PyWaveGuide

Technical Reference Manual

Avram Alexandru-Valentin

January 3, 2026

Abstract

This document details the architecture, theoretical foundations, and algorithmic implementation of **PyWaveGuide**, an open-source Python software suite designed for the simulation and analysis of integrated optical components. The application bridges the gap between analytical approximation methods and rigorous numerical simulations, offering a hybrid approach for rapid photonic prototyping. Key features include a vectorized Finite-Difference Time-Domain (FDTD) engine, dynamic geometry generation for Multimode Interference (MMI) couplers, and a comprehensive material database.

Contents

1 System Architecture

PyWaveGuide adopts a modular architecture that separates the user interface, business logic, and computational kernels. This design ensures maintainability and allows for the independent scaling of the simulation engine.

1.1 Architectural Components

- **GUI Layer (View - `gui_app.py`):** Built on `Tkinter`, providing a zero-dependency, cross-platform interface. It handles event-driven user interactions and renders real-time schematic previews of optical components using the `Canvas` API.
- **Orchestration Layer (Controller - `optimizer.py`):** Implements the *Factory Method* pattern to dynamically instantiate component models. It acts as a bridge between the raw physics data and the high-level application logic, formatting results for the comparative datasheets.
- **Physics Core (Model - `waveguide_models.py`):** Contains the analytical implementations for waveguide modes, bend losses, and interference patterns. It follows an Object-Oriented approach where every component (e.g., `SBend`, `YBranch`) inherits from a `GenericComponent` base class.
- **Simulation Engine (Kernel - `fdtd_sim.py`):** A standalone high-performance computing module utilizing `NumPy` for tensor operations. It is decoupled from the GUI, allowing it to potentially run as a background process or on a remote server.

2 Theoretical Framework

2.1 Waveguide Mode Theory

The software determines the operational regime (single-mode vs. multi-mode) using the normalized frequency parameter (V-number) for step-index waveguides:

$$V = \frac{2\pi a}{\lambda}\text{NA} \approx \frac{\pi w}{\lambda}\sqrt{n_{core}^2 - n_{clad}^2} \tag{1}$$

Where w is the waveguide width and λ is the vacuum wavelength. The cut-off condition for the fundamental mode is $V < 2.405$.

2.2 Multimode Interference (MMI)

The MMI couplers are designed based on the self-imaging principle. The beat length L_π, defined as the propagation distance where the fundamental and first-order modes acquire a phase difference of π, is approximated by:

$$L_\pi \approx \frac{4n_{eff}W_e^2}{3\lambda} \tag{2}$$

For an N-port splitter, the optimal device length is derived as $L_{opt} = L_\pi/N$. This analytical model allows for instant dimensioning before numerical verification.

3 Numerical Implementation (FDTD)

The core simulation engine employs the Finite-Difference Time-Domain (FDTD) method to solve Maxwell's curl equations in a 2D Transverse Magnetic (TM) grid (E_z, H_x, H_y).

3.1 The Yee Algorithm

The continuous space-time derivatives are discretized using central differences. The update equations for the TM mode are:

Magnetic Field Update (H at $n + 1/2$):

$$H_x|_{i,j}^{n+1/2} = H_x|_{i,j}^{n-1/2} - \frac{\Delta t}{\mu \Delta y} \left(E_z|_{i,j+1}^{n} - E_z|_{i,j}^{n} \right) \tag{3}$$

$$H_y|_{i,j}^{n+1/2} = H_y|_{i,j}^{n-1/2} + \frac{\Delta t}{\mu \Delta x} \left(E_z|_{i+1,j}^{n} - E_z|_{i,j}^{n} \right) \tag{4}$$

Electric Field Update (E at $n + 1$):

$$E_z|_{i,j}^{n+1} = E_z|_{i,j}^{n} + \frac{\Delta t}{\epsilon_{i,j}} \left(\frac{H_y|_{i,j}^{n+1/2} - H_y|_{i-1,j}^{n+1/2}}{\Delta x} - \frac{H_x|_{i,j}^{n+1/2} - H_x|_{i,j-1}^{n+1/2}}{\Delta y} \right) \tag{5}$$

3.2 Vectorization Performance

To overcome the interpretation overhead of Python loops, the update equations are fully vectorized using `NumPy` slicing. This allows the operations to be executed in optimized C-level routines.

```
# Efficient grid update without explicit loops
Hx[:, :-1] -= 0.5 * (Ez[:, 1:] - Ez[:, :-1])
Hy[:-1, :] += 0.5 * (Ez[1:, :] - Ez[:-1, :])
Ez[1:, 1:] += C_inv[1:, 1:] * (
    (Hy[1:, 1:] - Hy[:-1, 1:]) -
    (Hx[1:, 1:] - Hx[1:, :-1])
)
```

Listing 1: Vectorized FDTD Update Kernel

3.3 Procedural Geometry Construction

Instead of loading static mesh files, PyWaveGuide generates simulation grids procedurally.

- **S-Bends:** Modeled using a cosine function $y(x) \propto (1 - \cos(\pi u))$ to minimize mode mismatch loss (adiabatic transition).
- **Material Mapping:** The permittivity matrix $\epsilon(x, y)$ is populated dynamically based on the selected material properties ($n_{Si} \approx 3.47$, $n_{SiO2} \approx 1.44$).

4 Dependencies

Library	Usage Context
`NumPy`	Matrix manipulation, tensor math, FDTD vectorization.
`Matplotlib`	Field visualization, animation rendering, spectral plotting.
`Tkinter`	Graphical User Interface (Standard Python Library).

Table 1: Software Dependencies

5 Conclusion

PyWaveGuide successfully demonstrates the integration of analytical engineering models with numerical physics simulations. It serves as a rapid prototyping tool, allowing designers to validate geometric parameters and estimate insertion losses before proceeding to commercial-grade solvers.