Design Changes to a Laser-Plasma Simulation Code to Permit Twisted Light Studies

by

Blaine Armstrong

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Photonics and Plasmas

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Schemes to implement laser bandwidth wide enough to mitigate laser-plasma instabilities will be both intrusive and expensive. As an alternate approach, work is presented that investigates the mitigating effects of spatial, rather than temporal, laser beam conditioning on cross-beam energy transfer (CBET) [I.V. Igumenshchev *et al.,* Phys. Plasmas **19**, 056314 (2012)]. Such conditioning might be generated by phase plates alone and could therefore be implemented more easily. We have quantified the energy exchange occurring between crossing laser beams that possess orbital angular momentum (OAM) as the amount of OAM exchange between the beams is varied [*cf. e.g.,* M. Padgett *et al.,* Physics Today **57**, 35 (2004)]. This work enables studies to be performed in 3-D using the non-paraxial wave-based *LPSE* simulation code [J.F. Myatt *et al.,* J. Comp. Phys **399**, 108916 (2019)]. It required significant modifications to the code to allow for the beams with OAM. The modifications required changes to the boundary conditions and the total field scattered field (TF-SF) to be implemented successfully.

Preface

This thesis is an original work by Blaine Armstrong. Figures 1.1 and 1.2 are reproduced from "Inertial Confinement Fusion: An Introduction The ENERGY of the STARS" from The University of Rochester's Laboratory for Laser Energetics. Copyright permissions were obtained for the following figures: 1.3, 3.1, 3.10, and 3.11. Fig. 3.2 was reproduced with Creative Commons License from https://commons.wikimedia-.org/wiki/File:GaussianBeamWaist.svg

“The only true wisdom is in knowing you know nothing.”

-Socrates

Acknowledgements

I acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference numbers RGPIN-2018-05787,RGPAS-2018-522497]. Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéros de référence RGPIN-2018-05787, RGPAS-2018-522497].

I also acknowledge the Laboratory for Laser Energeticss for the *LPSE* code made available by special agreement.

First of all, I would like to thank my supervisor, Dr. Jason Myatt, for taking me on as a student and guiding me through the process of my Masters. I would also like to thank my colleague Andrew Longman for his insightful discussions and guidance. I would like to thank my parents for their love and support, as well as, allowing me to do my Masters. Finally, I would like to thank my friends Nicolas Faria and Mark Robinson for their talks and support throughout the entire process.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the human race advances in technology and knowledge, the requirement of energy increases to feed the advancement. With this increase of energy consumption comes the consequence of an evolving society. This is the carbon dioxide emissions for the most prominent energy production methods. The main production of energy is burning hydrocarbons to make steam to power turbines. The process of burning the hydrocarbons make carbon dioxide. The increase of carbon dioxide in the atmosphere is the main contributor to the climate change of Earth casing extreme weather events and is the most prominent threat to current human civilization. Thus the pursuit to solve this dilemma without stopping the advancements of society is the main problem at hand for this and coming generations of future people of science.

There are already multiple avenues being worked on to solve this crisis, such as solar energy, wind energy, hydroelectric energy, and nuclear fission energy. These are the top alternatives but not all the possible alternatives. Each one of these energy sources has its benefits and drawbacks as well. Each of these are being researched and advancements are being made in efficiency, capability, and availability.

The benefit of solar energy is the abundance of it available from the sun. Solar energy requires light from the sun, which is then turned into usable electrical energy from solar panels made of crystalline silicon. The problems associated with solar panels though is the lifetime of the crystalline silicon is about 25 years, along with energy efficiency rate of 25 percent and is easily damaged by weather such as hail and heavy snow. Also, solar panels are not very efficient in regions with lower levels of direct sunlight and winters with snow which cover the panels stopping the light being converted in electrical energy.

Wind energy is another alternative energy source which uses wind to rotate large scale turbines to change the energy from the wind into mechanical energy then into electrical energy. The upside to wind energy is there is no direct waste product from the production of the energy. The downside is wind is not constant and it is hard to store electrical energy on the scale of city power generation.

Hydroelectric energy is the use of large bodies of water to rotate turbines to generate electrical energy. This is done using a dam to block a river to fuel the hydroelectric dam. Dams are an alternative energy source with its own short comings. First, land must be flooded by the dam to power it causing damage to ecosystems and the displacement of people living around it. Also, if the dam fails and breaks there is the potential for a huge catastrophe causing large scale damage.

Then there is nuclear energy. There are two different forms of nuclear energy: fission and fusion. Fission is when a heavy element with a large nuclei, such as uranium, splits into two or more smaller nuclei along with other particles accompanied by the release of a large amount of energy. Nuclear fusion occurs when two or more light nuclei overcome the Coulomb force of the particles and the nuclear force takes over and fuses the two nuclei into a heavier one. This has a change in mass which results in an output of energy. Below in equation 1.1 is the fusion reaction with the largest cross section, where deuterium and tritium fuse together making helium and a free neutron.

$$D + T \rightarrow He^4(3.5MeV) + n(14.1MeV) \tag{1.1}$$

Nuclear fission in the reaction used in current nuclear power plants. Nuclear power plants work by having a nuclear reactor with a sub-critical mass of fissile material is assembled and the reaction is controlled by varying the neutron flux using control rods. There is a possibility of criticality and a runaway chain reaction. The heat from the reaction is used to make steam that rotates a turbine that powers a generator making electrical energy. Nuclear power plants are very large scale facilities with lots of regulations due to the possibility of the reactor going critical and exploding. The damage to the plant and the surrounding environment can be serious and long lasting. Nuclear power planets also produce nuclear waste that needs to be handled appropriately due to its radioactive nature.

In the nuclear fusion category there is two major subcategories being researched: magnetic confinement fusion and inertial confinement fusion (ICF). Magnetic confinement fusion uses strong magnetic fields to confine the fuel used for fusion, that is in the plasma state, to very high pressure for the fusion reaction to occur. The most researched approach to ICF uses lasers to confine and compress the fuel to get to conditions required for the nuclear fusion reaction to overcome the Coulomb force causing the fuel to ignite and burn.

1.1 Inertial Confinement Fusion using lasers

The ICF approach, as seen in Fig. 1.1 uses a laser or particle beam to compress, as well as heat, a small capsule of fuel. This fuel is a deuterium and tritium mixture in a shell to keep it contained. When the energy of these beams is applied to the outside of the fuel capsule it ablates. The ablation pressure causes the capsule to expand inward, while the ablated plasma expands outward. The converging target causes the fuel inside to be compressed and be heated to the point where it can undergo fusion reactions and release energy.

In ICF there are two basic schools of thought used to accomplish fusion with laser drivers; indirect-drive method and the direct-drive method. In the direct-drive method laser beams directly irradiate the spherical capsule to accomplish the process mentioned in Fig. 1.1. The target for this method is a thin plastic shell containing

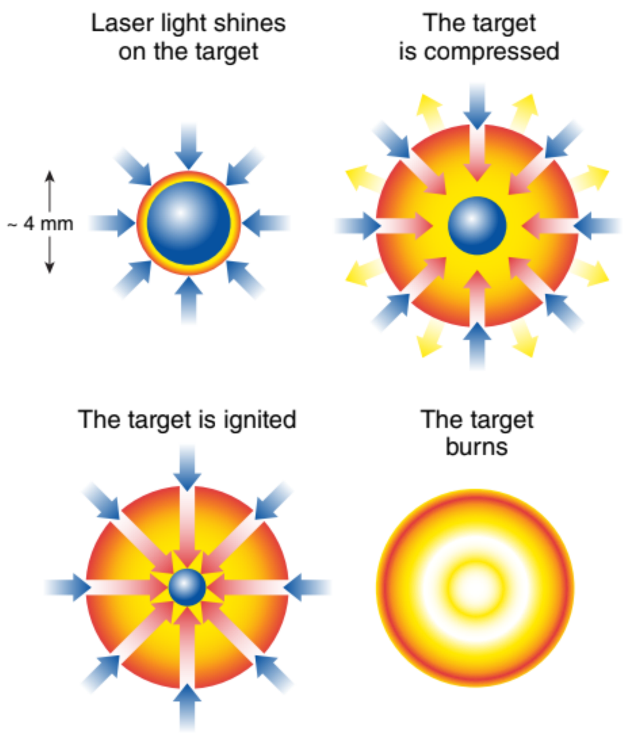


Figure 1.1: Inertial Confinement Fusion uses lasers to compress and implode the target. The imploded target undergoes the nuclear fusion process and burns the fuel. Reproduced from "Inertial Confinement Fusion: An Introduction The ENERGY of the STARS" from The University of Rochester's Laboratory for Laser Energetics.

deuterium and tritium that has been solidified into a solid layer inside the shell. The target is about 1 millimeter in diameter and suspended from a support. The surface of the capsule absorbs the laser light turning the surface into a very high temperature ionized gas known as a plasma which then expands to form a plasma corona around the target (Fig. 1.2). Figure 1.2 shows the incident laser light on the capsule causing it to ablated and the plasma to form on the exterior of the shell.

In the indirect-drive method, the target is suspended inside a cylindrical enclosure called a hohlraum, which is usually made from gold or some other heavy elements. Laser beams irradiate the interior of the hohlraum instead of the fuel capsule directly. The hohlraum walls emits x-ray radiation as they are heated by absorption of the laser light. The process is then the same in Fig. 1.1 where instead of laser light

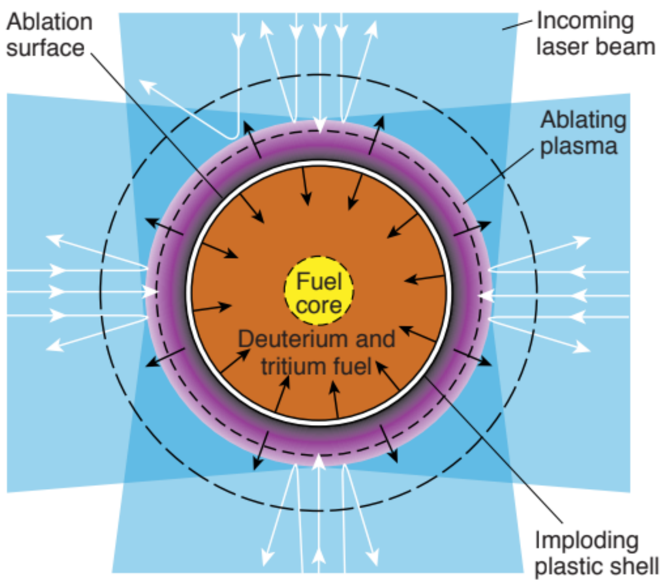


Figure 1.2: A laser-fusion target is compressed from the ablation of the flowing plasma. Reproduced from "Inertial Confinement Fusion: An Introduction The ENERGY of the STARS" from The University of Rochester's Laboratory for Laser Energetics

driving the process the x-rays cause the target to be compressed and heated from ablation forming a plasma corona. The advantage of indirect-drive is it leads to a more uniform irradiation of the fuel capsule because the cylindrical enclosure makes the x-rays more isotropic. In direct-drive it requires a special effort to make the incident light as uniform as possible. There is a disadvantage to indirect-drive in that there is an efficiency loss for the conversion of laser light to x-rays. As a result, only a fraction of the total energy reaches the fuel capsule.

In both direct-drive and indirect-drive methods a plasma is formed on the exterior of the fuel capsules. The plasma can also cause the laser energy to be poorly absorbed by the fuel capsule in both methods due to instabilities of the plasma that occur due to high intensity of the laser light. There are two main instabilities of concern: Stimulated Raman Scattering (SRS) and Stimulated Brillouin Scattering (SBS).

Stimulated Raman scattering is a type of instability that involves the coupling of a large amplitude light wave to a scattered light wave and a electron plasma wave. An electron plasma wave is a electrostatic longitudinal wave that oscillates in a plasma. The process of how stimulated Raman scattering works is an incident light wave propagating through a plasma where the density is rippled with amplitude δn. Due to the oscillation of the electrons in the electric field light wave with velocity v_{osc}, an electrical current is generated $\delta y = -e\delta n v_{osc}$. The transverse current will generate a scattered light wave which will interfere with the incident light to create a variation in the intensity of the electric field. Intensity variations act like a varying pressure that pushes on the plasma generating a density fluctuation if the wave numbers and frequencies of the perturbation are properly matched. They are properly matched if they meet the frequency and wave number criteria of resonant decay described by $\omega_o = \omega_s + \omega$ and $k_o = k_s + k$ where ω_o and k_o are the incident light wave frequency and wave number, ω_s and k_s are the scattered wave frequency and wave number, and ω and k are the ion acoustic wave frequency and wave number. The small fluctuation in density is enhanced leading to an enhanced transverse current that increases the strength of the scattered light, and so on. This is a feedback loop which causes instability.

Stimulated Brillouin scattering is similar to stimulated Raman scattering in that it involves the coupling of a large amplitude light wave to a scattered light wave and a density perturbation. In stimulated Brillouin scattering, the density perturbation is associated with an ion acoustic wave instead of electron plasma wave. An ion acoustic wave is an electrostatic longitudinal ion wave that oscillates at a low frequency. The frequency of ion acoustic waves is connected to the wave number by $\omega = kc_s(1 + M)$, where M is the mach number of plasma flow.

Both of these instabilities are problematic for the fusion process because energy can be lost from the incident laser (a coupling loss) and is scattered (a symmetry problem). Therefore there has been research into reducing the affects of these instabilities on approach to mitigate.

1.2 Motivation

Cross-beam energy transfer (CBET) is a special case of SBS [1] that occurs when multiple EM waves (e.g. laser beam) of overlap in a plasma. The CBET mechanism, as it applies to direct-drive ICF, is illustrated in Figs. 1.3(a) and 1.3(b) figures reproduced from [2]. These show simulations using the "laser plasma simulation environment" (*LPSE*) code that solves wave equations describing CBET as described later in Secs. 2. Two crossing EM beams labeled ("0" and "1") propagate through a radially symmetric (about the origin $x = y = 0$) plasma with a density profile that is fixed in time and decreases exponentially with radius and a plasma flow velocity, also fixed, directed radially outward and increasing linearly with radius (not shown). Figure 1.3(a) shows the entire simulated region, which is reduced in scale ($80 \times 80\ \mu\mathrm{m}^2$) compared with OMEGA coronal plasmas, but still thousands of laser wavelengths across in each direction. The laser light has a wavelength of $\lambda_o = 0.351\mu m$ in vacuum. Figure 1.3(b) is a blowup of the white rectangular region shown in Fig. 1.3(a), but it is shown at an earlier time when CBET has not developed.

Contours of electric field intensity are shown. Beam "0" is incident from the left simulation boundary, while beam "1" is incident from the upper left. Both beams are polarized in the same direction (out of the plane shown), and the red circle indicates the location of the critical density $n_c = \frac{m_e \omega_o}{4\pi e^2}$, where ω_o is the frequency of the incident light and m_e and e are the electron mass and charge respectively. As can be seen, EM waves cannot propagate at electron plasma densities greater than the critical density. Ray trajectories are shown for each beam (blue and white lines, respectively). These have been computed by ray tracing chosen in a way to match the boundary conditions used in the *LPSE* full-wave solver. The ray trajectories define the local wave vectors $\mathbf{k}_0$ and $\mathbf{k}_1$ of both beams, such that the electric field looks like plane waves locally [e.g., at the point $\mathbf{r}_0$ where $E_o \sim exp(ik_o x - i\omega_0 t)$]. Where rays from both beams overlap, the electric field is the coherent sum (i.e., $E = E_o + E_1$) of the contribution

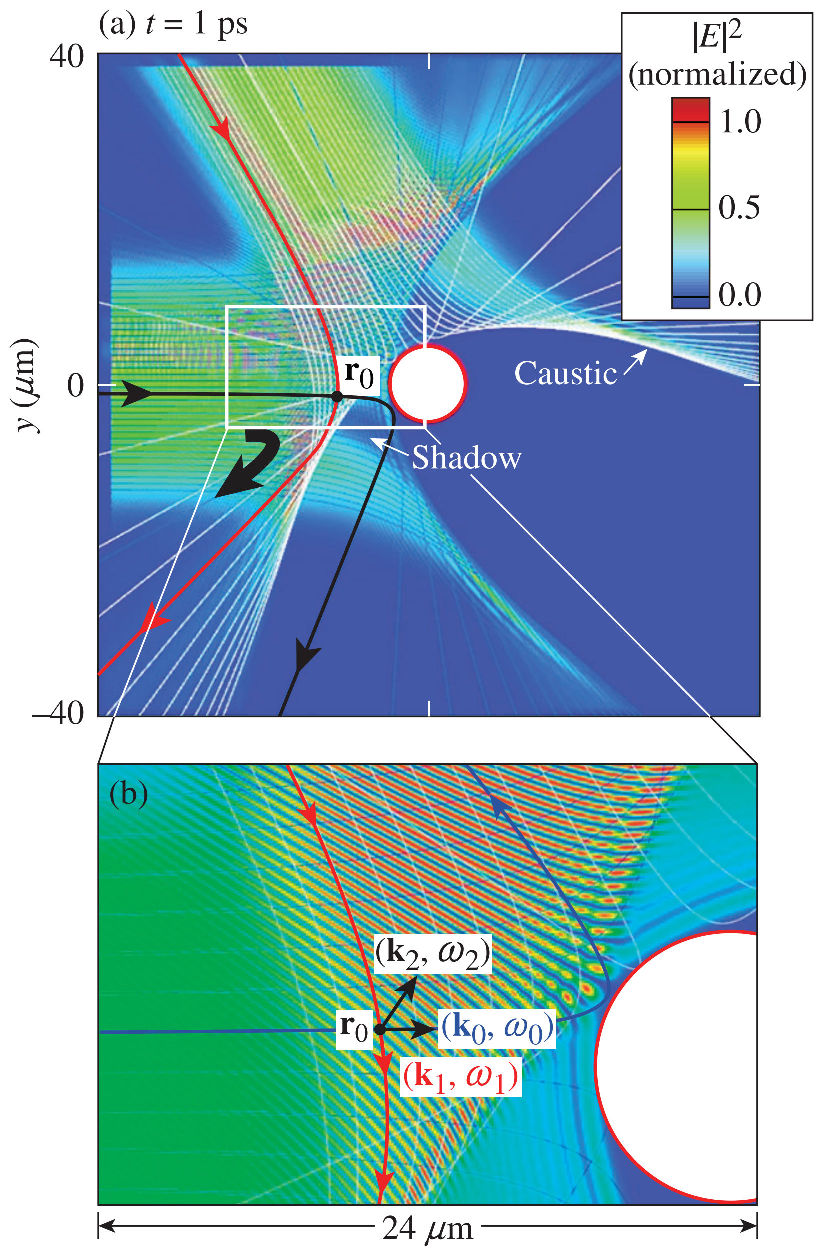


Figure 1.3: (a) and (b): A 2-D *LPSE* simulation of a small-scale preformed plasma profile, illustrating the geometry for cross-beam energy transfer. Snapshots of the electric field intensity are shown, normalized to its maximum value, for two overlapping laser beams (0 and 1). Snapshot (a) corresponds to a time 1 ps after injection of the beams from the boundary, while (b) is taken at 10 ps when CBET has fully developed. Snapshot (a) show the whole simulated region, while (b) is an enlargement of the region indicated by the white box in (a). The red circle gives the location of the critical density. The blue and white lines are ray trajectories for beams 0 and 1, respectively. The black arrow in (a) indicates the direction of energy transfer near the point $\mathbf{r}_0$, giving rise to a downstream shadow.

from each beam.

The wave vector $\mathbf{k}_2 = \mathbf{k}_0 - \mathbf{k}_1$ of the interference pattern in $|E^2|$ between beams 0 and 1 where they overlap at the position $\mathbf{r}_0$ is shown in Fig. 1.3(b). In general, the interference pattern will oscillate in time with frequency $\omega_2 = \omega_0 - \omega_1$, but in Figs. 1.3(a) and 1.3(b) the pattern is static ($\omega_2 = 0$) as both beams have the same frequency. A perturbation (grating) is generated in the plasma density with the same wave number $\mathbf{k}_2$ as the interference pattern by the ponderomotive force of the laser light $F_p \propto \nabla|E^2|$ [1].

The frequencies (ω_0 and ω_1) of the overlapping EM beams and the plasma flow velocity $\mathbf{U}_0$ control the proximity of the plasma response at the frequency ω_2, to a harmonic ponderomotive force with wave number $\mathbf{k}_2$, to a plasma resonance at the ion-acoustic-wave (IAW) frequency $\omega_{IAW}(\mathbf{k}_2)$. The ion acoustic wave dispersion relation is given by $\omega_{IAW} = kc_s(1+\hat{k}\frac{\mathbf{u}_o}{c_s})$, where c_s is the ion sound speed and $\hat{k} = \frac{\mathbf{k}}{|k|}$. If ω_2 is far from $\omega_{IAW}(\mathbf{k}_2)$, the pondermotive force will only produce a weak density perturbation. The IAW frequency is very small in comparison with the laser frequency; as a result, Doppler shifts caused by $\mathbf{U}$, play an important role in determining resonance. At (or near) IAW resonance, the density grating becomes enhanced so the coupled EM waves $[(\mathbf{k}_0, \omega_0)$, and $(\mathbf{k}_1, \omega_1)]$ and the IAW wave $(\mathbf{k}_2, \omega_2)$ become parametrically unstable in the same manner as its been described for SRS. This instability usually manifests itself as spatial amplification of the EM wave $(\mathbf{k}_1, \omega_1)$ and plasma density perturbation $(\mathbf{k}_2, \omega_2)$ at the expense of the EM pump $(\mathbf{k}_0, \omega_0)$. Substantial power can be transferred from the higher-frequency (pump) EM wave to the lower-frequency EM wave (where “higher” and “lower” refer to frequencies determined in the local reference frame where the plasma flow velocity $\mathbf{U}_0$ vanishes) [3].

Figure 1.3(a) shows the electric field intensities at times that are sufficiently late to cause CBET to develop and reach steady state ($t \gtrsim 10$ ps). CBET has caused laser energy to be exchanged from light directed along the highlighted blue ray to light following the highlighted white ray near the point $\mathbf{r}_0$. As a result, a “shadow” (dark blue region) can be seen along the blue ray downstream of point $\mathbf{r}_0$ because of the redirection of the light. The redirection occurs preferentially in the neighborhood of point $\mathbf{r}_0$ because the local Doppler shift is such to bring the grating between beams $(\mathbf{k}_0, \omega_0)$ and $(\mathbf{k}_1, \omega_1)$ into IAW resonance at this position. Just as in the *LPSE* simulation shown in Fig. 1.3(a), CBET is believed to preferentially transfer energy from the central portion of each laser beam to the outer portions (or "wings") in spherically symmetric direct-drive experiments, where there are many more overlapping beams.[4, 5] CBET is believed to significantly reduce absorption and drive, thereby influencing direct-drive target designs [6].

The reduction in absorption is sufficiently large such that mitigation schemes must be developed and implemented to either reduce, or eliminate, its effects. To date, most of the schemes that have been proposed either propose modifying the frequency of neighbouring beams, or increasing the temporal bandwidth of every beam. The temporal bandwidth is both challenging to achieve and expensive to implement. As a result, its proposed to modify the spatial, rather than temporal, structure of the laser beam. For example, the spatial phase can be modified in such a way as to introduce (orbital) angular momentum to the laser beam.

Orbital Angular Momentum (OAM) is a phenomena where the component of angular moment of a light beam is dependent on the light beams field spatial structure distribution instead of the polarization. These beams can be produced using a off axis spiral phase plate seen in Fig. 1.4 where the phase plate has a helical stepping giving a helical wave front and pattern to the incident laser light [7]. OAM has been used in applications such as rotation of particle in optical tweezers [8], as well as, being used in terabit optical communication using OAM in free space and fiber [9]. We are led to ask the question, what will OAM do in the application OAM light beams in ICF? It is speculated that it will be more difficult for beams to exchange energy if they carry angular momentum. To quantify this effect it first must be simulated. To describe the parametric instabilities resonance that causes CBET a simulation code using a wave solver type system is required and *LPSE* is such a code. However, it must first be modified to allow for angular momentum carrying beams to be injected into the simulation.

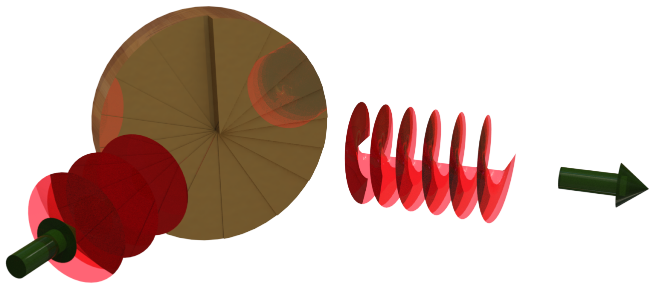


Figure 1.4: A off axis spiral phase plate with a incident light reflecting off the plate showing how the beam obtains a helical phase front [10].

1.3 Thesis Objectives

The objective of the thesis is to showcase the changes to the *LPSE* code to allow for higher order laser modes with orbital angular momentum (OAM) to propagate in the simulation and highlight the affects of higher order laser modes with OAM on the laser plasma instabilities that occur in the laser direct-drive fusion method. To complete the first objective, a discussion of the *LPSE* code, how it solves the Maxwell equations, and what physics are included in the simulation. The next objective is modifications to the code for the Higher order OAM modes to be injected and propagated as boundary conditions, as well as, the physics of how OAM and higher order laser modes are derived from fundamental equations. Then, how the code handles boundary conditions using Perfectly matched layers to attenuate the scattered lights on the boundaries and how the total field scattered field work is discussed and showing the results of implementing the boundary conditions in the code. The next objective is showing the crossing of two LG modes and the transfer of OAM. Lastly, the conclusion of the entire thing and future considerations on how these OAM beams can mitigate CBET SBS and other instabilities.

1.4 Thesis Outline

This thesis is organized as follows: Chapter 1 describes the need for a green energy source that is more sustainable to the current green sources that have very apparent flaws. Then it goes into the fusion energy route of green energy and how it works and the benefits compare to current fission energy and green energy sources. The discussion of the current challenges of fusion energy are talked about and the motivation behind the research. Chapter 2 goes into detail on the *LPSE* code used for the simulations done in this research and the equations this code solves. Chapter 3 talks about the boundary conditions on the code and how and why they were modified for this research. Chapter 4 shows the simulation results from the code modifications and the results from adding the higher order laser modes. Chapter 5 concludes the thesis and all the main points from it.

References

[1] W. L. Kruer, “The Physics of Laser Plasma Interactions,” in, ser. Frontiers in Physics, D. Pines (Ed.) Vol. 73, Redwood City, CA: Addison-Wesley, 1988.

[2] J. Myatt, J. Shaw, R. Follett, D. Edgell, D. Froula, J. Palastro, and V. Goncharov, “Lpse: A 3-d wave-based model of cross-beam energy transfer in laser-irradiated plasmas,” *Journal of Computational Physics*, vol. 399, p. 108 916, Sep. 2019. DOI: 10.1016/j.jcp.2019.108916.

[3] W. L. Kruer, S. C. Wilks, B. B. Afeyan, and R. K. Kirkwood, “Energy transfer between crossing laser beams,” *Phys. Plasmas*, vol. 3, pp. 382–385, 1996. DOI: 10.1063/1.871863.

[4] D. H. Edgell, W. Seka, J. A. Delettrez, R. S. Craxton, V. N. Goncharov, I. V. Igumenshchev, J. F. Myatt, A. V. Maximov, R. W. Short, T. C. Sangster, and R. E. Bahr, “Cross-beam energy transport in direct-drive implosion experiments,” *Bull. Am. Phys. Soc.*, vol. 54, p. 145, 2009.

[5] I. V. Igumenshchev, D. H. Edgell, V. N. Goncharov, J. A. Delttrez, A. V. Maximov, J. F. Myatt, W. Seka, and A. Shvydky, “Modeling crossed-beam energy transfer in implosion experiments on OMEGA,” *Bull. Am. Phys. Soc.*, vol. 54, p. 145, 2009.

[6] V. N. Goncharov, T. C. Sangster, R. Betti, T. R. Boehly, M. J. Bonino, T. J. B. Collins, R. S. Craxton, J. A. Delettrez, D. H. Edgell, R. Epstein, R. K. Follett, C. J. Forrest, D. H. Froula, V. Yu. Glebov, D. R. Harding, R. J. Henchen, S. X. Hu, I. V. Igumenshchev, R. Janezic, J. H. Kelly, T. J. Kessler, T. Z. Kosc, S. J. Loucks, J. A. Marozas, F. J. Marshall, A. V. Maximov, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, D. T. Michel, J. F. Myatt, R. Nora, P. B. Radha, S. P. Regan, W. Seka, W. T. Shmayda, R. W. Short, A. Shvydky, S. Skupsky, C. Stoeckl, B. Yaakobi, J. A. Frenje, M. Gatu-Johnson, R. D. Petrasso, and D. T. Casey, “Improving the hot-spot pressure and demonstrating ignition hydrodynamic equivalence in cryogenic deuterium–tritium implosions on omega,” *Physics of Plasmas*, vol. 21, no. 5, p. 056 315, 2014. DOI: 10.1063/1.4876618. eprint: https://doi.org/10.1063/1.4876618. [Online]. Available: https://doi.org/10.1063/1.4876618.

[7] A. Longman and R. Fedosejevs, “Mode conversion efficiency to laguerre-gaussian oam modes using spiral phase optics,” *Opt. Express*, vol. 25, no. 15, pp. 17 382–17 392, Jul. 2017. DOI: 10.1364/OE.25.017382. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-25-15-17382.

[8] H. He, M. E. J. Friese, N. R. Heckenberg, and H. Rubinsztein-Dunlop, “Direct observation of transfer of angular momentum to absorptive particles from a laser beam with a phase singularity,” *Phys. Rev. Lett.*, vol. 75, pp. 826–829, 5 Jul. 1995. DOI: 10.1103/PhysRevLett.75.826. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.75.826.

[9] J. Wang, J.-Y. Yang, I. Fazal, N. Ahmed, Y. Yan, H. HUANG, Y. Ren, Y. Yue, S. Dolinar, M. Tur, and A. Willner, “Terabit free-space data transmission employing orbital angular momentum multiplexing,” *Nature Photonics*, vol. 6, pp. 488–496, Jul. 2012. DOI: 10.1038/nphoton.2012.138.

[10] A. Longman, “Under-dense laser-plasma interactions in relativistic optical vortices,” pp. 171–175, 2020.

Chapter 2

The *LPSE* laser-plasma simulation code

2.1 Introduction

The *LPSE* code system [2] is particularly suitable for the investigation of the non-linear behaviour of OAM beams in plasma. There are several reasons for this suitability: The first is that investigations of OAM interactions require the simulation volume to be three dimensional. While three dimensional particle-in-cell simulations are possible, they are very expensive computationally. As a result, most practical PIC simulations are performed in 1 or 2 spatial dimensions. The *LPSE* code adopts a "reduced" description of the plasma which is less computationally intensive than solving the Vlasov-Maxwell or Fokker-Planck-Maxwell system of equations. It does have the disadvantage that the plasma nonlinearities are approximated and most kinetic effects are absent. A further advantage of *LPSE* is that the boundary conditions have been designed to be flexible such that arbitrary incident light beams can be injected from any boundary. Often, plasma simulation codes adopt very simplified boundary conditions (e.g., plane waves, or Gaussian beams), or make assumptions regarding the direction of light propagation (e.g., pF3D) [11].

As a consequence of the reduced plasma description, the *LPSE* adopts the use a various modules that can be enabled so as to describe different physical process. For example, there is a model for stimulated Raman scattering [12], a module for the two-plasmon decay instability [13] and another for stimulated Brillouin scattering. [2] Below, we describe the equations that are solved in the SBS module, as these are sufficient to describe cross-beam energy transfer. We also give an outline of the operation of the code that solves these equations in sufficient detail that it will aid future developers.

2.2 Equations solved in the *LPSE* CBET model

The *LPSE* model of SBS couples the time-enveloped Maxwell's equations with a fluid-moment model of the low-frequency plasma response. Recall that in free space the Maxwell equations are (in cgs units):

$$\nabla \cdot \mathbf{E} = 4\pi\rho \tag{2.1}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{2.2}$$

$$\nabla \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{B}}{\partial t} \tag{2.3}$$

$$\nabla \times \mathbf{B} = \frac{4\pi}{c}\mathbf{J} + \frac{1}{c}\frac{\partial \mathbf{E}}{\partial t} \tag{2.4}$$

where $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic field, ρ, $\mathbf{J}$, and c are the charge density and current density, and the speed of light in vacuum, respectfully.

The Vlasov-Maxwell system of equations combines the above Maxwell equations and a Vlasov equation for each plasma species:.

$$\frac{\partial f_s(\mathbf{x},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}} f_s + \frac{q_s}{m_s}(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B})\cdot\nabla_{\mathbf{v}} f_s = 0, \tag{2.5}$$

where f_s is the probability distribution function of particles in a plasma, the sub script s denotes the particle species (e - i), $\mathbf{x}$ and $\mathbf{v}$ subscripts denotes dependence on either position or velocity, q_s is the charge and m_s is the mass. The Vlasov equation and Maxwell equations form a closed self-consistent system as the charge and current density can be written in terms of the particle distribution functions:

$$\rho = \sum_s q_s \int d\mathbf{v} f_s, \qquad \mathbf{J} = \sum_s q_s \int d\mathbf{v}\mathbf{v} f_s \tag{2.6}$$

Reduced models are obtained by taking velocity moments of the Vlasov equation. The first three are:

$$\frac{\partial n_s}{\partial t} + \frac{\partial}{\partial \mathbf{x}} \cdot (n_s \mathbf{u_s}) = 0 \tag{2.7}$$

$$n_s \frac{\partial \mathbf{u_s}}{\partial t} + n_s \mathbf{u_s} \cdot \frac{\partial \mathbf{u_s}}{\partial \mathbf{x}} = \frac{n_s q_s}{m_s} \left(\mathbf{E} + \frac{\mathbf{u} \times \mathbf{B}}{c} \right) - \frac{1}{m_s} \frac{\partial p_s}{\partial \mathbf{x}} \tag{2.8}$$

$$\frac{\partial p_s}{\partial t} + u_s \frac{\partial p_s}{\partial x} + 3 p_s \frac{\partial u_s}{\partial x} + 2 \frac{\partial \mathbf{Q_s}}{\partial x} = 0, \tag{2.9}$$

where n_s is the density of the species, u_s is the mean velocity of the species, q_s is the charge, p_s is the is the pressure, and $Q = \frac{m}{2} \int (v - u)^3 f \, dv$ is the heat flux. These equations are used to model a plasma as a fluid when the system is closed by writing either Q_s or P_s in terms of density on assumption of an iso-thermal or adiabatic equation of state.

In *LPSE*, the Maxwell's equations are solved for the real-valued electric field $\tilde{\mathbf{E}}$ in the plasma that is obtained by eliminating the magnetic field by first taking the curl of Faraday's equation:

$$\nabla \times (\nabla \times \mathbf{E}) = \nabla \times \left(-\frac{1}{c} \frac{\partial \mathbf{B}}{\partial t} \right) \tag{2.10}$$

The vector identity $\nabla \times (\nabla \times \mathbf{A}) = \nabla(\nabla \cdot \mathbf{A}) - \nabla^2 \mathbf{A}$ is applied and, noting that the time and space derivatives commute; it obtains:

$$\nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E} = -\frac{1}{c} \frac{\partial}{\partial t} (\nabla \times \mathbf{B}). \tag{2.11}$$

Next, Ampere's Law is substituted into the equation.

$$\nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E} = -\frac{1}{c}\frac{\partial}{\partial t}\left(\frac{4\pi}{c}\mathbf{J} + \frac{1}{c}\frac{\partial \mathbf{E}}{\partial t}\right) \tag{2.12}$$

The equation is rearranged and Eq.(2.13) is obtained.

$$\nabla^2 \tilde{\mathbf{E}} - \nabla(\nabla \cdot \tilde{\mathbf{E}}) - \frac{1}{c^2}\frac{\partial^2 \tilde{\mathbf{E}}}{\partial t^2} = \frac{4\pi}{c^2}\frac{\partial \tilde{\mathbf{J}}}{\partial t}. \tag{2.13}$$

Equation (2.13) is exactly equivalent to the usual first-order formulation of Maxwell's equations (given in terms of the divergence equations plus the Ampere-Maxwell and Faraday's laws) provided that the divergence equations are satisfied at the initial time. If the magnetic field **B** is required, it can be recovered from **E** once its solution is known.

The major approximation that is applied to simplify the plasma equations is time-enveloping. Physically, this approximation takes advantage of the fact that the ion-acoustic waves in the plasma have frequencies that are very much smaller that that of the laser light. As a result, the SBS scattered light is only shifted very slightly in frequency from that of the incident light $\Delta\omega/\omega_0 \ll 1$. This is also true in the presence of applied temporal bandwidth schemes [14], and those proposed for the future [15]. The Maxwell's equations have to be solved, not in complete generality, but for only a very narrow range of frequencies that are close to those of the laser light ω_o. In this way, one can write $\tilde{\mathbf{E}} = 1/2[\mathbf{E}(\mathbf{x},t)\exp(-i\omega_0 t) + c.c.]$ (and similarly for the plasma current $\tilde{\mathbf{J}}$), with the assumption that the complex-valued envelope function [e.g., $\mathbf{E}(\mathbf{x},t)$] is a very slowly varying function of time in comparison with the fast phase factor. Assuming a linear relationship between the plasma current and the laser electric field envelopes, we have:

$$\mathbf{J}(\omega_0;\mathbf{x},t) = \overline{\overline{\sigma}}(\omega_0;\mathbf{x},t) \cdot \mathbf{E}(\mathbf{x},t), \tag{2.14}$$

where (for unmagnetized plasma) the electrical conductivity tensor $\overline{\overline{\sigma}}$ [16] at the frequency ω_0 is given by

$$\overline{\overline{\sigma}} = \sigma \mathbf{I}, \quad \text{where} \quad \sigma = \frac{i\omega_{\mathrm{pe}}^2(\mathbf{x}, t)}{4\pi(\omega_0 + i\nu_{\mathrm{ei}})}. \tag{2.15}$$

The scalar conductivity σ is given in terms of the electron plasma frequency $\omega_{\mathrm{pe}}^2 = 4\pi n_{\mathrm{e}} e^2/m_{\mathrm{e}}$, where e is the elementary charge, and m_{e} is the electron mass. The quantity $\nu_{\mathrm{ei}} = 4\sqrt{2\pi} n_{\mathrm{i}} Z_{\mathrm{i}}^2 e^4 \Lambda/(3m_{\mathrm{e}}^{1/2} T_{\mathrm{e}}^{3/2})$ is the electron–ion collision frequency, n_{i} is the ion number density, Z_{i} is the ion charge state, Λ is the Coulomb logarithm, and T_{e} is the electron temperature.

Inserting the Ohms's law [Eqs. (2.14) and (2.15)] into Eq. (2.13) and applying the envelope approximation $\mid \partial_t \mathbf{E} \mid \ll \mid \omega_0 \mathbf{E} \mid$, Eq. (2.13) becomes

$$\frac{2i\omega_0}{c^2} \frac{\partial}{\partial t} \mathbf{E} + \nabla^2 \mathbf{E} - \nabla(\nabla \cdot \mathbf{E}) + \frac{\omega_0^2}{c^2} \epsilon(\omega_0; \mathbf{x}, t) \mathbf{E} = 0, \tag{2.16}$$

where the plasma dielectric function $\epsilon = 1 + i\, 4\pi\sigma/\omega_0$ has been introduced:

$$\epsilon(\omega_0; \mathbf{x}, t) = 1 - \frac{\omega_{\mathrm{pe}}^2(\mathbf{x}, t)}{\omega_0(\omega_0 + i\nu_{\mathrm{ei}})}. \tag{2.17}$$

The plasma dielectric function is related to the index of refraction for EM waves in the plasma n ($\equiv ck/\omega_0$) by $n(\mathbf{x}, t) = \sqrt{\epsilon(\omega_0; \mathbf{x}, t)}$ [16], where c is the speed of light in vacuum and k is its wave number.

Closure of the system of Maxwell and plasma equations take advantage of the fact that the electron plasma number density n_{e} is slowly varying and can be assumed to be quasineutral ($n_{\mathrm{e}} \simeq n_{\mathrm{i}}/Z_{\mathrm{i}}$).The *LPSE* SBS model assumes that the total (low frequency) density is a combination of the inhomogeneous plasma plume, or corona, and the variations associated with small amplitude ion acoustic waves that are associated with the SBS process. As such, we define a "static," spatially inhomogeneous part $n_0(\mathbf{x})$ and a small ($\delta n/n_0 \ll 1$) time-dependent perturbation δn that is responsible for SBS (and hence CBET): $n_{\mathrm{e}}(\mathbf{x}, t) = n_0(\mathbf{x}, \tau) + \delta n(\mathbf{x}, t)$, respectively. The static part n_0 (and the associated flow velocity $\mathbf{U}_0$) should satisfy the steady-state plasma hydrodynamic equations. These profiles could be imported from a radiationhydrodynamics code for example. This separation of the plasma density, as the sum of two pieces, implies that the plasma dielectric function is also similarly decomposed: $\epsilon = \epsilon^{(0)} + \delta\chi_{\rm e}$, where the piece $\delta\chi_{\rm e}$ is responsible for the SBS . The dielectric function Eq. (2.17) becomes:

$$\begin{aligned}
\epsilon(\omega_0; \mathbf{x}, \tau, t) &= \epsilon^{(0)}(\omega_0; \mathbf{x}, \tau) + \delta\chi_{\rm e}(\omega_0; \mathbf{x}, \tau, t) \qquad (2.18)\\
&\simeq \left[1 - \frac{n_0(\mathbf{x}, \tau)}{n_{\rm c}} + i\frac{\nu_{\rm ei}}{\omega_0}\frac{n_0(\mathbf{x}, \tau)}{n_{\rm c}}\right] - \frac{\delta n(\mathbf{x}, t)}{n_{\rm c}}, \qquad (2.19)
\end{aligned}$$

where $n_{\rm c}$ $[= m_{\rm e}\omega_0^2/(4\pi e^2)]$ is the critical density, and the fact that both the EM wave damping rate $\nu_{\rm ei}/\omega_0 \ll 1$ and density perturbation are small $(\delta n/n_0 \ll 1)$ has been used. The plasma hydrodynamic flow velocity $\mathbf{U}(\mathbf{x}, \tau, t) = \mathbf{U}_0(\mathbf{x}, \tau) + \delta\mathbf{U}(\mathbf{x}, t)$ is decomposed in the same way. Again, n_0 and $\mathbf{U}_0$ are assumed to be steady state solutions (or solutions varying on a sufficiently slow time scale that their time dependence can be ignored).

Completion of the SBS model requires equations describing the plasma response to the laser electric field that provide the density perturbation $\delta n(\mathbf{x}, t)$. These equations are further simplified by noticing the separation of spatial scale between the large scale plasma flow and the short wavelength ion acoustic waves. The spatial gradients of the zero-order quantities are considered negligible with respect to derivatives of the perturbations (characteristic hydrodynamic length scales $L_{\rm n} = |\nabla \log(n_0)|^{-1}$ and $L_U = |\nabla \log(U)|^{-1} \gtrsim 100\ \mu$m, while the wavelengths associated with SBS IAW are submicron $L_{\rm IAW} \sim \lambda_0 \lesssim 1\ \mu$m).

Under these assumptions, the mass and momentum conservation equations (see Nicholson for example [17]) become:

$$\left[\frac{\partial}{\partial t} + \mathbf{U}_0(\mathbf{x}, \tau) \cdot \nabla\right]\left(\frac{\delta n}{n_0}\right) = -\mathcal{W}, \qquad (2.20)$$

$$\left[\frac{\partial}{\partial t} + \mathbf{U}_0(\mathbf{x}, \tau) \cdot \nabla + 2\hat{\nu}_{\rm ia}\right]\mathcal{W} = -\nabla^2\left[c_{\rm s}^2\left(\frac{\delta n}{n_0}\right) + \phi_{\rm p}\right], \qquad (2.21)$$

where $\mathcal{W} \equiv \nabla \cdot \delta\mathbf{U}$, $\phi_\mathrm{p} = Z_\mathrm{i} e^2 |\mathbf{E}|^2/(4m_\mathrm{e} m_\mathrm{i} \omega_0^2)$ is the ponderomotive potential [1], $c_\mathrm{s} = (Z_\mathrm{i} T_\mathrm{e}/m_\mathrm{i})^{1/2}(1+Z_\mathrm{i} T_\mathrm{e}/T_\mathrm{i})^{1/2}$ is the ion-acoustic sound speed, with m_i and T_i being the ion mass and temperature, respectively. The slow time scale τ, to be regarded as a parameter, has been included. A phenomenological term $2\hat{\nu}_\mathrm{ia}$ (a nonlocal operator in real space) has been added to the right-hand side (r.h.s) of Eq. (2.21) to reproduce Landau damping of IAW's.

The model equations solved by the *LPSE* SBS module are therefore: Equation (2.16) for the light wave, together with the expression for the plasma dielectric function in terms of the two components of plasma density [Eq. (2.18) and Eq. (2.19)], and the low frequency plasma equations [Eqs. (2.20) and (2.21)]. These equations are solved as an initial boundary value problem in time.

We do not give the numerical algorithm here (see [2]). We do however give an outline of the operation of the *LPSE* SBS module where we highlight the parts of the code that have been modified during this work.

2.3 Outline of the operation of the *LPSE* code

This section explains the general operation of the *LPSE* code in detail that is sufficient only to describe the areas of the code where modifications were required to permit the injection OAM beams into a *LPSE* simulation. The *LPSE* code contains modules and algorithm options that are not described here because they are not required for simulations of CBET. The actual changes made are given in Section 3.4 where the explicit form of the boundary conditions is described. While *LPSE* is documented to some degree, it is a research code developed by the Laboratory for Laser Energetics at the University of Rochester, and not a commercial code. As such, a significant amount of time examining and testing the source code was required to fully understand its operation and to isolate the parts that needed modification. Table 2.1 indicates which C++ [18] classes required modification (right column).

An instance of the `Lpse()` class called `lpse` is created by the main program

Table 2.1: Some important classes used in the *LPSE* code

Class	**Major purpose**	**Instance name**	**Modified in this work?**
Lpse	Manages simulation	lpse	no
ParameterManager	parses input deck	pm	no
ZakharovSolver	split step evolution of equations	zs	no
LightSolver	light manager	laser	yes
SchrodingerSolver3	FD solver for EM wave equations	ws	yes

[`main()`]. It is responsible for performing a simulation. This object performs three major tasks, using three methods which are called by `main()` in sequence:

- `lpse.initialize`
- `lpse.simulate`
- `lpse.finalize`

We examine each of these methods in the three subsections that follow:

2.3.1 The `lpse.initialize` method

The `lpse.initialize` method is responsible for setting up the simulation at some initial time (that may not necessarily be $t = 0$ if it is a continuation from a previous run) so that the initial value problem posed by Eqs. (2.16)-(2.21) can be advanced by stepping discretely in time (using the `lpse.simulate` method described in Subsection 2.3.2). As the initialization procedure is quite complicated, a flow chart is presented in Fig. 2.1 that should be referenced with Table 2.1. This table lists the main classes that are instantiated, the name of the instance, and the tasks that it performs.

The initialization part comprises a sequence function (method) calls:

- `lpse.resetSimulation()`
- `cpuTimer.start()`
- **pm.readParameters()**
- `lpse.installSignalHandler`
- **lpse.setupUtilityClasses**
- `lpse.setStartTime`
- `cpuTimer.stop()`

First the simulation is reset via the `resetSimulation` function. This reset all saved variables and arrays generated by the code so no errors arise from stopping the code mid-simulation then restarting the simulation from the beginning. The next function `cpu.Timer` starts a timer that times how long the setting up process takes for the *LPSE* code for diagnostic purposes. The `pm.readParameters` method is used to parse an ASCII text file that the user must provide (a `<runName>.parms` file). This file describes exactly how the simulation is to be performed. While this is an extremely important file for running the code and setting up a *LPSE* simulation, we do not describe it here as no changes were required to this (`ParameterManager`) class. Likewise, we do not describe the `installSignalHandler` function (it handles signals that are used to stop, pause, continue and abort a simulation).

The `setupUtilityClasses` method is important and it does require some explanation: Its role is to call the `setup` methods for the `ZakharovSolver` instance `zs`, the "laser manager" class `LightSolver` instance (`laser`), and for several others that

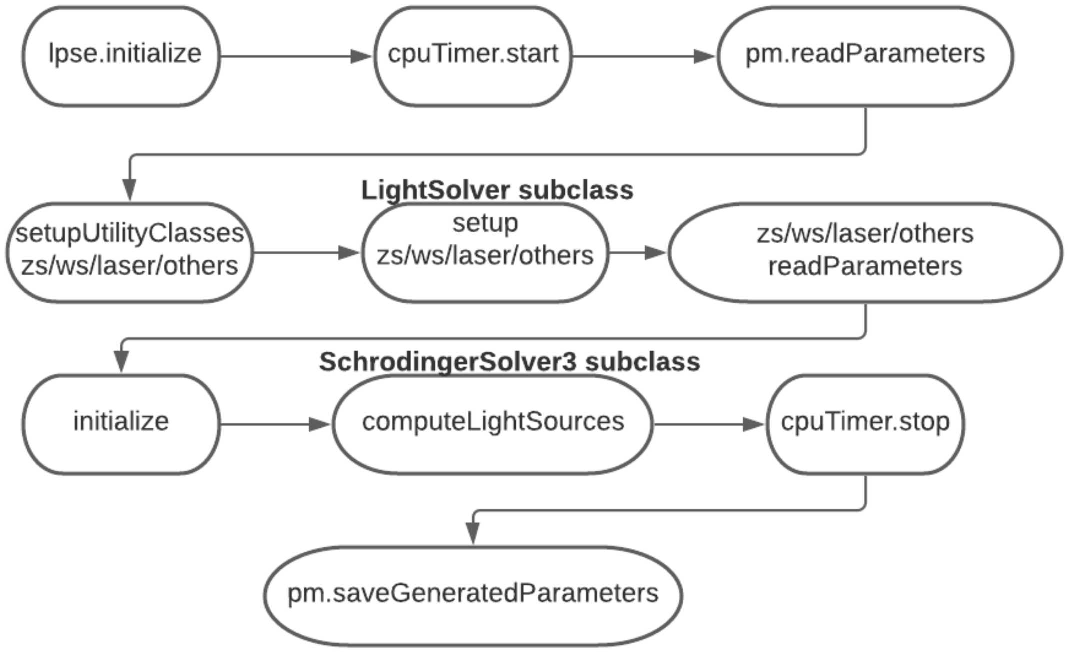


Figure 2.1: `lpse.initialize` code flowchart for the `LightSolver` physics module.

we do not describe (e.g., for other physics modules, synthetic diagnostics and instrumentation). Unfortunately, the `zs` object's class name is not particularly descriptive. The role of this class is to organize and coordinate the split-step time evolution of Eqs. (2.20)-(2.21) by calling solvers for the light wave equations [Eq. (2.20)] and the IAW equations [Eqs. (2.20) and (2.21)] in sequence. Our modifications require changes to be made to the solver for the EM waves only [Eq. (2.16)]. As shown in Fig. 2.1, the `LightSolver()` class requires modification to its `setup` method.

With reference to Fig. 2.1, the `laser.setup` method calls `laser.readParameters` and then `laser.initialize`. The `laser initialize` method calls the wave solver class' methods `ws.createPlanewaveSource` and `ws.setPlanewaveParameters`. Changes must be made to all of these.

The `laser.readParameters` method uses the parameter manager instance (`pm`) to parse the ASCII input file (`<runName>.parms`) in order to determine the type of laser light that is incident on the simulation boundaries. Before modification, this consists of any number of plane EM waves that are described by their intensity, direction, and polarization among other things (such as an optional Gaussian envelope shape function and temporal bandwidth). Once the information has been read in, `ws.createPlanewaveSources` allocates a one dimensional array of structures (structs) of type `Planewave_t` called `pw` (see Listing 2.1). Each element of this array represents one plane wave. The fields of each element of the array are filled in by `ws.setPlanewaveParameters`. This is done by using an intermediate "beams" array (array of `Beams_t`) that was generated by `laser.readParameters` that stores the parameters of each incident light sources characteristics as a element of the array.

Listing 2.1: planewave sources as defined in SchrodingerSolver3.h

```
struct Planewave_t {
   GridSide side;      //!< grid-side (0-5)
   XcFloat3 Kdir;      //!< direction of wavevector (with amplitude Ko)
   XcFloat3 Koff;      //!< offset of wavevector (with amplitude Ko)
   XcFloat3 ampl;      //!< plane wave amplitude
      float omega;     //!< 2*PI/frequency

   ... snipped code ....

}; // end Planewave_t
```

Once the array `pw` has been created, `light.setup` calls the `SchrodingerSolver3` methods `ws.computeLightSources`. So far as we are concerned, this is the most important step. It includes creating another one dimensional array (called `wi`). Each element of this array represents a "wave injector". It is struct of type `WaveInjector_t`. The definition of this WaveInjector type is shown schematically in Listing 2.2 below:

Listing 2.2: The type associated with wave injectors as defined in SchrodingerSolver3.h

```
protected: // wave injection functions
  struct WaveInjector_t {
    int i,j,k;             //!< local grid coordinates for this injector
    unsigned index;        //!< local grid index for this injector,
                           //!< mapped from (i,j,k)
    //
    // .... suppressed fields
    //

    XcComplex3 *src;       //!< total source term at some time for this grid point
  };
```

One injector (element of the array) is allocated by `computeLightSources` for every grid cell (indexed by the integers i, j, and k) that requires correction due to the presence of a light source that is incident on the simulation domain.

Listing 2.3: computeLightSources function in SchrodingerSolver3.cpp

```
void SchrodingerSolver3::computeLightSources(const float microTimeStep)
{

  // ... Snip!

  // Allocate memory for the waveInjector[] array
  //
  createWaveInjectors(nInjectors); // could be 0 injectors for this MPI process!
  memoryUsage += waveInjectorMemory;

  if (nInjectors > 0) {

    // Fill in the waveInjector[] array.

    int wiid= 0; // wave-injector id (incremented when assigned)

    for (int i=idxLower; i<=idxUpper; i++) {
      for (int j=0; j<Ny; j++) {
        for (int k=0; k<Nz; k++) {

          bool found=false;
          XcComplex3 src; // initialzied to zero

          if (hasPlanewaveSources) {
            src += addPlanewaveSource(startTime,i,j,k,found,...);
    }

          if (found) {

        // ...snip
        //
              // Initialize the injector and allocate memory
              //
              initWaveInjector(wiid,i,j,k,...);

              WaveInjector_t &wi= waveInjector[wiid];
              wi.src[0]= src; // at the start time!
              wiid++;

          } // endif(found)

        } // endfor k
      } // endfor j
    } // endfor i

  } // endif (nInjectors>0)


} // end computeLightSources()
```

The wave injectors are used to correct the finite-difference time step for the electric field on the TF-SF boundary as described later (in Sec. 3.3). The information required to make this correction is stored in the `src` field of each struct. It is computed by the method `addPlanewaveSource`.

The array of plane wave structs are used by `addPlanewaveSource` in the above `computeLightSources` method:

Listing 2.4: addPlanewaveSource function in SchrodingerSolver3.cpp

```
XcComplex3 SchrodingerSolver3::addPlanewaveSource(const float time, int i, int j, int k,
                                                  bool &found)
{
  XcComplex3 b; // The source starts at zero

   ... snip ...

  for (int sid=0; sid<numPlanewaves; sid++) {
    const Planewave_t &pw= planewave[sid]; // a reference for speed

    if (is1D) { // or 2D or 3D
      if (injectInsideY && injectInsideZ) {
        if (pw.side == SIDE_X_MIN) {
          //
          if (i == inject_min_i) {  // JFM i.e., we are on the TF side
            XcFloat3 Koff= pw.Koff; Koff.x= float(inject_min_i)/float(Nx1);
            XcFloat3 Kdir= pw.Kdir;

            const float x= (is1D) ? (float(i)/float(Nx1) - Koff.x)*gridSize.x : 0.0f;
            const float y= (is2D) ? (float(j)/float(Ny1) - Koff.y)*gridSize.y : 0.0f;
            const float z= (is3D) ? (float(k)/float(Nz1) - Koff.z)*gridSize.z : 0.0f;

            const XcFloat3 eu= UNIT3D(pw.ampl);     // unit polarization vector
            const XcFloat3 gu= CROSS3D(Kdir,eu);    // orth-norm basis for perp plane
            const XcFloat3 Rvec= XcFloat3(0,y,z);
            const XcFloat3 Rperp= DOT3D(eu,Rvec)*eu + DOT3D(gu,Rvec)*gu;

            const float weight= superGaussian(Rperp,pw)/EoScale(i,j,k);
            const float Ko_ijk= Ko(i,j,k);

            const XcFloat3 Rm1((x - h_xyz),y,z);
            const float KdotRm1= DOT3D(Ko_ijk*Kdir,Rm1);
            const XcComplex3 wPwsM1= weight*planewaveSource(KdotRm1,time,pw);

            const XcFloat3 R0((x),y,z);
            const float KdotR0= DOT3D(Ko_ijk*Kdir,R0);
            const XcComplex3 wPwsR0= weight*planewaveSource(KdotR0,time,pw);

            const XcFloat3 Rp1((x + h_xyz),y,z);
            const float KdotRp1= DOT3D(Ko_ijk*Kdir,Rp1);
            const XcComplex3 wPwsP1= weight*planewaveSource(KdotRp1,time,pw);

            const XcComplex I(0.0f,1.0f);
            XcComplex3 s;

            s.x=    (0.5f*I*h_xyz*Ko_ijk*Kdir.y)*wPwsP1.y
                  + (0.5f*I*h_xyz*Ko_ijk*Kdir.z)*wPwsP1.z
                  -      sqr(h_xyz*Ko_ijk*Kdir.x)*wPwsR0.x;

            s.y= wPwsM1.y +       (0.5f*I*h_xyz*Ko_ijk*Kdir.y)*wPwsP1.x
                          -            sqr(h_xyz*Ko_ijk*Kdir.y)*wPwsR0.y
                          - (sqr(h_xyz*Ko_ijk)*Kdir.y*Kdir.z)*wPwsR0.z;

            s.z= wPwsM1.z +       (0.5f*I*h_xyz*Ko_ijk*Kdir.z)*wPwsP1.x
                          -            sqr(h_xyz*Ko_ijk*Kdir.z)*wPwsR0.z
                          - (sqr(h_xyz*Ko_ijk)*Kdir.z*Kdir.y)*wPwsR0.y;

            b += s;                   // accumulate the contribution of this pw
            found= true;
          } // endif inject_min_i

          if (i == inject_min_i-1) {  // JFM i.e., we are on the SF side

          // corrections are two grid points deep on each boundary

         ... snip ....
```

```
            found= true;
          } // endif inject_min_i-1
          //
        } // endif(planewave::SIDE_X_MIN)

      } // endif
    } // endif(is1D)

  // similar things for all the other boundaries

  ... snip ...

  } // endfor(sid)

  return clipInjectionSources(b/sqr(h_xyz));
  //
} // end addPlanewaveSource()
```

As shown in the Listing for `computeLightSources`, a loop is performed over every grid cell (belonging to the MPI process!) and if it is determined that an injector is required at that grid point (i, j, k) the `addPlanewaveSource` function is called to compute the required correction (note that `addPlanewaveSource` is also used to test for the presence of an injector). This correction is stored in the `src` field for that particular injector. As each injector is initialized, the index of the injector array `wiid` is incremented.

The `addPlanewaveSource` function (see Listing) uses the previously defined `pw` array to compute this local correction (at grid index i, j, k). This function is called for every grid cell in the simulation. It sets the `bool` variable `found` to `false` if no injector is required because of the location of the grid point. It loops over every element (plane wave) in the `pw` array. If it finds at least one plane wave on a given boundary, then the `src` is computed by accumulating the contribution of each plane wave incident on that side. Each contribution requires the computation of the electric field vector of the plane wave evaluated over a finite difference stencil. This requires knowledge of its wave vector and other properties that are fields of each `Planewave_-t` struct (see Listing for `addPlanewaveSource`). This completes the `LightSolver` initialization.

Finally, `cpu.Stop` is called and the run time for the initialization is recorded and displayed in the terminal.

2.3.2 The `lpse.simulate` method

While no changes are required in this method, we describe it here in order to show how the previously initialized wave injectors are used to inject light into the simulation as time evolves. The `lpse.simulate` method performs the following steps in sequence (roughly):

```
starts the cpu timer
sets the end time
computes the number of time steps
carries out a while loop that evolves the equations
stops the timer
```

The simulation part of *LPSE* the starts the timer again with `cpu.start` to time the duration of the simulation phase of the code. Next the simulation stop time is set. Then the optimum time set is calculated and set. Then the while loop for the main simulation begins. In pseudo-code, the while loop looks like:

```
while (need another step) {
    zs.evolve(...)
    save data
    save checkpoint
    write feedback to std out
}
```

First `zs.evolve` is called which is the main function in the simulation loop which will be described in more detail. It coordinates the split-step time evolution of the system of equations to be solved. As this is quite complicated, it might prove useful to reference Fig. 2.2 that illustrates the control flow.

The ZakharovSolver instance `zs`'s method `zs.evolve()` calls the following:

```
zs.advanceLight
zs.advanceLW_fd
zs.advanceLW_fft
zs.advanceNelfAndDivV_fd
zs.advanceNelfAndDivV_fft
```

We see that `zs.evolve` function is split into five main functions (listed above): The second and third are not required in the current context (CBET calculations). The

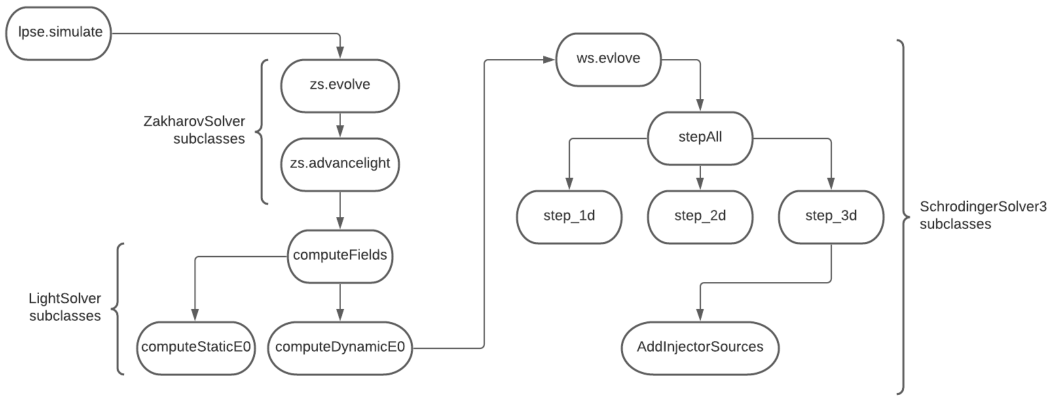


Figure 2.2: The `lpse.simulate` code flowchart for `LightSolver` physics module

EM equations are advanced in time by `advanceLight`, while the last two methods evolve the IAW equations. We only need to describe the `advanceLight` function.

With reference to Fig. 2.2, observe that `advanceLight` calls the function `computeFields` which gives the ability to solve Eq. (2.13) using different numerical algorithms. The desired function here is `computeDynamicE0`. The `computeDynamicE0` method solves the EM wave equations using a finite-difference method that is described in Myatt *et al.* [19]. As shown in Fig. 2.2, `computeDynamicE0` calls the `ws.evolve` method. The `ws.evolve` function the calls `step_all` (which is a wrapper for `step_3d`). This advances the finite-difference equation for Eq. (2.13) over one time step, absent any incident light sources. A subsequent call to the `addInjectorSources` method (see Listing) adds the correction required to inject the plane waves described by the `pw` array. It accomplishes this using the previously computed wave injectors. They contain the required corrections together with the grid indices (i, j, and k) where the correction needs to be applied (see Listing 2.5).

Listing 2.5: A skeleton describing the content of addInjectorSources.cpp

```
void SchrodingerSolver3::addInjectorSources(const float time, const float dt,
              const bool updateImaginaryPart)
{
  if (nWaveInjectors == 0) return;

  const float Cdt= dt/Cgroup;

  // This function is called twice per step_*d() loop. The first time,
  // updateImaginaryPart is true, the second its false.
  //
  XcComplex3 *Eo= Eo_field();

  for (int wiid=0; wiid<nWaveInjectors; wiid++) {

    WaveInjector_t &wi= waveInjector[wiid];
    XcComplex3 src; // zero

    switch (injectionMethod) {
      case NoBandwidth:
      {
        src= wi.src[0]; // 1 sample exists
      } break;
      // other cases suppressed....
      // ...

    } // end switch

    const unsigned ijk= wi.index;  // injector knows its grid index
    XcComplex3 dEo= Cdt*src;       // modification to E-field due to
                                   // sources

    if (updateImaginaryPart) {
      Eo[ijk]= XcComplex3(Eo[ijk].real(), Eo[ijk].imag() + dEo.real());
    } else {
      Eo[ijk]= XcComplex3(Eo[ijk].real() - dEo.imag(), Eo[ijk].imag());
    }

  } // endfor (wiid)
  //
} // end addInjectorSources()
```

Finally, the time step is incremented and checks to see if a exit signal is flagged to exit the loop. If not, then the data is saved and there is then feedback in the terminal and loops back. Once the simulation is over `cpu.stop` is called and the simulation time is calculated and output to the terminal.

2.3.3 The `lpse.finalize` method

The `lpse.finalize` part of the *LPSE* code saves the data and cleans up the simulation. First the code saves the last frame which can be used to restart the simulation if an error happened or a later continuation of the run is required. This is also done by saving the last checkpoint. After saving these parameters the job metrics are printed out, which would be the simulation start up, time the simulation run time, data saved, and more. Then all the files that were opened in the initialization part of the code are then closed and the heap is cleaned up.

The method `lpse.finalize` does (roughly)

```
save last frame
save checkpoint
print job metrics
close files and clean up the heap
```

References

[1] W. L. Kruer, "The Physics of Laser Plasma Interactions," in, ser. Frontiers in Physics, D. Pines (Ed.) Vol. 73, Redwood City, CA: Addison-Wesley, 1988.

[2] J. Myatt, J. Shaw, R. Follett, D. Edgell, D. Froula, J. Palastro, and V. Goncharov, "Lpse: A 3-d wave-based model of cross-beam energy transfer in laser-irradiated plasmas," *Journal of Computational Physics*, vol. 399, p. 108 916, Sep. 2019. DOI: 10.1016/j.jcp.2019.108916.

[11] R. L. Berger, C. H. Still, E. A. Williams, and A. B. Langdon, "On the dominant and subdominant behavior of stimulated Raman and Brillouin scattering driven by nonuniform laser beams," *Phys. Plasmas*, vol. 5, pp. 4337–4356, 1998. DOI: 10.1063/1.873171.

[12] R. K. Follett, J. G. Shaw, J. F. Myatt, H. Wen, D. H. Froula, and J. P. Palastro, "Thresholds of absolute two-plasmon-decay and stimulated raman scattering instabilities driven by multiple broadband lasers," *Physics of Plasmas*, vol. 28, no. 3, p. 032 103, 2021. DOI: 10.1063/5.0037869. eprint: https://doi.org/10.1063/5.0037869. [Online]. Available: https://doi.org/10.1063/5.0037869.

[13] J. F. Myatt, H. X. Vu, D. F. DuBois, D. A. Russell, J. Zhang, R. W. Short, and A. V. Maximov, "Mitigation of two-plasmon decay in direct-drive inertial confinement fusion through the manipulation of ion-acoustic and Langmuir wave damping," *Phys. Plasmas*, vol. 20, p. 052 705, 2013. DOI: 10.1063/1.4807036.

[14] S. Skupsky and R. S. Craxton, "Irradiation uniformity for high-compression laser-fusion experiments," *Phys. Plasmas*, vol. 6, no. 5, pp. 2157–2163, 1999. DOI: 10.1063/1.873501.

[15] J. W. Bates, J. F. Myatt, J. G. Shaw, R. K. Follett, J. L. Weaver, R. H. Lehmberg, and S. P. Obenschain, "Mitigation of cross-beam energy transfer in inertial-confinement-fusion plasmas with enhanced laser bandwidth," *Phys. Rev. E*, vol. 97, 061202(R), 2018. DOI: 10.1103/PhysRevE.97.061202.

[16] T. H. Stix, *Waves in Plasmas*, 2nd. New York: Springer-Verlag New York, Inc., 1992.

[17] D. R. Nicholson, "Chapter 7: Fluid equations," in *Introduction to plasma theory.* Krieger Pub. Co., 1992, pp. 129–132.

[18] B. Stroustrup, "A history of c++: 1979–1991," in *History of Programming Languages—II.* New York, NY, USA: Association for Computing Machinery, 1996, pp. 699–769, ISBN: 0201895021. [Online]. Available: https://doi.org/10.1145/234286.1057836.

[19] J. F. Myatt, R. K. Follett, J. G. Shaw, D. H. Edgell, D. H. Froula, I. V. Igumenschev, and V. N. Goncharov, "A wave-based model of cross-beam energy transfer in direct-drive inertial confinement fusion," *Phys. Plasmas*, vol. 24, p. 056 308, 2017. DOI: 10.1063/1.4982959.

Chapter 3

Implementation of orbital angular momentum boundary conditions in *LPSE*

3.1 Introduction

In general we envisage a wide variety of potential boundary schemes. One consideration, that we ignore for now, is that beam spots are matched to the target size, which is of the order of $\sim 1mm$. To improve the uniformity of irradiation of the target, laser beams first pass through an optical element that introduces randomized phase shifts on the beam wavefront. This gives rise to a speckled beam that is that is uniform when averaged over the short scale associated with the speckles. Fig. 3.2 shows an *LPSE* simulation of two such beams incident on an inhomogeneous plasma similar to that shown in Fig. 3.1.

The speckles are associated with intensities that are several times that of the average. Much can be learned about CBET and the effect of OAM on CBET by considering CBET between these speckles. This has two benefits: first, simulations can be performed in smaller plasma volumes, and secondly we can take advantage of known analytic formulas that approximately describes such speckles. The ability to use smaller simulations is important, as OAM studies must be performed in 3D. The analytical formulas are those corresponding to the Gaussian beam solutions to the

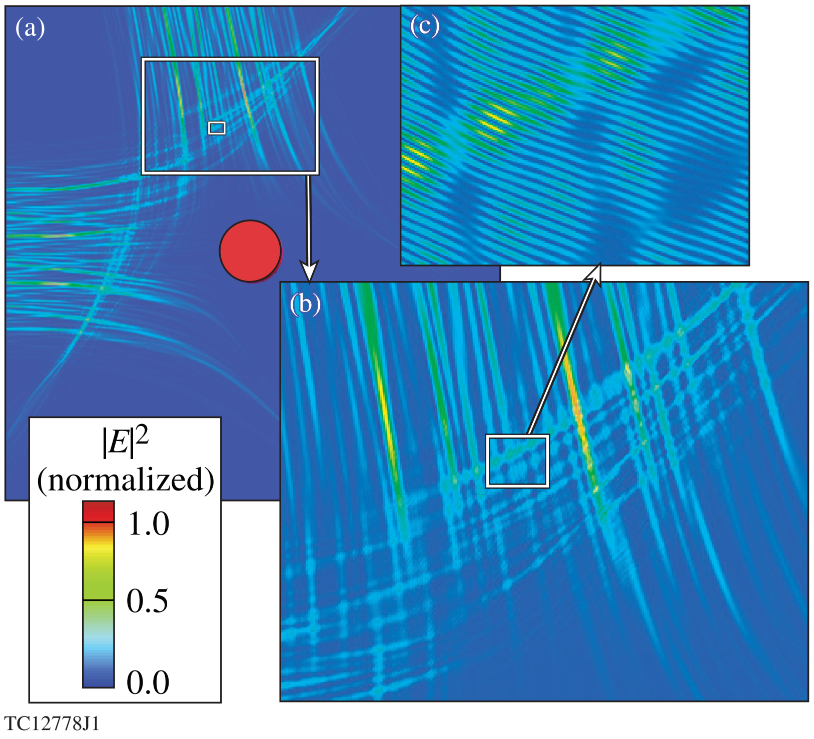


Figure 3.1: Results of a 2-D *LPSE* simulation that shows: (a) the intensity of two speckled laser beams entering from the left and upper boundaries. The speckles refract and are absorbed in a plasma density profile (not shown) that decreases radially from the center of the red circle (indicating the overdense region). Sub figures (b) and (c) are magnified regions, indicated by white boxes, of the original ($80\,\mu$m $\times$ $80\,\mu$m) computational domain.

paraxial EM wave equation, In the next section we describe these solutions, giving the analytic formulas in Sec 3.3 we describe how the knowledge of these formulas can be used to inject these beams from any simulation boundary in *LPSE*. Section 3.4 details the implementation details.

3.2 Gaussian beam solutions

The paraxial wave equation is an approximation equation. obtained from Eq. (2.13) for solutions that are beam-like. This permits the envelope approximation to be made in space. In other words, the temporal envelope $\mathbf{E}$ defined in sec. 2.2 is further decomposed as $E'(\mathbf{x}, t) = E(\mathbf{x}, t)e^{i\mathbf{k}_o\mathbf{x}}$ where $\frac{\mathbf{k}_o}{|k_o|}$ is the direction of propagation of the beam. As with temporal enveloping, the spatial dependence of the envelope is considered weak in comparison with the phase factor.

Applying the envelope approximation and expanding the operators in equation (2.16) results in :

$$\left(\frac{\partial^2 \mathbf{E}}{\partial x^2} + \frac{\partial^2 \mathbf{E}}{\partial y^2} + \frac{\partial^2 \mathbf{E}}{\partial z^2} + 2ik_0 \frac{\partial \mathbf{E}}{\partial z} - k_0^2 \mathbf{E}\right) + \frac{\omega_0^2}{c^2}\epsilon(\omega_0)\mathbf{E} = 0, \tag{3.1}$$

Since $\frac{\partial^2 \mathbf{E}}{\partial z^2} \ll \frac{\partial^2 \mathbf{E}}{\partial x^2}, \frac{\partial^2 \mathbf{E}}{\partial y^2}$ this becomes:

$$\left(\frac{\partial^2 \mathbf{E}}{\partial x^2} + \frac{\partial^2 \mathbf{E}}{\partial y^2} + 2ik_0 \frac{\partial \mathbf{E}}{\partial z} - k_0^2 \mathbf{E}\right) + \frac{\omega_0^2}{c^2}\epsilon(\omega_0)\mathbf{E} = 0, \tag{3.2}$$

For a plasma of constant (uniform) density and no collisional absorption ($\nu_{\text{ei}} = 0$), Eq. (3.2) becomes:

$$2ik_0 \frac{\partial}{\partial z}\mathbf{E} + \nabla_\perp^2 \mathbf{E} + \frac{\omega_0^2}{c^2}\left[\epsilon(\omega_0) - 1\right]\mathbf{E} = 0, \tag{3.3}$$

Where we have defined $\frac{\partial^2 \mathbf{E}}{\partial x^2} + \frac{\partial^2 \mathbf{E}}{\partial y^2} \equiv \nabla_\perp^2 \mathbf{E}$.

The fundamental Gaussian solution is obtained for Eq. (3.3) using standard methods and with the assumption of cylindrical symmetry around the z-axis defining the direction of propagation. The solution for plane polarization in the x direction, is:

$$\mathbf{E}(r, z) = \hat{x}E_0 \frac{w_0}{w(z)} \exp\left[-i\left(kz - \tan^{-1}\left(\frac{z}{z_o}\right)\right)\right] \times \tag{3.4}$$

$$\exp\left[-i\frac{k}{2R(z)}r^2\right] \exp\left[-\frac{r^2}{w(z)^2}\right], \tag{3.5}$$

where we have $r^2 = x^2 + y^2$, and the following definitions have been made:

$$w(z) = w_0\sqrt{1 + \left(\frac{z}{z_0}\right)^2}, \tag{3.6}$$

$$R(z) = z\left[1 + \left(\frac{z_0}{z}\right)^2\right]. \tag{3.7}$$

The quantity $z_0 = w_0^2\pi/\lambda$ is the Rayleigh length (shown as z_R in Fig. 3.2), and w_o is the beam waist parameter.

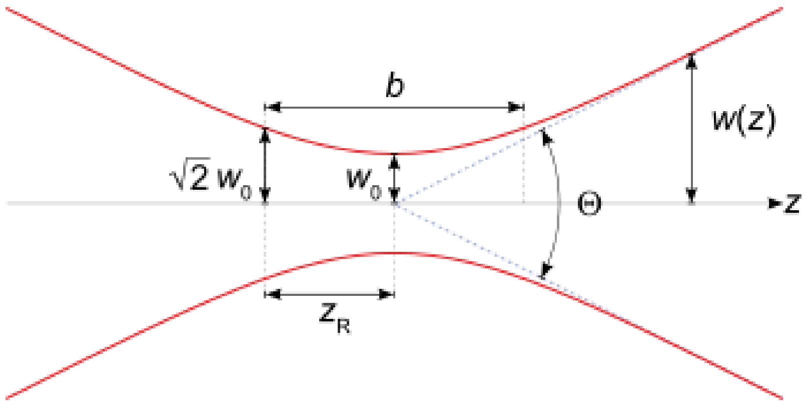


Figure 3.2: Illustration of the Gaussian beam parameters of waist w_0, width $w(z)$, and Rayleigh length z_R; the radius of curvature is not shown.

We can use *Cartesian coordinates* and apply the method of separation of variables to find further solutions that are not cylindrically symmetric. It is straightforward, but tedious so we do not reproduce it here. This results in the *Hermite-Gauss* (HG) modes:

$$\mathbf{E}_{m,n}(x,y,z) = E_0 \frac{w_0}{w(z)} H_m\left(\frac{\sqrt{2}x}{w(z)}\right) H_n\left(\frac{\sqrt{2}y}{w(z)}\right)$$
$$\times \exp\{-i[kz-(1+m+n)\phi]\} \exp\left[-i\frac{k(x^2+y^2)}{2R(z)}\right] \exp\left[-\frac{(x^2+y^2)}{w^2(z)}\right]\hat{x} \qquad (3.8)$$

where the $H_m(x)$ are the Hermite Polynomials [20], and $\phi = \tan^{-1}(\frac{z}{z_R})$. The Hermite polynomials can be computed using the Rodrigues formula [20]:

$$H_m(x) = (-1)^m \exp(x^2)\frac{d^m}{dx^m}\exp(-x^2). \qquad (3.9)$$

The first few are: $H_0 = 1$, $H_1 = 2x$, and $H_2 = 2(2x^2 - 1)$. In the above solution for $\mathbf{E}$ [Eq. (3.8)], notice that the polynomials, in each direction, are multiplied by a Gaussian factor [e.g, $H_m(x)\exp(-x^2)$ in the x direction]. Fig. 3.3 illustrates the first five of these Hermite-Gauss factors ($m = 1, 2, .., 5$). Notice also that the polarization of the HG beam solution given in Eq. (3.8) is arbitrarily given in the x direction ($\hat{x}$). Solutions can be plane polarized or elliptically polarized in the transverse plane. Unlike the case of non-paraxial beam solutions the polarization and spatial beam

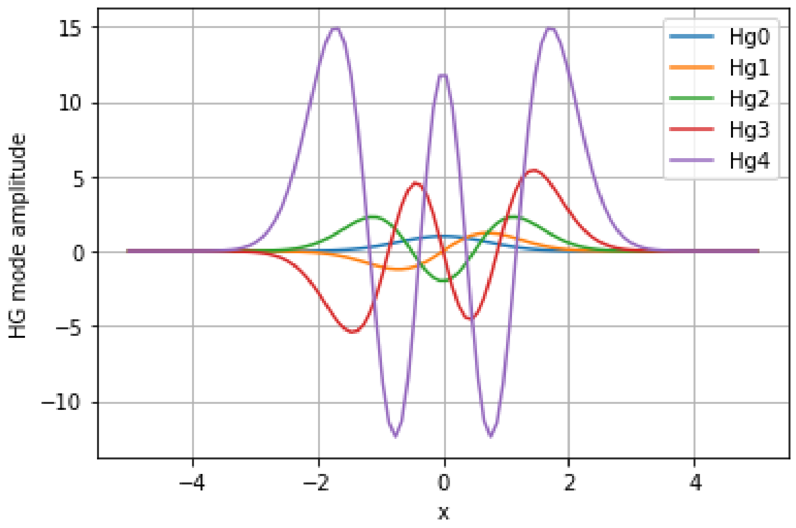


Figure 3.3: Lineouts of the first five Hermite-Gauss functions $n = 0 - 4$ (see text for details)

profile are independent (separable). The transverse to z electric field distribution can be investigated by plotting the function

$$H_n(\sqrt{2}x)H_m(\sqrt{2}y)\exp -(x^2+y^2) \tag{3.10}$$

as the distribution is invariant in z (i.e., under propagation) except for a scaling. – In the above, the coordinates x and y have been normalized to the beam waist w. Eq. (3.10) is plotted for various values of m and n in Fig. 3.4. The absolute value of the distribution [Eq. (3.10)] is shown in Fig. 3.5. Notice that the number of nodes in each coordinate direction is given by the mode number (m and n).

If we had used *polar coordinates* (r, θ) for the transverse plane instead of Cartesian, we would have found solutions to Eq. (3.3) of the following *Laguerre-Gauss* (LG) form:

$$\begin{gathered}\vec{E}_{p,l}(r,\theta,z) = E_0\frac{w_0}{w(z)}\left(\frac{\sqrt{2}r}{w(z)}\right)^{|l|}L_p^{|l|}\left(\frac{2r^2}{w^2(z)}\right)\\ \times\exp\{-i[\pm l\theta-(1+2p+|l|)\phi]\}\exp\left[-i\frac{kr^2}{2q(z)}\right]\exp\left[-\frac{r^2}{w^2(z)}\right]\hat{x}\end{gathered} \tag{3.11}$$

Here, L_p^l are the associated (generalized) Laguerre polynomials [20], which can be

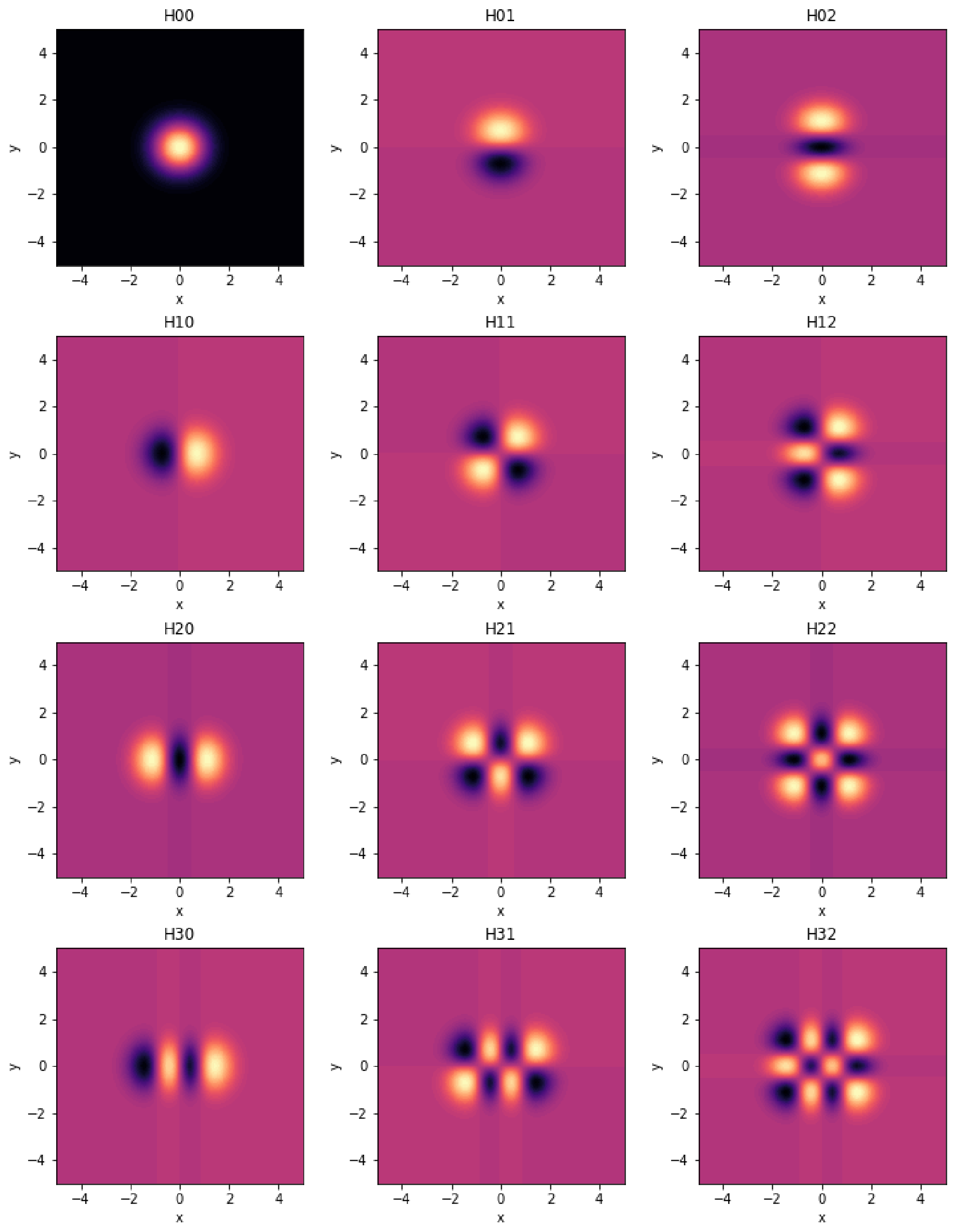


Figure 3.4: A matrix of plots showing the normalized electric field amplitude [the formula given in Eq. (3.10)] in the $x - y$ plane. The (zero based) row number sets n, while the column determines the value of m.

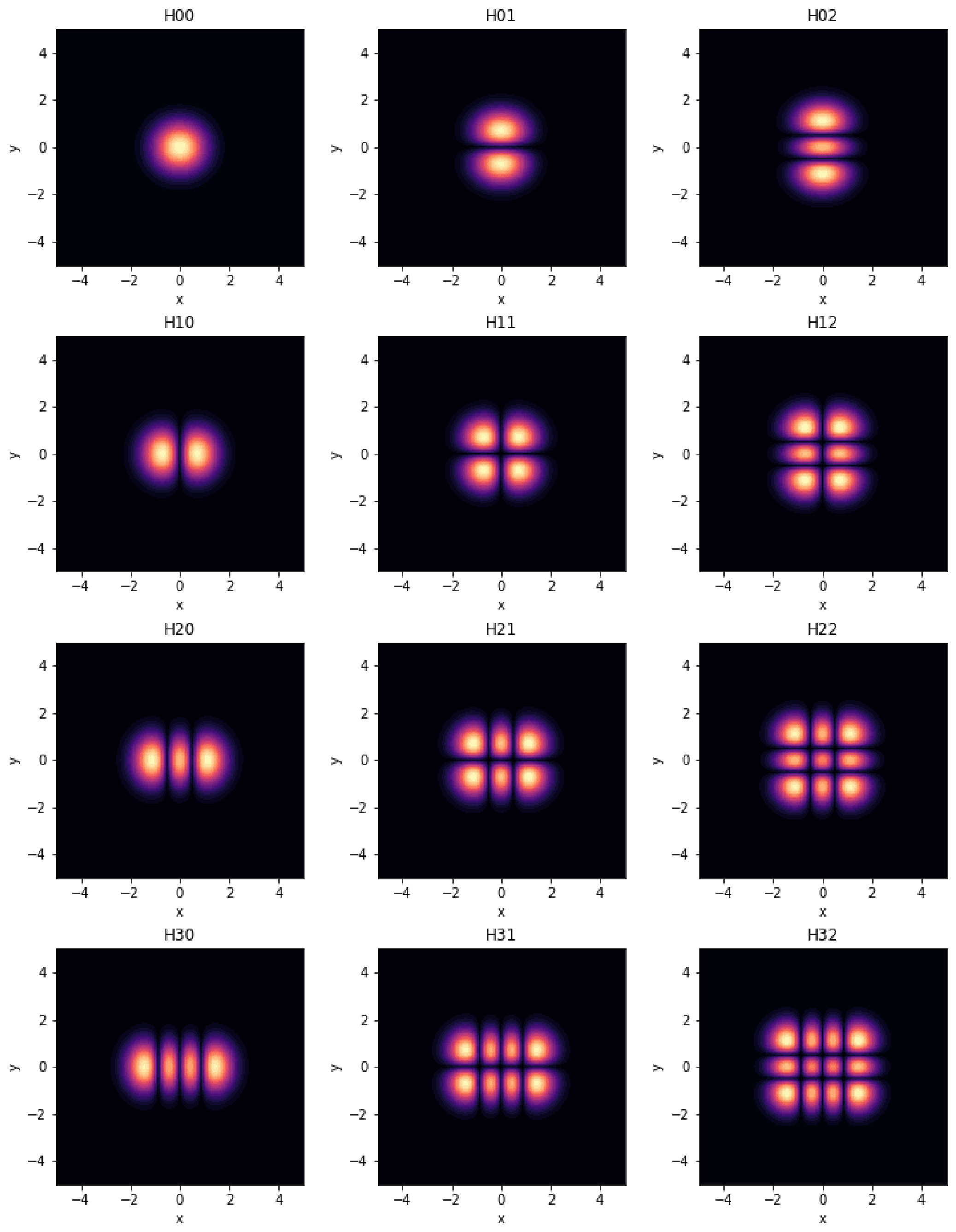


Figure 3.5: A matrix of plots showing the absolute value of the electric field amplitude [the modulus of the formula given in Eq. (3.10)] in the $x - y$ plane. The (zero based) row number sets n, while the column determines the value of m.

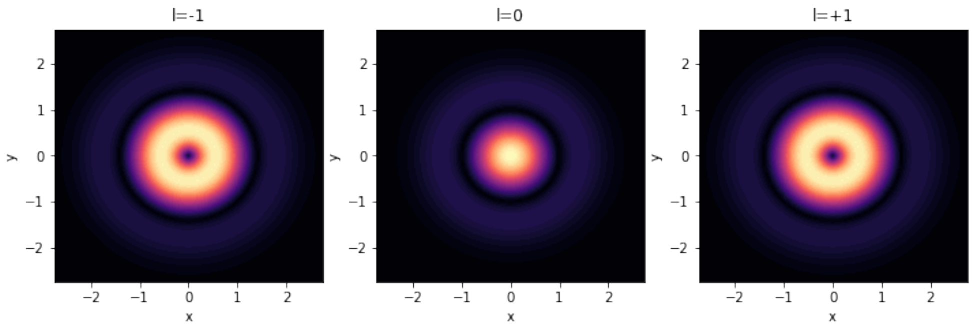


Figure 3.6: Normalized electric field intensity in the $x - y$ plane for $p = 1$ and (left to right) $l = -1$, 0, and +1.

computed by

$$L_p^l(x) = \frac{d^l}{dx^l}[L_{p+l}(x)](-1)^l, \quad \text{where} \quad L_p = \frac{(-1)^n}{n!} e^x \frac{d^p}{dx^p}(e^x x^p) \tag{3.12}$$

are the Laguerre polynomials [20]. The number of circular zeros is given by the integer p, while the number of azimuthal zeros (nodal lines) by the integer l. Some examples of the associated Laguerre polynomials of low order are:

$$L_0^l(x) = 1, \quad L_1^l = l + 1 - x, \quad \text{and} \quad L_2^l = \frac{1}{2}(l+1)(l+2) - (l+2)x + \frac{1}{2}x^2. \tag{3.13}$$

Again, as with the HG solutions, the transverse profile is invariant under propagation except for a scaling. It can therefore be investigated by examining the function

$$r^{|l|} L_p^{|l|}(r^2) \exp(-i[\pm l\theta]) \exp(-r^2). \tag{3.14}$$

Figure 3.6 shows the absolute value this function for $p = 1$ and $l = -1$, 0, and 1, while Fig. 3.7 shows the absolute value for $p = 2$ and various values of l.
The effect of the azimuthal phase factor, that depends on l, can investigate this by plotting the real part of Eq. (3.14). This is shown in Fig. 3.8 for $p = 2$ and $l = -2$, -1, 0, 1, and 2. Notice the angular nodal lines.

The following section describes how Eqs. (3.8) and (3.11) can be used to inject HG and LG beams into an *LPSE* simulation from any boundary.

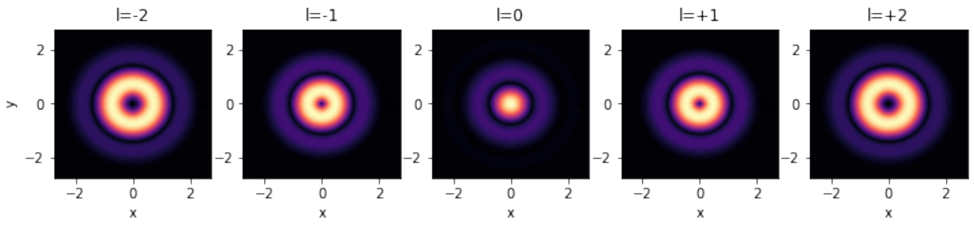


Figure 3.7: Absolute value of the function given in Eq. (3.14) for $p = 2$ and (left to right) $l = -2$, -1, 0, 1, and 2.

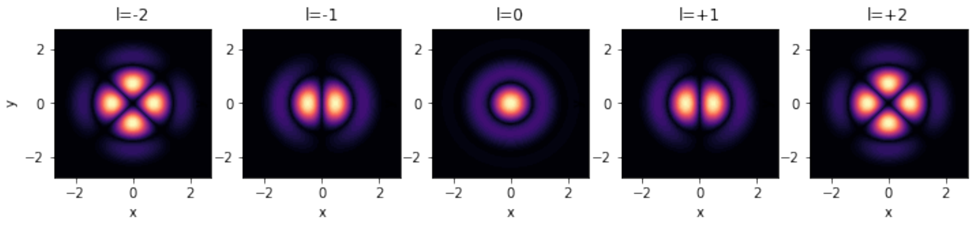
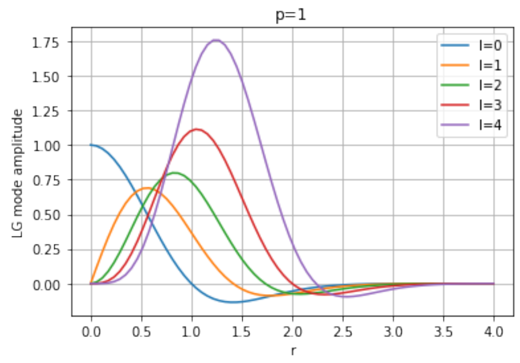


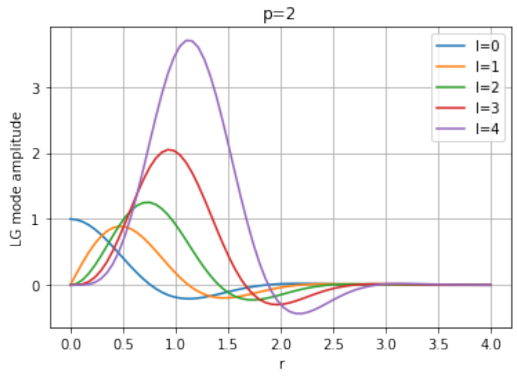
Figure 3.8: The real part of Eq. (3.14) for the value $p = 2$ and $l = -2, -1, 0, 1$, and 2.

3.3 Methods and Procedure

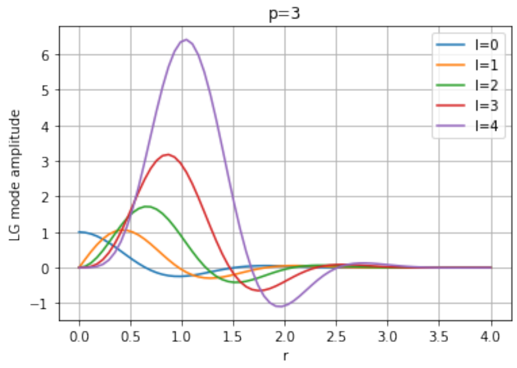
Figure 3.10 illustrates the treatment of the boundaries by way of a simple 1-D case. For illustration, this is a linear EM scattering problem only (i.e., there are no plasma perturbations $\delta N = \delta \mathcal{W} = 0$), ($\nu_{\mathrm{ei}} = 0$; $\epsilon_{\mathrm{I}}^{(0)} = 0$). A plane EM wave of the form $\mathbf{E}_{\mathrm{in}} = \hat{\mathbf{e}}_z E_0 \exp(ik_0 x)$, of wavelength $\lambda_0 = 0.351\,\mu\mathrm{m}$, is incident from the left and propagates through a plasma profile $n_0(x)$ that is non-zero at $x = x_{\mathrm{l}}$ ($= -3\,\mu$m) and increases linearly [$n_0(x) = \max\{0, n_{\mathrm{c}}(x - x_{\mathrm{l}})/L_{\mathrm{n}}\}$, with scale length $L_{\mathrm{n}} = 5.25\,\mu$m]. The scattering source (the EM wave cutoff at $n_{\mathrm{e}} = n_{\mathrm{c}}$) is contained within the total field (TF) region $\Omega_{TF} = \{x \mid x \in [x_{\mathrm{l}}, x_{\mathrm{r}}]\}$.The total field region is, in general, chosen to be large enough to contain all interactions. Both linear and nonlinear, that can generate scattered waves. In contrast, the scattered field regions. Ω_{sf}, exterior to Ω_{TF}, must be regions where the EM wave propagation is linear, such that the superposition principle is valid. In the scattered field region, the electric field that is computed is that of outgoing waves only. The difference between the total electric field and the scattered electric field being just what is incident. For example, the incident wave is



(a)



(b)



(c)

Figure 3.9: Simple lineouts showing the effect of increasing orbital angular momentum l for a given radial mode number p.

not seen in the left SF region $x \in [x_{\mathrm{L}}, x_{\mathrm{l}}]$ because here the algorithm solves for the scattered field only. This is accomplished simply by substituting $\mathbf{E} \rightarrow \mathbf{E}_{\mathrm{SF}}$ in the discretized wave equation (in the TF region $\mathbf{E} \rightarrow \mathbf{E}_{\mathrm{TF}}$ is substituted instead). When the finite-difference stencil (either for $\mathbf{E}_{\mathrm{TF}}$ or $\mathbf{E}_{\mathrm{SF}}$) straddles the TF–SF boundary at $x = x_{\mathrm{l}}$, the known form for $\mathbf{E}_{\mathrm{in}}$ (above) is used to write the difference equations entirely in terms of $\mathbf{E}_{\mathrm{TF}}$ or $\mathbf{E}_{\mathrm{SF}}$ as appropriate. Therefore a beam can be injected from a given boundary, using this formula, if its electric field is known in the neighbourhood of the TF-SF boundary. For example Eqs. (3.8) and (3.11) can be used to inject HG and LG beams respectively (after a suitable coordinate transform). In Fig. 3.11 The magnitude of the field in the TF region oscillates because of the interference between the incident wave and the wave reflected from the cutoff. The reflected wave

$\mathbf{E}_{\rm SF} \sim \hat{\mathbf{e}}_z E_0 \exp(-ik_0 x)$ is seen exiting the left SF region. No sources are present from the right, so all that is seen in the right SF region is an evanescent EM field.

Dirichlet conditions are imposed at the true computational boundary $\partial\Omega_c$ [$=\{x_{\rm L}, x_{\rm R}\}$], which, by themselves, would result in artificial reflections of the outgoing waves at $\partial\Omega_c$. To prevent these reflections, a perfectly matched layer (PML) absorbing layer is introduced that extends from the boundary $\partial\Omega_c$ into $\Omega_{\rm SF}$ several grid cells (schematically shown as the green-shaded region labeled "PML" in Fig. (3.10) [21]. We do not describe the implementation of the PML here.

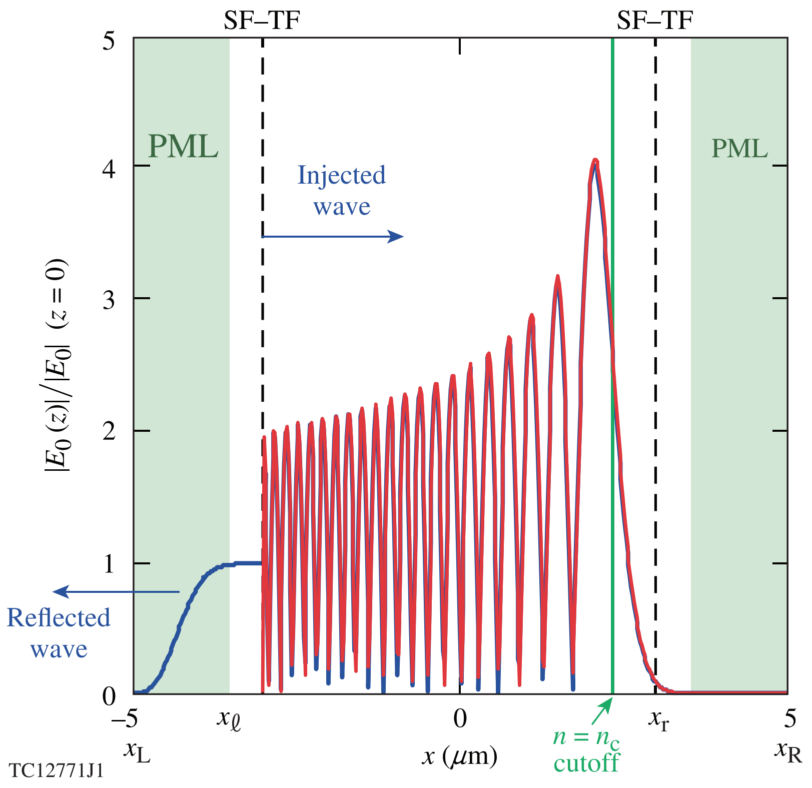


Figure 3.10: The linear 1-D scattering problem of a plane EM wave, incident from the left (indicated by the blue arrow), on a plasma slab whose density $n_0(x)$ increases linearly from vacuum (at $x = x_{\rm l}$) to include the wave turning point (solid green line labeled $n_{\rm e} = n_{\rm c}$). The problem is solved on $\Omega_{\rm c} = \{(x, t) \in [x_{\rm L}, x_{\rm R}] \times \mathbb{R}^+\}$, where $x_{\rm L}$, $x_{\rm R} = \pm 5$ μm, respectively. The vertical dashed lines indicate the TF-SF boundaries and the blue-green shaded regions on the far left and far right indicate the regions where the PML is applied.

As illustrated in Fig. (3.10) the situation is the same in three dimensions. The only complication being that in 3D Cartesian grid used by *LPSE*, six boundaries must be considered.

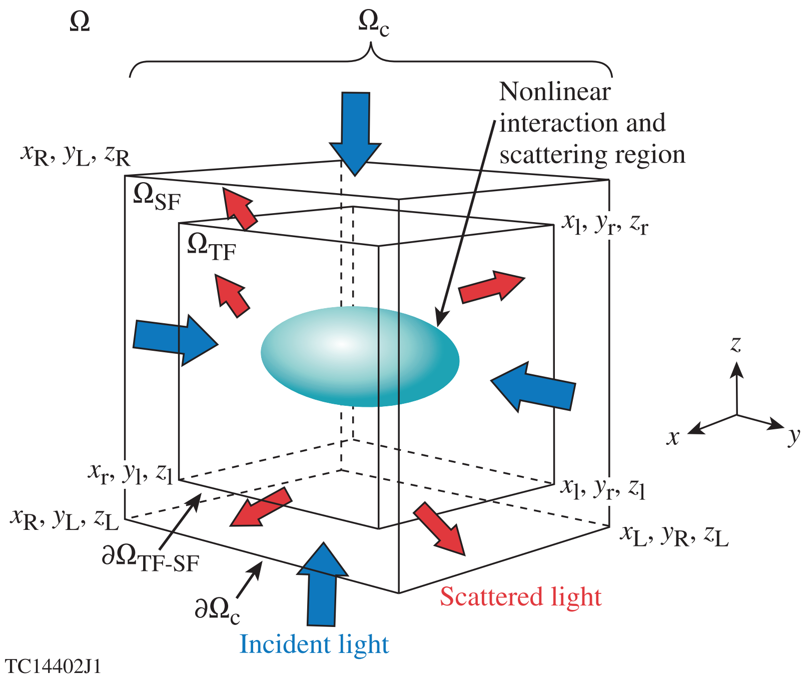


Figure 3.11: A schematic diagram showing the interior computational domain Ω_c, having the exterior boundary $\partial\Omega_c$, where the numerical problem is to be solved. The domain Ω_c is the union of a scattered-field region Ω_{SF} and a total-field region Ω_{TF}, which share the common total-field/scattered-field boundary $\partial\Omega_{\text{TF-SF}}$.

3.4 *LPSE* Code Modifications for LG/HG Boundary Conditions

This Section describes of the modifications made to the *LPSE* code to inject LG/HG laser modes from the boundary using wave injectors. The modifications replicate the `planewave` method described earlier. It is implemented for the LG/HG laser modes by adding a new structure that, depending on the beam type, will have different parameters, these are set in the framework built for the existing `planewave` boundary injection. This required modifications to the `lpse.initialize` method, described above in Ch. 3.

In the `lpse.initialize` method there are a sequence of functions called to. The main function requiring modification from this sequence is `lpse.setupUtilityClasses` (seen in the listing of main functions in Ch. 3): The `setupUtilityClasses` method calls the `setup` methods for the `ZakharovSolver` instance `zs`, the "laser manager" class `LightSolver` instance (`laser`), and for several others such as physics modules, synthetic diagnostics and instrumentation. The modifications require changes to be made to the solver for the EM waves only [Eq. (2.16)]. As shown in Fig. 2.1, the `LightSolver()` class requires modification to its `setup` method.

With reference to Fig. 2.1, the `laser.setup` method calls `laser.readParameters` where the first changes are. The first modification adds the ability to select the incident light type using an enumerated type "PlaneWave", "LaguerreGauss", or "HermiteGauss". A parameter named the `basis` which is a string that defines the matching incident light type. Then the intensity, direction, and polarization among other things (such as and optional Gaussian envelope shape function and temporal bandwidth) are defined for each mode type. Additional parameters are needed for the new modes defined by Eq.(3.11) for the LG mode and Eq. (3.8) for the HG mode. For the LG mode, the parameters `l` and `p` are added and the parameters `m` and `n` for the HG mode. These are the mode orders described above in this chapter. The modified `laser.readParameters` method uses the parameter manager instance (`pm`) to parse the ASCII input file (`<runName>.parms`) in order to determine the type of laser light that is incident on the simulation boundaries.

Listing 3.1: readparameters changes in LightSolver.cpp

```
void LightSolver::readParameter(const int id, const bool hasMultipleIDs)
{
    ...
      Suppressed code
                  ...


  // required
  beam[id].intensity= 0.0f;
  beam[id].basisStr= LightSolver::DefaultBasisString; // (JFM 14/NOV/2018)
  beam[id].basis= PLANE_WAVE;  // (JFM 31/OCT/2018)
  // optional
  beam[id].phase= 0.0f;
  beam[id].polarization= 0.0f;
 ...
      Suppressed code
                     ...
  // for evolution only
  beam[id].offset= Zero3D; // microns
  beam[id].beamWidth= 0.0f; // microns
  beam[id].riseTime= 30.0f; // fs
  beam[id].sgOrder= 2;
  beam[id].sourceTag= -1; // derived
  // 16/NOV/2018 JFM: for LgBeam and HgBeams only...
  beam[id].focalDistance= 0.0f; //microns
  beam[id].l= 0;  // LG OAM
  beam[id].p= 0;  // LG
  beam[id].m= 0;  // HG
  beam[id].n= 0;  // HG


 ...
    Suppressed code
                 ...
} // end readParameter()
```

After `laser.readParameters` has completed parsing the parameters from the input deck, the `laser.initialize` method is called. The `laser initialize` method calls the wave solver class' methods `ws.createPlanewaveSource` and `ws.setPlanewave-Parameters`. The changes made here extended the wave solver class' methods by adding `ws.createLaguerreGaussSources`, `ws.createHermiteGaussSources`, `ws.set-LaguerreGaussSources`, and `ws.setHermiteGaussSources`. These changes allocate a one dimensional array of structures (structs) of type `Planewave_t`, `LaguerreGauss_-t`, or `HermiteGauss_t` called `pw`, `lg`, or `hg` respectively. Each element of the different arrays now represents one beam that can be of three types: plane wave, Laguerre-Gauss, or Hermite-Gauss. The fields of each element of the arrays are filled in by `ws.setPlanewaveParameters`, and the new methods `ws.setLaguerreGaussSources`, or `ws.setHermiteGaussSources`. [This is done by using an intermediate "beams" array (array of `Beams_t`, `LaguerreGauss_t`, or `HermiteGauss_t`) that was generated by `laser.readParameters` and not described here).

Listing 3.2: initialize changes in LightSolver.cpp

```
void LightSolver::initialize(void)
{
   ...
    Suppressed code
                 ...

    ws.createPlanewaveSources(nPwBeams, nBeamsToList);
    ws.createLaguerreGaussSources(nLgBeams, nBeamsToList);
    ws.createHermiteGaussSources(nHgBeams, nBeamsToList);

 ...
    Suppressed code
                 ...

    for (int bn=0; bn<nBeams; bn++) {
      const Basis          basis= beam[bn].basis;
      const float        beamAmp= sqrtf(beam[bn].intensity/Io); // 435
      const float      beamPhase= beam[bn].phase * degreesToRadians; // radians
      const float   polarization= beam[bn].polarization; // degrees
  ...
     Suppressed code
               ...
      const float  focalDistance= beam[bn].focalDistance;
      const int              lGl= beam[bn].l;
      const int              lGp= beam[bn].p;
      const int              hGm= beam[bn].m;
      const int              hGn= beam[bn].n;

 ...
     Suppressed code
               ...

      SchrodingerSolver3::GridSide side= SchrodingerSolver3::GridSide(sourceTag - 1);

        switch (basis)   // JFM (27/NOV/2018)
        {
            case 0:    // PLANE_WAVE
                ws.setPlanewaveParameters(side,beamDir,beamOff,amplitude,
                                          beamOmega,beamColor,beamGroup,beamPhase,
                                          beamWidth,beamRise,sgOrder);
                if (lpse->printLevel2) {  // debugging changes
                 printf("  intitialize is calling setPlaneWaveParameters() \n");
                 printf("    numPlanewaves= %d \n",ws.numPlanewaves);
                }
                break;
            case 1:    // LAGUERRE_GAUSS
                ws.setLaguerreGaussParameters(side,beamDir,beamOff,amplitude,
                                          beamOmega,beamColor,beamGroup,beamPhase,
                                          beamWidth,beamRise,focalDistance,lGl,lGp);
                if (lpse->printLevel2) {  // debugging changes
                 printf("  initialize is calling setLaguerreGaussParameters \n");
                 printf("    numLaguerreGaussBeams= %d \n",ws.numLaguerreGaussBeams);
                }
                break;
            case 2:    // HERMITE_GAUSS
                ws.setHermiteGaussParameters(side,beamDir,beamOff,amplitude,
                                          beamOmega,beamColor,beamGroup,beamPhase,
                                          beamWidth,beamRise,focalDistance,hGm,hGn);
                if (lpse->printLevel2) {  // debugging changes
                 printf("  initialize is calling setHermiteGaussParameters \n");
                 printf("    numHermiteGaussBeams= %d \n",ws.numHermiteGaussBeams);
                }
                break;
        } // end switch
      } // endif

 ...
     Suppressed code
               ...
} // end initialize()
```

Once the three arrays have been created, `light.setup` calls the `SchrodingerSolver3` methods `ws.computeLightSources` which also has modifications for the new beam types. This step includes creating another one dimensional array (called `wi`). Each element of this array represents a “wave injector”. It is struct of type `WaveInjector_t`. One injector (element of the array) is allocated by `computeLightSources` for every grid cell (indexed by the integers i, j, and k) that requires correction due to the presence of a light source incident on the simulation domain. The information required to correct the finite-difference time step for the electric field on the TF-SF boundary is stored in the `src` field. It is computed by `addPlanewaveSource`, and the new methods `addLaguerreGaussSource`, and `addHermiteGaussSource`. The pseudo code is given in the code listing below.

Listing 3.3: computeLightSources changes in SchrodingerSolver3.cpp

```
void SchrodingerSolver3::computeLightSources(const float microTimeStep)
{
    ...
     Suppressed code
                ...

  for (int i=idxLower; i<=idxUpper; i++) {
    for (int j=0; j<Ny; j++) {
      for (int k=0; k<Nz; k++) {

        bool found_loc= false; float minOmega_loc= BIG_FLOAT, maxOmega_loc= 0.0f;

        if (hasPlanewaveSources) addPlanewaveSource
    (startTime,i,j,k,found_loc,minOmega_loc,maxOmega_loc);
        if (hasLaguerreGaussSources) addLaguerreGaussSource
    (startTime,i,j,k,found_loc,minOmega_loc,maxOmega_loc);
        if (hasHermiteGaussSources) addHermiteGaussSource
    (startTime,i,j,k,found_loc,minOmega_loc,maxOmega_loc);
        if (hasSphericalSources) addSphericalSource
    (startTime,i,j,k,found_loc,minOmega_loc,maxOmega_loc);

        if (found_loc) {
          #pragma omp critical (updateOmegaAndInjectors_crit)
          {
            minOmega= min(minOmega,minOmega_loc);
            maxOmega= max(maxOmega,maxOmega_loc);
            nInjectors++;
          } // critical (updateOmegaAndInjectors_crit)
        } // endif

      } // endfor k
    } // endfor j
  } // endfor i
  ...
     Suppressed code
                ...
  XcString errMsg= NULL; // OK
  if (nInjectors > 0) {

  ...
     Suppressed code
                ...


    for (int i=idxLower; i<=idxUpper; i++) {
      for (int j=0; j<Ny; j++) {
        for (int k=0; k<Nz; k++) {

          bool found=false; float minOmega= BIG_FLOAT, maxOmega= 0.0f;

          XcComplex3 src; // zero
          if (hasPlanewaveSources) src += addPlanewaveSource...
```

```
                ...(startTime,i,j,k,found,minOmega,maxOmega);
          if (hasLaguerreGaussSources) src += addLaguerreGaussSource...
                ...(startTime,i,j,k,found,minOmega,maxOmega);
          if (hasHermiteGaussSources) src += addHermiteGaussSource...
                ...(startTime,i,j,k,found,minOmega,maxOmega);
          if (hasSphericalSources) src += addSphericalSource...
                ...(startTime,i,j,k,found,minOmega,maxOmega);

  ...
     Suppressed code
                ...

              wiid++;
            } // end critical (foundInjectorNode_crit)
          } // endif(found)

        } // endfor k
      } // endfor j
    } // endfor i

  } // endif (nInjectors>0)

...
     Suppressed code
                ...
} // end computeLightSources()
```

As shown in the Listing for `computeLightSources`, a loop is performed over every grid cell and if an injector is required at grid point (i, j, k) the `addPlanewaveSource`, `addLaguerreGaussSource`, or `addHermiteGaussSource` functions are now is called to compute the required correction (note that `addPlanewaveSource`, `addLaguerreGaussSource`, or `addHermiteGaussSource` is also used to test for the presence of the respective injector). This correction is stored in the `src` field for that particular injector. As each injector is initialized, the index of the injector array `wiid` is incremented.

Listing 3.4: Laguerre-Gauss changes in SchrodingerSolver3.cpp

```
XcComplex3 SchrodingerSolver3::addLaguerreGaussSource...
          ...(const float time, int i, int j, int k,...
                    ...bool &found, float &minOmega, float &maxOmega)
{
  ...
     Uneditted code
                ...

  for (int sid=0; sid<numLaguerreGaussBeams; sid++) {
    const LaguerreGauss_t &lg= laguerregauss[sid]; // a reference for speed

    // NOTE: is3D implies is2D implies is1D
    //
    // inject from X min or max
    if (is1D) { // or 2D or 3D
      if (injectInsideY && injectInsideZ) {

        if (lg.side == SIDE_X_MIN) {
          //
          if (i == inject_min_i) {
            // need to see what Koff represents here  -->
            XcFloat3 Koff= lg.Koff; Koff.x= float(inject_min_i)/float(Nx1);
            XcFloat3 Kdir= lg.Kdir;

            // NOTE: is3D implies is2D implies is1D
            const float x= (is1D) ? (float(i)/float(Nx1) − Koff.x)*gridSize.x : 0.0f;
            const float y= (is2D) ? (float(j)/float(Ny1) − Koff.y)*gridSize.y : 0.0f;
            const float z= (is3D) ? (float(k)/float(Nz1) − Koff.z)*gridSize.z : 0.0f;


            const XcFloat3 eu= UNIT3D(lg.ampl);
            const XcFloat3 gu= CROSS3D(Kdir,eu);
            const XcFloat3 Rvec= XcFloat3(0,y,z);

            const XcFloat3 Rperp= DOT3D(eu,Rvec)*eu + DOT3D(gu,Rvec)*gu;


            const XcFloat3 Rm1((x − h_xyz),y,z);
            const XcFloat3 R0((x),y,z);
            const XcFloat3 Rp1((x + h_xyz),y,z);

            const float Ko_ijk= Ko(i,j,k);

            const float KdotRm1= DOT3D(Ko_ijk*Kdir,Rm1);
            const float KdotR0= DOT3D(Ko_ijk*Kdir,R0);
            const float KdotRp1= DOT3D(Ko_ijk*Kdir,Rp1);

      const float weight= superGaussian(Rperp,lg)/EoScale(i,j,k);

            //
            const float beamWidth = lg.sd2;
            const float zr = M_PI*beamWidth/0.351; //rayleigh range
            const float beamWidth2 = lg.sd2;
            const float RperpW = ((2.0f*DOT3D(Rperp,Rperp))/(beamWidth2));
            const XcComplex I(0.0f,1.0f); // complex for the phase term
            const float weightH = weight*laguerreMode...
      ...(RperpW,lg.l,lg.p)*powf(sqrtf(RperpW),fabsf(lg.l));
      // readjusted weighting for the LG modes

            // The weighted waves phase terms adjusted
            const XcComplex3 wPwsM1= weightH*planewaveSource...
      ...(KdotRm1,time,lg)*exp(I*(−1.0f*float(lg.l)*atan2f(Rperp.z,Rperp.y)));
            const XcComplex3 wPwsR0= weightH*planewaveSource...
      ...(KdotR0,time,lg)*exp(I*(−1.0f*float(lg.l)*atan2f(Rperp.z,Rperp.y)));
            const XcComplex3 wPwsP1= weightH*planewaveSource...
      ...(KdotRp1,time,lg)*exp(I*(−1.0f*float(lg.l)*atan2f(Rperp.z,Rperp.y)));

            XcComplex3 s;

            s.x=    (0.5f*I*h_xyz*Ko_ijk*Kdir.y)*wPwsP1.y
                  + (0.5f*I*h_xyz*Ko_ijk*Kdir.z)*wPwsP1.z
                  −      sqr(h_xyz*Ko_ijk*Kdir.x)*wPwsR0.x;

// JFM (20/NOV/2018) dominant terms are at i minus 1 and are added. i.e.,
//   add the incident wave on the SF side to get TF update at i,j,k (since i,j,k
//   is on the TF side).
            s.y= wPwsM1.y +       (0.5f*I*h_xyz*Ko_ijk*Kdir.y)*wPwsP1.x
                          −             sqr(h_xyz*Ko_ijk*Kdir.y)*wPwsR0.y
                          − (sqr(h_xyz*Ko_ijk)*Kdir.y*Kdir.z)*wPwsR0.z;

            s.z= wPwsM1.z +       (0.5f*I*h_xyz*Ko_ijk*Kdir.z)*wPwsP1.x
                          −             sqr(h_xyz*Ko_ijk*Kdir.z)*wPwsR0.z
                          − (sqr(h_xyz*Ko_ijk)*Kdir.z*Kdir.y)*wPwsR0.y;

            b += s;
            found= true;
         } // endif inject_min_i

         if (i == inject_min_i −1) {  // JFM i.e., we are on the SF side
```

```
...
    Suppressed code
             ...
} // end addLaguerreGaussSource()
```

The `addLaguerreGaussSource` function (see Listing) uses the previously defined `lg` array to compute the local correction (at grid index i, j, k). This function is called for every grid cell in the simulation. It sets the `bool` variable `found` to `false` if no injector is required because of the location of the grid point. It loops over every element in the `lg` array. If it finds at least one LG mode on a given boundary, then the `src` is computed by evaluating the factors that appear in Eq. (3.11). Evaluation of each factor requires knowledge of the wave vector and other properties that are fields of each `LaguerreGauss_t` struct (see Listing for `addLaguerreGaussSource`). This evaluation reuses the existing planewave function (`planewaveSource`) to compute $\exp(i\mathbf{k}\cdot\mathbf{x})$, but a new function has been written to evaluate the generalized Laguerre polynomials (LaguerreMode; see listing). This procedure repeats for the minimum and maximum coordinates boundaries for all dimensions x, y,and z depending.

Listing 3.5: Laguerre-Gauss changes in SchrodingerSolver3.cpp

```
inline float SchrodingerSolver3::laguerreMode(float rho, const int l, const int p)
{
   float rho2 = sqr(rho);
   float rho3 = rho2*rho;
   float tmp= 0.0;
   int labs = fabsf(l);
   float lf = float(labs);

  switch (p) {      // First 4 polynomials are hard coded (BA 5/JUNE/2019)
    case 0:
      tmp= 1.0f;      // L_0
      break;

    case 1:
      tmp= 1.0f+lf-rho;      // L_1
      break;

    case 2:
      tmp= 0.5f*((1.0f+lf)*(2.0f+lf)-2.0f*(2.0f+lf)*rho+rho2);      // L_2
      break;

    case 3:
      tmp= (1.0f/6.0f)*((lf+3.0f)*(lf+2.0f)*(lf+1.0f)-3.0f*...
    ...(lf+3.0f)*(lf+2.0f)*rho+3.0f*(lf+3.0f)*rho2-rho3);      // L_3
      break;
  } // end switch

  return tmp;
}  // end laguerreGaussMode()
```

The same has been done for the HG modes where `addHermiteGaussSource` is now the function name. The incident light is computed by accumulating the contribution of each plane wave with the modifications to the plane wave with the terms needed for the HG mode (i.e., the Hermite polynomials are evaluated by 'hermiteMode'). Various "housekeeping" functions were added that manage the memory that is required by the new additions but we do not describe them here. This completes the `LightSolver` initialization and the modifications for the `lpse.initialize` method of `lpse`. As the existing infrastructure is reused, no changes are required beyond the initialization stage.

References

[20] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York: Dover Publications, 1961.

[21] J.-P. Berenger, “A perfectly matched layer for the absorption of electromagnetic waves,” *Journal of Computational Physics*, vol. 114, no. 2, pp. 185–200, 1994, ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1994.1159. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999184711594.

Chapter 4

Simulation of OAM beams and the implementation of angular momentum diagnostics.

4.1 Introduction

The operation of the new code is illustrated by several different cases, chosen in both two and three spatial dimensions. For the 2-D case we compute the angular momentum density and the total angular momentum of the electromagnetic field. The density of angular momentum $\boldsymbol{\rho}^J$ (i.e., the angular momentum per unit volume) of an EM field is given by

$$\boldsymbol{\rho}^J(x) = \hat{\mathbf{x}} \times \left(\frac{\mathbf{E} \times \mathbf{B}}{4\pi} \right). \tag{4.1}$$

Hence, the total angular momentum $\mathbf{J}$ is then given by integration:

$$\mathbf{J} = \int d\mathbf{x} \quad \hat{\mathbf{x}} \times \left(\frac{\mathbf{E} \times \mathbf{B}}{4\pi} \right). \tag{4.2}$$

To investigate the value of these quantities in the simulations, several MATLAB scripts have been written. Note that we may identify the quantity $\mathbf{S} = (\mathbf{E} \times \mathbf{B})/4\pi$ as the Poynting flux. As such, it is convenient to first compute $\mathbf{S}(\mathbf{x}, t)$ from the simulation data; then $\boldsymbol{\rho}^J$ and $\mathbf{J}$ (or any component thereof) can be computed for any choice of coordinate origin (Sec. 4.2).

There are two types of orbital momentum associated with light waves: spin angular momentum (SAM) and orbital angular momentum (OAM). In Eqs. (4.1) and (4.2), these two components are combined. However, spin angular momentum is related to the optical polarization of light, e.g. circular polarization corresponds to $\pm\hbar$ angular momentum per photon. Circular polarization is when the magnitude of the light wave is constant as its electric field vector rotates around the lights propagation axis transversely. Orbital angular momentum is the component of angular momentum of light that depends of the field spatial structure distribution instead of the polarization.

Beams with a phase factor $e^{il\theta}$ have a well-defined orbital angular momentum, [e.g. our LG beams defined by Eq.(3.11)]. For such beams, the ratio of angular momentum flux to energy flux is l/ω. Since the energy of a photon is $\hbar\omega$, each photon carries $\hbar l$ angular momentum.

An additional complication, that arises when investigating light beams carrying OAM is that the OAM may be either "internal" or "external". The distinction is merely due to a choice of origin, for example, a beam having no internal OAM would still have angular momentum if the coordinate origin [used to compute $\boldsymbol{\rho}^J$ or $\mathbf{J}$ using Eqs. (4.1) and (4.2), respectively] were not chosen to fall on the beam axis. Above considerations should be kept in mind as we analyze the simulation data shown in Sec. 4.2 and Sec. 4.3 below.

4.2 Computation of the Poynting flux from the time-enveloped Maxwell equations

If the dielectric function in Eq. (2.17), describing the propagation of EM waves, is real (i.e., no dissipation), a conservation law can be obtained from the time enveloped wave equation [Eq.(2.16)]. This is achieved by taking the dot product of Eq. (2.16) with $\mathbf{E}^*$ and taking the difference with the equation obtained by taking the dot product of the conjugate of Eq. (2.16) with $\mathbf{E}$. This results in the conservation law for the energy density $W_E \equiv |\mathbf{E}|^2/(8\pi)$:

$$\frac{\partial}{\partial t} W_E + \nabla \cdot \mathbf{S} = 0, \tag{4.3}$$

where, in index notation, the Poynting vector **S** is given by

$$S_j \equiv -i \frac{c^2}{16\pi\omega_0} \left(E_i^* \frac{\partial E_i}{\partial x_j} - E_i \frac{\partial E_i^*}{\partial x_j} - E_j^* \frac{\partial E_i}{\partial x_i} + E_j \frac{\partial E_i^*}{\partial x_i} \right), \tag{4.4}$$

$$= \frac{1}{8\pi} \frac{c^2}{\omega_0} \Im m \left\{ E_i^* \frac{\partial E_i}{\partial x_j} + E_j \frac{\partial E_i^*}{\partial x_i} \right\}. \tag{4.5}$$

The expression for **S** can be used to evaluate the power flowing though the simulation boundaries. In addition to the angular momentum **J** and angular momentum density $\boldsymbol{\rho}^J$, as described above (Sec. 4.1).

4.3 Results in two dimensions.

As the Hermite-Gauss beams are defined in a coordinate system, they can be simulated in a two-dimensional *LPSE* simulation. Figure 4.1 shows a plot of the electric field intensity from such a simulation, at a time $t = 2$ ps after the beams, injected from the boundaries, have been turned on. A fundamental Gaussian beam propagates from the left boundary to the right and a Hermite Gaussian mode beam of order 4 propagates from the bottom to the top, having being injected from the lower boundary. Initially, the fundamental Gaussian beam has its centroid at the location $x = 4\ \mu m$ and it propagates in the y-direction. The beam is seen to curve in the downward direction as it propagates. This is because of refraction: the plasma density n_0 has a linear variation in the x (but no variation in y). Its gradient is in the positive x direction. The HG beam has its centroid at $y = 0$. The four nodes are evident and can be compared with Fig. 3.4. Both the fundamental Gaussian and the HG beam are polarized in a direction that points out the simulation plane. As a result, the beams interfere. This interference is evident in the figure below. It can also be seen that the intensity of the fundamental Gaussian beam is higher as it exits the right boundary ($x = 0$) compared with the injected intensity (left boundary). This is because energy has been exchanged from the HG beam due to the CBET mechanism. As the motivation for the code development described in this thesis was primarily to investigate the impact of OAM on such energy exchanges, we apply the diagnostics described in Sec. 4.1 and 4.2 to these results.

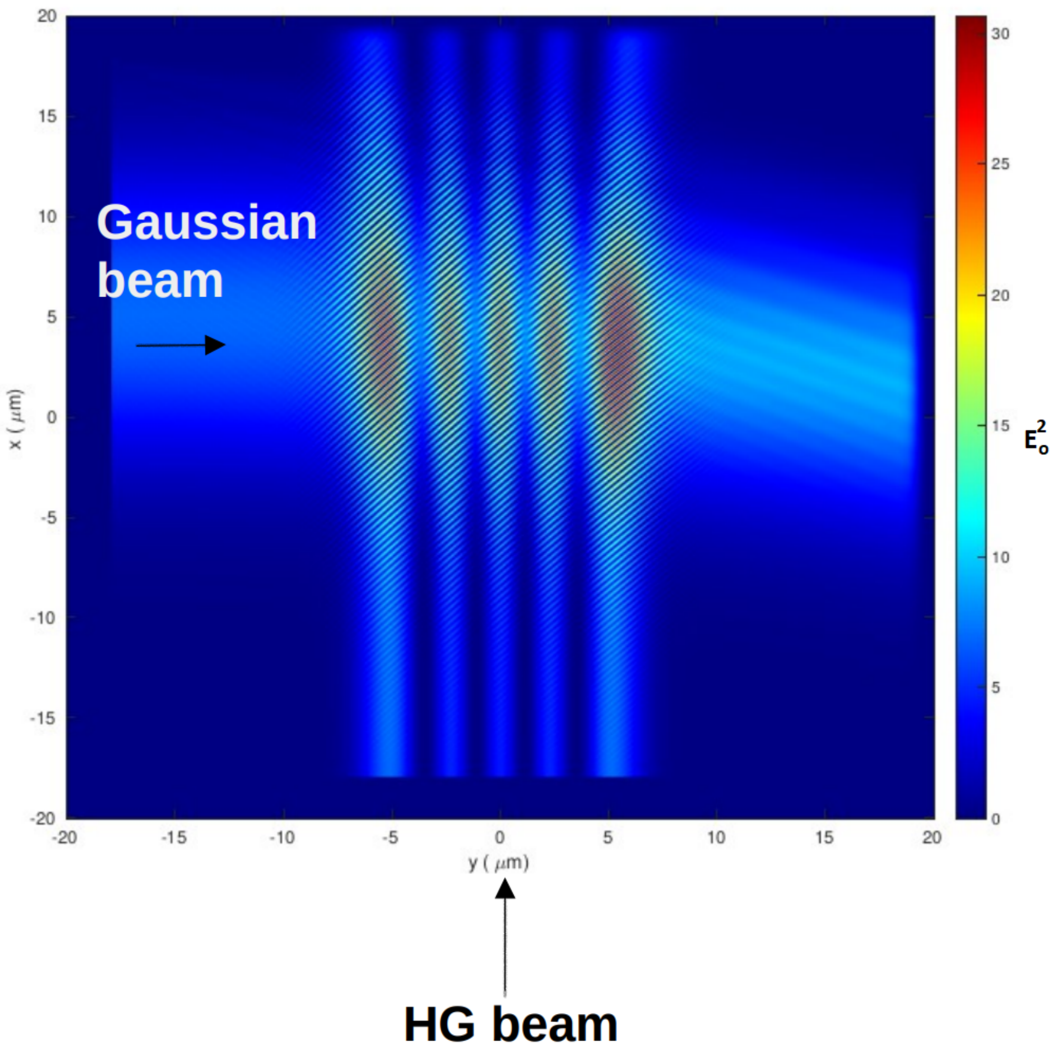


Figure 4.1: In the case shown, a Gaussian laser beam is seen to cross a Hermite-Gauss (HG) beam (order 4). The colour bar indicates laser intensity in arbitrary units.

Figure 4.2 shows the plot from Fig. 4.1 (on the left) together with a zoom of the crossing region (white box) on the left. The color bars on the zoomed plot indicate the magnitude of the Poynting flux that has been computed using Eq. (4.5). The direction of the Poynting flux is indicated by the small black arrows.

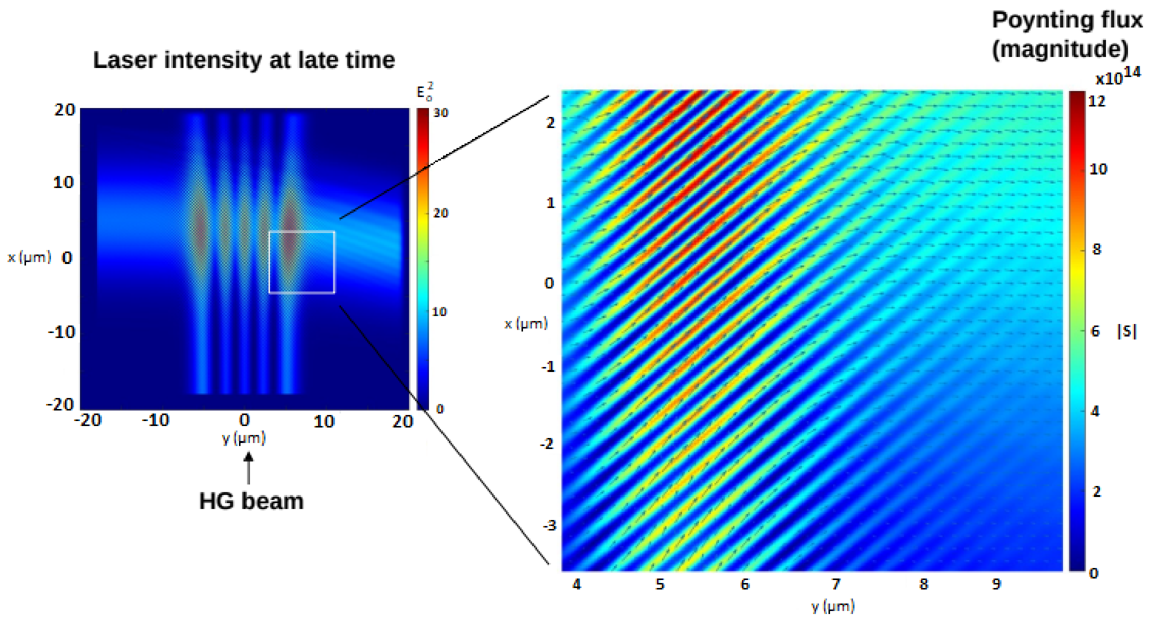


Figure 4.2: The direction (arrows) and magnitude (colour bar) of the Poynting flux, in W/cm^2, gives a physical picture of the transfer of linear momentum density.

Figure 4.3 shows a zoom of a different spatial region of the simulation (white box) where the laser intensity is lower and the contribution from the fundamental Gaussian beam is negligible at the lower region $x \lesssim 3\ \mu m$. The contribution from the HG beam starts to be felt for $x \gtrsim 4\ \mu m$ and the direction of the Poynting flux can be seen to rotate from the vertical direction to the right.

Knowledge of the Poynting flux (linear momentum density) now permits the angular momentum density $\boldsymbol{\rho}^J$ to be computed. The Poynting flux, shown in Fig. 4.2 and 4.3 has been processed using Eq. (4.2), where the origin is as simulated.

Figure 4.4 shows ρ_z^J (the angular momentum density) in the z-direction at time $t = 10$ ps. The figure can be understood by first considering the beam at the left ($y = -20\ \mu m$) boundary: The yellow colour indicates a positive value because the Poynting flux is in the y-direction, and the x-coordinate values are positive. The

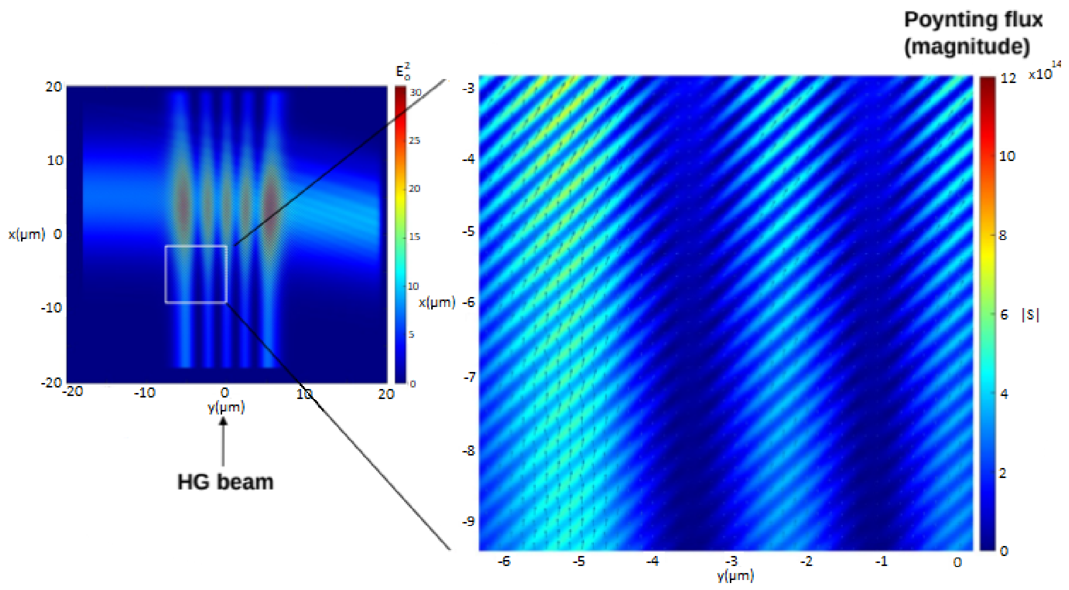


Figure 4.3: The momentum of light is proportional to the Poynting flux (shown on the right)

sense of rotation, imparted about the origin, is therefore in the clockwise (positive) direction. The opposite is true for the parts of the beam having a negative x value. As the fundamental Gaussian beam was injected with its centroid located at positive x ($x = 4\ \mu m$), the beam will contain extrinsic OAM, while the HG beam will not. This can be seen more clearly when examining the total angular momentum in the z direction (J_z).

Figure 4.5 shows the increasing orbital angular momentum J_z that has been computed as a function of time from the angular momentum density ρ_z^J. A snapshot of ρ_z^J is shown for time $t = 10$ ps in Fig. 4.4. Notice that the angular momentum is initially positive due to the extrinsic angular momentum of the Gaussian beam. It increases in time and then saturates at a higher positive value. The increase is due to a combination of refraction in the linear density profile and angular momentum exchange from the HG beam. This is a good illustration of the type of effects that can now be studied as a result of the code development work described in this thesis.

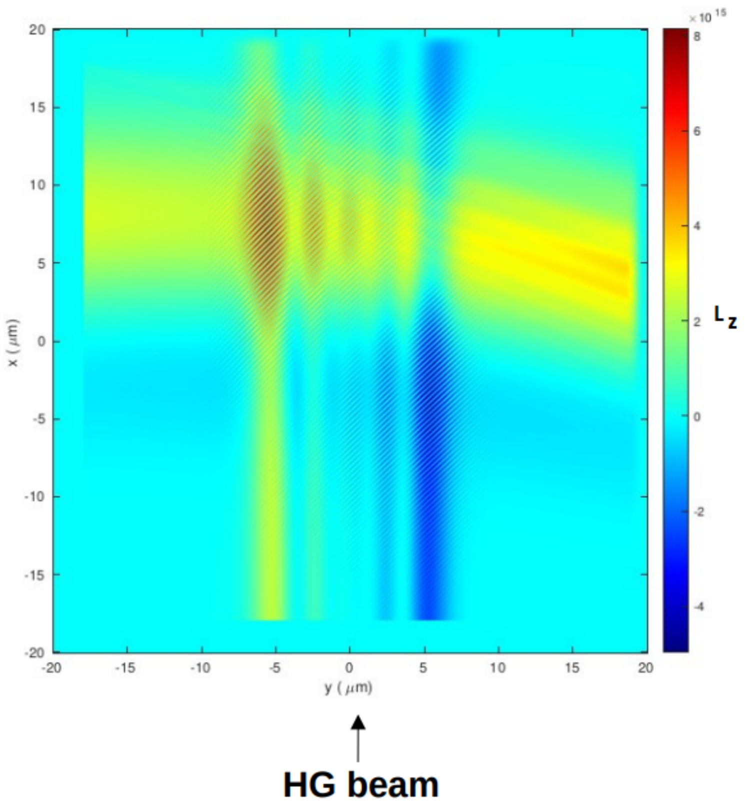


Figure 4.4: The angular momentum density in the z direction ρ_z^J at time $t = 10$ ps (in arbitrary units).

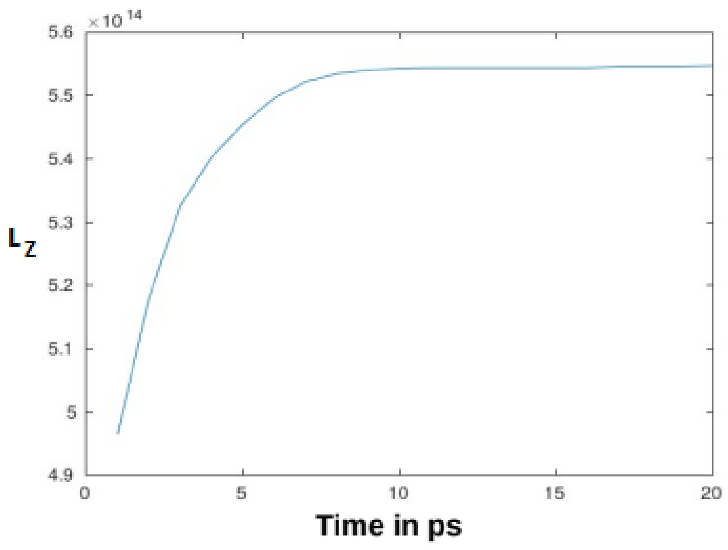


Figure 4.5: The total angular momentum (in the z direction) as a function of time, for the simulation, as computed by Eq. (4.4) in arbitrary units.

4.4 3-D simulation results of crossing LG beams

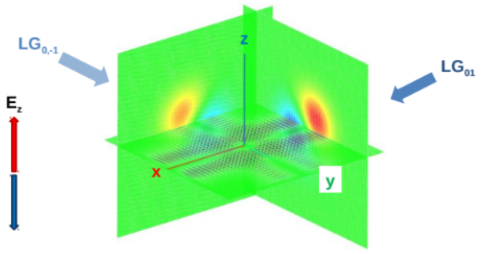
Testing the implementations of LG beams requires that simulations be performed in three dimensions. A number of such simulations were performed and some examples of which are shown here.

Figure 4.6 shows five snapshots (taken at the same fixed time) of a simulation in which two LG beams cross. One LG beam (of order $p = 0, l = -1$) enters the simulation boundary from the minimum y boundary, propagating in the y-direction. Such a beam carries $-\hbar$ orbital angular momentum per photon in its direction of propagation. A second LG beam ($p = 0, l = +1$) enters from the minimum x boundary. Both beams are linearly polarized in the z-direction and contain no spin angular momentum.

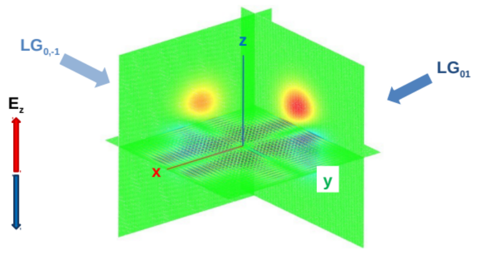
The individual sub-figures in Fig. 4.6 shows slices of the real electric field amplitudes on three planes: one plane taken at constant x, one at constant y, and the other at constant z. In moving sequentially from Fig. 4.6a to Fig. 4.6e the planes at constant x and y advance in their respective (positive) coordinate directions. In doing so, it is evident from the figures that the two lobes of each beam (cf. e.g., Fig. 3.8) rotate about their respective axis' on propagation.

Figure 4.7 shows the same simulation, but the electric field intensity is plotted instead of the real amplitude (cf. e.g, Fig. 3.7). Notice the donut-like structure of the intensity pattern as is consistent with the radial mode number $p = 0$. In this simulation, energy is exchanged from the $LG_{0,1}$ beam to the $LG_{0,-1}$ beam via the CBET mechanism. Note that the intensity of the $LG_{0,-1}$ beam is initially one half that of the $LG_{0,1}$ beam. This is most easily seen in Fig. 4.8 which shows iso-surface contours of the electric field intensity before and after CBET has occured (Fig. 4.8a and Fig. 4.8b, respectively). The helical structure of the OAM carrying beams are very evident in the figure.

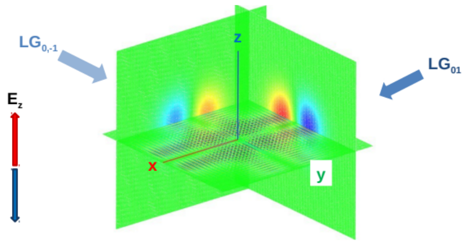
The results presented here highlight some of the rich physical processes that can



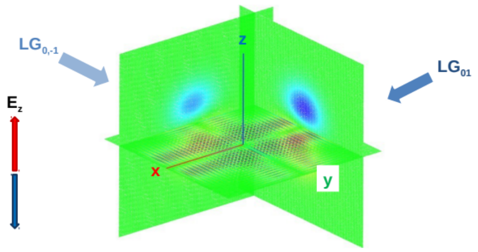
(a)



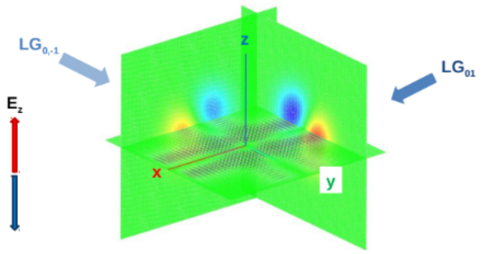
(b)



(c)



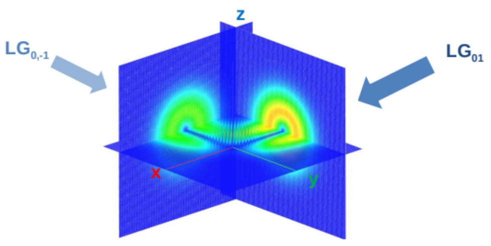
(d)



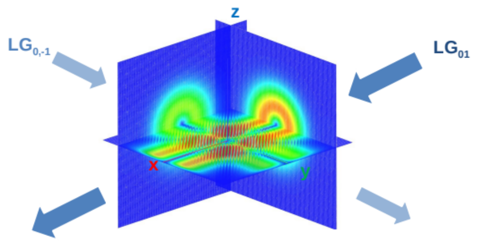
(e)

Figure 4.6: Two crossing beams spiral as they propagate due to their orbital angular momentum. The real amplitude of the E_z electric field is shown. [both beams are plane polarized in the z direction (no SAM)].

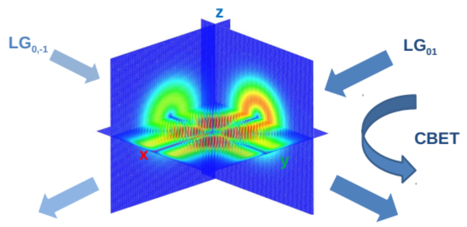
now be studied, and evaluated, as a result of the work described in this thesis.



(a)

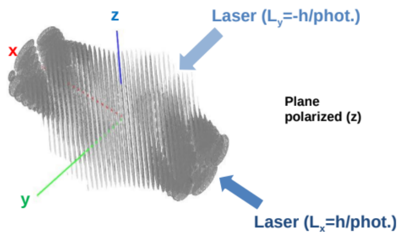


(b)

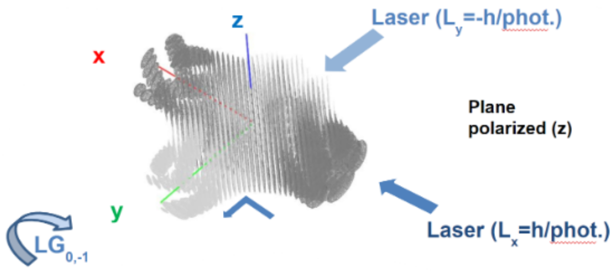


(c)

Figure 4.7: A transfer of energy is observed between the crossing beams shown in Fig. 4.6. Energy is exchanged from the more intense beam $LG_{0,1}$ to the weaker beam $LG_{0,-1}$.



(a)



(b)

Figure 4.8: Iso-surface of the electric field intensity for crossing beams. Before (a) and after (b) cross-beam energy transfer has occured.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

As the human race advances in technology and knowledge, the requirement of energy increases to feed the advancement. With this increase of energy consumption comes the consequence of an evolving society. Thus the pursuit to solve this dilemma without stopping the advancements of society is the main problem at hand for this and coming generations. There are already multiple avenues being worked on, however nuclear energy seems to be most promising.

In the nuclear fusion category there is two major subcategories being researched: fission and fusion. Fission has its drawbacks like nuclear power plants reactor going critical and exploding, damage to the plant and the surrounding environment can be serious and long lasting, and produce nuclear waste that needs to be handled appropriately due to its radioactive nature. Thus leaving us with fusion energy.

In the nuclear fusion category there is two major subcategories being researched: magnetic confinement fusion and inertial confinement fusion (ICF). In ICF, The most researched approach uses lasers to confine and compress the fuel to get to conditions required for the nuclear fusion reaction to overcome the Coulomb force causing the fuel to ignite and burn. The plasma can also cause the laser energy to be poorly absorbed by the fuel capsule in both methods due to instabilities of the plasma that occur due to high intensity of the laser light. There are two main instabilities of concern: Stimulated Raman Scattering (SRS) and Stimulated Brillouin Scattering (SBS). Cross-beam energy transfer (CBET) is a special case of SBS that occurs when multiple EM waves (e.g. laser beam) of overlap in a plasma. As a result, its proposed to modify the spatial, rather than temporal, structure of the laser beam.

Schemes to implement laser temporal structure wide enough to mitigate laser-plasma instabilities will be both intrusive and expensive. As an alternate approach, work is presented that investigates the mitigating effects of spatial, rather than temporal, laser beam conditioning on cross-beam energy transfer (CBET). Such conditioning might be generated by phase plates alone and could therefore be implemented more easily. We have quantified the energy exchange occurring between crossing laser beams that possess orbital angular momentum (OAM) as the amount of OAM exchange between the beams is varied. This work enables studies to be performed in 3-D using the non-paraxial wave-based *LPSE* simulation code. It required significant modifications to the code to allow for the beams with OAM. The modifications required changes to the boundary conditions and the total field scattered field to be implemented successfully.

The modifications that were made to the *LPSE* code was to inject OAM laser modes from the boundary using wave injectors. The modifications replicated the `planewave` method described earlier. It is implemented for the LG/HG laser modes, which can carry OAM, by adding a new structure that, depending on the beam type, will have different parameters. These are set in the framework built for the existing `planewave` boundary injection.

The results show that the modifications made to the code gave the expected spatial structures when comparing with already known solutions for the LG/HG laser modes. Also seen was the helical structure of the OAM carrying beams in the iso-surface figure (Fig. 4.8). The results presented here highlight some of the rich physical processes that can be studied, and evaluated, as a result of the work described in this thesis.

5.2 Future Work

Future work for work done in this thesis would be using the newly implemented boundary wave injectors and diagnostics to investigate the potential for mitigation using OAM for cross beam energy transfer (CBET), stimulated Brillouin scattering (SBS), two-plasmon decay (TPD), stimulated Raman scattering (SRS), and any other nonlinear plasma effects that affect ICF and can be simulated in the *LPSE* code.

Bibliography

[1] W. L. Kruer, “The Physics of Laser Plasma Interactions,” in, ser. Frontiers in Physics, D. Pines (Ed.) Vol. 73, Redwood City, CA: Addison-Wesley, 1988.

[2] J. Myatt, J. Shaw, R. Follett, D. Edgell, D. Froula, J. Palastro, and V. Goncharov, “Lpse: A 3-d wave-based model of cross-beam energy transfer in laser-irradiated plasmas,” *Journal of Computational Physics*, vol. 399, p. 108 916, Sep. 2019. DOI: 10.1016/j.jcp.2019.108916.

[3] W. L. Kruer, S. C. Wilks, B. B. Afeyan, and R. K. Kirkwood, “Energy transfer between crossing laser beams,” *Phys. Plasmas*, vol. 3, pp. 382–385, 1996. DOI: 10.1063/1.871863.

[4] D. H. Edgell, W. Seka, J. A. Delettrez, R. S. Craxton, V. N. Goncharov, I. V. Igumenshchev, J. F. Myatt, A. V. Maximov, R. W. Short, T. C. Sangster, and R. E. Bahr, “Cross-beam energy transport in direct-drive implosion experiments,” *Bull. Am. Phys. Soc.*, vol. 54, p. 145, 2009.

[5] I. V. Igumenshchev, D. H. Edgell, V. N. Goncharov, J. A. Delttrez, A. V. Maximov, J. F. Myatt, W. Seka, and A. Shvydky, “Modeling crossed-beam energy transfer in implosion experiments on OMEGA,” *Bull. Am. Phys. Soc.*, vol. 54, p. 145, 2009.

[6] V. N. Goncharov, T. C. Sangster, R. Betti, T. R. Boehly, M. J. Bonino, T. J. B. Collins, R. S. Craxton, J. A. Delettrez, D. H. Edgell, R. Epstein, R. K. Follett, C. J. Forrest, D. H. Froula, V. Yu. Glebov, D. R. Harding, R. J. Henchen, S. X. Hu, I. V. Igumenshchev, R. Janezic, J. H. Kelly, T. J. Kessler, T. Z. Kosc, S. J. Loucks, J. A. Marozas, F. J. Marshall, A. V. Maximov, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, D. T. Michel, J. F. Myatt, R. Nora, P. B. Radha, S. P. Regan, W. Seka, W. T. Shmayda, R. W. Short, A. Shvydky, S. Skupsky, C. Stoeckl, B. Yaakobi, J. A. Frenje, M. Gatu-Johnson, R. D. Petrasso, and D. T. Casey, “Improving the hot-spot pressure and demonstrating ignition hydrodynamic equivalence in cryogenic deuterium–tritium implosions on omega,” *Physics of Plasmas*, vol. 21, no. 5, p. 056 315, 2014. DOI: 10.1063/1.4876618. eprint: https://doi.org/10.1063/1.4876618. [Online]. Available: https://doi.org/10.1063/1.4876618.

[7] A. Longman and R. Fedosejevs, “Mode conversion efficiency to laguerre-gaussian oam modes using spiral phase optics,” *Opt. Express*, vol. 25, no. 15, pp. 17 382–17 392, Jul. 2017. DOI: 10.1364/OE.25.017382. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-25-15-17382.

[8] H. He, M. E. J. Friese, N. R. Heckenberg, and H. Rubinsztein-Dunlop, "Direct observation of transfer of angular momentum to absorptive particles from a laser beam with a phase singularity," *Phys. Rev. Lett.*, vol. 75, pp. 826–829, 5 Jul. 1995. DOI: 10.1103/PhysRevLett.75.826. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.75.826.

[9] J. Wang, J.-Y. Yang, I. Fazal, N. Ahmed, Y. Yan, H. HUANG, Y. Ren, Y. Yue, S. Dolinar, M. Tur, and A. Willner, "Terabit free-space data transmission employing orbital angular momentum multiplexing," *Nature Photonics*, vol. 6, pp. 488–496, Jul. 2012. DOI: 10.1038/nphoton.2012.138.

[10] A. Longman, "Under-dense laser-plasma interactions in relativistic optical vortices," pp. 171–175, 2020.

[11] R. L. Berger, C. H. Still, E. A. Williams, and A. B. Langdon, "On the dominant and subdominant behavior of stimulated Raman and Brillouin scattering driven by nonuniform laser beams," *Phys. Plasmas*, vol. 5, pp. 4337–4356, 1998. DOI: 10.1063/1.873171.

[12] R. K. Follett, J. G. Shaw, J. F. Myatt, H. Wen, D. H. Froula, and J. P. Palastro, "Thresholds of absolute two-plasmon-decay and stimulated raman scattering instabilities driven by multiple broadband lasers," *Physics of Plasmas*, vol. 28, no. 3, p. 032 103, 2021. DOI: 10.1063/5.0037869. eprint: https://doi.org/10.1063/5.0037869. [Online]. Available: https://doi.org/10.1063/5.0037869.

[13] J. F. Myatt, H. X. Vu, D. F. DuBois, D. A. Russell, J. Zhang, R. W. Short, and A. V. Maximov, "Mitigation of two-plasmon decay in direct-drive inertial confinement fusion through the manipulation of ion-acoustic and Langmuir wave damping," *Phys. Plasmas*, vol. 20, p. 052 705, 2013. DOI: 10.1063/1.4807036.

[14] S. Skupsky and R. S. Craxton, "Irradiation uniformity for high-compression laser-fusion experiments," *Phys. Plasmas*, vol. 6, no. 5, pp. 2157–2163, 1999. DOI: 10.1063/1.873501.

[15] J. W. Bates, J. F. Myatt, J. G. Shaw, R. K. Follett, J. L. Weaver, R. H. Lehmberg, and S. P. Obenschain, "Mitigation of cross-beam energy transfer in inertial-confinement-fusion plasmas with enhanced laser bandwidth," *Phys. Rev. E*, vol. 97, 061202(R), 2018. DOI: 10.1103/PhysRevE.97.061202.

[16] T. H. Stix, *Waves in Plasmas*, 2nd. New York: Springer-Verlag New York, Inc., 1992.

[17] D. R. Nicholson, "Chapter 7: Fluid equations," in *Introduction to plasma theory.* Krieger Pub. Co., 1992, pp. 129–132.

[18] B. Stroustrup, "A history of c++: 1979–1991," in *History of Programming Languages—II.* New York, NY, USA: Association for Computing Machinery, 1996, pp. 699–769, ISBN: 0201895021. [Online]. Available: https://doi.org/10.1145/234286.1057836.

[19] J. F. Myatt, R. K. Follett, J. G. Shaw, D. H. Edgell, D. H. Froula, I. V. Igumenschev, and V. N. Goncharov, "A wave-based model of cross-beam energy transfer in direct-drive inertial confinement fusion," *Phys. Plasmas*, vol. 24, p. 056 308, 2017. DOI: 10.1063/1.4982959.

[20] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York: Dover Publications, 1961.

[21] J.-P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, no. 2, pp. 185–200, 1994, ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1994.1159. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999184711594.