigure exception large pixel. In this case, we also pre-compute and store the grayscale results in the lookup table for each single small pixel within the large pixel. This will only increase the number of entries of lookup table by n^2. Fig. 7 illustrates this idea. Given an exception large pixel, we firstly compute its bitmap B_{exp} which is shown in Fig. 7(a). Then, we try to find a non-exception large pixel in the lookup table whose bitmap has highest similarity with that of the exception large pixel. We denote the bitmap of the corresponding non-exception large pixel as $B_{non-exp}$. It is shown in Fig. 7(b). Given B_{exp} and $B_{non-exp}$, we compute the difference between two bitmaps which is denoted as B_{diff}. It is shown in Fig 7(c). In order to obtain accurate grayscale results for the exception large pixel, we need to add/subtract grayscale results of B_{diff} to/from grayscale results of $B_{non-exp}$. The grayscale results of $B_{non-exp}$ can be easily acquired by one lookup table query. The grayscale results of B_{diff} also can be acquired by adding/subtracting grayscale results of its non-zero entries. Each non-zero entry in B_{diff} corresponds to one lookup table query of single small pixel. Consequently, we can compute grayscale results of exception large pixel by multiple lookup table queries.

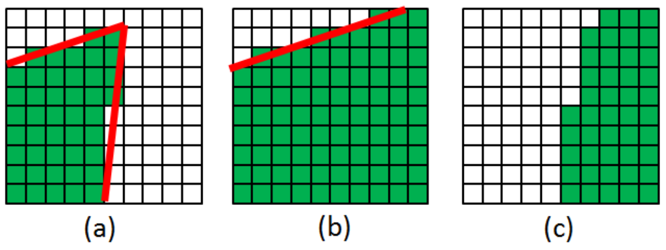


Figure 8: An inappropriate choice of non-exception large pixel will cause unnecessary lookup table queries and computational time. (a) Given an exception large pixel. (b) An inappropriate choice of corresponding non-exception large pixel. (c) B_{diff} with relatively more number of non-zero entries.

The number of non-zero entries in B_{diff} determines the number of lookup table queries in order to accurately compute grayscale results of the exception large pixel. Meanwhile, the number of non-zero entries in B_{diff} is greatly affected by the choice of corresponding non-exception large pixel. Fig. 8 illustrates the importance of good choice of corresponding non-exception large pixel. Given an exception large pixel in Fig. 8(a), if we choose the corresponding non-exception large pixel in Fig. 8(b), the B_{diff} in Fig. 8(c) has relatively more number of non-zero entries than that in Fig. 7(c). Each one of non-zero entries in B_{diff} requires one lookup table query and an operation of adding/subtracting grayscale results to/from grayscale results of corresponding non-exception large pixel. We can see an inappropriate choice of non-exception large pixel will cause unnecessary lookup table queries and computational time. Therefore, the current problem is how to find corresponding non-exception large pixel for given exception large pixel which results in as few non-zero entires in B_{diff} as possible. We propose a smart heuristic for this problem.

For each clipping configuration of non-exception large pixel stored in the lookup table, we quadrisect its bitmap and record number of non-zero entries for each quadrant. We denote the

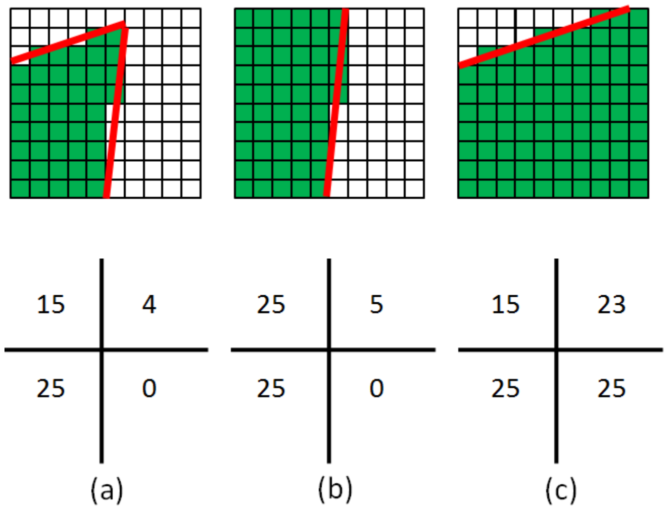


Figure 9: Fast heuristic to find corresponding non-exception large pixel in lookup table. (a) Given an exception large pixel. (b) Non-exception large pixel with diversity degree of $\sqrt{101}$. (c) Non-exception large pixel with diversity degree of $\sqrt{986}$.

four records as N_I, N_{II}, N_{III}, and N_{IV}. Thus, four records are associated with each entry in the lookup table. Given an exception large pixel, we also quadrisect its bitmap and calculate N'_I, N'_{II}, N'_{III}, and N'_{IV} for each quadrant. We define the *diversity degree* between the given exception large pixel and non-exception large pixel stored in the lookup table as $\sum_{i=1}^{4} |N_{ß} - N'_{ß}|$. The corresponding non-exception large pixel can be obtained by finding a entry in the lookup table whose has minimum diversity degree. Fig. 9 gives an example. Consider an exception large pixel in Fig. 9(a), the non-exception large pixel in Fig. 9(b) which has diversity degree of 11 is a better choice than the non-exception large pixel in Fig. 9(c) which has diversity degree of 44.

3.6 Cost of lookup table

As we have mentioned above, the number of entries for non-exception large pixel in lookup table is $6n^2$ with additional n^2 number of entry for single small pixels. We also pre-compute and store grayscale results of a large pixel in the lookup table for the case which is entirely within polygon interior. In sum, the total number of lookup table entries is $7n^2 + 1$. The size of lookup table can be kept within a reasonable manner and the time spent on building lookup table is due before start of program. Once the program starts, only one lookup table query is needed for each non-exception large pixel and each large pixel which is entirely within polygon interior. For exception large pixel, one lookup table query of corresponding non-exception large pixel and several lookup table queries of single small pixel are needed. In practice, the number of lookup table queries of single small pixel is small due to the proposed heuristic.

3.7 Technique to obtain clipping configuration

In order to correctly identify each large pixel as non-exception large pixel or exception large pixel, we should firstly obtain clipping configuration of each large pixel with given set of polygons. This is actually the problem of polygon clipping where large pixel is the clipping window and given set of polygons are the clipping polygons. We adopt WeileråĂŞAtherton clipping algorithm [4, 6] in our implementation to solve this problem. The Weiler-Atherton algorithm is capable of clipping a concave polygon with interior holes to the boundaries of another concave polygon, also with interior holes. The polygon to be clipped is called the subject polygon (SP) and the clipping region is called the clip polygon (CP). The algorithm has

Table 1: Performance of our proposed algorithm

Benchmarks	Percentage of Non-ex (%)	Runtime of clipping (s)	Runtime of convolution (s)	Runtime of algorithm (s)	Handling exception large pixels		
					#Ex	#LUT queries	Runtime (s)
gdsii1	99.55	0.12	2.26	2.38	48	5.77	0.22
gdsii2	98.74	26.31	171.81	198.12	3400	5.39	6.99
gdsii3	95.09	25.63	189.22	214.85	7396	5.30	15.02
gdsii4	97.49	25.75	213.26	239.01	3796	16.62	18.40
gdsii5	97.50	27.89	232.05	259.94	3793	15.77	19.23
gdsii6	97.54	31.97	239.40	271.37	3789	13.44	16.90
gdsii7	97.61	44.24	270.46	314.70	3798	11.59	14.20
gdsii8	97.39	46.41	278.22	324.63	4154	10.47	13.81
gdsii9	97.49	25.44	210.9	236.34	3799	17.02	18.49
gdsii10	95.09	28.24	222.01	250.25	7395	5.29	17.47
Average	97.35	28.20	202.96	231.16	4136.8	10.67	14.07

four major steps: (1)Determine the intersections of the subject and clip polygons; (2)Process non-intersecting polygon borders; (3)Create two intersection vertex lists; (4)Perform the actual clipping. Due to the page limit, we do not explain the detail of this algorithm.

4. EXPERIMENTAL RESULTS

We implement our algorithm in the C/C++ programming language. We run all experiments on a machine with an Intel Core i5 2.66GHz CPU and 4GB of memory. All ten benchmarks are provided by Synopsys in the format of GDSII. Except for the first benchmark, the dimensions of all other nine benchmarks are approximately $10\mu m \times 10\mu m$. The size of small pixel is defined as $1nm \times 1nm$ and the size of large pixel is defined as $10nm \times 10nm$. The size of 2D filter that we are using during the convolution process is $500nm \times 500nm$.

In Section 4.1, the performance of our proposed algorithm is demonstrated. In Section 4.2, our proposed algorithm is compared with the other two rasterization approaches. One is small pixel approach, the other is area coverage approach. In Section 4.3, the property of shift invariance of our proposed approach is evaluated.

4.1 Performance of our proposed algorithm

Before the start of the program, we firstly build up the lookup table. The time spent on building lookup table is 607.1s which is reasonable. Table I shows the performance of our proposed algorithm. Column 2 shows percentages of non-exception large pixel among all large pixels in the output image. For all the benchmarks, the percentages are more than 95% and on average 97.35% which confirms the key observation in our algorithm. The runtime of our proposed algorithm consists of two parts: one is capturing clipping configuration of input polygon patterns to each large pixel and the other is convolution process for all large pixels in the output grayscale image. Column 3 shows the runtime of the first part and column 4 shows the runtime of the second part. Column 5 shows the total runtime of our proposed algorithm. We can see that the convolution process takes most of algorithm runtime. Traditional 2D convolution operation requires 4 levels of nested loops to go through every pixel of the image and every element of the filter matrix. It is usually slow unless one uses small filter. Bigger filters can be applied with much faster runtime, if one uses the Fast Fourier Transform (FFT). Thus, the ideal approach is firstly to use FFT to handle large pixels which are not cut by polygon boundary. Then, using LUT to take care of large pixels which are cut by polygon boundary and refine in spatial domain the image generated by FFT. However, we do not implement FFT since it is not the focus of this paper. For each exception large pixel in the output grayscale image, the convolution process requires multiple lookup table queries. Columns 6, 7 and 8 demonstrate experimental results of handling exception large pixels. Column 6 shows the number of exception large pixels in output image for each benchmark. Due to the proposed heuristics, the number of lookup table queries for each exception large pixel stays within a reasonable number, which is on average 10.67 as shown in Column 7. Thus, handling all the exception large pixels does not take too long, which is on average 14.07s.

4.2 Compared with other rasterization approaches

In this subsection, we compare our approach with the other two rasterization approaches, which do not apply the idea of lookup table. One is small pixel approach, the other is area coverage approach. For the small pixel approach, both the scan conversion and convolution process are undertaken in the small pixel level. The grayscale result of each large pixel in the output image is obtained by averaging out all the small pixels contained inside. For this approach, the runtime is expected to be extremely high. Thus, we modified each gdsii benchmark in Table 1 and generated smaller scale testcase to perform experiments for this part. For the area coverage approach, the grayscale value for each large pixel is directly determined by the percentage of area covered by the input polygons. Thus, the runtime is expected to be very fast. As shown in Table 2, compared with small pixel approach, our algorithm on average has almost 500× speedup. Besides, our proposed algorithm does not do any approximation for the whole process. Thus, the grayscale value of each large pixel in the output image is exactly the same as that computed by the traditional rasterization approach. The area coverage approach has even faster runtime However, since no anti-alising filter is applied for area coverage approach. The shift invariance property can be damaged. We will show this in the next subsection.

Table 2: Runtime comparision

Testcases	Small pixel (s)	Area cover (s)	Ours (s)
testcase1	240.19	0.11	0.22
testcase2	614.29	0.35	0.60
testcase3	244.04	0.30	0.52
testcase4	239.75	0.32	0.51
testcase5	307.76	0.34	0.55
testcase6	307.44	0.43	0.65
testcase7	321.06	0.62	0.90
testcase8	283.58	0.73	1.14
testcase9	255.40	0.30	0.53
testcase10	238.30	0.26	0.51
Average	305.18	0.37	0.51
Normalized	497.84	0.73	1

4.3 Evaluation of shift-invariance property

As we have already mentioned before, shift-invariance is a critical factor which should be considered during the rasterization process. We evaluate the quality of shift-invariance by analyzing variation of critical dimension (CD) error due to rotation

Table 3: CD error variation caused by rotation

		Small pixel approach		Area coverage approach		Ours	
Benchmarks	Rotation (°)	Ave. wcde (nm)	Ave. hcde (nm)	Ave. wcde (nm)	Ave. hcde (nm)	Ave. wcde (nm)	Ave. hcde (nm)
Benchmark1	0	0.191	0.6068	6.6339	1.6699	0.0536	0.5179
Benchmark2	1	0.4606	0.4456	6.8844	2.1603	0.4308	1.5508
Benchmark3	3	1.3556	2.5336	7.5850	2.9584	1.3039	3.6488
Benchmark4	10	2.1633	2.7312	7.5064	2.8858	1.7591	2.9286
Benchmark5	30	2.7658	0.4925	8.5512	2.6745	2.3684	1.8378
Benchmark6	45	5.0739	2.0091	11.6156	3.9560	5.9772	2.0257
Benchmark7	89	0.9059	0.6035	7.3052	1.8888	0.8238	1.7438
Benchmark8	90	0.2106	0.5667	6.6389	1.6699	0.0706	0.5705
SD		2.2645	1.5576	7.9904	2.5907	2.4239	2.1044
Normalized SD		1	1	3.53	1.66	1.07	1.35

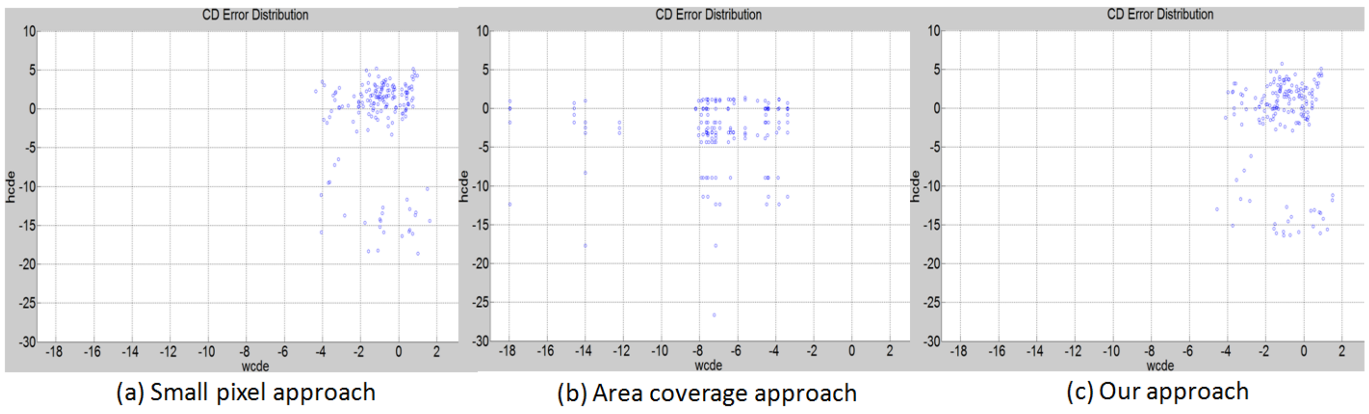


Figure 10: CD error distribution of the three approaches

of input polygons. In order to demonstrate our proposed algorithm has a good quality of shift-invariance, we also compare with both small pixel and area coverage approaches. For the small pixel approach, the convolution process is undertaken in the small pixel level and the output grayscale image is also based on small pixels. The CD error variation is expected to be very small when the input polygons are rotated. Thus, this approach provides the baseline CD error variation for us to evaluate the other two approaches. However, its runtime and memory usage is prohibited in practice. For the area coverage approach, the grayscale value for each large pixel is directly calculated by the percentage of area covered by the input polygons. No anti-alising filter is applied. Thus, the CD error variation is probably very large. All the eight benchmarks are provided by Synopsys and each consists of an array of aligned rectangles rotated by a different degree. The width of each rectangle is 79nm and height is 167nm. "Ave. wcde"/"Ave. hcde" is the average CD error in width/height over all the input rectangles. "SD" is the standard deviation of "Ave. wcde or "Ave. hcde" over all eight benchmarks. As shown in Table 3, compared with small pixel approach, our approach has a little bit more CD error variation when rectangles are rotated. Meanwhile, the area coverage approach has much more variation on CD error when the rotations occur. This demonstrates that our proposed algorithm has good quality of shift-invariance. Besides, Fig. 10 is a plot presenting CD errors in both width and height of each rectangle for Benchmark4 from Table 3. Each circle represents a rectangle in the benchmark. We can see that the CD error distribution of our approach is similar to that of small pixel approach, while CD error distribution of area coverage approach is really far away.

5. CONCLUSION

In this paper, we propose an efficient rasterization algorithm for all-angle polygons. The algorithm is based on a pre-computed lookup table. By taking advantage of lookup table, the whole rasterization process can be undertaken on the large pixel level without loss of any accuracy. Meanwhile, for a majority of large pixels, the patterns within them can be handled with only one lookup table query. Experimental results show that our proposed algorithm can speed up by almost 500× compared with accurate small pixel approach. Meanwhile, our proposed algorithm demonstrates a good quality of shift-invariance property. In the future work, we want to explore the application of FFT in the convolution process and further speed up the whole rasterization process.

6. REFERENCES

[1] Alfred Kwok-Kit Wong. *Optical Imaging in Projection Microlithography*, volume 66 of *SPIE Tutorial Texts*. SPIE Publications, Vancouver, Washington, 2005.

[2] Yong Liu, Dan Abrams, Linyong Pang, and Andrew Moore. Inverse lithography technology principles in practice: unintuitive patterns. In *Proc. of SPIE 5992*, 2005.

[3] Michael L. Rieger, Micheal Cranford, and John P. Stirniman. Flash-based anti-aliasing techniques for high-accuracy high-efficiency mask synthesis, Jan 2011. Patent.

[4] John F. Hughes, Andries van Dam, and et al. *Computer Graphics: Principles and Practice (3rd edition)*. Addison-Wesley Professional, Readings, Massachusetts, 2013.

[5] Marc Levoy. Introduction to computer graphics: Raterization algorithm. http://graphics.stanford.edu/courses/cs248-08/scan/scan1.html, Autumn Quarter 2008. Lecture notes.

[6] Kevin Weiler and Peter Antherton. Hidden surface removal using polygon area sorting. In *Proc. of SPIE 5992*, volume 11, pages 214–222, New York, NY, 1977.