Python in Nanophotonics Research

The authors describe how they use Python for nanophotonics research—specifically, they describe using it for electromagnetic modeling, mask design, and process simulation.

In our photonics research group at Ghent University, we're very active in the field of *nanophotonics*, which takes a complex optical system the size of a large table and shrinks it so that it fits onto a photonic chip of a few mm^2. Miniaturization and integration have done wonders for electronics in the past few decades, and the hope is that a similar strategy can work for photonics, too, leading to highly performant optical chips for fields as diverse as high-speed telecommunications, optical computing, and biosensors.

The electronics industry has spent billions of dollars perfecting fabrication technology in silicon, so it seems like a smart idea to piggyback on their mature processes for photonics applications. This is why our group is working on photonic structures made in silicon-on-insulator, which we're fabricating with the same state-of-the-art deep-UV lithography that produced the latest computer chip. These structures are typically less than a micron in size and can guide light along narrow waveguides, make very tight bends, or squeeze light into extremely small volumes.

In getting to a working device, however, we face several challenges and rely heavily on Python to overcome them.

Challenges and Advantages

A first step for any photonics research involves figuring out how light behaves in complicated structures. For our particular studies, we use an in-house-developed Maxwell solver (http://camfr.sourceforge.net). Its core is written in C++, and it uses a few legacy Fortran routines, but its interaction with our simulator (to define the structure to be simulated, the quantities to be calculated, and so forth) happens via Python scripts, glued to C++ via Boost.Python. This C++ library makes it easy to expose C++ code to Python, is very powerful, and provides support for advanced C++ options. Its drawback, though, is that compilation times and memory requirements can be quite heavy. Figure 1 shows an example of a nanolaser's optical field, as calculated with our electromagnetics solver CAMFR.

Using Python

Once we come up with a good design, we still have to fabricate it, which involves designing a mask. Python scripts can help us define these masks: because Python is a full-fledged programming language, it's easy to parameterize the design or create repetitive structures using loops. Once the mask is finished, we place it in a deep-UV stepper to project the design on a photosensitive resist spun on a silicon wafer. Unfortunately, the pattern that ends up on the wafer isn't the same as the one on the mask, due to the projected light's diffraction, peculiarities in the etching process, and so forth. To get around this, we used NumPy or SciPy to write a process simulator that can calculate various effects.

As Figure 2 illustrates, Python can take us from electromagnetic design to mask layout and process technology simulation. This ability also lets us close the loop and, for example, recalculate the electromagnetic properties of the actual resulting geometry as predicted by the technology simula-

1521-9615/07/$25.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

PETER BIENSTMAN, LIEVEN VANHOLME, WIM BOGAERTS, PIETER DUMON, AND PETER VANDERSTEEGEN
Ghent University

tor, compare it to the nominal design, and make some modifications and precorrections, if needed.

All Python tools have a single aspect in common: they must be able to handle a structural definition (in terms of geometric primitives). For our research, we designed a generic class library that deals with geometric prototypes and creates a "little" language on top of Python to define a structure (such as the one in Figure 3):

```
air = Material(1) # Air has a refrac-
tive index of 1.
mat = Material(3) # Our material has a
refractive index of 3.

g = Geometry(air) # Air is the back-
ground material.

# Now add some shapes.

g += Rectangle(Point(0.0, 1.0),
Point(2.0, 2.0), mat)
g += Triangle (Point(2.0, 2.0),
Point(2.0, 1.0), Point(3.0, 1.5), mat)
g += Circle   (Point(4.5, 1.5), 0.5, mat)
```

Although we could probably implement similar approaches in different languages, Python's increased productivity makes it a very attractive option for us.

We're currently extending and formalizing the definition of our "little" language, such that it will be powerful enough to use as input for our Maxwell solver and to automatically generate a mask description from it. We also plan to write a wrapper around other third-party simulation software, such that these generic structure definitions can be used as inputs for a wide variety of tools.

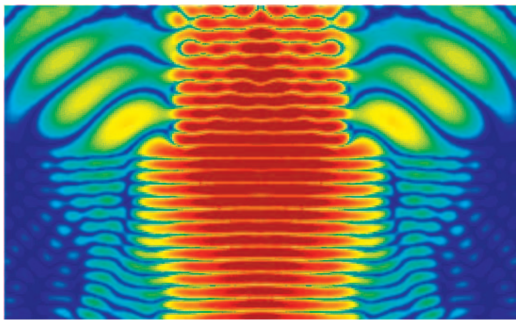

Figure 1. Optical field in a nanolaser, as calculated by our electromagnetics solver CAMFR.

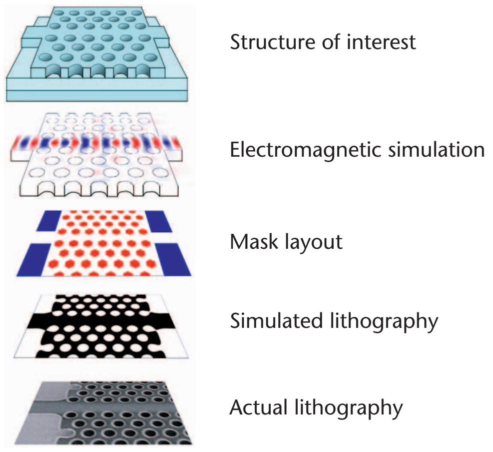

Figure 2. Python in the process.

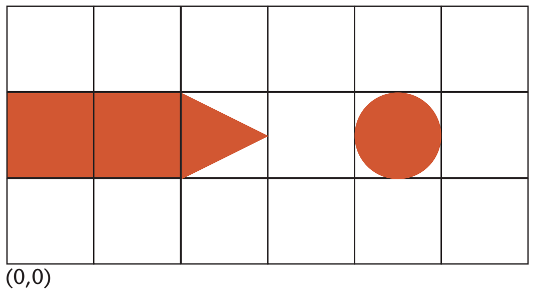

Figure 3. Example of a simple geometry.

Peter Bienstman *is an associate professor at Ghent University, Belgium. His research interests include nanophotonics and scientific computing. Contact him at Peter.Bienstman@UGent.be.*

Lieven Vanholme *works in the Photonics Research Group at Ghent University. His research interests include programming and physics. Contact him at Lieven.Vanholme@UGent.be.*

Wim Bogaerts *is a postdoc in the photonics group at Ghent University. His research interests include silicon nanophotonics. Contact him at Wim.Bogaerts@UGent.be.*

Pieter Dumon *is a PhD student in electronic engineering at Ghent University. His research interests include simulation and fabrication of silicon nanophotonic components. Contact him at Pieter.Dumon@UGent.be.*

Peter Vandersteegen *is a PhD student in the photonics group at Ghent University. His research focuses on organic LEDs and simulation methods. Contact him at Peter.Vandersteegen@UGent.be.*