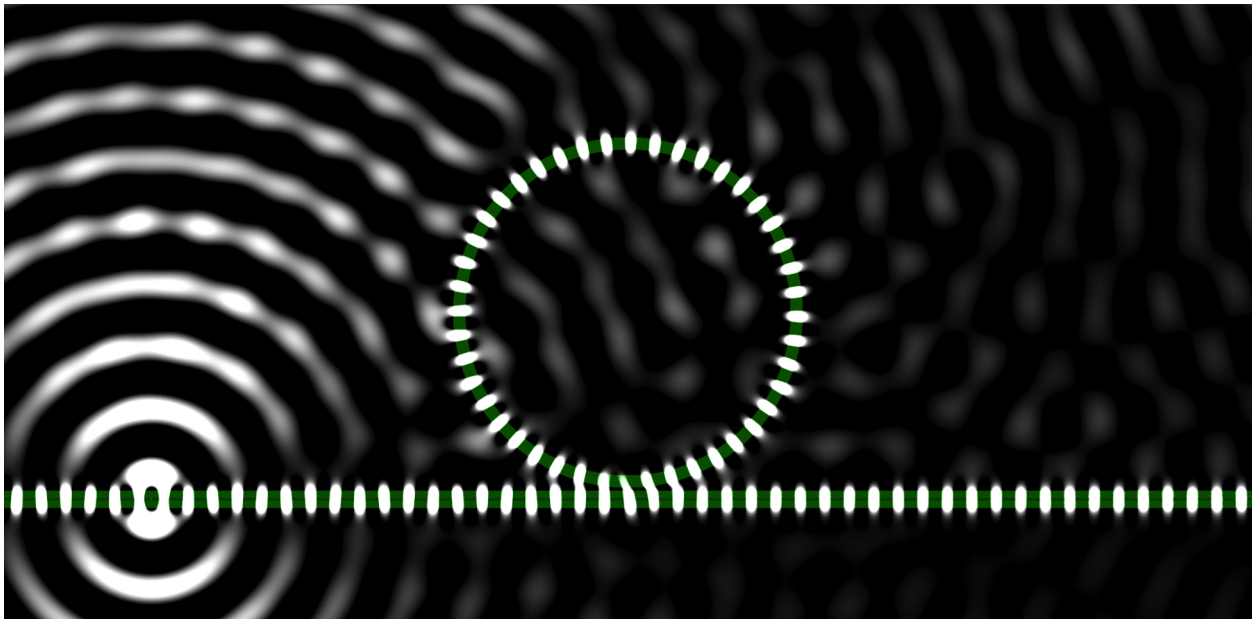
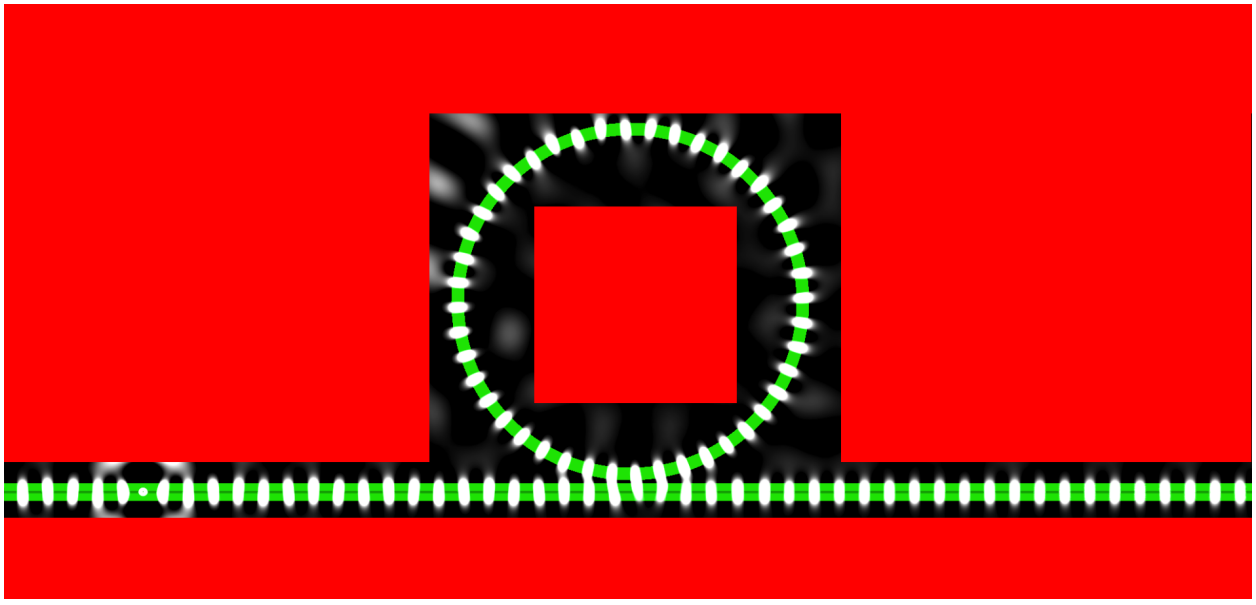
David Lively
SMU ID# 2252-3817
29 May 2022
EE 7390

PML Sinks

FDTD simulation provides a powerful method of simulating complex optical circuits. However, such simulations require substantial processing power. Approaches such as GPU-based simulation and coordinate stretching can increase performance. However, they ignore a significant performance bottleneck.



In the image above, features of interest are limited to the area surrounding the linear waveguide, its interface to the spherical WGM sensor, and the sensor itself. The majority of the domain exists simply to contain the WGM and linear guide but ultimately provides no information of interest. This presents some interesting opportunities for optimization.



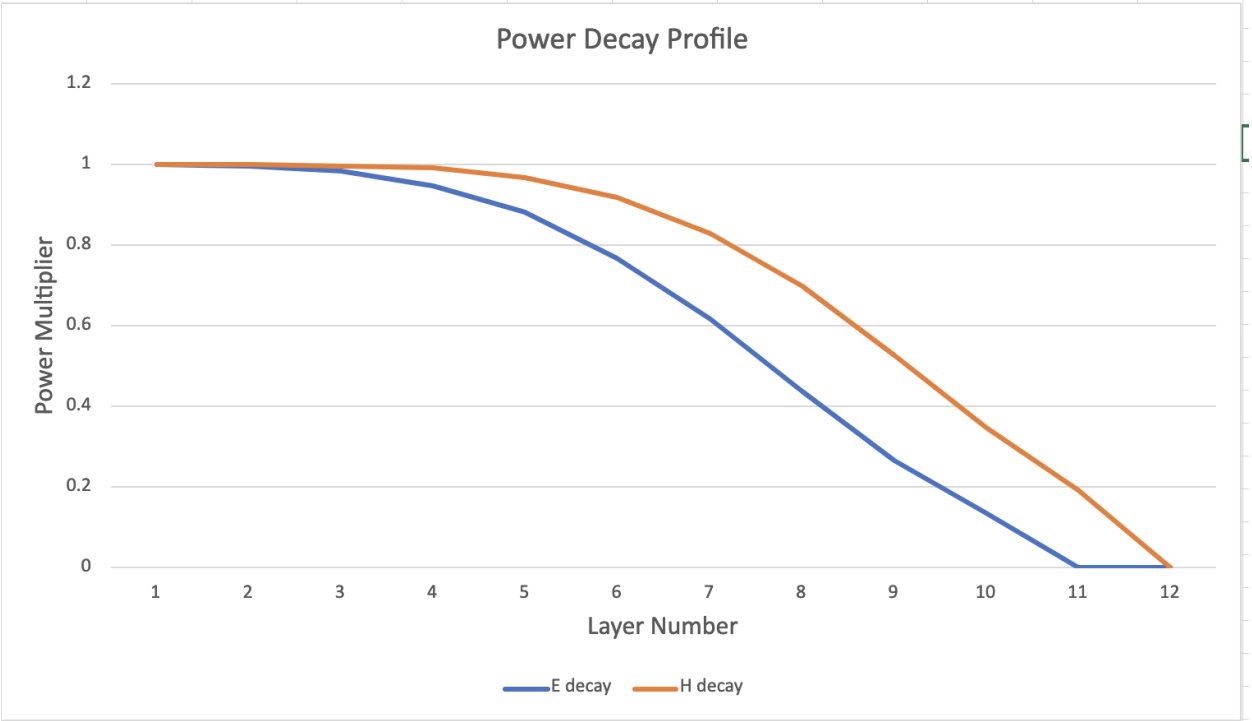
In this image, uninteresting domain regions are shown in red. Although the rectangular simulation domain must be large enough to contain both the linear waveguide and the circular whispering gallery mode sensor, the majority of the domain provides no useful output. Simulation throughput could be greatly improved by only performing FDTD update calculations in areas that contribute to outputs of interest.

However, we cannot simply ignore the cells in the indicated region as incident waves would reflect from the interface to the uncomputed region. This is the same behavior presented by a simulation with Direchlet boundaries. In order to simulate a finite domain that is part of an infinite space, we typically apply a Perfectly-Matched Layer (PML) technique which absorbs waves as they encounter the edge of a domain. This method may be adapted to contain arbitrarily-shaped domains.

Here, we propose a method for implementing PML “sinks,” or regions within the larger domain which absorb power to reduce the number of Yee cells that must be updated to maintain simulation fidelity. This application reduces the number of discrete cells within a domain that must be calculated while delivering high-fidelity results that are free of simulation artifacts.

Perfectly Matched Layers

PML implementation relies on a set of decay functions which control how power is coupled from FDTD’s field arrays into impedance-matched boundary layers. A typical sigmoid-based PML profile is depicted in the following graph.



At the edges of the simulation domain, power is coupled from the electric and magnetic field arrays into corresponding imaginary impedance-matched arrays. At this interface and subsequent layers, field components are multiplied by a scalar gradient which gradually reduces the field values to zero.

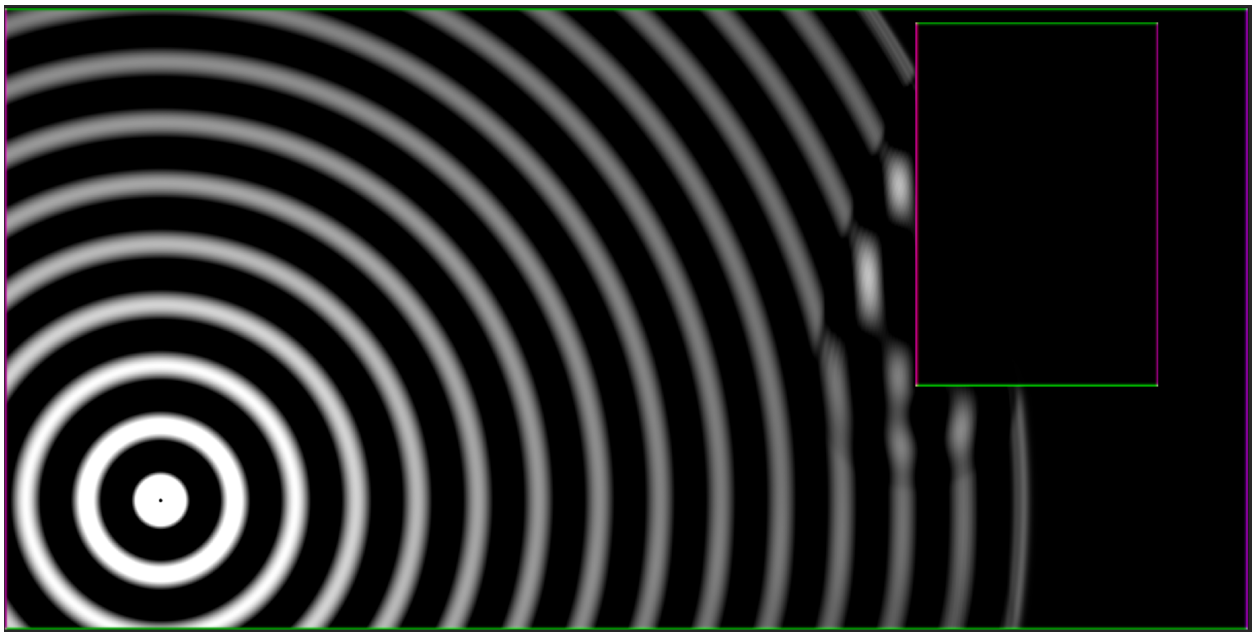
A typical implementation utilizes 10 PML layers. Use of fewer layers causes numerical stability issues as well as reflections while more than 10 layers provides no benefit. In the previous chart, 12 layers are indicated in order to illustrate the spatial offset between E and H profiles as dictated by the FDTD algorithm.

PML Sink Implementation

In order to efficiently remove uninteresting areas of the simulation, we require the ability to draw PML regions of arbitrary size and shape. For simplicity, this implementation relies on rectangular blocks. (Note that raster-based PML profiles could be added with appropriate tooling support.)

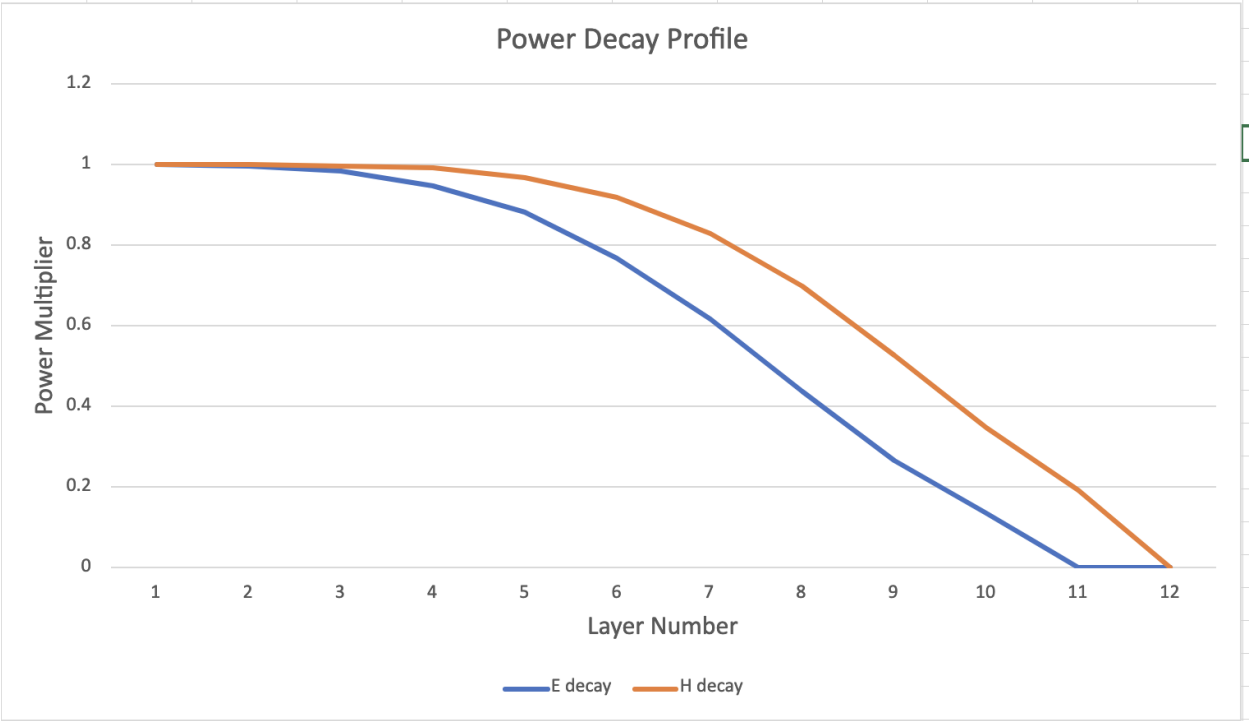
A typical PML implementation stores field values only at the edge of the domain. This is done to reduce the memory requirements of the simulation. For this exercise, I extended these fields to cover the entire domain. This provides a convenient way to map each cell in a simulation to a corresponding point in the PML domain.

For the first experiment, I wrote a function to draw a PML boundary at an arbitrary position within the domain.



Unfortunately, this approach failed. The PML region (as indicated by the rectangle in the upper-right portion of the previous figure) generated substantial reflections similar to what one would expect from a Dirichlet boundary.

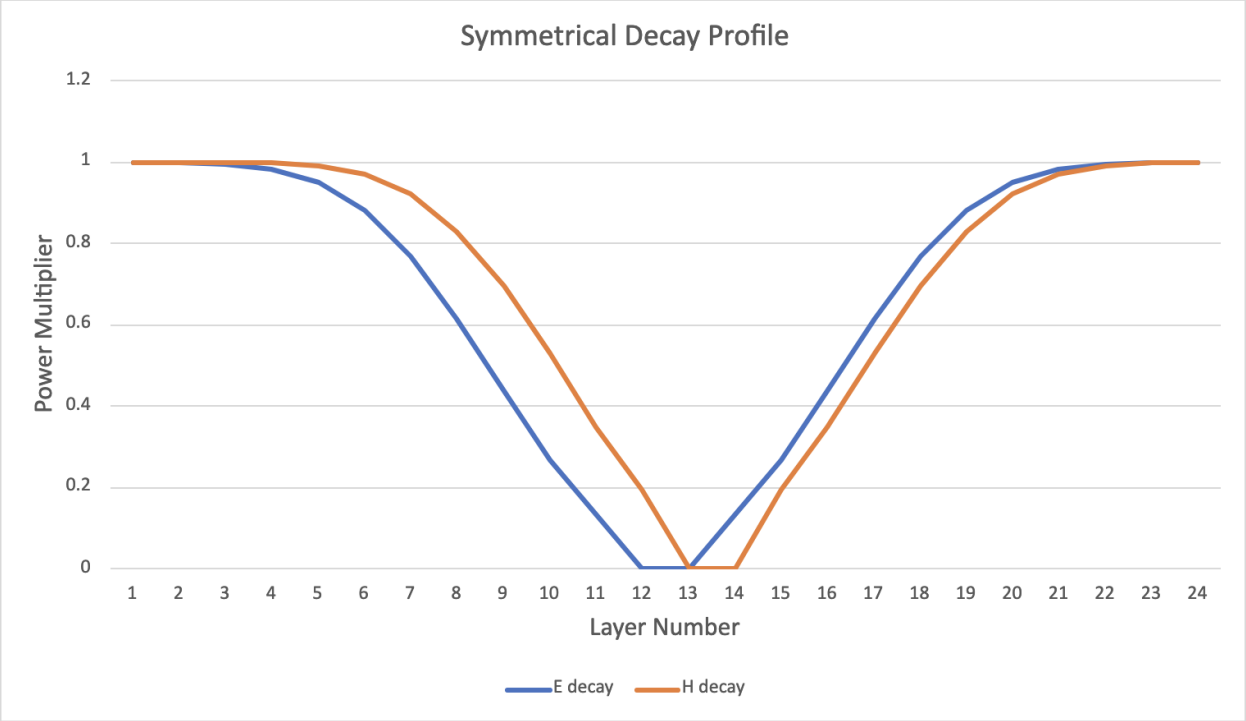
Recalling our earlier graph of the PML decay curve:



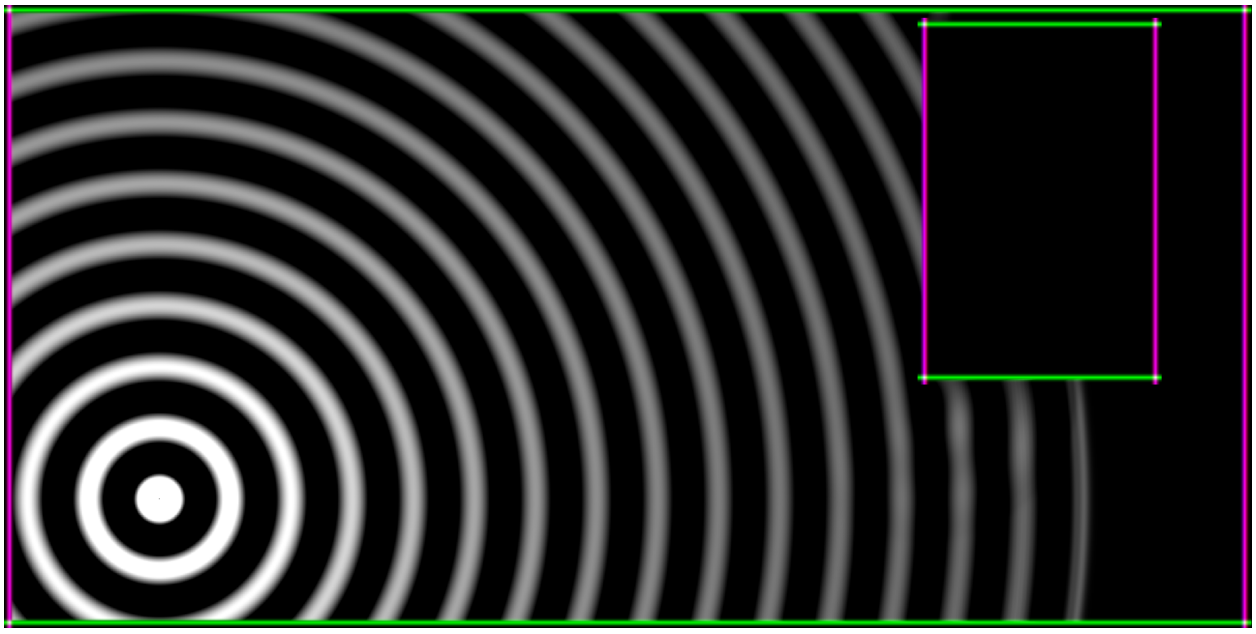
One can hypothesize about the underlying cause of the issue. This PML curve is asymmetrical. Unsurprisingly, it perfectly contains a source (subdomain) as indicated in the following image:

The solution is to generate a symmetrical PML curve. This is a simple matter of reflecting the decay factor and extending the boundary region from 10 layers (in each field) to 20 layers. Since we've extended the PML region to encompass the entire domain, this modification is fairly straightforward. The updated coefficient array is shown below:

E decay	H decay
1	1
0.999798	1
0.996781	0.999987
0.983809	0.99898
0.949718	0.992159
0.881656	0.970211
0.770145	0.920684
0.616398	0.831596
0.438039	0.697861
0.266546	0.528539
0.133287	0.349247
0	0.193701
0	0
0.133287	0
0.266546	0.193701
0.438039	0.349247
0.616398	0.528539
0.770145	0.697861
0.881656	0.831596
0.949718	0.920684
0.983809	0.970211
0.996781	0.992159
0.999798	0.99898
1	0.999987
1	1

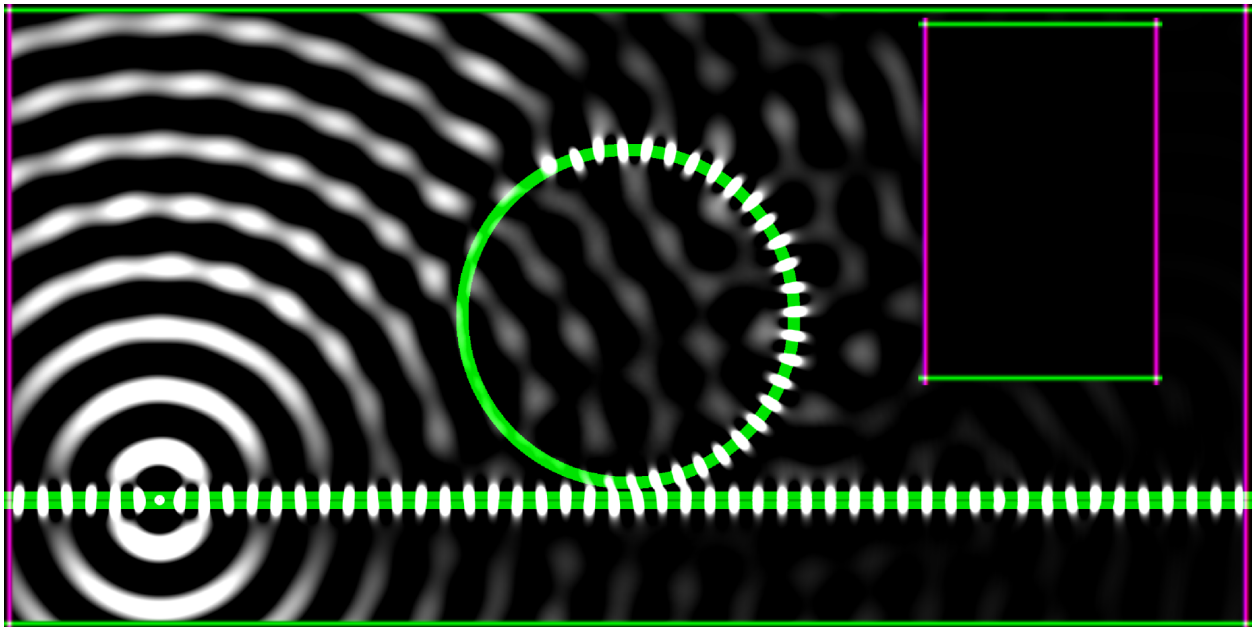


Note how the gradient rather unceremoniously encounters the X axis. While aesthetically displeasing, this generates no reflections or numerical artifacts.
Applying this symmetrical gradient to our initial simulation yields the following result:

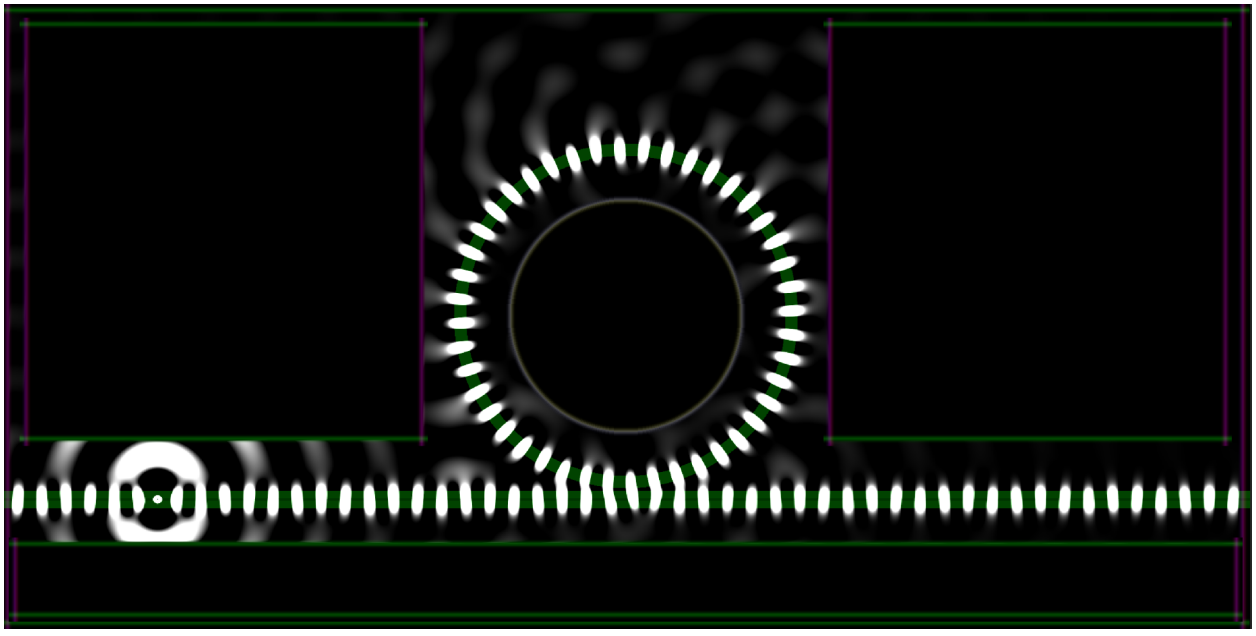


The modified PML curve yields no significant artifacts.

Extending this approach to the WGM test model yields the following result:



This appears to work quite well. The next step was to create multiple PML blocks to eliminate the parts of the domain that are not of interest.



This simulation includes a circular PML region at the center of the WGM sensor. This is a somewhat simpler artifact to construct. Given a center and radius, the absorption value for a given point within the given boundary is merely a function of distance of the given pixel from the center of the circle, scaled to fit the absorption gradient.

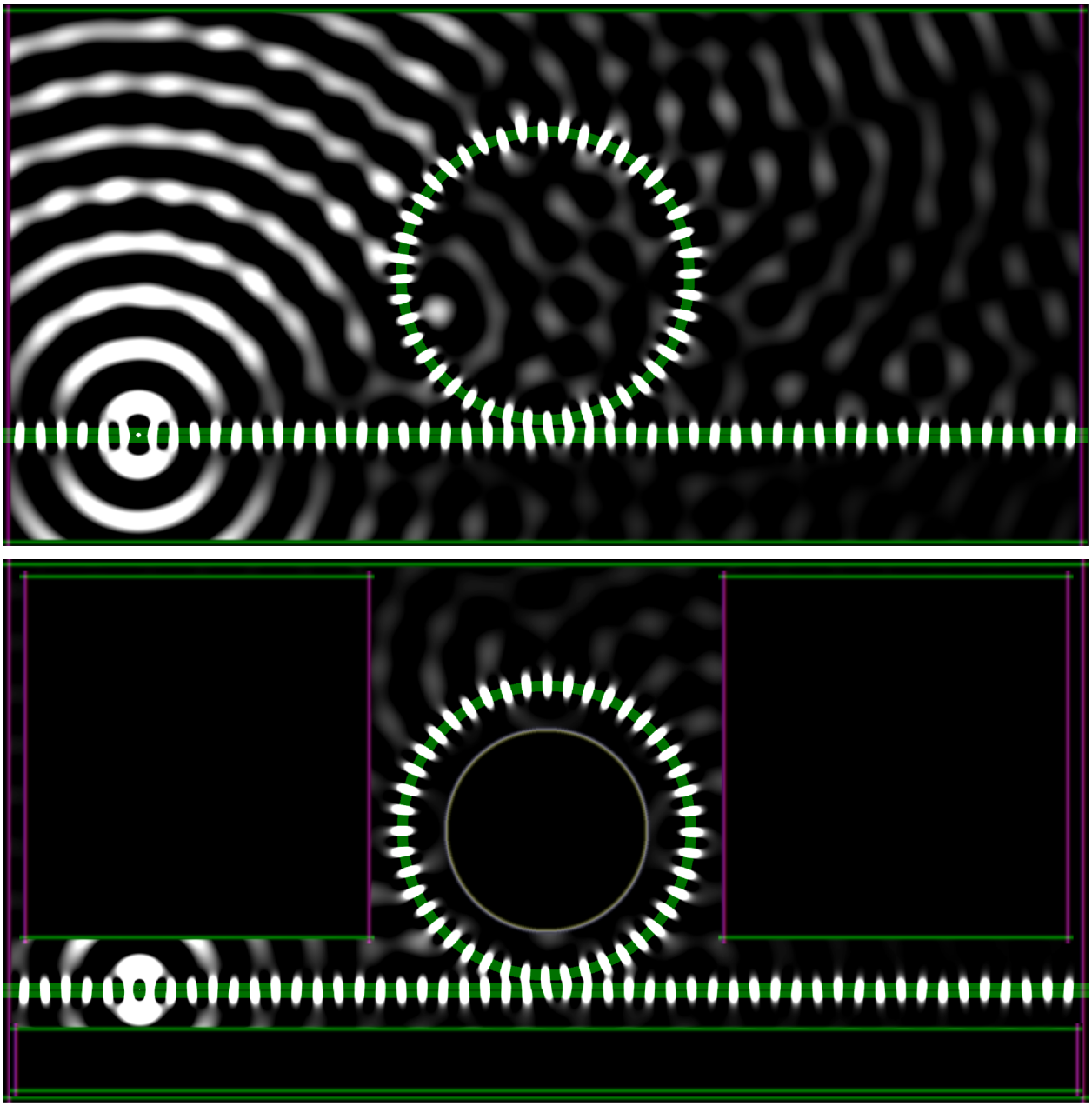
```
private void CreateSink(float4[] decay, int2 center, int radius)
{
    Assert.IsTrue(radius >= e_decay.Length);
    for (var i = -radius; i <= radius; ++i)
    {
        for (var j = -radius; j <= radius; ++j)
        {
            // calculate layer
            var d = radius - (int)Mathf.Sqrt(i * i + j * j);
            var o = offsetOf(i + center.x, j + center.y);
            var v = decay[o];

            if (d > 0 && d < e_decay.Length - 1)
            {
                /*
                x -> ezx decay
                y -> ezy decay
                z -> hyx decay
                w -> hxy decay
                */
                v.x = e_decay[d];
                v.y = e_decay[d];
                v.z = h_decay[d + 1];
                v.w = h_decay[d + 1];
                decay[o] = v;
            }
        }
    }
}
```

Note the `+1` indices indicating the offset between E and H layers.

Conclusions

Comparing the original WGM simulation with the modified PML Sink version yields the following results.

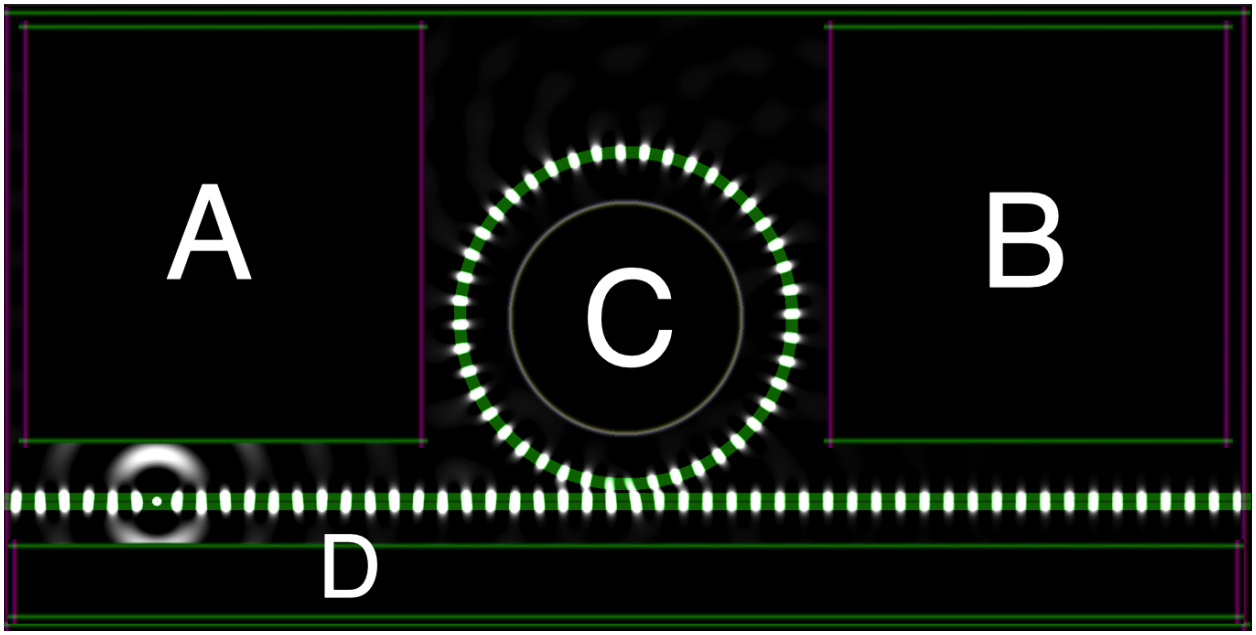


(Note that there is a slight phase variation between these images.)

It can be seen that the outputs are virtually identical. However, there are some artifacts. The shadow of any PML sink presents the opportunity for energy to flow around the sink. This appears to be a visual artifact with no significant numerical impact and, in fact, is simply highlighted by the image-based analysis here. In the same way that destructive EM interference does not indicate a lack of power at a given position, the long tails that bleed into a shadow do not indicate power that would not otherwise have existed at that location without the PML sinks.

Results

The example circuit in previous sections uses a domain of 2048x1024 Yee cells. In order to calculate the potential performance gain from this method, we examine the sizes of each additional PML region:



ID	Width	Height	Area
A	670	700	469,900
B	670	700	469,900
D	2024	138	279,312
C	Radius	200	125664

The total area of all additional PML regions is therefore 1,344,776 cells.

The total area of the 2048x1024 domain is 2,097,152 cells.

Domain	2097152
PML	1344776
Work reduction (PML / DomainSize)	**0.641…**

Knowing that FDTD throughput is a linear function of the number of Yee cells to be processed, we can see that the 64% reduction in work should yield a corresponding speedup.

Future work

A per-Yee-cell implementation would require a layer of indirection between domain cells and kernel launches in order to fully realize the memory savings that this technique enables. This implementation demonstrates the potential for such performance gains. For simplicity, a fully-optimized approach integrating both reduced memory footprint and reduced computational requirements is left for a future endeavor.