



Moving forward with multilayer metastructures - simple design of complex optics [Invited]

DAVID A. B. MILLER* 

Ginzton Laboratory, Stanford University, 348 Via Pueblo Mall, Stanford, CA 94305, USA

*dabm@stanford.edu

Abstract: I propose a simple and progressive way of designing and fabricating complex multilayered optics, even as we scale to large numbers of layers. The approach relies on successive layers of two-by-two blocks in which light only flows in the forward direction. It exploits a proposed trapezoidal architecture that can also support self-configuration in programmable structures, with the desired linear function or matrix set up using input vectors that correspond to the matrix rows, without any iteration other than successive single-parameter power maximizations. A key required component is a layer of 2 by 2 beamsplitters. The approach can support functions corresponding to a broad range of banded diagonal matrices. Example applications include a camera that also measures phase gradients and the direct convolution of a line of inputs with a kernel based on wavelets.

© 2026 Optica Publishing Group under the terms of the [Optica Open Access Publishing Agreement](#)

1. Introduction

Metamaterials have opened a wide range of possibilities in optics [1,2]. They can embed remarkable optical complexity in thin layers. Like complex photonics more generally, there are many potential applications in optics itself, for sensing, communicating and processing light, and possibly also in information processing through sophisticated linear functions with zero energy dissipation [3–8]. Yet, we understand that many functions cannot be performed in just one such layer [9]. Using the impressive and growing power of inverse design and other techniques [10,11], we can design complex multilayered structures for more sophisticated functionality [12]. However, designs with large numbers of custom layers of precise thickness and alignment could be very difficult to make. Inverse design may also not scale well to larger design volumes because of difficulty in simulating waves in large complex structures [11]. We face, too, the challenge that there may be limited applications for fixed complex designs that solve just one specific problem. We would therefore like some layered approach that would enable scalable design for a broad range of problems, that would lead to structures that were relatively easy and inexpensive to fabricate, and that possibly even would allow programmability or even self-configuration to the problem of interest.

Here we introduce an architecture for multilayered structures that has (i) simple progressive design and/or programming even for complex problems, (ii) simplifies manufacture by using standardized layers and avoiding sensitivity to precise layer thicknesses, and (iii) can even self-configure to problems of interest. This architecture is particularly appropriate where the number of input channels or pixels is much greater than the number of layers we can fabricate, as is naturally the case for layered metastructures. It could have immediate implementation in planar silicon photonic circuits, where Mach-Zehnder interferometer (MZI) building blocks are physically generally much longer than they are wide, leading to long, narrow circuits. It poses one challenge and opportunity – devise a layered metastructure that functions as a 2-dimensional array of 2×2 beamsplitters – though we can certainly propose various ways of doing this, at least in principle.

I introduce the architecture in Section 2 and analyze the simplest, single-layer case in Section 3, and show simple examples. Section 4 analyzes multilayer meshes and Section 5 gives an example

design and application for wavelet-based filtering. Approaches to manufacturable multilayer meshes are discussed in Section 6, some extensions are briefly discussed in Section 7, and features and challenges are discussed in Section 8 before drawing conclusions in Section 9.

2. Architectural concept

The concept of the approach is illustrated in Fig. 1. This “trapezoidal” architecture consists of a number m of layers of 2×2 blocks forming an interconnected mesh. Such blocks can be constructed as Mach-Zehnder interferometers (MZIs) with two control phases [13] – a (differential) input phase $\Delta\phi$ and a (differential) internal phase $\Delta\theta$ as in Fig. 1(a) – but other approaches are also possible, such as a simple beamsplitters with an input phase delay [14,15]. We can use the simplified notation of Fig. 1(b) for any such blocks. We presume that we have “pixelated” the input optical field into a set of separate inputs that can be fed individually into the block inputs. Such pixelation could be accomplished by an array of lenslets, for example, or we could just regard ourselves as sampling the incoming field at the points or small areas corresponding to the mesh inputs.

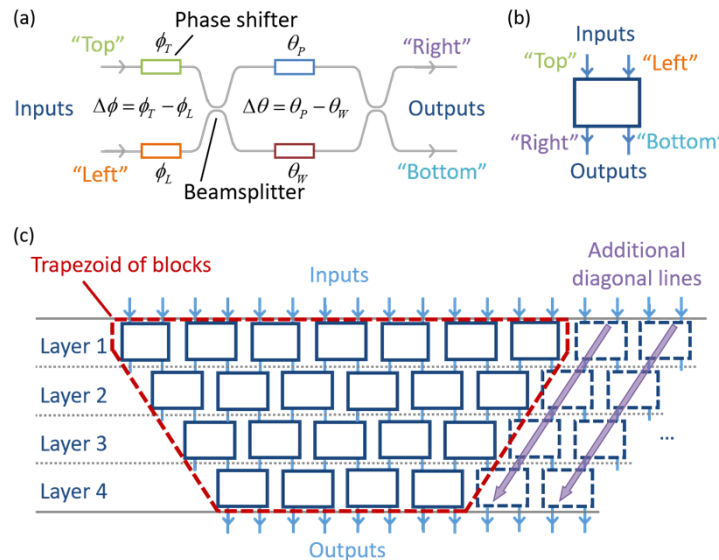


Fig. 1. Trapezoidal architecture for multilayered structure. (a) Conceptual illustration of a 2×2 block as a waveguide Mach-Zehnder interferometer controlled by two (differential) phase shifts, $\Delta\phi$ and $\Delta\theta$, with labeling of ports as in [13]. (Top, Left, Right and Bottom labels are by analogy with a cube beamsplitter [13]) (b) Simplified notation for a 2×2 block. (Note that the counterintuitive labelling of the ports here, especially “Left” and “Right”, is required since we label Inputs and Outputs below from left to right in the figures.) (c) Explicit architecture as a forward-only mesh of 2×2 blocks, in an example with 4 layers. The architecture includes a core trapezoid of blocks that can then be extended laterally by additional diagonal lines of blocks to make a layered structure of arbitrary width.

The core of the architecture is a trapezoid of such blocks as in the example for $m = 4$ layers of blocks in Fig. 1(c), which can then be extended laterally by any number of successive additional diagonal lines of blocks. With m such layers, the bottom of the core trapezoid would have m blocks, expanding to $2m - 1$ blocks at its top.

A key point of this architecture is that light only flows in one direction, here from inputs on the top to outputs on the bottom. This “forward-only” aspect [14–16] is crucial in making such a structure easy to design, program or even self-configure [13–15,17–24]. (“Forward” means from

top to bottom in Fig. 1.) It also avoids extreme sensitivity to layer thicknesses since there are no interferences forward and backwards in the structure.

One might think that restricting an architecture to being forward-only would also substantially restrict the functions it could perform; we already know, however, that forward-only architectures of sufficient depth can implement quite arbitrary linear transforms at a given wavelength [15]. Those architectures demonstrate that reflections back and forward inside a structure, which tend to happen implicitly in many layered structures, are not required for such arbitrary transforms. Reflections would be required to create spectral filters based on resonances; it has, however, recently been understood that forward-only systems can make arbitrary filters also [22,23] (at least in the finite impulse response case). Hence, the restriction of forward-only operation is arguably not a major constraint on what the device can do. Of course, especially with a limited number of layers, such a device is restricted to certain linear mappings, in particular to ones with a given overlapping nonlocality [9] (see also [Supplement 1](#) Section S1), but the same is true of any other approach to layered structures. We can also state quite explicitly the broad class of linear transforms that can be performed by our trapezoidal forward-only approach, and we return to this below.

Note, too, that, if the blocks are programmable, this architecture can be self-configured using a novel algorithm we present below in the sense that it can be physically set up by shining in just the physical vectors $|\psi_j\rangle$ of amplitudes in the set of inputs that correspond to the rows of the matrix that represents the device. (The matrix rows are formally the Hermitian adjoints $\langle\psi_j|$ of those input vectors.) If we shine in those vectors one by one, we can perform a progressive series of single-parameter power minimizations or maximizations as we adjust phase shifters (or, in non-MZI approaches, possibly also beamsplitter ratios) inside the blocks, also one by one. So, if the system has controllable phase shifters, it can be set up without calibration and can automatically compensate for minor path-length differences inside the system. The existence of such a physical configuration algorithm also means we can calibrate the system progressively [13], and we can mathematically progressively calculate the necessary settings of all the blocks for any desired design (even if the structure cannot be programmed after manufacture).

3. Analysis of single blocks and a single layer architecture

As a preliminary, we need to understand the function of an individual 2×2 block. I analyze this in detail in [Supplement 1](#) Section S2, summarizing key results here. Presuming 50:50 beamsplitters, and neglecting the overall phase in passing through the MZI (any such common-mode phase shift ϕ_{CM} can be added as an overall multiplying factor $\exp(i\phi_{CM})$ later if needed), the matrix describing the block can be written [13] as

$$M_T(\Delta\phi, \Delta\theta) = \begin{bmatrix} \exp\left(i\frac{\Delta\phi}{2}\right) \sin\left(\frac{\Delta\theta}{2}\right) & \exp\left(-i\frac{\Delta\phi}{2}\right) \cos\left(\frac{\Delta\theta}{2}\right) \\ \exp\left(i\frac{\Delta\phi}{2}\right) \cos\left(\frac{\Delta\theta}{2}\right) & -\exp\left(-i\frac{\Delta\phi}{2}\right) \sin\left(\frac{\Delta\theta}{2}\right) \end{bmatrix} \equiv \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{bmatrix} \quad (1)$$

where we have also introduced a notation $u_{i,j}$ for the matrix element in the i th row and j th column. If we write the input vector of mode amplitudes as

$$|e_{in}\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \equiv \begin{bmatrix} |a| \exp(i\alpha) \\ |b| \exp(i\beta) \end{bmatrix} \quad (2)$$

with a and b being the electric field amplitudes in the “Top” and “Left” inputs respectively, the resulting output vector $|e_{out}\rangle$ of amplitudes c and d in the “Right” and “Bottom” outputs

respectively is given by

$$|e_{out}\rangle \equiv \begin{bmatrix} c \\ d \end{bmatrix} = M_T |e_{in}\rangle \quad (3)$$

For our later discussions, it is also useful to understand how to self-configure such a block to route all the power in an arbitrary pair of inputs to a given output [13–15,17,25]. Here, with any mutually coherent relative amplitudes and phases in the “Top” and “Left” inputs, we can progressively configure the MZI to route all the input power to the “Right” output by minimizing the power out of the “Bottom” output:

first adjust the $\Delta\phi$ phase shift to minimize the “Bottom” output power

this sets the portions of the “Top” and “Left” inputs to be in antiphase at the “Bottom” output

then adjust the phase shift to minimize the “Bottom” power again

this makes the “Top” and “Left” input portions at this output have equal and opposite amplitudes that cancel completely at the “Bottom” or “drop port” output to give zero output.

If we self-configure this way, then we are setting

$$\Delta\phi = \beta - \alpha \quad (4)$$

and

$$\Delta\theta = 2\tan^{-1}(|a/b|) \quad (5)$$

When we do this, we are setting the vector $\begin{bmatrix} u_{1,1} & u_{2,1} \end{bmatrix}$ corresponding to the first row in the matrix M_T to be the Hermitian adjoint of the input vector, i.e., $u_{1,1} = a^*$, $u_{1,2} = b^*$. Hence, the self-configuration operation with some input vector $|e_{in}\rangle$ sets the first row of the matrix to be its Hermitian adjoint $\langle e_{in}|$. Equivalently, if we want to configure a mesh to represent a matrix whose first row is $\langle e_1| = \begin{bmatrix} u_{1,1} & u_{2,1} \end{bmatrix}$, then we should self-configure it with the input vector $|e_{in}\rangle = |e_1\rangle$. This concept of being able to configure any desired matrix using input vectors that are Hermitian adjoints of the rows can be regarded as a general property of self-configuring approaches. Note too that the mathematical design of this mesh to represent a given matrix just requires simple arithmetic calculations, which is also an attribute more generally of meshes that can be set up by self-configuration.

3.1. Simple example – a single layer mesh of identical blocks

We can consider the simplest possible case of this kind of architecture, which is a single layer of blocks, all set identically, as shown in Fig. 2. Though this might seem almost trivial, it could make an image sensing system that also measures angle of arrival at each pixel.

Formally, we are interested in using these blocks to detect the relative phase of the beams arriving at the two inputs to the block, with the field at the “Top” input leading that at the input labelled “Left” by some amount ϕ_a .

For this example, we set $\Delta\phi = \pi/2$ and $\Delta\theta = \pi/2$. The choice of $\Delta\theta = \pi/2$ means that $\cos(\Delta\theta/2) = \sin(\Delta\theta/2) = 1/\sqrt{2}$, so the MZI is functioning as a 50:50 beamsplitter, and

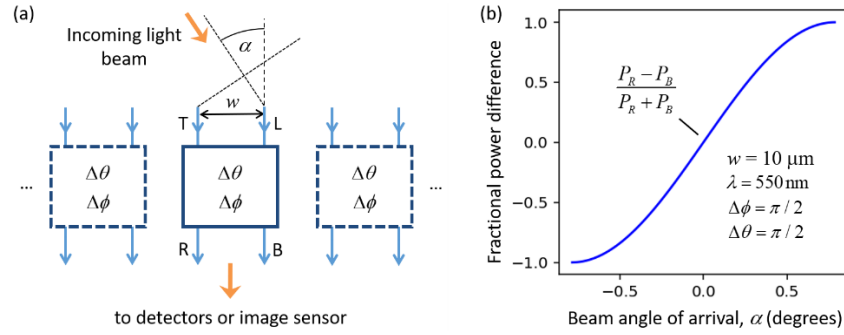


Fig. 2. (a) A single-layer mesh with blocks with identical $\Delta\phi$ and $\Delta\theta$ settings, illuminated by a beam at an angle α to the vertical. The inputs of a given block are separated by a distance w . Labels T, L, R, and B refer to the “Top”, “Left”, “Right”, and “Bottom” ports as in Fig. 1(a). The output powers landing on detectors or image sensor pixels would allow the detection angle of arrival α of the beam. (b) The fractional difference in the powers P_R and P_B emerging from the R and B block outputs as a function of the arrival angle α of the beam.

$\exp(i\Delta\phi/2) = (1/\sqrt{2})[1 + i]$. The resulting mesh matrix, from Eq. (1), is

$$M_T(\Delta\phi = \pi/2, \Delta\theta = \pi/2) = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1+i & -1+i \end{bmatrix} \quad (6)$$

Then, presuming equal powers arriving at the two inputs, the corresponding (normalized) input vector can be written as

$$|e_{in}\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} \exp(-i\phi_a/2) \\ \exp(i\phi_a/2) \end{bmatrix} \quad (7)$$

where we have split the phase delay symmetrically between the two inputs to simplify the algebra.

With this matrix M_T as in Eq. (6) and this input vector $|e_{in}\rangle$, and writing the output powers as $P_{right} = |c|^2$ and $P_{bottom} = |d|^2$ in appropriate units, the fractional difference in the powers becomes, after some algebra

$$F \equiv \frac{P_{right} - P_{bottom}}{P_{right} + P_{bottom}} = \sin(\phi_a) \quad (8)$$

So, the phase angle between the two (equal power) inputs to the block can be simply deduced from the fractional difference F in powers at the block output, giving, from Eq. (8)

$$\phi_a = \sin^{-1} F \quad (9)$$

Suppose that, for a beam whose intensity is relatively constant over the separation w between the block inputs, we record the powers at the two outputs, e.g., on adjacent pixels on an image sensor. Averaging over the two outputs of the block gives a conventional image, while the difference gives us phase gradient information that we could calculate in real time or simply save for future interpretation.

Specifically, suppose that the input to this block is a beam at an angle α to the vertical, as shown in Fig. 2, such that the phase of the wave arriving at the “Top” input leads that arriving at

the “Left” input by an amount

$$\phi_a = kw \sin \alpha \quad (10)$$

where $k = 2\pi/\lambda$ with λ as the free-space wavelength. Then, for a monochromatic beam of wavelength λ , we can use these blocks to measure the input angles of the beam at each such block. Explicitly, from Eqs. (9) and (10), we would deduce

$$\alpha = \frac{1}{kw} \sin^{-1}(\sin^{-1} F) \quad (11)$$

For example, if we choose a separation between inputs to the block of $w = 10$ microns and a wavelength of 550 nm, then the relation between beam angle and the fractional power difference is as shown in Fig. 2(b). Smaller separation w or a longer wavelength will give proportionally larger angular ranges.

Alternatively, at a fixed angle α , we could deduce the wavelength λ or frequency of locally monochromatic light from F in a similar manner. See [Supplement 1](#) section S3.

3.2. Embedded binary tree

Before proceeding to more general examples, we note that it can be straightforward to embed existing powerful and self-configuring architectures in the trapezoidal mesh. Specifically, the binary tree architecture [13,14] can be implemented just by choosing some MZI blocks to be fixed as illustrated in Fig. 3, where Layer 2 of the mesh is set in “cross” and “bar” states to perform the necessary routing to embed this version of a binary tree.

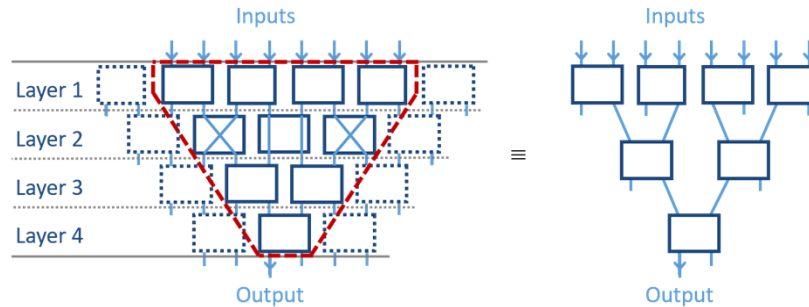


Fig. 3. Example of embedding an 8-input binary tree in a 4-layer trapezoidal mesh, using Layer 2 purely for routing in “cross” and “bar” configurations.

This 8-input binary tree architecture allows the power in any set of 8 relative amplitudes and phases as the input vector to be combined to the one chosen Output; with the Layer 2 blocks fixed as shown the necessary settings for the remaining blocks can be self-configured just by shining in the desired input vector. Running backwards by shining light backwards into the Output port allows any 8-element vector of relative amplitudes and phases to be generated [13,14]. Hence, this serves as an example of how this trapezoidal architecture is capable of some quite arbitrary linear operations. To understand the full range of operations can be performed by this trapezoidal architecture class, we proceed to a more formal analysis.

4. Analysis of multilayer meshes

4.1. Constructing the full mesh matrix from block settings

As we analyze the mesh we need the matrix that defines the linear transformation from the inputs to the outputs for the full mesh, based on the matrices for the individual 2×2 blocks. For a

mesh with N waveguides or inputs, the effect of an individual 2×2 block p between adjacent waveguides or inputs j and $j + 1$ can be written as the $N \times N$ matrix

$$M_p = \begin{matrix} & \begin{matrix} \text{outputs} \\ j \\ j+1 \end{matrix} & \begin{matrix} \text{inputs} \\ j \\ j+1 \end{matrix} \end{matrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ & & M_{Tp} & \\ \vdots & & & \ddots & 0 \\ 0 & \cdots & & 0 & 1 \end{bmatrix} \quad (12)$$

where M_{Tp} is the 2×2 matrix for the block as in Eq. (1). The matrix for the entire mesh is simply the product of the matrices for all the blocks, taken in order starting from the inputs and connected according to the mesh connections. For blocks within a given layer in the mesh as in Fig. 1(c), the matrices for individual blocks in the layer can be multiplied in any order because they do not interact with one another [26]; as a result, these can also be written as a set of blocks in the diagonal of one matrix M_{Lr} for layer r . Then we can multiply those layer matrices M_{L1} , M_{L2} , $\dots$ to get the overall matrix U_M for the mesh [26] – e.g., for the four-layer mesh of Fig. 1, we would have

$$U_M = M_{L4} M_{L3} M_{L2} M_{L1} \quad (13)$$

4.2. Setting up the mesh

We see that, if we set the mesh up by self-configuring each block to give zero at one of its outputs, we can either physically set up a programmable mesh directly by shining in vectors of interest or equivalently calculate the matrix for the entire mesh in this way using the resulting expressions Eqs. (4) and (5) and the larger matrix construction as in Eqs. (12) and (13).

For this analysis, we label the inputs, outputs, blocks and configuration vectors as in Fig. 4. We discuss this mesh as if we were self-configuring it with vectors that correspond to the various rows of the matrix it is to implement. If we can devise such a self-configuring algorithm, then it is also straightforward to calculate the required settings of the blocks because those emerge as we perform the self-configuration (physically or mathematically).

Incidentally, not all mesh architectures appear to support self-configuration. Meshes formed from one of more cascaded so-called “self-configuring layers”, such as diagonal lines or binary trees [13–15,19,20], do support self-configuration just using the vectors of interest – that is, the ones that are to be routed entirely to the output of that layer and that correspond to the Hermitian adjoints of the rows of the matrix to be implemented. Rectangular meshes [27], though they can be set up using a larger set of input vectors [25], do not appear to support self-configuration. Self-configuring layers have the topological property that there is one and only one path through the mesh layer from its output to each input [13], but a rectangular mesh has many such paths. We can see that the trapezoidal architecture also does not have this topological feature. For example, there are multiple paths from Output 1 back to Input 3, passing through blocks B32 or B31 in Layer 3, or blocks B21 or B22 on Layer 2. Hence, we might presume this network cannot be self-configured. However, as we show below, the network can still be set up using only single-parameter power maximizations or minimizations and using only the vectors that correspond to the rows of the matrix to be configured, though we may have to use those vectors

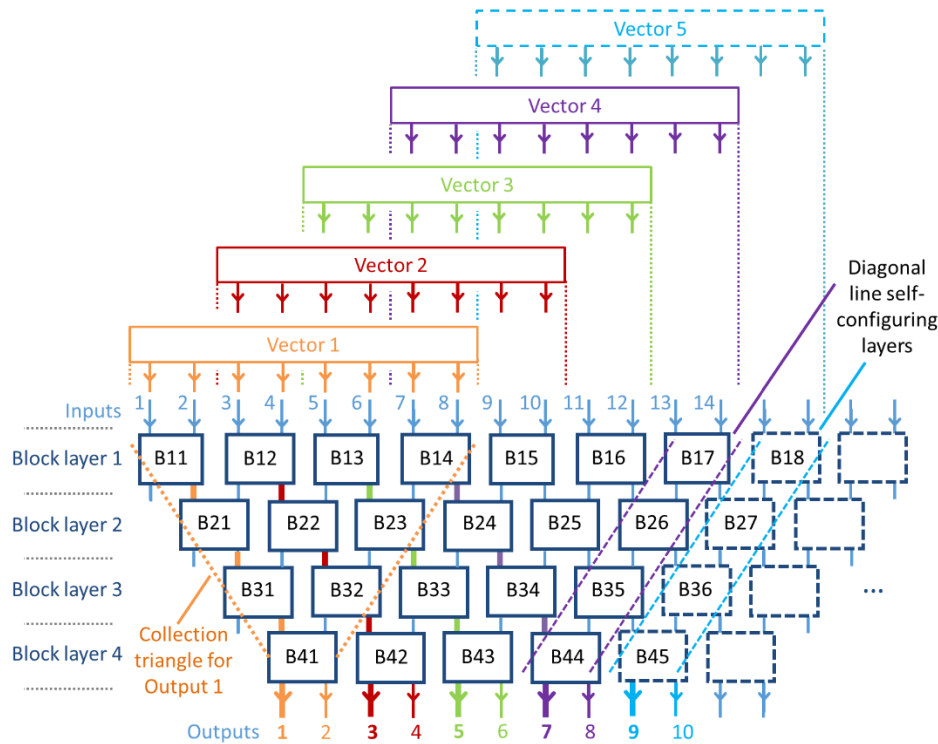


Fig. 4. Example set of 4 layers of 2×2 blocks – B11, B12, etc. – that can be programmed by shining in 8-element vectors of amplitudes – Vector 1, Vector 2, etc. – just by successive single-parameter power maximizations or minimizations. Appropriately orthogonal Vectors 1 through 4 are sufficient to set the blocks to route Vectors 1 through 4 one by one each to its corresponding Output 1, 3, 5, or 7. Because of the mesh construction, each Output can only collect light arriving within its collection triangle, as shown for Output 1. The system can be extended laterally and progressively by subsequent vectors, such as Vector 5, each orthogonal to the preceding 3 vectors. Those subsequent Vectors are each sufficient, by self-configuration of the corresponding diagonal line, similarly to route all their power to the corresponding outputs – e.g., Output 9 for Vector 5.

more than once in the algorithm. So, in this broader sense, the network can be self-configuring. Also, once the main trapezoid (as shown in Fig. 1) is set up, we will see that each additional diagonal line on the right (e.g., the one starting with block B18) is set up just by self-configuring that line with the one additional input Vector for that diagonal line.

In describing the algorithm for setting up (or calculating the settings of) the mesh, we presume or pretend that we have (mostly transparent) photodetectors at the outputs of each block so we can monitor the powers at the outputs of each block [14,15]. However, even if we only have detectors on the overall output ports, we can effectively use those for this purpose if we can temporarily set the various other blocks as needed in their “cross” state (i.e., inputs swapped to the outputs) or “bar” state (i.e., inputs straight through to the outputs) to route the block outputs temporarily as needed to detectors at the outputs of the mesh [13–15]. Finding “cross” and “bar” states of blocks within the mesh is straightforward if we can illuminate the individual mesh inputs; then simple maximization or minimization of the detected output power can readily and progressively find the required settings for “cross” or “bar” states of all the blocks.

In the example 4-layer mesh in Fig. 4, we see immediately that only the 14 numbered inputs can contribute to light at the first 8 output ports; light at other input ports has no path by which it can get to any of these 8 output ports. Also, only the 8 ports in the collection triangle for a given output can contribute to that output. So immediately we can write down the form of the matrix that maps the amplitudes at those 14 input ports to those 8 output ports, which is therefore

$$U_M = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,5} & u_{1,6} & u_{1,7} & u_{1,8} & 0 & 0 & 0 & 0 & 0 & 0 \\ u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,5} & u_{2,6} & u_{2,7} & u_{2,8} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u_{3,3} & u_{3,4} & u_{3,5} & u_{3,6} & u_{3,7} & u_{3,8} & u_{3,9} & u_{3,10} & 0 & 0 & 0 & 0 \\ 0 & 0 & u_{4,3} & u_{4,4} & u_{4,5} & u_{4,6} & u_{4,7} & u_{4,8} & u_{4,9} & u_{4,10} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & u_{5,5} & u_{5,6} & u_{5,7} & u_{5,8} & u_{5,9} & u_{5,10} & u_{5,11} & u_{5,12} & 0 & 0 \\ 0 & 0 & 0 & 0 & u_{6,5} & u_{6,6} & u_{6,7} & u_{6,8} & u_{6,9} & u_{6,10} & u_{6,11} & u_{6,12} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & u_{7,7} & u_{7,8} & u_{7,9} & u_{7,10} & u_{7,11} & u_{7,12} & u_{7,13} & u_{7,14} \\ 0 & 0 & 0 & 0 & 0 & 0 & u_{8,7} & u_{8,8} & u_{8,9} & u_{8,10} & u_{8,11} & u_{8,12} & u_{8,13} & u_{8,14} \end{bmatrix} \quad (14)$$

Note that this matrix U_M has 14 columns, each of which corresponds to one of the inputs, 1 through 14, to the first layer of blocks, and 8 rows, each of which corresponds to one of the outputs 1 through 8 of the last layer of blocks. Each row has only 8 non-zero elements, which corresponds to the fact that only those 8 inputs contribute, in the “collection triangle” of blocks in Fig. 4, to the corresponding output. We can also conveniently choose the various rows to be normalized; writing a row in Dirac notation as $\langle \psi_n |$ for the n th row of (complex) matrix elements, then normalization corresponds to $\langle \psi_n | \psi_n \rangle = 1$ (where $|\psi_n\rangle$ is the mathematical column vector that is the Hermitian adjoint (conjugate transpose) of $\langle \psi_n |$). In what follows, we show how we can set up the blocks to implement any such matrix (with orthogonal and normalized rows). For simplicity of discussion, we refer to U_M as being unitary; as a matrix it technically is not (it has different numbers of rows and columns) and can be called a semi-unitary or partially unitary matrix, though as an operator it can be rigorously unitary (See Supplement 1 Section S4).

We can immediately deduce some properties of this matrix. From the physics of the system, we see that the matrix must preserve power; the network between inputs and outputs is nominally lossless. As a consequence,

$$\text{the matrix row vectors are orthogonal to one another (i.e., } \langle \psi_n | \psi_m \rangle = \delta_{nm} \text{)} \quad (15)$$

This is because the corresponding outputs, being physically separated, are necessarily physically orthogonal, which means the input vectors corresponding to them must be orthogonal – lossless physical networks must preserve orthogonality, otherwise we could violate the second law of thermodynamics (see the Supplement 1 Section S1 of Ref. [9]).

Because of this orthogonality of the rows and their normalization, quite generally a (unit power) input vector of amplitudes $|\psi_n\rangle$ would be combined to give unit output at Output n . So, as mentioned above,

$$\begin{aligned} &\text{for an input vector of amplitudes } |\psi_n\rangle \text{ to give an output only in} \\ &\text{Output } n, \text{ the corresponding matrix row is } \langle \psi_n | \end{aligned} \quad (16)$$

We note also that, once we have set up the blocks to route to a specific output, such as the blocks in the collection triangle for Output 1 in Fig. 4, and hence having set the first row $\langle \psi_1 |$ of the matrix U_M , we have also defined exactly what the second row $\langle \psi_2 |$ of the matrix must be (at

least within an overall phase factor). So,

choosing the n th matrix row $\langle\psi_n|$, where Output n corresponds to a “left” output from a 2×2 block, determines the $(n + 1)$ row $\langle\psi_{n+1}|$ by simple calculation, (17) where Output $n + 1$ is the “right” output of the same block

We will refer to this predetermined vector $\langle\psi_{n+1}|$ as the “complement” of the corresponding vector $\langle\psi_n|$ from which it can be calculated. Explicitly, if we calculate the mesh matrix U_M , even if only for the “collection triangle” of blocks, then this complement vector $\langle\psi_{n+1}|$ is just the next $(n + 1)$ th row of the matrix, and the input vector $|\psi_{n+1}\rangle$ that would route to the $n + 1$ th output is just the Hermitian adjoint $|\psi_{n+1}\rangle$ of the $n + 1$ th row ($\langle\psi_n|$) of U_M .

To see why this complementary vector $\langle\psi_{n+1}|$ is necessarily predetermined once we have set the mesh to route $|\psi_n\rangle$ to output n , consider the case of $n = 1$, which has all the power in input Vector 1, $|\psi_1\rangle$, routed to output 1. Now we shine light backwards into output 2; the resulting vector of amplitudes $|\gamma\rangle$ that emerges backwards from the inputs is completely defined by the settings of the blocks within the collection triangle for Output 1, and those settings have already been chosen as we routed all of Vector 1 ($|\psi_1\rangle$) to Output 1. The phase conjugate of vector $|\gamma\rangle$ would be the input vector that would all be routed to Output 2 [13,25], which is, of course the vector $|\psi_2\rangle$, which is just the Hermitian adjoint of the second matrix row, $\langle\psi_2|$. So, that second matrix row is set automatically once we configure the mesh to route $|\psi_1\rangle$ all to Output 1. We could refer to these corresponding complementary input vectors as Vector 1' $\equiv |\psi_2\rangle$, Vector 2' $\equiv |\psi_4\rangle$, and so on.

This continues for row 4 of the matrix being implicitly determined once we have set row 3 of the matrix, and so on. So only every second row (that is, rows 1, 3, 5, etc.) is set independently (other than possibly an overall phase). This behavior is consistent with the rows being orthogonal; for example, matrix rows 1 and 2 must be orthogonal anyway since they would correspond to input vectors that route to different output ports.

If we add diagonal lines of blocks (as in blocks 18, 27, 36, and 45 in Fig. 4), then we just add additional pairs of rows to the bottom of this matrix (so adding 2 rows to the matrix), shifting the non-zero elements in each pair to the right by two columns (and so also increasing the number of columns by 2).

4.3. Algorithm for setting the blocks by self-configuration

Now we examine the algorithm that will allow us to configure the entire mesh based only on the input vectors corresponding to these rows 1, 3, 5, etc. of the desired mesh matrix; we do this by example for a 4 layer mesh, with the generalization for different numbers of layers then being straightforward.

Given that we want the mesh of blocks to implement some specific matrix U_M , we can consider the input vectors of amplitudes $|\psi_1\rangle$ (Vector 1), $|\psi_3\rangle$ (Vector 2), $|\psi_5\rangle$ (Vector 3), and $|\psi_7\rangle$ (Vector 4) that correspond to the desired matrix rows $\langle\psi_1|$, $\langle\psi_3|$, $\langle\psi_5|$ and $\langle\psi_7|$. Physically, these will be the input vectors whose powers are to be routed respectively to Outputs 1, 3, 5, and 7. Now we construct an algorithm for setting the blocks to achieve this, using just these input Vectors and the self-configuring algorithm for the relevant 2×2 blocks.

Note first that there is a total of 10 blocks that need to be set in routing Vector 1 to Output 1 – specifically B11 to B14, B21 to B23, B31, B32, and B41. However, the Vector 1, which has amplitudes defined by 8 complex numbers, each of which is defined by 2 real numbers, does not have enough degrees of freedom to set the 20 real parameters required for 10×2 blocks if these are to have their most general possible setting. As a result, the setting of these blocks requires the use of more than just the amplitudes of Vector 1. In the algorithm we construct, we

will also use Vectors 2, 3 and 4. Though we may have to use these Vectors more than once in the configuration, there is a progressive sequence of using these Vectors and setting specific block outputs to zero that can be used to configure the mesh completely. In this broader sense, this architecture can be self-configuring – we can set it up just using the vectors we want to be routed to specific outputs, though we may have to use a given vector more than once in the algorithm, at least in configuring this main trapezoid of blocks.

Note as in Fig. 4 that successive Vectors shine into different sets of inputs, translated successively to the right by two waveguides (or one block). Note too that these vectors must all be mathematically orthogonal to one another; otherwise they could not be routed by the mesh to separate (and hence orthogonal) outputs.

We state the algorithm based on power minimization in an output port of a block; it could also be run by power maximization in the other output port of the block. The basic trick in this algorithm is that we exploit the situations where we know the input fields for a given block must be routed to just one output of that block. The configuration of the blocks proceeds by a succession of such steps. We can view this as a physical process that configures the blocks, but we can also use it mathematically as a design process.

Because of the finite collection triangle for a given output, we can deduce immediately that, when shining in Vector 1, which is the vector of input amplitudes that is to be routed to Output 1, B11 must route all its input power to its “right” output, and similarly block B14 must route all its input power to its “left” output. (Note: here we use the usual meaning of “right” and “left”, though these correspond to “Bottom” and “Right” respectively in the formal notation of a 2×2 block in Fig. 1(a).) These kinds of operations will form the basis for the construction of the algorithm. Having done that, we can then use Vector 2 to set B12 to route all its power to its right output, and Vector 3 to set B13 to route all its power to its right output. Returning to Vector 1, we can now set block B21 to route all its power to its right output. We can continue in a similar fashion to set all the blocks in the “collection triangle” for block B41, hence routing all of Vector 1 to output 1. Proceeding in this way just with these four input Vectors, we can route all the powers in each of Vectors 1 to 4 to their corresponding single outputs 1, 3, 5, and 7, thus setting all 10 blocks (B11, B12, B13, B14, B21, B22, B23, B31, B32, and B41) in the trapezoid of Fig. 1. A detailed algorithm is given in [Supplement 1](#) section S5.

Now that we have configured this trapezoid of blocks, it is straightforward to extend this arbitrarily to the right by adding diagonal lines of blocks as in Vector 5 in Fig. 4. We can configure the corresponding diagonal line of blocks B18, B27, B36, B45 now just by progressively zeroing out the power in the right output port of each such block in sequence with Vector 5 as the input. Vector 5 must, of course, be orthogonal to Vectors 2, 3, and 4 (and it will automatically be orthogonal to Vector 1 because it does not overlap with it). We can proceed similarly for any further diagonal lines added on the right.

An alternative configuration algorithm, based only on self-configuring diagonal lines, is given in [Supplement 1](#) Section S6. In this approach, we discard the first several inputs and outputs of the mesh but get a simpler algorithm as a result.

Hence, we can self-configure a layered structure like this of arbitrary width, just using input vectors that are the Hermitian adjoints of the desired (odd-numbered) matrix rows. This gives us a general and simple design method to represent any banded matrix of the form of U_M (with orthogonal rows) or its extensions with additional columns and rows by adding diagonal lines of blocks. As mentioned, we can also use this algorithm mathematically to calculate the block settings in our design even if we never perform physical power minimizations or maximizations.

5. Example multilayer mesh – implementing a filter based on Daubechies wavelet coefficients

A good example to show the larger capabilities of the trapezoidal architecture is design one to implement a filter (here in the spatial domain) based on the coefficients associated with a Daubechies wavelet. Such wavelets are widely used in signal processing. Specifically, we can consider the db4 wavelet, which can correspond to an 8 tap finite-impulse-response filter. The decomposition wavelet coefficients for that filter (obtained, for example, through the PyWavelets [28] `pywt.Wavelet('db4').dec_hi` function) are, approximately

$$\text{db4wv} = [-0.2304, 0.7148, -0.6309, -0.028, 0.187, 0.0308, -0.0329, -0.0106] \quad (18)$$

with the corresponding decomposition scaling coefficients (through the PyWavelets `pywt.Wavelet('db4').dec_lo` function)

$$\text{db4sc} = [-0.0106, 0.0329, 0.0308, -0.187, -0.028, 0.6309, 0.7148, 0.2304] \quad (19)$$

(Note, incidentally, that the scaling coefficients are the wavelet coefficients in reverse order, with alternate signs switched.) Using the wavelet coefficients as the taps in such a filter leads to a “high pass” filter, and the scaling coefficients lead to a “low pass” filter. We can configure a four-layer mesh to implement a one-dimensional spatial filter with the db4wv as the configuration vector.

An important property of the vector db4wv (and also of db4sc) is that it is orthogonal to versions of itself shifted by any multiple of 2 inputs. Writing these vectors in an extended form (as will be appropriate as rows that are 14 “input” elements long to correspond to the trapezoidal architecture for 4 layers), the vector

$$\text{db4wv}(0) = [-0.2304, 0.7148, -0.6309, -0.028, 0.187, 0.0308, -0.0329, -0.0106, 0, 0, 0, 0, 0, 0]$$

is orthogonal to a version of itself shifted to the right by two positions, i.e., the vector

$$\text{db4wv}(2) = [0, 0, -0.2304, 0.7148, -0.6309, -0.028, 0.187, 0.0308, -0.0329, -0.0106, 0, 0, 0, 0]$$

and to the vectors db4wv(4) and db4wv(6) that are similarly shifted by 4 and 6 positions, respectively. In fact, all 4 of these vectors are orthogonal to one another. (If we were to extend these vectors by adding more zeros to correspond to the inputs to a wider trapezoid with added diagonal lines, any further such shifts by two positions would also be orthogonal because the non-zero elements would not overlap with each other.)

Hence, because of these orthogonalities, these vectors db4wv(0), db4wv(2), db4wv(4) and db4wv(6) could be used as Vectors 1 to 4 to configure the trapezoidal mesh, as in Fig. 4 above, and further such shifted versions could continue to configure a further extended trapezoidal mesh with added diagonal lines.

Because these configuration vectors are simply shifted versions of one another, as a result, all the blocks in a given layer are configured identically. We can see this if we think of the collection triangles for each of the outputs 1, 3, 5, etc. Since the vectors that configure for each of these shifted outputs are identical, so also must the settings of each such collection triangle be identical. The only way this can be the case is if all the blocks in a given layer are identical. We can prove this more mechanistically by following through the configuration algorithm. For example, the first two elements Vector 1 must lead to the output of block B11 being routed to its right output; but the first two elements of Vector 2 (which is just a shifted version of Vector 1) must similarly lead to the output of block B12 being routed to its right output, so therefore blocks B11 and B12 are set identically, and so on. The resulting mesh design to implement this db4 (spatial) filter is shown in Fig. 5.

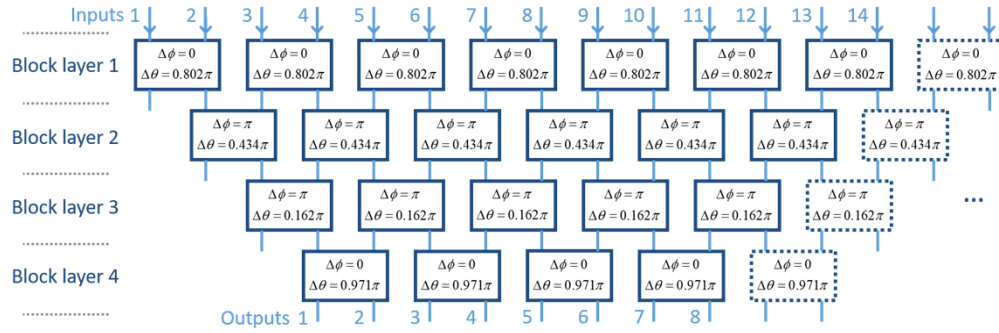


Fig. 5. Settings of blocks to implement filters based on db4 wavelet coefficients, with all blocks in layer 1 set to $\Delta\phi = 0$, $\Delta\theta \approx 0.802\pi$, all blocks in layer 2 set to $\Delta\phi = \pi$, $\Delta\theta \approx 0.434\pi$, all blocks in layer 3 set to $\Delta\phi = \pi$, $\Delta\theta \approx 0.162\pi$, and all blocks in layer 4 set to $\Delta\phi = 0$, $\Delta\theta \approx 0.971\pi$.

One intriguing point is that not only does this mesh perform a db4 wavelet-based filter to outputs 1, 3, 5, 7, etc., it also performs the corresponding db4 scaling-based filter to outputs 2, 4, 6, 8, etc. That is, the complementary vector to db4wv is actually db4sc.

The mesh matrix for the trapezoid is, explicitly (rounding to 2 figures for compactness),

$$U_M = \begin{bmatrix} -0.23 & 0.71 & -0.63 & -0.03 & 0.19 & 0.03 & -0.03 & -0.01 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.01 & 0.03 & 0.03 & -0.19 & -0.03 & 0.63 & 0.71 & 0.23 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.23 & 0.71 & -0.63 & -0.03 & 0.19 & 0.03 & -0.03 & -0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.01 & 0.03 & 0.03 & -0.19 & -0.03 & 0.63 & 0.71 & 0.23 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.23 & 0.71 & -0.63 & -0.03 & 0.19 & 0.03 & -0.03 & -0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.01 & 0.03 & 0.03 & -0.19 & -0.03 & 0.63 & 0.71 & 0.23 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.23 & 0.71 & -0.63 & -0.03 & 0.19 & 0.03 & -0.03 & -0.01 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.01 & 0.03 & 0.03 & -0.19 & -0.03 & 0.63 & 0.71 & 0.23 \end{bmatrix} \quad (20)$$

So, if we were to present some signal, such as a row of a set of image pixels, to the inputs to the mesh, the odd outputs would be that signal filtered by the db4wv kernel and the even outputs by the db4sc kernel.

If we made a wide version of this mesh by adding many diagonal lines, or equivalently had some wide stacked metastructure to implement this function, then when shining a line of inputs, such as a line of pixels from an image, onto the mesh inputs, the mesh effectively would perform the convolution of that line of inputs with these db4 coefficient kernels, with the results for db4wv and db4sc kernel appearing at alternating outputs of the mesh.

The example of the mesh to implement these db4-based filters is obviously periodic when extended laterally. It repeats as a set of diagonal lines that could be viewed as some kind of unit cell of a crystalline structure, and we could refer to this as a simple periodic mesh (with all diagonal lines the same). It is possible also to construct more complex periodic structures, for example with groups of 4 diagonal lines repeating in a larger parallelogram unit cell. We will not pursue such periodic structures further here, but the formalism for analyzing them as a kind of “forward” photonic crystal is given in the [Supplement 1](#), section S7, with associated configuration algorithms in [Supplement 1](#) sections S8 and S9.

6. Approaches to manufacturable multilayer structures

As mentioned above, this architecture could be one novel way to work with large planar silicon photonic MZI meshes. Making deep meshes with the number of MZI blocks in each layer comparable to the number of layers is challenging because the MZIs are typically much longer, e.g., by a factor of ~ 10 , than their width, so such a circuit would be ~ 10 times longer than its width. This architecture allows us to contemplate circuits with relatively smaller depth in terms of the number of layers of MZIs so that can exploit the relatively square shape of a full lithographic reticle and a large number of inputs. Such circuits could be fabricated with current technology.

More speculatively, we could contemplate separately fabricating standardized metasurfaces that could then be stacked to create the overall structure, as in Fig. 6. Such a fabrication could, of course, be challenging, but it is likely more feasible than complex metastructures with tens or more layers of subwavelength dimensions that also have to be stacked with subwavelength precision in all three dimensions [12]. While subwavelength precision in fabrication laterally within a layer is achievable because of advanced lithographic techniques, precise vertical thickness is more difficult to achieve with comparable precision. The forward-only nature of the present concept means that such vertical precision is not required because there are no reflections inside the structure between the layers that could lead to interlayer interference effects.

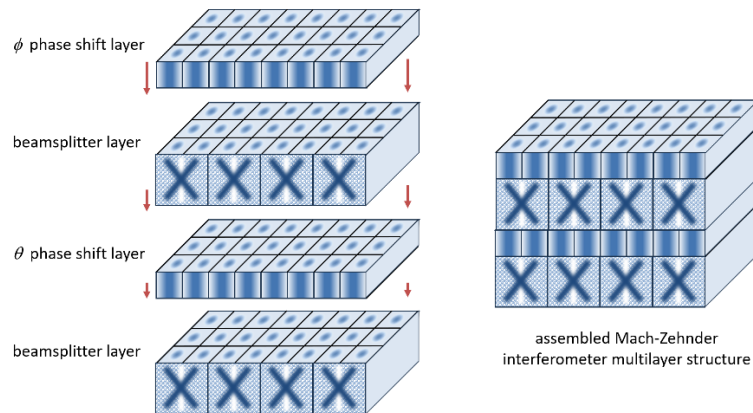


Fig. 6. Conceptual assembly of a compound layer of Mach-Zehnder interferometers by stacking phase shift layers and 50:50 beamsplitter layers.

Note too that this approach can work with standardized layers, especially for the beamsplitter layers. The design of the function of the mesh is then given entirely by the phase shift layers, which can likely be designed with some precision using metasurface techniques, even if all such phase shifts elements are to have the same overall thickness to allow a uniform layer thickness for stacking.

Other key challenges with this approach are, first, that these layers should all be designed so they have little or no reflection once stacked and, second, that the approach needs layers corresponding to 2×2 beamsplitters. Though many metasurface beamsplitters have been proposed [29], it appears that most and possibly all such work has targeted 1 to 2 (“Y”) or 1 to N splitters. Such splitters can be made with a single thin layer. However, a 2×2 beamsplitter necessarily has an overlapping nonlocality [9] of 2 (see Supplement 1 section S1) which strongly suggests that some thickness capable of supporting two channels in a given direction is required for any such design. The design of this layer of 2×2 metasurface beamsplitters appears to remain an interesting challenge. Figure 7 sketches various conceptual or actual approaches to such beamsplitters

In general with metasurfaces, if we stack them very closely, even with forward-only propagation, it is possible we could have evanescent coupling between layers, including coupling backwards,

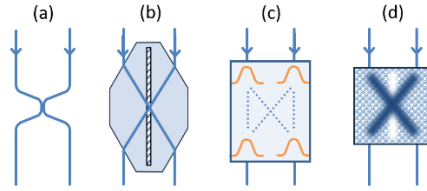


Fig. 7. Illustration of several approaches to beamsplitters for use in mesh layers. (a) A waveguide splitter operating by evanescent coupling, as in silicon photonics. (b) A hypothetical glass or dielectric beamsplitter structure with parallel inputs and outputs. (c) A multimode interference coupler. (d) A figurative notation for a beamsplitter, possibly made from some metastructure design.

which would prevent the simple “forward-only” analysis and behavior here. However, this could be easily avoided just by allowing a small space or adding a thin spacer layer between successive beamsplitter or phase shifter layers in the structure.

The approaches discussed so far, other than for planar waveguide structures, are not obviously programmable after manufacture. Even if we had materials in the phase shift layers that were themselves potentially programmable, such as some electro-optic material, there is no obvious way to get the control signals into some large number of such phase shifters or other controllable elements, especially if they are in multiple different layers of the structure.

Figure 8 shows a concept where the architecture is folded so that it can use a spatial light modulator to provide programmable phase shifts. Figure 8(a) and (b) show a single MZI block. (For a fixed, non-programmable device, the spatial light modulator could also be replaced with a set of fixed phase shifts, such as thin plates of different thicknesses.) Fig. 8(c) and (d) show a larger collection of blocks forming part of the larger architecture. If the mirror on the right is made mostly but not entirely reflecting, an image sensor or detector array on the right can monitor the powers at the outputs of the MZI blocks so as to provide the signals for self-configuring algorithms to set up or adapt the mesh. This approach would require just one 2D array of fixed 50:50 beamsplitters, so fabrication of the overall structure could be relatively straightforward. Optionally, appropriate lensing could be added to these blocks to keep the beams from diverging as they pass through the structure.

7. Extensions

7.1. Non-unitary meshes

So far, we have discussed unitary forward-only meshes. Many transformations may, however, be represented by non-unitary matrices, which will in turn require non-unitary meshes to implement them. For example, operations such as spatial differentiation are generally non-unitary. To implement non-unitary matrices, we can employ a similar strategy to that introduced in Ref. [15], which constructs the non-unitary matrix through a product $D = VD_{diag}U^\dagger$, where U (and hence also its Hermitian adjoint $U^\dagger$) and V are unitary matrices and D_{diag} is a diagonal matrix. This is constructing a non-unitary matrix through its singular-value decomposition (SVD), with the diagonal matrix elements being the (generally complex) singular values. Physically, this defines a set of orthogonal input vectors – the columns of U – that map one-by-one, with connection amplitudes given by the corresponding singular value, to the corresponding column of V , with those columns being a corresponding set of orthogonal output vectors. In our case, the resulting matrix would be a banded diagonal non-unitary matrix.

Figure 9 shows one way of implementing such a non-unitary forward-only layered mesh. Two layered meshes can each represent one of the unitary matrices $U^\dagger$ and V . The diagonal matrix D_{diag} of singular values, located between the two unitary meshes, can be implemented by a set of

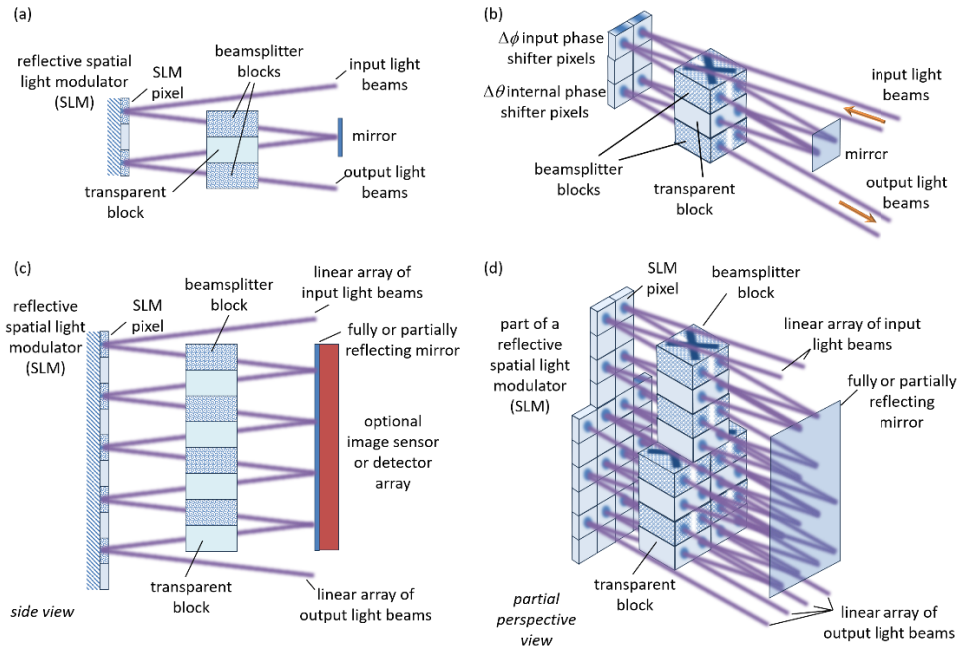


Fig. 8. Sketches of an approach to a programmable architecture, using spatial light modulator pixels as the phase shifters. (a) and (b) A stack of two beamsplitter blocks spaced by a transparent block together with two pairs of spatial light modulator (SLM) pixels form a complete MZI. (c) and (d) A linear array of light beams, corresponding to the input light beams in Fig. 1, is processed by the stack of MZIs. Only 2 layers of MZIs are shown. The perspective view shows only a part of the overall structure, corresponding to one MZI block (such as block B12) in the first layer (the upper column of blocks) feeding two MZI blocks (such as blocks B21 and B22) in the second layer. Only 2 input beams and 4 output beams out of the larger set are shown. The full structure would have a complete solid layer of MZI blocks and transparent blocks in 2D array.

attenuators, reflectors or active amplitude modulators to set the magnitude of the singular values, followed by phase shifts or active phase shifters to set their phase. If the singular-value coupling strengths are implemented using reflectors, the resulting “dropped” light amplitude is reflected back out of the top of the structure. As long as there are no other reflections in the structure, this will not compromise the “forward-only” behavior of the structure.

7.2. Spectral devices

If we illuminate the inputs of the mesh at an angle, it becomes spectrally sensitive. We briefly discussed this for the case of one block in Supplement 1 section S3. With multiple inputs in our architecture, illuminating with a plane wave at an angle gives a similar equal phase delay increment per input to that in an arrayed waveguide grating. Recently, spectral devices exploiting this set of delays at the input to an interferometer mesh have been proposed [22] and demonstrated [23]. This trapezoidal mesh could exploit many of the same kinds of programmable spectral functions just by illuminating it with a plane wave at an angle.

7.3. Two-dimensional arrays

So far, we have only considered one-dimensional arrays or collections of them. There are many possibilities for two-dimensional arrays. We show one example in Fig. 10, appropriate for a

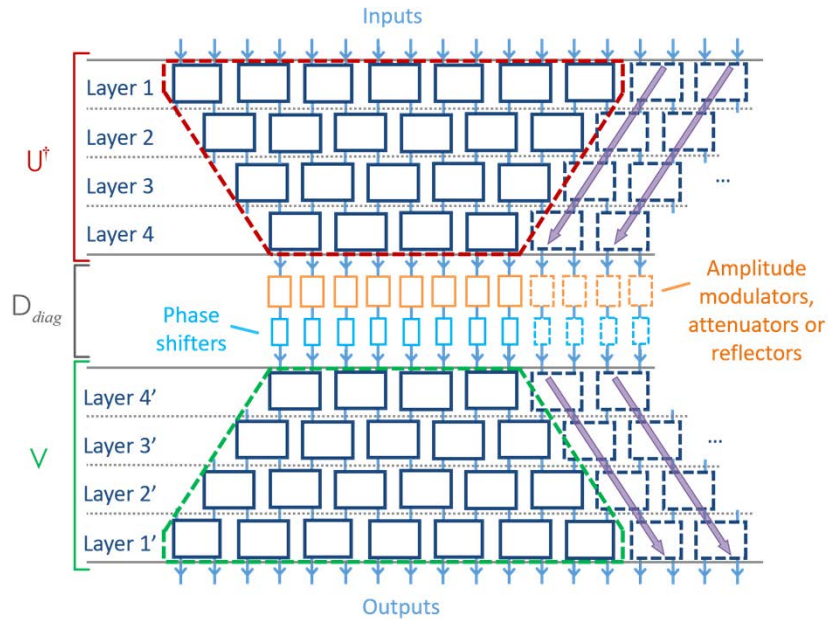


Fig. 9. Singular-value decomposition architecture for non-unitary multi-layer structures. Layers 1 to 4 give the first unitary transformation $U^\dagger$. Amplitude modulators, attenuating absorbers, or reflectors, together with phase shifters give the diagonal matrix of (complex) singular values D_{diag} . Layers 4' to 1' give the second unitary transformation V . Together these make the non-unitary matrix through its singular value decomposition form $D = VD_{diag}U^\dagger$.

single layer of blocks. This would allow the detection or measurement of phase gradients or beam angles in two orthogonal directions.

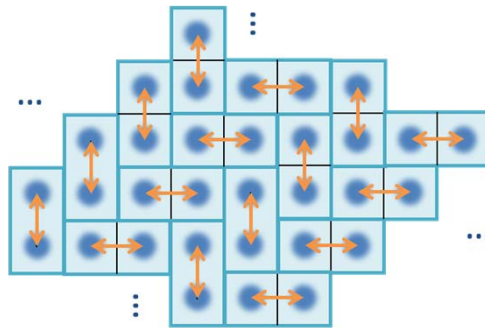


Fig. 10. Top view of the inputs to a single layer of blocks arranged in alternating “horizontal” and “vertical” orientations, as emphasized by the arrows.

8. Discussion

I have argued that there is a way of designing complex optical structures made from multiple successive layers, even for layers of arbitrary width or area. The capability, architecture and design complexity scale only proportionately with the number of layers in the structure. A key feature and requirement of this approach is that light only flows forwards in the structure as it progresses through successive layers. This forward-only approach directly avoids forward

and backward interference within the structure, hence making it relatively insensitive to precise layer thicknesses or spacings, a potentially significant advantage for manufacturing. Because it can exploit standardized layers, such individual layers could be made and functionally tested separately in advance and then stacked, further simplifying manufacture and increasing yield. Hence there is arguably a path to very complex, functional, and manufacturable multilayered optical structures, avoiding effective design volume limitations of approaches such as inverse design and the challenging fabrication of vertically and horizontally precise stacks of very thin, complex layers.

The architecture exploits layers of 2×2 interferometer blocks, and it could be applied immediately to planar silicon photonic circuits, offering a new architecture suited to wide circuits of limited architectural depth and hence compatible with the long and thin nature of programmable MZI blocks in such technologies, which restricts the number of successive layers of blocks.

An important attribute of this architecture is that it supports self-configuration. In technologies that support full programmability (e.g., of phase shifters in the blocks), this is an obvious benefit because it means that the device can be set up just using input vectors corresponding to the rows of the matrix to be implemented by the device, without calibration or calculations, and just by successive single-parameter power optimizations. Even if we cannot program the device after manufacture, the self-configuration property guarantees that, if we know what linear function (and hence matrix) we want the device to implement, we have a simple progressive technique for designing the settings of the structure without any iterations or optimization procedures; essentially, we can pretend we are self-configuring the structure, simply writing down the resulting settings as we proceed through the nominal self-configuration, block by block.

Unlike design approaches based on global optimization, where it may not be clear in advance if a desired design is possible given starting materials and sizes, we can also be quite clear just what mathematical functions can be implemented by this layered approach. Mathematically, for a structure of m successive layers of 2×2 blocks, it can implement arbitrary unitary (or, technically, semi-unitary) banded-diagonal matrices with bandwidth (number of the possibly non-zero elements in the rows) of $2m - 2$, with rows grouped in pairs (so technically a set of $2 \times (2m - 2)$ blocks on the matrix diagonal). We can also readily deduce the overlapping nonlocality [9] C that such a layered structure can support (e.g., a 4 layer structure can support C up to 4 – see [Supplement 1](#) section S1).

We have shown some examples of useful functions with such an approach. Even a single layer of such blocks could allow a camera that measures both the conventional intensity image as well as the phase tilt of the incident light. Other known highly flexible architectures such as a binary tree, which can route any 2^m element vector to a single output, can be embedded in this approach, allowing an array of such functions. Sophisticated signal processing functions such as filters based on Daubechies wavelet coefficients can be efficiently implemented, allowing parallel convolution of such a filtering function or kernel over a whole array. By illuminating at an angle, complex spectral functions can be implemented. Even a single block can be configured to measure the wavelength of an incoming monochromatic light beam.

Despite these many advantages and capabilities, significant challenges remain in applying this concept to layered metastructures. The first and arguably most important challenge is to design a metastructure that can implement an array of 2×2 beamsplitters. While there are approaches in principle to this, it is not clear if any existing design accomplishes this. Second, we need a metastructure layer of uniform thickness that can nonetheless be designed potentially to implement any of a range of phase delays in transmission (though generating such phase delays is a major strength of metasurfaces, with many possible approaches [30]). Third, all these layered components should have negligible back-reflection.

We note too that this approach requires that we take a “pixelated” approach to the optical system; we must divide the input field in this way so that we can prepare it for such an architecture

of connected blocks. Though many approaches to this are possible, such as arrays of lenslets at the inputs, this is a departure from typical metastructure design. Such a pixelation requirement and the implementation of the 2×2 functionality of the blocks might also tend to limit the minimum effective pixel size. Also, though we have argued above that the manufacture could be relatively simplified by using separately manufactured and standardized layers (such as arrays of 2×2 beamsplitters), we would still need to devise a practical technique for stacking multiple such layers. We have, however, sketched one approach that would allow the use of just one such layer of 2×2 beamsplitters together with a spatial light modulator and an optional image sensor that would allow a fully programmable and even self-configuring multiple layer structure for a linear array of inputs.

In this introduction of this architecture and associated algorithms, I have illustrated a few example designs and introduced various key concepts. There are, however, many open questions for further research, on topics that we have only been able to introduce briefly here.

- 1) With extensions as in the architecture in Fig. 9, this approach can implement non-unitary as well as unitary operations. Many important operations, such as derivatives, for example, require non-unitary operations [17].
- 2) Most of the discussion has explicitly considered only linear arrays of inputs and outputs. Many two-dimensional arrays of blocks could also be considered, of which only one simple example has been presented here (Fig. 10).
- 3) When made with repetitive collections of blocks, this approach can formally implement a kind of “forward” photonic crystal (discussed in Supplement 1 section S7). This approach may enable the simple design of a variety of such periodic structures. Possible connections to and extensions of other photonic crystal work based on constructive lateral couplings over finite distances [31–34] are intriguing for future work.

More subtly and speculatively, we can also argue that, if we can design some structure that performs a function of interest using this approach, it proves that the function is possible in principle, which might then give us the confidence to approach implementing the function by some other more highly optimized design approach, with this approach also giving us some hints as to how to do that.

9. Conclusions

I have argued that there are architectures for layered photonic structures that can allow the simple, progressive design of complex and highly functional optical structures; the design itself can be extended to high complexity because of the simplicity of the progressive calculations to design or simulate its behavior, which only require simple matrix multiplications. With some technological approaches, the desired functions could also be implemented by self-configuration; the design algorithm supports this explicitly. This simplicity does require that we pixelate the system and that we approach the design and the structures themselves in different ways from typical metastructures; this approach is based on a succession of relatively standard layers, though, importantly, existing metastructure approaches, such as inverse design, could themselves be very useful in the design of such standard layers. Hopefully, this approach encourages us that there is a broad range of highly functional complex multilayered optics that we will be able both to design and to manufacture.

Disclosures. The author declares no conflicts of interest related to this article.

Data availability. No data were generated or analyzed in the presented research.

Supplemental document. See Supplement 1 for supporting content.

References

1. A. I. Kuznetsov, M. L. Brongersma, J. Yao, *et al.*, “Roadmap for Optical Metasurfaces,” *ACS Photonics* **11**(3), 816–865 (2024).
2. S. A. Schulz, Rupert. F. Oulton, M. Kenney, *et al.*, “Roadmap on photonic metasurfaces,” *Appl. Phys. Lett.* **124**(26), 260701 (2024).
3. A. Silva, F. Monticone, G. Castaldi, *et al.*, “Performing Mathematical Operations with Metamaterials,” *Science* **343**(6167), 160–163 (2014).
4. N. Mohammadi Estakhri, B. Edwards, and N. Engheta, “Inverse-designed metastructures that solve equations,” *Science* **363**(6433), 1333–1338 (2019).
5. D. C. Tzarouchis, M. J. Mencagli, B. Edwards, *et al.*, “Mathematical operations and equation solving with reconfigurable metadevices,” *Light: Sci. Appl.* **11**(1), 263 (2022).
6. D. C. Tzarouchis, B. Edwards, and N. Engheta, “Programmable wave-based analog computing machine: a metastructure that designs metastructures,” *Nat. Commun.* **16**(1), 908 (2025).
7. A. Cordaro, B. Edwards, V. Nikkhah, *et al.*, “Solving integral equations in free space with inverse-designed ultrathin optical metagratings,” *Nat. Nanotechnol.* **18**(4), 365–372 (2023).
8. A. Overvig and A. Alù, “Diffractive Nonlocal Metasurfaces,” *Laser Photonics Rev.* **16**(8), 2100633 (2022).
9. D. A. B. Miller, “Why optics needs thickness,” *Science* **379**(6627), 41–45 (2023).
10. S. Molesky, Z. Lin, A. Y. Piggott, *et al.*, “Inverse design in nanophotonics,” *Nat. Photonics* **12**(11), 659–670 (2018).
11. L. Schul, S. Mason, S. Eun, *et al.*, “Inverse design for scalable photonic systems,” *arXiv* (2026).
12. H. Wang, W. Jin, C. Guo, *et al.*, “Design of Compact Meta-Crystal Slab for General Optical Convolution,” *ACS Photonics* **9**(4), 1358–1365 (2022).
13. D. A. B. Miller, “Analyzing and generating multimode optical fields using self-configuring networks,” *Optica* **7**(7), 794–801 (2020).
14. D. A. B. Miller, “Self-aligning universal beam coupler,” *Opt. Express* **21**(5), 6360–6370 (2013).
15. D. A. B. Miller, “Self-configuring universal linear optical component,” *Photonics Res.* **1**(1), 1–15 (2013).
16. W. Bogaerts, D. Pérez, J. Capmany, *et al.*, “Programmable photonic circuits,” *Nature* **586**(7828), 207–216 (2020).
17. D. A. B. Miller, “Perfect optics with imperfect components,” *Optica* **2**(8), 747–750 (2015).
18. A. Ribeiro, A. Ruocco, L. Vanacker, *et al.*, “Demonstration of a 4×4 -port universal linear circuit,” *Optica* **3**(12), 1348–1357 (2016).
19. A. Annoni, E. Guglielmi, M. Carminati, *et al.*, “Unscrambling light—automatically undoing strong mixing between modes,” *Light: Sci. Appl.* **6**(12), e17110 (2017).
20. S. SeyedinNavadeh, M. Milanizadeh, F. Zanetto, *et al.*, “Determining the optimal communication channels of arbitrary optical systems using integrated photonic processors,” *Nat. Photonics* **18**(2), 149–155 (2024).
21. C. Roques-Carnes, S. Fan, and D. A. B. Miller, “Measuring, processing, and generating partially coherent light with self-configuring optics,” *Light: Sci. Appl.* **13**(1), 260 (2024).
22. D. A. B. Miller, C. Roques-Carnes, C. G. Valdez, *et al.*, “Universal programmable and self-configuring optical filter,” *Optica* **12**(9), 1417–1426 (2025).
23. C. G. Valdez, A. R. Kroo, M. Vlk, *et al.*, “Programmable optical filters based on feed-forward photonic meshes,” *Light: Sci. Appl.* **15**(1), 350 (2026).
24. P.-A. Mor, A. R. Kroo, C. G. Valdez, *et al.*, “Separating partially coherent light,” *arXiv* (2026).
25. D. A. B. Miller, “Setting up meshes of interferometers; reversed local light interference method,” *Opt. Express* **25**(23), 29233–29248 (2017).
26. S. Pai, I. A. D. Williamson, T. W. Hughes, *et al.*, “Parallel Programming of an Arbitrary Feedforward Photonic Network,” *IEEE J. Sel. Top. Quantum Electron.* **26**(5), 1–13 (2020).
27. W. R. Clements, P. C. Humphreys, B. J. Metcalf, *et al.*, “Optimal design for universal multiport interferometers,” *Optica* **3**(12), 1460–1465 (2016).
28. “Wavelets — PyWavelets Documentation,” <https://pywavelets.readthedocs.io/en/latest/ref/wavelets.html>.
29. Z. Shen and D. Huang, “A Review on Metasurface Beam Splitters,” *Nanomanufacturing* **2**(4), 194–228 (2022).
30. M. Haroon, S. Kim, and D.-Y. Choi, “Phase Control Mechanisms in Metasurfaces: From Static Approaches to Active and Space–Time Modulation,” *Sensors* **26**(6), 1781 (2026).
31. Y. Chen, M. Kadic, and M. Wegener, “Roton-like acoustical dispersion relations in 3D metamaterials,” *Nat. Commun.* **12**(1), 3278 (2021).
32. Y. Chen, M. A. A. Abouelatta, K. Wang, *et al.*, “Nonlocal Cable-Network Metamaterials,” *Adv. Mater.* **35**, e2209988 (2023).
33. A. Kazemi, K. J. Deshmukh, F. Chen, *et al.*, “Drawing Dispersion Curves: Band Structure Customization via Nonlocal Phononic Crystals,” *Phys. Rev. Lett.* **131**(17), 176101 (2023).
34. M. Ciabattini and F. Monticone, “Nonlocal microwave engineering: Shaping dispersion relations and enabling pulse transformations via time-switched long-range couplings,” *Sci. Adv.* **12**(34), eaei2400 (2026).