Master Thesis

Algorithms for the Computation of Continuous Transforms of Rectilinear Polygons from IC Layouts

Author:
Robin Scheibler
robin.scheibler@epfl.ch

Supervisors:
Dr. Paul Hurley
Dr. Amina Chebira
Prof. Martin Vetterli

October 16, 2009

Abstract

In this work, we present a novel way of computing the continuous Haar, Fourier and cosine series coefficients of rectilinear polygons. We derive algorithms to compute the inner products with the continuous basis functions directly from the vertices of the polygons. We show that the overall computational complexity of those algorithms is lower than that of the traditional corresponding discrete transforms when the number of vertices is small, in addition to sparing the memory needed for a discrete image. This makes those continuous transforms particularly suitable for applications in Computational Lithography (CL) where speed and memory are critical requirements. We validate the presented algorithms through an implementation in a CL software under development at the IBM Zürich Research Laboratory and benchmark against discrete state of the art transforms on real Integrated Circuit (IC) layouts. Finally, we measure the approximation power of the Haar transform when applied to rectilinear polygons from IC layouts in order to evaluate its potential for pattern matching applications.

Acknowledgements

I would like to thank particularly my advisor, Dr. Paul Hurley, for giving me the opportunity to work on this exciting project. His constant guidance and his enlightened advice helped me a lot, especially at times where I could not see clearly where the project was going. He was always there when I needed his support.

Dr. Amina Chebira was my advisor from EPFL and kept a close look at what I was doing all throughout the project. I owe a lot to her thoughts, comments and suggestions regarding my work. I am extremely grateful for her time and effort.

Thanks also to Dr. Patrick Droz for caring so much and looking after me during my whole stay at IBM.

I also take the chance here to express the deepest gratitude to my parents who supported and encouraged me during all the time of my studies and much more.

Many thanks also to my friends who made all this time a lot of fun: Vincent, Julien, Christophe, Jean, Grégoire, Marcel, Donato, Nan, and all the others.

Contents

Chapter 1

Introduction

Since Gordon Moore stated his famous law in 1965, it has been a major driving force behind the effort to shrink transistors in integrated circuits (IC). The smallest feature of those circuits has shrunk from a few micrometers when optical lithography was created to 45 nanometers for the latest commercially available technology. Unfortunately, the light sources used in optical lithography have not seen a corresponding reduction in wavelength and current systems still use deep ultra-violet light sources (with a wavelength of 193 nanometers). Although the next generation extreme ultra-violet source is under development it seems it will not be ready for the next technology size target, namely 22 nanometers.

In optical lithography, the IC are printed by shining light through a mask onto a photosensitive wafer. Unfortunately, the optical resolution of this system is accurate only as long as the wavelength of the light source is on the order of the smallest feature of the mask. Various techniques and tricks have been worked out to mitigate the severe optical degradation due to feature size shrinking far below the source wavelength. This includes immersion lithography and Resolution Enhancement Techniques (RET), such as Optical Proximity Correction (OPC) and phase-shift masks [1]. However, these techniques proved insufficient in going below 32 nanometers. In order to break this limit, traditional RET techniques are being enhanced to take advantage of all degrees of freedom in the lithography process such as illumination amplitude, direction and phase [2]. In addition, mask constraints are being removed by the introduction of pixelated masks [3]. This has led to the introduction of new techniques that try to globally optimize the lithography process. Such techniques include Source-Mask Optimization (SMO) which tries to jointly optimize the light source and the mask in order to print a given pattern [4] or inverse mask problems [5].

1.1 Problem Statement and Motivation

The common point of all these new techniques, usually referred to as a whole as Computational Lithography (CL), is that they are very computationally intensive. That is exactly where a second problem due to ever shrinking transistors kicks in: data explosion. Thanks to Moore's law, the number of elements in an IC layout (and hence its size) roughly doubles every eighteen months. This leads to sizes over a terabyte for a single layer of a layout beyond 40nm technology, and over three terabytes for the 22nm technology. Combined to this data explosion, current CL techniques become intractable.

However, IC layouts are composed of very repetitive patterns. Exploiting this property could allow the CL techniques to run in a reasonable time. A small team at the IBM Zürich Research Laboratory (ZRL) has set out to develop fast pattern matching algorithms dedicated to IC layouts.

As part of this effort, this project investigates fast transform algorithms (Fourier, cosine, Haar) dedicated to rectilinear polygons, which are the building blocks of IC layouts. The core idea is to go from the computational geometry world where the rectilinear polygons lie to the signal processing world. The reason we want to easily move back and forth between the computational geometry and signal processing worlds is that although the polygons have a very compact description in the computational geometry domain, many problems found there are NP-hard while, on the other hand, the world of signal processing has nice properties such as fast algorithms, linear spaces and good approximation power.

So far, what has traditionally been done, for example in SMO where the Fourier transform is used, is to first create a discrete image by sampling the layout and then use a discrete transform on this image,

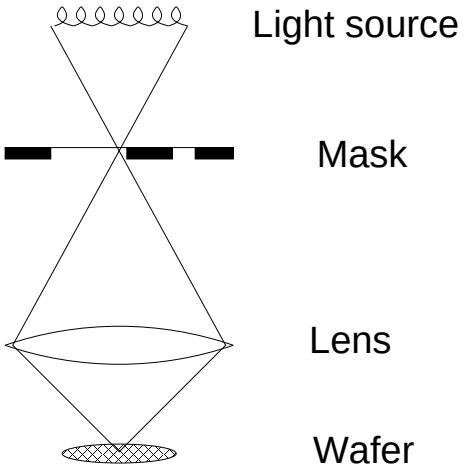


Figure 1.1: Basic principle of optical lithography.

the fast Fourier transform (FFT) in the case of the SMO.

What we propose is to take the best of both worlds. In the computational geometry world, polygons have a compact representation which defines a continuous subset of the real plane. It is therefore possible to compute continuous transforms directly on the polygons, thus performing both sampling and transform at the same time. We will show that this not only spares the creation and storage of the digital image, relaxing the amount of memory used, but is also on average faster than doing a discrete transform.

Three different transforms are investigated:

- Fourier Series. The FFT is already widely used in lithography because the diffraction of the light shining through the mask can be modelled using the Fourier Transform.
- Cosine Series. The cosine transform has the advantage over the Fourier transform of having a real output and leads to more compact approximations. It has been used sometimes as an alternative to the Fourier transform in inverse lithography [6, 7].
- Haar Series. The Haar basis is by its nature particularly suited for rectilinear polygons and thus can lead to very sparse approximations. This is a very desirable property for many applications that are targeted by the pattern matching project that is underway at IBM ZRL. Such applications include fast pattern matching within a layout, feature extraction or transform coding.

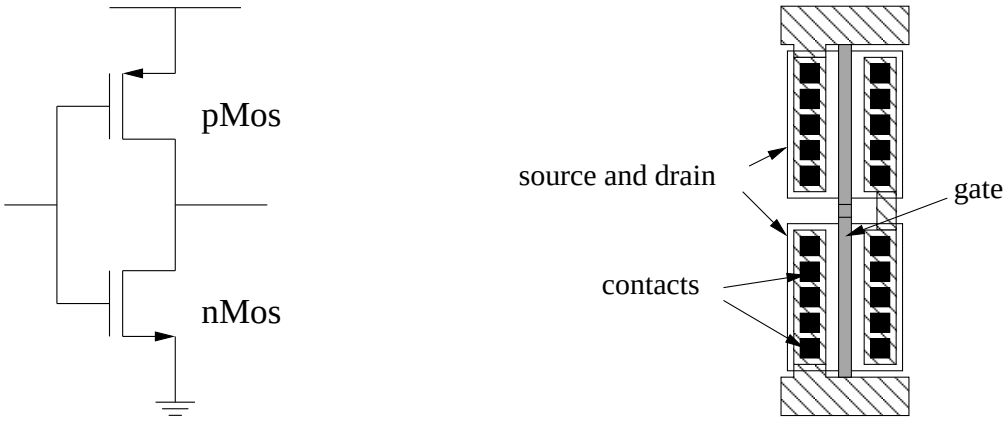
The second goal of this project is to investigate the Haar transform of rectilinear polygons. Because of the rectilinear nature of the Haar basis functions themselves, very sparse approximations can be found. The possible applications lie mainly in feature extraction and pattern matching with the particular goal of finding portions of the layout which are the same or similar in terms of shape or functionality. We will show that it is possible to find good approximations with only a few coefficients in the Haar basis.

1.2 Related Work

The basic principle of lithography is to project light through a mask which is followed by a lens that will reduce the size of the image of the mask and print it on a wafer as illustrated in Fig. 1.1. Because of the Fourier transforming properties of lenses [8], the projected image is the convolution of the mask with the following ideal filter with frequency response:

$$H(f,g) = \begin{cases} 1 & \text{if } \sqrt{f^2+g^2} \leq \frac{NA}{\lambda} \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

where f and g are the spatial frequencies, NA the numerical aperture of the lens and λ the wavelength of the source [1]. It is thus possible to estimate the image printed by computing the Fourier transform of the



(a) schematic (b) layout

Figure 1.2: The transformation of an inverter composed of two transistors into a layout of rectilinear polygons [1].

mask and only keep the spatial frequencies such that $\sqrt{f^2 + g^2} \leq \frac{NA}{\lambda}$. This makes the Fourier transform a tool of choice in lithography. Already in 1983, the FFT has been used to compute a precompensation filter to reduce the proximity effect [9]. More recently, in SMO, Rosenbluth and al. use a discrete Fourier transform to approximate the diffraction coefficients of the mask for a given source [4]. They first sample the mask with a sampling period suitable for the maximum diffraction order they need to compute and subsequently use a 2D FFT to compute the Fourier coefficients. In [10], the authors develop a road map for the efficient use of the 2D FFT in computational lithography.

In lithography, the cosine transform is not as widespread as the Fourier transform. However, being real-valued and having compacter approximation properties, it is an interesting alternative to the Fourier transform. In inverse lithography, where the mask design is considered as an inverse problem, techniques based on the 2D discrete cosine transform (DCT) have been proposed by [6, 7].

Finally, the Haar transform has not been used much in lithography. The only example we are aware of is from Haslam and al. who used it to compress the Fourier precompensation filters for electron beam lithography [11]. They used a Discrete Haar Transform on the coefficients of the discrete 2D precompensation filter and subsequently kept only the largest coefficients.

All the transforms used so far in lithography are discrete transforms. Discrete transforms are typically derived from continuous transforms and are, under some assumptions, equivalent in some sense. In most cases, continuous transforms cannot be computed in practice, while efficient algorithms for discrete transforms that can run on any modern computer are nowadays widely available. Even the so-called continuous wavelet transform is computed in practice as the projection of a discrete signal on a frame, namely the redundant counterpart of a basis [12].

However, in lithography the rectilinear polygons that form the IC layouts are actually subsets of $\mathbb{R}^2$ described by their boundaries. If the boundary of a subset of $\mathbb{R}^2$ is known, it is possible to compute any integral on this domain, using for example Green's theorem. In addition, whereas a discrete transform takes as parameters all the pixels that compose the image to be transformed, the continuous transform of a rectilinear polygon takes only the position of the vertices of the polygon. As it will be shown in this work, it is thus possible to create fast algorithms for the continuous transform of rectilinear polygons.

1.3 Integrated Circuits Layouts

In this section, we will first give an overview of what are IC layouts and how they are created. Then, the rectilinear polygons that compose those layouts are described mathematically in order to lay the path for the algorithms that will be described in the next chapter.

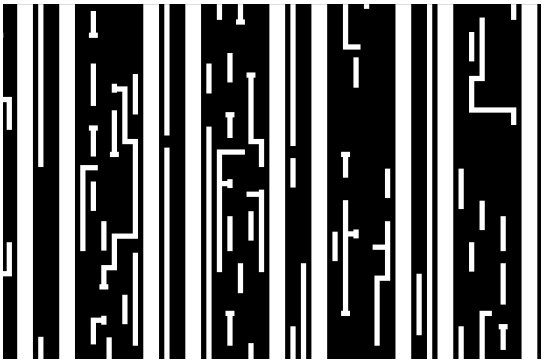


Figure 1.3: A typical piece of IC layout.

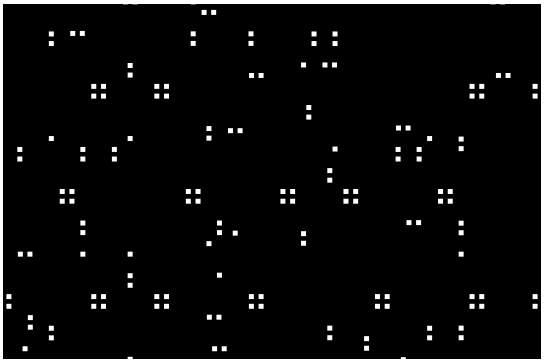


Figure 1.4: A piece of a contact layer from an IC layout.

1.3.1 Layouts

IC layouts are composed of million of rectangles and more generally rectilinear polygons. They are first created from a functional electric circuit. The elements of the circuit are transformed into specific rectilinear polygons that can be printed using an optical lithography system. Typically such a layout is composed of several layers. The transformation of a transistor into rectilinear polygons is illustrated in Fig. 1.2.

These shapes are then stored into vector graphic format such as GL/1 [13] or the more modern OASIS [14]. A typical design is split between many files of manageable size. Each file contains only a part of the design in the form of different types of disjoint rectilinear shapes such as rectangles, polygons and lines.

A layout is composed of many layers. There are basically two types of layers. The first type exemplified in Fig. 1.3 contains mostly elongated rectangles and polygons which are the actual elements making the electronic functionality (e.g. source/drain of transistors, gate, etc). As shown in Fig. 1.4, the second type of layer contains only small squares which are contacts between the different layers of the first type.

1.3.2 Rectilinear Polygons

We will now give a mathematical definition of the polygons found in IC layouts. The polygons we are dealing with are rectilinear simple lattice polygons. Rectilinear simple lattice polygons have the following properties:

- Only right angles (rectilinear);
- Edges do not intersect each other (simple);
- No holes (simple);
- The vertices are all on the integer lattice (lattice).

An example of such a polygon is shown in Fig. 1.5.

Usually, such a polygon is defined by the set of the ordered coordinates of its K vertices

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{K-1}, y_{K-1})\} \quad , \quad (x_i, y_i) \in \mathbb{Z}^2. \tag{1.2}$$

This set of vertices separates the plane in two parts, one called the inside of the polygon and the other the outside of the polygon. In addition, we need to choose an order for the sequence. We arbitrarily decided to order the vertices clockwise. Finally, to have a disjoint partition of the plane, we include in the polygons the edges that go up and from right to left while we exclude the edges that go down or from left to right as depicted in Fig. 1.5.

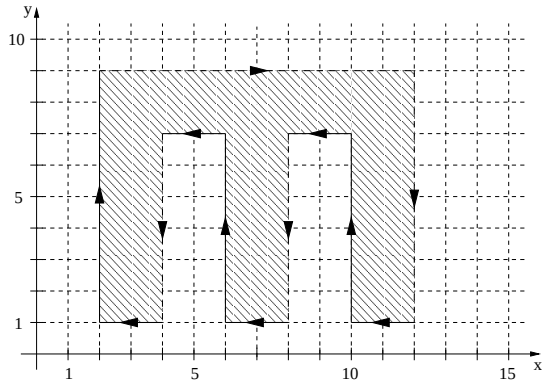


Figure 1.5: A lattice simple rectilinear polygon. The interior of the polygon is shaded. The plain edges (up and right to left) are included while the dashed edges (down and left to right) are not. The arrows indicate the ordering of the vertices.

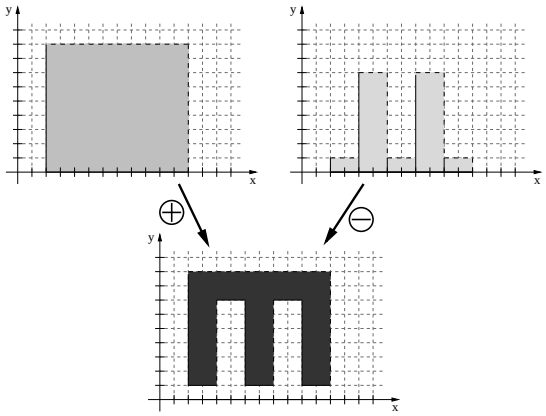


Figure 1.6: Illustration of the construction of a rectilinear polygon from disjoint rectangles. The plus and minus here illustrate, respectively, the set union and difference operators.

A rectangle is a special case of rectilinear polygon with four vertices. They are considered separately as they have properties that make them simpler to handle. They are also easier to define. For example, a rectangle R defined by its lower left and upper right vertices can be written as:

$$R_{(x_1,y_1)}^{(x_2,y_2)} = \{(x,y)|x_1 \leq x < x_2, y_1 \leq y < y_2\}.$$

As illustrated in Fig. 1.6, it is possible to express a rectilinear polygon with K vertices as unions and differences of $K/2$ rectangles:

$$\mathcal{P} = \bigcup_{\{i:x_{i+1}>x_i\}} R_{(x_i,0)}^{(x_{i+1},y_i)} - \bigcup_{\{i:x_{i+1}<x_i\}} R_{(x_{i+1},0)}^{(x_i,y_i)} \tag{1.3}$$

$$= \bigcup_{\{i:y_{i+1}<y_i\}} R_{(0,y_{i+1})}^{(x_i,y_i)} - \bigcup_{\{i:y_{i+1}>y_i\}} R_{(0,y_i)}^{(x_i,y_{i+1})} \tag{1.4}$$

where the indices are to be understood modulo K. As we will see, this definition makes it easy to take inner products over rectilinear polygons.

So far, polygons have been defined as subsets of $\mathbb{R}^2$. As such, it is not possible to directly transform them. Therefore, we will instead consider their indicator function:

$$f_{\mathcal{P}}(x,y) = \mathbb{1}_{\{(x,y)\in\mathcal{P}\}} \tag{1.5}$$

where $\mathcal{P} \subset \mathbb{R}^2$ is the polygon.

1.4 Achievements

The contributions of this thesis are:

- We found a practical way to compute the inner product of rectilinear polygons with continuous basis functions based on the vertices of the polygon.
- We developed fast algorithms to compute the continuous series coefficients for the following bases:
  - Haar basis: The standard fast orthogonal wavelet transform was found to be inefficient when directly used with the continuous inner product. We thus developed a pruned algorithm dedicated to rectilinear polygons.

  - Fourier basis: Using the continuous inner product for rectilinear polygons, we were able to map the Fourier series coefficients computations to discrete Fourier transforms with a very sparse input, namely as many non-zero inputs as the number of vertices of the polygon being transformed.
  - Cosine basis: In the same way as with the Fourier basis, we could map the cosine series coefficients computations to discrete sine transforms with only as many non-zero inputs as the number of vertices of the polygon.

- We implemented those algorithms in C++ in a CL tool that is currently under development at IBM ZRL. We added the following functionality:
  - Continuous Haar, Fourier and cosine series coefficients computations.
  - Discrete Haar, Fourier and cosine transforms for performance comparison with the continuous algorithms.
  - Sparse memory storage of Haar coefficients.
  - Output to Matlab data file format.
  - Benchmark for the transform runtime measurement.
  - Approximation error computation in the Haar basis.
- We derived the theoretical computational complexities of these algorithms which allow us to decide whether it is better to use the continuous or discrete transforms.
- We provided benchmark results of the runtime of the implemented continuous and discrete transforms on real IC layouts.
- We showed that the IC layouts have a sparse representation in the Haar basis by measuring how the approximation error decreases when we increase the number of coefficients in the approximation.

Chapter 2

Continuous Transforms of Rectilinear Polygons

In this chapter, the algorithms to compute the Fourier, cosine and Haar series coefficients of rectilinear polygons are derived. Their theoretical computational complexity is evaluated and compared to that of discrete transform algorithms. At the end of this chapter, a summary of the complexity of both the continuous and discrete algorithms is given in Table 2.2.

The primary reason the continuous transform can be faster than the discrete for rectilinear polygons is that the continuous inner product of a basis function is a direct function of the vertices of the polygon. This means that the polygon description in (1.2) is a natural sparse representation. In addition, we spare the memory needed to form and store a discrete image.

Secondly, the sampling of the polygons in order to create a discrete image can itself be considered a projection on a Dirac basis. By doing sampling followed by a discrete transform we are effectively doing two transforms. As illustrated in Fig. 2.1, using a continuous transform allows to skip the sampling operation.

Finally, in the case of the DFT, it is assumed that the sampled signal is band limited. However, this is not the case of the rectilinear polygons as they have infinite bandwidth due to their sharp edges.

2.1 Continuous Inner Product over Rectilinear Polygons

In practice, the layouts are divided into smaller disjoint or overlapping rectangular tiles before applying the transform to each of the individual tile. Therefore we consider continuous transforms over a rectangular subset $\boldsymbol{T} \in \mathbb{R}^2$ that we call a tile:

$$\boldsymbol{T} = [0, T_x) \times [0, T_y). \tag{2.1}$$

This is not restrictive as we can ultimately increase the size of the tile to cover the whole layout. An orthogonal basis of functions $\phi_{k,l}$ can be written as:

$$\{\phi_{k,l} : \boldsymbol{T} \longrightarrow \mathbb{C}\}_{k=0,l=0}^{\infty,\infty}$$

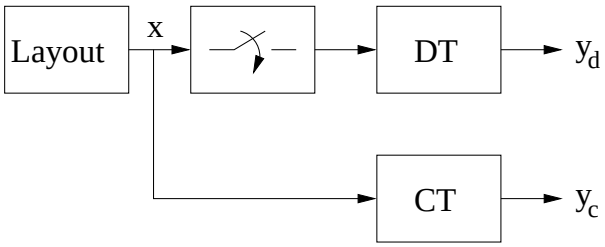


Figure 2.1: The standard discrete transform approach versus our proposed continuous transform approach.

and the transform coefficients of a function $f \in L^2(\boldsymbol{T})$ are simply the inner products with the basis functions $\{\phi_{k,l}\}$:

$$X_{k,l} = \langle f \,|\, \phi_{k,l} \rangle = \iint\limits_{\boldsymbol{T}} f(x,y)\phi^*_{k,l}(x,y)\,dx\,dy.$$

Since the polygons in a layout are all disjoint, the image f_T of a tile containing polygons $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1} \subseteq \boldsymbol{T}$ is the sum of the indicator functions of the polygons:

$$f_T(x,y) = \sum_{i=0}^{M-1} f_{\mathcal{P}_i}(x,y) = \sum_{i=0}^{M-1} \mathbb{1}_{\{(x,y)\in\mathcal{P}_i\}}$$

Using the linearity of the inner product, we can write:

$$\begin{aligned}
\langle f_T \,|\, \phi_{k,l} \rangle &= \iint\limits_{\boldsymbol{T}} \sum_{i=0}^{M-1} \mathbb{1}_{\{(x,y)\in\mathcal{P}_i\}} \phi^*_{k,l}(x,y)\,dx\,dy \\
&= \sum_{i=0}^{M-1} \iint\limits_{\boldsymbol{T}} \mathbb{1}_{\{(x,y)\in\mathcal{P}_i\}} \phi^*_{k,l}(x,y)\,dx\,dy \\
&= \sum_{i=0}^{M-1} \iint\limits_{\mathcal{P}_i} \phi^*_{k,l}(x,y)\,dx\,dy.
\end{aligned}$$

Therefore the inner product is the sum of the double integral of the basis functions over the M polygonal domains.

Computing the integrals over the polygonal domain also turns out to be rather straightforward. Using (1.3) (or (1.4)) we can rewrite the integral over a polygonal domain $\mathcal{P}$ as a sum of integrals over rectangles:

$$\langle f_{\mathcal{P}} \,|\, \phi_{k,l} \rangle = \iint\limits_{\mathcal{P}} \phi^*_{k,l}(x,y)\,dx\,dy = \sum_{i=0}^{K-1} \int_0^{y_i} \int_{x_i}^{x_{i+1}} \phi^*_{k,l}(x,y)\,dx\,dy$$

where again, the indices should be understood modulo K. If $\phi_{k,l}$ is separable, i.e. $\phi_{k,l}(x,y) = \phi_k(x)\phi_l(y)$, which is the case for the Haar, Fourier and cosine basis, this can be further simplified:

$$\begin{aligned}
\langle f_{\mathcal{P}} \,|\, \phi_{k,l} \rangle &= \sum_{i=0}^{K-1} \int_0^{y_i} \int_{x_i}^{x_{i+1}} \phi^*_{k,l}(x,y)\,dx\,dy \\
&= \sum_{i=0}^{K-1} \int_0^{y_i} \phi^*_l(y)\,dy \int_{x_i}^{x_{i+1}} \phi^*_k(x)\,dx \\
&= \sum_{i=0}^{K-1} \left(\Phi^*_l(y_i) - \Phi^*_l(0)\right)\left(\Phi^*_k(x_{i+1}) - \Phi^*_k(x_i)\right) \\
&= \sum_{i=0}^{K-1} \Phi^*_l(y_i)\left(\Phi^*_k(x_{i+1}) - \Phi^*_k(x_i)\right) - \Phi^*_k(0) \underbrace{\sum_{i=0}^{K-1} \left(\Phi^*_k(x_{i+1}) - \Phi^*_k(x_i)\right)}_{=0} \\
&= \sum_{i=0}^{K-1} \Phi^*_l(y_i)\left(\Phi^*_k(x_{i+1}) - \Phi^*_k(x_i)\right) \qquad (2.2)
\end{aligned}$$

where Φ_k is a primitive of ϕ_k. Note also that every second term in this sum is actually zero because for vertical edges $x_i = x_{i+1}$. In practice, we can thus reduce the number of operations by two. However, to keep the notation light we keep it as in (2.2).

2.2 Haar Series

Now that we have seen in the previous section how we can compute continuous inner products, we will show how this can be applied to the computation of the continuous Haar series coefficients. First we start by providing a quick recall of the 1D and 2D Haar bases. Then, we will derive the algorithm and analyze its complexity.

2.2.1 Haar Basis

We start by describing the 1D Haar basis as it makes it easy to later present the 2D case.

1D Haar Basis

The Haar basis is an orthonormal basis on $[0, 1)$. It is composed of the scaling function

$$\varphi(t) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

and of the wavelet functions

$$\begin{aligned} \psi_{j,k}(t) &= 2^{\frac{j}{2}}\psi(2^j t - k) \\ j &= 1, 2, 3, \ldots \quad k = 0, \ldots, 2^j - 1 \end{aligned} \tag{2.3}$$

where $\psi(x)$ is the mother wavelet given by

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq t < 1 \\ 0 & \text{otherwise} \end{cases}.$$

2D Haar Basis

As the Haar basis is separable with regard to the x and y axes, we can define the 2D basis using the 1D basis. Now we want to work on the surface $\boldsymbol{T}$ defined in (2.1). We thus need to scale everything by $\frac{1}{\sqrt{T_x T_y}}$ to keep unit energy. The scaling function is thus:

$$\varphi(x, y) = \frac{1}{\sqrt{T_x T_y}} \varphi\left(\frac{x}{T_x}\right) \varphi\left(\frac{y}{T_y}\right). \tag{2.4}$$

We then have the product between the scaling function (corresponding to the low-pass filter in the terminology of DWT) in the direction of the y-axis and the wavelet function (high-pass of DWT) on the x-axis

$$\psi^{(hl)}_{j,k_x,k_y}(x, y) = \frac{2^j}{\sqrt{T_x T_y}} \psi\left(\frac{2^j}{T_x} x - k_x\right) \varphi\left(\frac{2^j}{T_y} y - k_y\right), \tag{2.5}$$

and vice versa

$$\psi^{(lh)}_{j,k_x,k_y}(x, y) = \frac{2^j}{\sqrt{T_x T_y}} \varphi\left(\frac{2^j}{T_x} x - k_x\right) \psi\left(\frac{2^j}{T_y} y - k_y\right). \tag{2.6}$$

Finally we have the wavelets

$$\psi^{(hh)}_{j,k_x,k_y}(x, y) = \frac{2^j}{\sqrt{T_x T_y}} \psi\left(\frac{2^j}{T_x} x - k_x\right) \psi\left(\frac{2^j}{T_y} y - k_y\right) \tag{2.7}$$

with $j \in \mathbb{N}$, $k_x \in \{0, \ldots, 2^j - 1\}$, $k_y \in \{0, \ldots, 2^j - 1\}$. j represents the scale, k_x, k_y represent the shifts in the x, respectively y, directions. The space-frequency tiling induced by this basis is represented in Fig. 2.2.

Since the polygons boundaries are on integer lines, if we restrict the width and height of the tile to be $T_x = T_y = 2^J$ (this is not restrictive, since we can zero-pad it), then the continuous transform will yield the same result as the discrete Haar filter bank. In addition, all transform coefficients with scale $j \geq J$ will be zero.

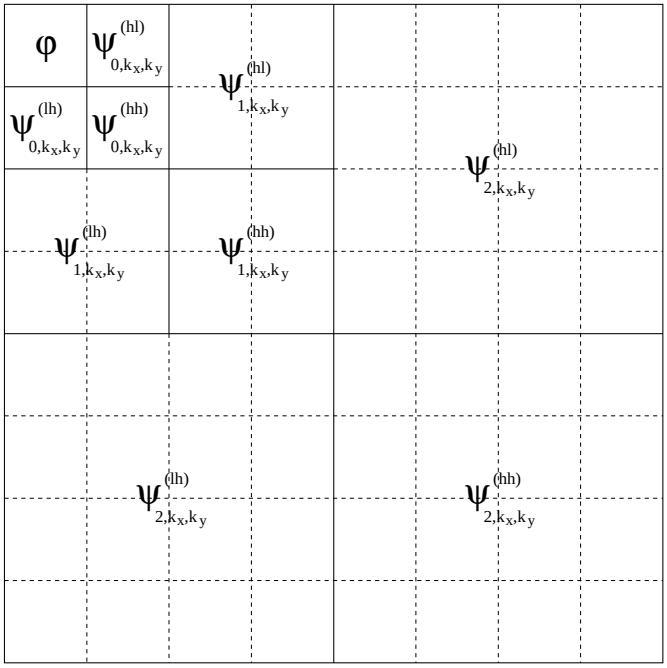


Figure 2.2: The space-frequency tiling on three scales. h and l indicate respectively the high pass and low pass operations.

2.2.2 Pruned Haar Transform

The decomposition into a basis of continuous wavelets constructed from iterated filter bank can be computed using the fast orthogonal wavelet transform (FWT) [15]. First the 1D algorithm for the continuous FWT (CFWT) is described and then extended to 2D. Secondly, we will argue why the Discrete FWT is generally more efficient. Finally, a pruned version of the FWT specifically for rectilinear polygons will be described.

1D CFWT

A continuous wavelet transform constructed using an iterated filter bank has the following properties:

$$\varphi(t) = \sqrt{2}\sum_n g_n \varphi(2t-n) \tag{2.8}$$

$$\psi(t) = \sqrt{2}\sum_n h_n \varphi(2t-n) \tag{2.9}$$

where g_n and h_n are the taps of the discrete-time filter bank [16]. Now in the same way we defined $\psi_{j,k}(t)$ in (2.3) we can define the scaling function at different scales:

$$\begin{aligned}\varphi_{j,k}(t) &= 2^{\frac{j}{2}}\varphi(2^j t-k)\\ j &= 1,2,3,\ldots \quad k=0,\ldots,2^j-1\end{aligned} \tag{2.10}$$

Now it is easy to rewrite (2.8) and (2.9) for an arbitrary scale :

$$\varphi_{j,k}(t) = \sum_n g_n \varphi_{j+1,2k+n}(t), \tag{2.11}$$

$$\psi_{j,k}(t) = \sum_n h_n \varphi_{j+1,2k+n}(t). \tag{2.12}$$

Suppose that we fix the maximum desired resolution of the transform to J, then the linearity of the inner product and the recursivity of those two relations mean that we can express every transform

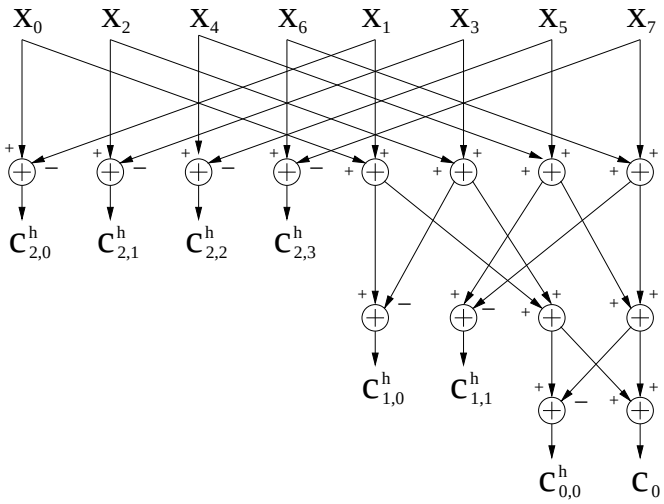


Figure 2.3: An example of Cooley-Tukey type 1D Haar FWT signal flow with a maximum resolution of $J = 3$ scales. $X_k = \langle f \,|\, \varphi_{J,k} \rangle$, $k = 0, \ldots, 2^J - 1$. $C_{j,k}$ are the transform coefficients. For simplicity, the scaling of the transform coefficients is omitted. The inner products can be understood as either discrete or continuous.

coefficients as a linear combination of $\{\langle f \,|\, \varphi_{J,k} \rangle\}_{k=0}^{2^J-1}$, that is the inner products with the shifts of the scaling function at scale J. Thus, those inner products need only be computed once.

Finally, a Cooley-Tukey like butterfly structure can be used to compute the transform coefficients from this set of inner products [17]. An example for $J = 3$ is given in Fig. 2.3.

2D CFWT

Using the separability of the 2D transform, we can directly apply (2.11) and (2.12) to (2.4) to (2.7) to obtain:

$$\varphi_{j,k_x,k_y}(x,y) = \sum_n \sum_m g_n g_m \varphi_{j+1,2k_x+n,2k_y+m}(x,y), \tag{2.13}$$

$$\psi^{(hl)}_{j,k_x,k_y}(x,y) = \sum_n \sum_m h_n g_m \varphi_{j+1,2k_x+n,2k_y+m}(x,y), \tag{2.14}$$

$$\psi^{(lh)}_{j,k_x,k_y}(x,y) = \sum_n \sum_m g_n h_m \varphi_{j+1,2k_x+n,2k_y+m}(x,y), \tag{2.15}$$

$$\psi^{(hh)}_{j,k_x,k_y}(x,y) = \sum_n \sum_m h_n h_m \varphi_{j+1,2k_x+n,2k_y+m}(x,y). \tag{2.16}$$

where n and m take values 0 and 1 for the Haar basis. As for the 1D transform, these relations induce a 2D butterfly structure as illustrated in Fig. 2.4.

As in the 1D case, for a transform with maximal resolution J, all the transform coefficients can be computed as a linear combination of

$$\left\{ \left\langle f \,|\, \varphi_{J,k_x,k_y} \right\rangle \right\}_{k_x=0,k_y=0}^{2^J-1,2^J-1}$$

using the relations given by (2.13) to (2.16).

CFWT vs. DFWT

If we suppose that the tile size is $N \times N$, with $N = 2^J$, then all coefficient with $j \geq J$ will be zero. Thus, if we want to perform a full decomposition using the CFWT, we need to compute $N \times N$ inner products with the scaling function at scale $j = J$. Since each inner product has complexity $O(K)$, where K is the

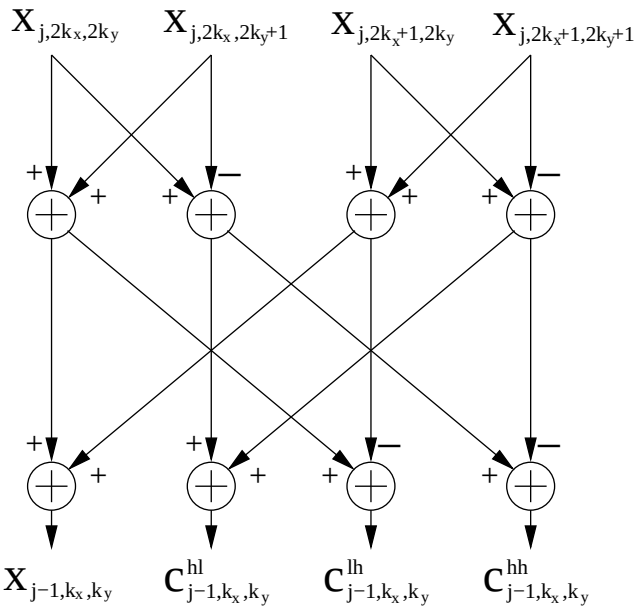


Figure 2.4: One butterfly of the 2D Haar CFWT. $X_{j,k_x,k_y} = \langle f \,|\, \varphi_{j,k_x,k_y} \rangle$. C^{xx}_{j,k_x,k_y} are the transform coefficients. Scaling of the transform coefficients is omitted for simplicity.

number of vertices of the polygon, the total complexity of computing only those inner product at $j = J$ is $O(KN^2)$. However, the scaling function at scale $j = J$ is just a 1×1 square and hence taking all those inner product corresponds in fact to sampling the polygon and creating a digital image. But this is not the smartest way to do it.

Better is to follow the rectilinear polygon boundaries and directly fill it. This has roughly complexity of $O(A)$ where A is the area of the polygon. This is necessarily smaller than the tile area N^2 and hence much smaller than KN^2.

In addition, the CFWT and the DFWT are identical, except for the definition of the inner product. Since in this case the discrete and continuous inner product agree, the complexity of the butterfly structure of the algorithm will be $O\left(N^2\right)$ in both cases. The DFWT is thus faster.

If for some reason however, we only want to compute the coefficients up to some scale $j_0 < J$, things change. Only $N^2 2^{-2j_0}$ inner products are required and one inner product no longer corresponds to one pixel of the digital image. Thus, the complexity of taking the discrete inner products is $O\left((2^{j_0} - 1)^2 2^{-2j_0} N^2\right)$. While the complexity of the DFWT is now $O\left(A + \left((2^{j_0} - 1)^2 + 1\right) 2^{-2j_0} N^2\right)$, the CFWT is reduced to $O\left((K + 1)2^{-2j_0} N^2\right)$, K being the total number of vertices of the polygons in the tile.

We will now see how the knowledge of the rectilinear polygons can help prune the butterfly signal flow and reduce the complexity.

Pruned Haar Transform

The Haar basis acts as a discontinuity detector and all the transform coefficients will be zero except for basis functions that intersect an edge of the polygon. Luckily the description of the polygon given by (1.2) gives exactly this information. This means that we can simply follow the perimeter of the polygon and compute the inner products that are not zero as illustrated in Fig. 2.5. For the full decomposition, we start at $j = J$, compute the inner products on the perimeter and then go up by one scale. Then, once more we go around the polygon and reuse whenever possible the inner products computed at the scale below. If one inner product is missing, we use (2.2) to compute it. An example of the pruned signal flow obtained for a 1D signal is shown in Fig. 2.6.

In addition the three types of Haar basis functions $\psi^{(hl)}_{j,k_x,k_y}$, $\psi^{(lh)}_{j,k_x,k_y}$ and $\psi^{(hh)}_{j,k_x,k_y}$ detect different kinds of features, respectively vertical edges, horizontal edges and vertices (i.e. corners). This means that the

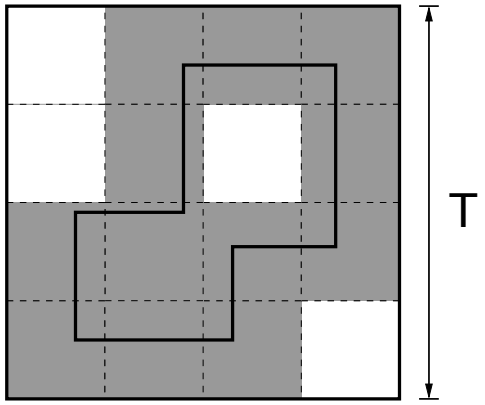


Figure 2.5: The Haar basis functions have zero yield zero inner product except when they intersect the edge of a polygon. The big thick square is the tile with $T_x = T_y = T$. The dashed squares are the supports of the basis functions at a given scale j_0. The thick polygonal border marks the contour of the polygon. Basis functions yielding non-zero inner product are in grey.

inner product with $\psi^{(hl)}_{j,k_x,k_y}$, $\psi^{(lh)}_{j,k_x,k_y}$, $\psi^{(hh)}_{j,k_x,k_y}$ are non-zero if and only if they respectively intersect a vertical edge, a horizontal edge and a vertex. However, if the feature, edge or vertex is placed on the j^{th} scale grid, the inner product will be zero anyway.

The algorithm for computing the Haar inner products is described in Algorithm 1.

2.2.3 Computational Complexity

We will now estimate the complexity of the pruned Haar transform algorithm that was derived in the previous section. As the complexity of the algorithm is highly dependent on the geometry of the polygon to be transformed, we will make a few assumptions to simplify the derivation. First, we suppose that we do not make use of inner products that were computed at previous scales. This is a conservative hypothesis and thus will lead to an overestimation of the complexity. We also leave out the scaling by a constant operations as they are equivalent in both continuous and discrete algorithms.

To begin with, computing the intersection area between two rectangles requires 2 additions and 1 multiplication. Since a rectilinear polygon can be described as union and differences of $K/2$ rectangles (see (1.3) or (1.4), where K is the number of vertices of the polygon, the intersection area has a complexity of K additions and $K/2$ multiplications.

Since the inner product of a polygon with $\psi^{(HL)}_{j,k,l}$, $\psi^{(LH)}_{j,k,l}$ or $\psi^{(HH)}_{j,k,l}$ can be computed by calculating the intersection area of the positive and negative parts of the basis function with the polygon and then add/subtract them to/from each other, the number of operations needed is:

$$\text{MUL}_{HL} = \text{MUL}_{LH} = K + 1, \qquad \text{ADD}_{HL} = \text{ADD}_{LH} = 2K + 1,$$

$$\text{MUL}_{HH} = 2K + 3, \qquad \text{ADD}_{HH} = 4K + 3,$$

for a polygon (i.e. $K > 4$). For a rectangle, this reduces to:

$$\text{MUL}_{HL} = \text{MUL}_{LH} = 3, \qquad \text{ADD}_{HL} = \text{ADD}_{LH} = 6,$$

$$\text{MUL}_{HH} = 7, \qquad \text{ADD}_{HH} = 11.$$

On an edge of length L, there are roughly $\left\lceil \frac{L}{2^{J-j_0}} \right\rceil$ inner products (HL or LH) to take at a given scale $j = j_0$. In addition, there is, also roughly, one inner product (HH) to take on each vertex. Thus the complexity of the transform is approximately:

$$C_{CFWT} \approx \sum_{j=0}^{J-1} \left(\sum_{\text{all edges } e_i} \left\lceil \frac{L_i}{2^{J-j}} C_{HL} \right\rceil + K C_{HH} \right)$$

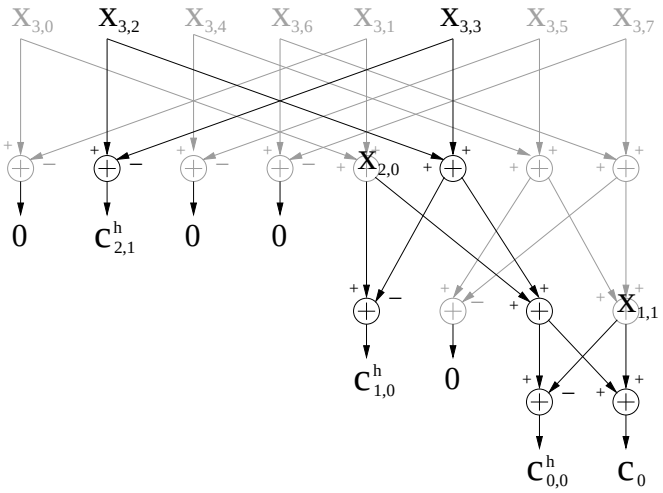


Figure 2.6: An example of the pruned signal flow of the Haar CFWT. The signal transformed is $f(t) = u(t-3)$ where $u(t)$ is the Heaviside function on the interval $[0, 8)$.

where C_{HL} and C_{HH} are the complexity of taking the inner product with $\psi^{(HL)}_{j,k,l}$ and $\psi^{(HH)}_{j,k,l}$ respectively. The rounding operation makes it difficult to estimate the complexity. We use an optimistic approximation and remove it.

$$\begin{aligned} C_{CFWT} &\approx \sum_{j=0}^{J-1} \left(\sum_{\text{all edges } e_i} \frac{L_i}{2^{J-j}} C_{HL} + K C_{HH} \right) && (2.17) \\ &= \frac{P}{2^J} C_{HL} \sum_{j=0}^{J-1} 2^j + JK C_{HH} && (2.18) \\ &= \left(1 - 2^{-J}\right) P C_{HL} + JK C_{HH} && (2.19) \\ &\approx P C_{HL} + JK C_{HH} \end{aligned}$$

where P is the perimeter of the polygon. So finally, in number of multiplications and additions, this gives for a polygon ($K > 4$):

$$\text{MUL}_{CFWT} = 2JK^2 + (3J + P)K + P, \qquad \text{ADD}_{CFWT} = 4JK^2 + (3J + 2P)K + P,$$

for a total of:

$$C_{CFWT} = 6JK^2 + (6J + 3P)K + 2P$$

operations. On the other hand, for the Discrete FWT, there are two steps, the creation of the discrete image and the Cooley-Tukey type algorithm. The creation of the discrete image does not involve any addition or multiplication and we found it to be, in practice, time-wise negligible compared to the actual transform (however it still requires $O(N^2)$ storage), and we thus neglect it.

The Cooley-Tukey structure has 2^{2j_0} butterflies at scale $j = j_0$ and each one of them requires eight additions. This makes a total of:

$$\text{ADD}_{DFWT} \approx 8 \sum_{j=0}^{J-1} 2^{2j} = 8 \frac{4^J - 1}{3} = 8 \frac{N^2 - 1}{3}$$

additions and no multiplications.

The complexity for different numbers of vertices and tile sizes is shown in Fig. 2.7 and Fig. 2.8. We observe that the pruned CFWT is faster than the DFWT only up to some number in the order of thirty vertices above which the DFWT should be preferred. For example, for a tile size of 256nm×256nm, we can see in Fig. 2.7 that the number of operations required by the DFWT is higher than for the pruned

Algorithm 1 PrunedHaarFWT($\boldsymbol{x}, \boldsymbol{y}, K, J, LL, HL, LH, HH$)

Require: The lists $\boldsymbol{x}$ and $\boldsymbol{y}$ of the K vertices of the rectilinear polygon. The number of scales J assuming that the tile size is $2^J \times 2^J$. IntersectVerticalEdge and IntersectHorizontalEdge are functions that check if a given basis function intersects the polygon. ContainsVertex checks if a given basis function contains a vertex of the polygon. The InnerProduct functions compute the inner products with the different basis functions.

Ensure: LL,HL,LH,HH contain the coefficients corresponding to the appropriate basis functions.

```
for j = J − 1 down to 0 do
  for k_x = 0 to 2^j − 1 do
    for k_y = 0 to 2^j − 1 do
      if IntersectVerticalEdge(x, y, K, j, k_x, k_y) then
        HL[j, k_x, k_y] ← InnerProductHL(x, y, K, j, k_x, k_y)
      end if
      if IntersectHorizontalEdge(x, y, K, j, k_x, k_y) then
        LH[j, k_x, k_y] ← InnerProductLH(x, y, K, j, k_x, k_y)
      end if
      if ContainsVertex(x, y, K, j, k_x, k_y) then
        HH[j, k_x, k_y] ← InnerProductHH(x, y, K, j, k_x, k_y)
      end if
    end for
  end for
end for
LL ← InnerProductLL(x, y, K)
```

CFWT until we reach the critical number of thirty vertices. Our experiments show that this number is actually around twenty-six for our current implementation. In addition, we also see that for tile smaller than smaller of equal to 32nm×32nm, the DFWT is better whatever the number of vertices is. This actually also means that when we reach the scale $j_0 = 5$, at which the DFWT becomes faster, we should stop pruning the transform. In the particular case of IC layouts, it should be noted that some parameters like the number of vertices in a tile is strongly linked to the tile size. For example a large tile can contain many polygons and thus many vertices while a small tile can hardly contain more than a single rectangle. A summary of the computational complexity of all the transforms is given in Table 2.2.

From this analysis, we see that the number total number of vertices of all the polygons in a tile is critical parameter of the complexity analysis. As can be seen in Fig. 3.5 from Section 3.1, this number is in most cases between four and thirty with extreme values going up to seventy making our algorithm suitable for IC layout processing. A more detailed analysis is given in Section 3.1.

2.3 Fourier Series

In this section, we will develop the algorithm for the computation of the Fourier series coefficients of rectilinear polygons. The computational complexity of this algorithm will also be evaluated and compared to that of the FFT.

2.3.1 Fourier Basis

The 2D Fourier basis is:

$$\left\{ \frac{1}{\sqrt{T_x T_y}} e^{j(w_x kx + w_y ly)} \right\}_{(k,l)\in\mathbb{Z}^2}$$

with $w_x = \frac{2\pi}{T_x}$ and $w_Y = \frac{2\pi}{T_y}$. The Fourier basis assumes that f_T is periodic with period T_x in the x direction and with period T_y in the y direction.

The main difference with the Discrete Fourier Transform (DFT) is that, since the functions are continuous in the spatial domain, they are not periodic in the frequency domain. This means that we need an infinite number of coefficients to have a perfect reconstruction of the image.

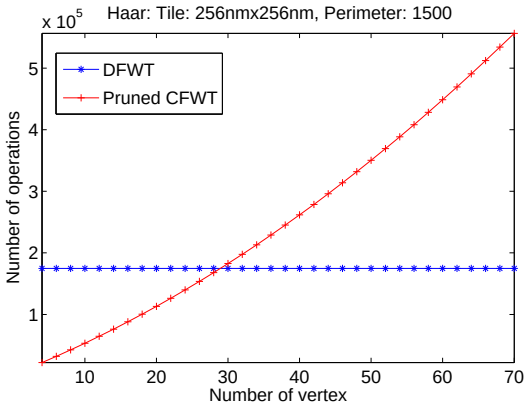


Figure 2.7: Comparison of the complexity of the Pruned CFWT versus the DFWT as a function of the number of vertices of the polygon to transform for a fixed tile size of 256nm×256nm.

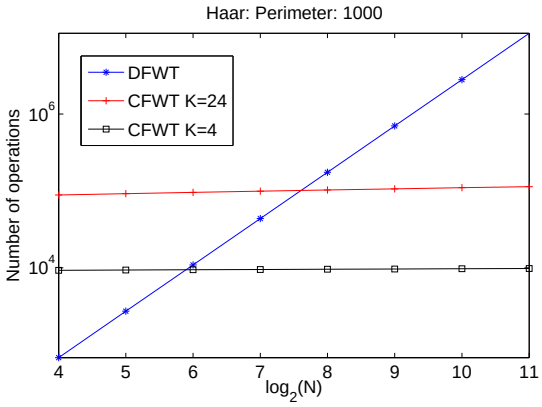


Figure 2.8: Comparison of the complexity of the pruned CFWT versus the DFWT as a function of the tile side size N. The complexity of the pruned CWFT is given for both a rectangle and a polygon with 24 vertices.

On the other hand, the DFT makes the assumption that the discrete signal is a sampled version of a band-limited continuous signal. This is clearly not the case for rectilinear polygons since they have an infinite bandwidth due to the presence of discontinuities at the edges. Therefore, the Fourier Series will yield a better approximation of the spectrum of the continuous image.

In addition, the Fourier series coefficients retain the Hermitian symmetry property for real-valued signals:

$$X_{k,l} = X^*_{-k,-l}. \tag{2.20}$$

2.3.2 Algorithm Derivation

Using (2.2), we can write the Fourier series coefficients as a function of the polygon vertices:

$$\begin{aligned}
X_{0,0} &= \alpha_{0,0} \sum_{i=0}^{K-1} y_i(x_{i+1} - x_i) \\
X_{k,0} &= \alpha_{k,0} \sum_{i=0}^{K-1} (y_{i-1} - y_i) e^{-jw_x k x_i} \\
X_{0,l} &= \alpha_{0,l} \sum_{i=0}^{K-1} (x_{i+1} - x_i) e^{-jw_y l y_i} \\
X_{k,l} &= \alpha_{k,l} \sum_{i=0}^{K-1} e^{-jw_y l y_i} \left(e^{-jw_x k x_{i+1}} - e^{-jw_x k x_i} \right),
\end{aligned}$$

where the scaling factor $\alpha_{k,l}$ is defined as:

$$\alpha_{k,l} = \begin{cases} \frac{1}{\sqrt{T_x T_y}} & \text{if } k = l = 0 \\ \frac{j}{2\pi k}\sqrt{\frac{T_x}{T_y}} & \text{if } k \neq 0 \text{ and } l = 0 \\ \frac{j}{2\pi l}\sqrt{\frac{T_y}{T_x}} & \text{if } k = 0 \text{ and } l \neq 0 \\ -\frac{\sqrt{T_x T_y}}{4\pi^2 k l} & \text{if } k \neq 0 \text{ and } l \neq 0 \end{cases}.$$

If we overlook the scaling factor, we notice that:

1. $X_{0,0}$ is proportional to the area of the polygon.

2. $X_{k,0}$ is proportional to the DFT of a 1D discrete signal which is zero everywhere except at the horizontal position of the vertices of the polygon where it is equal to the cumulated length of the vertical edges at this position:

$$\tilde{f}_x[n] = \sum_{i:n \equiv x_i \bmod T_x} (y_{i-1} - y_i), \quad n = 0, \ldots, T_x - 1. \tag{2.21}$$

The edges have a positive or negative value depending on their direction.

3. $X_{0,l}$ is proportional to the DFT of a discrete signal which is zero everywhere except at the vertical position of the vertices of the polygon where it is equal to the cumulated length of the horizontal edges at this position:

$$\tilde{f}_y[n] = \sum_{i:n \equiv y_i \bmod T_y} (x_{i+1} - x_i), \quad n = 0, \ldots, T_y - 1. \tag{2.22}$$

The edges have have a positive or negative value depending on their direction.

4. $X_{k,l}$ is proportional to the DFT of a 2D discrete image which is zero everywhere except at the position of the vertices of the polygon where it is either $+1$ or -1:

$$\tilde{f}_{x,y}[m,n] = \sum_{i=0}^{K-1} \left(\mathbb{1}_{\{m \equiv x_{i+1} \bmod T_x, n \equiv y_i \bmod T_y\}} - \mathbb{1}_{\{m \equiv x_{i+1} \bmod T_x, n \equiv y_i \bmod T_y\}} \right), \\ m = 0, \ldots, T_x - 1, \quad n = 0, \ldots, T_y - 1. \tag{2.23}$$

This means that we can compute all the coefficients using two 1D Fast Fourier Transforms (FFT) with $N/2$ non-zero inputs and length respectively T_x and T_y and one 2D FFT with K non-zero inputs and length $T_x \times T_y$ and later multiply by the scaling factor $\alpha_{k,l}$. FFTs with only a few non-zero input coefficients can be efficiently computed using the transform decomposition (TD) technique [18]. The full algorithm is described in Algorithm 2 where TD-FFT-1D, and TD-FFT-2D are respectively the 1D and 2D TD fast Fourier transforms.

However, if we want to compute only a few output coefficients, for example when we compute the convolution with a filter such as (1.1), it might be more efficient to use Goertzel algorithm [19] or traditional pruning techniques [20, 21].

For the sake of comparison to a standard FFT, we only implemented the algorithm described first which computes the first $T_x \times T_y$ coefficients. Furthermore, since the data are real-valued, we actually need only compute roughly half of those values as the rest can be determined by symmetries. Indeed, we can compute from this reduced set of values any given coefficient of the Fourier series by using the periodicity of the DFT and (2.20).

For example, let $\tilde{X}_{k,l}$ be a DFT coefficient of (2.23). Then

$$\begin{aligned} X_{k+nT_x, l+mT_y} &= \alpha_{k+nT_x, l+mT_y} \tilde{X}_{k+nT_x, l+mT_y} && (2.24) \\ &= \alpha_{k+nT_x, l+mT_y} \tilde{X}_{k,l} && (2.25) \end{aligned}$$

and thus

$$\begin{aligned} X_{k+nT_x, l+mT_y} &= \frac{\alpha_{k+nT_x, l+mT_y}}{\alpha_{k,l}} X_{k,l} \\ &= \frac{kl}{(k+nT_x)(l+mT_y)} X_{k,l}. \end{aligned}$$

for $n, m \in \mathbb{Z}$.

2.3.3 Computational Complexity

As in the Haar case, the computational complexity required to compute the Fourier series coefficients will depend on the geometry of the polygon to transform. In this case, only the number of vertices K

Algorithm 2 FourierSeries($\boldsymbol{x}, \boldsymbol{y}, K, N, F$)

Require: The lists $\boldsymbol{x}$ and $\boldsymbol{y}$ of the K vertices of the rectilinear polygon. The tile side length $N = 2^n$.
Ensure: F is an $N \times N$ matrix containing the $N \times N$ first Fourier series coefficients of the Polygon.

$K0[k] \leftarrow 0,\ k = 0, \ldots, N-1$
$L0[l] \leftarrow 0,\ l = 0, \ldots, N-1$
$KL[k,l] \leftarrow 0,\ k,l = 0, \ldots, N-1$
for $i = 0$ to $K-1$ **do**
  if $x_i = x_{i+1}$ **then**
    $K0\,[x_i] \leftarrow y_i - y_{i+1}$
  else if $y_i = y_{i+1}$ **then**
    $L0\,[y_i] \leftarrow x_{i+1} - x_i$
    $KL\,[x_{i+1}, y_i] \leftarrow 1$
    $KL\,[x_i, y_i] \leftarrow -1$
  end if
end for
DFT-1D($K0$)
DFT-1D($L0$)
DFT-2D(KL)
$F[0,0] \leftarrow \alpha_{0,0}\text{Area}(\boldsymbol{x}, \boldsymbol{y}, K)$
for $k = 1$ to $N-1$ **do**
  $F[k,0] \leftarrow \alpha_{k,0} K0[k]$
  $F[0,k] \leftarrow \alpha_{0,k} L0[k]$
  for $l = 1$ to $N-1$ **do**
    $F[k,l] \leftarrow \alpha_{k,l} KL[k,l]$
  end for
end for

of the polygon is important. To compare the different algorithms, we consider the input data to be complex-valued. It is not a problem since the complexity reduction induced by a real-valued input would be roughly one quarter for each algorithm (one half in each dimension) since we only need to compute a quarter of the outputs.

We will compare the split-radix FFT [22] performed on the discrete image of the polygon with our Fourier series algorithm using Goertzel or Transform Decomposition to compute the reduced input set DFTs required.

For the complexity analysis, we consider the transform of a $N \times N$ tile where N is a power of two. The complexity of the 2D $N \times N$ split-radix FFT is:

$$\begin{aligned} \text{MUL}_{FFT} &= 2N^2 log_2 N - 6N^2 + 8N \\ \text{ADD}_{FFT} &= 6N^2 log_2 N - 6N^2 + 8N \end{aligned}$$

for a total of:

$$C_{FFT} = 8N^2 log_2 N - 12N^2 + 16N$$

real operations. Goertzel algorithm requires roughly $O(8N)$ per output points for an N-points DFT. The TD algorithm requires [18]:

$$\begin{aligned} \text{MUL}_{TD-FFT} &= N log_2 P - 3N + 4\frac{N}{P} + 4\frac{LN}{P} - 4L \\ \text{ADD}_{TD-FFT} &= 3N log_2 P - 3N + 4\frac{N}{P} + 2\frac{LN}{P} - 2L \end{aligned}$$

where L is the number of input points, vertices in our case, and P is the size of the sub-FFT used:

$$P = \lfloor 4(L+1)\log_e 2 \rfloor . \tag{2.26}$$

Our algorithm requires two length-N DFTs with $K/2$ inputs each and one 2D $N \times N$ DFT with K inputs. If we use Goertzel algorithm the number of real operations required is then roughly:

$$C_{FS-G} = 8KN^2 + 8KN + 2K. \tag{2.27}$$

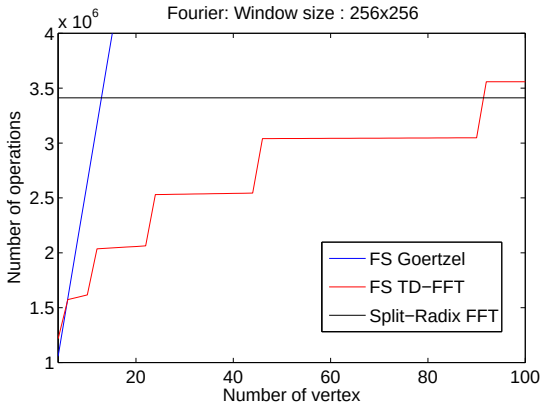


Figure 2.9: Comparison of the complexity of the Fourier series (FS) versus the FFT as a function of the number of vertices of the shape to transform for a fixed tile size of 256nm×256nm.

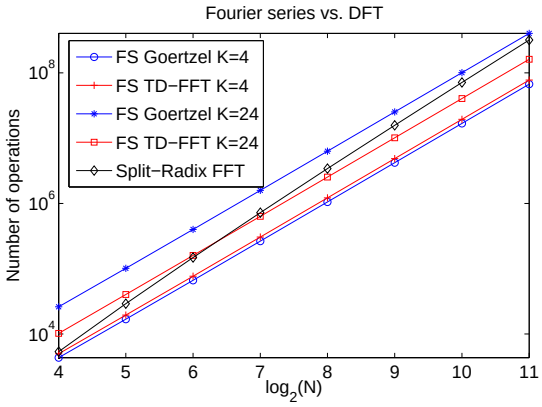


Figure 2.10: Comparison of the complexity of the Fourier series (FS) versus the FFT as a function of the tile side size N. The FS complexity is given for both a rectangle and a polygon with 24 vertices.

On the other hand if we use the TD algorithm, we need a total of:

$$C_{FS-TD} = 8N\log_2 P_1 - 12N + 16\frac{N}{P_1} + 12\frac{KN}{2P_1} - 4K + 8N^2\log_2 P_2 - 12N^2 + 16\frac{N^2}{P_2} + 40\frac{KN^2}{P_2^2} \quad (2.28)$$

where P_1 and P_2 are computed using (2.26) with respectively $L = K/2$ and $L = K$.

The complexity of the different algorithms are compared for different polygon size in Fig. 2.9 and for different tile size in Fig. 2.10. We can see that for rectangles, Goertzel offer the best performance for any given tile size. However, for non-rectangular polygons it degrades quickly when the number of vertices increases. On the other hand, TD increases more slowly with the number of vertices. At $N = 256$, it outperforms the FFT up to a hundred vertices. Therefore, to get the best performance, we should use Goertzel for rectangles and TD until the number of vertices exceeds a threshold given as a function of the tile size. A summary of the computational complexity of all the transforms is given in Table 2.2.

2.4 Cosine Series

In the previous section, an algorithm for the computation of Fourier series coefficients of rectilinear polygons was presented. In this section, we will show that it is possible to develop a similar algorithm using a continuous cosine basis. The computational complexity of this algorithm is compared to that of the FFT based DCT.

2.4.1 Cosine Basis

The projection on the cosine basis corresponds to the projection on the Fourier basis of the symmetric extension of $f_{\boldsymbol{T}}(x, y)$:

$$\hat{f}_{\boldsymbol{T}}(x,y) = \begin{cases} f_{\boldsymbol{T}}(|x|,|y|) & \text{if } -T_x < x < T_x \\ & \text{and } -T_y < y < T_y \\ 0 & \text{o.w.} \end{cases}.$$

The 2D continuous cosine basis is:

$$\left\{\gamma_{k,l}\cos\left(\frac{\pi}{T_x}kx\right)\cos\left(\frac{\pi}{T_y}ly\right)\right\}_{(k,l)\in\mathbb{N}^2}$$

where $\gamma_{k,l}$ is a scaling factor defined as:

$$\gamma_{k,l} = \begin{cases} \frac{1}{\sqrt{T_x T_y}} & \text{if } k = 0 \text{ and } l = 0 \\ \frac{2}{\sqrt{T_x T_y}} & \text{if } k \neq 0 \text{ and } l \neq 0 \\ \frac{\sqrt{2}}{\sqrt{T_x T_y}} & \text{otherwise} \end{cases}.$$

Since the cosine series is a particular case of the Fourier series applied to real-valued symmetric signals, we have the same distinction between the cosine series and the discrete cosine transform (DCT) than between the Fourier series and the DFT.

2.4.2 Algorithm Derivation

We have just seen how we can derive a fast algorithm for computing the Fourier series coefficients of a rectilinear polygon. Not surprisingly, it is possible to find a very similar algorithm for the computation of the cosine series coefficients.

Again by applying (2.2) to the cosine basis, we find the following expression for the coefficients:

$$X_{0,0} = \hat{\gamma}_{0,0} \sum_{i=0}^{K-1} y_i (x_{i+1} - x_i) \tag{2.29}$$

$$X_{k,0} = \hat{\gamma}_{k,0} \sum_{i=0}^{K-1} (y_{i-1} - y_i) \sin\left(\frac{\pi}{T_x} k x_i\right) \tag{2.30}$$

$$X_{0,l} = \hat{\gamma}_{0,l} \sum_{i=0}^{K-1} (x_{i+1} - x_i) \sin\left(\frac{\pi}{T_y} l y_i\right) \tag{2.31}$$

$$X_{k,l} = \hat{\gamma}_{k,l} \sum_{i=0}^{K-1} \left\{ \sin\left(\frac{\pi}{T_y} l y_i\right) \left(\sin\left(\frac{\pi}{T_x} k x_{i+1}\right) - \sin\left(\frac{\pi}{T_y} k x_i\right) \right) \right\} \tag{2.32}$$

where the scaling factor $\hat{\gamma}_{k,l}$ is:

$$\hat{\gamma}_{k,l} = \begin{cases} \frac{1}{\sqrt{T_x T_y}} & \text{if } k = l = 0 \\ \frac{\sqrt{2}}{\pi k} \sqrt{\frac{T_x}{T_y}} & \text{if } k \neq 0 \text{ and } l = 0 \\ \frac{\sqrt{2}}{\pi l} \sqrt{\frac{T_y}{T_x}} & \text{if } k = 0 \text{ and } l \neq 0 \\ 2 \frac{\sqrt{T_x T_y}}{\pi^2 k l} & \text{if } k \neq 0 \text{ and } l \neq 0 \end{cases}.$$

As was the case with the Fourier series, we notice that if we neglect the scaling factor, those four equations are respectively given by the area of the polygon, the 1D discrete sine transform (DST) of (2.21), the 1D DST of (2.22) and the 2D DST of (2.23).

Since the DFT and the DST are closely related transforms, it is possible to map the DST to an FFT. Therefore, we face the same implementation choice: Goertzel [19], TD [18], pruning [20, 21] or direct FFT-based implementation.

At the time of the implementation, we were not aware of the TD technique and have thus implemented the Goertzel algorithm. In addition, the DST that we would need to use is not a standard DST and is thus not implemented in common FFT libraries like FFTW.

We use the following induction to compute the sequence $S_k = a \sin kx$:

$$\begin{aligned} S_0 &= 0, \\ S_1 &= a \sin x, \\ S_k &= 2 S_{k-1} \cos x - S_{k-2}, \quad k = 2, 3, \ldots \end{aligned}$$

The full algorithm for the Goertzel based continuous cosine series coefficients computation can be found in Algorithm 3.

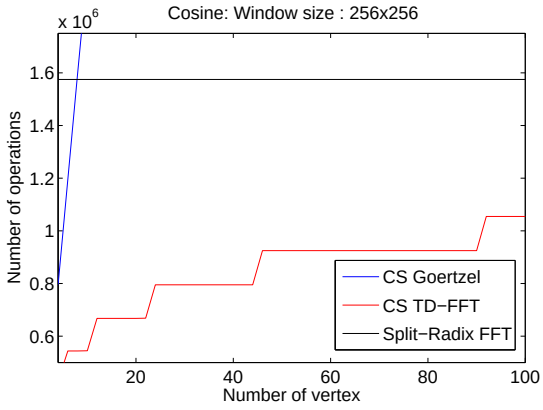


Figure 2.11: Comparison of the complexity of the cosine series (CS) versus the FFT for real symmetric data as a function of the number of vertices of the shape to transform for a fixed tile size of 256nm×256nm.

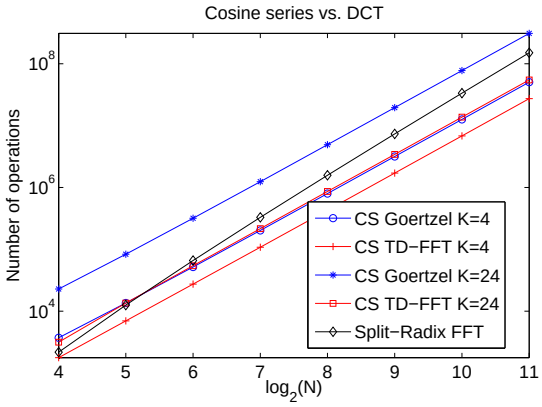


Figure 2.12: Comparison of the complexity of the cosine series (CS) versus the FFT for real symmetric data as a function of the tile side size N. The CS complexity is given for both a rectangle and a polygon with 24 vertices.

2.4.3 Computational Complexity

The computational complexity of computing the cosine transform is very similar to the Fourier transform since it is actually the Fourier transform of real symmetric data. Therefore, computing the cosine coefficients require about half the number of operations.

Computing the DCT of an $N \times N$ tile using the split-radix implementation requires:

$$C_{DCT} = 4N^2 \log_2 N - 8N^2 + 8N$$

real operations. The algorithm using Goertzel presented in the previous section uses a total of:

$$C_{CS-G} = 3N^2K + 10NK + K$$

real operations. Finally the same algorithm using TD for symmetric data instead of Goertzel would require roughly [18]:

$$C_{CS-TD} = 2K + 4N \log_2 P_1 + 8\frac{N}{P_1} - 8P_1 + 2N^2 \log_2 P_2 + 4\frac{N^2}{P_2} - 4P_2$$

where P_1 and P_2 are computed using (2.26) with respectively $L = K/2$ and $L = K$.

The complexity for different values of the number of vertices and the tile size is shown in Fig. 2.11 and Fig. 2.12. We can observe that Goertzel outperforms the standard DCT only when the polygons have up to a dozen vertices but then its complexity quickly rises. On the other hand, TD outperforms the DCT even for polygons with large numbers of vertices. For rectangles, TD outperforms the DCT whatever the tile size. For polygons with more vertices, the DCT might be faster for small tile size. However, the smaller the tile size, the fewer the number of vertices polygons in the tile can have. A summary of the computational complexity of all the transforms is given in Table 2.2.

List of parameters:	
$N \times N$	The tile size with $N = 2^J$.
J	$J = \log_2 N$.
K	The number of vertices of a single polygon.
P	The perimeter of a single polygon.
P_1	$P_1 = \lfloor 4(K/2+1)\log_e 2 \rfloor$
P_2	$P_2 = \lfloor 4(K+1)\log_e 2 \rfloor$

Table 2.1: List of the parameters used in the computational complexity analysis. Note that some of those parameters are highly interdependent. For example, the larger the tile size, the more likely it is to contain many polygons and thus many vertices. On the other hand, a small tile can hardly contain more than a single rectangle.

Complexity of the Transforms:			
	Continuous		Discrete
Haar	$6JK^2 + (6J+3P)K + 2P$		$8\frac{N^2-1}{3}$
	Goertzel	Transform Decomposition	
Fourier	$8KN^2 + 8KN + 2K$	$8N\log_2 P_1 - 12N + 16\frac{N}{P_1} + 12\frac{KN}{2P_1}$ $-4K + 8N^2\log_2 P_2 - 12N^2$ $+16\frac{N^2}{P_2} + 40\frac{KN^2}{P_2^2}\ 8KN^2$ $+8KN + 2K$	$8N^2 log_2 N - 12N^2 + 16N$
Cosine	$3N^2K + 10NK + K$	$2K + 4N\log_2 P_1 + 8\frac{N}{P_1} - 8P_1$ $+2N^2\log_2 P_2 + 4\frac{N^2}{P_2} - 4P_2$	$4N^2\log_2 N - 8N^2 + 8N$

Table 2.2: Summary of the computational complexity of the continuous and discrete transforms of a single rectilinear polygon in terms of total number of additions and multiplications. A list of the parameters is given in Table 2.1.

Algorithm 3 CosineSeries($\boldsymbol{x}, \boldsymbol{y}, K, N, F$)

Require: The lists $\boldsymbol{x}$ and $\boldsymbol{y}$ of the K vertices of the rectilinear polygon. The tile side length N is a power of two.

Ensure: F is an $N \times N$ matrix containing the $N \times N$ first cosine series coefficients of the Polygon.

for $i = 0$ to $K - 1$ **do**
  $S_i^x \leftarrow \sin\left(\frac{\pi}{N} x_i\right)$
  $C_i^x \leftarrow \cos\left(\frac{\pi}{N} x_i\right)$
  $S_i^y \leftarrow \sin\left(\frac{\pi}{N} y_i\right)$
  $C_i^y \leftarrow \cos\left(\frac{\pi}{N} y_i\right)$
  $S_i^{x,0} \leftarrow 0$
  $S_i^{y,0} \leftarrow 0$
  $S_i^{x,1} \leftarrow (y_{i-1} - y_i) S_i^x$
  $S_i^{y,1} \leftarrow (x_{i+1} - x_i) S_i^y$
end for
for $k = 1$ to $N - 1$ **do**
  for $i = 0$ to $K - 1$ **do**
    $F[0, k] \leftarrow F[0, k] + S_i^{y,k}$
    $F[k, 0] \leftarrow F[k, 0] + S_i^{x,k}$
    $S_i^{x,k+1} \leftarrow 2 S_i^{x,k} C_i^x - S_i^{x,k-1}$
    $S_i^{y,k+1} \leftarrow 2 S_i^{y,1} C_i^y - S_i^{y,k-1}$
  end for
end for
for $i = 0$ to $K - 1$ **do**
  $S_i^{x,0} \leftarrow 0$
  $S_i^{x,1} \leftarrow S_i^x$
end for
for $k = 1$ to $N - 1$ **do**
  for $i = 0$ to $K - 1$ **do**
    $S_i^{y,0} \leftarrow 0$
    $S_i^{y,1} \leftarrow \left(S_{i+1}^{x,k} - S_i^{x,k}\right) S_i^y$
  end for
  for $l = 1$ to $N - 1$ **do**
    for $i = 0$ to $K - 1$ **do**
      $F[k, l] \leftarrow F[k, l] + S_i^{y,1}$
      $S_i^{y,l+1} \leftarrow 2 S_i^{y,l} C_i^y - S_i^{y,l-1}$
    end for
  end for
  for $i = 0$ to $K - 1$ **do**
    $S_i^{x,k+1} \leftarrow 2 S_i^{x,k} C_i^x - S_i^{x,k-1}$
  end for
end for
for $k = 0$ to $N - 1$ **do**
  for $l = 0$ to $N - 1$ **do**
    $F[k, l] \leftarrow \hat{\gamma}_{k,l} F[k, l]$
  end for
end for

Chapter 3

Performance Evaluation

In this chapter, we first give an overview of the implementation of the algorithms presented in the previous chapter. These algorithms were implemented into the CL software tool in development at ZRL. we then present the results of benchmark where the performance of those algorithms is compared to that of traditional discrete transform algorithms on real IC layouts. Finally, the approximation power of the Haar basis for rectilinear polygons is evaluated. This evaluation is done both a normal and a contact layer from a real IC layout.

3.1 Algorithms Performance Evaluation

3.1.1 Implementation

The algorithms described Chapter 2 were implemented in the computational lithography software tool currently under development at ZRL. Due to critical speed requirements, we have written the software in C++. It takes as input a GL/1 file, parse it, chops polygons and places them in the corresponding tile. It is furthermore possible to perform online processing on the shapes as they are attributed to the tiles. Finally, different writer structures allow one to do some additional processing tile per tile once all the shapes in the layout have been attributed and/or to output some data (e.g. images of the tiles, transform coefficients, etc).

Since all the polygons are disjoint, the transforms can be done online as shapes are read from the GL/1 file as shown in Fig. 3.1. However, this has the disadvantage that the transform coefficients must be kept in memory until all the shapes are read. Even though only non-empty tiles are created and only non-zero coefficients are stored, the memory requirement is huge since the number of non-empty tiles can be well over 2 million, even for a modestly sized layout (i.e. a few megabytes file size). For 2 million tiles with each 600 non-zero coefficients, 8 GB of memory are needed, without accounting for overhead.

As this is clearly not practical, we store instead the polygons in the tiles they belong to, and perform the transform tile by tile once shapes have been read and attributed. This is illustrated in Fig. 3.2. However, at this level, the transform is still performed on a polygon by polygon basis.

Both the pruned CWFT and DWFT as described in Section 2.2.2 were implemented in the software tool. The cosine series was implemented using the Goertzel algorithm version while the DCT is performed using the FFTW3 library [23]. The Fourier series was implemented using the TD-FFT algorithm for the sub-FFT while FFTW3 was also used for the DFT. However, the Fourier series implementation is currently still not stable and therefore not included in the results presented here. For the discrete transforms, an image of the tile is first created and then fed to the transform algorithm.

3.1.2 Results

For the evaluation, the algorithms were run on a fairly large layout layer containing both rectangles and non-rectangular polygons. The tile size was chosen to be 256nm×256nm because it is close to what is used in practice in CL. The runtime was averaged for every number of vertices by tile present in the design. For the Haar transform it was also averaged per cumulated perimeter length of the polygons in a

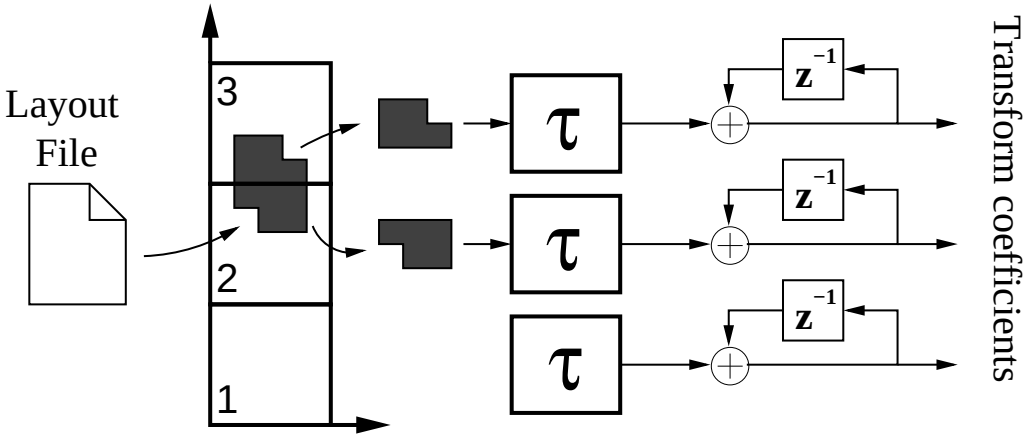


Figure 3.1: Illustration of how the transform can be computed online when the polygons are read from the layout file. Each polygon read from the layout file is chopped and placed in the corresponding tile where it is right away transformed. After transformation, the polygon do not need to be kept in memory. Since polygons are disjoint, the transform coefficients of the polygons in a tile can be accumulated to get the transform coefficients of the tile. The τ block represents the transform.

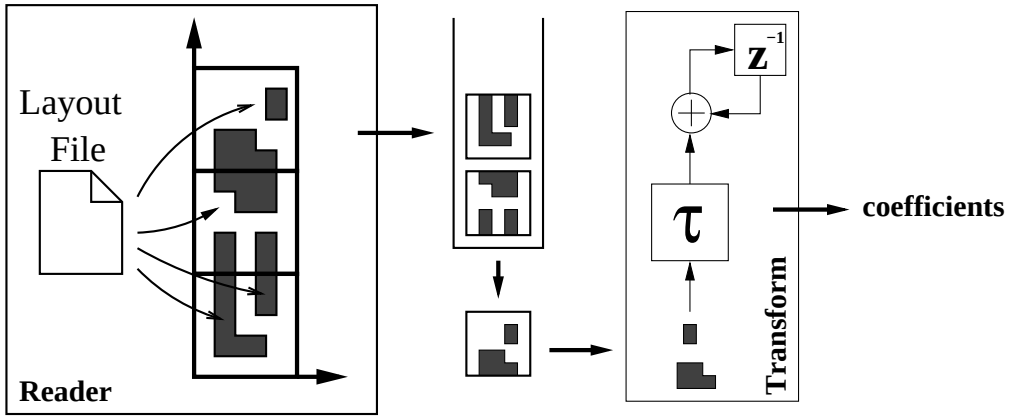
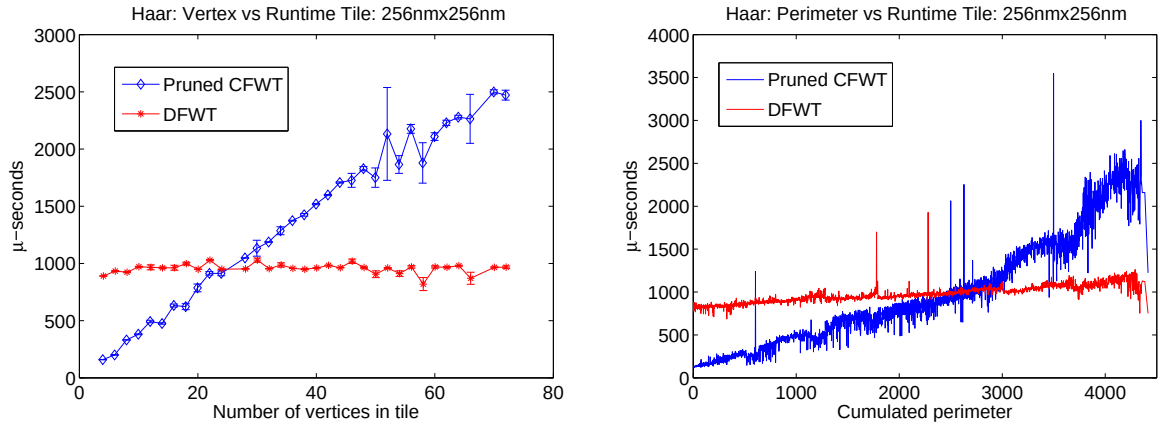


Figure 3.2: Illustration of how the transform can be computed offline when all the tiles have been filled. First, all the polygons are read from the layout file, chopped and stored in the corresponding tile. Then, tiles are transformed one by one. The τ block is the transform.

tile to highlight the dependency between the perimeter and the complexity. For the discrete transform, the time needed to create the discrete image is added to the runtime of the transform.

Results for the Haar transform are shown in Fig. 3.3. We can see that those results follow closely the complexity comparisons from Section 2.2.3. In Fig. 3.3 (a), the runtime as a function of the number of vertices in the tile is shown. We can observe that, as expected from the complexity analysis, the runtime of the pruned CFWT is highly dependent on the number of vertices while the runtime of the DFWT is constant for any number of vertices. Fig. 3.3 (b) shows the runtime as a function of the perimeter of the polygon. Again, as we expect from the complexity analysis, the pruned CFWT increases as the perimeter increases. But, in addition, we also observe that the runtime of the DFWT increases slightly with the perimeter. This comes from the fact that the discrete image creation process is a function of the vertices. However, compared to the transform itself, it is not significant.

For the cosine transform, the results are shown in Fig. 3.4. Here we only looked at the runtime versus the number of vertices in the tile. Here again, the results follow closely what we expect: the runtime of the Goertzel based cosine series coefficients computations rises sharply with the number of vertices



(a) Vertex vs. Runtime

(b) Perimeter vs. Runtime

Figure 3.3: The average runtime of the Haar transform as a function of (a) the sum of the number of vertices of all the polygons in the tile and (b) the cumulated perimeter of all the polygons in the tile. The tile size is fixed to 256nm×256nm. For (a), 95% confidence intervals are also plotted. These results validate the theoretical computational complexities derived in the previous chapter.

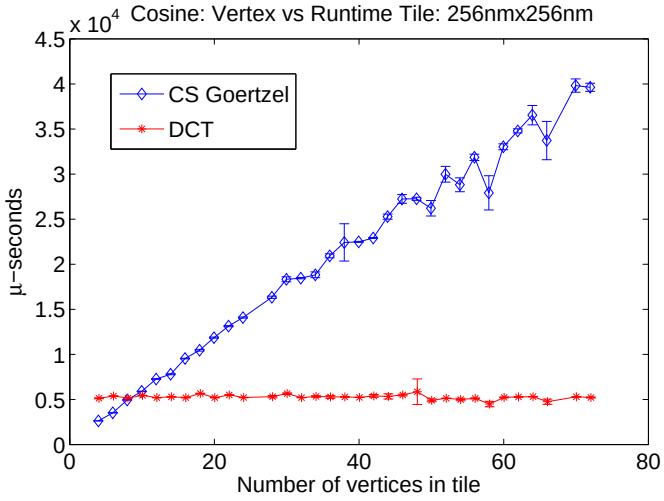
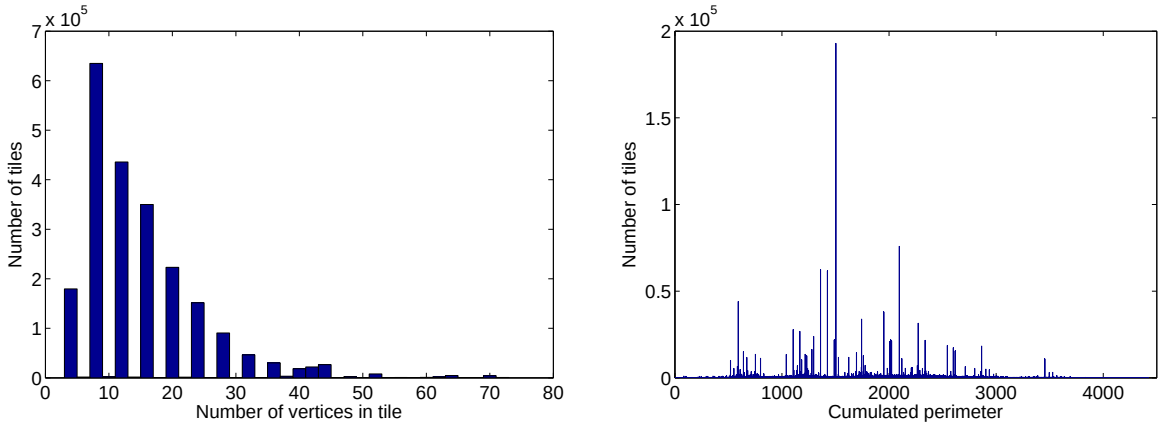


Figure 3.4: The runtime of the cosine transform as a function of the sum of all the number of vertices of the polygons in a tile. The tile size is fixed to 256nm×256nm. 95% confidence intervals are plotted. As expected from the theoretical complexity, the continuous algorithm based on Goertzel outperforms the discrete transform algorithm only when the number of vertices is below ten.

while the FFT-based DCT runtime stays constant for a given tile size. It will therefore be necessary to implement the cosine series algorithm based on TD-FFT rather than Goertzel in the future.

Finally we give also a histogram of the number of vertices in a tile and the cumulated perimeter in Fig. 3.5 in order to give an idea of where the average complexity lies. It can be observed in the former that although design layouts are mostly composed of rectangles, there are usually several of them in a single tile. In addition, one can commonly find tiles containing up to 50 vertices. The second histogram shows that the cumulated perimeter length is in most cases less than 3000nm, which falls in the area where the pruned CFWT outperforms the DFWT.

In Table 3.1 we give an overview of the properties of the layout and the average runtime of the Haar and cosine transform. About 93% of the polygons in the layout are rectangles while only 7% have more vertices. However, as seen in the histogram, most tiles contain between 10 and 20 vertices. This still falls



(a) (b)

Figure 3.5: Histograms of (a) the number of vertices per tile and (b) the cumulated perimeter length of the polygons a tile. We see that the average number of vertices per tile is far less than the area of the tile, we thus expect the continuous transform algorithms to be very useful in practice. We see as well that most of the cumulated perimeter is less than 3000nm, in the zone where the pruned CFWT outperforms the DFWT.

Properties of layout:		**Average Runtime:**		
# non-empty tiles	2233726		Continuous	Discrete
# Rectangles	1994018	Haar	601 μs	943 μs
# Polygons	144296	Cosine	9111 μs	5193 μs

Table 3.1: General properties of the layout used for performance comparisons and average runtime of all the transforms.

in the range where the pruned CFWT is faster than the DFWT, but it is too much for the Goertzel based cosine series as can be seen from the average runtime. The FFT-based DCT is on average almost twice faster. This means that we must implement the transform decomposition based cosine series algorithm before we can use it in practice.

3.2 Sparsity of Layouts in Haar Basis

Our second experiment concerns a measure of the sparsity of IC layouts in the Haar basis. This will be later useful to apply the Haar transforms to pattern matching or feature extraction applications in the context of the CL project underway at IBM ZRL. Concretely, in this experiment, we first transform a tile using the Haar basis and retain only L coefficients with the largest magnitude and then compute the resulting approximation error.

The simplest way to do this is to use the following approximation to the tile:

$$\hat{f}_{\boldsymbol{T}}(x,y) = \sum_{j=0}^{L-1} \tilde{C}_j \phi_j(x,y),$$

where $\tilde{C}_j$ is the j^{th} largest transform coefficient and ϕ_j the corresponding Haar basis function, and then compute the mean squared error (MSE). The squared error is given by:

$$\|e\|^2 = \iint_{\boldsymbol{T}} \left(f_{\boldsymbol{T}}(x,y) - \hat{f}_{\boldsymbol{T}} \right)^2 dx\, dy.$$

Using the orthogonality of the Haar basis, we can simplify this to:

$$\begin{aligned}\|e\|^2 &= \iint_{\boldsymbol{T}} \left|f_{\boldsymbol{T}}(x,y)\right|^2 \, dx\, dy - \iint_{\boldsymbol{T}} \left|\hat{f}_{\boldsymbol{T}}(x,y)\right|^2 \, dx\, dy \\ &= \sum_{i:\mathcal{P}_i \in \boldsymbol{T}} \text{Area}(\mathcal{P}_i) - \sum_{j=0}^{L-1} |\tilde{C}_j|^2.\end{aligned}$$

One can further normalize this by the cumulated area to get an error measure in the $[0,1]$ interval:

$$\|e\|^2 = 1 - \frac{1}{\sum\limits_{i:\mathcal{P}_i \in \boldsymbol{T}} \text{Area}(\mathcal{P}_i)} \sum_{j=0}^{L-1} |\tilde{C}_j|^2.$$

This is the error function that we use to measure the approximation power of the Haar basis for rectilinear polygons. Fig. 3.6 shows the result for a normal layer which do not contain only rectangles and a contact layer which contains only relatively small squares. The tile chosen size is 256nm $\times$ 256nm.

We observe that the Haar basis is indeed a good match for the rectilinear polygons from IC layouts. With about 35 coefficients (0.05% of the total number of coefficients, 256^2), the MSE falls under 10% of the polygon area. The MSE for the contact layer decreases sharply before going to zero after about 250 coefficients. However, in the case of the normal layer, after decreasing sharply for the first hundred coefficients, the MSE takes a much slower decrease rate with several hundred coefficients needed to go down by only one or two orders of magnitude. After about 800 coefficients, the MSE finally goes to zero. This difference is likely due to the fact that the rectangles of the contact layer are quite small and require thus much less coefficients to be described than the bigger polygons present in the normal layer.

However, these results do not tell the full story. One thing omitted in the approximation function we use is that the original image takes only values 0 or 1. Although we have not been able to pursue this approach yet due to time constraints, it is possible to use a better approximation function:

$$\tilde{f}_{\boldsymbol{T}}(x,y) = \sigma\left(\hat{f}_{\boldsymbol{T}}(x,y)\right)$$

where $\sigma(t)$ is a thresholding function with threshold θ:

$$\sigma(t) = \begin{cases} 1 & \text{if } t \geq \theta \\ 0 & \text{o.w.} \end{cases}$$

and then, the error measure becomes:

$$\|e\|^2 = \iint_{\boldsymbol{T}} \left(f_{\boldsymbol{T}}(x,y) - \sigma\left(\hat{f}_{\boldsymbol{T}}(x,y)\right)\right)^2 \, dx\, dy.$$

This error measure has however one practical problem: it is not clear what the value for the threshold θ should be. From a mathematical point of view, the solution would be to compute the optimal threshold to recover the original image given its statistical model. From a lithography point of view, we might be able to link this threshold to the amplitude threshold of the resist material used which determines the shape of the object printed in the end. The best would be of course to take both of these aspects into account.

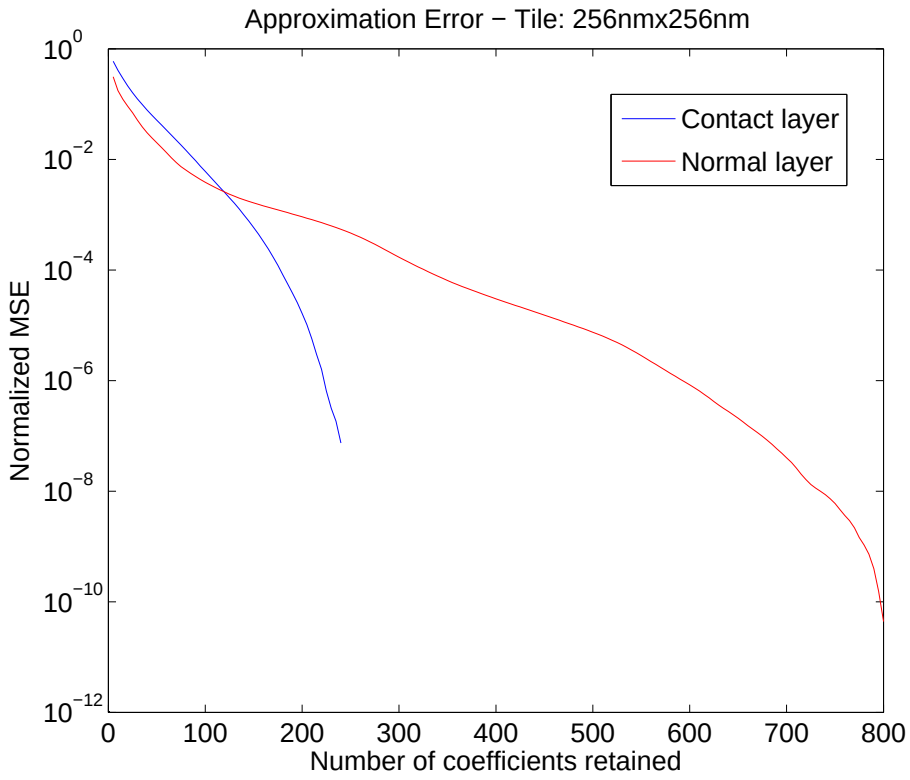


Figure 3.6: The normalized MSE for the linear Haar approximation of the tiles. We observe that the MSE for the contact layer decreases much more than for a normal layer as the number of coefficients in the approximation increases. This is because the elements in the contact layer are small, thus more spatially localized. The elements present in normal layers are usually bigger, thus require more coefficients to be better approximated. 95% confidence intervals were omitted because they were too narrow to be distinguishable in the figure.

Chapter 4

Conclusions

In this work, we developed a few algorithms to compute the continuous Haar, Fourier and cosine transform of rectilinear polygons. We showed that these algorithm can be faster than the standard discrete algorithms when applied to IC layout where the tiles typically contain polygons with relatively few vertices. However, when the number of vertices exceeds some threshold (typically a few tens), standard discrete transforms should be preferred. Faster Fourier and cosine transform are hoped to speed up the diffraction orders computation in SMO.

We computed and validated the theoretical complexities of the algorithms by a practical implementation in a computational lithography tool currently in development at ZRL and ran a benchmark on real IC layouts.

The second objective was to explore the use of the Haar transform for future pattern matching applications within IC layouts. A first step in this direction was accomplished by showing that the tile images are very sparse in the Haar basis. We hope to be able to use this to perform feature extraction in the future as part of the CL pattern matching project.

4.1 Future Work

Theory

Inverse transforms In this work we mainly developed how one goes from the computational geometry representation of polygons to their signal processing counterpart (transformed coefficients). From a theoretical point of view, it would be very interesting to find an inverse transform which from the transform coefficients would yield directly the vertices of the polygons. This leads to interesting questions such as how many coefficients would be needed to recover the original polygon, or what happens when you use less coefficients for the reconstruction. There might be connections with compressive sampling which would allow to develop faster lithography algorithms.

Statistical model of IC layouts Having a statistical model of the IC layouts could allow to develop optimal tools for the goal we are pursuing. Rate-distortion theory could be used to compute the degradation caused by dropping coefficients in approximations. Optimal decorrelating transforms for IC layouts could then be derived.

Pattern Matching

Frames Because we ultimately want to be able to identify tiles under rotation, symmetry or translation, translation invariant frames might be a powerful tool and should be investigated. The only concern here is the computational overhead introduced by frames. However, it might be possible to overcome this by using the knowledge of the structure of the polygons as we did with the pruned CFWT.

Non-linear approximation Because of the binary nature of the IC layouts it might be more suitable to use another non-linear approximation function to measure the IC layout sparsity in the Haar basis, as explained in the last section of the previous chapter.

Implementation

Fourier series The Fourier series algorithm is already under implementation but requires some more work to be fully operational.

Cosine series The cosine series implementation currently uses Goertzel algorithm. This is fairly inefficient and should be replaced by the TD algorithm for symmetric real data.

Pruned CFWT The pruned transform implementation requires a lot of control statement and is thus not fully efficient. One way to make this more efficient would be to create a transform plan that would decide when a new inner product should be computed or if we can reuse results from the previous scale.

Bibliography

[1] A. K. Wong, *Resolution Enhancement Techniques in Optical Lithography*, ser. Tutorial Texts in Optical Engineering. Bellingham, Washington USA: SPIE Press Book, Mar. 2001, vol. TT47.

[2] F. M. Schellenberg and B. W. Smith, "Resolution enhancement technology: the past, the present, and extensions for the future," in *Optical Microlithography XVII*, vol. 5377. Santa Clara, CA, USA: SPIE, May 2004, pp. 1–20.

[3] V. Singh, G. Guo, S. Liu, G. Jin, K. A. S. Immink, K. Shono, C. A. Mack, J. Kang, and J. en Yao, "Computational lithography: the new enabler of Moore's law," in *Quantum Optics, Optical Data Storage, and Advanced Microlithography*, vol. 6827. Beijing, China: SPIE, Nov. 2007, pp. 68 271Q–5.

[4] A. E. Rosenbluth, S. J. Bukofsky, M. S. Hibbs, K. Lai, A. F. Molless, R. N. Singh, A. K. K. Wong, and C. J. Progler, "Optimum mask and source patterns to print a given shape," in *Optical Microlithography XIV*, vol. 4346. Santa Clara, CA, USA: SPIE, 2001, pp. 486–502.

[5] A. Poonawala and P. Milanfar, "OPC and PSM design using inverse lithography: A non-linear optimization approach," in *Proceedings of the SPIE*, vol. 6154, 2006, pp. 1159–1172.

[6] J. Zhang, W. Xiong, M. Tsai, Y. Wang, and Z. Yu, "Efficient mask design for inverse lithography technology based on 2D discrete cosine transformation (DCT)," in *Simulation of Semiconductor Processes and Devices 2007*, 2007, pp. 49–52.

[7] X. Ma and G. R. Arce, "PSM design for inverse lithography with partially coherent illumination," *Optics Express*, vol. 16, p. 20126, Nov. 2008.

[8] J. W. Goodman, *Introduction to Fourier Optics*, 3rd ed. Roberts & Company Publishers, Dec. 2004.

[9] D. G. L. Chow, J. F. McDonald, D. C. King, W. Smith, K. Molnar, and A. J. Steckl, "An image processing approach to fast, efficient proximity correction for electron beam lithography," *Journal of Vacuum Science Technology B: Microelectronics and Nanometer Structures*, vol. 1, pp. 1383–1390, Oct. 1983.

[10] J. F. Chen, H. Liu, T. Laidig, C. Zuniga, Y. Cao, and R. Socha, "Development of a computational lithography roadmap," in *Proceedings of SPIE*, San Jose, CA, USA, 2008, pp. 69 241C–69 241C–12.

[11] M. E. Haslam, J. F. McDonald, D. C. King, M. Bourgeois, D. G. L. Chow, and A. J. Steckl, "Two-dimensional haar thinning for data base compaction in fourier proximity correction for electron beam lithography," *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, vol. 3, no. 1, pp. 165–173, 1985.

[12] J. Antoine, R. Murenzi, P. Vandergheynst, and S. T. Ali, *Two-Dimensional Wavelets and their Relatives*. Cambridge University Press, 2004.

[13] D. R. Lambert, "Graphics language / one - IBM Corporate-Wide physical design data format," in *Proceedings of the 18th Design Automation Conference*. Nashville, Tennessee, USA: IEEE Press, 1981, pp. 713–719.

[14] SEMI, "OASIS - open artwork system interchange standard," http://webstore.ansi.org/RecordDetail.aspx?sku=SEMI+P39-0308, 2008.

[15] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. Academic Press, Dec. 2008.

[16] M. Vetterli, J. Kovacevic, and V. K. Goyal, *The World of Fourier and Wavelets: Theory, Algorithms and Applications*. Unpublished, 2009.

[17] N. U. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*. Springer-Verlag New York, Inc., 1975.

[18] H. Sorensen and C. Burrus, "Efficient computation of the DFT with only a subset of input or output points," *Signal Processing, IEEE Transactions on*, vol. 41, no. 3, pp. 1184–1200, 1993.

[19] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *The American Mathematical Monthly*, vol. 65, no. 1, pp. 34–35, 1958.

[20] D. Skinner, "Pruning the decimation in-time FFT algorithm," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 24, no. 2, pp. 193–194, 1976.

[21] J. Markel, "FFT pruning," *Audio and Electroacoustics, IEEE Transactions on*, vol. 19, no. 4, pp. 305–311, 1971.

[22] P. Duhamel and H. Hollmann, "Split radix FFT algorithm," *Electronics Letters*, vol. 20, no. 1, pp. 14–16, 1984.

[23] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.