Nacka Strands GYMNASIUM

Magisim

Open source resource center for simulations

Jonas Cronholm
Gymnasiearbete 100 poäng
Klass: 21 TE
Teknikprogrammet
Läsåret: 2023-2024
Handledare: Peter Eriksson

Abstract

With serious competition in the modern market it is not trivial to create a product that can match the expectations of professional users and still appeal to hobby engineers, especially at a modest price point. The gap in capabilities between enterprise and open source simulation software is growing bigger due to lack of funding, which often results in limited research and development for free and open source projects. This growing disparity poses a significant challenge for developers aiming to balance advanced features with affordability.

Enterprise simulation software benefits from substantial financial backing, enabling teams to invest in cutting-edge research, continuous enhancements, and dedicated customer support. This translates to robust features, high accuracy, and comprehensive technical support, making them the preferred choice for intricate and mission-critical projects. However, the cost associated with these solutions can be prohibitive for smaller businesses and individual hobbyists. This gap is further exacerbated by the intricate nature of simulation software. Meeting the needs of professionals frequently involves complex algorithms, intricate modeling, and high-performance computing. Striving to offer similar capabilities within a modest price range for hobbyists can be daunting, as the associated costs can quickly spiral upwards.

Addressing this challenge, Magisim, an open-source project developed using modern tools such as machine learning technology, is introduced as a versatile solution. Designed primarily as an educational tool, Magisim allows for the integration of a wide range of simulation algorithms, and the programming of data flow using a node graph interface. This approach not only democratizes the learning and application of complex simulations but also bridges the gap in performance between high-cost enterprise solutions and open-source software. Consequently, Magisim serves as a testament to the potential of open-source development in narrowing the divide in technological capabilities, fostering an environment where both professionals and hobbyists can explore, learn, and contribute to the evolving field of simulation and data analysis.

Keywords: Computational electromagnetics, CEM, MoM, FEM, parallelization, simulator, CUDA, gradio, Data Analysis, Visualization

"An idiot admires complexity. A genius admires simplicity." - Terry A. Davis

Contents

1 Introduction

1.1 Background

The field of radio engineering, particularly within the amateur radio sphere, necessitates a deep understanding of electromagnetic (EM) theory and its practical applications. Traditionally, this domain has been supported by a variety of simulation tools, aiming to provide insights into the complex interactions of electromagnetic fields with their surrounding environments. However, a significant challenge has emerged from the existing landscape of these simulation tools, marked by their prohibitive cost and steep learning curves. This divide presents a substantial barrier to entry for amateur radio enthusiasts and learners, who often seek both affordability and simplicity in educational resources.

1.2 Problem

Historically, professional-grade simulation software in electromagnetics has been tailored to meet the demanding needs of industry experts, incorporating advanced features and comprehensive simulation capabilities. While powerful, these tools come at a price point that is often beyond the reach of hobbyists and educational users. Concurrently, the free or low-cost alternatives available in the market tend to compromise either on the width of features or on user accessibility. The complexity of these tools, coupled with often inadequate documentation, further exacerbates the challenge for those new to the field, impeding practical, hands-on learning.

This gap in the market led to the conceptualization of Magisim, initially envisioned as a user-friendly and cost-effective simulator specifically for electromagnetics. Targeted towards the amateur radio community, the primary goal was to demystify EM theory through interactive simulations, making it more approachable for non-professionals. However, as the project progressed, we realized the broader potential for the underlying technology. The modularity of our initial design, made the software applicable in more fields related to simulations or data analysis. This steered the development of Magisim towards becoming a more versatile platform, transcending its original scope.

In the last few years, machine learning has become one of the most important tools in many fields, including software development. This project also aims at benchmarking the code production quality, efficacy and finding new ways of accelerating the development process of software using Large Language Models (LLMs) as a central tool.

1.3 Introduction to Computational Electromagnetics

Computational electromagnetics (CEM) is a field of study that employs numerical methods to solve problems that involve electromagnetic interactions. These computational methods are essential for designing, analyzing, and optimizing electronic and optical devices, such as antennas, radar, waveguides, and optical systems.

A key technique in CEM is the Finite-Difference Time-Domain (FDTD) method, which simulates electromagnetic field behavior by solving Maxwell's equations across a discretized spatial and temporal grid. This method is prized for its ability to model complex interactions in various materials and structures accurately. FDTD is particularly useful in fields such as antenna design, optical device analysis, and electromagnetic compatibility testing. It allows for a detailed examination of how electromagnetic waves propagate and interact with their environment, making it an essential tool for both researchers and engineers. In educational settings, tools like Magisim utilize FDTD to help students and hobbyists understand and visualize electromagnetic phenomena in a straightforward and accessible manner.

1.4 Related Work

1.4.1 Proprietary Software

Many solutions exist for simulating electromagnetic systems, particularly in the realm of proprietary software that offers advanced capabilities but often at a cost that is prohibitive for amateurs. Tools like Ansys HFSS, Altair FEKO, and XFdtd are common in this field, each providing specialized features tailored to complex and precise electromagnetic simulation needs.

Ansys HFSS (high-frequency structure simulator) excels in using the Finite Element Method for detailed electromagnetic field simulations, ideal for high-frequency electronics and antenna design. **Altair FEldberechnung für Körper mit beliebiger Oberfläche (FEKO)** offers a broad range of computational methods, making it versatile for applications across electromagnetic compatibility and antenna design, amongst others. **XFdtd** utilizes the Finite-Difference Time-Domain method, suitable for transient electromagnetic simulations and interactions, such as EMC testing and bioelectromagnetic analyses.

Despite their robust features and extensive applications, these tools are often not accessible to the common amateur due to their high cost and the complex technical expertise required. This creates a significant barrier for hobbyists and smaller enterprises who might not have the resources or need for such advanced functionalities.

1.4.2 Open Source Projects

In the field of electromagnetic simulations, the openEMS tool, described by Liebig et al. (2012)[2], stands out as a noteworthy open-source platform leveraging the Finite-Difference Time-Domain (FDTD) method. This tool, freely available to the public, supports various simulation needs, particularly for high-frequency electromagnetic applications. The openEMS platform uses an equivalent-circuit (EC) FDTD method, which simplifies the implementation of material dispersion, thus enhancing the simulation of complex electromagnetic environments. Although openEMS is a powerful tool, its depth and the required scripting for setup and control can be daunting for amateurs and new users.

In his PhD thesis, Novel architectures for brain-inspired photonic computers. Ghent University (2020)[1], Floris Laporte employs a custom Finite-Difference Time-Domain (FDTD) simulator to analyze photonic structures that mimic neural processes for neuromorphic computing applications. Laporte's work focuses on the simulation aspect, using this method to delve into the dynamic behaviors of photonic cavities and their potential as elements in brain-inspired computing architectures. In developing our simulator, Magisim, we have integrated Laporte's FDTD code as a core component.

1.5 Scope

Initially, we aimed to develop a CUDA-accelerated Finite-Difference Time-Domain (FDTD) simulator that could leverage the computational power of modern GPUs to enhance simulation speed significantly. However, during the early stages of development, it became apparent that designing and implementing a fully CUDA-accelerated simulator within the project's limited timeframe was infeasible.

- **Technical Complexity:** The integration of CUDA technology into the FDTD algorithm involves deep knowledge of parallel programming and optimization strategies that were beyond the immediate capabilities of the team.
- **Resource Constraints:** Limited access to necessary hardware and software resources to test and develop CUDA-accelerated features effectively.
- **Development Time:** The time required to learn, implement, and debug a CUDA-accelerated approach exceeded the project's scheduled timeline.

Given these challenges, the focus shifted towards enhancing modularity. This aimed at laying a robust foundation that could support future integrations of CUDA acceleration and other advanced features. To facilitate development and ensure the software's long-term viability, a set of design rules for extensions was established. These guidelines serve multiple purposes:

- **Ease of Development:** Simplifying the process for future contributors to develop and integrate new modules or features without extensive reworking of the existing codebase.
- **Consistency:** Maintaining a uniform architecture that supports the seamless functioning of diverse components developed by different contributors.

- **Expandability:** Allowing the simulator to evolve over time by accommodating new technologies and methodologies as they become viable.

2 Method

The initial phase of the project involved comprehensive planning using the Kanban system implemented through Atlassian Trello. This enabled efficient organization and prioritization of envisioned functionalities for the simulator and its interface. The systematic arrangement of tasks on the Kanban board facilitated a clear visualization of the project scope, ensuring effective integration of all functionalities. Subsequently, a Gantt chart was utilized to structure the project's timeline and responsibilities, allowing for a detailed outline of the time plan and division of labor. We also utilized machine learning models, such as Generative Pre-trained Transformer 4 (GPT-4)[3] to further aid project planning, as well as for code completion.

2.1 Project Structure

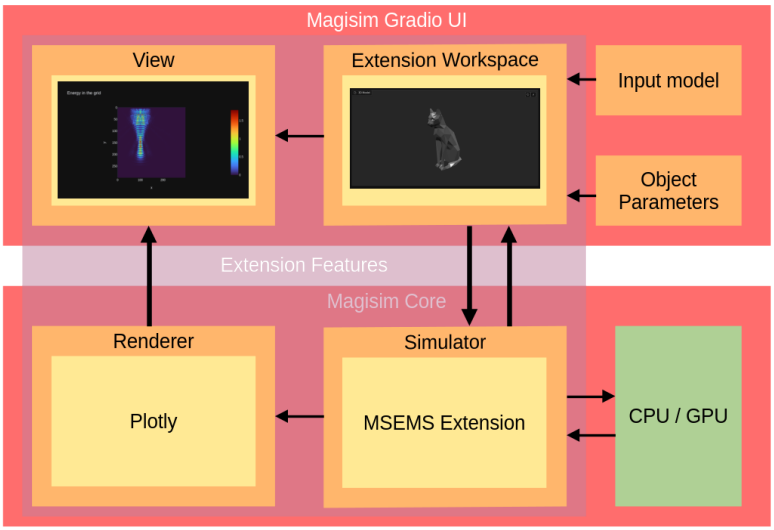
The directory structure is as follows:

```
magisim
├── docs
│   └── <website sources for magisim.com>
├── notebooks
│   └── <development notes as well as this paper>
├── resources
│   └── <images, videos, gifs for the readme and website>
└── src
    ├── tests
    │   └── <development tests>
    └── ui
        ├── extensionmgr
        ├── extensions
        │   └── <user extensions get installed here>
        ├── nodemgr
        ├── projects
        │   └── <user saved projects are stored here>
        ├── settingsmgr
        ├── shapeloader
        ├── shared
        │   └── <magisim resources such as type definitions>
        └── <scripts>
```

2.2 User Interface

Following the establishment of the project's framework and timeline, focus shifted to the user interface (UI) design for Magisim. Several UI development toolkits, including QT and GTK, were evaluated to determine the most suitable option based on modularity, integration ease, and development efficiency. Despite the admirable features of QT and GTK, challenges were encountered with their steep learning curves and complex integration processes, including licensing and cost constraints.

The search ultimately led to the selection of Gradio, a growing UI library distinguished by its modular design and streamlined development process. Gradio's capability for rapid prototyping and its intuitive user experience made it the preferred choice for the project, facilitating the swift iteration of UI components.



Each extension describes its own user interface by using modules available in the Gradio package. This allows third parties to rapidly develop intuitive and efficient interfaces that integrate seamlessly with the core functionalities of Magisim. The use of Gradio's modules offers developers the flexibility to customize and extend the interface according to specific needs, developers may also create "custom components" allowing even more degrees of freedom.

2.3 Extension Model and Templates

Aiming for high modularity, the project was structured so that every component was defined as an "Extension". This approach enabled the development of a unified system where all components adhered to a set of predefined rules, including a universal code structure. Such standardization allowed for the components to be loaded by a single function at startup, simplifying the development process and reducing program complexity.

```
# shared extension imports
import gradio as gr
from shared.builtin import Extension
from shared.config import Config

class ExtensionMeta:
name: str = "name"
uuid: str = "some-uuid-v4"
authors: list = ["author"]
version: str = "1.0.0"
license: str = "license"
description: str = """multiline description if needed"""

class ExtensionType:
types: list = [Extension.Simulator, Extension.Workspace]
hasNodes: [(Extension,(list,list))] = [
(Extension.Simulator, # define a simulator node
        (
                [ # inputs
                        ("some-input","some-type")
                ],
                [ # outputs
                        ("some-output1","some-type"),
                        ("some-output2","some-type")
                ]
        )
)]

def load_workspace(app: gr.Blocks):
        with gr.Tab(ExtensionMeta.name, id=ExtensionMeta.uuid):
                gr.Markdown(ExtensionMeta.description)
                # Extension UI code
```

2.4 Message Queue and Routing

In defining component interaction and data handling, the initial consideration was to develop an in-house asynchronous communication layer. However, it was soon recognized that many solutions existed for such functionality. A message queue system in combination with a "Router" was eventually chosen. The router maps connections between extensions in a Redis database, allowing extensions to remain unaware of the destinations for their outputs. Using Python's Pickle library, the system enables the transmission of diverse data types, including 3D models and tensor arrays.

2.5 FDTD Simulator

Examples from Laportes[1] original library were tailored to specific scenarios. We extended these to be more versatile by introducing parameters for material properties, boundary conditions, and source configurations. Though this library claimed CUDA accelerated code, it would not work in any configuration tried during development. After communicating this to the original developer, and trying several bug-fixes and patches as well as utilizing input from a Language Model, we had simply run out of time to develop parallelized code in order to reach our original goals. Instead we settled on using PyTorch as the computation backend.

Post adaptation of the FDTD core, a user interface (UI) was developed to allow intuitive setup and execution of simulations. The UI was designed to be simple yet effective, enabling users to configure simulations easily and visualize results in real time. The UI offers real-time visualization of the simulation grid and allows the option between Matplotlib and Plotly for visualizations. For common simulation setups, users can choose from pre-configured settings, which simplifies the simulation process and saves time.

2.6 Software Distribution

It is certain that the importance of keeping scientific resources and research available to the public can not be stressed enough.

Hence the source code of Magisim is distributed under the GNU Lesser General Public License (LGPLv3) allowing for all content within the non-commercial section of the codebase to be released publicly. The desicion to use LGPLv3 instead of GNU Public License v3 (GPLv3) was made in order to give third party developers more freedom to chose a license that fits their software, while being able to redistribute altered versions of our code. GitHub was chosen for version management and publishing.

3 Result

3.1 User interface

3.2 Simulation Process

Once the software is started using the included script, a user interface is available at *http://localhost:8000*. By default, the program listens only for connections from the same computer, but this can be modified in the configuration file.

Our interface lands the user at the greeter, see figure 1. From here, the user can browse through all installed extensions (represented as tabs, denoted workspaces) and perform actions. To run a simple simulation, the connection between extensions must be formed by creating and saving a node graph, see figure 2.

Once the connection has been formed, a model can be uploaded to the slicing interface (note that this is only necessary for simulating a 3d object represented in 2d space, 2d only simulations can be done by either generating a simple lens object or uploading a *Magisim Object (.mso)* file), as seen in figure 3.

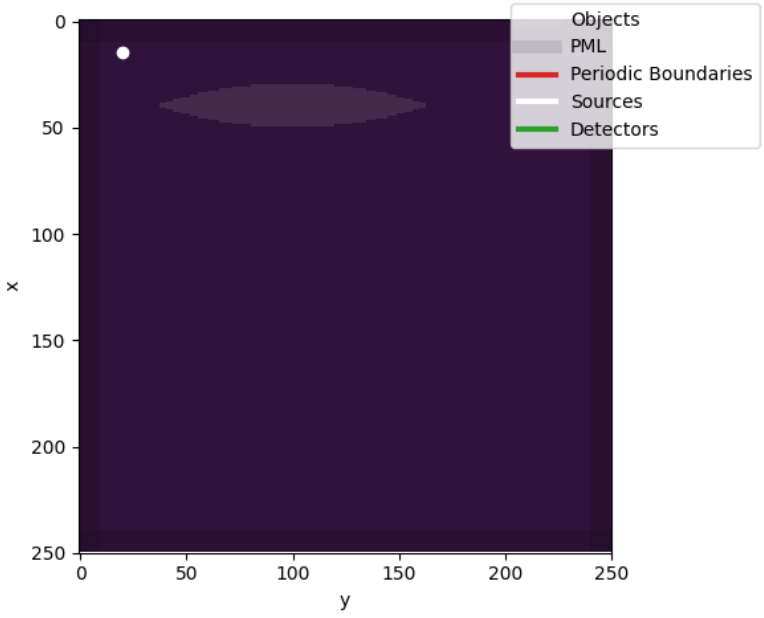
The user can then enter the *MSEMS* workspace and edit the object parameters as well as configure sources, PML:s and detectors for analysis. (figure 4 and 5)

After configuring the crucial parameters, the user switches to the simulation sub-workspace, see figure 6. Here the user chooses how many timesteps will be simulated and if detector readings should be generated.

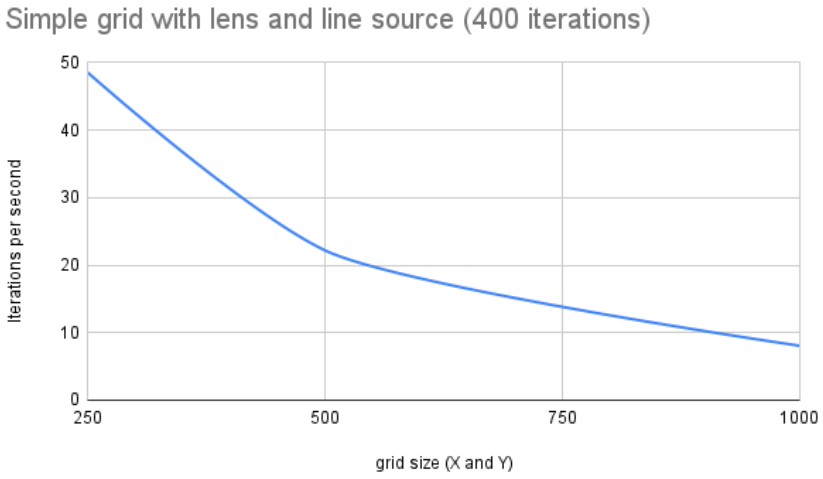
Once the simulation is complete, a resulting image can be saved using the plot viewer interface.

3.3 Simulation Performance Characterization

A benchmarking simulation was set up, with an initial grid size of 250^2, where each unit represents $\frac{1}{10}\lambda$. A lens object was subsequently added to the grid. A point source was placed at $(15, 20)$ configured as a continuous wave function with the wavelength of 635nm.



Five simulations were ran in conjunction with a timer function and the result was appended to a spreadsheet (see figure 7). This process was then repeated two more times, each with a quadrupled grid size.



When the simulation grid size quadruples, resulting in a fourfold increase in the space (n becomes $4n$), the calculation speed drops by 50%. This change can be represented by the equation:

$$S' = \frac{1}{2}S$$

From this, we infer that speed S is inversely proportional to the square root of the simulation space size n:

$$S(n) \propto \frac{1}{\sqrt{n}}$$

This relationship shows that the computation speed scales as $O(n^{-1/2})$ in big-O notation, which is significantly worse than initially planned. This is most likely due to our plotting code, since it has to convert the simulation grid points into a plot every single frame, in order to get real time simulations. This could be solved by animating the simulation and then pushing the result to the front-end. This would however add significant loading times and would degrade user experience drastically. Another potential source of performance decreases lie within the simulation code itself as the simulation is inherently multidimensional. Currently, the calculations are performed serially using the CPU, accellerating the code using i.e. Nvidia CUDA technology would reduce the processing time. With parallelization, the effectiveness of a multidimensional computation can be increased drastically, essentially allowing for $n-1$ dimensional computation as illustrated in figure 12 and 13.

4 Conclusion

The development of Magisim aimed to provide a modular and accessible tool for electromagnetic simulation, with specific emphasis on educational use and ease of expansion. This conclusion outlines the major achievements and reflects on the challenges faced throughout the project.

Achievements:

- **Modular Interface:** A significant achievement of this project was the creation of a modular interface featuring a functional node graph interface for data flow programming. This interface supports dynamic interaction with the simulation processes and enhances user experience by providing intuitive control over complex simulations.
- **FDTD Simulation Integration:** Successfully integrating the Finite-Difference Time-Domain (FDTD) simulation code into the system as an extension. This integration underscores the interface's capacity to handle advanced computational models and adapt to diverse simulation requirements.
- **User Interface Development:** The selection and implementation of Gradio as the UI framework facilitated rapid prototyping and effective user interaction, which has been pivotal in making the simulation tool more accessible to non-experts.

Challenges:

- **Team Collaboration:** The project faced hurdles in team coordination, impacting productivity. Enhancing communication, project management practices and increasing group motivation could mitigate similar issues in future developments.
- **Outdated and Unsupported Code:** Dependency on out-of-date, unsupported code led to significant setbacks. Future efforts will need to focus on either updating this code or replacing it with more current, supported alternatives.
- **Limited CUDA Acceleration:** The lack of effective CUDA acceleration due to the outdated codebase was a significant technical challenge, which limited the performance enhancements expected from such technologies.
- **ML Assistance Limitations:** Utilizing Machine Learning Models to assist in code generation presented challenges, particularly in maintaining high code quality. This aspect underscores the necessity for thorough validation and refinement of LLM-generated code before its integration into production environments.

References

[1] Laporte, Floris. "Novel architectures for brain-inspired photonic computers". ISBN: 9789463553599. PhD thesis. Ghent University, 2020. xxxvi, 157.

[2] Thorsten Liebig et al. "openEMS – a free and open source equivalent-circuit (EC) FDTD simulation platform supporting cylindrical coordinates suitable for the analysis of traveling wave MRI applications". In: *International journal of numerical modelling (Print)* 26.6 (Dec. 2012), pp. 680–696. DOI: 10.1002/jnm.1875. URL: https://doi.org/10.1002/jnm.1875.

[3] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

Appendices

Figure 1: Magisim Greeter

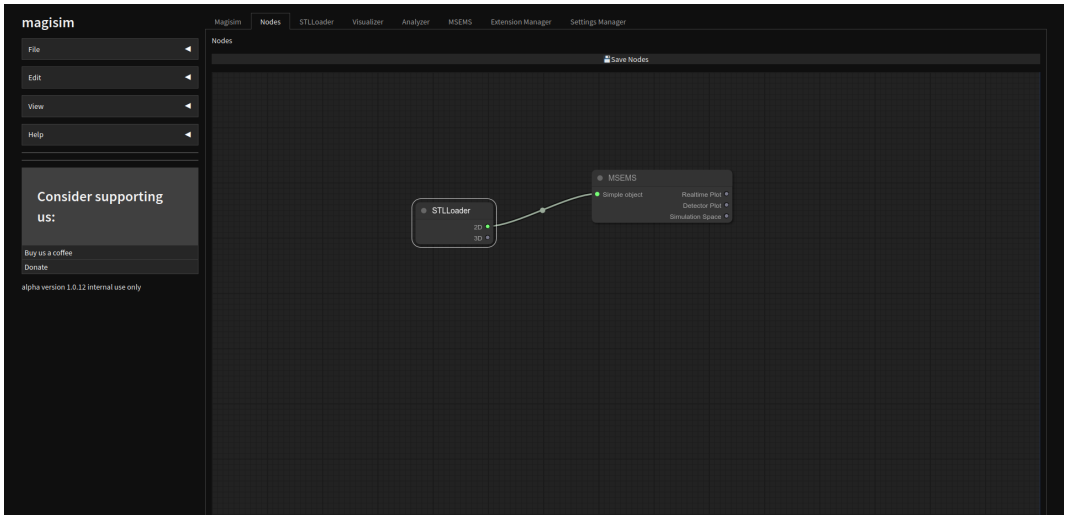


Figure 2: Magisim Node Graph Interface

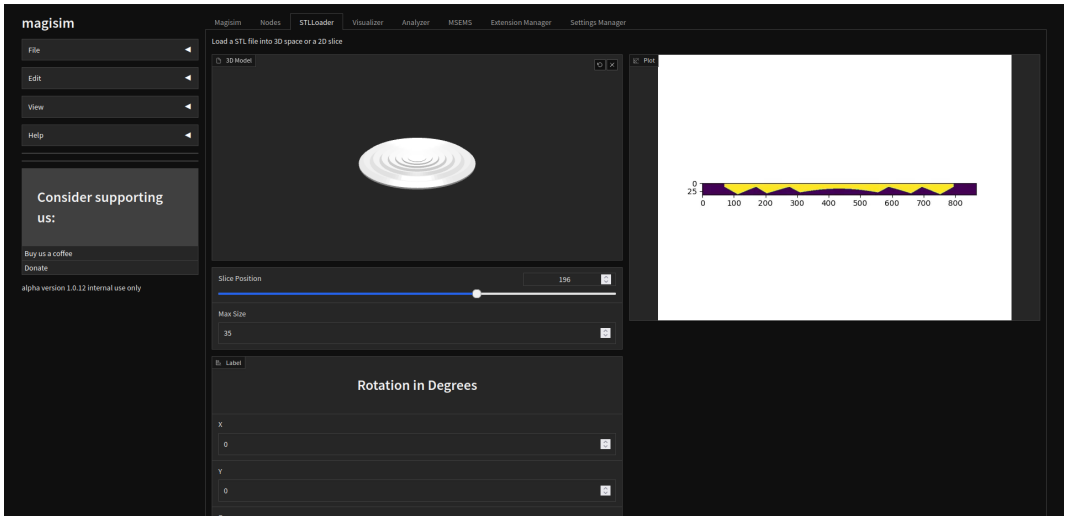


Figure 3: Magisim 3D Slicer

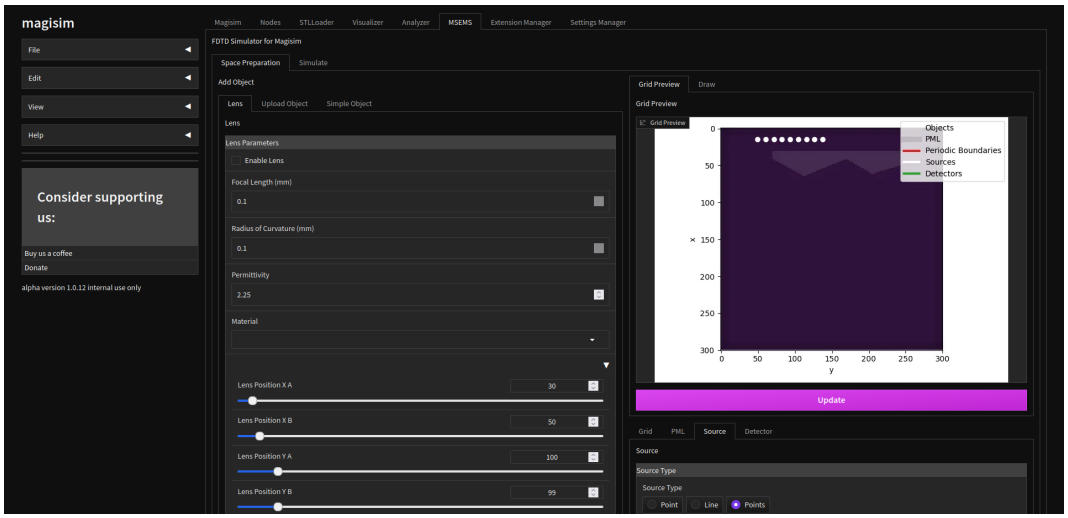


Figure 4: MSEMS Grid Preparation Interface

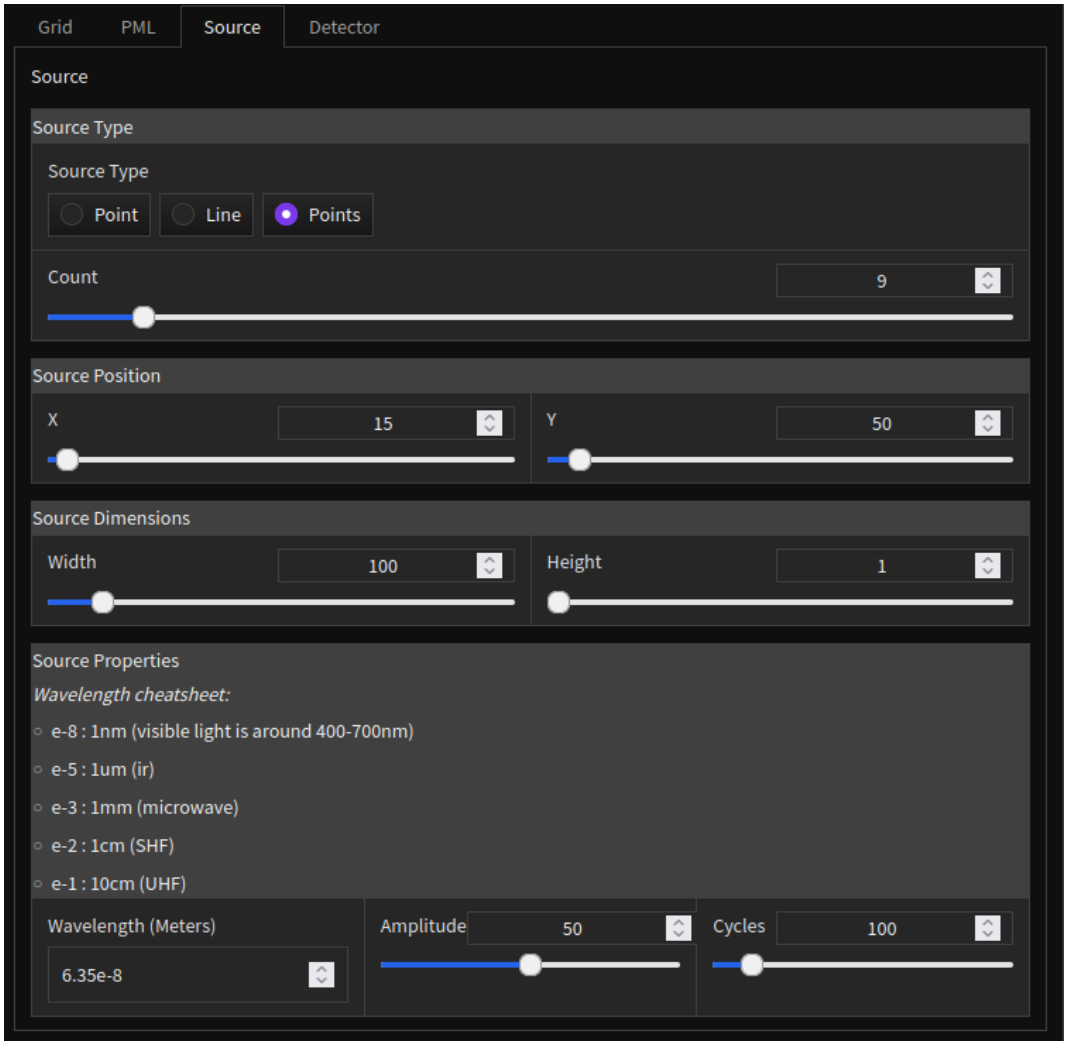


Figure 5: MSEMS Source Properties

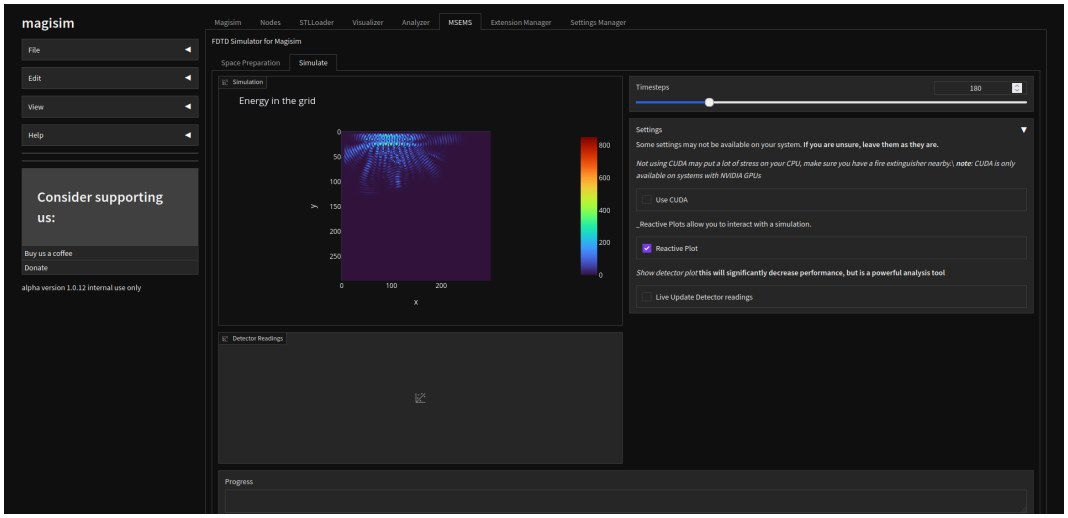


Figure 6: MSEMS Simulation

400 it	Grid x size	grid y size	it/s	
	250	250	48,27	
	250	250	47,68	
	250	250	52,75	
	250	250	43,66	
	250	250	50,41	48,554
	500	500	22,6	
	500	500	23,55	
	500	500	21,01	
	500	500	22,73	
	500	500	21,04	22,186
	1000	1000	7,36	
	1000	1000	9,36	
	1000	1000	8,69	
	1000	1000	7,58	
	1000	1000	7,29	8,056

Figure 7: Benchmarking Data

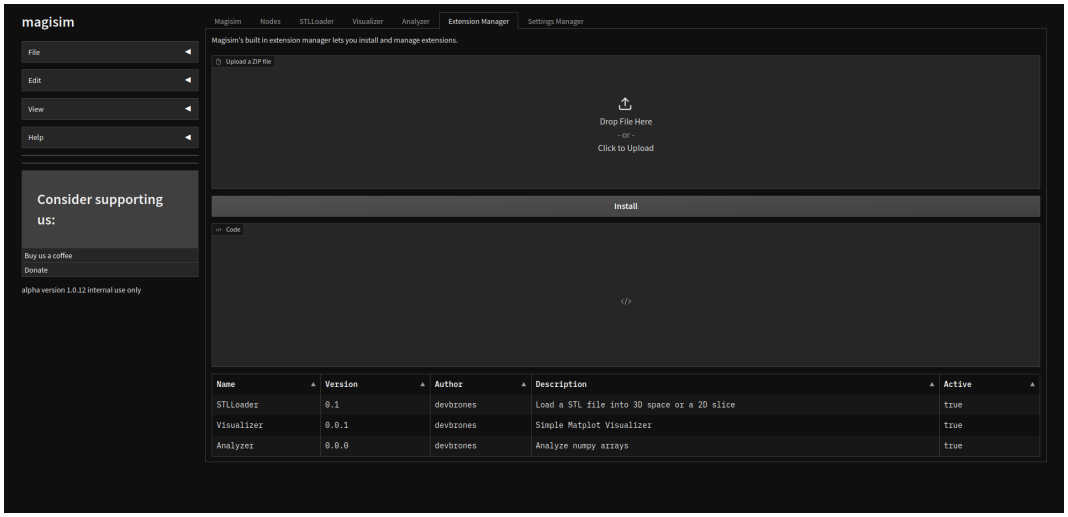


Figure 8: Extension Manager

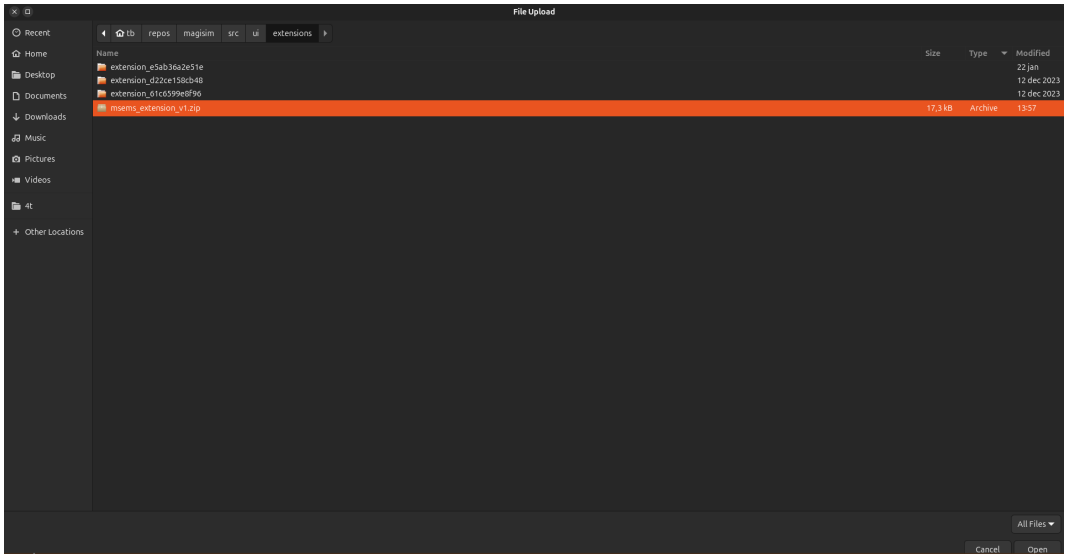


Figure 9: Selecting a extension zip file

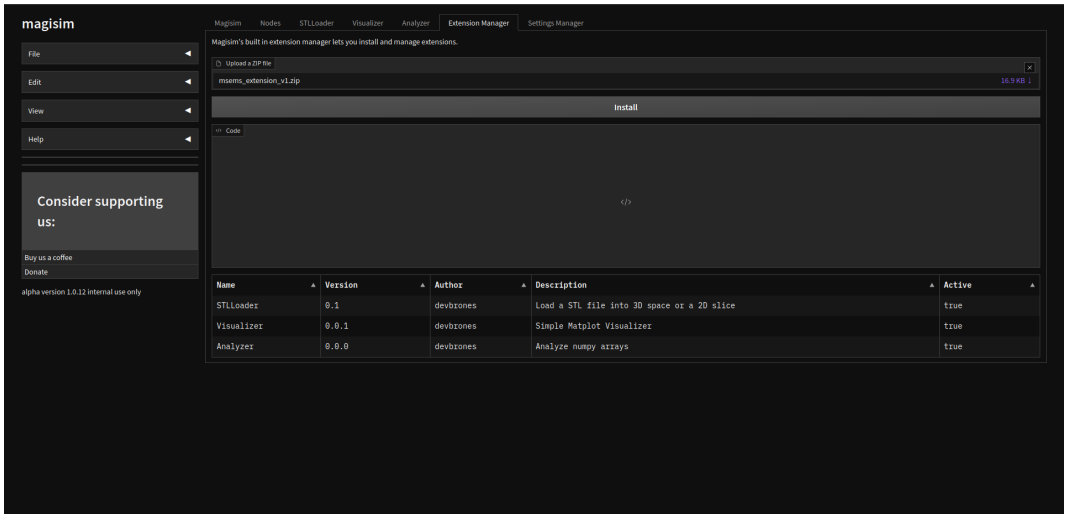


Figure 10: Extension uploaded to Magisim

Name	Version	Author	Description	Active
STLLoader	0.1	devbrones	Load a STL file into 3D space or a 2D slice	true
Visualizer	0.0.1	devbrones	Simple Matplot Visualizer	true
Analyzer	0.0.0	devbrones	Analyze numpy arrays	true
MSEMS	1.1	devbrones	FDTD Simulator for Magisim	true

Figure 11: Extension installed after restart

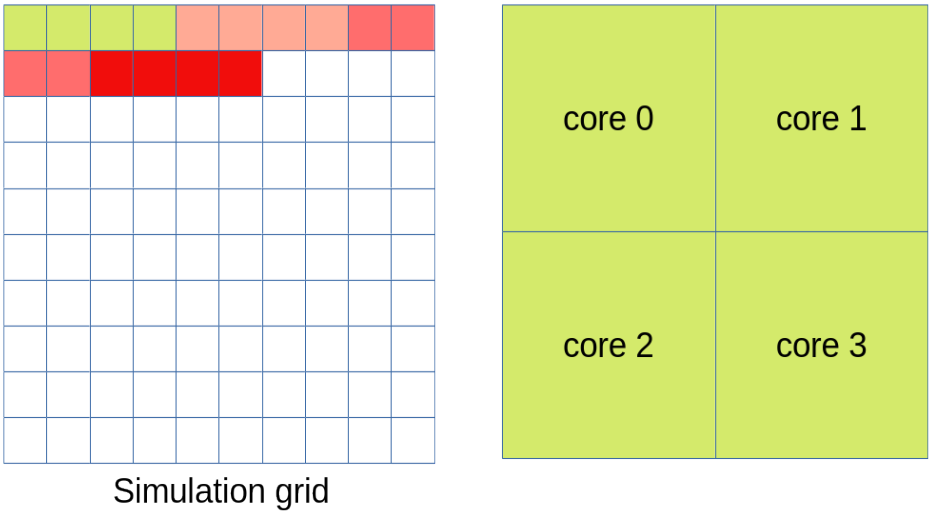


Figure 12: Serial Computation on a quad core CPU, red indicates next segment

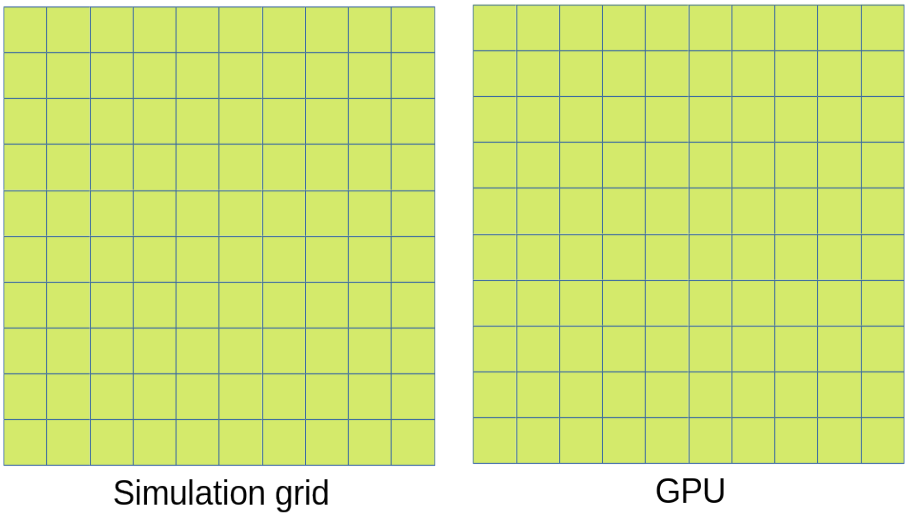


Figure 13: Parallel Computation on a n-Core GPU, simultaneous cell processing

Written in LaTeX