Ikarus

High-fidelity RCWA simulation
and inverse design for metasurfaces

Dr. Liam Shelling Neto
CAVITY technologies GmbH · 2026

doi.org/10.5281/zenodo.21966455

Summary

Rigorous coupled-wave analysis (RCWA) is the natural method for metasurfaces. It solves Maxwell's equations semi-analytically on exactly the periodic, layered geometry a metasurface has, and returns diffraction efficiencies, phase, and near-fields directly, fast enough to sit inside an optimization loop.

Ikarus is built to make that power easy to reach, and to keep it honest. A forward simulation is a handful of readable lines of Python. A full inverse design is a handful more, with the solver choosing the right optimization engine on its own. Underneath the simple interface it is faithful by default: correct Fourier factorization, and convergence judged on the coefficients rather than the energy balance. The result that comes easily is also one you can trust.

This paper walks through both halves. First, why that faithfulness is harder than it looks, and how we check it against independent solvers. Then how the same tool carries a design from a first forward sweep to a fabrication-ready, inverse-designed layout.

HOW TO CITE

Shelling Neto, L. (2026). *Ikarus: High-fidelity RCWA simulation and inverse design for metasurfaces* (white paper). CAVITY technologies GmbH. doi.org/10.5281/zenodo.21966455

```
@techreport{shellingneto2026ikarus,
  author = {Shelling Neto, Liam},
  title = {Ikarus: High-fidelity RCWA simulation and inverse design for metasurfaces
     },
  institution = {CAVITY technologies GmbH},
  year = {2026},
  type = {White paper},
  doi = {10.5281/zenodo.21966455},
}
```

Author ORCID 0000-0002-0952-5570 ·

Contents

1 Why metasurface simulation is deceptively hard

A metasurface is easy to describe. It is a thin layer of subwavelength structures that reshapes a wavefront [1, 2]. Simulating one *faithfully* is a good deal harder, and not because the physics is unsettled: rigorous coupled-wave analysis (RCWA) has been a standard method for decades [3, 4]. The trouble is that several of the ways it goes wrong keep quiet about it. The simulation runs, returns a clean number, passes every obvious check, and is wrong anyway. Knowing where those failures hide is most of the work.

Three of them matter in practice.

Factorization. Real devices are built from high-index materials like silicon, TiO_2, or GaP. For those, the standard "direct" way of handling a sharp material boundary converges to the wrong answer in TM polarization. It gets there slowly, and from the wrong side, all while conserving energy perfectly, so nothing looks amiss. The next section takes this apart in full; it is the clearest example we know of a solver that looks right and isn't.

Convergence. A result is converged only once the number you care about has stopped moving as you add Fourier orders. The tempting shortcut is to check energy conservation ($R + T = 1$) instead, since it is easy and always to hand. But a lossless structure conserves energy at *every* truncation, converged or not. Leaning on energy balance is the most common way a high-contrast result gets reported with confidence and is quietly off by a few percent.

The gap to fabrication. An optimizer will exploit any idealization you leave in the model. Maybe the substrate is semi-infinite in the simulation but really a thin film on a handle wafer. Maybe the refractive index came from a table your deposition does not quite reproduce, or a feature landed one step below your minimum printable width. Each is invisible on the screen and decisive on the bench, and no tool decides for you which idealizations are safe and which will cost you.

None of these throws an error. They just produce a device that does not behave the way you drew it. Avoiding them has less to do with a more powerful simulator than with knowing where to look, and that is the point of view this paper is written from. **Ikarus**, the tool we built, encodes as much of that judgment as can be encoded. The rest we bring to the problems people ask us to solve.

2 Faithful by construction

The factorization trap is worth seeing in full. It is the clearest case of a solver that looks right and isn't, and it is the first thing **Ikarus** was built to avoid. The question is how a coupled-wave solver factorizes the product of two discontinuous quantities, the permittivity and the electric field, across a sharp material boundary. The naive approach is the "direct" or Laurent rule that many popular open-source solvers still use. In TM polarization it converges only as $\mathcal{O}(1/M)$ in the number

of retained Fourier orders M, painfully slowly and from the wrong side. **Ikarus** instead applies the inverse rule along the true local boundary normal. That is the fast-factorization work of Lalanne–Morris, Li, and Granet–Guizal [5, 6, 7], extended to curved boundaries by the normal-vector method [8].

> **LESSON FROM THE FIELD**
>
> Throughout that slow, wrong convergence, energy stays *perfectly* conserved: $R + T = 1$ to machine precision at every order. So a direct-rule tool looks converged, self-consistent, and trustworthy while it sits 60–75% wrong. Energy balance is not a convergence test. The thing that has to stop moving is the complex coefficient itself, magnitude *and* phase, and that is what **Ikarus** watches, and what it warns you about when it hasn't settled.

2.1 The convergence that matters

Take the canonical stress test: a free-standing high-index ($n = 3.5$) lamellar grating in TM polarization. Sweep the number of Fourier orders and watch the reflectance settle. The faithful methods in **Ikarus** reach the true value by $M \approx 10$. The direct rule crawls in from above and is still a quarter too high at $M = 30$ (Fig. 1). We do not get to assert that "true value" ourselves; it is what FMMax [9], a separate solver, computes independently at high truncation.

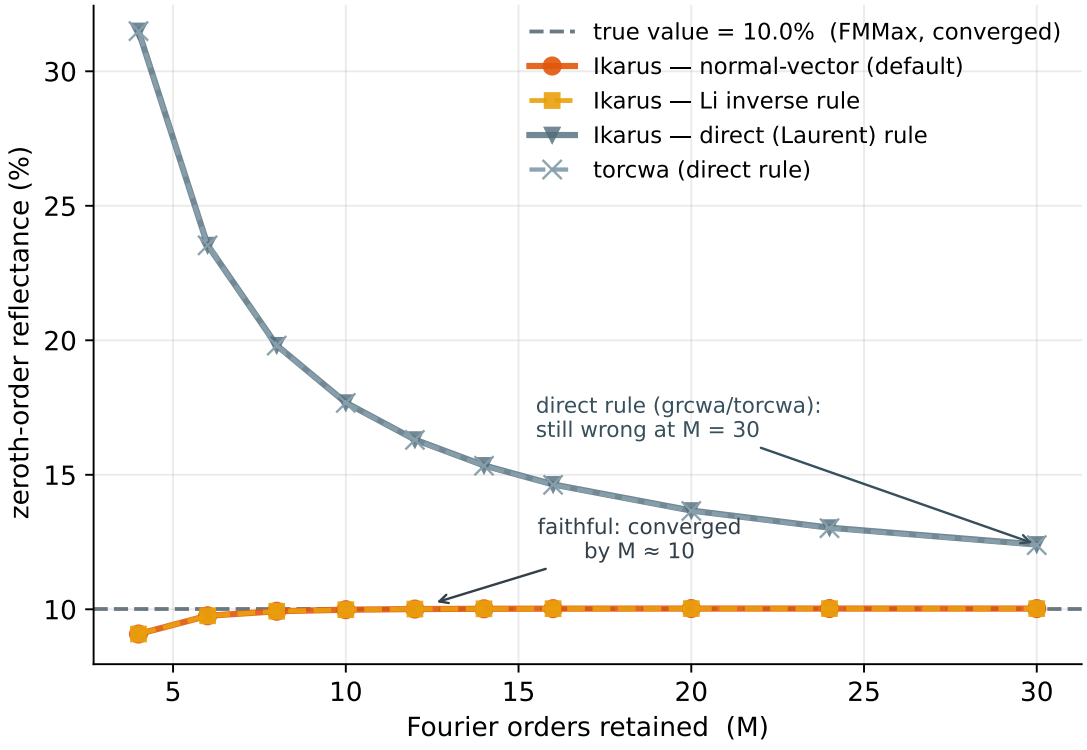

Figure 1. Faithful methods (**Ikarus**, orange/gold) snap to the true reflectance by $M \approx 10$. The direct rule crawls in from above and is still wrong at $M = 30$; here it is **Ikarus**'s own `laurent` mode and the solver `torcwa`, sitting exactly on top of one another. The dashed line is FMMax's independent value.

2.2 Checked against the codes that don't lie

Being right is a claim, so we made it falsifiable. **Ikarus** is validated *live*, in its own test suite, against three independent solvers. **FMMax** [9] is a faithful, separately implemented method, and **Ikarus** agrees with it to $\sim 2 \times 10^{-4}$ on the grating above and to $\sim 2.5 \times 10^{-3}$ on a curved two-dimensional cylinder [10], where the boundary runs oblique to every axis. **grcwa** [11] and **torcwa** [12] are two free, open-source Python solvers that use the direct rule, and **Ikarus** reproduces their $\mathcal{O}(1/M)$ error exactly. That check matters on its own. It shows the disagreement is the factorization rule, not a bug on anyone's part. Table 1 is the bottom line at a truncation a user would actually pick.

Solver	Factorization	Reflectance
FMMax (NORMAL)	faithful	10.0%
Ikarus (default)	faithful	**10.0%**
Ikarus (`laurent`)	direct rule	16.3%
torcwa	direct rule	16.3%
grcwa	direct rule	17.5%

Table 1. The same high-contrast TM grating, at a practical order count. The faithful solvers agree on the truth. The direct-rule tools miss by 60–75%, and every one of them still reports perfect energy conservation.

None of this is a static claim on a slide. It is a committed, reproducible test that runs on every change. (The cross-check solvers are optional and skip when they are not installed.) So "faithful" is something we re-earn on every commit rather than assert once. That discipline is the point, and it is the same care we put into the designs we are hired to deliver.

3 Simulation, end to end

Ikarus is meant to disappear into the work. A complete forward simulation runs to about a dozen lines: build the stack, illuminate it, read the answer. Everything past that is the same handful of objects, used more ambitiously. This section is the tool from the outside.

3.1 A whole simulation

A structure is a stack of layers, from cover to substrate. A uniform layer is just a material and a thickness. A patterned layer is a geometry plus the materials that fill it. Set a source and solve.

```
import numpy as np
from ikarus import RCWA, shapes

rcwa = RCWA(period_x=600e-9, period_y=600e-9, n_orders=(10, 10))
rcwa.add_uniform_layer(np.inf, "Air") # cover
rcwa.add_layer(400e-9, shapes.Circle(radius=0.3), ["Air", "Si"])
rcwa.add_uniform_layer(np.inf, "SiO2") # substrate

rcwa.set_source(wavelength=1100e-9, polarization="linear")
T, R, result = rcwa.simulate() # transmission FIRST
```

That is the entire forward problem. The result is not a single number. It is everything the solve produced, and you read off whichever part the question needs:

```
result.T_total, result.R_total # total power, over all orders
result.T_orders, result.orders # per order, with (p, q)
result.T_phase # zero-order phase (radians)
result.theta_out_trn # exit angle, per order
result.energy_balance # R + T, your smoke test
```

Illumination lives in the source: wavelength, incidence angle, and polarization (linear at any angle, or circular). Anything you do not set is remembered between calls, so a scan changes only what you name.

```
# 0 = TE, 90 = TM; incidence angle in degrees
rcwa.set_source(wavelength=1550e-9, theta=15, linear_pol_angle=90)
rcwa.set_source(polarization="RCP") # circular; others kept
```

3.2 Seeing the fields

Efficiencies tell you where the light went. The near-field tells you why. Ask for the field on any plane through the structure and **Ikarus** reconstructs it from the modal solution, whether that is a side view, a top view, or a slice at some chosen height.

```
from ikarus.visualization import plot_field

fields = rcwa.get_fields(plane="xz") # also "yz", "xy"
plot_field(fields["xz"], component="intensity", overlay=True)
```

Figure 2 is that call on the pillar from above, at a wavelength where it rings. The light piles up inside the silicon, and the two planes catch the same resonance from the side and from the top. `component` takes any of `Ex` … `Hz`, their phase, or the intensity, and `overlay` traces the material boundaries.

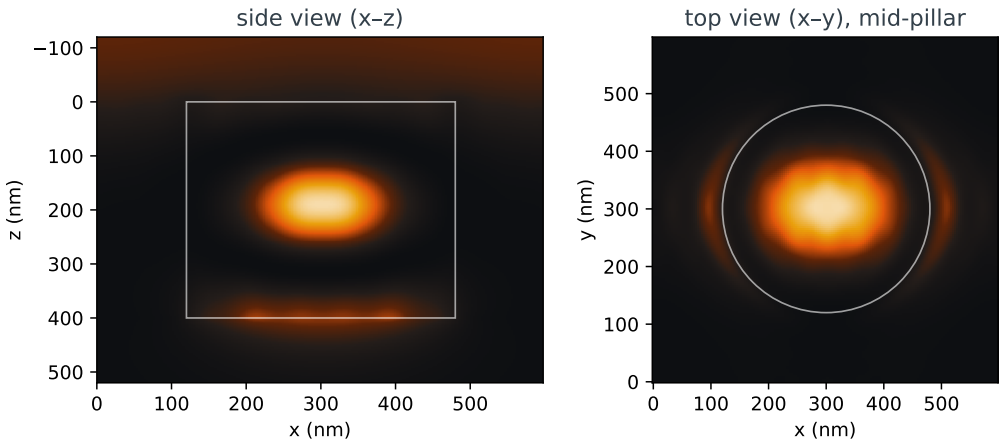

Figure 2. One field, two planes. Near-field intensity of the resonant silicon pillar from the simulation above: a side view (*x*–*z*, sliced through the pillar centre) and a top view (*x*–*y*, at mid-height), each from one `get_fields` call. At this wavelength the light concentrates inside the pillar. The white outline is the structure.

3.3 Describing the structure

The geometry can be as plain or as intricate as the device needs. A library of parametric shapes drops straight into a layer: `Circle`, `Ellipse`, `Rectangle`, `Ring`, `Cross`, `SplitRing`. Every dimension is either a number you set or a free parameter you optimize later. And a patterned layer is not limited to a single inclusion. Hand it an integer map and one material per index, and it fills each region in turn.

```
topology = np.zeros((128, 128), dtype=int)
topology[ring] = 1 # index 1 -> a TiO2 ring
topology[core] = 2 # index 2 -> a Si core inside it
rcwa.add_layer(220e-9, topology, ["Air", "TiO2", "Si"])
```

Multi-layer devices, or ones whose dimensions are tied together, become a single `Structure` that declares its parameters once (some fixed, some free, some derived from the others), says how the layers assemble, and gathers the whole stack into one object. A thickness shared between layers, or a period reused three times, is written once and honored everywhere. The same object drops straight into the optimizer of the next section.

```
from ikarus.inverse import Structure, free
from ikarus.shapes import Circle

class Resonator(Structure):
   cover, substrate = "Air", "SiO2"
   period = free(0.30e-6, 0.50e-6) # one free dimension...
   def define(self, p):
      self.add_layer(60e-9, "Si3N4") # a spacer, then
      self.add_layer(200e-9, Circle(radius=0.3), ["Air", "Si"])
```

3.4 Exploring with sweeps

Most projects start as exploration. You watch how a resonance moves as wavelength, angle, or some dimension changes, and only then commit to a design. A Sweep maps that space in one call, over any source parameters at once, and hands back arrays shaped like the axes.

```
from ikarus import Sweep

wl = np.linspace(1.4e-6, 1.7e-6, 200)
th = np.linspace(0, 30, 60)
sweep = Sweep(rcwa).over(wavelength=wl, theta=th).run()
sweep.R_total # a 200 x 60 array; also T_total, ...
```

Figure 3 is one such map: the specular reflectance of a guided-mode-resonance grating [13] across wavelength and incidence angle. The resonance traces a sharp ridge that disperses steadily with angle, climbing roughly 15 nm per degree. A single spectrum would step right over it. Twenty thousand solves went into the map, and it came back in a few minutes.

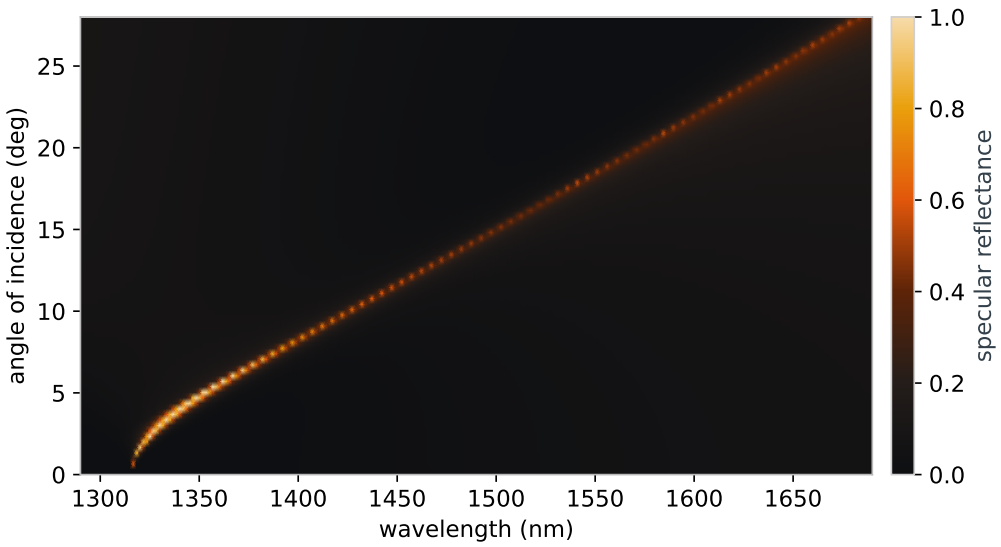

Figure 3. A design space mapped in one Sweep. Specular reflectance of a subwavelength Si_3N_4 grating on silica, over wavelength and angle of incidence. The guided-mode resonance shows up as a bright, angle-dispersing ridge. Reading sensitivity to angle and wavelength off a map like this, here 200×60 solves at real material dispersion, is the everyday work of the forward solver.

> **LESSON FROM THE FIELD**
>
> Explore coarse, then converge honestly. A two-dimensional solve scales as the sixth power of the Fourier order, so mapping a space cheaply and refining once is not a shortcut. It is the method. When it is time to trust a number, pass `auto_converge="once"` to `simulate`, and **Ikarus** raises the order count at the worst-case corner until the complex coefficient

(magnitude *and* phase) stops moving. And remember the trap from the earlier pages: $R+T=1$ holds at every truncation, so it is never the test.

4 From simulation to design

A fast, faithful forward solver is half of what a metasurface project needs. The other half is the inverse question. Not "what does this structure do?" but "what structure does what I want?" [14] **Ikarus** answers both with the same objects, and it leaves the hard decisions where they belong, with the designer rather than the optimizer.

The mechanics are deliberately small. Describe a meta-atom with a few free parameters or a freeform pixel map, state what you want as one or more targets, and call `optimize`:

```
from ikarus.inverse import MetaAtom, Target, optimize, free
from ikarus.shapes import Circle

atom = MetaAtom(period=900e-9, cover="Air", substrate="SiO2")
pillar = Circle(radius=free(0.1, 0.45))
atom.add_pattern(pillar, ["Air", "TiO2"], height=1.35e-6)

# pick the radius that gives a chosen phase:
target = Target.match("t_phase", value=1.2, at=1550e-9)
result = optimize(atom, target)
```

That is a complete inverse design. Behind it, the solver picks the engine on its own. When the design is differentiable it uses reverse-mode gradients, the adjoint method [15]; when it is not, it falls back to a genetic algorithm.

That choice matters. The gradient of a freeform objective with respect to *every* pixel costs about one extra forward solve, which makes topology optimization over thousands of degrees of freedom routine [16]; a genetic algorithm could never follow that many. For a handful of parametric knobs, discrete material choices, or a full trade-off front, the genetic algorithm is the better fit. You do not have to pick between them, though. `optimize` reads the problem and reaches for whichever one suits it.

> **LESSON FROM THE FIELD**
>
> An optimizer gives you exactly what you ask for, which is rarely what you mean. Ask for maximum *total* reflectance and it will hand you a uniform slab, a flawless mirror that is not a metasurface at all. Freeform objectives are especially prone to these degenerate answers. The discipline is to encode the property you actually care about, say power into a chosen order or a coefficient at a target phase, and to catch the trivial solution the moment it shows up.

4.1 One knob is often enough

Not every design needs that freedom. A metalens [17] is a wall of atoms, each with one job: pass the light through while imparting a chosen phase, so the wavefront bends where you want it. That is a single constraint, and it takes a single knob. Grow a dielectric pillar's radius and its transmission phase sweeps the full 0 to 2π while transmission stays high (Figure 4). Read off the radius for the phase you need, and you are done. `optimize` sees one bounded parameter and reaches for the genetic engine; a thousand freeform pixels here would be a thousand knobs for a one-knob job.

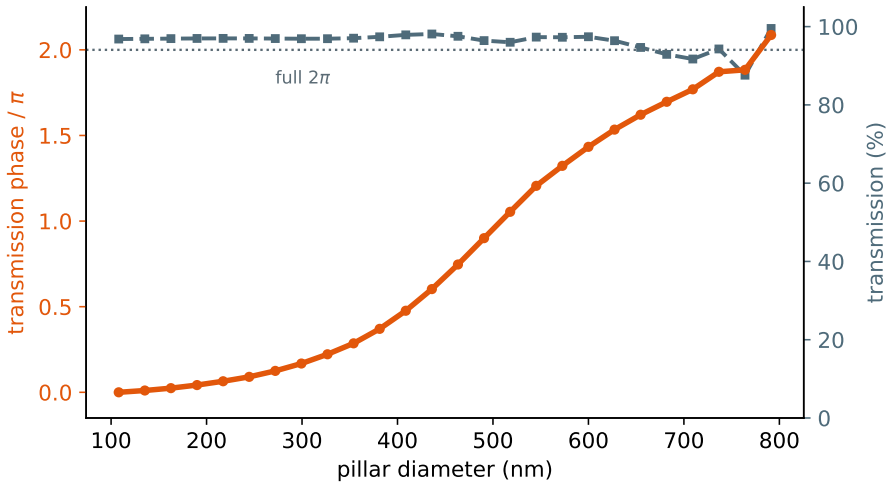

Figure 4. The parametric case, where a shape is enough. Sweeping a single TiO_2 pillar's diameter walks its transmission phase through a full 2π (orange) while transmission stays above 88% (grey, dashed). One knob covers every phase a metalens needs, and freeform freedom would be wasted on it.

4.2 Targets, and how they compose

What you can ask for is deliberately broad. A `Target` names a metric, such as power into a chosen diffraction order, a complex reflection or transmission coefficient, or a phase, and pins it at a single wavelength, averaged across a band, or worst-cased across it.

```
Target.maximize("T", order=(1, 0), at=1550e-9) # into an order
Target.match("r_co", value=1j, at=1550e-9) # a complex coeff.
Target.minimize("R", band=(1.5e-6, 1.6e-6, 20)) # over a band
```

Hand `optimize` a single target and you get one design. Hand it a *list* and you get a Pareto front, the real trade-off read straight off a curve, instead of a weighted compromise you tuned by hand.

```
targets = [
   Target.maximize("T", order=(1, 0), at=1550e-9), # peak
   Target.maximize("T", order=(1, 0), band=(1.5e-6, 1.6e-6)),
]
result = optimize(atom, targets) # a list -> a Pareto front
```

Freeform maps have their own economy. Declare a symmetry and you optimize one fundamental domain while the rest follows, which shrinks the search and enforces the symmetry exactly. A multi-layer or coupled design is just the same `Structure` from the previous section, shared and derived parameters and all, dropped straight into `optimize`.

```
from ikarus.inverse import pixels

grid = pixels(64, 64, symmetry="c4v") # optimize one octant
```

4.3 When a shape isn't enough

Freeform earns its keep when the constraints pile up and no shape has enough handles. Take a splitter that fans one incident beam into a 3×3 grid of equal-intensity spots, the diffractive heart of a structured-light depth sensor [18]. Nine diffraction orders now have to come out equal and bright at once. That is nine coupled constraints on a single surface, far more than a pillar or a cross could ever satisfy [19]. The figure of merit here, lift the weakest spot while keeping all nine even, is not one of the built-in targets, and that is exactly where **Ikarus** hands you the controls. The whole solver is differentiable. Build any real-valued figure of merit on `ikarus.grad.solve`, and `jax.grad` of it *is* the adjoint method: the gradient with respect to every pixel for roughly the cost of one extra solve.

```
import jax
from ikarus.grad import solve

def merit(rho): # rho: a 64x64 pixel density in [0, 1]
   eps = eps_air + rho * (eps_si - eps_air)
   sol = solve([eps], [height], eps_air, eps_sub, grid,
               0.0, 0.0, period, period, wl, pol)
   e = sol.T_orders[nine] # the nine 3x3 diffraction orders
   return e.min() - e.std() # lift the worst spot, keep them even

gradient = jax.grad(merit) # <- this is the adjoint method
```

Starting from random noise on that 64×64 grid, held to four-fold symmetry and lit with circular polarization so the device works whatever the input polarization, the optimizer evolves the silicon island in Figure 5. It throws 89% of the incident light into the nine spots at 74% uniformity, and leaves everything outside the 3×3 block dark. No pillar, grating, or closed-form shape does this, and no amount of

parameter-sweeping would have stumbled onto it. That is what freeform, and a differentiable solver under it, are for.

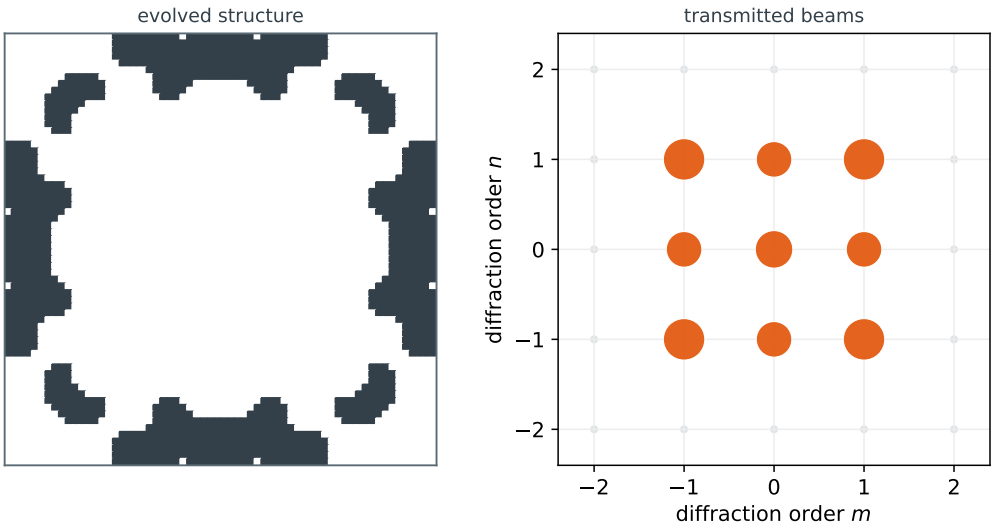

Figure 5. Freeform, where a shape is not enough. A single patterned silicon layer, inverse-designed over a 64×64 pixel grid (four-fold symmetric, from random noise) to split one beam into a 3×3 array of equal beams. The evolved structure (left) sends 89% of the light into the nine transmitted orders (right) at 74% uniformity, under polarization-insensitive circular illumination. No parametric shape reaches either that objective or that structure.

These designs are meant to survive contact with a fab. The freeform optimizer carries a minimum-feature-size filter, so a converged layout respects the smallest line your process can print instead of inventing structure you cannot make. That closes one more gap between the design on the screen and the device on the wafer.

> **LESSON FROM THE FIELD**
>
> Optimize at a modest Fourier truncation for speed, but never report the optimizer's own order count as the final number. A freeform pattern can quietly exploit an under-resolved simulation. **Ikarus** runs the check for you: pass `verify_n_orders=16` to `optimize`, and it re-simulates the finished design at that higher truncation and warns you if the efficiency is still drifting. Trust the converged number, not the one the optimizer worked with.

5 In practice

A tool earns its place in a real project through the unglamorous things: the right material data, results a colleague can reproduce months later, a way to learn it without reading the source. **Ikarus** is built for those too.

Start with the materials, since a metasurface is only as accurate as the indices you feed it. **Ikarus** ships a dispersive library (silicon, silica, titania, silicon nitride, gold, silver, and more) and also takes measured data, either as a two-column CSV or a Lorentz-oscillator model. A design can then run against the $n(\lambda)$ your process

actually deposits, not a convenient constant. Birefringent films are first-class too: hand any layer an anisotropic index and the same solver takes it in stride [20].

> **LESSON FROM THE FIELD**
>
> The most expensive error in a metasurface model is usually not the method. It is the material. An index lifted from the wrong paper, a film that came out five percent off its nominal value, an absorption tail ignored in the near-infrared: any one of these moves a resonance further than a Fourier-truncation error ever will. Simulate with the $n(\lambda)$ you measured, not the one you were hoping for, and treat a suspiciously perfect result as a question rather than an answer.

A result you cannot reproduce is not really a result. Every simulation saves to a self-describing HDF5 file and reloads to the same numbers, and the package is open source and cross-checked against three independent solvers on every change, as the earlier pages showed. When the built-in machinery runs out, as it did for the beam splitter, the solver underneath is still differentiable, so a figure of merit you write by hand is gradient-optimized rather than left to a slow search. The idea is that you extend the tool, not work around it.

None of it has to be learned from the source. The documentation[1] takes a new user from a five-minute first simulation, through a hands-on tutorial series we call Flight School, to a full API reference. There is even a companion page written for AI assistants, so a teammate's chatbot reaches for the real API instead of inventing one. A new hire, or their tools, can be productive the same afternoon.

6 Work with us

Ikarus is free and open source, and it is yours to use. It was never the product, though. It is more a look at how we work.

We are CAVITY technologies GmbH, and we build cavity-based laser systems. Part of that job is rethinking the mirrors a cavity is built from. We design metamirrors, engineered surface by surface, to stand in for conventional optics, and that turns out to be metasurface design under about the least forgiving constraints there are: near-perfect reflectivity, an exact phase, and a layout a fab can actually make.

That work does not stop at cavities. The same methods shape a lens, a filter, a beam-splitter, or a polarizer, and they go hand in hand with machine-learning-driven inverse design, which is the background we come from. **Ikarus** is a big part of that know-how, turned into software you can actually run.

So please use **Ikarus** for your own work. And if a metasurface or an inverse-design problem ever grows past what you want to take on alone, whether in an optical cavity or anywhere else light meets structure, we would be glad to hear from you.

[1] cavitytechnologies.github.io/ikarus

CAVITY technologies GmbH

www.cavity-technologies.com • github.com/CAVITYtechnologies/ikarus

Software DOI: 10.5281/zenodo.21744163

References

[1] N. Yu, P. Genevet, M. A. Kats, F. Aieta, J.-P. Tetienne, F. Capasso, and Z. Gaburro, "Light propagation with phase discontinuities: generalized laws of reflection and refraction," *Science* **334**, 333–337 (2011). doi:10.1126/science.1210713

[2] N. Yu and F. Capasso, "Flat optics with designer metasurfaces," *Nat. Mater.* **13**, 139–150 (2014). doi:10.1038/nmat3839

[3] M. G. Moharam and T. K. Gaylord, "Rigorous coupled-wave analysis of planar-grating diffraction," *J. Opt. Soc. Am.* **71**, 811–818 (1981). doi:10.1364/JOSA.71.000811

[4] M. G. Moharam, E. B. Grann, D. A. Pommet, and T. K. Gaylord, "Formulation for stable and efficient implementation of the rigorous coupled-wave analysis of binary gratings," *J. Opt. Soc. Am. A* **12**, 1068–1076 (1995). doi:10.1364/JOSAA.12.001068

[5] P. Lalanne and G. M. Morris, "Highly improved convergence of the coupled-wave method for TM polarization," *J. Opt. Soc. Am. A* **13**, 779–784 (1996). doi:10.1364/JOSAA.13.000779

[6] L. Li, "Use of Fourier series in the analysis of discontinuous periodic structures," *J. Opt. Soc. Am. A* **13**, 1870–1876 (1996). doi:10.1364/JOSAA.13.001870

[7] G. Granet and B. Guizal, "Efficient implementation of the coupled-wave method for metallic lamellar gratings in TM polarization," *J. Opt. Soc. Am. A* **13**, 1019–1023 (1996). doi:10.1364/JOSAA.13.001019

[8] T. Schuster, J. Ruoff, N. Kerwien, S. Rafler, and W. Osten, "Normal vector method for convergence improvement using the RCWA for crossed gratings," *J. Opt. Soc. Am. A* **24**, 2880–2890 (2007). doi:10.1364/JOSAA.24.002880

[9] M. F. Schubert and A. M. Hammond, "Fourier modal method for inverse design of metasurface-enhanced micro-LEDs," *Opt. Express* **31**, 42945–42960 (2023). doi:10.1364/OE.503481

[10] L. Li, "New formulation of the Fourier modal method for crossed surface-relief gratings," *J. Opt. Soc. Am. A* **14**, 2758–2767 (1997). doi:10.1364/JOSAA.14.002758

[11] W. Jin, W. Li, M. Orenstein, and S. Fan, "Inverse design of lightweight broadband reflector for relativistic lightsail propulsion," *ACS Photonics* **7**, 2350–2355 (2020). doi:10.1021/acsphotonics.0c00768

[12] C. Kim and B. Lee, "TORCWA: GPU-accelerated Fourier modal method and gradient-based optimization for metasurface design," *Comput. Phys. Commun.* **282**, 108552 (2023). doi:10.1016/j.cpc.2022.108552

[13] S. S. Wang and R. Magnusson, "Theory and applications of guided-mode resonance filters," *Appl. Opt.* **32**, 2606–2613 (1993). doi:10.1364/AO.32.002606

[14] S. Molesky, Z. Lin, A. Y. Piggott, W. Jin, J. Vučković, and A. W. Rodriguez, "Inverse design in nanophotonics," *Nat. Photonics* **12**, 659–670 (2018). doi:10.1038/s41566-018-0246-9

[15] T. W. Hughes, M. Minkov, I. A. D. Williamson, and S. Fan, "Adjoint method and inverse design for nonlinear nanophotonic devices," *ACS Photonics* **5**, 4781–4787 (2018). doi:10.1021/acsphotonics.8b01522

[16] J. S. Jensen and O. Sigmund, "Topology optimization for nano-photonics," *Laser Photonics Rev.* **5**, 308–321 (2011). doi:10.1002/lpor.201000014

[17] M. Khorasaninejad, W. T. Chen, R. C. Devlin, J. Oh, A. Y. Zhu, and F. Capasso, "Metalenses at visible wavelengths: diffraction-limited focusing and subwavelength resolution imaging," *Science* **352**, 1190–1194 (2016). doi:10.1126/science.aaf6644

[18] H. Dammann and K. Görtler, "High-efficiency in-line multiple imaging by means of multiple phase holograms," *Opt. Commun.* **3**, 312–315 (1971). doi:10.1016/0030-4018(71)90095-2

[19] D. Sell, J. Yang, S. Doshay, R. Yang, and J. A. Fan, "Large-angle, multifunctional metagratings based on freeform multimode geometries," *Nano Lett.* **17**, 3752–3757 (2017). doi:10.1021/acs.nanolett.7b01082

[20] E. Popov and M. Nevière, "Maxwell equations in Fourier space: fast-converging formulation for diffraction by arbitrary shaped, periodic, anisotropic media," *J. Opt. Soc. Am. A* **18**, 2886–2894 (2001). doi:10.1364/JOSAA.18.002886