DESING AND SIMULATION SOFTWARE FOR RHODOTRON TYPE ELECTRON ACCELERATORS

by

Muhammet Furkan Er

B.S., Physics, Boğaziçi University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in M.S. Physics

Boğaziçi University

2023

DESING AND SIMULATION SOFTWARE FOR RHODOTRON TYPE ELECTRON ACCELERATORS

APPROVED BY:

Prof. Veysi Erkcan Özcan .....
(Thesis Supervisor)

Assoc. Prof. Gökhan Ünel .....
(Thesis Co-supervisor)

Assist. Prof. Bora Akgün .....

Assoc. Prof. Bora Işıldak .....

DATE OF APPROVAL: 29.08.2023

ABSTRACT

DESING AND SIMULATION SOFTWARE FOR RHODOTRON TYPE ELECTRON ACCELERATORS

The focus of this study is to design and optimize rose-pattle-type electron accelerators; a new simulation tool called *Rhodotron Simulation* has been developed for this purpose. A coaxial cavity with an operating frequency of 107.5 MHz is currently being manufactured in the Kandilli Detector Accelerator and Instrumentation Laboratory (KAHVELab). This new tool's capabilities will be tested with the cavity and will be used to design complementary bending magnets for reaching $1 - 5$ MeV beam energy.

ÖZET

RHODOTRON TİPİ ELEKTRON HIZLANDIRICILARI İÇİN TASARIM VE SİMÜLASYON YAZILIMI

Bu çalışma, gül yaprağı tipi elektron hızlandırıcıları tasarlamak ve optimize etmek üzerine odaklanmaktadır. Bu amaçla *Rhodotron Simulation* adında yeni bir benzetim uygulaması geliştirilmiş, 107.5 MHz hedef frekansında çalışmak üzere tasarlanmış bir koaksiyal kavitesinin üretimi, Kandilli Algıç Hızlandırıcı ve Enstrümentasyon Laboratuvarı'nda devam etmektedir. Yeni benzetim uygulaması, kabiliyetleri üretilen kavite ile test edildikten sonra bükme mıknatısları tasarım ve optimizasyonunda $1 - 5$ MeV demet enerjisine ulaşmak için kullanılacaktır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\vec{a}$	Acceleration
$\vec{B}$	Magnetic field
c	Speed of light in vacuum
E	Energy
e^-	Electron
$e^- - \vec{E}$	Electron Electric Field Interaction
$e^- - \vec{B}$	Electron Magnetic Field Interaction
$e^- - EM$	Electron Electro-Magnetic Field Interaction
$\vec{E}$	Electric field
f	Frequency
$\vec{F}$	Force
$\vec{p}$	Momentum
$\vec{r}$	Position
t	Time
T	Period
$\vec{v}$	Velocity
Z	Shunt Impedance
γ	Lorentz Factor
β	Beta Factor
ϕ_{lag}	Phase Lag
μX	Mean Value of X
σX	Standard Deviation of X

LIST OF ACRONYMS/ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
eV	Electron Volt
Hz	Hertz
KAHVELab	Kandilli Detector Accelerator and Instrumentation Laboratory
LF	Leap-Frog
Linac	Linear Particle Accelerator
ODE	Ordinary Differential Equation
RF	Radio Frequency
RK	Runge-Kutta
RK4	Forth Order Runge-Kutta
RMS	Root Mean Square
POC	Proof of Concept
CPU	Central Processing Unit
s	Second
T	Tesla
V	Volt
W	Watt

1. INTRODUCTION AND THEORY

Particle accelerators are essential tools for scientific research. They enable scientists to probe the fundamental constituents of matter, study particle interactions, and explore the laws of physics. Accelerators have been instrumental in discovering and characterizing fundamental particles, such as quarks, leptons, and more recently the Higgs boson. They also facilitate the production of high-intensity particle beams for applications in industrial processes. A conservative estimate puts the number of industrial high-current electron accelerators to be in the thousands, worldwide.

MeV-level electron beams, both as is and as X-Ray sources, are especially useful for industrial and medical applications. For instance, high-energy beams (ranging from 5 to 10 MeV) find application in medical accelerators and in services such as sterilizing medical devices and irradiating food. Meanwhile, medium-energy devices (ranging from 400 keV to 5 MeV) contribute to the improvement of wires, cables, and tires, and also play a role in treating factory flue gases and wastewater. Although most of the industrial electron accelerators operating at MeV-level are linear accelerators at the moment, due to their compact nature, circular accelerators have been gaining prominence.

Rose-pattle accelerator is a type of circular accelerator suitable for producing electron beams at MeV-level energy. It was developed in 1989 and was named *Rhodotron* [1]. This name was trademarked by *IBA Group* in the following years.

1.1. Accelerating Charged Particles

1.1.1. Relation between momentum and acceleration

In classical mechanics, Newton's second law defines force, $\vec{\mathbf{F}}$, and relation

$$\vec{\mathbf{F}} = \frac{d\vec{\mathbf{p}}}{dt} = m\frac{d\vec{\mathbf{v}}}{dt} = m\vec{\mathbf{a}}, \tag{1.1}$$

where $\vec{\mathbf{p}}$ is the momentum, $m\vec{\mathbf{v}}$ of the particle.

In special relativity however, relativistic momentum is defined as $\vec{p} = \gamma m_0 \vec{v}$, where

$$\gamma = \frac{1}{\sqrt{1 - v^2/c^2}} = \frac{1}{\sqrt{1 - \beta^2}} = \frac{1}{\sqrt{1 - (\vec{v}\cdot\vec{v})/c^2}} = \frac{1}{\sqrt{1 - \vec{\beta}\cdot\vec{\beta}}} \tag{1.2}$$

is the Lorentz Factor.

Considering these two statements, we can find the relation between momentum and acceleration as

$$\frac{d\vec{p}}{dt} = m_0\frac{d(\gamma\vec{v})}{dt} = m_0\{\frac{d\gamma}{dt}\vec{v} + \gamma\frac{d\vec{v}}{dt}\}, \tag{1.3}$$

$$\frac{d\gamma}{dt} = \gamma^3\vec{\beta}\cdot\frac{d\vec{\beta}}{dt} = \frac{\gamma^3}{c}\vec{\beta}\cdot\vec{a}, \tag{1.4}$$

$$\frac{d\vec{p}}{dt} = m_0\{\frac{\gamma^3}{c}(\vec{\beta}\cdot\vec{a})\vec{v} + \gamma\frac{d\vec{v}}{dt}\}. \tag{1.5}$$

Therefore force can be written as

$$\vec{\mathbf{F}} = \gamma m_0\{\vec{\mathbf{a}} + \gamma^2(\vec{\beta}\cdot\vec{\mathbf{a}})\vec{\beta}\}. \tag{1.6}$$

It is clear that acceleration is not necessarily parallel to the force. To start separating the parallel and perpendicular components relative to $\vec{\beta}$, we can find $\vec{\mathbf{a}}_{||}$ and $\vec{\mathbf{F}}_{||}$ as

$$\vec{\mathbf{a}}_{||} = \frac{(\vec{a}\cdot\vec{\beta})}{\beta^2}\beta, \qquad \vec{\mathbf{F}}_{||} = \frac{(\vec{F}\cdot\vec{\beta})}{\beta^2}\beta, \tag{1.7}$$

where

$$\begin{aligned} \vec{\mathbf{F}}\cdot\vec{\beta} &= \gamma m_0\{\vec{\mathbf{a}}\cdot\vec{\beta} + \gamma^2(\vec{\beta}\cdot\vec{\mathbf{a}})\beta^2\} \\ &= \gamma m_0(\vec{\mathbf{a}}\cdot\vec{\beta})\{\gamma^2\beta^2 + 1\}. \end{aligned} \tag{1.8}$$

Then, $\gamma^2\beta^2 + 1 = \gamma^2$ and Equation 1.7 can be used to achieve

$$\begin{aligned} \vec{\mathbf{F}}\cdot\vec{\beta} &= m_0\gamma^3(\vec{\mathbf{a}}\cdot\vec{\beta}), \\ \vec{\mathbf{F}}_{||} &= \frac{(\vec{\mathbf{F}}\cdot\vec{\beta})}{\beta^2}\vec{\beta} \\ &= m_0\gamma^3\frac{(\vec{\mathbf{a}}\cdot\vec{\beta})}{\beta^2}\vec{\beta} \\ &= m_0\gamma^3\vec{\mathbf{a}}_{||}. \end{aligned} \tag{1.9}$$

Therefore from Equations 1.6 and 1.9, force can be rearranged as

$$\begin{aligned} \vec{\mathbf{F}} &= m_0\gamma^3\vec{\mathbf{a}}_{||}\beta^2 + m_0\gamma\vec{\mathbf{a}} \\ &= m_0\gamma^3\vec{\mathbf{a}}_{||}\beta^2 + m_0\gamma\{\vec{\mathbf{a}}_{||} + \vec{\mathbf{a}}_{\perp}\} \\ &= m_0\vec{\mathbf{a}}_{||}\gamma\{\gamma^2\beta^2 + 1\} + m_0\gamma\vec{\mathbf{a}}_{\perp} \\ &= m_0\vec{\mathbf{a}}_{||}\gamma^3 + m_0\gamma\vec{\mathbf{a}}_{\perp} \\ &= \vec{\mathbf{F}}_{||} + m_0\gamma\vec{\mathbf{a}}_{\perp}. \end{aligned} \tag{1.10}$$

We finally have two separate equations which are in similar form with Equation 1.1,

$$\vec{\mathbf{F}}_{||} = \gamma^3 m_0\vec{\mathbf{a}}_{||}, \qquad \vec{\mathbf{F}}_{\perp} = \gamma m_0\vec{\mathbf{a}}_{\perp}. \tag{1.11}$$

1.1.2. Lorentz Force

Force acting on a charged particle moving in electromagnetic fields is called Lorentz Force and is given by

$$\frac{d\vec{p}}{dt} = \vec{F}_L = q(\vec{E} + \vec{v} \times \vec{B}), \tag{1.12}$$

where the q is the charge and $\vec{v}$ is the velocity of the particle.

1.1.3. Relativistic Lorentz Force

Similar to non-relativistic version, relativistic Lorentz Force is given by the 4-vector equality

$$\frac{dp^\mu}{d\tau} = qF^{\mu\nu}u_\nu, \tag{1.13}$$

where $d\tau = dt/\gamma$, greek letters are Lorentz indices and p, F, u are

$$p^\mu = \begin{bmatrix} W/c \\ p_x \\ p_y \\ p_z \end{bmatrix}, \qquad F^{\mu\nu} = \begin{bmatrix} 0 & -E_x/c & -E_y/c & -E_z/c \\ E_x/c & 0 & -B_z & B_y \\ E_y/c & B_z & 0 & -B_x \\ E_z/c & -B_y & B_x & 0 \end{bmatrix}, \qquad u_\nu = \gamma \begin{bmatrix} c \\ -v_x \\ -v_y \\ -v_z \end{bmatrix},$$

where W is the energy of the particle and γ is the Lorentz factor mentioned in the previous section.

For $\mu = 0$, we have the time component of the equation as

$$\frac{\gamma}{c}\frac{dW}{dt} = \frac{q\gamma\vec{E}\cdot\vec{v}}{c} = \frac{q\gamma}{c}\frac{\vec{E}\cdot d\vec{r}}{dt}, \tag{1.14}$$

$$\frac{dW}{dt} = q\frac{\vec{E}\cdot d\vec{r}}{dt}. \tag{1.15}$$

This is the definition of work done by an electric field. For $\mu = 1, 2, 3$, we have the spacial components

$$\frac{d\vec{p}}{d\tau} = \gamma \frac{d\vec{p}}{dt} = q\gamma(\vec{E} + \vec{v} \times \vec{B}),$$

which simplifies to non-relativistic Lorentz Force in Equation 1.12.

1.1.4. Acceleration caused by Lorentz force

Due to the nature of the cross product, Lorentz force caused by a magnetic field is always perpendicular to the velocity of the particle. Therefore the acceleration of the magnetic field is straightforward,

$$\vec{\mathbf{F}}_B = \vec{\mathbf{F}}_\perp = \gamma m_0 \vec{\mathbf{a}}_\perp = \gamma m_0 \vec{\mathbf{a}}_B.$$

The same thing cannot be said about electric field however. It can create force in any direction with respect to velocity. Therefore, we have the equality

$$\vec{\mathbf{a}}_{||} = \frac{q}{\gamma^3 m_0} \vec{\mathbf{E}}_{||}, \qquad \vec{\mathbf{a}}_\perp = \frac{q}{\gamma m_0} \{\vec{\mathbf{E}}_\perp + \vec{\mathbf{v}} \times \vec{\mathbf{B}}\}. \tag{1.16}$$

Acceleration due to electric field can be simplified as

$$\begin{aligned} \vec{\mathbf{a}}_{\{B=0\}} = \vec{\mathbf{a}}_E &= \vec{\mathbf{a}}_{||} + \vec{\mathbf{a}}_{\perp\{B=0\}} \\ &= \frac{q}{m_0\gamma} \{\frac{\vec{\mathbf{E}}_{||}}{\gamma^2} + \vec{\mathbf{E}}_\perp\} \\ &= \frac{q}{m_0\gamma} \{\{1-\beta^2\}\vec{\mathbf{E}}_{||} + \vec{\mathbf{E}}_\perp\} \\ &= \frac{q}{m_0\gamma} \{\vec{\mathbf{E}}_{||} + \vec{\mathbf{E}}_\perp - \beta^2 \vec{\mathbf{E}}_{||}\} \\ &= \frac{q}{m_0\gamma} \{\vec{\mathbf{E}} - \beta^2 \vec{\mathbf{E}}_{||}\}. \end{aligned} \tag{1.17}$$

Using the fact that $\vec{\mathbf{E}}_{||} = \vec{\beta}(\vec{\mathbf{E}} \cdot \vec{\beta})/\beta^2$, we finally have

$$\vec{\mathbf{a}}_E = \frac{q}{\gamma m_0} \{\vec{\mathbf{E}} - \vec{\mathbf{v}} \frac{(\vec{\mathbf{E}} \cdot \vec{\mathbf{v}})}{c^2}\}, \qquad \vec{\mathbf{a}}_B = \frac{q}{\gamma m_0} (\vec{\mathbf{v}} \times \vec{\mathbf{B}}), \tag{1.18}$$

where $\vec{\mathbf{a}}_E$ consists of two components in the direction of $\vec{\mathbf{E}}$ and $\vec{\mathbf{v}}$. The component in the direction of $\vec{\mathbf{v}}$ depends on the $\vec{\mathbf{E}} \cdot \vec{\mathbf{v}}$, resulting in $\vec{\mathbf{a}}_E || \vec{\mathbf{E}}$ if $\vec{\mathbf{E}} \perp \vec{\mathbf{v}}$.

1.2. Particle Accelerators

Particle accelerators are sophisticated scientific instruments designed to accelerate charged particles, such as electrons, protons, or ions, to high speeds and energies. These accelerators play a crucial role in advancing our understanding of the fundamental properties of matter and the universe. They are widely used in various fields of research, including particle physics, nuclear physics, materials science, and medicine.

At their core, particle accelerators utilize electromagnetic fields to impart energy to particles and control their trajectories. These fields are generated by intricate arrangements of electromagnets and RF (radiofrequency) cavities within the accelerator structure. By precisely controlling these fields, accelerators can propel particles to speeds close to the speed of light, thus to high energies.

Accelerators can be categorized into two main types: linear accelerators (linacs) and circular accelerators. Linacs accelerate particles in a straight line, while circular accelerators use magnetic fields to bend the particle trajectory into a circular path.

The acceleration process in accelerators involves multiple stages. Initially, particles are injected into the accelerator at a relatively low energy. As they progress through the accelerator, they are subjected to electric fields that accelerate them, while magnetic fields guide their trajectories. Focusing elements, such as solenoid, dipole or quadrupole magnets, ensure the particles remain tightly controlled.

As particles gain energy in the accelerator, they approach relativistic speeds, where relativistic effects become significant. Special relativity governs the increase in mass and energy of the particles as they approach the speed of light, providing insights into the behavior of matter at high energies.

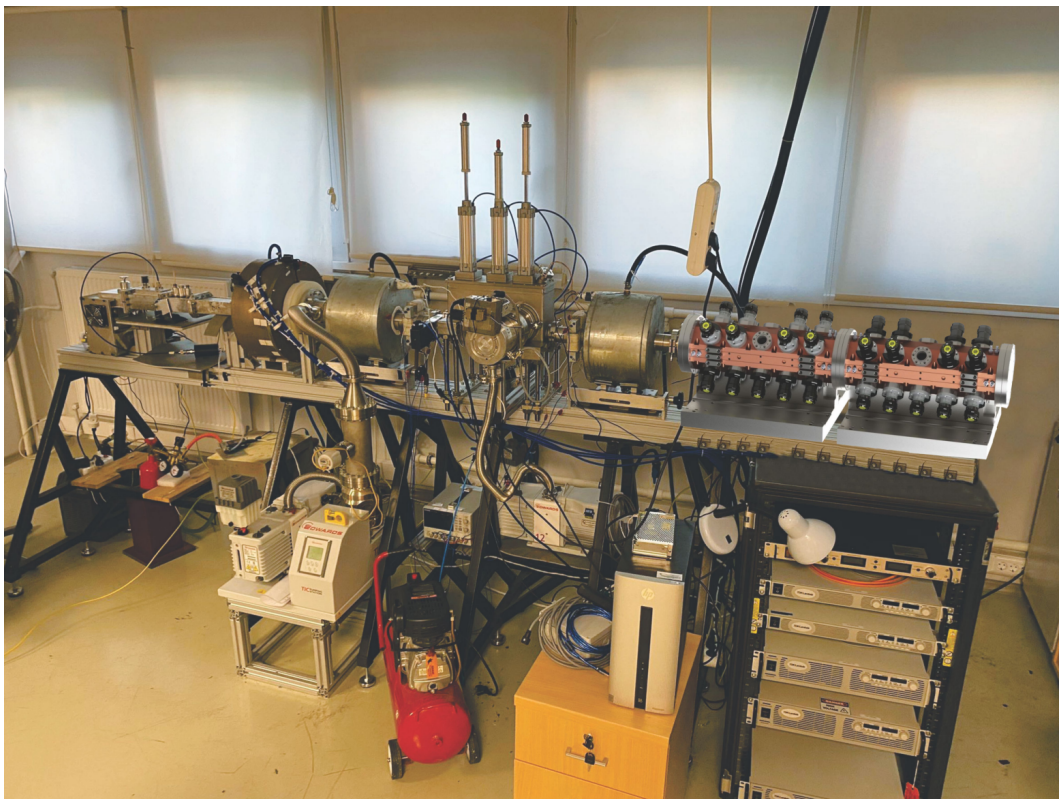


Figure 1.1. A linear proton accelerator in KAHVELab [2].

1.2.1. Acceleration Cavities

Radiofrequency (RF) cavities, also known as accelerating cavities or resonant cavities, are key components in particle accelerators. These cavities generate strong electromagnetic fields at specific frequencies to accelerate charged particles through clever engineering.

RF cavities are typically hollow metallic structures made of or coated with high-conductivity materials such as copper. The cavity is often cylindrical or spherical in shape, and its inner surface is polished to minimize energy losses through resistive heating. The RF cavity is designed to be resonant, meaning that it naturally amplifies the electric fields at its resonant frequency. The resonant frequency is determined by the cavity's dimensions and the speed of light in the cavity material.

To achieve efficient energy transfer to the particles, the RF cavity is driven by an external RF power source operating at the resonant frequency. The power source supplies radiofrequency energy to the cavity, which causes the electric fields inside the cavity to oscillate at the desired frequency. These oscillating fields then transfer energy to the passing particles, increasing their kinetic energy by pushing and pulling on the charged particles as they pass through the cavity.

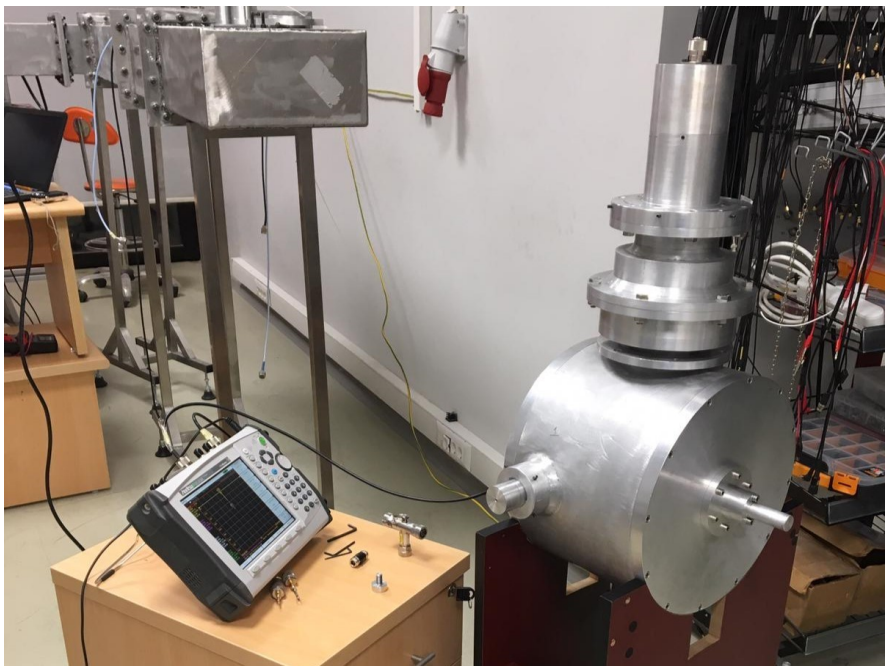


Figure 1.2. An RF cavity used in KAHVELab.

In addition to accelerating the particles, RF cavities are often designed to provide focusing forces. By carefully shaping the cavity and adjusting the electromagnetic fields, the particles can experience focusing effects as they pass through the cavity. This helps to maintain a tight and controlled beam. To ensure efficient acceleration, it is essential to maintain phase stability. This means that the particles should experience the strongest electric fields at the correct time during their passage through the cavity. Precise timing and synchronization of the RF power source with the particle beam are crucial to achieve phase stability and maximize energy transfer.

1.2.2. Bending Magnets

Bending magnets, also known as dipole magnets, are fundamental components used in particle accelerators to control the trajectory of charged particles. They utilize the Ampere's Law to exert a magnetic field that interacts with the charged particles in the accelerator.

According to the Lorentz Force Law (Section 1.1.2), when a charged particle moves through a magnetic field, it experiences a force perpendicular to both its velocity vector and the magnetic field direction. This force causes the particle's trajectory to curve, resulting in a bending effect.

1.2.3. Key Concepts

1.2.3.1. Resonance Frequency. Resonance frequency is the frequency in which the electromagnetic fields form standing waves in a cavity. It is determined by the physical dimensions and the speed of light in the cavity's medium. In a rectangular cavity for example, the resonance frequency can be calculated by

$$f_{klm} = \frac{c}{2\pi\sqrt{\mu_r \epsilon_r}} \sqrt{(\frac{k}{w})^2 + (\frac{l}{u})^2 + (\frac{m}{v})^2}, \tag{1.19}$$

where w, u, v are the dimensions of the cavity, μ_r and ϵ_r are relative permability and permittivity of the cavity respectably.

1.2.3.2. Bunch. Bunch refers to a tightly grouped collection of charged particles, such as electrons or protons, that are accelerated and maintained close together within a particle accelerator.

1.2.3.3. Phase Stability. Phase stability refers to the preservation of the timing relationship between particles and fields within an accelerator. It ensures that the phases of various electromagnetic fields or particles remain synchronized, which is crucial for

achieving efficient particle acceleration. Maintaining phase stability is essential to prevent particles from becoming out of phase as they travel through accelerator structures, ensuring that they receive the correct energy boosts and interact predictably with detectors. Deviations in phase stability can lead to particle loss and decreased beam quality.

1.2.3.4. Phase Lag. Phase lag refers to the time delay between the oscillations of two interacting waveforms or particles. It describes the difference in phase angles within their respective cycles, between two signals.

1.2.3.5. Shunt Impedance. The shunt impedance of an RF accelerator is a measure of the efficiency at which the accelerator can transform the supplied RF power into acceleration. It is defined as

$$Z_s = \frac{V_{acc}^2}{P_{diss}}, \tag{1.20}$$

where V_{acc} is the accelerating potential in which the particle is subjected to, P_{diss} is the power dissipated on the cavity walls. An example *shunt impedance* calculation can be found in Section 1.2.4.2.

1.2.4. Rhodotron Accelerator

Rhodotron Accelerator is a type of particle accelerator that was proposed by Jacques POTTIER in 1989 [1]. First prototype was built at CEA Saclay later in 1992 [3]. It is named after the greek word for rose, *rhodos*, due to the shape of the design [4].

The design of a rhodotron mainly consists of a coaxial cylindrical RF cavity and bending magnets surrounding it. RF cavity is fed by an external RF source, accelerating the electrons entering from an attached electron injector.

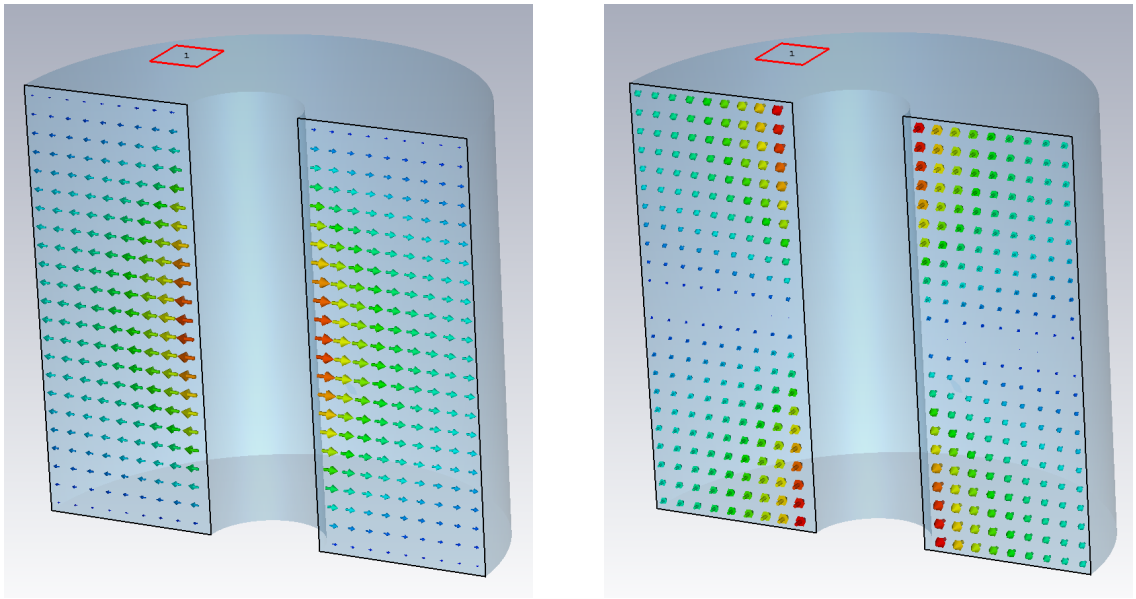
1.2.4.1. Acceleration Cycle of Rhodotron. Electrons undergo four different stages inside the accelerator. They are accelerated between the cylinderical plates and are shielded from the changing RF field while inside the inner cylinder and outside the cavity. This stages are explained further below.

- *First Acceleration:* Electrons in the rhodotron cavity are accelerated by the electric field created between two coaxial cylinders, towards the inner cylinder when they are ejected into the cavity (Figure 1.7).
- *Inner Shielding:* Inner shielding While inside the inner cylinder, the cylinder acts as a faraday cage and shields the electrons inside while the electric field is being reversed (Figure 1.8).
- *Second Acceleration:* Once the electrons leave the inner cylinder, they accelerate towards the outer cylinder by the reversed electric field until they leave the cavity (Figure 1.9).
- *Recirculating Magnets:* After leaving the cavity, an electromagnet placed in their path steers the electrons back into the cavity in which time the electric field changes the direction again (Figure 1.10).

This cycle can be repeated as long as real world constraints such as; placements and dimensions of the electromagnets, power requirements due to increasing magnetic field in order for sharper turns, can be overcomed. After the desired amount of cycles, also called passes, has been completed, the electrons exit the accelerator.

This process is explained further in the Figures 1.7, 1.8, 1.9 and 1.10 where T is the period of the electric field.

1.2.4.2. Cavity of a Rhodotron. Coaxial design of the cavity concentrates the electric field, while the magnetic field diminishes in the middle of the cylinders. Therefore the electrons are injected and accelerated in the plane of zero magnetic field where the electric field is strongest (Figure 1.3).



(a) $\vec{\mathbf{E}}$ field (b) $\vec{\mathbf{B}}$ field

Figure 1.3. $\vec{\mathbf{E}}$ and $\vec{\mathbf{B}}$ eigenmode field distributions inside a coaxial cavity.

For a cavity defined by the volume between two coaxial cylinders of equal lengths (h) with radii of R_1 and R_2, where $R_1 < R_2$, located at the origin (Figure 1.4), first eigenmode solution of the E and B fields are [1]

$$E = \frac{E_0}{r}\cos\left(\frac{\pi z}{h}\right)\sin\left(\omega t + \phi\right), \tag{1.21}$$

$$B = \frac{B_0}{r}\sin\left(\frac{\pi z}{h}\right)\cos\left(\omega t + \phi\right), \tag{1.22}$$

where $\omega = 2\pi f$, f is the resonance frequency.

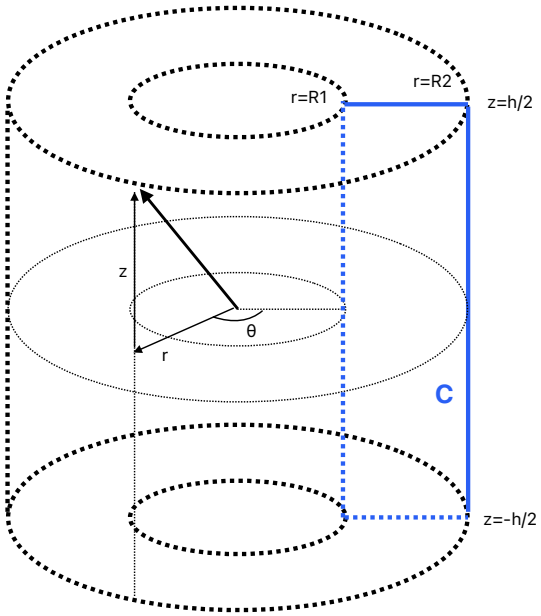


Figure 1.4. Illustration of a simple coaxial cavity with the curve **C** in Equation 1.24.

Because acceleration potential is located on the $z = 0$ plane and $\vec{\mathbf{E}} \parallel \hat{r}$, V_{acc} can be found by

$$\begin{aligned} V_{acc} &= 2\int_{R_1}^{R_2} |E|^2 dr \\ &= 2E_0 \int_{R_1}^{R_2} \frac{dr}{r} \\ &= 2E_0 \ln(\frac{R_2}{R_1}). \end{aligned} \tag{1.23}$$

Dissipated power P_{diss}, on the other hand, can be calculated as [5]

$$\begin{aligned} P_{diss} &= \frac{1}{2}\int\int \rho_s |H_{||}|^2 dA \\ &= \frac{\rho_s}{2\mu_0^2}\int_0^{2\pi}\int_C |B_{||}|^2 r ds d\theta. \end{aligned} \tag{1.24}$$

where ρ_s is the areal skin resistivity ($\rho_s \approx 2.51 \times 10^{-7} f^{1/2}$ for copper [1]), $B = \mu H$, μ is permeability, μ_0 is the vacuum permability. Integral curve **C** is defined as the

circumference of cylindrically symmetrical cross sectional area of the cavity (Figure 1.4). This curve can be seperated to its line components and total power dissipation of this curve, P_C will be equal to sum of the power dissipated in these lines. Since $\vec{\mathbf{B}}$ is always parallel to the surface we can use $\vec{\mathbf{B}}$ directly as

$$\begin{aligned} P_{diss} &= \frac{\rho_s}{2\mu_0^2}\int_0^{2\pi}(2\int_0^{\frac{h}{2}}|B|^2 r dz\Big|_{r=R_1} + 2\int_0^{\frac{h}{2}}|B|^2 r dz\Big|_{r=R_2} + 2\int_{R_1}^{R_2}|B|^2 r dr\Big|_{z=\frac{h}{2}})d\theta \\ &= \frac{\rho_s}{2\mu_0^2}\int_0^{2\pi}(2I_A + 2I_B + 2I_C)d\theta = \frac{2\rho_s\pi}{\mu_0^2}(I_A + I_B + I_C), \qquad (1.25) \\ I_A &= \int_0^{\frac{h}{2}}|B|^2 r dz\Big|_{r=R_1} = B_0^2\frac{h}{4R_1}, \qquad (1.26) \\ I_B &= \int_0^{\frac{h}{2}}|B|^2 r dz\Big|_{r=R_2} = B_0^2\frac{h}{4R_2}, \qquad (1.27) \\ I_C &= \int_{R_1}^{R_2}|B|^2 r dr\Big|_{z=\frac{h}{2}} = B_0^2\ln(\frac{R_2}{R_1}). \qquad (1.28) \end{aligned}$$

Inserting $H_0 = B_0/\mu_0$, finally we have the dissipated power

$$P_{diss} = \rho_s\pi H_o(\frac{h}{2R_1} + \frac{h}{2R_2} + 2\ln(\frac{R_2}{R_1})). \qquad (1.29)$$

Therefore, from Equation 1.20, using Equations 1.23 and 1.29 also $E_0/H_0 = Z_0 \approx 120\pi$ we have

$$\begin{aligned} Z_s &= \frac{4E_0^2}{H_0^2\pi\rho_s}\frac{\ln^2(\frac{R_2}{R_1})}{(\frac{h}{2R_1} + \frac{h}{2R_2} + 2\ln(\frac{R_2}{R_1}))} \\ &= \frac{8\pi 60^2}{\rho_s}\frac{\ln^2(\frac{R_2}{R_1})}{(\frac{h}{4}(\frac{1}{R_1} + \frac{1}{R_2}) + \ln(\frac{R_2}{R_1}))}, \qquad (1.30) \end{aligned}$$

where, time dependencies have been removed from $\vec{\mathbf{E}}$ and $\vec{\mathbf{B}}$ and the maximum values for V_{acc} and P_{diss} have been used. However, particles do not interact with constant $\vec{\mathbf{E}}$ field during acceleration. Therefore, a more useful parameter called *effective shunt impedance* can be defined as

$$Z_{se} = Z_s T^2, \qquad (1.31)$$

where T is the *transit time factor*, a correctional coefficent that contain the changing field effects during acceleration.

For a relativistic electron crossing the axis at time 0, T can be found by [1]

$$T = \frac{S_i(\frac{2\pi R_2}{\lambda}) - S_i(\frac{2\pi R_1}{\lambda})}{\ln(\frac{R_2}{R_1})}, \tag{1.32}$$

$$S_i(\theta) = \int_0^\theta \frac{\sin(x)}{x} dx. \tag{1.33}$$

For a relativistic electron crossing the axis at $2\pi f t_0 = \phi$ on the other hand, T needs to be multiplied by $\cos(\phi)$ as

$$T(\phi) = T\cos(\phi). \tag{1.34}$$

Putting all these calculations together, energy gain of a relativistic electron, passing the origin at ϕ/ω as calculated by *J. Pottier* is [1]

$$\begin{aligned} \Delta E &= qV_{acc}^{ef} \\ &= qZ_{se}^{1/2} P_{diss}^{1/2} \cos(\phi), \end{aligned} \tag{1.35}$$

where $V_{acc}^{ef} = V_{acc}T(\phi)$ is the effective accelerating potential. If ΔE is taken in MeV, Z_{se} in $M\Omega$ and P_{diss} in MW, this equality becomes

$$\Delta E = Z_{se}^{1/2} P_{diss}^{1/2} \cos(\phi) \quad MeV. \tag{1.36}$$

With the expectation that the electrons will accelerate to speeds $\approx c$ after the first pass, a rhodotron cavity is designed so that the length of the path between successive passes is an integer multiple of λ, wavelength of the RF field ($l = p\lambda$). This constraint helps with phase stability and synchronization of the beam. In the Figure 1.5, optimized characteristics of a rhodotron cavity can be observed.

Table 1
Optimized characteristics

P	R_2 (m)	R_1/R_2	Z_{se} (MΩ)	Z_{sp} (MΩ)
1	0.27λ	1/4	$5.77\lambda^{1/2}$	$4.9\lambda^{1/2}$
2	0.5λ	1/7	$10.4\lambda^{1/2}$	$8.83\lambda^{1/2}$

Figure 1.5. Optimized characteristics of a rhodotron cavity [1].

Here, p is the integer multiplier in the equation ($l = p\lambda$) mentioned above, R_1 is the radius of the inner cylinder, R_2 is the radius of the outer cylinder, Z_{se} is effective shunt empedance, Z_{sp} is practical shunt empedance which was taken to be $0.85Z_{se}$. Typically, phase lag ϕ is taken as 15° [1].

Considering $Z_{sp} \propto \lambda^{1/2}$, $\quad \Delta E \propto \lambda^{1/4}$, $\quad V \propto \lambda^3$, where V is the volume of the cavity, implementing the $p = 1$ design in Figure 1.5 is much more space efficient.

Table 2
Energy W (MeV) for $P = 100$ kW and $f = 130$ MHz

R_2 (m)	2	3	4	5	6	7	8	9	10	11	12
0.62	1.8	2.7	3.6	4.5	5.4	6.3	7.2	8.1	9	9.9	10.8
1.15	2.4	3.6	4.8	6	7.2	8.4	9.6	10.8	12	13.2	14.4

Figure 1.6. Energy of a synchronous electron after each pass for both $p = 1$ and $p = 2$ [1].

Total energy gain after n passes, ΔE_n, can then be found by Equation 1.36, taking $\phi = 15°$, $p = 1$, i.e $R_2 = 0.27\lambda$, P in W, λ in m as

$$\Delta E_n \approx 2.14\lambda^{1/4} P^{1/2} n \quad keV. \tag{1.37}$$

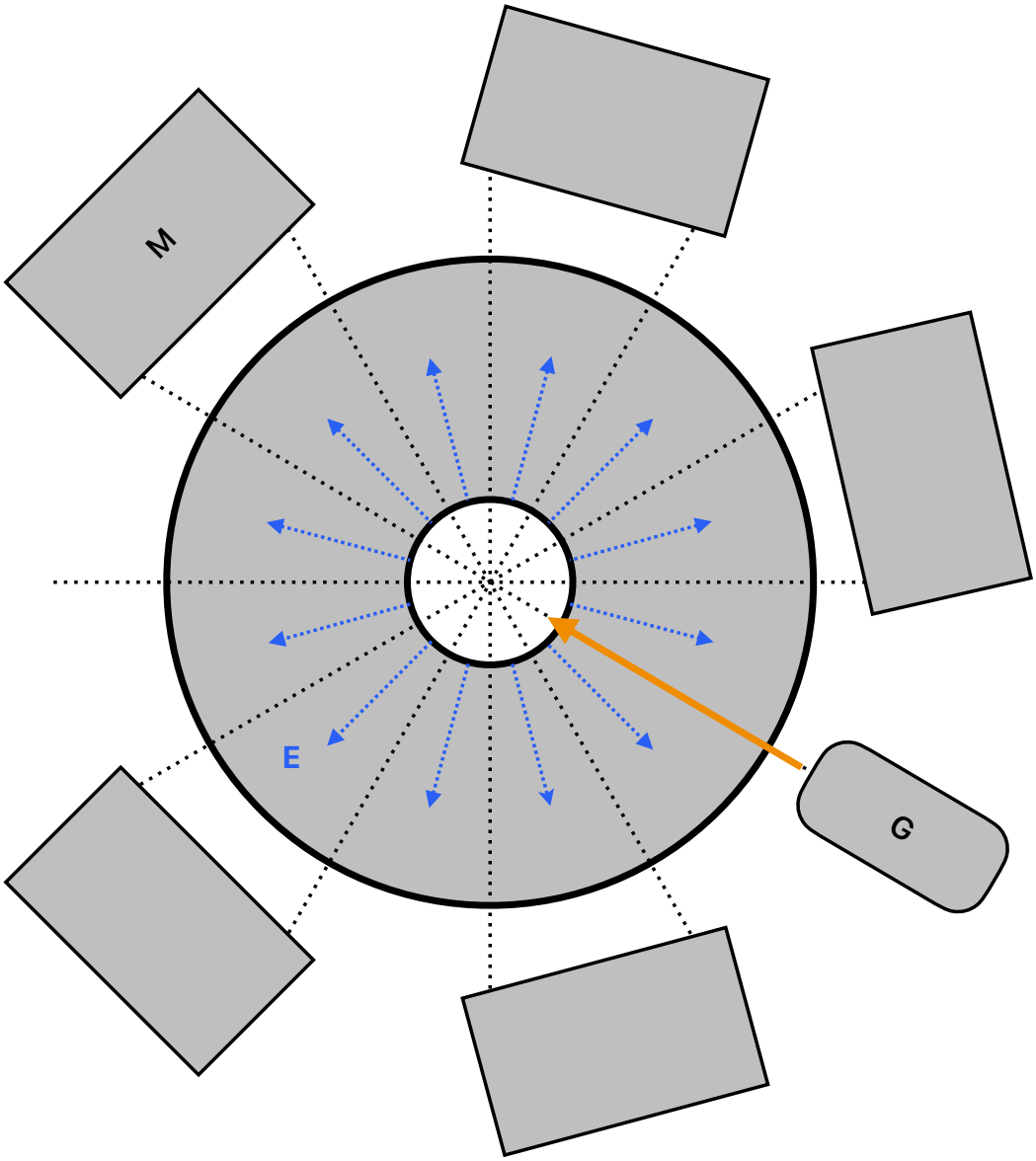


Figure 1.7. $[0, \frac{T}{4}]$ time frame of a rhodotron.

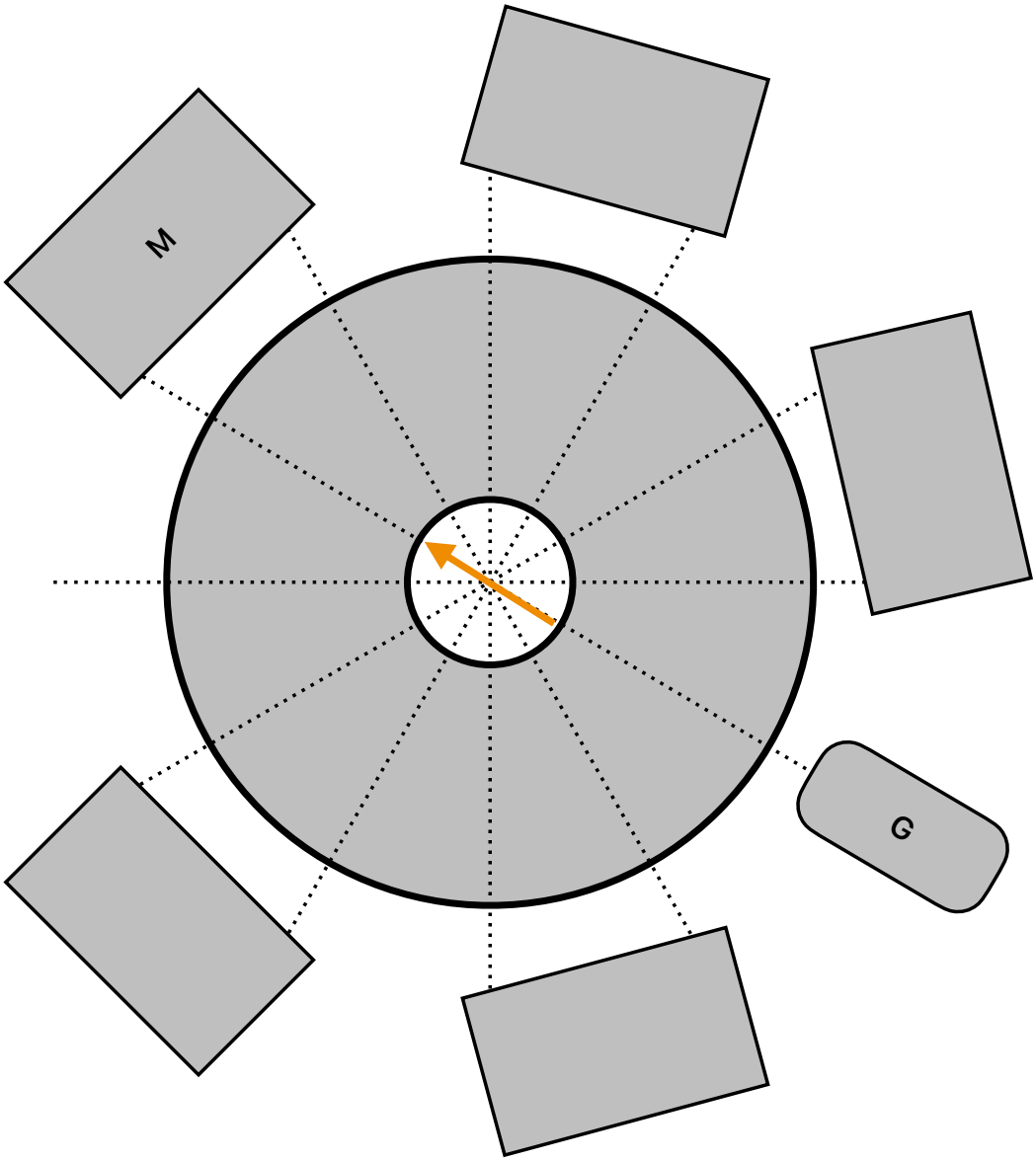


Figure 1.8. $[\frac{T}{4}, \frac{T}{2}]$ time frame of a rhodotron.

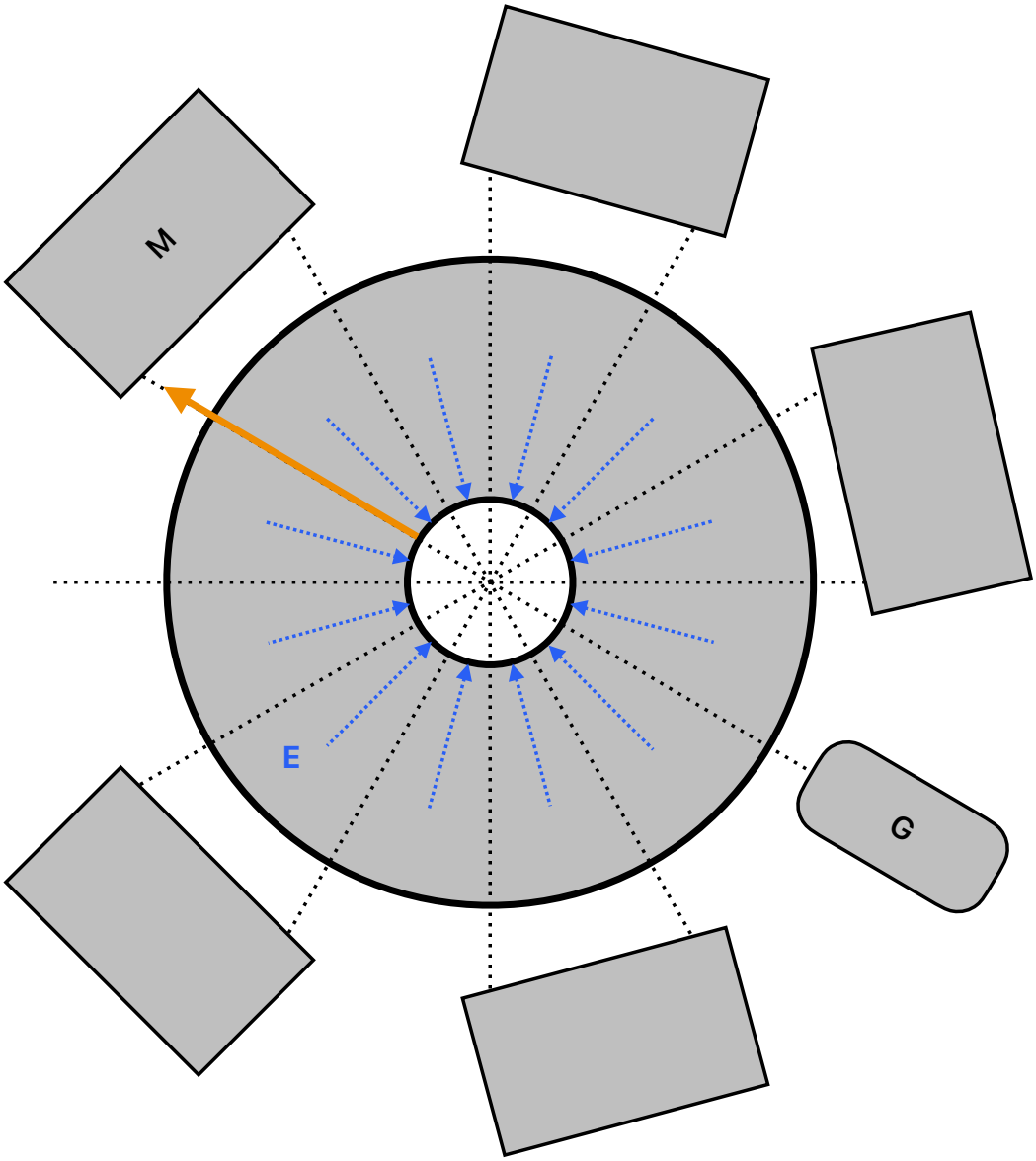


Figure 1.9. $[\frac{T}{2}, \frac{3T}{4}]$ time frame of a rhodotron.

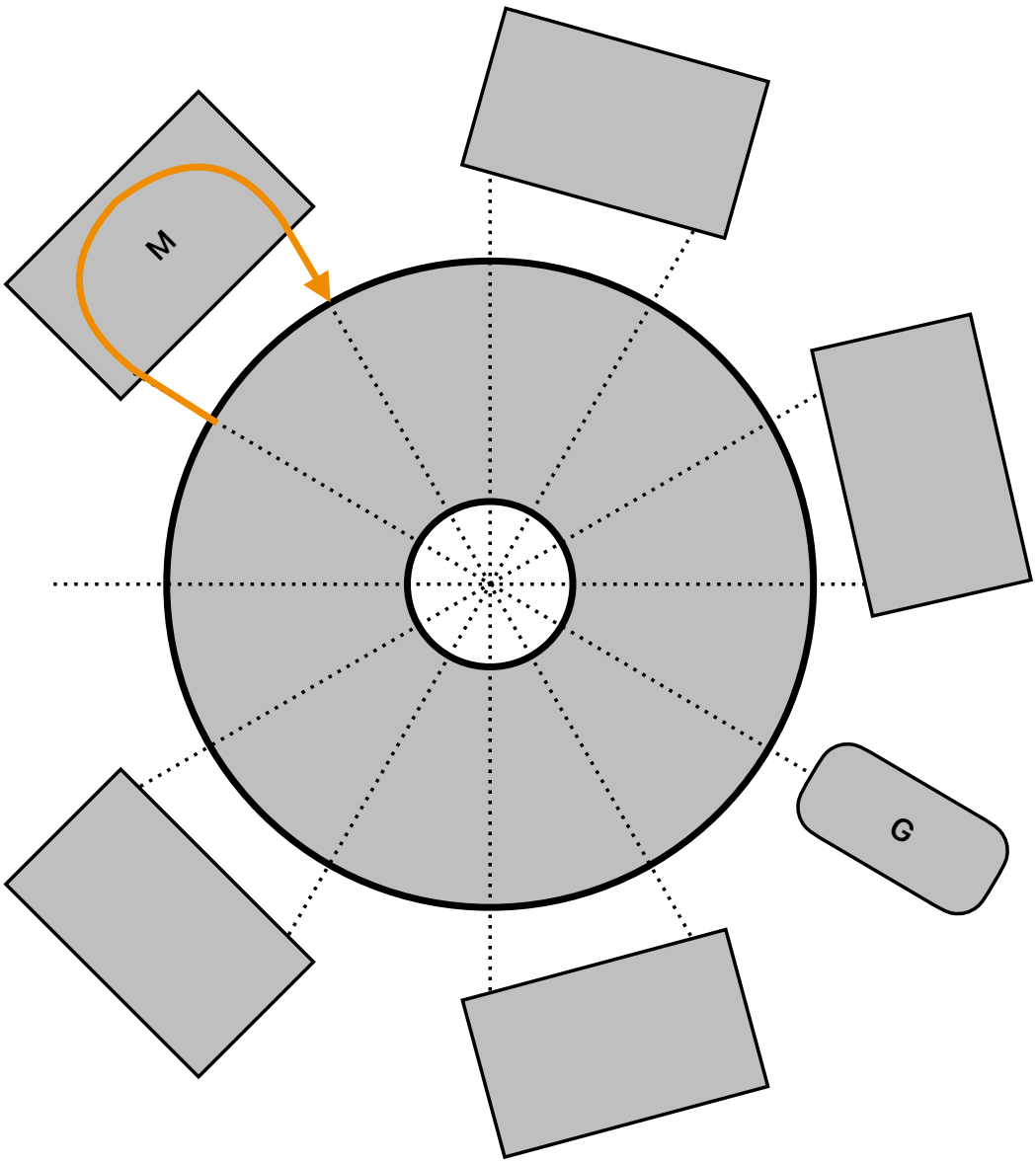


Figure 1.10. $[\frac{3T}{4}, T]$ time frame of a rhodotron.

1.3. Basic Concepts in Programming

1.3.1. Clock Cycle

In programming, a clock cycle refers to a fundamental unit of time measurement used in computer systems. It represents the basic rhythm or timing mechanism of a computer's central processing unit (CPU) and is typically measured in terms of the CPU's clock speed, expressed in hertz (Hz).

A clock cycle represents one complete pulse or oscillation of the CPU's clock signal. It serves as a synchronization mechanism, coordinating the execution of instructions and the timing of various operations within the CPU. Each clock cycle is associated with a specific duration, which is determined by the clock speed of the CPU.

The concept of clock cycles is often used when considering the performance and efficiency of algorithms and code. Since the execution time of instructions is influenced by the number of clock cycles required, minimizing the number of clock cycles needed for a program or algorithm can lead to faster and more efficient code execution. Table below shows the amount of clock cycles required for some mathematical calculations in various processors.

Note that division is by far the most time consuming basic mathematical operation between addition, subtraction and multiplication. Although not discussed in the following chapters, mathematical operations were reduced to addition, subtraction and multiplication when found to be possible while developing computationally intensive software in Chapter 4.

MEASURED CPU TIMES OF BASIC MATHEMATICAL OPERATIONS

	DOUBLE PRECISION					QUADRUPLE PRECISION	
OPERATION	i486	Pentium	PA-7000	μSPARC-II	Alpha	PA-7000	μSPARC-II
$x = y$	0.46	0.24	1.00	1.00	0.41	5.1	2.8
$x = y + z$	0.91	1.00	1.00	1.00	1.00	131	11.3
$x = yz$	1.09	1.00	1.00	1.00	1.00	457	99.5
$x = y/z$	2.81	4.49	4.92	3.32	6.59	1100	168
$x = y^{1/2}$	12.0	19.6	4.95	24.4	38.5	1100	149
$x = \sin y$	19.8	23.3	49.8	22.0	18.5	9510	1740
$x = \log y$	20.2	23.9	47.8	31.9	22.0	5420	1240
$x = \tan^{-1} y$	19.8	27.3	46.3	37.3	23.5	5930	1270
$x = \tan y$	18.0	28.3	63.2	30.0	35.1	8450	2040
$x = \exp y$	24.9	32.7	42.0	19.0	23.2	6560	1370

Figure 1.11. Amount of clock cycle for mathematical operations in various processors [6].

1.3.2. Concurrency

Concurrency in programming refers to the ability of a program to execute multiple tasks or processes simultaneously. It allows different parts of a program to make progress independently, potentially improving performance, responsiveness, and resource utilization.

Concurrency in single core can be achieved by implementing clever scheduling of list of operations called threads. This results in non-blocking execution of multiple threads, but only one thread would be executed at any given time. True concurrency on the other hand, can only be achieved by using multiple cores. Each core would be able to execute one thread at any time.

Programs that implement concurrency using multiple cores executes more operations per clock cycle. However, this does not directly lead to an improved performance due to the heavy burden of scheduling and managing multiple threads.

Threads utilize mutexes, short for “mutual exclusion,” to synchronize and manage

access to shared resources in a multi-threaded or multi-process environment.

1.3.3. Object Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of objects. Definition called class serves as a blueprint that defines the structure and behavior of objects. Each object encapsulates both data, and the functions that manipulate that data. This encapsulation promotes modular code design and enhances data security by controlling access to the object's internal state.

In OOP, inheritance enables the creation of new classes based on existing ones, facilitating code reuse and hierarchy. Polymorphism allows objects of different classes to be treated uniformly through a common interface, enhancing flexibility and extensibility. By providing abstraction, OOP simplifies complex systems by focusing on essential features and interactions, making software development more manageable, scalable, and maintainable.

2. TOOLS

For the purpose of designing, enhancing, and optimizing particle accelerators, certain tools can be employed. The following sections will explore several of the software and algorithms that were put to use.

2.1. Simulation

A simulation software is a computer program or tool that enables the creation and execution of simulations to model and analyze real-world systems or processes. It allows users to replicate the behavior, interactions, and outcomes of the system or process under study, providing insights and predictions that can be valuable for decision-making, optimization, or understanding complex phenomena.

Simulation software provides a virtual environment where users can define the parameters, variables, and rules of the system being simulated. The software then uses mathematical and physical models, algorithms, and computational techniques to simulate the behavior of the system over time.

2.2. Available Tools

2.2.1. Poisson Superfish

Poisson Superfish is a software package developed by the Los Alamos National Laboratory for the design and analysis of electromagnetic fields, particularly those related to particle accelerators and high-energy physics experiments [7]. It is commonly used in the field of accelerator physics to simulate and optimize the behavior of charged particle beams as they pass through various electromagnetic structures, such as cavities and magnets.

2.2.2. CST Studio Suite

CST Studio Suite is a powerful software package developed by Simulia for electromagnetic simulation and analysis. It is widely used in various industries, including electronics, telecommunications, automotive, aerospace, and more, to design and optimize electromagnetic devices and systems. The software provides a comprehensive set of tools for simulating and analyzing the behavior of electromagnetic fields and their interactions with different materials, structures and charged particles.

2.2.3. ROOT

ROOT is a widely used open-source software framework developed by CERN (European Organization for Nuclear Research) for data analysis, visualization, and storage in the field of high-energy physics, particularly in experiments conducted at particle accelerators like the Large Hadron Collider (LHC) [8]. It has become an essential tool in particle physics research, and it is used by physicists around the world to analyze and interpret data.

2.2.4. gnuplot

GNUPLOT is a popular open-source software used for creating and visualizing data plots and graphs [9]. It is widely utilized in various fields, including scientific research, data analysis, engineering, and more, to represent and analyze data in a graphical format.

2.3. Algorithms

2.3.1. Leapfrog

The Leapfrog method is a numerical method commonly used to solve ordinary differential equations (ODEs) that involve second-order time derivatives. Such an ODE can be written as

$$\ddot{x} = \frac{d^2x}{dt^2} = f(x). \tag{2.1}$$

The Leapfrog method is a variant of the finite difference method, and it approximates the solution of an ODE by discretizing both time and space. The method gets its name from the way it calculates the values of the solution at each time step, which resembles a leapfrogging motion. It is a simple and efficient algorithm that is often used in simulations of physical systems, such as celestial mechanics or molecular dynamics. The idea is straight forward; in the time interval Δt,

$$\begin{aligned} a(t_0) &= f(x_0), & (2.2)\\ x(t_0+\Delta t) &= x(t_0) + v(t_0)\Delta t + a(t_0)\frac{\Delta t^2}{2}, & (2.3)\\ v(t_0+\Delta t) &= v(t_0) + \{a(t_0) + a(t_0+\Delta t)\}\frac{\Delta t}{2}. & (2.4) \end{aligned}$$

For more stability, this version can be rearranged to what is called 'kick-drift-kick' form as

$$\begin{aligned} v(t_0+\Delta t/2) &= v(t_0) + a(t_0)\frac{\Delta t}{2}, & (2.5)\\ x(t_0+\Delta t) &= x(t_0) + v(t_0+\Delta t/2)\Delta t, & (2.6)\\ v(t_0+\Delta t) &= v(t_0+\Delta t/2) + a(t_0+\Delta t)\frac{\Delta t}{2}. & (2.7) \end{aligned}$$

This version provides more time resolution to our calculation; however, it increases the number of calculations needed by about 50%.

2.3.2. Runge Kutta

The Runge-Kutta methods, named after the German mathematicians Carl Runge and Martin Wilhelm Kutta [10] [11], family of numerical methods used to solve ordinary differential equations (ODEs) that are in the form as

$$\frac{dy}{dx} = f(x, y). \tag{2.8}$$

The basic idea behind the Runge-Kutta method is to approximate the solution of an ODE by taking small steps and using a weighted average of function evaluations at different points within each step as

$$y_{n+1} = y_n + \delta x \sum_{i=1}^{m} b_i k_i, \qquad x_{n+1} = x_n + \delta x, \tag{2.9}$$

where

$$k_i = f(x_n + c_i \delta x, y_n + \delta x \sum_{j=1}^{i-1} a_{ij} k_j). \tag{2.10}$$

These Equations 2.9 and 2.10 define the family of methods. To specify a particular method, order m, coefficients a_{ij}, b_i and c_i should be provided. The coefficients of any Runge-Kutta method can be visualized by a tableau called Butcher Tableau which can be observed in Figure 2.1.

$$\begin{array}{c|cccc} 0 & & & & \\ c_1 & a_{21} & & & \\ c_2 & a_{31} & a_{32} & & \\ \vdots & & \cdots & & \\ c_m & a_{m1} & a_{m2} & \cdots & a_{m,m-1} \\ \hline & b_1 & b_2 & \cdots & b_m \end{array}$$

Figure 2.1. Butcher Tableau.

The simplest Runge-Kutta method is the Euler's method the Butcher tableau of which can be observed in Figure 2.2.

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Figure 2.2. Butcher tableau of Euler's method.

The most commonly used version of the Runge-Kutta method is the fourth-order Runge-Kutta method, also known as RK4. The RK4 method involves four function evaluations per step and has an error term that is proportional to the step size raised to the fifth power. Two of the most used Butcher tableaus for RK4 given in Figure 2.3. The second tableau in Figure 2.3 is called the "3/8 rule". Its main advantage is that its error coefficients are smaller than the other. But it costs more floating point operations per step. Resulting in slower calculations.

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \qquad \begin{array}{c|cccc} 0 & & & & \\ 1/3 & 1/3 & & & \\ 2/3 & -1/3 & 1 & & \\ 1 & 1 & -1 & 1 & \\ \hline & 1/8 & 3/8 & 3/8 & 1/8 \end{array}$$

Figure 2.3. Butcher Tableau for RK4.

3. DESIGN

In this chapter, the primary design factors of rhodotron-type accelerators are examined, along with an exploration of the design created within the *KAHVELab*. Rhodotron-type accelerators are composed of two principal components: a coaxial acceleration cavity and recirculating magnets. The cavity's design is of utmost significance for attaining resonance and improving beam-RF interactions, while the magnet design focuses on preserving the beam's phase stability.

3.1. Cavity Design

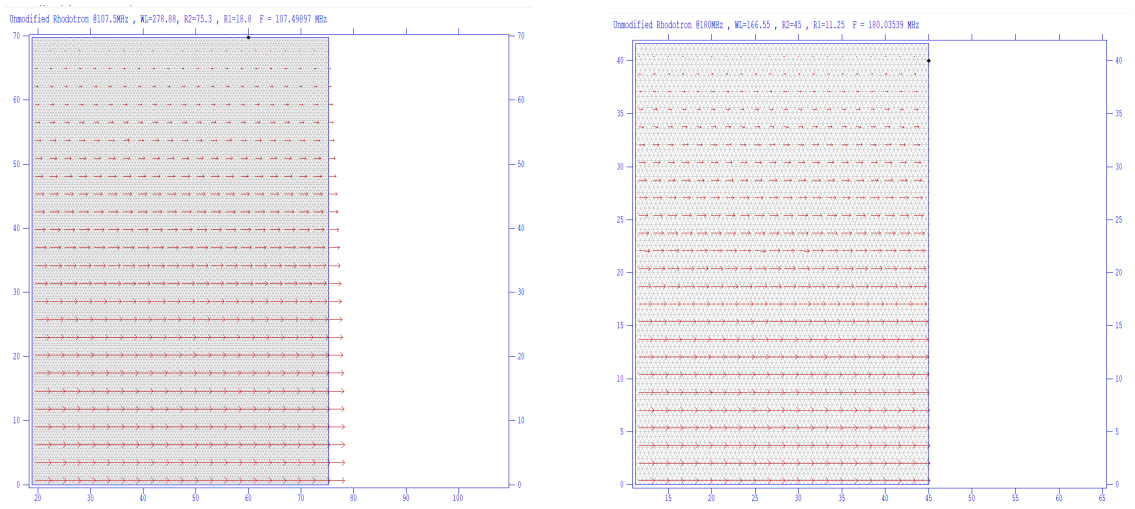
The first and the most important design parameter for a cavity is the operating RF frequency. After an operating RF frequency is set and the desired $R1/R2$ relation in Figure 1.5 is selected, design parameters of acceleration plane of the cavity is fully determined.

By following the cylinderical design mentioned in Section 1.2.4, the only main design parameter remaining is the height of the cylindrical cavity. This parameter can be found using the constraint mentioned in Section 1.2.1; the fact that operating RF frequency must be equal to resonant frequency of the cavity. For simple coaxial cavity, the height should be $\lambda/2$, where λ is the wavelength of the external RF supply [1]. Simulation tools such as CST Studio and Poisson Superfish can be used to confirm this condition.

In the following examples, $p = 1$ from Figure 1.5 was used, and it will be the focus of all further calculations. Using $f_{RF} = 107.5$ MHz & $f_{RF} = 180$ MHz for comparison:

$$\begin{aligned} f_{107.5} &= 107.5 \text{ MHz}, & f_{180} &= 180 \text{ MHz}, \\ \lambda_{107.5} &= \frac{c}{f_{107.5}} = 2.789m, & \lambda_{180} &= \frac{c}{f_{180}} = 1.666m, \\ R_2 &= 0.27 \times \lambda_{107.5} = 0.753m, & R_2 &= 0.27 \times \lambda_{180} = 0.450m, \\ R_1 &= \frac{R_2}{4} = 0.188m, & R_1 &= \frac{R_2}{4} = 0.113m, \\ h &= \frac{\lambda}{2} = 1.394m. & h &= \frac{\lambda}{2} = 0.833m. \end{aligned} \tag{3.1}$$

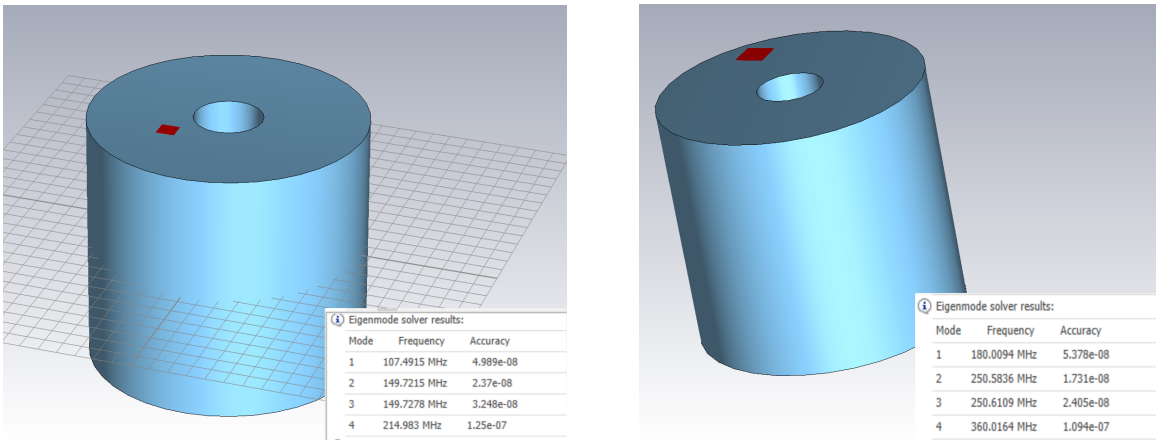
In the Figures 3.1, 3.2 and 3.3, Poisson Superfish and CST simulations of two cavities defined by Equation 3.1 can be observed.



(a) 107.5 MHz

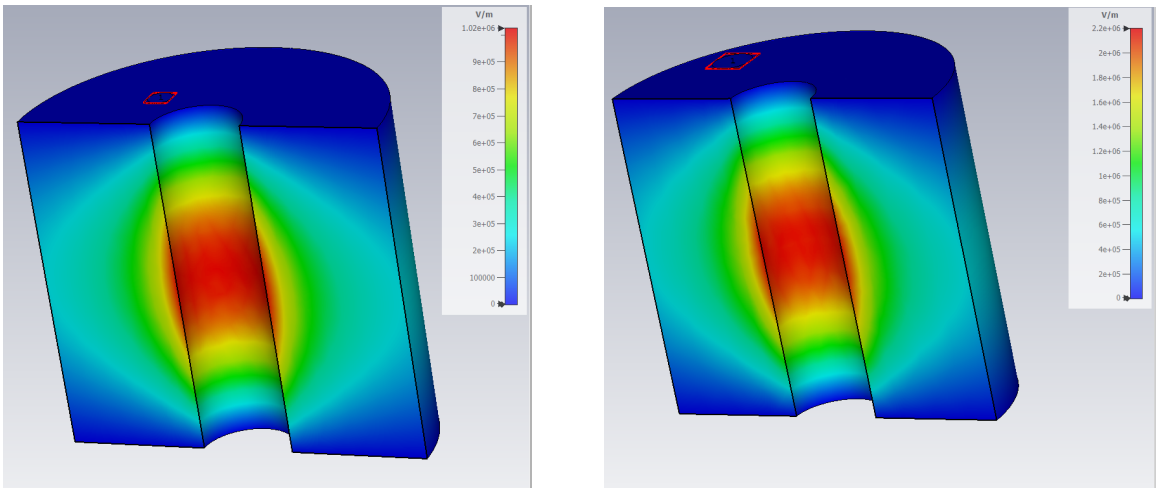
(b) 180 MHz

Figure 3.1. Poission Superfish results for Equation 3.1.



(a) 107.5 MHz (b) 180 MHz

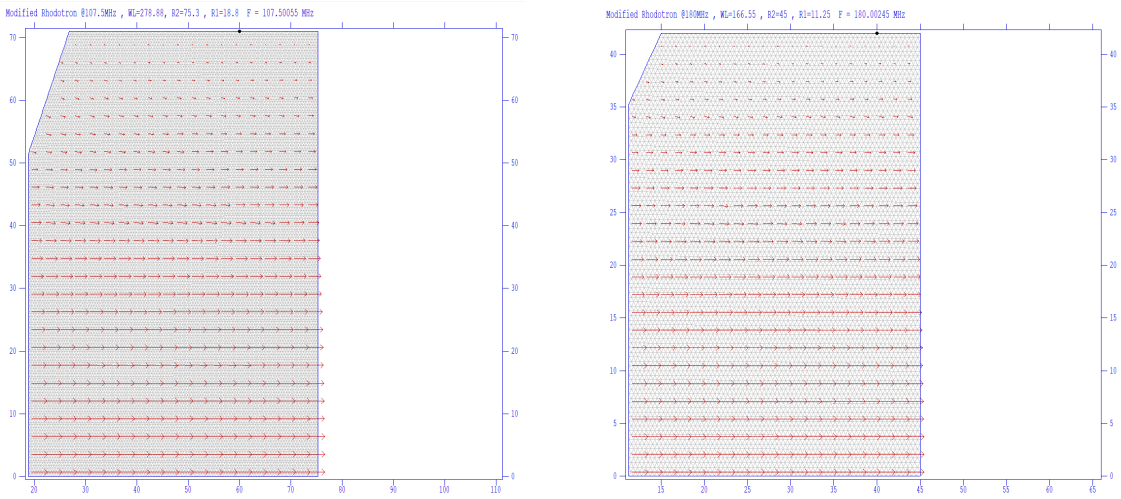
Figure 3.2. CST Eigenmode results for Equation 3.1.



(a) 107.5 MHz (b) 180 MHz

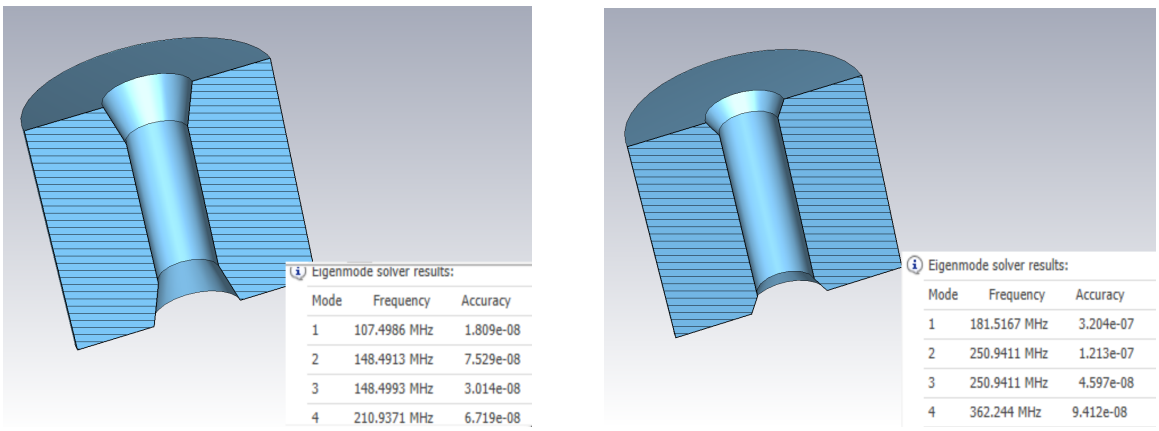
Figure 3.3. CST Electric field solutions for Equation 3.1.

As mentioned by Pottier, truncated cone terminations in inner cylinder can improve the shunt empedance Z of the cavity [1]. In the Figure 3.4, Poission Superfish results of such a modification with matching height increase to keep resonant frequency can be found.



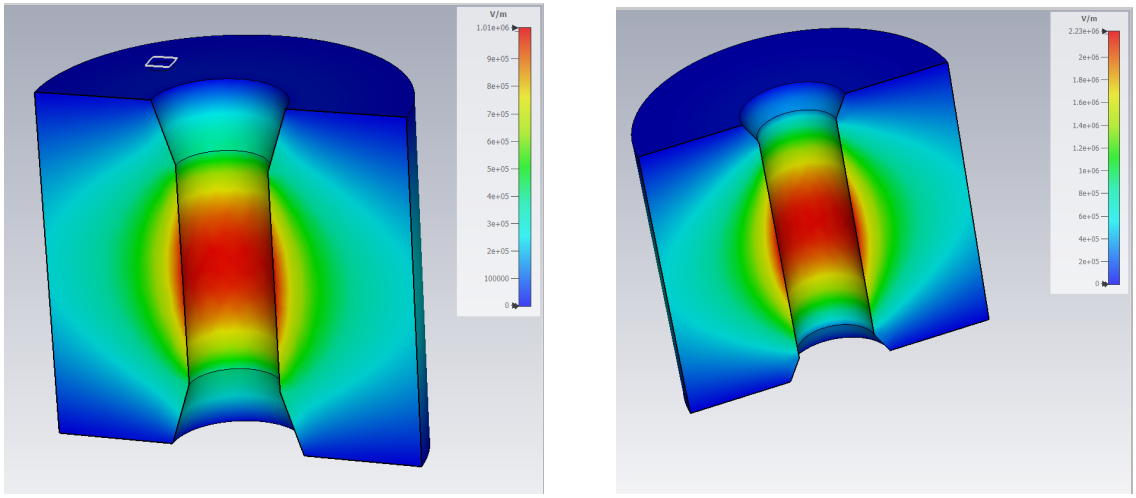
(a) 107.5 MHz truncated (b) 180 MHz truncated

Figure 3.4. Poisson Superfish field results of truncated cavities.



(a) 107.5 MHz truncated (b) 180 MHz truncated

Figure 3.5. CST eigenmode results of truncated cavities.



(a) 107.5 MHz truncated (b) 180 MHz truncated

Figure 3.6. CST electric field results of truncated cavities.

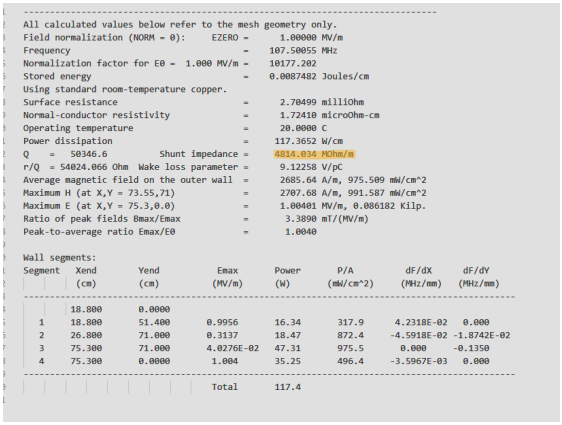
```
All calculated values below refer to the mesh geometry only.
Field normalization (NORM = 0):     EZERO =      1.00000 MV/m
Frequency                                =    107.49897 MHz
Normalization factor for E0 =  1.000 MV/m =    10001.202
Stored energy                            =    0.0087215 Joules/cm
Using standard room-temperature copper.
Surface resistance                       =      2.70497 milliOhm
Normal-conductor resistivity             =      1.72410 microOhm-cm
Operating temperature                    =      20.0000 C
Power dissipation                        =      120.2895 W/cm
Q    =   48972.0          Shunt impedance =    4697.002 MOhm/m
r/Q  = 54190.222 Ohm  Wake loss parameter =      9.15051 V/pC
Average magnetic field on the outer wall  =      2654.74 A/m, 953.183 mW/cm^2
Maximum H (at X,Y = 61.05,69.72)         =      2654.52 A/m, 953.03 mW/cm^2
Maximum E (at X,Y = 75.3,0.0)            =    0.999958 MV/m, 0.085835 Kilp.
Ratio of peak fields Bmax/Emax           =       3.3359 mT/(MV/m)
Peak-to-average ratio Emax/E0            =       1.0000

Wall segments:
Segment   Xend        Yend          Emax       Power      P/A          dF/dX      dF/dY
          (cm)        (cm)         (MV/m)      (W)      (mW/cm^2)     (MHz/mm)   (MHz/mm)
-------------------------------------------------------------------------------------------
          18.800      0.0000
   1      18.800      69.720       0.9999      33.22      476.5      1.1154E-06   0.000
   2      75.300      69.720      7.2038E-03   53.85      953.0       0.000      -0.1542
   3      75.300      0.0000        1.000      33.22      476.5      1.1140E-06   0.000
-------------------------------------------------------------------------------------------
                                    Total      120.3
```

(a) 107.5 MHz unmodified

```
All calculated values below refer to the mesh geometry only.
Field normalization (NORM = 0):     EZERO =      1.00000 MV/m
Frequency                                =    180.03539 MHz
Normalization factor for E0 =  1.000 MV/m =     9985.526
Stored energy                            =    0.0031127 Joules/cm
Using standard room-temperature copper.
Surface resistance                       =      3.50058 milliOhm
Normal-conductor resistivity             =      1.72410 microOhm-cm
Operating temperature                    =      20.0000 C
Power dissipation                        =      92.9927 W/cm
Q    =   37863.7          Shunt impedance =    3629.317 MOhm/m
r/Q  = 32350.061 Ohm  Wake loss parameter =      9.14856 V/pC
Average magnetic field on the outer wall  =      2655.62 A/m, 1.23436 W/cm^2
Maximum H (at X,Y = 43.511,41.63)        =      2654.93 A/m, 1.23372 W/cm^2
Maximum E (at X,Y = 45,0.0)              =    0.999842 MV/m, 0.070717 Kilp.
Ratio of peak fields Bmax/Emax           =       3.3368 mT/(MV/m)
Peak-to-average ratio Emax/E0            =       0.9998

Wall segments:
Segment   Xend        Yend          Emax       Power      P/A          dF/dX      dF/dY
          (cm)        (cm)         (MV/m)      (W)      (W/cm^2)      (MHz/mm)   (MHz/mm)
-------------------------------------------------------------------------------------------
          11.250      0.0000
   1      11.250      41.630       0.9998      25.68      0.6168     1.1704E-05   0.000
   2      45.000      41.630      1.5343E-02   41.64       1.234       0.000      -0.4325
   3      45.000      0.0000       0.9998      25.68      0.6168     1.1756E-05   0.000
-------------------------------------------------------------------------------------------
                                    Total      92.99
```

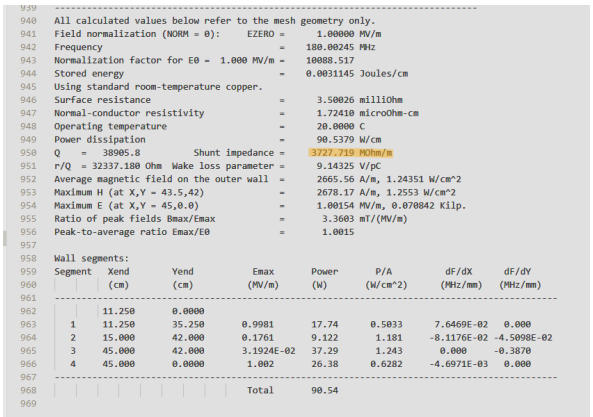
(b) 180 MHz unmodified

Figure 3.7. Poisson Superfish calculations with unmodified cavities.



```
---------------------------------------------------------------------------
All calculated values below refer to the mesh geometry only.
Field normalization (NORM = 0):     EZERO =      1.00000 MV/m
Frequency                                =    107.50055 MHz
Normalization factor for E0 =  1.000 MV/m =    10177.202
Stored energy                            =    0.0087482 Joules/cm
Using standard room-temperature copper.
Surface resistance                       =      2.70499 milliOhm
Normal-conductor resistivity             =      1.72410 microOhm-cm
Operating temperature                    =      20.0000 C
Power dissipation                        =      117.3652 W/cm
Q    =   50346.6          Shunt impedance =     4814.034 MOhm/m
r/Q  = 54024.066 Ohm  Wake loss parameter =      9.12258 V/pC
Average magnetic field on the outer wall  =      2685.64 A/m, 975.509 mW/cm^2
Maximum H (at X,Y = 73.55,71)            =      2707.68 A/m, 991.587 mW/cm^2
Maximum E (at X,Y = 75.3,0.0)            =      1.00401 MV/m, 0.086182 Kilp.
Ratio of peak fields Bmax/Emax           =       3.3890 mT/(MV/m)
Peak-to-average ratio Emax/E0            =       1.0040

Wall segments:
Segment   Xend        Yend          Emax       Power      P/A          dF/dX      dF/dY
           (cm)        (cm)         (MV/m)     (W)      (mW/cm^2)    (MHz/mm)   (MHz/mm)
-----------------------------------------------------------------------------------------
        18.800       0.0000
    1   18.800       51.400       0.9956       16.34      317.9      4.2318E-02   0.000
    2   26.800       71.000       0.3137       18.47      872.4     -4.5918E-02 -1.8742E-02
    3   75.300       71.000       4.0276E-02   47.31      975.5       0.000      -0.1350
    4   75.300       0.0000        1.004       35.25      496.4     -3.5967E-03   0.000
-----------------------------------------------------------------------------------------
                                              Total      117.4
```

(a) 107.5 MHz truncated



```
---------------------------------------------------------------------------
All calculated values below refer to the mesh geometry only.
Field normalization (NORM = 0):     EZERO =      1.00000 MV/m
Frequency                                =    180.00245 MHz
Normalization factor for E0 =  1.000 MV/m =    10088.517
Stored energy                            =    0.0031145 Joules/cm
Using standard room-temperature copper.
Surface resistance                       =      3.50026 milliOhm
Normal-conductor resistivity             =      1.72410 microOhm-cm
Operating temperature                    =      20.0000 C
Power dissipation                        =      90.5379 W/cm
Q    =   38905.8          Shunt impedance =     3727.719 MOhm/m
r/Q  = 32337.180 Ohm  Wake loss parameter =      9.14325 V/pC
Average magnetic field on the outer wall  =      2665.56 A/m, 1.24351 W/cm^2
Maximum H (at X,Y = 43.5,42)             =      2678.17 A/m, 1.2553 W/cm^2
Maximum E (at X,Y = 45,0.0)              =      1.00154 MV/m, 0.070842 Kilp.
Ratio of peak fields Bmax/Emax           =       3.3603 mT/(MV/m)
Peak-to-average ratio Emax/E0            =       1.0015

Wall segments:
Segment   Xend        Yend          Emax       Power      P/A          dF/dX      dF/dY
           (cm)        (cm)         (MV/m)     (W)      (W/cm^2)     (MHz/mm)   (MHz/mm)
-----------------------------------------------------------------------------------------
        11.250       0.0000
    1   11.250       35.250       0.9981       17.74      0.5033     7.6469E-02   0.000
    2   15.000       42.000       0.1761       9.122       1.181     -8.1176E-02 -4.5098E-02
    3   45.000       42.000       3.1924E-02   37.29       1.243       0.000      -0.3870
    4   45.000       0.0000        1.002       26.38      0.6282     -4.6971E-03   0.000
-----------------------------------------------------------------------------------------
                                              Total      90.54
```

(b) 180 MHz truncated

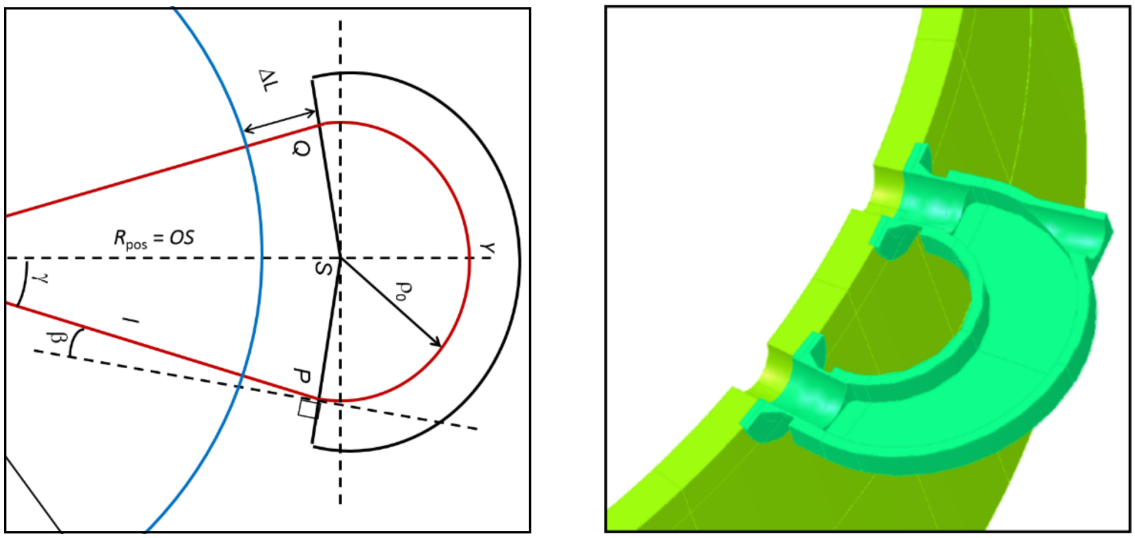
Figure 3.8. Poisson Superfish calculations with truncated cavities.

One can observe the shunt impedance gain between truncated and unmodified cavities in Figures 3.7 and 3.8 as

$$\Delta Z_{107.5} = 2.5\%, \qquad \Delta Z_{180} = 2.7\%. \tag{3.2}$$

3.2. Magnet Design

Magnet design in rhodotron type accelerators depend heavily on the design of the cavities. Because the limiting factors usually are frequency and total volume of the accelerator, magnet design parameters are considered after cavity design has been completed. Considering the nature of coaxial cavity, simplest path for beams inside magnets is major segment of a circle, which was used as the reference for following discussions.



(a) (b)

Figure 3.9. Geometry and modeling, outside trajectory of a rhodotron (TT300) [12].

In rhodotron accelerators, magnets are used for not only guiding the electron beam to desired path, but also for synchronizing them with the RF field in the cavity. This syncronization, also called phase stability, is the fundamental constraint of magnet design. Time spent outside of the RF cavity should be precisely tuned to maximize acceleration in the succeeding pass.

Two different approaches have been used to achieve phase stability. These approaches are discussed further in the following sections.

3.2.1. $n\lambda$ Technique

First approach for phase stability considiration assumes that from the start of the current pass, $\beta_{avg} \approx 1$ for a synchronous particle.

To maintain the phase synchronization, a particle that started the current pass at $t = 0$, should start the next pass at $t = nT$, where n is an integer and T is the period of RF field inside cavity. As discussed earlier, by assuming the electron is traveling at c, phase stability can be achieved by designing the whole trajectory to be $n\lambda$, where λ is the wavelength of the RF field. In other words, total trajectory of synchronous particle needs to satisfy

$$L_{pass} = n\lambda. \tag{3.3}$$

Since $n > 1$ would require unnecessarily long and expensive beam guide and magnets, $n = 1$ is the most efficient choice, which leads to

$$L_{pass} = \lambda. \tag{3.4}$$

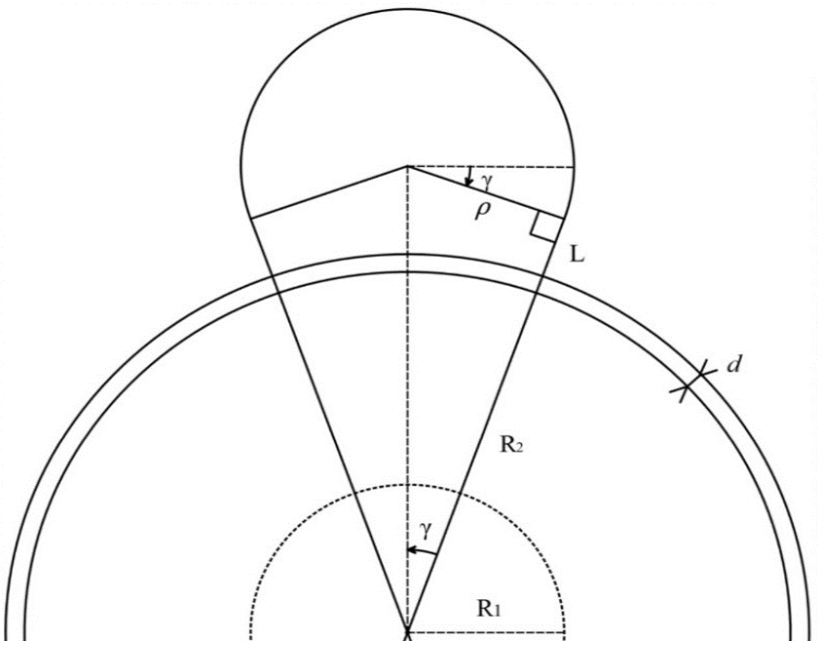


Figure 3.10. Trejectory of a particle in single pass.

From Figure 3.10, where γ is half the angle between trajectories of successive passes, ρ is the radius of the circular path inside magnet, L is the length of the magnet guide, d is the thickness of the RF cavity wall the constraints are

$$\begin{aligned} L_{in} &= 2R_2, && (3.5)\\ L_{out} &= 2L + 2d + (\pi + 2\gamma)\rho, && (3.6)\\ L_{pass} &= L_{in} + L_{out}, && (3.7)\\ \lambda &= 2R_2 + 2d + 2L + (\pi + 2\gamma)\rho. && (3.8) \end{aligned}$$

After the cavity design is complete, only L, γ and ρ remain, which are the design parameters of magnets. Observing the Figure 3.10 and using Equation 3.8,

$$\begin{aligned} \tan(\gamma) &= \frac{\rho}{R_2 + d + L}, && (3.9)\\ \rho &= \tan(\gamma)(R_2 + d + L), && (3.10)\\ 2\rho &= \tan(\gamma)(\lambda - (\pi + 2\gamma)\rho), && (3.11)\\ \rho(2 + \tan(\gamma)(\pi + 2\gamma)) &= \tan(\gamma)\lambda, && (3.12)\\ \rho &= \frac{\lambda}{\pi + 2\gamma + \frac{2}{\tan(\gamma)}}, && (3.13)\\ L + d &= \frac{\rho}{\tan(\gamma)} - R_2. && (3.14) \end{aligned}$$

Equations 3.13 and 3.14 define 2 constraints. As already discussed, it has been assumed that cavity design is set (*i.e* R_2, λ, d are already defined). Therefore only the magnet design parameters ρ, γ and L are left. Together with the constraints, one free variable defines the whole magnet design. Considering the importance of γ for maximum number of possible passes, it will be used as the free magnet design variable in the further discussions.

3.2.2. L_{out} Parameter Sweep

As mentioned previously in Section 3.2.1, Equation 3.8 assumes that the particles are fast enough so that $\Delta t_{pass} \approx nT$. This assumption is not guaranteed however, especially in the first few passes if the particles are not accelerated enough. This scenario can happen when RF power is not sufficiently high.

Consider an RF supply with $f = 1$ GHz, $P = 50$ kW. Using Equation 1.37, after the first pass the synchronous electron that entered the accelerator with 40 keV and $\phi = 15^\circ$ phase lag, it can be observed that

$$W_{gain} = 2.14 \times 0.2998^{1/4} \times 50000^{1/2} = 354keV, \tag{3.15}$$

$$W_{total} = 394keV, \tag{3.16}$$

$$\beta_{40keV} \approx 0.374, \tag{3.17}$$

$$\beta_{394keV} \approx 0.825. \tag{3.18}$$

It is clear that β_{avg} is not fast enough to sustain phase stability if the magnet is designed for $\beta = 1$. For these cases, Equation 3.8 fails to deliver phase stability. Another approach for designing a magnet would be to find a new constraint, K for total path length.

$$L_{pass} = K, \tag{3.19}$$

$$L_{in} + L_{out} = K. \tag{3.20}$$

Since L_{in} is set previously from cavity design, we can remove it from our equation and continue with

$$L_{out} = K. \tag{3.21}$$

To start, an optimization criteria for the following pass such as

- Ensure the phase stability of the synchronous electron
- Maximize the energy gain of the synchronous electron
- Minimize the energy spread of the beam
- Minimize the phase lag spread of the beam
- All of the above with decided weights

needs be selected. Using a simulation software such as CST Studio Particle Module, the optimum value of K for the criteria can be found by simulating for different values of K (sweeping) and analyzing the results. This process can be repeated for each magnet until the desired beam characteristics in the end of the accelerator are achieved.

However, one caviat of this technique is that, available simulation softwares are not well suited for this kind of process. As mentioned above, CST Studio Particle Module has a parameter sweep functionality. But the calculation times are too slow to be useful in this particular problem. A custom built software offering magnet optimizing sweep functionality will be discussed in the following chapter.

3.3. Initial Design at KAHVELab

A rhodotron with operation frequency of 107.5 MHz was decided to be built at KAHVELab. Frequency was selected to benefit from the RF power supply units at hand. After considering the earlier simulations mentioned in Sections 3.1 and 3.2, an initial design was created.

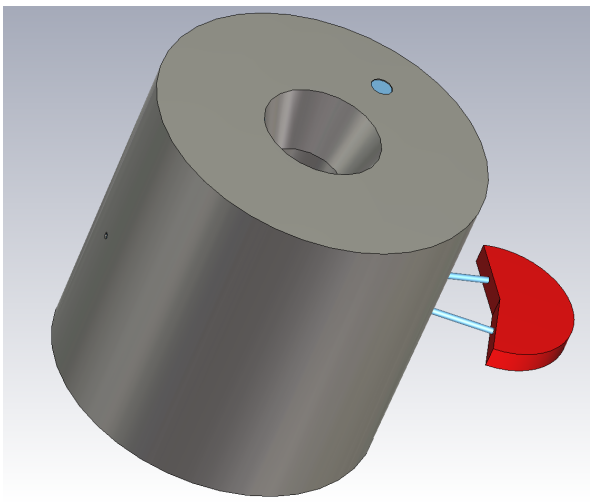
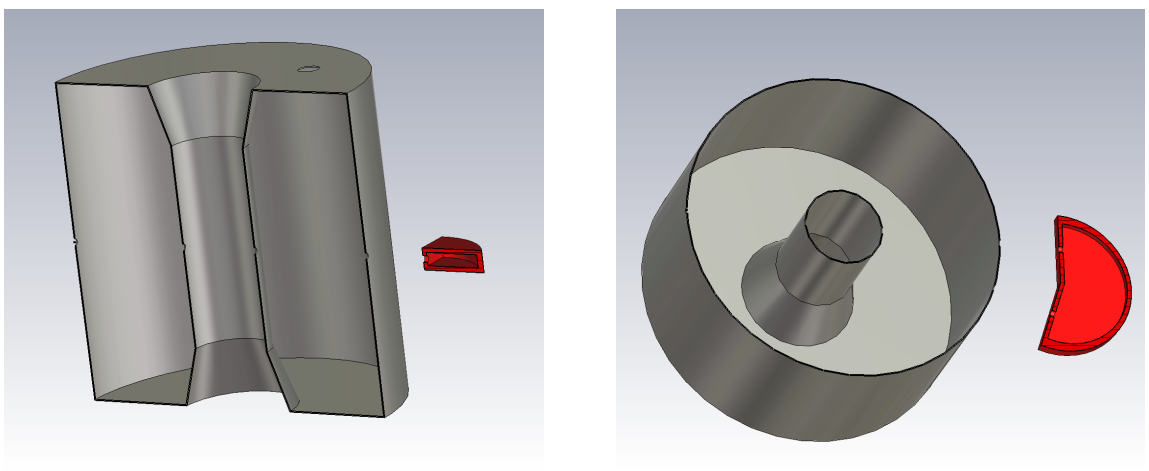


Figure 3.11. Initial design of the proposed rhodotron, $\gamma = 9°$.



(a) Axial cross section

(b) Cross section at acceleration plane

Figure 3.12. Initial design of the proposed rhodotron.

The magnets used in this design is the simplest in terms of design parameters. It consists of an iron casing encapsulating the shape of desired magnetic field, which are created by two coils inside.

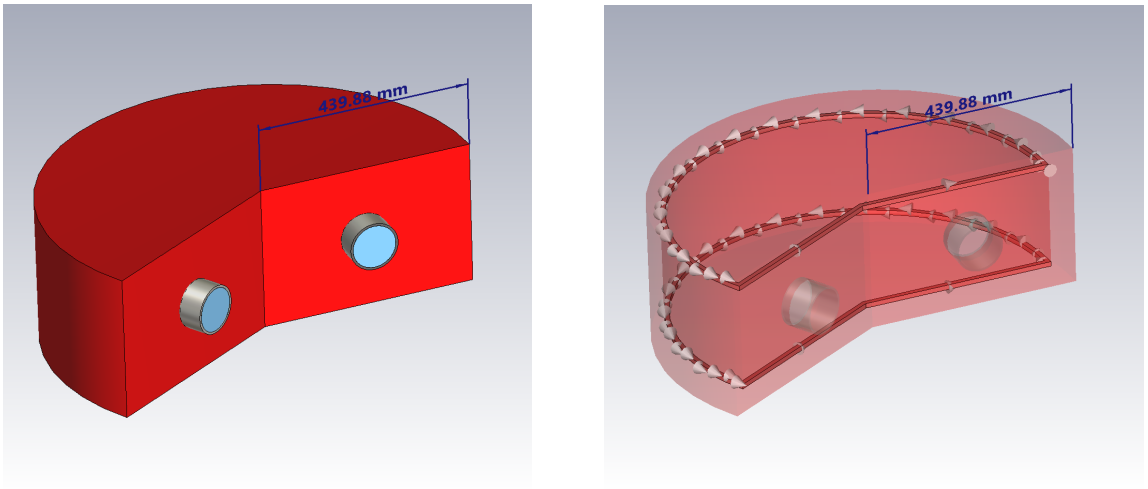


Figure 3.13. Initial magnet design.

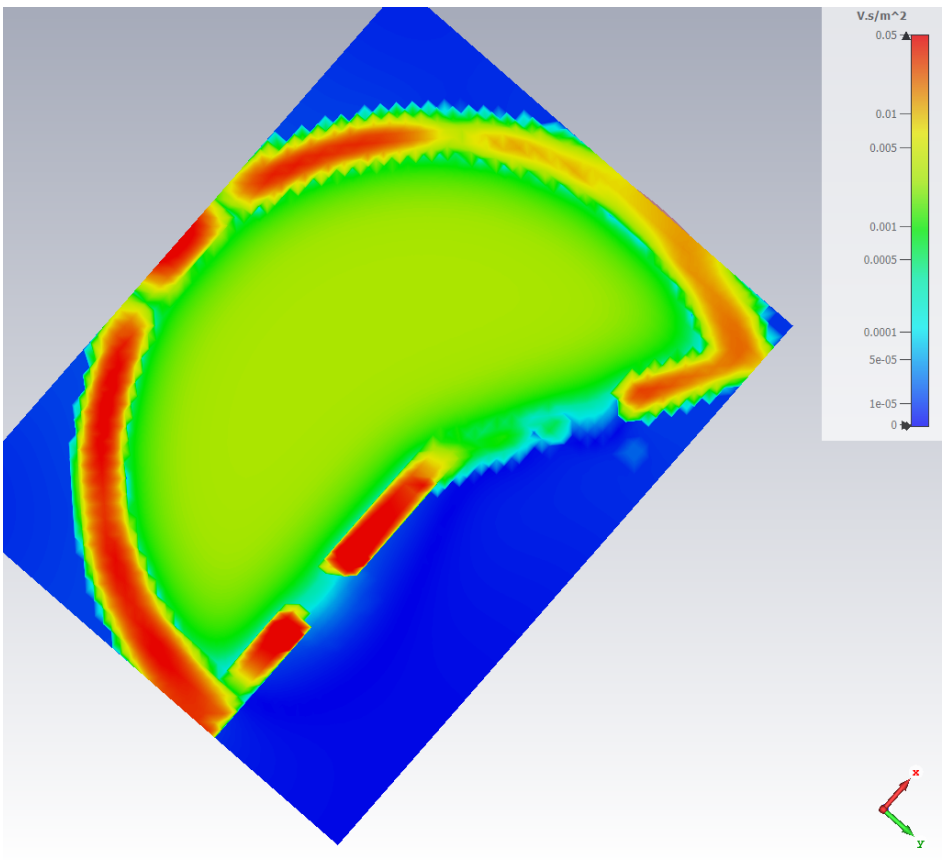


Figure 3.14. Magnetic field in acceleration plate of initial magnet design.

Figure 3.14 shows that this design provides relatively uniform magnetic field inside, although magnetic field gradient in the openings can be improved. After adding two more magnets and using $\gamma = 15°$, the second prototype design was created (Figure 3.15).

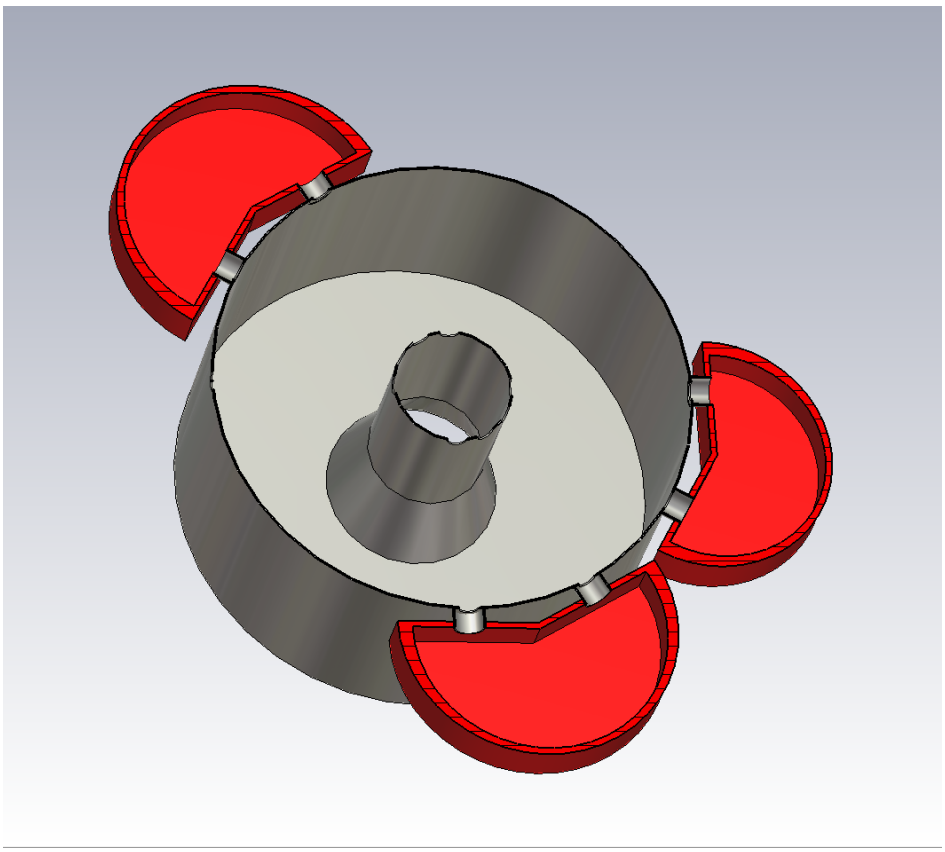
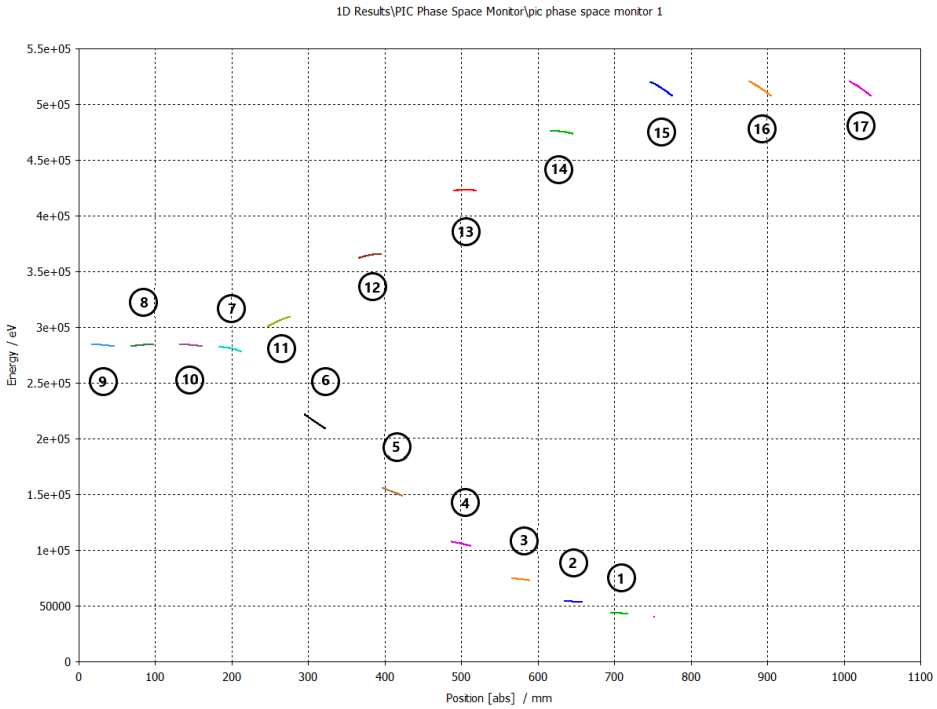
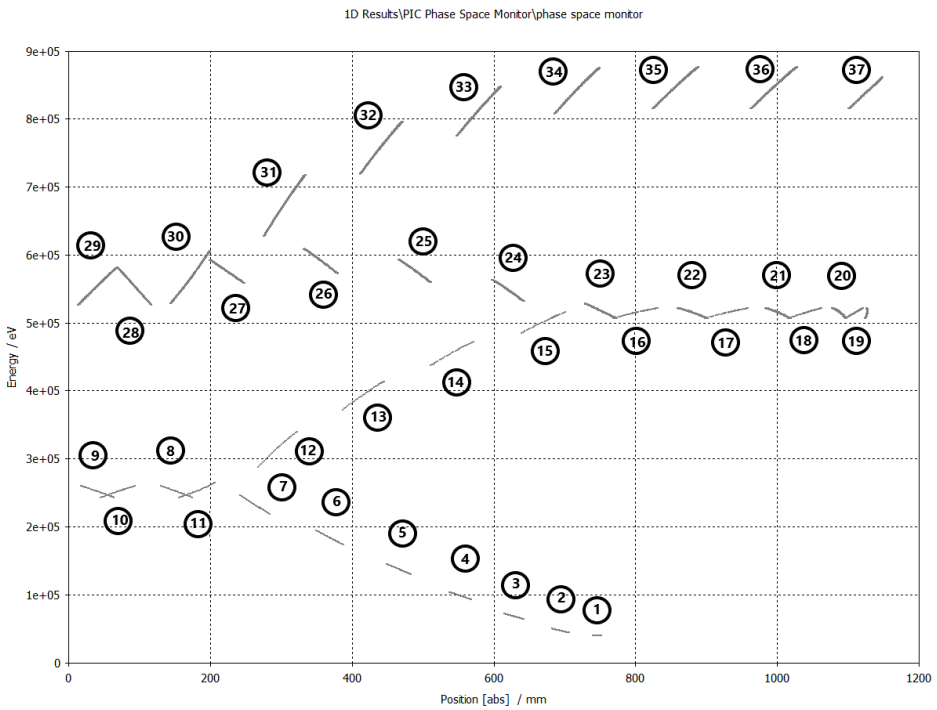


Figure 3.15. Initial cavity with three magnets as shown in Figure 3.13, $\gamma = 15°$.

These designs, together with 40 keV e^- gun injecting with phase lag of 15° for 1 ns, was simulated at RF power of 12 kW using CST Studio Particle Module PIC Solver. Results of these simulations can be found in Figure 3.16.



(a) One pass simulation of Figure 3.11



(b) Two pass simulation of Figure 3.15

Figure 3.16. $|r|$ vs *Energy* simulation of initial design.

The behavior of the beam in the Figure 3.16 after the first pass indicates that phase stability is not maintained. When the simulation results are examined further, it was observed that the beam started the second pass at $t \approx 12$ ns after the injection. Considering the starting phase lag $\phi_{lag} = 15°$, this would mean phase lag of the second pass was $\phi_{lag2} = 120°$. This result can be observed in Figure 3.16, as the beam decelerates after a short acceleration in the beginning of the second pass.

After various similar observations in CST similations, underlying cause of this problem was thought to be the $L_{pass} = n\lambda$ approach on magnet design failing in low energies, as anticipated and discussed earlier in the Section 3.2.2. The necessity for an alternative tool, capable of aiding researchers in improving beam behavior, was deemed evident.

Several improvements were made to KAHVELab Rhodotron initial designs [13]. These include curved edges on cavity which reduce the power loss on cavity walls, and redesigned iron casing on magnets for ease of production and maintenance. Improved designs can be seen in the figures below.

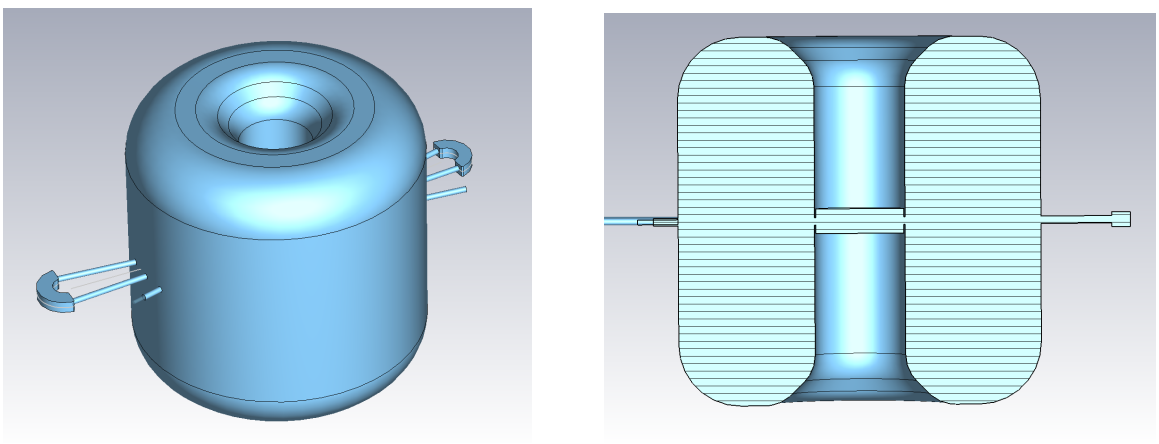


Figure 3.17. Improved cavity design.

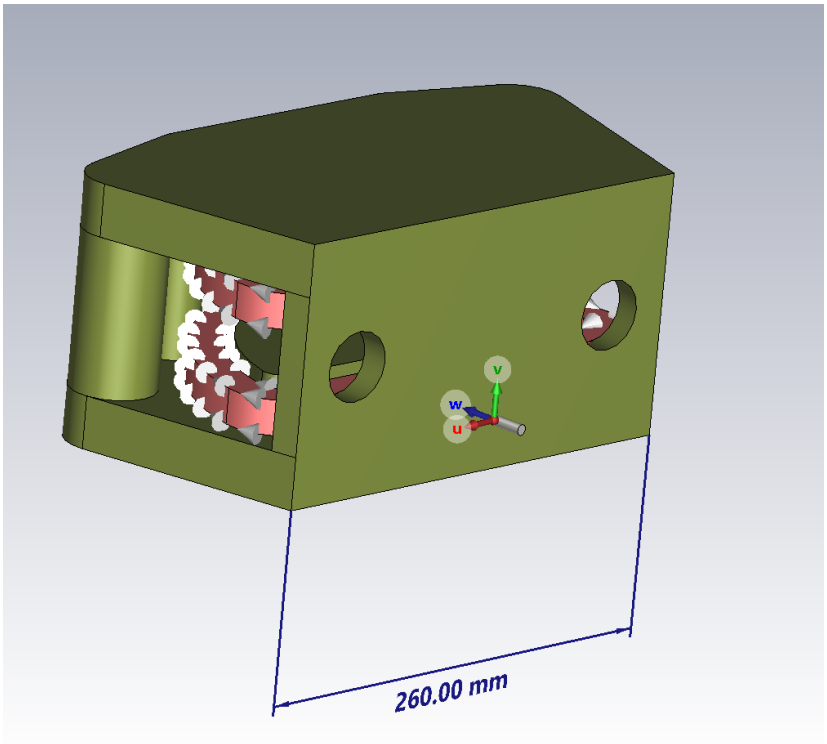


Figure 3.18. Improved magnet design.

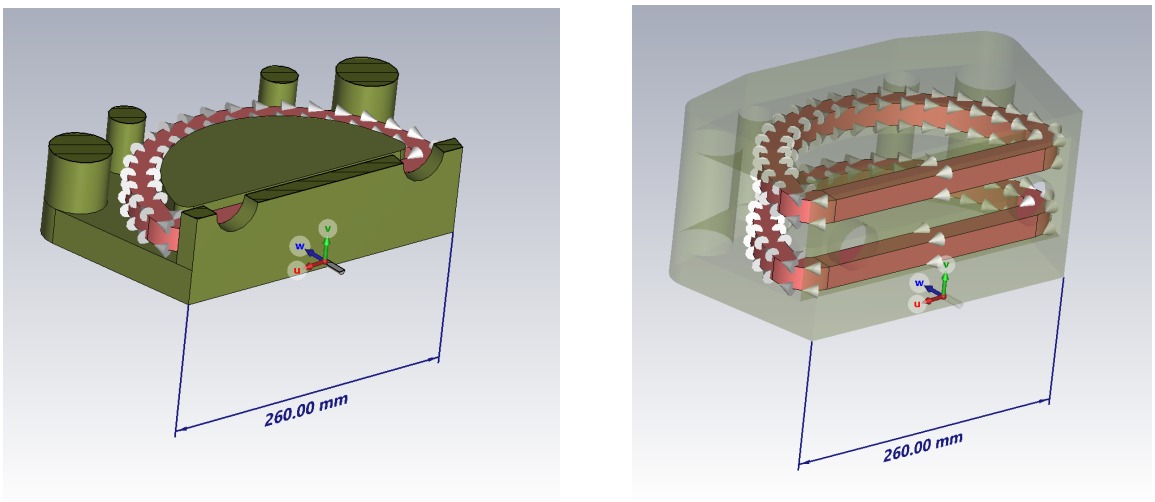


Figure 3.19. Cross section and coils of improved magnet design.

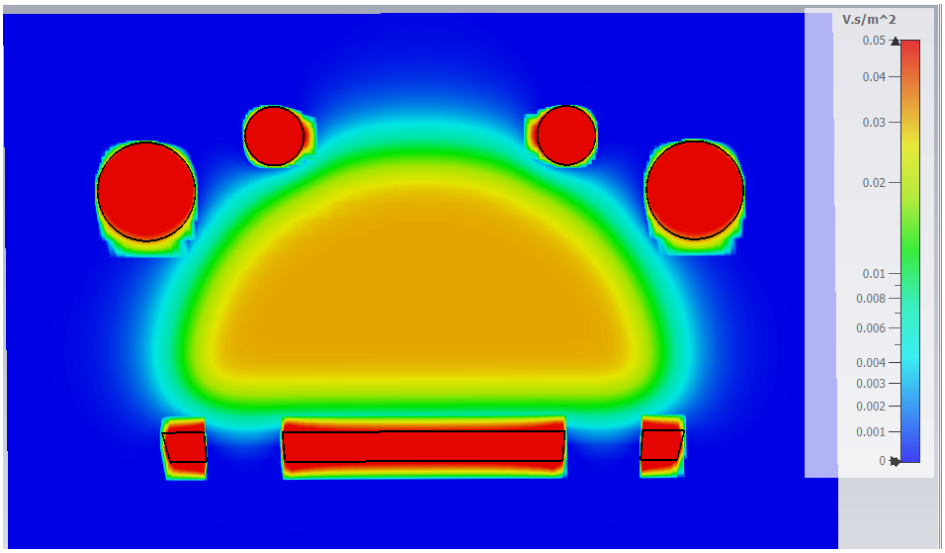


Figure 3.20. Magnetic field in acceleration plate of improved magnet design.

From Figures 3.14 and 3.20, it can be observed that improved magnet design reduces magnetic field leak in enter and exit openings considerably. This reduction helps containing the spread of entering beam, decreasing the amount of focusing needed further into trajectory.

4. SIMULATION

After considering the lack of design and simulation software available for *Rhodotron-type* accelerators, together with the energy gain disagreement between *CST Studio* and the calculations of *J. Pottier* in Equation 1.36, the need for a custom built tool became apparent. A simple 1D proof of concept was quickly implemented for this purpose.

4.1. Proof of Concept

The proof of concept (POC) worked with interacting an electron with the EM field provided, at discrete time steps dt. The core sequence of this software is provided below and the implemenation of this logic can be observed in Figure 4.1. Where $\vec{\mathbf{r}}$ is the position, $\vec{\mathbf{v}}$ is the velocity of the e^-:

- Get electron energy E_{in},
- Get the RF field definitions,
- Get the magnet design parameters if there is any,
- Do following until the simulation time has ellapsed,
  - (i) Calculate $\vec{\mathbf{E}}_{||}(\vec{\mathbf{r}})$,
  - (ii) Calculate $\vec{\mathbf{a}}_{E||}$ using Equation 1.16,
  - (iii) Calculate $\vec{\mathbf{r}}(t+dt)$ using Equation 2.3,
  - (iv) Calculate $\vec{\mathbf{v}}(t+dt)$ using Equation 2.4,
  - (v) Calculate new E_t.

```
for (double t=0; t<SimuTime; t+=dT){
    double RelBeta  = v/c;
    double RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);

    double ef=Eradial(r_pos*1000,t,0); // convert position to mm
    double acc=ef*1E6*eQMratio/(RelGamma*RelGamma*RelGamma);

    r_pos = r_pos + v * dT*ns + 1/2*acc*(dT*ns)*(dT*ns);
    v = v + acc*dT*ns;

    RelBeta  = v/c;
    RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);
    Et=RelGamma*E0;
}
```

Figure 4.1. Core logic loop of the POC.

After simulating the synchronous electron mentioned in Section 1.2.4.2 with $\phi_{lag} = 15°$, $P_{diss} = 40$kW, $R_1 = 0.188$m, $R_2 = 0.753$m, $f = 107.5$MHz for one pass, the results from POC, CST and Equation 1.36 were found to be

$$E_{Pottier} = 0.593MeV, \tag{4.1}$$

$$E_{CST} = 0.521MeV, \tag{4.2}$$

$$E_{POC} = 0.532MeV. \tag{4.3}$$

As can be observed in Equation 4.2, POC software and CST produced similar results; $\Delta E_{(POC-CST)}/E_{CST} \approx 2\%$. while Equation 1.36 was noticably different; $\Delta E_{(Pottier-CST)}/E_{CST} \approx 14\%$. This inconsistency between Equation 1.36 and CST simulation was noticed in the earlier simulations as well, which supported the idea of using another simulation software to reduce reliance on CST. Since using a single software would have made the process error prone.

After the results obtained from POC were found to be promising, a decision was made to develop a more robust simulation software. This software was named Rhodotron Simulation for the current stage and started development in October, 2021. The development and improvement efforts are still ongoing.

During the development of Rhodotron Simulation software, several intermediate builds have been implemented and tested. The following sections in this chapter investigates their implementations further.

4.2. Intermediate Versions

4.2.1. L_{out} Optimization For Single $\mathbf{e}^-$

Initial step of improving POC towards Rhodotron Simulation was to implement L_{out} optimizations to help optimazing magnet designs, as discussed in Section 3.2.2. First approach was to hang the $\mathbf{e}^-$ outside of the cavity for $t_{out} = L_{out}/v$, then inject it back to the cavity with reversed $\vec{\mathbf{v}}$. One can sweep the t_{out} parameter to find the optimal value. This simple implementation can be found in Figure A.1 of Appendix A.

Although the results from this optimization sweep were promising after they were simulated with CST, simulating one particle would not be sufficiently useful for designing a magnet.

4.2.2. ϕ_{lag} Optimization for Bunches

After successfully accelerating single $\mathbf{e}^-$, particle bunches were implemented to approximate a real $\mathbf{e}^-$ gun. They were modeled as N electrons fired from an $\mathbf{e}^-$ gun at even time intervals. This approach was taken because the amount of time gun fires, defined as Gun Active Time, t_g, is a crucial part of pulsed $\mathbf{e}^-$ gun design.

Addition of bunches would immediately proven useful when finding optimal phase lag for bunch entry. ϕ_{lag} for a bunch was defined as the RF phase when the first $\mathbf{e}^-$ of the bunch has entered the cavity and it defines the starting time of the current pass. To use the parameter sweep method, as used in L_{out} optimization, relevant bunch characteristics are defined as

- μE: Average energy,
- E_{rms}: Root mean square of energy,
- R_{rms}: Root mean square of $\mathbf{e}^-$ positions.

Optimal ϕ_{lag} would produce maximum μE while minimizing E_{rms} & R_{rms}. For the first pass, E_{rms} and R_{rms} would be vaguely dependent of each other; therefore, initial implementation of ϕ_{lag} sweep was based on minimizing E_{rms} during simulation (see Figure 4.2). For μE considerations, data from the software would be analyzed either manually or by using external tools such as ROOT framework.

```
1  int phase_opt(int phase_sweep_range){
2      double minrms = 1;
3      int opt_phase;
4      for(int RFphase = -phase_sweep_range; RFphase <= phase_sweep_range; RFphase++){
5        Bunch bunch1(RFphase);
6        double t1 = 0;
7        bunch1.bunch_gecis_t(t1);
8        bunch1.reset_pos();
9
10       if( bunch1.E_rms() < minrms ){
11         minrms = bunch1.E_rms();
12         opt_phase = RFphase;
13       }
14     }
15     return opt_phase;
16 }
```

Figure 4.2. ϕ_{lag} Optimization For Initial Bunch Design.

Since ϕ_{lag} is relatively easy to change after production, another version of Figure 4.2 that was modified for given magnet design parameters was also implemented (see Figure A.2 in Appendix A). This version can be useful for optimizing ϕ_{lag} in case of production issues in magnets.

After the bunch and ϕ_{lag} sweep implementations, L_{out} sweep was also updated to minimize E_{rms}. ρ and L calculations, using Equations 3.13 and 3.14, were also integrated. Two example runs from this intermediate build can be found in Figures B.1 and B.2 of Appendix B.

4.2.3. Simulation in 3D

After successfully implementing $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ interaction in 1D and confirming the usefulness of this tool, the decision was made to proceed with implementing a 3D version of the Rhodotron Simulation. Although complete refactoring of the software was necessary, this upgrade was crucial for implementation of $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction. The refactoring effort included proper implementation of OOP, details of which can be seen in Figures A.13, A.14, A.15, A.16, A.17 and A.18 of Appendix A.

Magnets were modeled as major segments of a circle, defined with $\vec{\mathbf{r}}_{mag}$, $\mathbf{R}$ and $|\mathbf{B}|$. For the initial implementation, $\vec{\mathbf{B}}$ assumed to be uniform and has no leaks outside the magnet boundary (Figure A.5 in Appendix A).

Implementations for $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ and $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interactions in 3D can be found in Figures 4.3 and 4.4, where * and % are, dot-product and cross-product respectably (Figure A.3 of Appendix A).

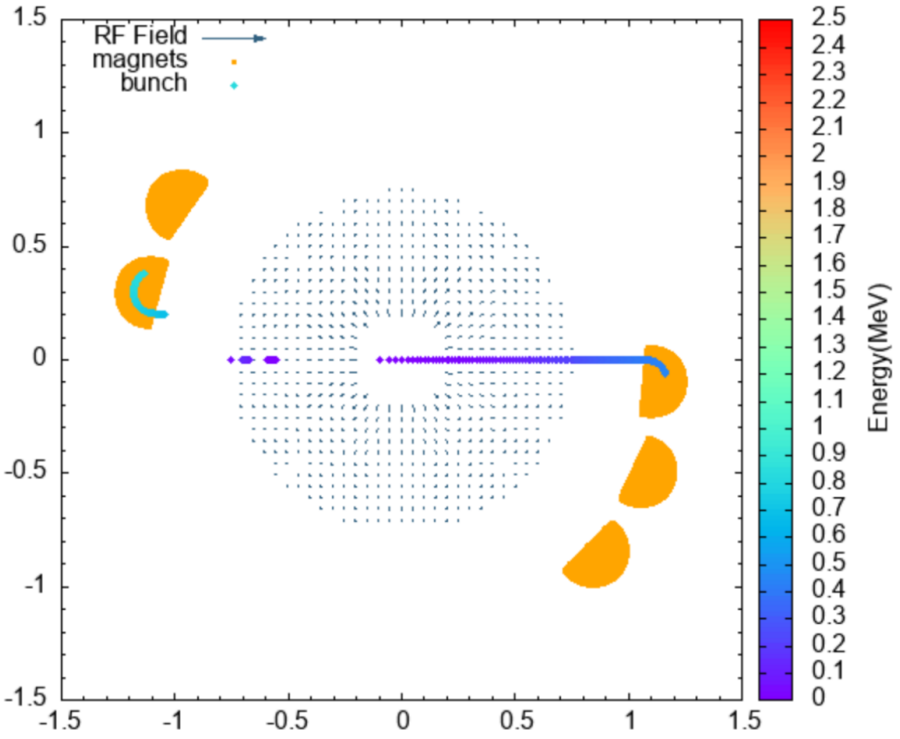
```
vector3d CoaxialRFField::actOn(Electron& e){
    vector3d Efield = getField(e.pos);                          // Calculate E vector
    vector3d F_m = Efield*1E6*eQMratio;                         // Calculate F/m vector
    vector3d acc = (F_m - e.vel*(e.vel*F_m)/(c*c))/e.gamma(); // Calculate a vector
    return acc;
}
```

Figure 4.3. $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ interaction logic from Equation 1.18.

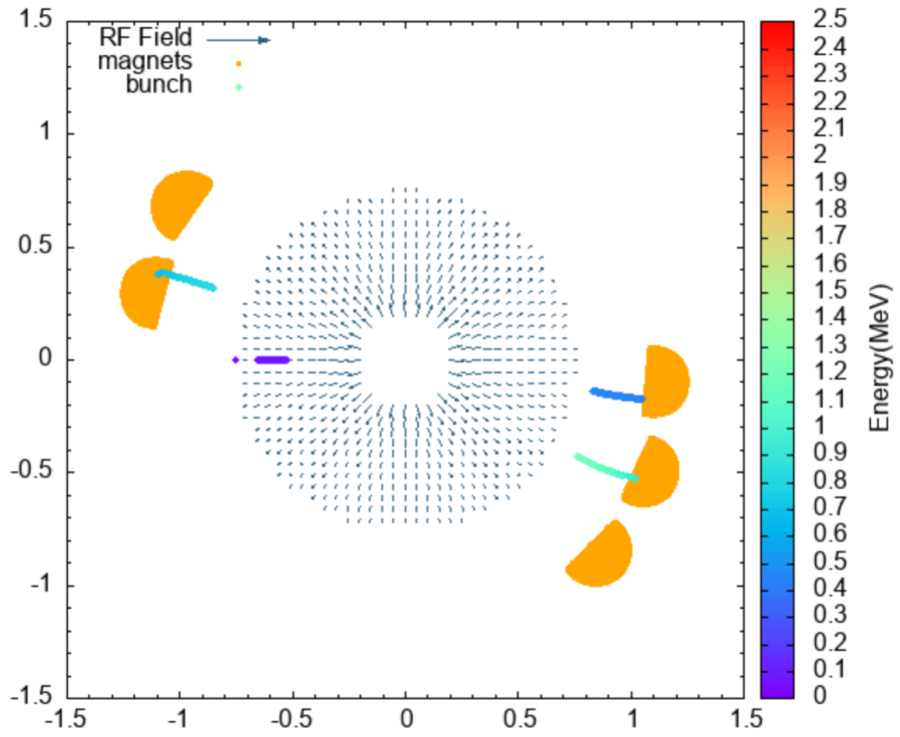
```
vector3d MagneticField::actOn(Electron& e){
    if (isInside(e.pos) == -1)
        return vector3d(0,0,0);
    vector3d Bfield = getField(e.pos);                          // Calculate B vector
    vector3d F_m = (e.vel % Bfield)*eQMratio;                   // Calculate F/m vector
    vector3d acc = (F_m)/e.gamma();                             // Calculate a vector
    return acc;
}
```

Figure 4.4. $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction logic from Equation 1.18.

Simulating in 3D had one other benefit; it was now possible to visualize the results by rendering the interaction data. For this purpose, gnuplot was integrated into Rhodotron Simulation to produce 2D visualization of acceleration plane. Rendered results could be stored as gif animations. Two of such renders can be seen in Figure 4.5.



(a) Unsyncronized $\mathbf{e}^-$ gun period, $T_g = 5ns$



(b) Syncronized $\mathbf{e}^-$ gun period, $T_g = 9.3ns$

Figure 4.5. Example gnuplot renders of Rhodotron Simulation, $P = 40\text{kW}$, $f = 107.5\text{MHz}$.

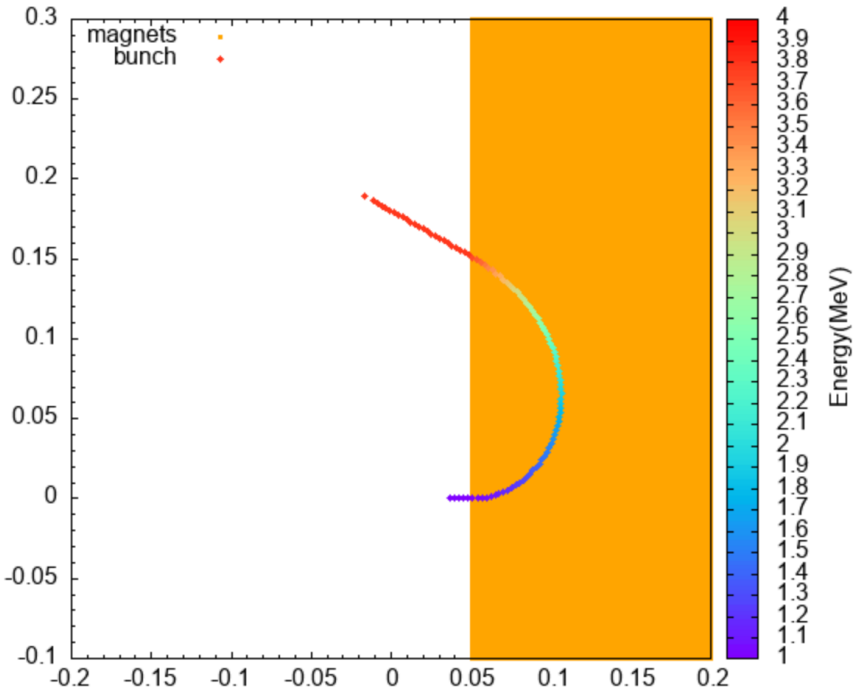
4.2.4. Acceleration in Magnetic Field

An issue regarding the $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction became apparent when energy gain during these interactions was observed. A setup simulation was implemented in which a bunch of 100$\mathbf{e}^-$ at 1MeV was fired into a uniform magnetic field of 0.1T located in $x > 0.05$m. Initial results at $dt = 0.01$ns proved the suspicion of $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction being broken. However, the energy gain would decrease tremendously as dt decreased. Decreasing dt would be the best way to increase accuracy of the results; however, this is not sustainable due to computing power limitations. Until this point, Rhodotron Simulation have been using Section 2.3.1 for $\mathbf{e}^-$ - **EM** interactions. To test newer approaches, two additional version of $\mathbf{e}^-$ - **EM** interaction that are using Section 2.3.2 were added.

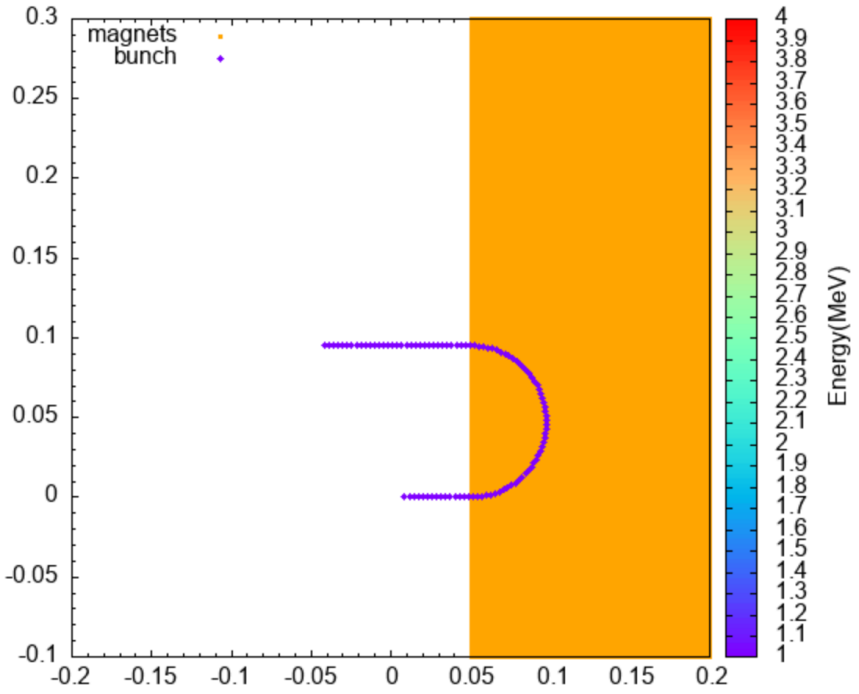
4.2.4.1. RK4-1. First approach for integrating Section 2.3.2 into $\mathbf{e}^-$ - **EM** interaction was to calculate $\vec{\mathbf{a}}_E$ and $\vec{\mathbf{a}}_B$ from Equation 1.18 using RK4. After $\vec{\mathbf{a}}_{EM} = \vec{\mathbf{a}}_E + \vec{\mathbf{a}}_B$ was calculated, $\mathbf{e}^-$ would move and accelerate using the Leap-frog method. The idea was to produce more refined interaction results, leading to improved accuracy especially in $\mathbf{e}^-$ - $\vec{\mathbf{B}}$. RF field was kept static during the RK4 computation, due to ongoing multithreading implementation efforts. The implementation can be found in Figures A.4 and A.6.

4.2.4.2. RK4-2. Following the implementation of RK4-1, revisions were made to the integration method for RK4 to replace Leap-frog. Instead of calculating $\vec{\mathbf{a}}_{EM}$ using RK4, $\vec{\mathbf{r}}$ and $\vec{\mathbf{v}}$ would be determined directly.

These three methods were then tested in the same setup as Figure 4.6. The results can be found in Figure C.1 of Appendix C. RK4-1 was decided to be abandoned as it produced the same accuracy in twice the simulation time of RK4-2.



(a) $dt = 10^{-2}$ns, $\Delta E = 2.783$MeV



(b) $dt = 10^{-4}$ns, $\Delta E = 0.011$MeV

Figure 4.6. Energy gain of 1MeV bunch in **B**=0.1T.

More rigorous testing was done with Leap-frog and RK4-2 however. Still using the setup in Figure 4.6, each dt configuration was simulated 10 times, calculating average and standard deviations afterwards.

Example renders from these tests can be observed in Figure 4.7 and the data from these tests can be found in Tables C.1 and C.2 of Appendix C. To investigate the data further, one can define a performance measurement, F as

$$F \propto 1/T, \tag{4.4}$$

$$F \propto 1/\Delta E. \tag{4.5}$$

When $dt = 10^{-5}$ ns was taken as reference point due to providing a good balance of accuracy and computational intensity, relative performance can be calculated as

$$\Delta E_{LF} = 110 \times 10^{-5} MeV, \tag{4.6}$$

$$\Delta E^1_{RK} = 64 \times 10^{-5} MeV, \tag{4.7}$$

$$T^1_{LF} = 9.44 \pm 0.03s, \tag{4.8}$$

$$T^1_{RK} = 21.33 \pm 0.02s, \tag{4.9}$$

$$\Delta E_{LF} \times T_{LF} = 104 \times 10^{-4} \pm 10^{-4} MeVs, \tag{4.10}$$

$$\Delta E_{RK} \times T_{RK} = 137 \times 10^{-4} \pm 2 \times 10^{-4} MeVs, \tag{4.11}$$

$$F_{LF}/F_{RK} = \frac{\Delta E_{RK} \times T_{RK}}{\Delta E_{LF} \times T^1_{LF}} = 1.32 \pm 0.01. \tag{4.12}$$

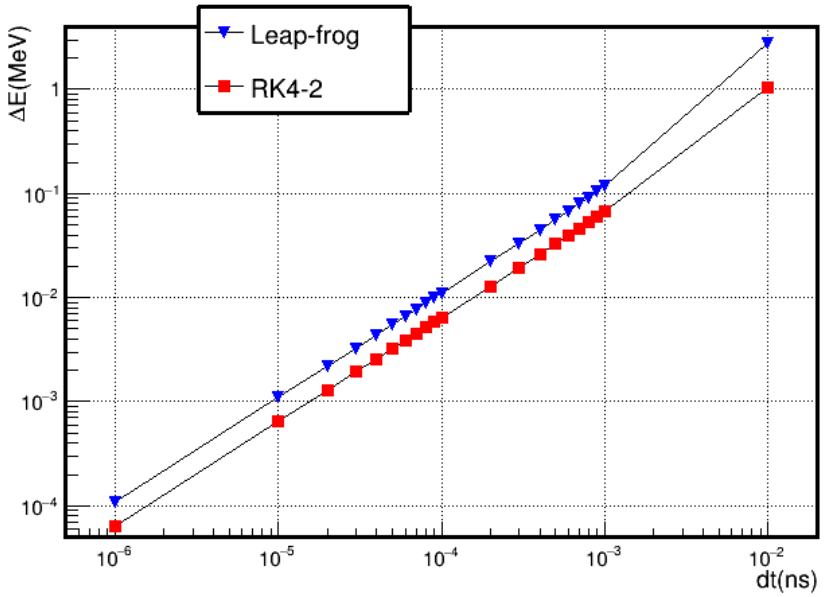
Also, another relative performance can be obsered from the data as

$$\Delta E_{LF}(dt = 3 \times 10^{-5}) \approx \Delta E_{RK}(dt = 5 \times 10^{-5}) \approx 32.5 \times 10^{-4} MeV, \tag{4.13}$$

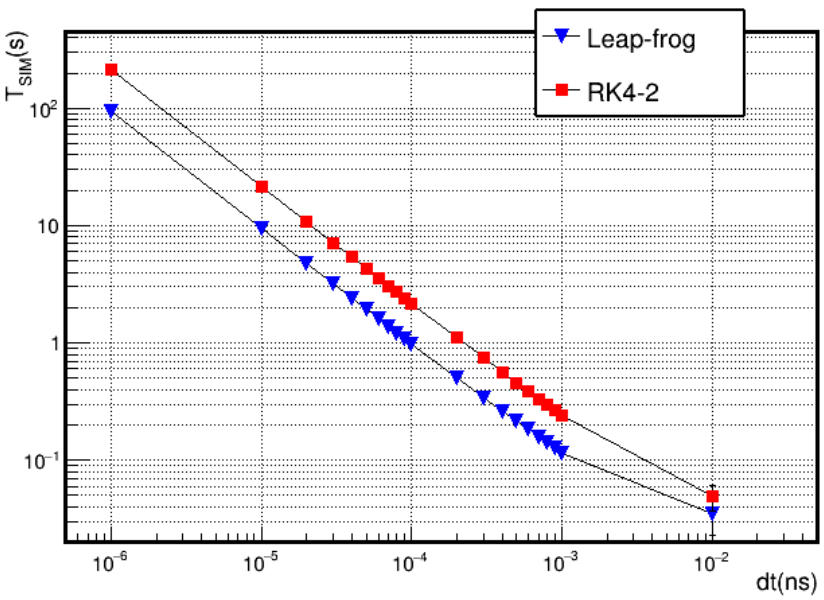
$$T_{LF}(dt = 3 \times 10^{-5}) = 3.18 \pm 0.02s, \tag{4.14}$$

$$T_{RK}(dt = 5 \times 10^{-5}) = 4.30 \pm 0.01s, \tag{4.15}$$

$$F_{LF}/F_{RK} = \frac{T_{RK}(dt = 5 \times 10^{-5})}{T_{LF}(dt = 3 \times 10^{-5})} = 1.4 \pm 0.1. \tag{4.16}$$



(a) dt vs ΔE



(b) dt vs T_{sim}

Figure 4.7. Comparing Leap-frog, RK4-2 performance on $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction $E_{in} = 1\text{MeV}$, $\mathbf{B}$=0.1T, $t_{end} = 5\text{ns}$.

Uncertainty of Equation 4.13 was taken high due to the approximation. After combining Equations 4.6 and 4.13, F can be calculated as

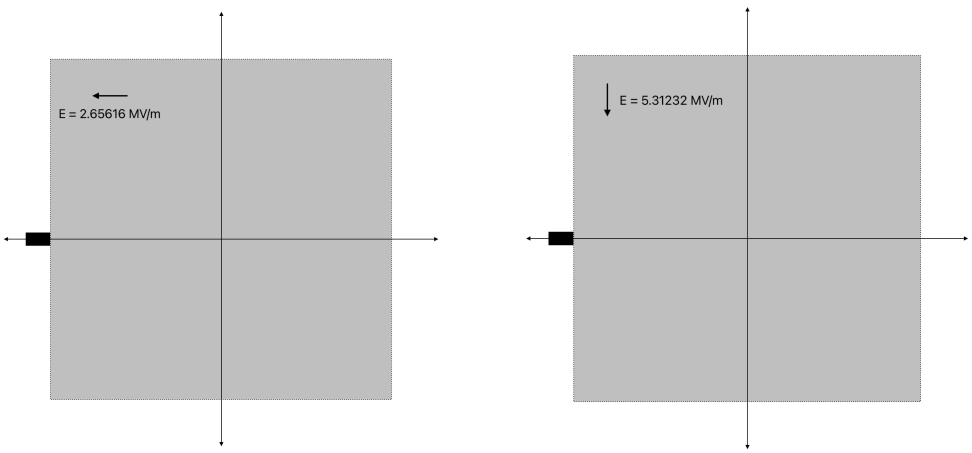
$$F_{LF}/F_{RK} = 1.36 \pm 0.05. \tag{4.17}$$

Therefore, Leap-frog was found to be the better choice as it provided with 1.36 ± 0.05 times accuracy in $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interactions at a given time with respect to RK4. However, RK4 was promising in situations where decreasing the stepsize, dt, is not viable. Therefore both were integrated into Rhodotron Simulation for the user to decide. RK4-2 renders from these test can be found in Figure B.3 of Appendix B.

4.2.5. Acceleration in Electric Field

After the accuracy concerns regarding the $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction were raised, it was decided to test $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ and compare the performance of Leap-frog and RK4.

As test setups, two simulation confgurations were made. They aimed to test the accuracy of acceleration of a beam in parallel and perpendicular static uniform electric fields. Both configurations had an $\mathbf{e}^-$-gun located at $(-0.753, 0, 0)$m, directed at $(1, 0, 0)$ firing electrons with the kinetic energy of 1MeV.



(a) Setup-1 parallel $\vec{\mathbf{E}}$ field (b) Setup-2, perpendicular $\vec{\mathbf{E}}$ field

Figure 4.8. Illustration of test setups.

In the first test, the beam would be injected into a static uniform electric field $\vec{\mathbf{E}} = (-2.65616, 0, 0)$ MV/m where $-0.753 < x < 0.753$ and $-0.753 < y < 0.753$, $\vec{\mathbf{E}} = 0$ elsewhere. Considering the $\vec{\mathbf{E}}$ is parallel to the beam path, beam energy at exit can be calculated as

$$\begin{aligned} \Delta V^1 &= -\int \vec{\mathbf{E}} \cdot \vec{\mathbf{ds}} \\ &= -\int_{-0.753}^{0.753} -2.65616 \times dx \\ &= 2.65616 \times 1.506 \\ &= 4MV, \end{aligned} \tag{4.18}$$

$$\Delta E^1 = 4MeV, \tag{4.19}$$

$$E^1_{exitTH} = 5MeV. \tag{4.20}$$

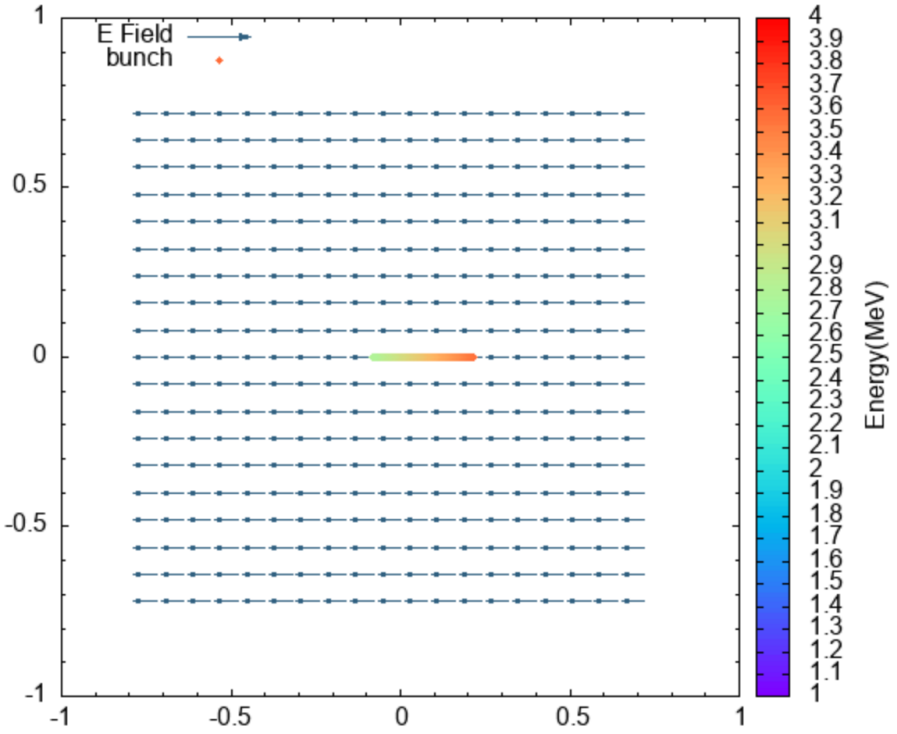
In the second test on the other hand, the beam would be injected into a different static uniform electric field, $\vec{\mathbf{E}} = (0, -5.31232, 0)$ MV/m where $-0.753 < x < 0.753$ and $-0.753 < y < 0.753$, $\vec{\mathbf{E}} = 0$ elsewhere. Beam energy at exit can be calculated as

$$\begin{aligned} \Delta V^2 &= -\int \vec{\mathbf{E}} \cdot \vec{\mathbf{ds}} \\ &= -\int_{0}^{0.753} -5.31232 \times dy \\ &= 5.31232 \times 0.753 \\ &= 4MV, \end{aligned} \tag{4.21}$$

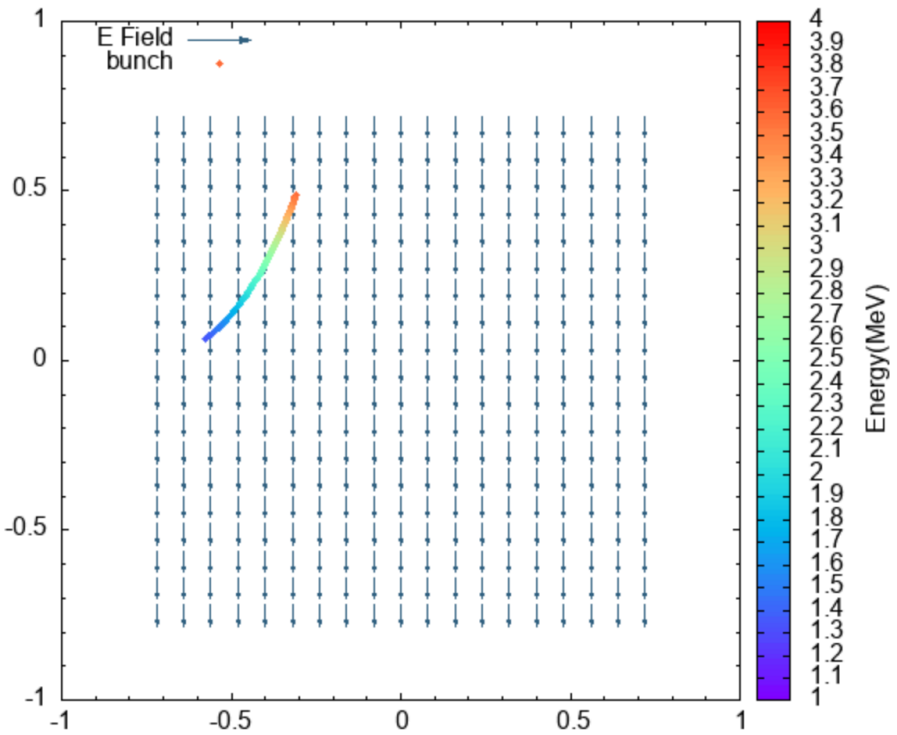
$$\Delta E^2 = 4MeV, \tag{4.22}$$

$$E^2_{exitTH} = 5MeV. \tag{4.23}$$

Therefore, in the both tests, the beam was expected to exit $\vec{\mathbf{E}}$ with $E_{exitTH} = 5$ MeV. To also measure the variance in simulation completion times, T_{sim}, set of 10 runs were completed at the configuration for each dt value. Results from these tests can be observed in Figure 4.10 and Figure C.2 of Appendix C.

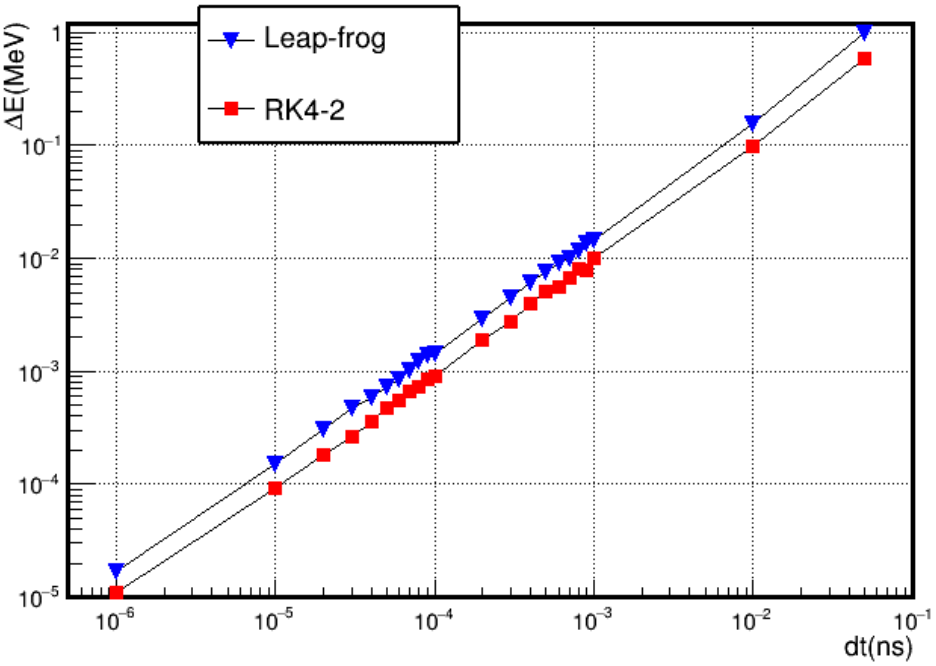


(a) Setup-1, $\mathbf{V}_{\parallel}$=4MV

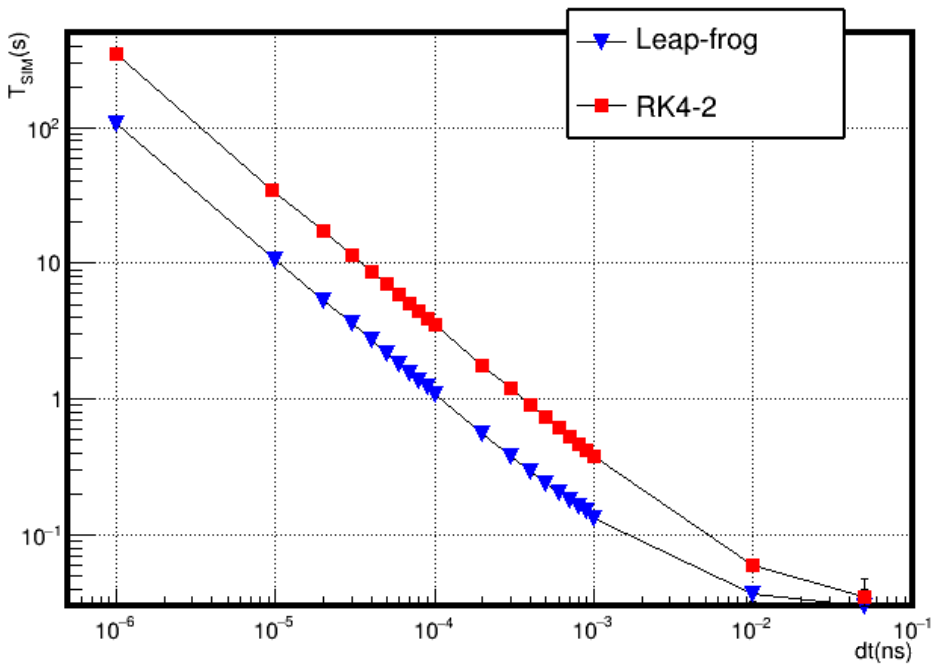


(b) Setup-2, $\mathbf{V}_{\perp}$=4MV

Figure 4.9. Render of the test setups. $E_{in} = 1$ MeV, $t_{end} = 6$ ns.



(a) dt vs ΔE



(b) dt vs T_{sim}

Figure 4.10. Comparing Leap-frog, RK4-2 performance on $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ interaction $E_{in} = 1\text{MeV}$, $\mathbf{V}_\perp$=4MV, $t_{end} = 6$ ns.

Taking the $dt = 10^{-5}$ ns for the reference point as before, using the data

$$\begin{aligned} \Delta E^1_{LF} = &\ 22 \times 10^{-6} MeV, & \Delta E^2_{LF} = &\ 15 \times 10^{-5} MeV, \\ \Delta E^1_{RK} = &\ 16 \times 10^{-6} MeV, & \Delta E^2_{RK} = &\ 9.3 \times 10^{-5} MeV, \\ T^1_{LF} = &\ 12.40 \pm 0.04 s, & T^2_{LF} = &\ 10.75 \pm 0.04 s, \\ T^1_{RK} = &\ 40.08 \pm 0.12 s, & T^2_{RK} = &\ 34.93 \pm 0.39 s, \end{aligned}$$

the relative performances can be calculated as

$$\Delta E^1_{LF} \times T^1_{LF} = 270 \times 10^{-6} \pm 10^{-6} MeVs, \tag{4.24}$$

$$\Delta E^1_{RK} \times T^1_{RK} = 640 \times 10^{-6} \pm 2 \times 10^{-6} MeVs, \tag{4.25}$$

$$F^1_{LF}/F^1_{RK} = \frac{\Delta E^1_{RK} \times T^1_{RK}}{\Delta E^1_{LF} \times T^1_{LF}} = 2.4 \pm 0.02, \tag{4.26}$$

and

$$\Delta E^2_{LF} \times T^2_{LF} = 160 \times 10^{-5} \pm 10^{-5} MeVs, \tag{4.27}$$

$$\Delta E^2_{RK} \times T^2_{RK} = 320 \times 10^{-5} 4 \pm 10^{-5} MeVs, \tag{4.28}$$

$$F^2_{LF}/F^2_{RK} = \frac{\Delta E^2_{RK} \times T^2_{RK}}{\Delta E^2_{LF} \times T^2_{LF}} = 2.0 \pm 0.03. \tag{4.29}$$

Leap-frog provides 2.4 times and 2.0 times less overacceleration per simulation time than RK4 in parallel and perpendicular electric fields respectably. This results can be tested further with obsering from the data (see Tables C.3, C.4, C.5 and C.6 of Appendix C) and calculating other relative performances as

$$\Delta E^1_{LF}(dt = 2 \times 10^{-5}) = \Delta E^1_{RK}(dt = 3 \times 10^{-5}) = 45 \times 10^{-6} MeV, \tag{4.30}$$

$$T^1_{LF}(dt = 2 \times 10^{-5}) = 6.23 \pm 0.02 s, \tag{4.31}$$

$$T^1_{RK}(dt = 3 \times 10^{-5}) = 13.40 \pm 0.03 s, \tag{4.32}$$

$$F^1_{LF}/F^1_{RK} = \frac{T^1_{RK}(dt = 3 \times 10^{-5})}{T^1_{LF}(dt = 2 \times 10^{-5})} = 2.15 \pm 0.02, \tag{4.33}$$

and

$$\Delta E^2_{LF}(dt = 3 \times 10^{-5}) \approx \Delta E^2_{RK}(dt = 5 \times 10^{-5}) \approx 470 \times 10^{-6} MeV, \quad (4.34)$$

$$T^2_{LF}(dt = 3 \times 10^{-5}) = 3.62, \quad (4.35)$$

$$T^2_{RK}(dt = 5 \times 10^{-5}) = 7.00, \quad (4.36)$$

$$F^2_{LF}/F^2_{RK} = \frac{T^2_{RK}(dt = 5 \times 10^{-5})}{T^2_{LF}(dt = 3 \times 10^{-5})} = 1.9 \pm 0.1. \quad (4.37)$$

The last uncertainty was taken high due to the approximation of $467 \approx 471$. After combining Equations 4.26, 4.29, 4.30 and 4.34, the performance of Leap-frog relative to RK4 in $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ interaction can be calculated as

$$F_{LF}/F_{RK} = 2.11 \pm 0.03. \quad (4.38)$$

Therefore, it can be concluded that Leap-frog outperforms RK4 with the relative performance of 2.11 ± 0.03 in $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ interactions with static uniform $\vec{\mathbf{E}}$ field.

4.2.6. Multithreading

As already mentioned before, multithreading implementation efforts were ongoing since right after the start of this project. There have been a number of different approaches for implementation. First implemenation was done when the Rhodotron Simulation was only capable of 1D simulations. After the sizable refactoring done for 3D capabilities, this implementation was obselete.

For the current version, a main thread that spawns and manages several other worker threads would be used as can be seen in Figure 4.11. The UI-Console thread would handle incoming and outgoing communication, notifying the user about the status of simulation (see Figure B.5 in Appendix B for example console notification, Figure A.12 in Appendix A for implemenation) or communucating with the GUI that will be disscussed in the next section.

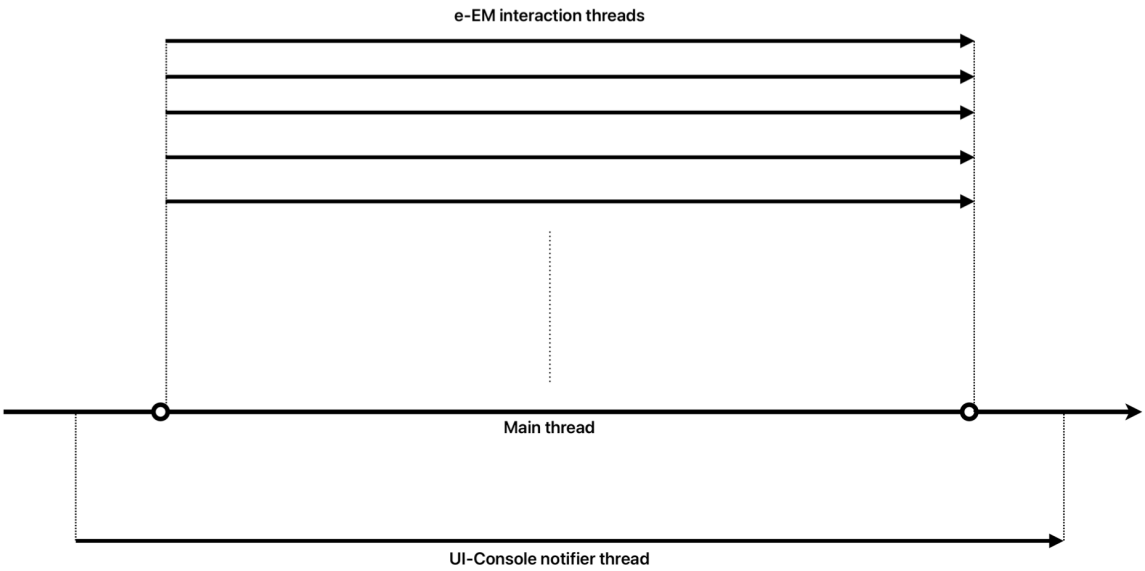


Figure 4.11. An Illustration of the multithreading architecture.

Focus of this section is the worker threads, also known as $\mathbf{e}^-$ - **EM** interaction threads. There were four competing architecture for these worker threads,

- Architecture 1: Have a thread pool that calculates $\mathbf{e}^-$ - **EM** in a queue.
- Architecture 2: Assign a thread to each bunch.
- Architecture 3: A ssign random electrons to each thread and calculate $\mathbf{e}^-$ - **EM** with global time.
- Architecture 4: Assign random electrons to each thread and calculate $\mathbf{e}^-$ - **EM** with local thread time.

Architecture 1 would require constant waiting in worker threads to get mutexes of $\vec{\mathbf{E}}$ field, especially in RK4.

Architecture 2 was performing well in configurations with a large number of bunches, but was not increasing performance in lower bunch count configurations as expected.

Architecture 3 was inefficient and wasteful since all the worker threads would wait the main thread to get the next time after they finish calculation, while the main thread would be waiting for the slowest worker thread.

Architecture 4 was thought to be the best performer. It would give freedom to calculate the whole simulation to each thread while giving up the global time. This would also mean thread-safety is ensured since there is no shared data between the worker threads. However, this architecture can cause issues if $\mathbf{e}^-$ - $\mathbf{e}^-$ interactions were decided to be implemented in the future.

Another caveat is that a copy operation of $\vec{\mathbf{E}}$ and $\vec{\mathbf{B}}$ objects for each worker thread would be needed. This would lead to larger memory allocations and more time spent setting up simulations. Therefore, Architecture 4 is not ideal for fast calculations and when the memory is an important constraint. An implementation that can use both Architecture 2 & Architecture 4 when necessary would be a better approach considering these methods; nevertheless, Architecture 4 was chosed to be implemented.

The implemenation of multithreading can be found in Figures A.8, A.9, A.10 and A.11 in Appendix A.

4.3. Graphical User Interface

Until this point, Rhodotron Simulation could be used with a configration file, defining the problem that would be simulated. An example of this configuration file can be found in Figure B.4 of Appendix B.

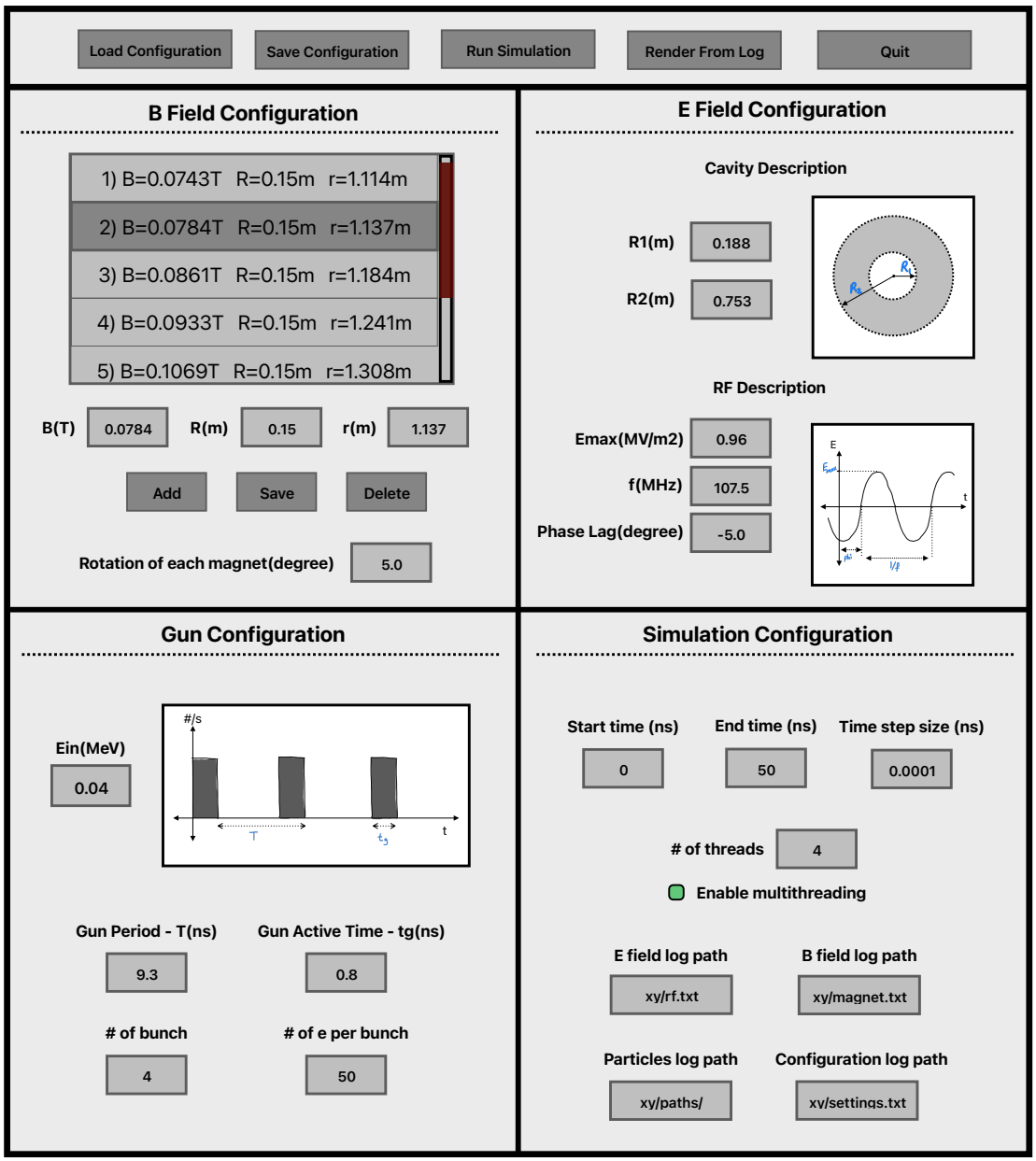


Figure 4.12. An Illustration of the first GUI design.

This approach was simple and fast; however, it was not suitable for the average target user since required basic knowledge of command line interface and was not up to modern standards. For this reason, a GUI was decided to be built, using ROOT framework. This would also enable Rhodotron simulation to make use of analysis tools offered by ROOT.

The initial design of the Rhodotron Simulation GUI can be observed in Figure 4.12. The GUI would be a standalone application, running the Rhodotron Simulation as a service when needed. For this reason, the now called simulation engine was updated to be able to run as a background service of GUI. By this approach, one could also ignore the GUI altogether and use the simulation engine as before, as these are two seperate products.

In the Figure 4.13, the implemented version of Figure 4.12 can be observed. Available frames of this version of the GUI are discussed further in the following sections.

4.3.1. Simulation Frame

This frame spawns and manages the simulation engine, configures and starts the simulation. It has a progress bar that shows the current progress of the simulation, communucating with UI handler thread in simulation engine. Simulation frame while a simulation is running can be observed in Figure D.2 of Appendix D.

4.3.2. Configuration Frame

This frame provides an interface for specifying, saving or loading a configuration, consists of $\vec{\mathbf{B}}$ field, $\vec{\mathbf{E}}$ field, RF description, $\mathbf{e}^{-}$-gun , simulation configuration regions. Each one having an illustration of what the parameters are as can be seen in Figure 4.13.

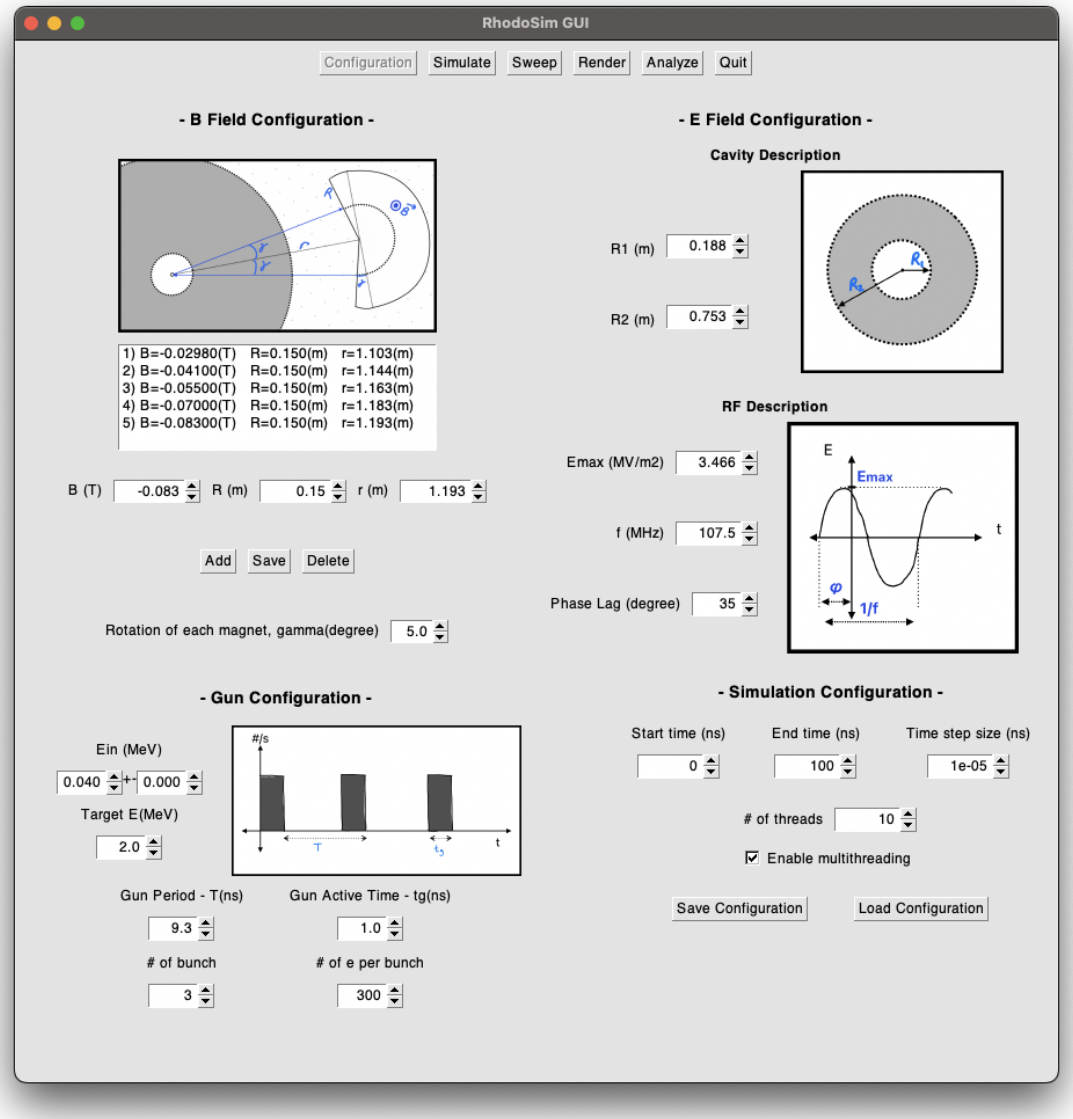


Figure 4.13. The configuration frame of implementated first GUI design.

4.3.3. Render Frame

Since the rendering capabilities of ROOT is superior than gnuplot, the user can render a playback of the simulation in real time, see a specific time frame, export as snapshot or animated gif file. Two example renders can be seen in Figure 4.14 and Figure D.3 of Appendix D.

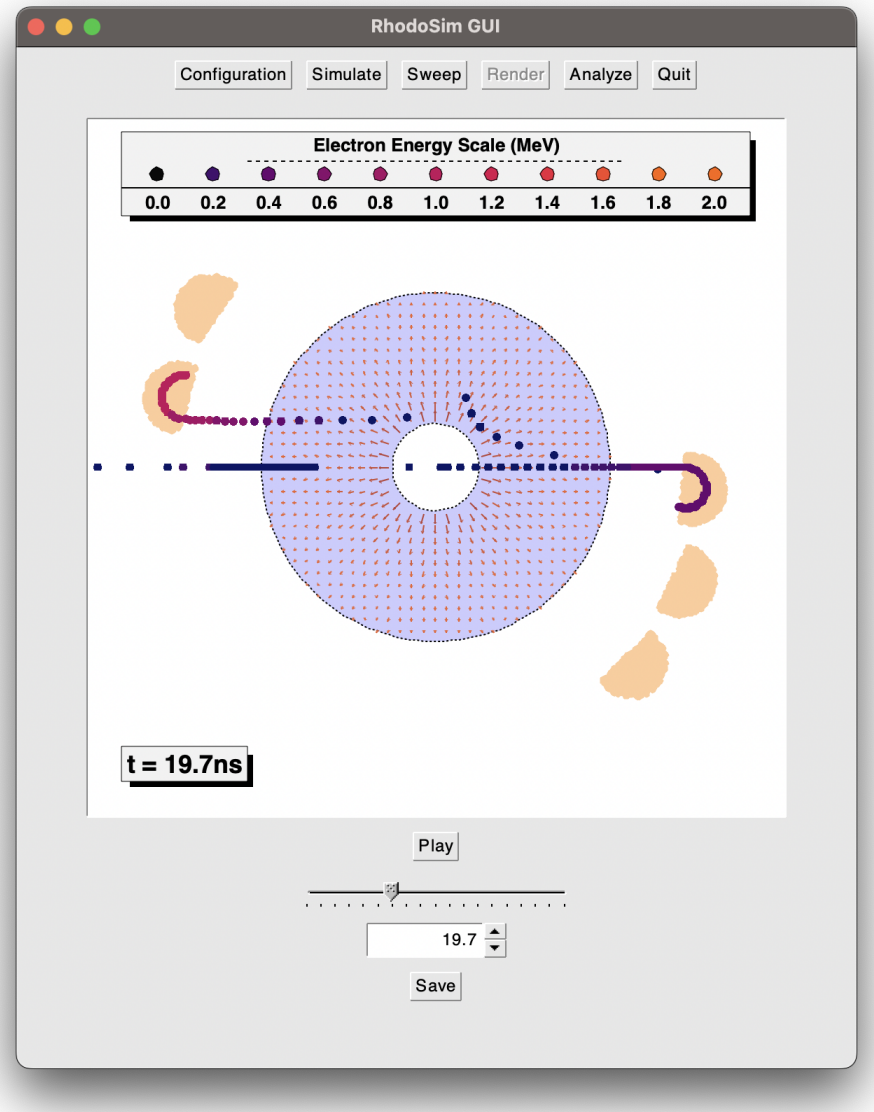


Figure 4.14. Render frame of GUI.

4.3.4. Analyze Frame

Analyze frame provides tools for analyzing and visualizing the simulation data. In the current version, energy distribution histogram and **E(t)** graph of each electron are implementated into this frame. This frame is a work in progress and will be the focus of improvement and become a really powerfull tool in the near future. In Figure 4.15, Analyze Frame plotting energy distribution histogram, and in Figure 4.16 and Figure D.4 of Appendix D, Analyze Frame plotting **E(t)** graphs can be observed.

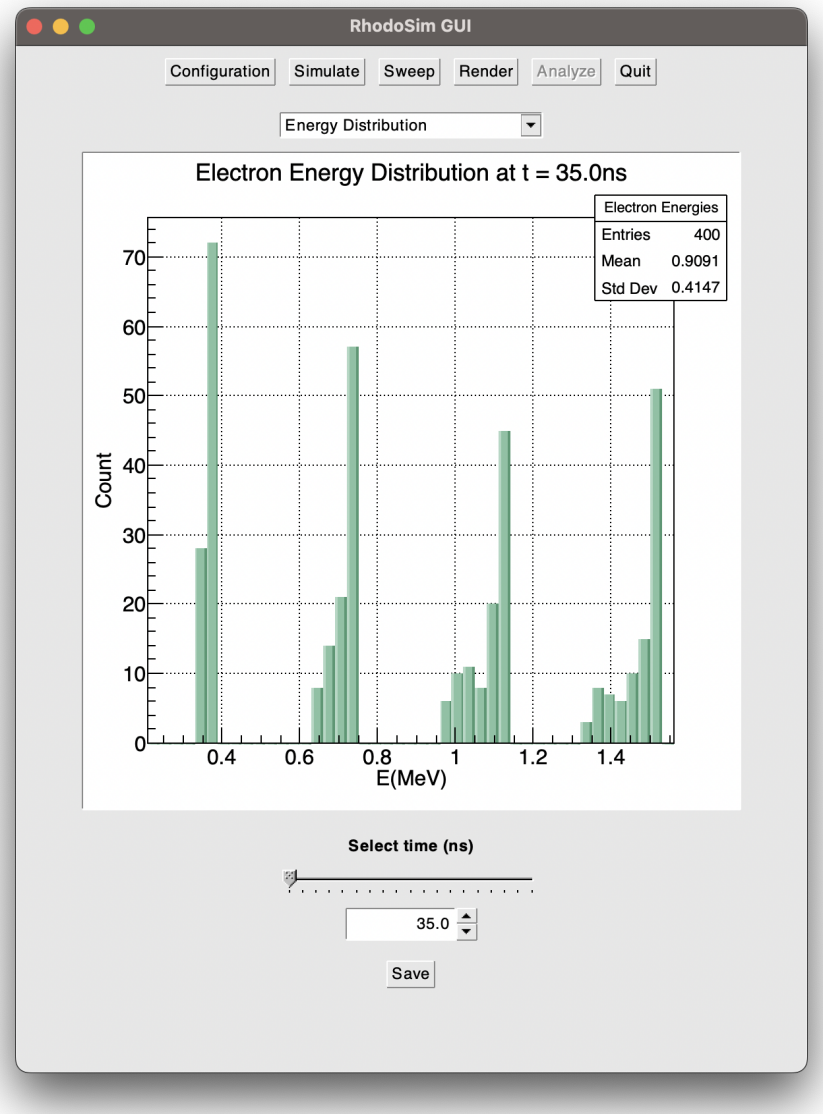


Figure 4.15. Analyze frame of GUI E distribution.

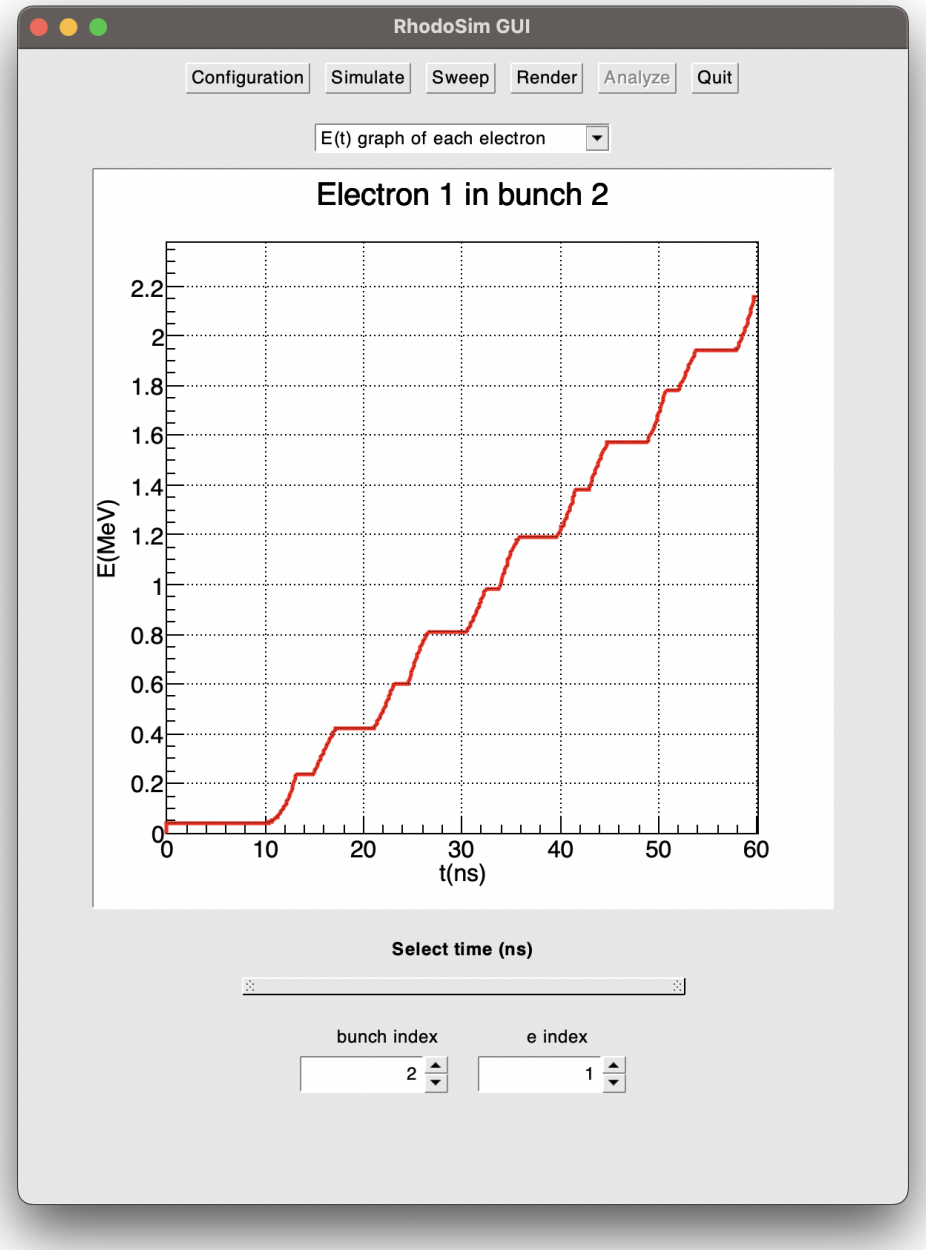


Figure 4.16. Analyze frame of GUI E(t).

4.3.5. Sweep Frame

Parameter sweep method that was discussed in Sections 4.2.1 and 4.2.2 was integrated into GUI in a seperate frame named Sweep Frame. ϕ_{lag} sweep was the first parameter to be implemented.

It takes the range of sweep, draws ϕ_{lag} vs μE, ϕ_{lag} vs σE and ϕ_{lag} vs σR mentioned in Section 4.2.2. This enables user with an already built accelerator to optimize $\mathbf{e}^{-}$-gun parameters quickly. In the Figures 4.17, 4.18, D.5 and D.6 of Appendix D, this frame can be inspected in detail.

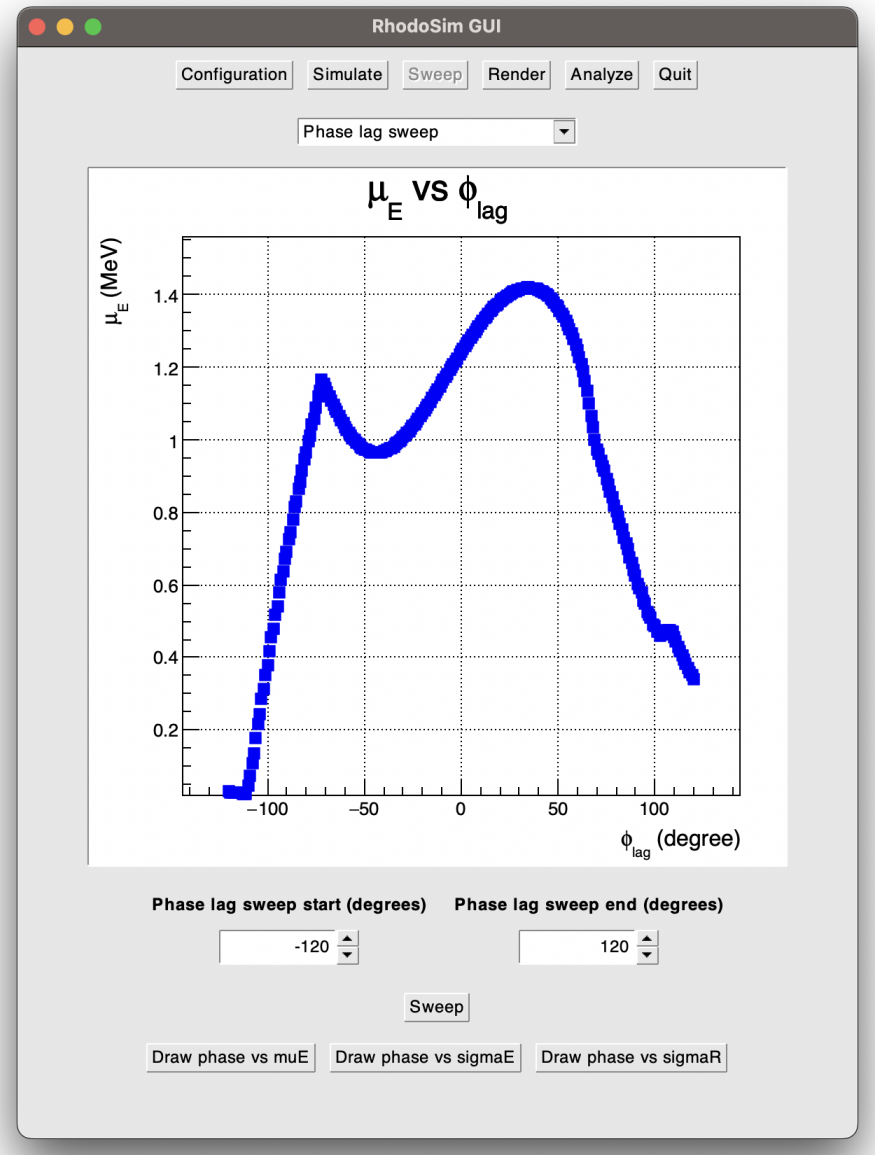


Figure 4.17. Sweep frame of GUI ϕ_{lag} μE result.

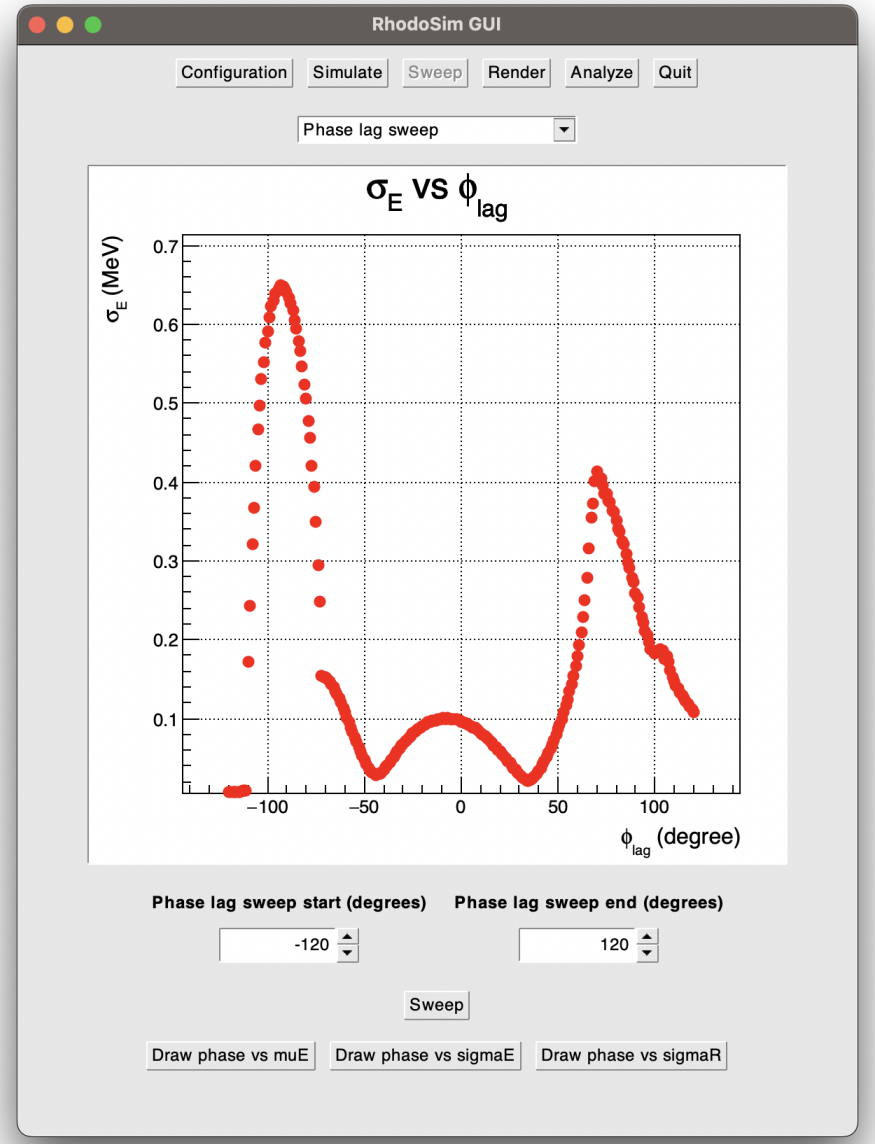


Figure 4.18. Sweep frame of GUI ϕ_{lag} sweep σE result.

As mentioned before, GUI is the latest and ongoing development effort in this project. New analysis and sweep features will be implemented and current capabilities will be improved in the near future.

5. PRODUCTION

As discussed before, production efforts of a Rhodotron-type accelerator are still ongoing in KAHVELab. This accelerator will be used at $f = 107.5$MHz $P = 50 - 100$kW to achieve beams with the energy range of $1 - 5$MeV.

5.1. Final Design

In the Figures 5.1, 5.2, 5.3 and 5.4, rendered visuals of the final design can be observed [13].

Figure 5.1. Rhodotron in KAHVELab.

Figure 5.2. Rhodotron cavity and $\mathbf{e}^-$ gun.

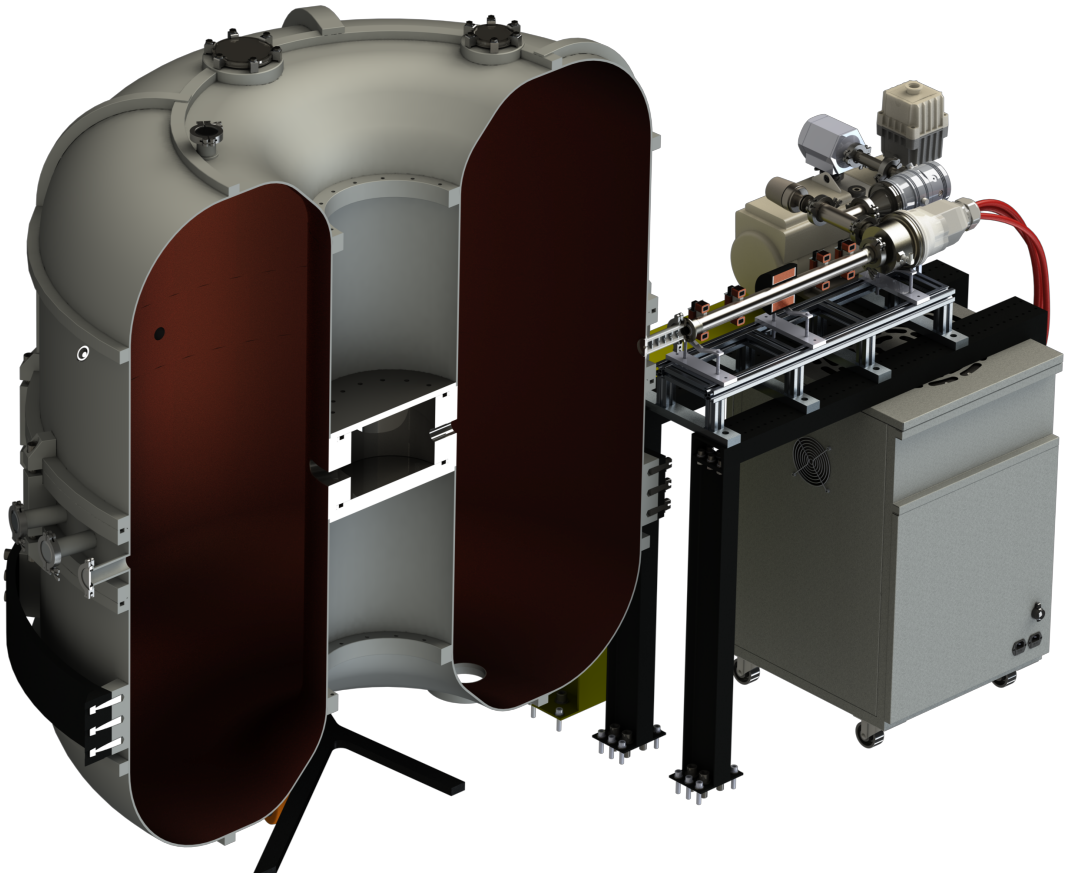


Figure 5.3. Vertical cross section of the beam injection line.

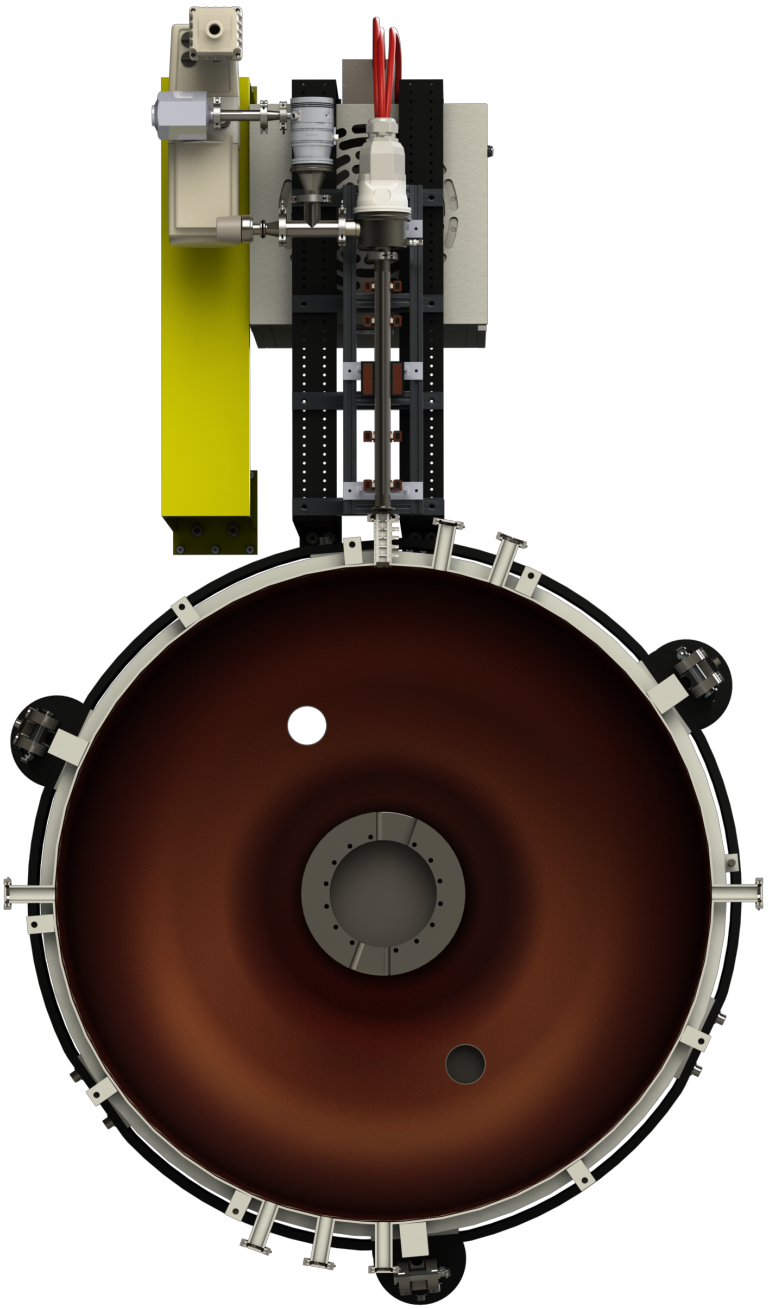


Figure 5.4. Cross section of the acceleration plane.

5.2. Technical Drawing

Technical drawings of the design mentioned in Section 5.1 can be found in Figures 5.5 and 5.6, Figure D.1 of Appendix D.

5.3. Manifacturing

Manifacturing of the rhodotron cavity has been ongoing, planned to be completed in the following months.

The cavity itself was manifactured as 5 main parts, using 5mm thick stainless steel sheet. Elastic version of the 304, 304L was chosen to be the production material for ease of bending. These sheets were then pressed to achieve the shapes of the parts. Several flanges were then machined, including

- 4.5in EIA RF flange for RF input,
- ISO-100KF vacuum connector flanges for vacuum pumps,
- ISO-63 KF flange for probe insertion,
- KF-40 flanges for beam line vacuum gauges.

After desired shapes were produced inside surface of the sheets were polished with 2000 grits. 304 stainless steel sheet bars of different sizes were then added by TIG welding to achieve pressure resistance.

Because the main body of the cavity was 5mm thick and 304L was used, deformations on cylinderical symmetry were encountered. This problem was solved by welding 6 10mm thick toroidal sheets between the coaxial cylinders, which would be removed after the heat treatment.

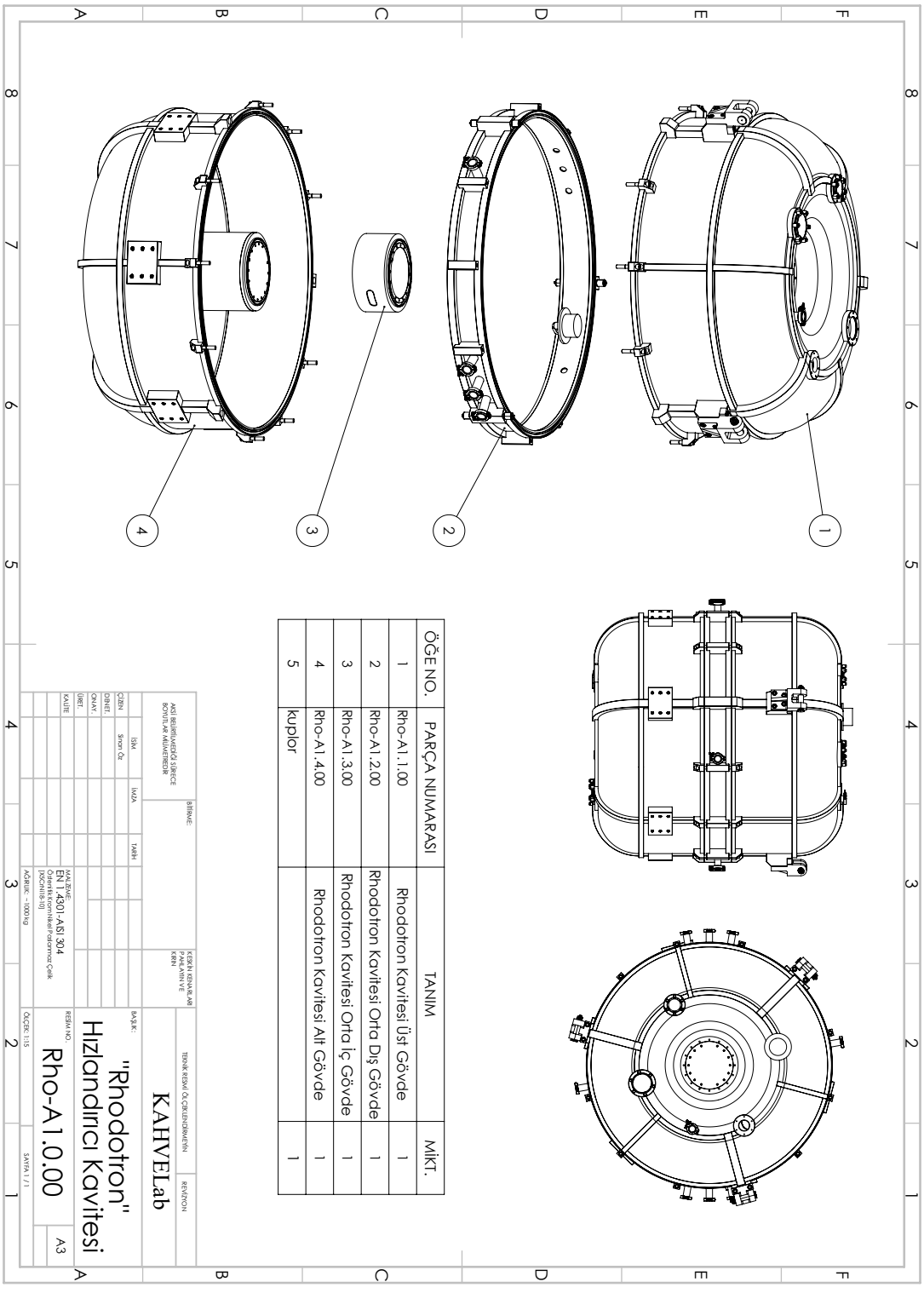


Figure 5.5. Technical drawing of the rhodotron cavity [13].

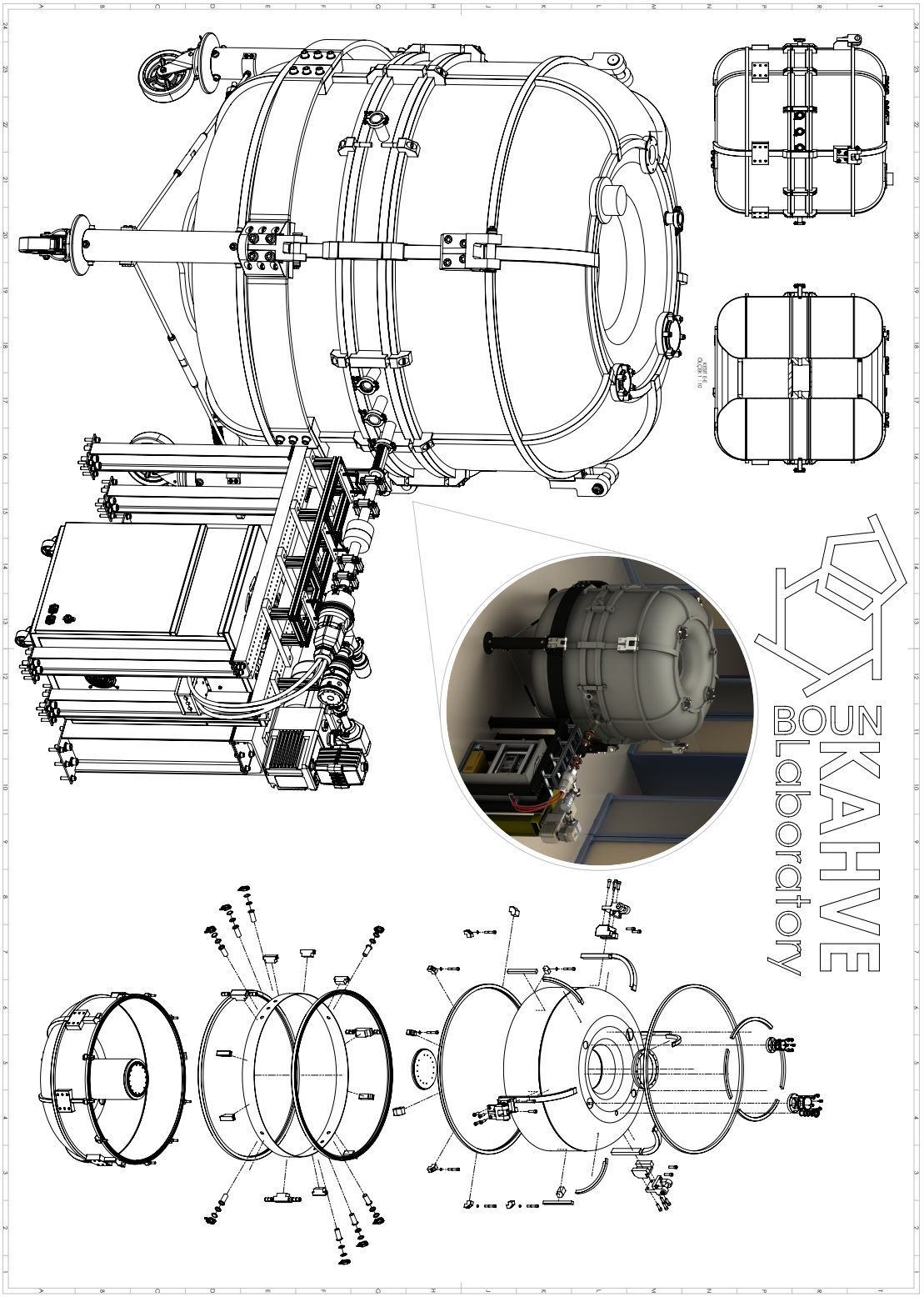


Figure 5.6. Technical drawing of middle part of the cavity which contains the beam line [13].

(a) Bottom part of the cavity while polishing

(b) Top part of the cavity after polishing

Figure 5.7. Polishing of the inner surface of cavity.

(a) Bottom part of the cavity with the exposed beam line in inner cylinder

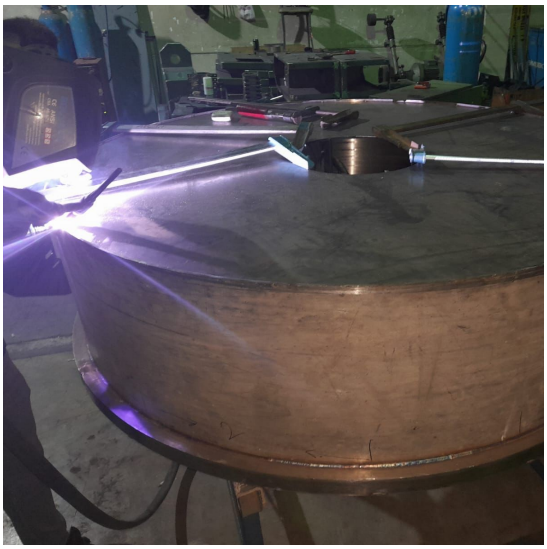
(b) Bottom part of the cavity with the beam line assembled

Figure 5.8. Deformation prevention measure, 10 mm thick toroidal sheets.

Figure 5.9. Welded sheet bars.

Figure 5.10. Middle part of the cavity containing the beam line.

(a) Welding sheet bars



(b) Welding toroidal sheets

Figure 5.11. TIG welding.

(a) Beam enter & exit flanges

(b) RF input & probe flanges

Figure 5.12. Flanges on the assembled cavity.

Figure 5.13. Current stage of the cavity.

6. CONCLUSION

At the time of publication, production efforts of a new rhodotron type electron accelerator are currently continuing in KAHVELab. Present bottleneck is the fabrication of the cavity.

6.1. Future Work

Regarding the cavity manufacturing process, it is in the concluding stages. The sheet bars and the flanges of the upper and the lower parts have been welded, the cavity is getting ready for heat treatment. After the heat treatment operation, the supporting toroidal sheets will be removed and the inside surface will be polished before cyanide copper plating.

As far as the magnets were concerned, it was decided to be kept on hold in the early stages of the project, due to the disagreement on the energy gain amount between the calculations of J. POTTIER [1] and the simulation results from CST Studio Suite & Rhodotron Simulation. The software is planned to be used in magnet design and beam optimizations in KAHVELab after the initial tests are completed.

Rhodotron Simulation, which has been the main focus of this thesis, is ready and waiting to be tested with the cavity that is being manufactured; Several improvements and new feature implementations are underway in the meantime:

- Electric and magnetic field import
- Field generator module to directly produce field files from specified cavity
- Synchrotron radiation calculations to determine radiation and energy loss
- $\mathbf{e}^-$ - $\mathbf{e}^-$ interactions
- Redesign of the GUI
- 3D render in GUI

- Refactoring of the $\mathbf{e}^-$ - **EM** interaction and logging to improve performance
- Refactoring of the GUI Render Frame to improve render speed
- Refactoring the magnet class to introduce field leaks
- Extention of analysis tools in Analyze Frame
- L_{out} sweep in Sweep Frame

After the cavity manufacturing is concluded, energy gain predictions of this software will be tested on this cavity. Complementary bending magnet design will be the first challange of Rhodotron Simulation, which will provide tremendous testing and improvement opportunities.

6.2. Discussion

In conclusion, a new computational $\mathbf{e}^-$ - **EM** interaction simulation and analysis tool that focuses on designing and improving Rhodotron-type accelerators has been successfully developed; a Rhodotron-type accelerator with an operating frequency of 107.5 MHz and a target energy of $1 - 5$ MeV has been designed and is currently being manufactured.

The software features a robust GUI that is capable of rendering and analyzing simulation results. Leap-frog and Runge-Kutta numerical methods have been implemented for initial speed and accuracy tests. The real-world performance and accuracy of this new tool, on the other hand, cannot be tested before the cavity production is completed and the performance is compared with the predictions of the tool; However, the capabilities it presents, coupled with the analogous results obtained from other extensively utilized simulation tools, lead to the inference that it holds promise.

REFERENCES

1. Jacques Pottier, *A new type of rf electron accelerator: The rhodotron*, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, Vol. 40–41, No 2, pp 943-945, 1989.

2. Adıgüzel, A., S. Açıksöz, A. Çağlar, H. Çetinkaya, Ş. Esen, D. Halis, A. Hamparsunoğlu, T.B. İlhan, A. Kılıçgedik, O. Koçer, S. Oğur, S. Öz, A. Özbey, V.E. Özcan and N.G. Ünel *Ion source and LEBT of KAHVELab proton beamline*, Journal of Instrumentation, Vol. 18, 2023

3. Bassaler, J.M., J.M. Capdevila, O. Gal, F. Lainé, A. Nguyen, J.P. Nicolaï, K. Umiastowski, *Rhodotron: an accelerator for industrial irradiation*, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, Volume 68, No. 1–4, Pages 92-95, 1992.

4. Jongen, Y., *Manufacturing of Electron Accelerators*, Ion Beam Applications (IBA) Chemin du Cyclotron 3, B-1348 Louvain-la-Neuve, Belgium. 2001.

5. Jensen E., in Proceedings of the CAS–CERN Accelerator School: Advanced Accelerator Physics CERN–2014–009, edited by W. Herr, Trondheim, Norway, pp. 405–430, 2013.

6. Fukushima, T., *Reduction of Round-off Errors in Symplectic Integrators*, The Astronomical Journal, Vol. 121, pp. 1768-1775, 2001.

7. User's Guide for the POISSON / SUPERFISH group of codes, Los Alamos National Laboratory, New Mexico, USA, 1987. Accessed on Aug 10, 2023.

8. The European Organization for Nuclear Research, "ROOT - An Object Oriented Data Analysis Framework Release v6.20/04, 01/04/2020", 2020, https://zenodo.org/record/3895855, accessed on Aug 10, 2023.

9. Williams, T, "Gnuplot 5.4: An Interactive Plotting Program", 2020, http://gnuplot.info, accessed on Aug 10, 2023.

10. Runge, C. D. T., *Über die numerische Auflösung von Differentialgleichungen*, Mathematische Annalen, Springer, Vol. 46, No. 2, pp. 167–178, 1895.

11. Kutta, W, *Beitrag zur näherungsweisen Integration totaler Differentialgleichungen*, Zeitschrift für Mathematik und Physik, Vol. 46, pp. 435–453, 1901.

12. Kleeven, W., M. Abs, J. Brison, E. Forton, J. Hubert and J. Van de Walle, *Design and Simulation Tools for the High-Intensity Industrial Rhodotron Electron Accelerator*, 9th International Particle Accelerator Conference, 2018.

13. Privite communication with Sinan Öz.

APPENDIX A: Intermediate Codes

```
for(double i = 2; i < 9; i += dT_out){
    t_dum += i;
    double enow = gecis(r_pos, vel, Et, t_dum);
    if( enow > maxE ){
      maxE = enow;
      t_opt = i;
    }
    t_dum = t;
}
```

```
double gecis(double r_pos, double vel, double Et, double &t){
    for(; r_pos >= -R2 && r_pos <= R2 ; t+=dT){
        vel = c*sqrt(Et*Et-E0*E0)/Et;
        double RelBeta  = vel/c;
        double RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);

        double ef=Eradial(r_pos*1000,t,RFphase*deg_to_rad);

        double acc=ef*1E6*eQMratio/(RelGamma*RelGamma*RelGamma);

        r_pos = r_pos + vel * dT*ns + 1/2*acc*(dT*ns)*(dT*ns);
        vel=vel+acc*dT*ns;
        RelBeta  = vel/c;
        RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);
        Et=RelGamma*E0;
    }
    return Et;
}
```

Figure A.1. L_{out} Optimization For Single $\mathbf{e}^-$.

```
int phase_opt(const vector<double>& Louts, int phase_sweep_range){
    double minrms = 1;
    int opt_phase;
    for(int RFphase = -phase_sweep_range; RFphase <= phase_sweep_range; RFphase++){
        Bunch bunch1(RFphase);
        double t1 = 0;
        bunch1.bunch_gecis_t(t1);
        bunch1.reset_pos();

        for(int i = 0; i < Louts.size(); i++){
            bunch1.bunch_gecis_d(Louts[i]);
            bunch1.reset_pos();
        }

        if( bunch1.E_rms() < minrms ){
            minrms = bunch1.E_rms();
            opt_phase = RFphase;
        }
    }
    return opt_phase;
}
```

Figure A.2. ϕ_{lag} Optimization For Initial Bunch Design.

```
double vector3d::operator* (const vector3d& other){
    double dot = 0;
    dot += this->x * other.x;
    dot += this->y * other.y;
    dot += this->z * other.z;
    return dot;
}
```

```
vector3d vector3d::operator% (const vector3d& other){
    double x_ = (this->y * other.z) - (this->z * other.y);
    double y_ = (this->z * other.x) - (this->x * other.z);
    double z_ = (this->x * other.y) - (this->y * other.x);
    vector3d crossed(x_, y_, z_);
    return crossed;
}
```

Figure A.3. * and % operators of *vector3d* class.

```
vector3d Electron2D::interactB_RK(const MagneticField& B, double time_interval){
    if (B.isInside(pos) == -1){
        return vector3d(0,0,0);
    }
    Electron2D e_dummy;
    e_dummy.Et = Et;
    e_dummy.pos = pos;
    e_dummy.vel = vel;
    double time_halved = time_interval*0.5;
    // get k1
    vector3d F_m = (e_dummy.vel % B.getField(pos))*eQMratio;
    vector3d k1 = F_m * e_dummy.gamma_inv();
    // get k2
    e_dummy.move(time_halved);
    e_dummy.accelerate(k1, time_halved);
    F_m = (e_dummy.vel % B.getField(pos))*eQMratio;
    vector3d k2 = F_m * e_dummy.gamma_inv();
    // get k3
    e_dummy.vel = vel;
    e_dummy.accelerate(k2, time_halved);
    vector3d k3 = F_m * e_dummy.gamma_inv();
    // get k4
    e_dummy.vel = vel;
    e_dummy.move(time_halved);
    e_dummy.accelerate(k3, time_interval);
    F_m = (e_dummy.vel % B.getField(pos))*eQMratio;
    vector3d k4 = F_m * e_dummy.gamma_inv();

    return (k1 + k2*2 + k3*2 + k4)/6;
}
```

Figure A.4. RK4-1 implemenation of $\mathbf{e}^{-}$ -$\vec{\mathbf{B}}$

```
bool isInsideHalfSphere(vector3d e_position, double r, vector3d hs_position){
    vector3d relative = e_position - hs_position;
    // r/5 can be changed, use this for now
    if ( relative.magnitude() <= r && relative * hs_position.direction() >= -r/5){
        return true;
    }
    return false;
}
```

Figure A.5. Logic of is $\mathbf{e}^-$ inside the shape of magnet.

```
void Electron2D::interactRK_ActorE(const RFField& E, const MagneticField& B, double time_interval){
    vector3d run_kut_E = interactE_RK(E, time_interval);
    vector3d run_kut_B = interactB_RK(B, time_interval);

    vector3d acc = run_kut_E + run_kut_B;

    move(acc, time_interval/2);
    accelerate(acc, time_interval);
    move(acc, time_interval/2);
}
```

Figure A.6. RK4-1 implemenation of $\mathbf{e}^-$ - **EM** .

```
1   void Electron2D::interactRK(RFField& E, MagneticField& B, const double time, double time_interval){
2       Electron2D e_dummy;
3       e_dummy.Et = Et;
4       e_dummy.pos = pos;
5       e_dummy.vel = vel;
6
7       // Calculate k1
8       vector3d acc_E = E.actOn(e_dummy);
9       vector3d acc_B = B.actOn(e_dummy);
10      vector3d acc = acc_E + acc_B;
11      e_dummy.move(time_interval);
12      e_dummy.accelerate(acc, time_interval);
13      vector3d pos_k1 = e_dummy.pos, vel_k1 = e_dummy.vel;
14
15      // Calculate k2
16      e_dummy.pos = (pos + pos_k1)*0.5;
17      e_dummy.vel = (vel + vel_k1)*0.5;
18      e_dummy.Et = e_dummy.gamma()*E0;
19      E.update(time + time_interval*0.5);
20
21      acc_E = E.actOn(e_dummy);
22      acc_B = B.actOn(e_dummy);
23      acc = acc_E + acc_B;
24      e_dummy.move(time_interval);
25      e_dummy.accelerate(acc, time_interval);
26      vector3d pos_k2 = e_dummy.pos, vel_k2 = e_dummy.vel;
27
28      // Calculate k3
29      e_dummy.pos = (pos + pos_k2)*0.5;
30      e_dummy.vel = (vel + vel_k2)*0.5;
31      e_dummy.Et = e_dummy.gamma()*E0;
32      E.update(time + time_interval*0.5);
33
34      acc_E = E.actOn(e_dummy);
35      acc_B = B.actOn(e_dummy);
36      acc = acc_E + acc_B;
37      e_dummy.move(time_interval);
38      e_dummy.accelerate(acc, time_interval);
39      vector3d pos_k3 = e_dummy.pos, vel_k3 = e_dummy.vel;
40
41      // Calculate k4
42      E.update(time + time_interval);
43
44      acc_E = E.actOn(e_dummy);
45      acc_B = B.actOn(e_dummy);
46      acc = acc_E + acc_B;
47      e_dummy.move(time_interval);
48      e_dummy.accelerate(acc, time_interval);
49      vector3d pos_k4 = e_dummy.pos, vel_k4 = e_dummy.vel;
50
51      E.update(time);
52      pos = (pos_k1 + pos_k2*2 + pos_k3*2 + pos_k4)/6;
53      vel = (vel_k1 + vel_k2*2 + vel_k3*2 + vel_k4)/6;
54      Et = gamma()*E0;
55  }
```

Figure A.7. RK4-2 implemenation of $\mathbf{e}^-$ - **EM** .

```
void RhodotronSimulator::_runMT(){
    gun.fireAllWithFireTimesMT();

    MTEngine.setupPool(time_interval, start_time, end_time, gun, E_field, B_field, gun.thread_bunchs);

    STEPS_TAKEN = 0;
    simulation_time = start_time;
    while (simulation_time < end_time + time_interval ){
        if (STEPS_TAKEN % log_interval() == 0){
            E_field.update(simulation_time);
            logEfield(simulation_time, simulation_time + time_interval > end_time);
            notifyUI(MTEngine.getAverageTime());
        }
        simulation_time+=time_interval;
        STEPS_TAKEN++;
    }
    bool end = false;
    while (!end){
        double time = MTEngine.getAverageTime();
        notifyUI(time);
        if ( time >= end_time ){
            end = true;
        }
        this_thread::yield();
    }
    MTEngine.join();

}
```

Figure A.8. Multithreading main-thread logic.

```
void Gun::fireAllWithFireTimesMT(){
    std::random_device rd;
    std::mt19937 e2(rd());
    std::normal_distribution<double> Edist(Ein, sEin);

    for(_fired_bunch= 0; _fired_bunch < bunch_count; _fired_bunch++){
        for(_fired_e_in_current_bunch= 0; _fired_e_in_current_bunch < e_per_bunch; _fired_e_in_current_bunch++){

            double E = (sEin == 0 ) ? Ein : Edist(e2);

            double fire_time = (ns_between_each_electron_fire * _fired_e_in_current_bunch) + _fired_bunch*gun_period;

            auto burrowed_e = bunchs[_fired_bunch].AddElectronGiveAddress(E, gunpos, gundir, fire_time);

            int thread_index = (_fired_e_in_current_bunch + _fired_bunch*e_per_bunch)%thread_bunchs.size();

            thread_bunchs[thread_index]->push_back(burrowed_e);
        }
    }
}
```

Figure A.9. Multithreading electron assign logic.

```
void MultiThreadEngine::setupPool( double _time_interval, double _start_time, double _end_time, Gun& gun,
    CoaxialRFField& RF, MagneticField& B, vector<shared_ptr<vector<shared_ptr<Electron2D>>>>& e_list){
    threads.reserve(thread_count);

    for(int i = 0; i < thread_count && i == threads.size(); i++){

        child_notifier_mutexes.push_back(make_shared<mutex>());
        child_times.push_back(make_shared<double>());

        auto _E = RF.Copy();
        auto _B = B.LightWeightCopy();
        double time_between_fires = thread_count*gun.getGunActiveTime()/gun.getElectronsPerBunch();
        double first_fire_time = i*gun.getGunActiveTime()/gun.getElectronsPerBunch();
        ThreadArguments thread_arguments(i, _time_interval, _start_time, _end_time, &gun, _E, _B, e_list[i], first_fire_time, time_between_fires);
        thread_arguments.parent_notifier_mutex = child_notifier_mutexes[i];
        thread_arguments.current_thread_time = child_times[i];
        threads.push_back(thread(threadLoop, thread_arguments));
    }
}
```

Figure A.10. Multithreading worker-threads setup logic.

```
void threadLoop(ThreadArguments thread_arguments){
    uint64_t count = 0;

    double sim_time = thread_arguments.start_time;

    while(sim_time < thread_arguments.end_time + thread_arguments.time_interval){
        thread_arguments.E->update(sim_time);
        thread_arguments.i_args.time = sim_time;

        if ( count % (unsigned long)(0.1/thread_arguments.time_interval) == 0){
            saveElectronInfoForSingleThread(thread_arguments.i_args);
            // Notifiy the main thread
            if(thread_arguments.parent_notifier_mutex->try_lock()){
                *thread_arguments.current_thread_time = sim_time;
                thread_arguments.parent_notifier_mutex->unlock();
            }
        }

        interactForSingleThread(thread_arguments.i_args);
        // save electron info here
        sim_time+= thread_arguments.time_interval;
        count++;
    }
    thread_arguments.parent_notifier_mutex->lock();
    *thread_arguments.current_thread_time = sim_time;
    thread_arguments.parent_notifier_mutex->unlock();
}
```

Figure A.11. Multithreading worker-thread logic.

```
1  void UIThreadWork(UIThreadArgs args){
2      int UI_WORK_PIECE = SIM_WORK_MASK;
3      if ( !args.isService ) {
4          UI_WORK_PIECE = 50;
5      }
6
7      double piece = (args.end_time - args.start_time)/UI_WORK_PIECE;
8
9      if ( !args.isService ) {
10         std::string sim_running_msg = "...Simulation is running...";
11         for(int i = 0; i < 26 - sim_running_msg.size()/2 ; i++){
12             std::cout << " ";
13         }
14         std::cout << sim_running_msg <<"\n";
15     }
16
17     args.ui_mutex->lock();
18     double simtime = *(args.simulation_time);
19     args.ui_mutex->unlock();
20
21     if ( !args.isService ) {
22         std::cout << "V";
23         for(int i = 0; i < 51; i++){
24             std::cout << "_";
25         }
26         std::cout << "V\n[" << std::flush;
27     }
28
29     int count = 0;
30     bool running = true;
31     while(running && (simtime < args.end_time || count < UI_WORK_PIECE )){
32         if( simtime > count * piece ){
33             if ( !args.isService ) {
34                 std::cout << "#" << std::flush;
35             }
36             count++;
37             args.state_mutex->lock();
38             running = *args.state_ptr & SIM_RUNNING;
39             *args.state_ptr = (*args.state_ptr & ~SIM_WORK_MASK) | (count & SIM_WORK_MASK);
40             if (args.isService) write(args._fd, args.state_ptr, SIGNAL_SIZE);
41             args.state_mutex->unlock();
42         }
43         std::this_thread::sleep_for(std::chrono::milliseconds(25));
44         args.ui_mutex->lock();
45         simtime = *(args.simulation_time);
46         args.ui_mutex->unlock();
47
48     }
49     if ( !args.isService ) {
50         std::cout << "#]\n\n" << std::flush;
51         std::cout << "     ...Simulation is finished successfully...\n\n" << std::flush;
52     }
53
54     args.state_mutex->lock();
55     *args.state_ptr |= SIM_RENDERING;
56     if ( args.isService ) write(args._fd, args.state_ptr, SIGNAL_SIZE);
57     args.state_mutex->unlock();
58 }
```

Figure A.12. UI-Console handler thread logic.

```
class Electron{
    std::vector<ElectronLog> log;
    uint64_t index;
    double Et;
    vector3d pos;
    vector3d vel;
    double fire_time;
public:
    Electron();
    Electron(double Ein, vector3d position, vector3d direction, double _fire_time = 0);
    ~Electron();

    void setEin(double E_in);
    void print_electron_info();

    void move(double dt);
    void move(const vector3d& acc, double dt);
    void move(const vector3d& acc, const vector3d& jerk, double dt);
    void accelerate(const vector3d& acc, double dt);
    void accelerate(const vector3d& acc, const vector3d& jerk, double dt);

    void interactLF(RFField& E, MagneticField& B, double time_interval);
    void interactRK(RFField& E, MagneticField& B, double time_interval);
    void interactRK(RFField& E, MagneticField& B, const double time, double time_interval);
    void interactRK_ActorE(const RFField& E, const MagneticField& B, double time_interval);

    vector3d interactE_RK(const RFField& E, double time_interval);
    vector3d interactB_RK(const MagneticField& B, double time_interval);

    void saveInfo(double t);
    void setLogSize(size_t size);
    void loge(DataStorage& path);
    const vector<ElectronLog>& getLog();

    double vel();
    double beta();
    double beta2();
    double gamma();
    double gamma_inv();
};
```

Figure A.13. Electron class definition.

```
class Bunch{
private:
    int e_count = 0;
    int index_fastest = 0;
    double max_energy = 0;
    double entry_time = 0;
    double E_in = 0.04;

    vector<Bunch> subBunchs;
    vector<shared_ptr<Electron>> e;
public:
    Bunch(unsigned int num_of_electrons, double Ein, vector3d gunpos, vector3d gundir, double gun_ns);
    Bunch();
    ~Bunch();

    void saveInfo(double time);

    Electron& getFastest();

    void AddElectron(double Ein, const vector3d& gunpos, const vector3d& gundir, double fire_time);

    shared_ptr<Electron> AddElectronGiveAddress(double Ein, const vector3d& gunpos, const vector3d& gundir, double fire_time);

    void setEntryTime(double entry_time);
    void setEin(double E_in);
    void setNSLen(double len);

    uint32_t get_e_count();
    double getEin();
    double E_ave();
    double E_rms();

    void interact(RFField& E, MagneticField& B, const double time, double time_interval);

    void divide(unsigned int num);
    void concat();
    Bunch& subBunch(unsigned int index);
    vector<Bunch*> subBunchPtr();

    void reset();

    void print_bunch_info();
    void print_summary();

    void logPaths(vector<DataStorage>& pathStorage, std::string header);
};
```

Figure A.14. Bunch class definition.

```
class Gun{
    double Ein;
    double sEin;
    double gun_period;
    double gun_active_time;
    double ns_between_each_electron_fire;

    vector3d gunpos;
    vector3d gundir = vector3d(1,0,0);

    uint64_t bunch_count = 0;
    uint64_t e_per_bunch = 1;

    uint32_t _fired_bunch = 0;
    uint32_t _fired_e_in_current_bunch = 0;
    bool _firing = false;

    bool _mt_enabled = false;
    uint32_t _child_thread_count = 0;

    vector<Bunch> bunchs;

    mutex _gun_mutex;
    vector<shared_ptr<vector<shared_ptr<Electron>>>> thread_bunchs;

    void setNSLen(double len);
public:
    Gun();
    Gun(double Ein, double gun_active_time, double pulse_interval, vector3d gunpos);
    ~Gun();
    void interact(RFField& E, MagneticField& B, const double time, double time_interval);

    void fireIfActive(double time);
    void fireAllWithFireTimesMT();

    void addBunch(unsigned int num_of_electrons, double Ein);

    void setGunActiveTime(double gt);
    double getGunActiveTime(){return gun_active_time;}
    void setGunInterval(double guninterval);
    void setGunPos(vector3d gun_pos);
    void setGunDir(vector3d gun_dir);
    void setEin(double Ein) ;
    void setEinStd(double EinStd) ;
    void setNumberOfElectrons(uint64_t e_num);
    void getElectronsPerBunch(){return e_per_bunch;}
    void setNumberOfBunchs(uint64_t b_num);
    void enableMT(uint32_t thread_count);

    void saveInfo(double time);
    void logPaths(vector<vector<DataStorage> >& pathsStorage, std::string pathsPath, std::string header);
};
```

Figure A.15. Gun class definition.

```
class Simulator{        // E in MV/m,   En in MeV,   B in T,    t in ns
protected:
    Gun gun;
    double Emax; double freq = 107.5; double phase_lag = 0;
    DataStorage EfieldStorage; DataStorage BfieldStorage;
    vector<vector<DataStorage>> pathsStorage;
    std::string configPath, pathsPath;
    std::string EfieldLogHeader, BfieldLogHeader, eLogHeader;

    double simulation_time = 0;
    double start_time = 0;
    double end_time = 45;
    double time_interval = 0.0001;
    uint64_t STEPS_TAKEN = 0;
    double GUN_ACTIVE_TIME = 1;
    double GUN_PERIOD = 9.3;
    uint64_t NUM_OF_ELECTRONS = 1;
    uint64_t NUM_OF_BUNCHS = 1;
    double Ein = 0.04;
    double EinStd = 0;
    vector3d gunPosition;
    vector3d gunDirection = vector3d(1,0,0);

    shared_ptr<mutex> ui_mutex;
    UIHandler ui_handler;
    shared_ptr<mutex> state_mutex;
    uint8_t state = 0x0;

    MultiThreadEngine MTEngine;
    unsigned int MAX_THREAD_COUNT = 1;
public:
    Simulator();
    ~Simulator(){}

    void enableMultiThreading(unsigned int thread_count);
    void setdT(double dT);
    void setEin(double E_in);
    void setEinStd(double Ein_Std);
    void setNumberofElectrons(uint64_t num_of_electrons);
    void setNumberofBunchs(uint64_t num_of_bunchs);
    void setSimTime(double starttime, double endtime);
    void setGunActiveTime(double gun_ns);
    void setGunPeriod(double pi);
    void setGunPosition(vector3d pos);
    void setGunDirection(vector3d dir);
    void setRFPath(std::string path);
    void setBPath(std::string path);
    void setPathsPath(std::string path);
    void setConfigPath(std::string path);

    virtual void run();
    void openLogs();
    void closeLogs();
    virtual void getConfig(Configuration& config) = 0;
    virtual void logEfield(double time, bool) = 0;
    virtual void logBfield() = 0;
    void logPaths();
    shared_ptr<mutex> getUIMutex();
};
```

Figure A.16. Simulator class definition.

```
class RhodotronSimulator : public Simulator{
private:
    double R1;
    double R2;
    CoaxialRFField E_field;
    MagneticField B_field;

    void _runMT();
    void _runST();
public:
    RhodotronSimulator(double phase_lag);
    RhodotronSimulator(Configuration& config);

    void StartUIHandler();
    void StopUIHandler();
    void DeclareService(std::string pipe_name);

    void setFreq(double frequency);
    void setPhaseLag(double phase_lag);
    void setEmax(double E_max);
    void setR1(double r1);
    void setR2(double r2);
    void addMagnet(double B, double r, vector3d position);
    void addMagnet(Magnet m);

    void updateSimulation();
    void getConfig(Configuration& config);
    void logEfield(double time, bool end);
    void logBfield();

    void run();
    void notifyUI(double time);

    void setState(uint8_t state_);
};
```

Figure A.17. RhodotronSimulator class definition.

```cpp
class RFField{
protected:
    double E;                       // MV/m
    double E_max = 0;               // MV/m
    double frequency = 107.5;       // MHz
    double phase_lag = 0;           // degree
public:
    RFField();
    RFField(double phase_lag);
    virtual ~RFField() {}

    virtual vector3d getField(vector3d position);
    virtual double getField(double R);
    virtual int log(DataStorage& rf, double time, bool end = false);
    vector3d actOn(Electron& e);
    vector3d actOnAndGetRungeKuttaCoef(Electron& e, double dt);

    double getE() {return E;}
    void setEmax(double E_max) {this->E_max = E_max; update(0);}
    void setFreq(double freq){ frequency = freq;}
    void setPhaseLag(double phaselag){ phase_lag = phaselag; update(0);}
    virtual void update(double time);
};
```

```cpp
class CoaxialRFField : public RFField{
private:
    double r1 = 0.188;                      // m
    double r2 = 0.753;                      // m
    double E_max_pos = r1;                  // m
    std::vector<CoaxialRFField*> _childs;
    double E_radial(double R) const;
public:
    CoaxialRFField();
    CoaxialRFField(double phase_lag);
    ~CoaxialRFField() override;
    void setR1(double r1);
    void setR2(double r2);
    void setEmaxPos(double Emaxpos);

    vector3d getField(vector3d position) const override;
    double getField(double R) const override;
    vector3d actOn(Electron& e);
    vector3d actOnAndGetRungeKuttaCoef(Electron& e, double dt);
    void update(double time) override;

    int log(DataStorage& rf, double time, bool end = false) override;

    std::shared_ptr<CoaxialRFField> Copy();
    void split(uint32_t amount_of_child);
    CoaxialRFField* child(uint32_t index);
};
```

Figure A.18. CoaxialRFField class definition.

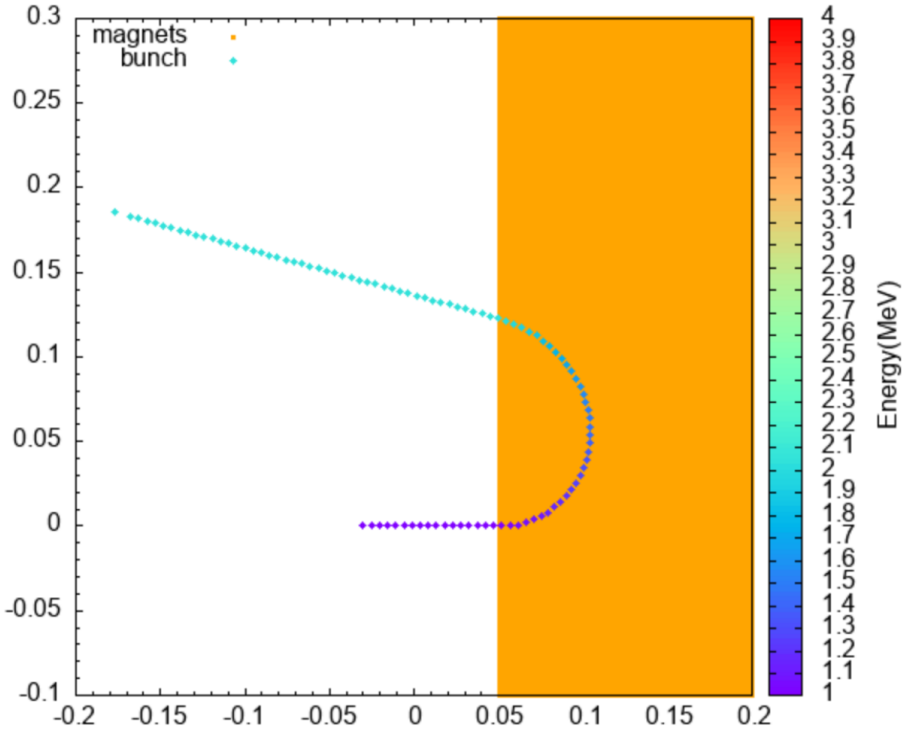
APPENDIX B: Example Simulation Runs

```
Optimal phase with the least RMS : -5

Simulation settings :
ph = -5 deg, gt = 1 ns, enum = 1000
dT = 0.001 ns, dT_out = 0.01 ns

For the 1th magnet:
Optimum out path = 0.81044 m
Magnet guide = 0.25852 m
Rho = 0.088477 m
Drift time of the first electron in the bunch : 7.688 ns
Drift time of the last electron in the bunch : 7.487 ns
Max energy = 0.47581 MeV
RMS = 0.0058165 MeV

For the 2th magnet:
Optimum out path = 1.0833 m
Magnet guide = 0.37766 m
Rho = 0.098898 m
Drift time of the first electron in the bunch : 5.597 ns
Drift time of the last electron in the bunch : 5.617 ns
Max energy = 0.89172 MeV
RMS = 0.0099018 MeV

For the 3th magnet:
Optimum out path = 1.1705 m
Magnet guide = 0.41573 m
Rho = 0.10223 m
Drift time of the first electron in the bunch : 5.314 ns
Drift time of the last electron in the bunch : 5.325 ns
Max energy = 1.298 MeV
RMS = 0.013879 MeV

Electron with the most energy : 623) 1.6999 MeV,        RMS of bunch : 0.017981 MeV

Total steps calculated : 12468052652
Simulation finished in : 632050015 us     ( 632.1 s )
```

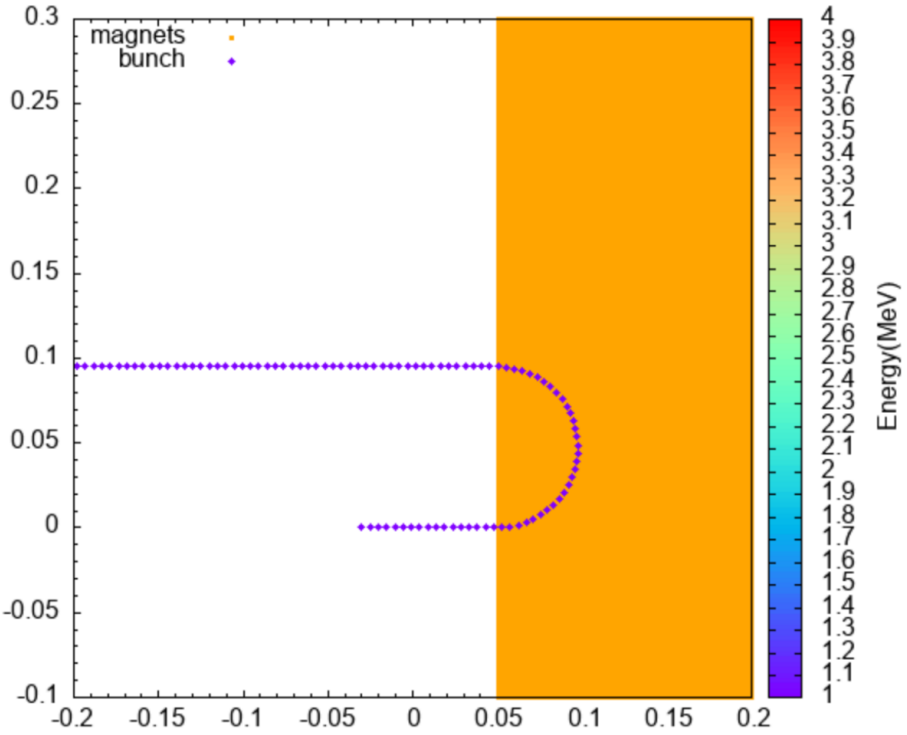
Figure B.1. ϕ_{lag}, ρ & L optimization at $P = 30$KW, $R_1 = 0.188$m, $R_2 = 0.753$m, $t_g = 1$ns, $E_{in} = 40$KeV.

```
Optimal phase with the least RMS : 0

Simulation settings :
ph = 0 deg, gt = 0.8 ns, enum = 1000
dT = 0.001 ns, dT_out = 0.01 ns

For the 1th magnet:
Optimum out path = 0.80787 m
Magnet guide = 0.2574 m
Rho = 0.088379 m
Drift time of the first electron in the bunch : 7.629 ns
Drift time of the last electron in the bunch : 7.48 ns
Max energy = 0.47579 MeV
RMS = 0.0038689 MeV

For the 2th magnet:
Optimum out path = 1.0833 m
Magnet guide = 0.37765 m
Rho = 0.098898 m
Drift time of the first electron in the bunch : 5.589 ns
Drift time of the last electron in the bunch : 5.605 ns
Max energy = 0.89169 MeV
RMS = 0.0068848 MeV

For the 3th magnet:
Optimum out path = 1.1705 m
Magnet guide = 0.41573 m
Rho = 0.10223 m
Drift time of the first electron in the bunch : 5.311 ns
Drift time of the last electron in the bunch : 5.318 ns
Max energy = 1.298 MeV
RMS = 0.0096887 MeV

Electron with the most energy : 629) 1.6999 MeV,        RMS of bunch : 0.012318 MeV

Total steps calculated : 12455378454
Simulation finished in : 631136046 us     ( 631.1 s )
```

Figure B.2. ϕ_{lag} & ρ & L optimization at $P = 30$KW, $R_1 = 0.188$m, $R_2 = 0.753$m, $t_g = 0.8$ns, $E_{in} = 40$KeV.



(a) $dt = 10^{-2}$ns, $\Delta E = 1.047$MeV



(b) $dt = 10^{-4}$ns, $\Delta E = 0.006$MeV

Figure B.3. Energy gain of 1MeV bunch in **B**=0.1T using RK4-2.

```
#            Rhodotron Simulation Configuration File
# ============================================================
#    M.Furkan Er                                    22/09/2022
# ============================================================
#
# emax = Maximum electric field strength (MV/m)
# ein = Energy of electrons coming out of the gun (MeV)
# einstd = Standard deviation of energy of electrons coming out of the gun (MeV)
# targeten = Max energy on the output gif (MeV)
# freq = Frequency of the RF field (MHz)
# phaselag = phase lag of the first electrons (degree)
# starttime = time to start firing the gun (ns)
# endtime = ns to run the simulation (ns)
# dt = time interval to do the calculations (ns)
# guntime = how long a gun pulse is (ns)
# gunperiod = time between two gun pulses (ns)
# enum = number of electrons to simulate in a bunch
# bunchnum = number of times the gun fires
# r1 = radius of the inner cylinder (m)
# r2 = radius of the outer cylinder (m)
# epath = path to store the electric field data
# bpath = path to store the magnetic field data
# cpath = path to store the settings
# ppath = path to store electron data
# multh = enable or disable multitheading
# thcount = set the maximum thread to be used
# magrotation = degrees of rotation to enter each magnet
# addmagnet = takes 3 input. (B , R, < Radial distance of center >)
# output = output file name


# E FIELD CONFIGURATION
emax=1.170
freq=107.3
phaselag=10.0
r1=0.1840
r2=0.7380

# B FIELD CONFIGURATION
magrotation=5.0

# GUN CONFIGURATION
einmean=0.040
einstd=0.0000
targeten=2.0
guntime=1.0
gunperiod=9.3
enum=50
bunchnum=1

# SIM CONFIGURATION
starttime=0
endtime=10
dt=0.0000010000
epath=xy/rf.dat
bpath=xy/magnet.dat
cpath=xy/settings.dat
ppath=xy/paths/
multh=1
thcount=10
```

Figure B.4. An example *config.in* file.

```
-- Simulation Configuration --
Emax : 0.96     MV/m
Freq : 107.5    MHz
Phase Lag : -5  degree
EndTime : 100   ns
dT : 0.0001     ns
guntime : 0.8   ns
gunperiod : 9.3 ns
enum : 100
bunchnum : 2
R1 : 0.188241   m
R2 : 0.752967   m
Magnet count :  5
Ein : 0.04      MeV
TargetE : 2.5   MeV
--------------------------------

             ...Simulation is running...
V_____________________________________________________V
[####################################                 ]
```

Figure B.5. Example of console output while simulation is running.

APPENDIX C: Data and Graphs

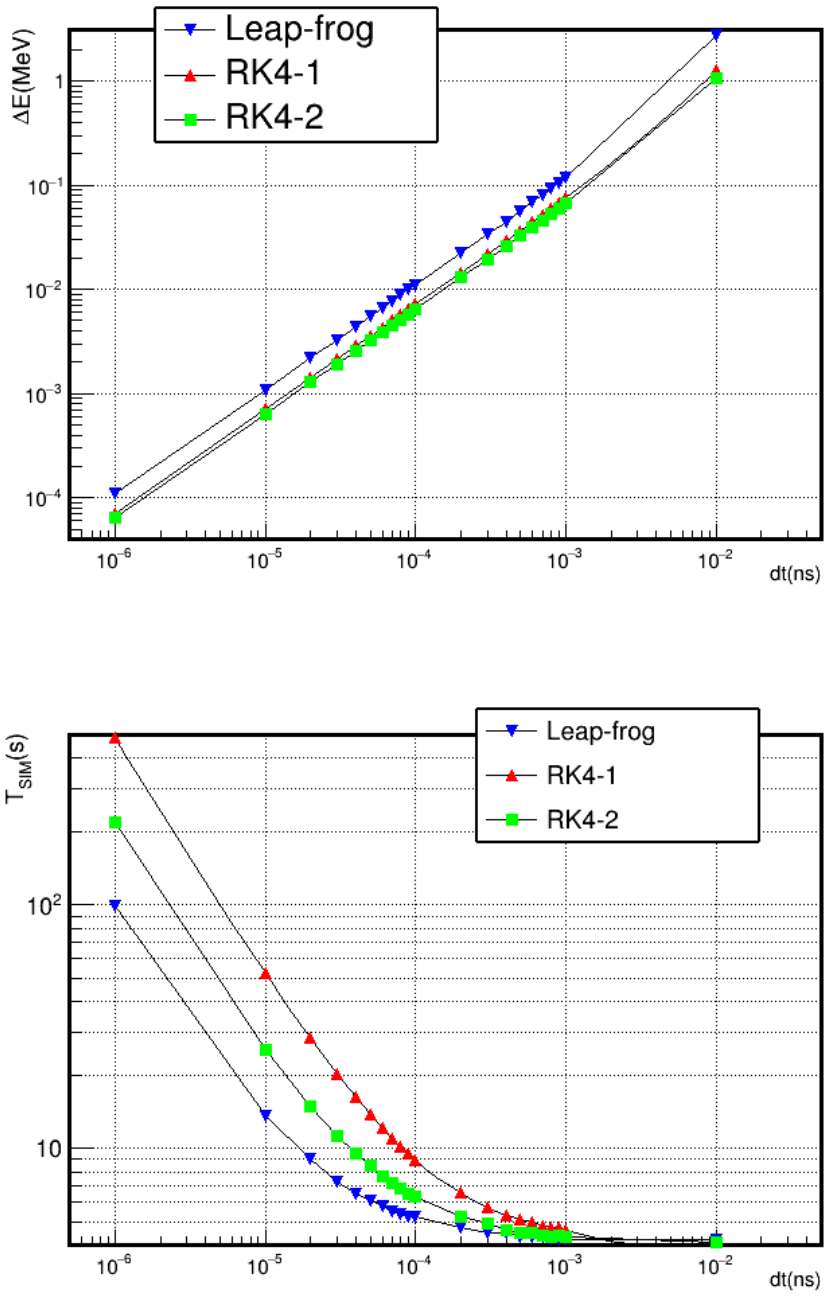


Figure C.1. Comparing Leap-frog, RK4-1, RK4-2 performance on $\mathbf{e}^-$ - $\vec{\mathbf{B}}$ interaction $E_{in} = 1\text{MeV}$, $\mathbf{B}$=0.1T, $t_{end} = 5\text{ns}$.

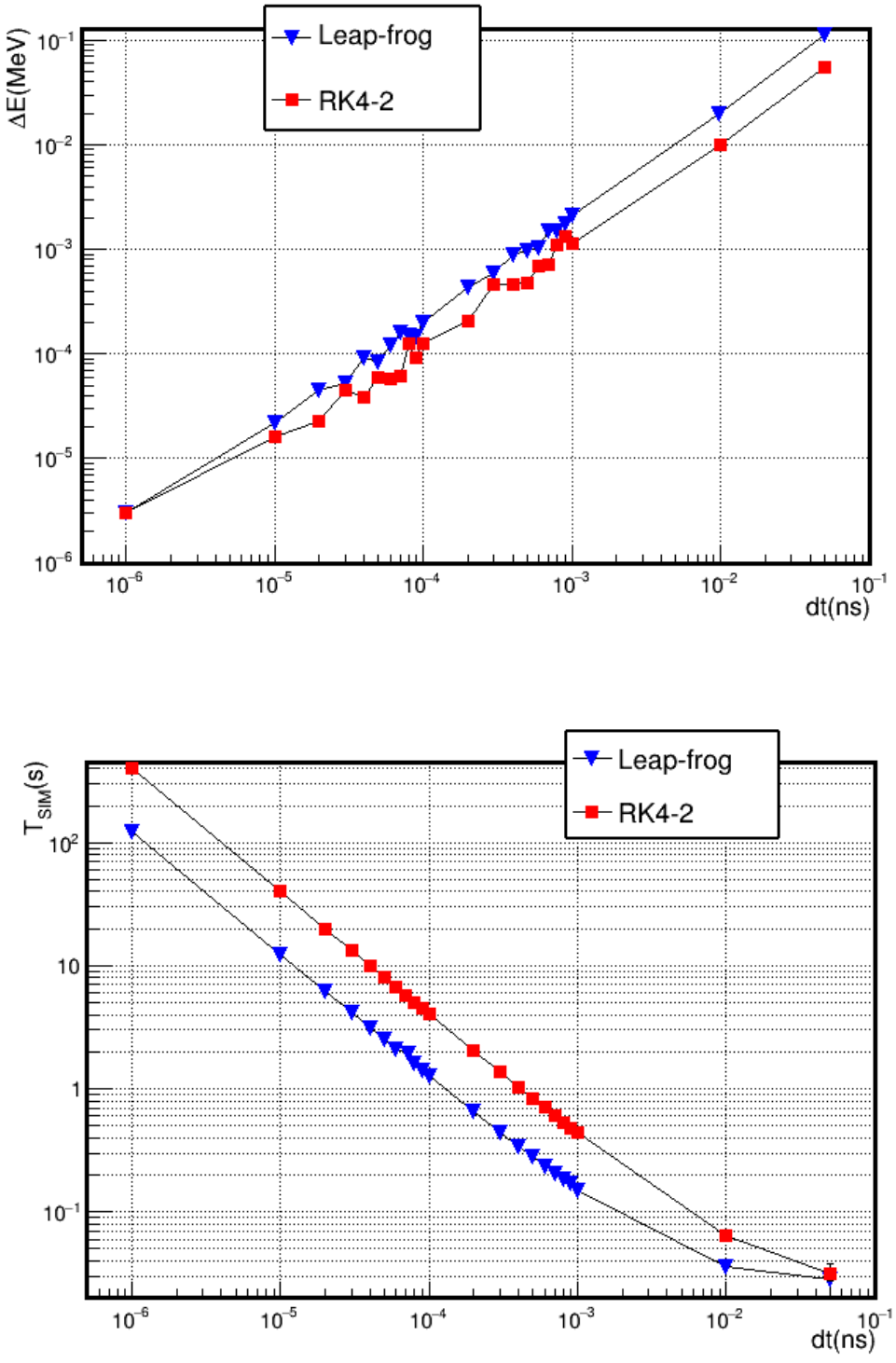


Figure C.2. Comparing Leap-frog, RK4-2 performance on $\mathbf{e}^-$ - $\vec{\mathbf{E}}$ interaction $E_{in} = 1\text{MeV}$, $\mathbf{V}_{\|}$=4MV, $t_{end} = 6\text{ns}$.

Table C.1. *Leap-frog* data on E_{in} = 1MeV, **B**=0.1T, t_{end} = 5ns.

dt(ns)	ΔE_{avg}(MeV)	μT_{sim}(s)	σT_{sim}(s)
1e-02	2.783228	0.034660	0.011537
1e-03	0.117124	0.115775	0.001071
9e-04	0.104552	0.128983	0.001820
8e-04	0.092252	0.143251	0.002585
7e-04	0.080144	0.158808	0.003228
6e-04	0.068258	0.182359	0.002426
5e-04	0.056554	0.215104	0.002807
4e-04	0.044931	0.262952	0.005119
3e-04	0.033467	0.341552	0.002610
2e-04	0.022158	0.501784	0.005709
1e-04	0.011006	0.973145	0.005849
9e-05	0.009899	1.084032	0.010985
8e-05	0.008792	1.216145	0.012486
7e-05	0.007688	1.387908	0.019031
6e-05	0.006586	1.604475	0.011775
5e-05	0.005485	1.926505	0.014535
4e-05	0.004384	2.395898	0.009702
3e-05	0.003286	3.178265	0.014099
2e-05	0.002189	4.740706	0.022709
1e-05	0.001094	9.441138	0.027266
1e-06	0.000109	93.888320	0.290820

Table C.2. *RK4-2* data on E_{in} = 1MeV, **B**=0.1T, t_{end} = 5ns.

dt(ns)	ΔE_{avg}(MeV)	μT_{sim}(s)	σT_{sim}(s)
1e-02	1.047130	0.048943	0.011642
1e-03	0.066912	0.239299	0.003483
9e-04	0.059899	0.268007	0.004530
8e-04	0.053028	0.296154	0.004146
7e-04	0.046183	0.333123	0.004259
6e-04	0.039452	0.384046	0.002458
5e-04	0.032734	0.456387	0.003888
4e-04	0.026072	0.563011	0.004803
3e-04	0.019474	0.742440	0.006169
2e-04	0.012926	1.103559	0.007649
1e-04	0.006437	2.178779	0.009733
9e-05	0.005791	2.411302	0.012266
8e-05	0.005145	2.704117	0.012683
7e-05	0.004500	3.078304	0.013281
6e-05	0.003856	3.589154	0.014472
5e-05	0.003212	4.297561	0.009546
4e-05	0.002568	5.369322	0.012127
3e-05	0.001925	7.136687	0.007845
2e-05	0.001283	10.679166	0.012126
1e-05	0.000641	21.325229	0.011661
1e-06	0.000064	212.824121	0.040967

Table C.3. *LF* data on $E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (-2.65616, 0, 0)$ MV/m, $t_{end} = 6\text{ns}$.

dt(ns)	ΔE_{avg}(MeV)	μT_{sim}(s)	σT_{sim}(s)
5.00E-02	0.114422	0.028398	0.009294
1.00E-02	0.019233	0.036245	0.002182
1.00E-03	0.002141	0.147979	0.00125
9.00E-04	0.001749	0.172161	0.00794
8.00E-04	0.001516	0.184629	0.004359
7.00E-04	0.00152	0.20445	0.001919
6.00E-04	0.001049	0.234947	0.004416
5.00E-04	0.000974	0.278938	0.010969
4.00E-04	0.0009	0.336847	0.003028
3.00E-04	0.000587	0.441771	0.005859
2.00E-04	0.000433	0.652066	0.006637
1.00E-04	0.0002	1.280059	0.016027
9.00E-05	0.000145	1.41908	0.013122
8.00E-05	0.000153	1.592492	0.011332
7.00E-05	0.000162	1.806561	0.009931
6.00E-05	0.000123	2.103571	0.011143
5.00E-05	0.000084	2.515006	0.01393
4.00E-05	0.000092	3.140471	0.014138
3.00E-05	0.000053	4.158715	0.009474
2.00E-05	0.000045	6.231734	0.015236
1.00E-05	0.000022	12.39871	0.039575
1.00E-06	0.000003	123.318	0.299559

Table C.4. *RK4-2* data on $E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (-2.65616, 0, 0)$ MV/m, $t_{end} = 6\text{ns}$.

dt(ns)	ΔE_{avg}(MeV)	μT_{sim}(s)	σT_{sim}(s)
5.00E-02	0.055355	0.031054	0.003281
1.00E-02	0.009868	0.063656	0.000893
1.00E-03	0.001141	0.435864	0.005332
9.00E-04	0.001325	0.478216	0.00595
8.00E-04	0.001112	0.533789	0.003897
7.00E-04	0.000705	0.60586	0.005441
6.00E-04	0.000687	0.703851	0.009447
5.00E-04	0.000474	0.840173	0.00913
4.00E-04	0.00046	1.039166	0.005874
3.00E-04	0.000456	1.378822	0.007995
2.00E-04	0.000207	2.045464	0.008499
1.00E-04	0.000127	4.052472	0.014711
9.00E-05	0.000091	4.500401	0.015404
8.00E-05	0.000124	5.048922	0.014392
7.00E-05	0.000062	5.773208	0.031443
6.00E-05	0.000057	6.726501	0.020609
5.00E-05	0.00006	8.045931	0.014246
4.00E-05	0.000039	10.04809	0.012756
3.00E-05	0.000045	13.39669	0.028057
2.00E-05	0.000023	20.03889	0.006436
1.00E-05	0.000016	40.0818	0.120627
1.00E-06	0.000003	400.2421	0.554172

Table C.5. *LF* data on $E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (0, -5.31232, 0)$ MV/m, $t_{end} = 6\text{ns}$.

dt(ns)	ΔE_{avg}(MeV)	μT_{sim}(s)	σT_{sim}(s)
5.00E-02	0.986005	0.030314	0.009209
1.00E-02	0.154314	0.036833	0.004068
1.00E-03	0.01476	0.131783	0.002352
9.00E-04	0.013618	0.150984	0.005618
8.00E-04	0.011863	0.161336	0.003117
7.00E-04	0.010109	0.179693	0.002658
6.00E-04	0.009277	0.205356	0.002912
5.00E-04	0.007527	0.242352	0.003093
4.00E-04	0.006084	0.294269	0.001551
3.00E-04	0.00449	0.383982	0.001711
2.00E-04	0.002897	0.563762	0.004214
1.00E-04	0.001458	1.101252	0.008064
9.00E-05	0.001376	1.229321	0.0089
8.00E-05	0.001232	1.381441	0.01057
7.00E-05	0.001027	1.574994	0.011933
6.00E-05	0.000853	1.833183	0.006065
5.00E-05	0.00074	2.184948	0.013416
4.00E-05	0.000596	2.72503	0.016356
3.00E-05	0.000467	3.617576	0.017581
2.00E-05	0.000308	5.394629	0.027885
1.00E-05	0.00015	10.75182	0.03301
1.00E-06	0.000017	106.9198	0.373175

Table C.6. *RK4-2* data on $E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (-2.65616, 0, 0)$ MV/m, $t_{end} = 6\text{ns}$.

dt(ns)	ΔE_{avg}(MeV)	μT_{sim}(s)	σT_{sim}(s)
5.00E-02	0.587777	0.034741	0.012735
1.00E-02	0.09943	0.059945	0.001318
1.00E-03	0.009987	0.38199	0.005941
9.00E-04	0.007946	0.417119	0.005683
8.00E-04	0.007999	0.466052	0.006398
7.00E-04	0.006732	0.525791	0.003782
6.00E-04	0.005541	0.612876	0.007114
5.00E-04	0.005022	0.732091	0.012129
4.00E-04	0.004031	0.906845	0.012156
3.00E-04	0.002779	1.199367	0.014536
2.00E-04	0.001897	1.782653	0.016993
1.00E-04	0.000895	3.52464	0.029778
9.00E-05	0.000858	3.916115	0.041066
8.00E-05	0.000731	4.409459	0.040828
7.00E-05	0.00066	5.012686	0.042727
6.00E-05	0.000561	5.851793	0.038488
5.00E-05	0.000471	7.002315	0.046152
4.00E-05	0.000363	8.747954	0.063936
3.00E-05	0.000264	11.64305	0.091404
2.00E-05	0.000184	17.44533	0.123198
1.00E-05	0.000093	34.93332	0.391012
1.00E-06	0.000011	348.494	3.237203

APPENDIX D: Supporting Figures

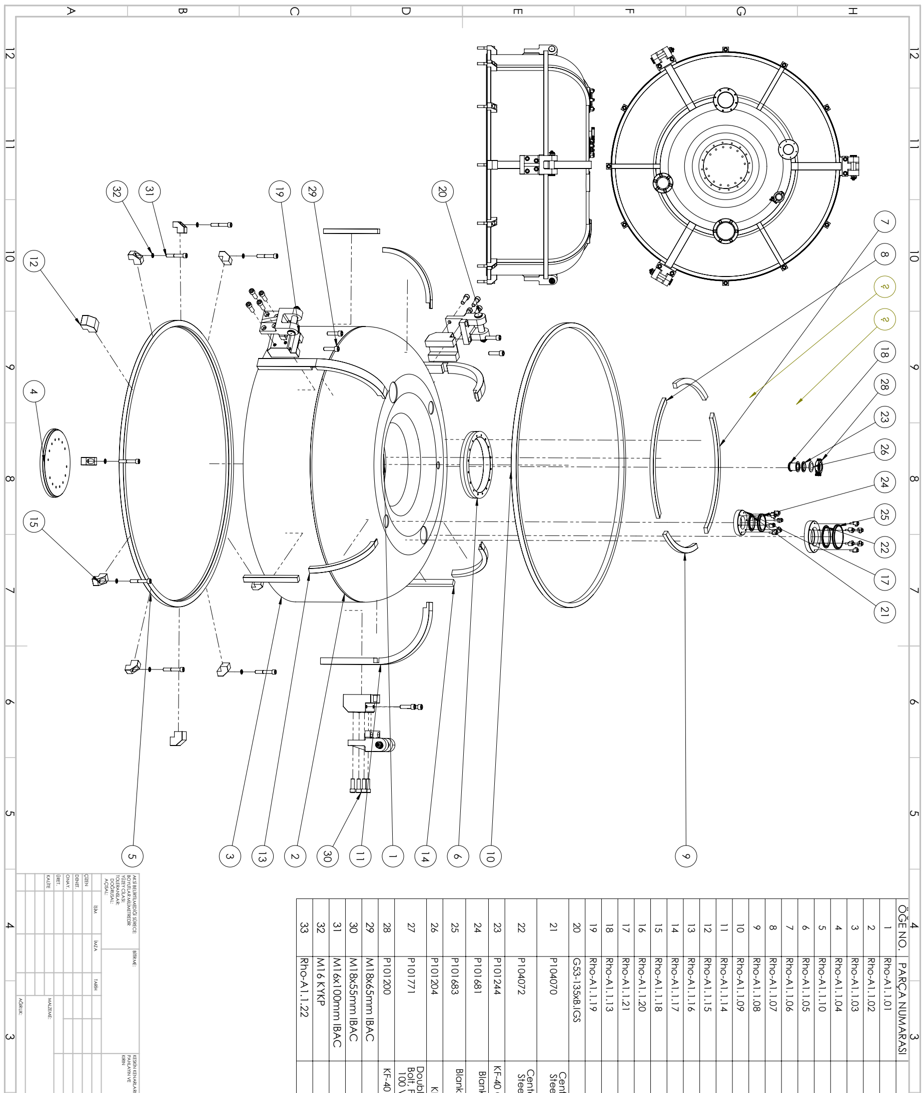


Figure D.1. Technical drawing of upper part of the cavity [13].

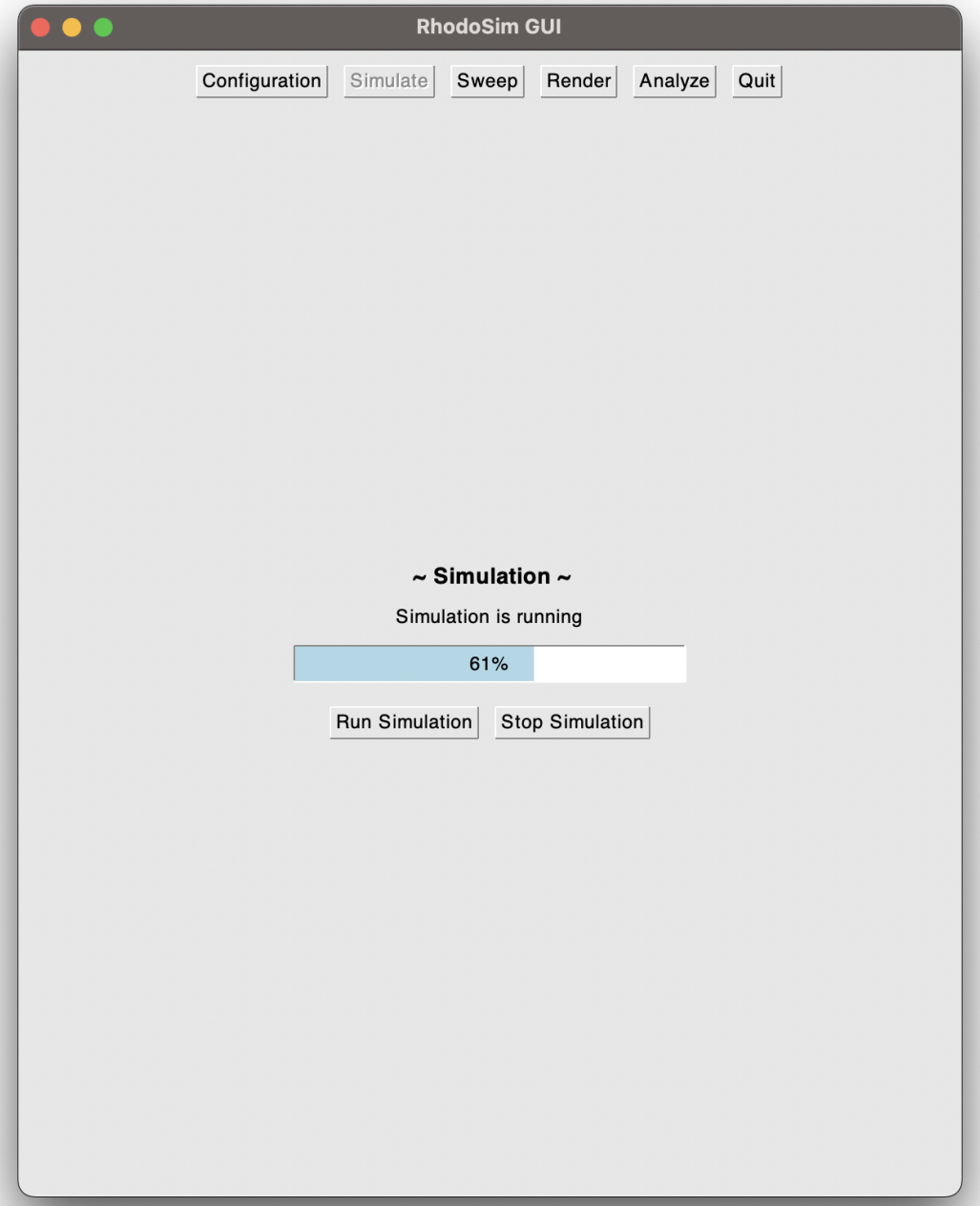


Figure D.2. Simulate frame of *GUI*.

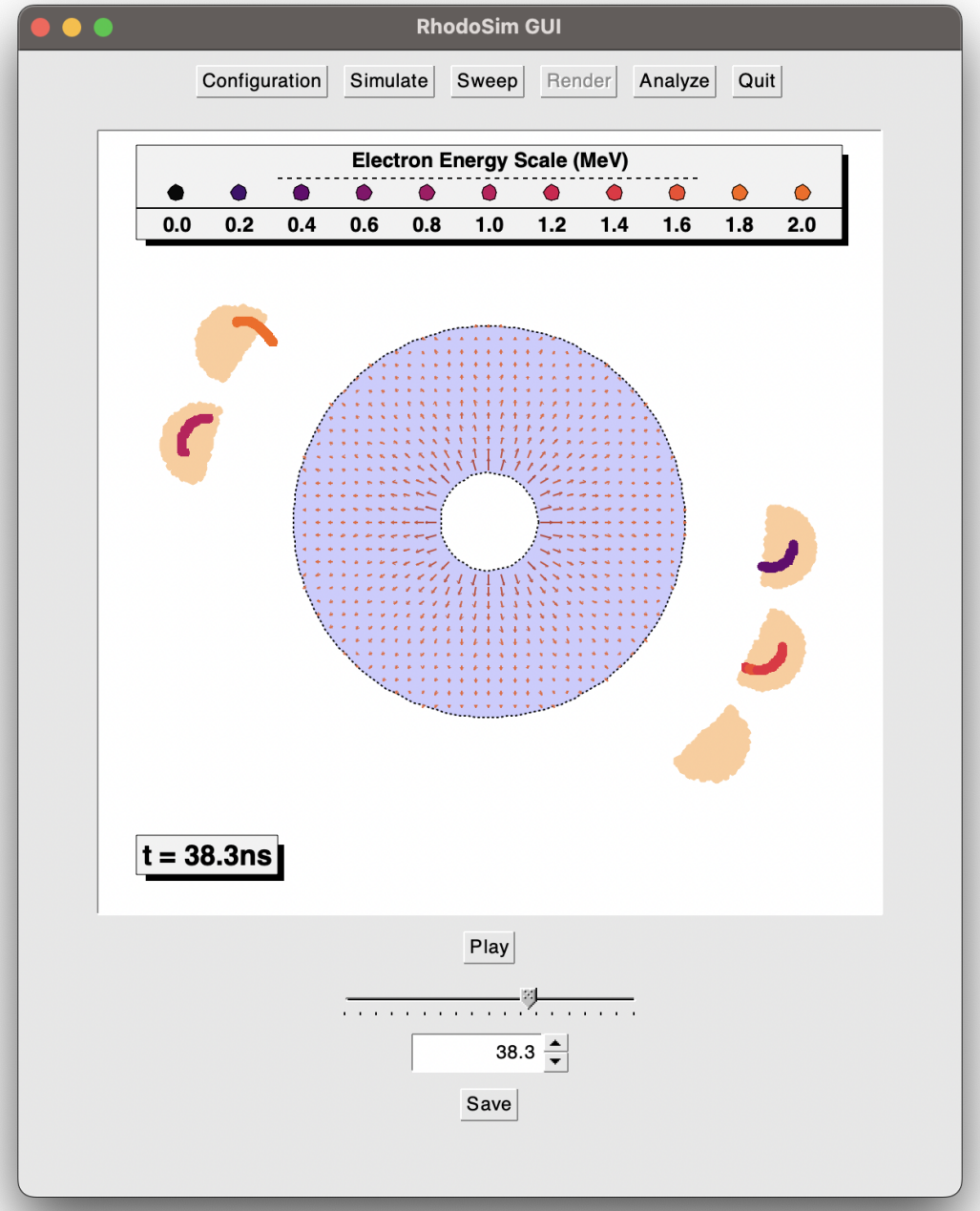


Figure D.3. Render frame of *GUI*.

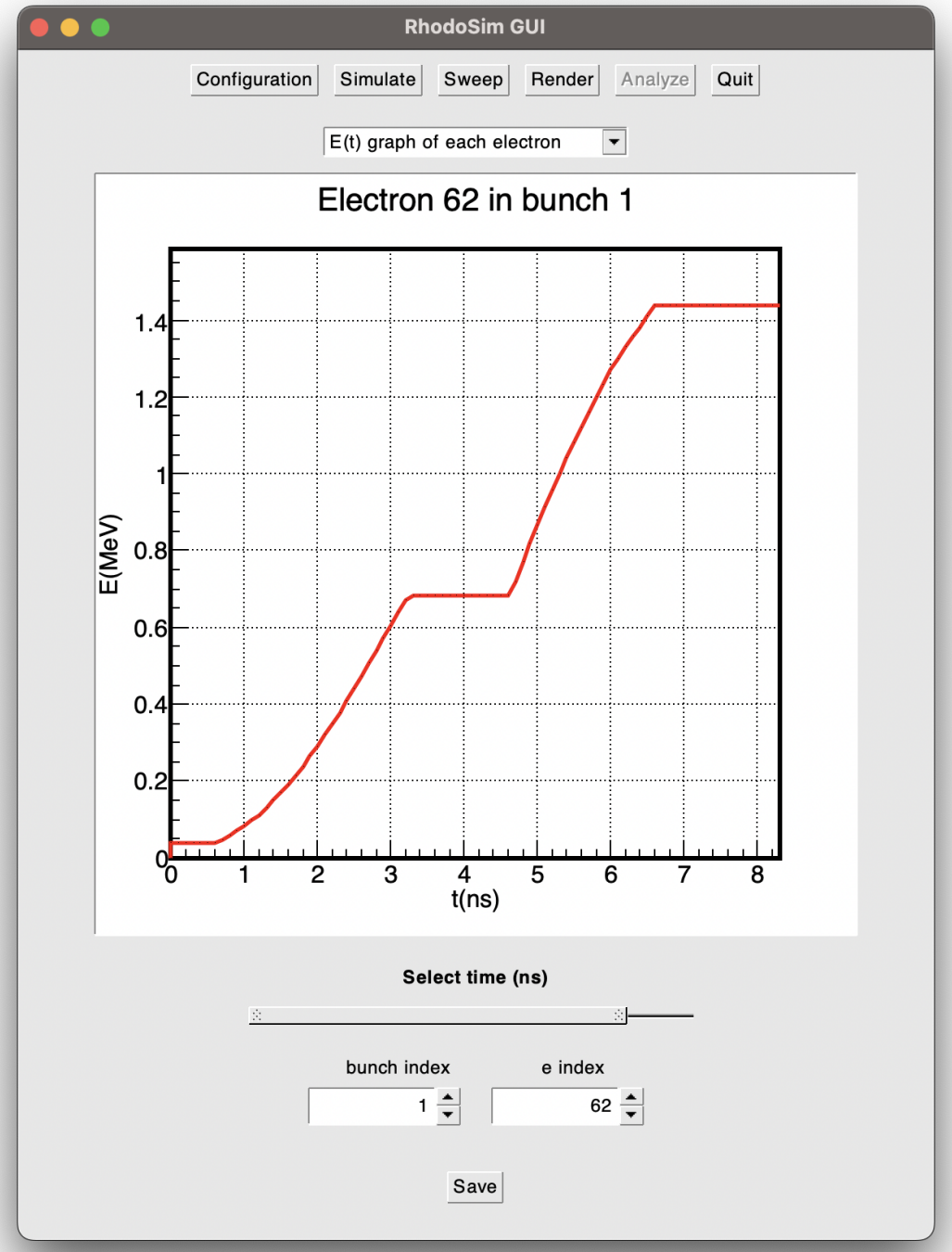


Figure D.4. Analyze frame of *GUI* E(t).

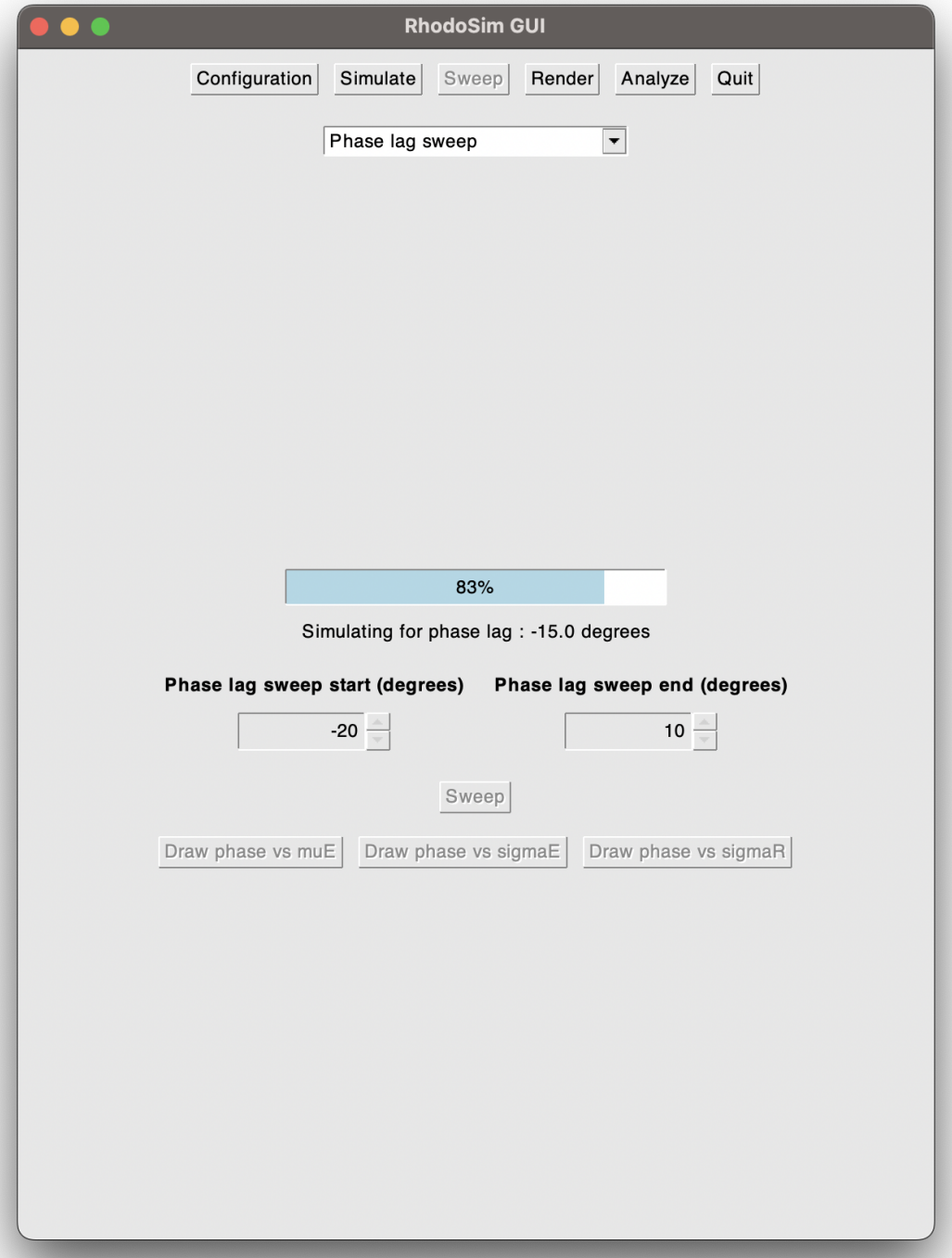


Figure D.5. Sweep frame of *GUI* ϕ_{lag} sweep running.

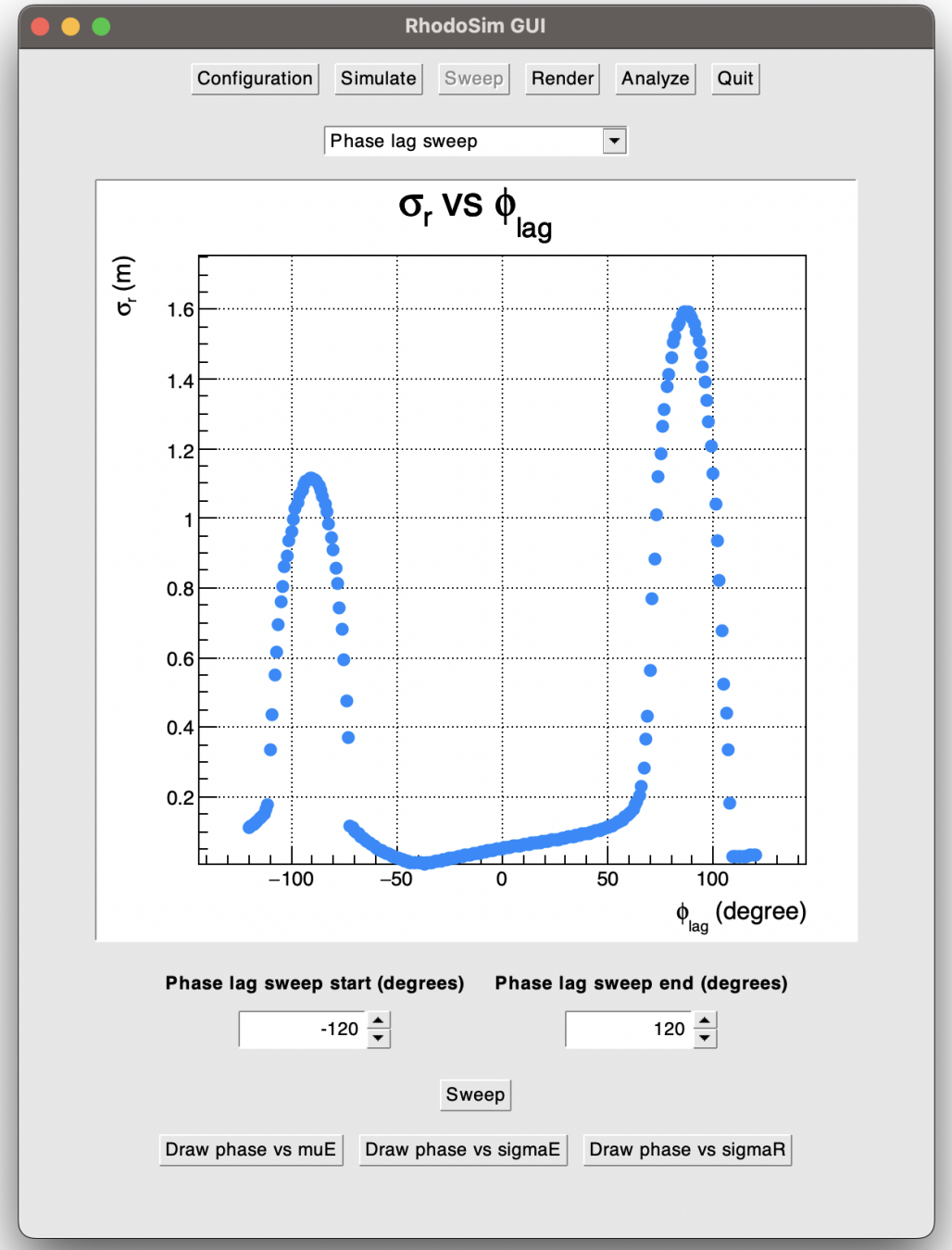


Figure D.6. Sweep frame of *GUI* ϕ_{lag} sweep σR result.