UNIVERSITY OF LJUBLJANA
Faculty of Electrical Engineering

Jordi Sans

INTERACTIVE 3D SIMULATIONS

Erasmus exchange project work

Supervisor: Assoc. prof. dr. Andrej Košir

Ljubljana, 2009

Table of content

1. Introduction

“Simulation” is a word that might be familiar to everybody. Its meaning is so expanded in all sectors (medicine, education, biology, engineering, psychology...), that the introduction of this paperwork will mainly focus only on computer simulations subject.

To better understand what a computer simulation is, the most suitable and simple definitions are below:

“The technique of representing the real world by a computer program”; "a simulation should imitate the internal processes and not merely the results of the thing being simulated";

“is a technique to perform tests using a model written in software”

Why do we use computer simulations? The answer to this question may seem very long, but the main reasons for using simulations are easy to find, although they depend on what area we are in:

Business simulations: Modern business has to stay competitive by keeping development and training costs and times to a minimum, while still keeping high levels of quality for both.
Modeling and simulation of systems can provide solutions for product development and personnel training without the costs usually associated with these. In other words: computer simulations save time and money, and they are as reliable as real world tests.

Education simulations: They provide the students with one *intermediate space*, which joins the reality with the models or theories. In addition, simulations allow interactive manipulation of the models, that will facilitate the acquirement of knowledge of the students.

Obviously the subject of this work is to program and to study an example of “educational simulation”.

More specifically, the simulation we work on is an *interactive simulation*, which intends to provide the students with a tool they can play with. The students can better understand the mathematical model used to explain the phenomena under study, because they can check and observe, in an interactive way, the reality the model represents.

Our Applet simulation is drawn on a 2D projection of a 3D interactive graph representing two plane waves, one electric and the other magnetic, with z axe as their direction of propagation.

It is an interactive Applet because the users can change the values of the wave equations through sliders, buttons and combo boxes.

2. Existing technologies and their evaluation

If only the points of a bidimensional graph want to be displayed, then the easiest languages to be used are Java and C++.

What language should be used for 3D programming?

There are many useful program languages to develop an interactive 3D simulation program. Some of them are listed below, as well as the reasons why Java language was chosen to do this paper work, which are explained in the following paragraphs:

Flash

The most serious competitor of Java on the internet is Flash. It allows web developers to create interactive content, such as animations, animated menus, movies, games...

Flash is based on vector graphics, which means that Flash animations can be rescaled without losing the image quality. Flash animations can be embedded in HTML pages as menus, movies or web site layouts.

There are also disadvantages of Flash, depending on its implementation in a web site structure. For example, it is recommended to avoid its excessive usage, especially in web sites introductions. It is not optimized for search engine indexing, such as Google or Yahoo.

There are obvious differences if we compare Java with Flash:

Main differing points between java and flash languages	
JAVA	**FLASH**
Universally available and usable	Depends on Macromedia's Flash plug-in propagation
Java Applets are typically larger	Smaller applets, faster loads/downloads
Integrates easily with other web pages elements (i.e. HTML web sites)	Less so
Pre-defined GUI, such as AWT or Swing	Vector-graphics "look", larger possibilities for web designers
Scalable to any processor-driven platform (computers, mobile phones...)	Less so

Figure 2.1 Comparisons between Java and Flash.

It appears that Java and Flash are designed to work together; Java scripts can activate and communicate with Flash files. Therefore, nowadays both languages are evolving, Java can be found in almost every existing device, while Flash is showing up more and more in the internet (i.e. movies sites).

2.1. Java Script

JavaScript is, in some way, similar to Java. Both are object-oriented programming languages. They mainly differ in two basic points:

1. Java can stand on its own, whereas JavaScript is text put in a browser that reads and plays it. As a result of it, JavaScript must be included in an HTML document so it works. On the contrary, a Java applet is considered a program itself that must be compiled in the JVM before being added in a website.

2. The second main difference is how the languages are presented to the final user. A Java applet is handed out so the machines that receive it see the program as an already set one, which cannot be changed, nothing can be either added or removed from it. On the contrary, JavaScript is text-based, it can be modified and run as many times as necessary and no compilation process is needed.

   To summarize, Java is much more powerful than JavaScript, because it is conceived to run in a general field, whereas JavaScript is meant to run in web pages.

On the other hand, these two languages are the most frequently used by website designers, as they can give outstanding designs and dynamism to the web pages.

Basically, FLASH was not chosen to be the language because it is more focused to design the user interface, and it is not very suitable for simulating mathematical models. Moreover, a plugin for FLASH is needed.

2.2. Visual Basic

It was one of the first languages to take advantage of GUI interfaces, so along with Java they allow the programmer to create an application much faster than other languages. VB can provide web programs as Java does, so both can be deployed in many modes (stand-alone, components of other systems...). VB is widely used in Microsoft Office programs (macros in Word, Excel...) and in Windows OS.

In conclusion, although it is difficult to affirm and it has not been carefully proved in this work, it makes sense to use VB for PC based applications, where Windows dominates, and to use Java for a browser interface. Java may be better in an internet environment due to Swing classes that makes GUI operations very fast.

2.3. C++ and JAVA

It was partially modeled after C and C++, and offers all their features but avoids the worst and most confusing parts of them.

As stated in Java language white paper by Sun Microsystems: "Java is simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic."

2.3.1. JAVA Overview and features

It is **simple**. It is similar to C++ although it is much simpler than it because Java uses automatic memory allocation and garbage collector while C++ requires the programmer to allocate memory and to collect garbage. It means that using Java, the programmer does not have to bother to free the dynamic memory. Also, Java eliminates the use of pointers, which caused a lot of troubles to programmers (parameters are passed only by value, not by reference). Besides, Java code can be easily read and written, and the language and the class libraries are all integrated together. Though it is true that Java standard library is considerably large in comparison with C++ standard libraries.

Java, as well as C++, is **object-oriented** because programming in Java is focused on creating objects, manipulating objects, and making objects work together (where everything can be considered as an object).

An object has properties and behaviors. Properties are described by using data, and behaviors are described by using methods. Objects are defined by using classes in Java. A class is like a template for the objects. The process of creating an object class is called instantiation. Java consists of one or more classes that are arranged in a treelike hierarchy, so that a child class is able to inherit properties and behaviors from its parent class. An extensive set of pre-defined classes, grouped in packages that can be used in programs are found in JAVA, for instance J2SE development kid (JDK library), provided freely by Sun Microsystems.
Creation of objects:

1. *Declaration:* name the object.
2. *Instantiation:* to asign memory to the object.
3. *Initialization:* to provide the object with an initial value.

Object myObject = new Object();

Figure 2.2 example of an object creation where the class is Object.

Object-oriented programming provides greater flexibility, modularity and reusability. Each object can stand alone, but the sum of them makes up the whole.

Java provides the programmers with libraries and tools so the programs can be **distributed**; that is they can be run from several computers in a network while interacting with each other. So Java has been built to interconnect with TCP/IP protocols and to interact with http or ftp.

Java is an **interpreted language**. Due to it, an interpreter (Java Virtual Machine) is needed in order to run JAVA programs. After the code is compiled, a *bytecode* (or pseudo-code) is created, which is not understandable by any machine.
Therefore, *bytecode* is machine independent and it can only be run on any machine that has a Java interpreter (JVM). Normally, a compiler will translate a high-level language program to machine code and the code is able to run only on the native machine. If the program is run on other machines, the program has to be recompiled on the native machine. For example, if you compile a C++ program in Windows, the executable code that is generated by the compiler can only be run on a Windows

platform. With Java, the program needs only to be compiled once, and the *bytecode* generated by the Java compiler can run on any platform (with a JVM installed).

Java code gets through a lot of checkings before being executed in a machine. Java is one of the first programming languages to consider **security** as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind. The compiler, interpreter, and Java-compatible browsers all contain several levels of security measures that are designed to reduce the risk of security compromise, loss of data and program integrity, and damage to system users.

Java is **Robust**. Robust means reliable and no programming language can really assure reliability. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages. For instance, Java does not support pointers, which eliminates the possibility of overwriting memory and corrupting data. Java has a runtime exception-handling feature to provide programming support for robustness, and can catch and respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs.

One advantage of Java, as it was already mention above, is that its programs can run on any platform without having to be recompiled. This is one positive aspect of **portability**. It goes on even further to ensure that there are no platform-specific features on the Java language specification. For example, in some languages, the integer bit size varies on different platforms. In Java, the size of the integer is the same on every platform, 32 bits with two's complement. Having a fixed size for numbers makes Java programs portable. The Java environment itself is portable to new hardware and operating systems, and actually the Java compiler itself is written in Java.

Java is **Multithreaded**, which means a program is capable of performing several tasks simultaneously within a program. For instance, downloading an .mp3 file while playing it would be considered multithreading. In Java, multithreaded programming has been integrated into it, whereas in other languages, operating system-specific procedures have to be called in order to enable multithreading. Multithreading is especially useful in graphical user interface (GUI) and network programming. In GUI programming, many things can occur at the same time. In network programming (even though it is not the goal of this work), a server can serve multiple clients at the same time. Multithreading is a necessity in visual and network programming.

The most remarkable feature of Java is that it is **architecture neutral**. Architecture neutral means that it is platform independent. A Java program can be run on any platform with a Java Virtual Machine (run-time). Many operating system companies have adopted the Java Virtual Machine, and soon Java will be able to run on all machines.. Java applets can be run from Web browsers. Stand-alone Java applications can also be run directly from operating systems using a Java interpreter. Nowadays, software vendors usually develop multiple versions of the same product so that it can run on different platforms, such as Windows, Macintosh, Linux.... Using Java, the developers need only to write one version, and this one version will be able to run on all of the platforms.

Java is Dynamic. The Java programming language was designed to adapt to an evolving environment. New methods and properties can be added freely in a class without affecting their clients. Also, Java is able to load classes as needed at runtime.

2.4. Disadvantages of JAVA

The main disadvantage of Java is **speed**.

Although Java's ability for producing portable, architecturally neutral code is desirable, the method used to create this code is inefficient. As mentioned above, once Java code is compiled into byte code, an interpreter called a Java Virtual Machine, specifically designed for computer architecture, runs the program. Why is it a problem? "Java, being an interpreted system, is currently an order of magnitude slower than C. An interpreter must first translate the Java binary code into the equivalent microprocessor instruction. Obviously, this translation takes some amount of time and, no matter how small a length of time this is, it is for sure slower than performing the same operation in machine code. So we can affirm that Java still can not compete with natively compiled C++ code.

Check point [1] for further information about JAVA pros and cons.

2.5. Chosen language

In conclusion, Java language has been chosen in this work to be the language to develop the simulation applet. This language removes the unnecessary complexities of C++ (Java is less hard to learn than C++). It does not have the same high performance than C++ does, but these powerful features and complexities can sometimes lead to many errors. In addition Java can be run in any machine independently of the OS, as long as it is equipped with JVM (Java Virtual Machine), which is free cost.

A java applet runs independent of HTML files that call it. A java applet can be compiled in user-friendly environments such as Eclipse or Netbeans.

Java has many advantages for an internet language, mainly the robustness, neutrality, portability and security derived from the use of *bytecode,* which even if they slow the applications speed (they need to be compiled and interpreted), they solve many other issues.

To sum up why Java language was chosen among all the others, some of its main features are summarized below:

Java main features & Advantages
Simple language to learn (syntax similar to C++ but no pointers)
Object-oriented language
Organizing of classes into packages (Java.swing, Java.applet)
Easy management of the memory (garbage collector)
Scalable to any processor-driven platform (computers, mobile phones...) Less so
Platform independent (Windows, Linux…)
Open-source
A lot of API allowing for instance 3D programming and Graphics

Figure 2.3 Main Java advantages

3. Implementation issues

In this chapter, there are going to be described the major issues that came up when implementing the simulation applet and how they were solved or how the program deals with the problem.

3.1. Video refreshment

One of the major encountered issues concerned the video refreshment. A previous simulation applet was developed. It consisted in drawing a TEM wave, where the drawn graph was supposed to be refreshed every time a slider changed its value.
It turned out that the wave was not refreshed until a whole period of it was drawn.

The structure of the program did not follow the MVC concept, there was only one class containing all the code. Furthermore, all the computed points to plot a period of the wave where held in a buffer before being painted. It is called “double-buffering” technique and was used on the program to avoid the flickering while painting.

“Double-buffering” creates off-screen images. It draws to the an off-screen image using the image's graphics object, then, in one step, calls drawImage method using the target window's graphics object and the off-screen image.

As a result of it, when user slides the scrollbar, the graph is not refreshed until a whole period of the wave is completely painted.

The main issue due to the lack of MVC structure was that painting method held the processor. That is, when user changed a slider value, an event was thrown, but it was never caught since the processor was busy plotting continuously the wave.

This applet was eventually abandoned and, starting from the beginning, simulation template was programmed.

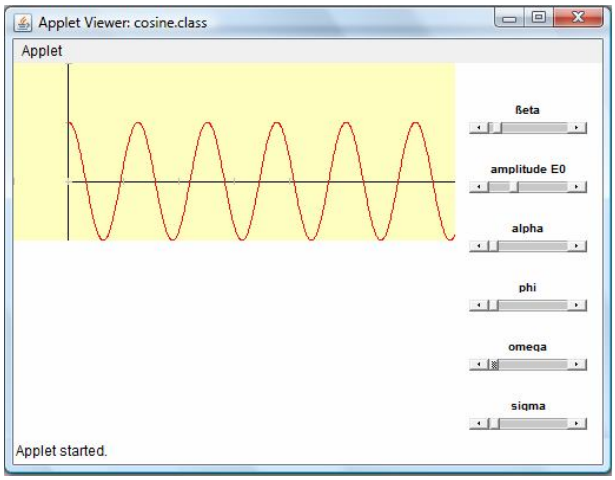


Figure 3.1 TEM wave, where no MVC concept is applied. It is drawn using AWT toolbox.

SimulationTemplate applet uses MVC concept and **runs a thread**, mainly in order to make sure the processor does not get stuck in a single duty. The key factor of MVC

lays on the use of the java Event listeners. These listeners keep the classes in touch, and when an event is thrown, it is notified immediately through the listeners to the appropriate class.

3.2. Hot Equation

There are some issues regarding the hot equations. These classes where somehow smoothly modified and added to the whole program. However further changes could be done to optimize them. As a matter of fact, the classes with regard to Hot Equations are too large. There are some methods that could be removed but have not been yet, because it may produce unexpected errors. They stay there as long as they do not affect the Hot Equation. In fact, when an equation is added using the HotEqn, the loading time of the applet in the JVM slightly increases.

The procedure of adding an equation must be thoroughly done. Although the steps to follow are explained on chapter 7.4, what is wanted in this chapter is to remark an open issue on the SimulationTemplate applet.

Currently, **only one equation can be added** on the applet using hot equations LaTeX writing.

The problem lies in the way of getting the LaTeX image from the HotEqn class. It does not recognize which string equation is inputted in SimulationTemplate, so simply creates one image from the first inputted string equation and the other equations are omitted.

3.3. Structure of MVC

It is not considered to be an issue, but SimulationTemplate applet does not follow strictly the MVC architecture. It basically has to do with the ControlUI class, which does not listen to the ViewUI. Instead of it, it sends the information directly to ViewNum01 class. There is no need to set listeners between ControlUI and ViewNum01, either. Everything that involves changes in the view is handled between the ControlUI and the ModelUI classes, and it is this last one which is in charge of notifying these view changes to ViewNum01 class.

Furthermore, it is true that the ViewNum01 class computes some equations, with regard to the 3D to 2D projections and to the units' conversion into pixels. H2.3.1r, these calculations were not considered to be the Model class duty, since all of them are strictly connected only to the view system.

Check the structure of the class listeners in figure 3.2.
This issue is also treated in chapter 5.1.4.

3.4. Unicode

Unicode (Universal Code) is a computing industry standard allowing computers to represent and manipulate text expressed in most of the world's writing systems.

Using Unicode was necessary to input Slovenian specific characters, such as “č” or “š” on the *properties files.*

The correspondence between characters and Unicode is found in point [10].

3.5. Model-view-controller (MVC) concept

In general terms, it is the architectural pattern used by programmers to implement their applications.
It was first introduced by “Smalltalk” language. Nowadays it is widely used in Web-based application frameworks.

The basic idea of MVC is to partition the interactive software, so the user interface is separated into a View, and the mathematical equations into a Model. To make it work out, a Controller is needed to response the user requests on the Model or the View. Thus, the Controller interacts with both the Model and the View.

Normally, the basic structure of MVC shows three basic bodies:

- **Model** – Computes and contains the data of the application, along with the logic that defines how to change and access that data. That is, it is in touch with both the View and the Controller so it is continuously listening and notifying to them any change. If a change occurs, then the Model must re-compute new data using the incoming inputs and send it to which it may concern (most likely the View or the Controller).
- **View** – It is typically the user's interface. Presents the Model data on the screen. All the imaginable presenting possibilities are considered here, from GUI to sounds, and they depend on where the programming language is capable of getting up to. In other words, the View is how the user views the current state of the Model.
- **Controller** – It gathers the user, or other possible inputs, and provides feedback to the Model, normally changing some of the data in that Model. Can also be done on the other way, if some changes occur in the Model, the Controller is in charge of providing feedback to the View so these changes can be seen by the final user.

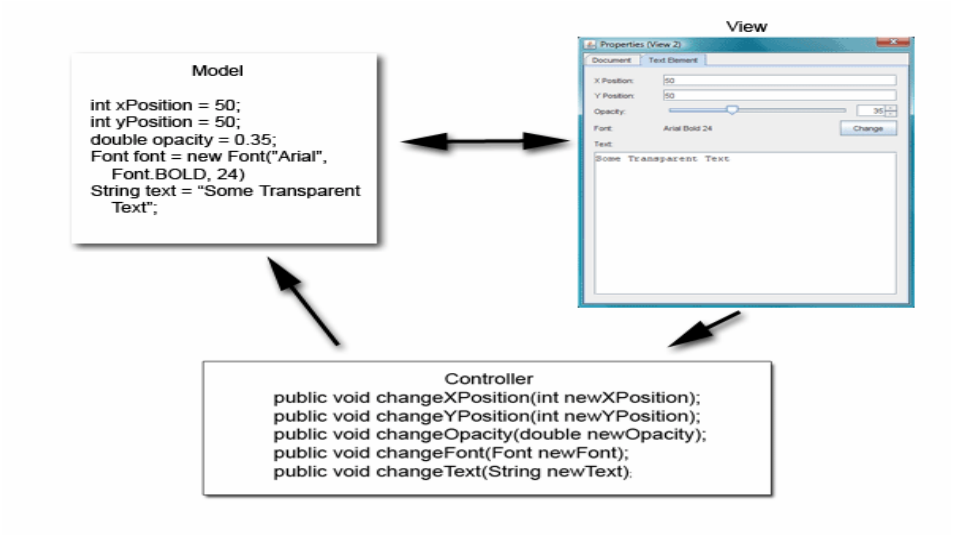


Figure 3.1 Detailed interactions between the three bodies. For further general information visit Sun's website: [3]

3.5.1. Why to use MVC structure?

It was designed to simplify the work of the software programmers. Moreover, it properly creates listeners of events, in a way that permits to implement separately the components of the program. Also, to keep the program well organized is fundamental to maintain it or to develop further versions of it.

It is time now to emphasize the applet SimulationTemplate. It is no matter at this point to get deep into the program structure, so it is only introduced the basic diagram of its MVC structure:

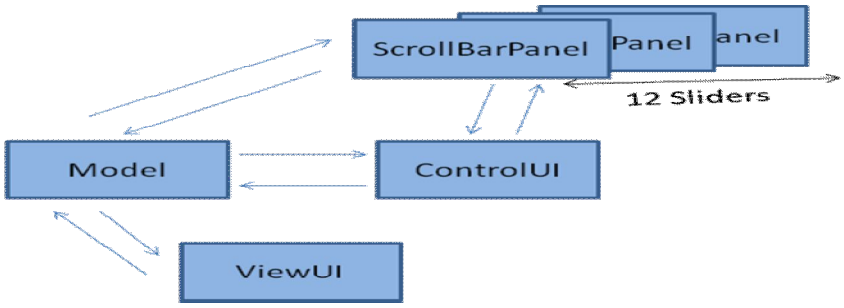


Figure 3.2 MVC listener structure of SimulationTemplate2D

Simulation Template applet does not follow strictly the MVC pattern, but it is still a very valid working example. The figure above shows the Listeners added between the classes so they listen and notify to each other the events thrown.

4. Interactive Java applet structure and graphic

4.1. Introduction of Java Applets

Sun Microsystems defines Java applet as a '*Java programming language that can be included in an HTML page, much in the same way an image is included in a page'.*

There is one thing to take into account, though. Java applets can run in a web browser as long as it has a Java Virtual Machine (JVM), which understands the byte-code of a Java class and makes it ready for run. A Java Run-time Environment (JRE) is provided by *Sun* and includes a JVM and some other library files that run the application.

It is also interesting to know that the applets run in a web browser cannot be accessed or changed by the remote clients that run them. It happens because web browsers use a mechanism to safely execute non-checked code. It is called *sandboxing*.

Commonly the goal (not the only one) of applets is to be embedded in websites, and to add interactive applications to HTML pages. But they can also be run in stand-alone applications provided by *Sun*. For instance, the applet developed in this paper work was programmed and tested in Eclipse environment, which is an open-source toolset for Java development.

To create an applet class, it is needed to extend the basic class of Java: *Applet*. This way, all the necessary things in order to create an applet are inherited.

Import java.applet.Applet;

Public class Example extends Applet{}

4.1 Example of applet class creation

All applet extensions are subclasses of *java.applet.Applet class.* So this class must be imported in any applet that wants to be embedded in a Web page or viewed by a Java Applet Viewer, because the *Applet* class provides a standard interface between applets and their environment.

Even for the simplest applet, some methods must be used:

init()	It is called only once, when the applet is created.
start():	It is called every time the applet wants to be activated. It is called right after init() is called.
destroy():	It is called once, when the applet will not be used anymore.
stop():	It is called every time the applet wants to be stopped, for example to stop an animation.

4.2 Basic methods for applets

And methods to paint components:

paint(Graphics g)	It is called every time the drawing area must be refreshed. It cannot be directly called by the programmer; instead calls repaint or update methods. It is the only one which really paints the components.

update(Graphics g)	Removes the area where the application has been painted in and calls paint() method.
repaint()	It is called when changes on components want to be visualized.

4.3 JAVA painting methods

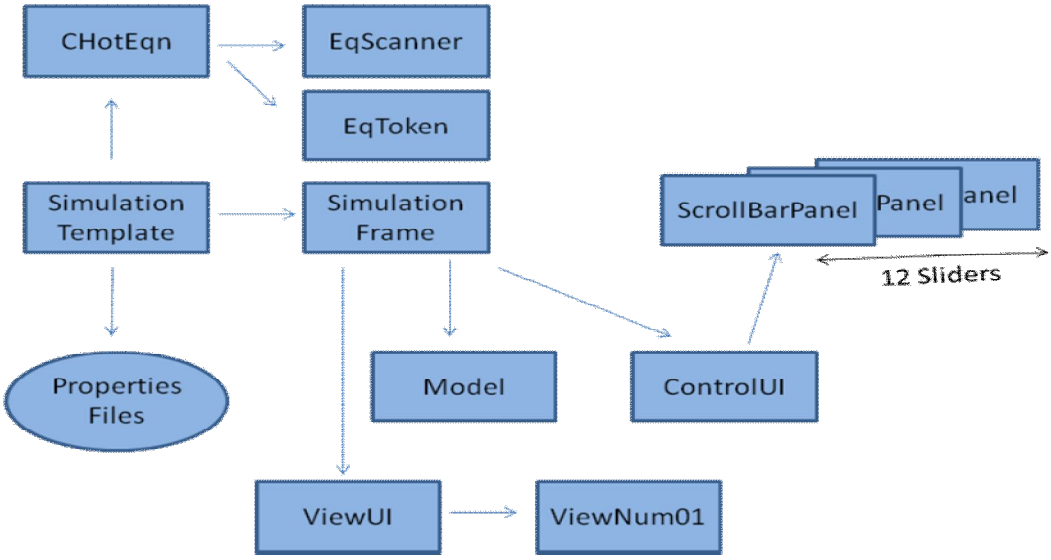
The difference between update() and repaint() methods is that update method includes an additional step in which the screen is first cleared, so it removes the previous drawn graphics before calling the paint method.

Check point [2] for further information on applets at Sun's website.

5. Basic description of structure and operation

It is going to be shortly described the role of each java class that appears on the simulation. All the connections between them are also going to be reviewed, as well as the chain of events that follows a user input.

The global scheme of the SimulationTemplate Applet is:



5.1 Figure showing the sequence of class creation

On the figure above it is observed the sequence of class creation, taking into account that SimulationTemplate class is the one that starts the program.

NOTE: Classes inherited from Java packages, such as AWT and SWING components have been omitted in this scheme (i.e. JPanel, JFrame, Applet…).

In order to better understand the relation between classes, they are now going to be reviewed all the steps the program follows when **an event is thrown** by the user.

5.1. Basic relations between classes

5.1.1. Starting the applet

It is first introduced the SimulationTemplate class. Its duty is to start up the whole simulation process. It extends the java Applet class.
By using init() method, it initiates the thread that runs continuously the program, and previously calls SimulationFrame class. User must notice that Model class is based on a **thread**, which runs continuously until exit button is pressed.

The main duties of SimulationTemplate are:

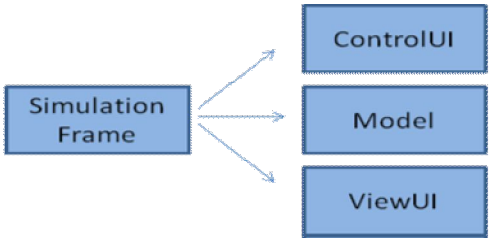
- To start up the thread that runs the applet.
- To stop the thread when exit button is pressed.
- To create “HotEqn” in order to input equations on the applet in LaTeX style.
- To load the properties files in order to choose the language of the applet.

5.1 SimulationTemplate creates SimulationFrame

5.1.2. Creation of MVC pattern

SimulationFrame adds all components and graphs of the applet. It extends JFrame class which belongs to java AWT toolkit. JFrame is the biggest container, so it is possible to add any component inside (JPanel, JButton,…). JPanel is a container itself, where it is also possible to add components and to distribute them using a layout manager. So basically its duties are:

- To distribute the simulation in the MVC structure, creating the Model, the View and the Controller.
- To call the Model class so the program gets started.
- To set the applet and the dimensions of JPanel components.
- To add Model and ControlUI listeners.



5.2 MVC concept

5.1.3. Model features

As said before, Model class runs consecutively due to a thread, until reset button is pressed. The aim of the Model is to calculate the necessary data so the view system can plot both the electric and magnetic waves. It is done by computing discrete values and by storing them into matrixes (called dataE and dataH). These matrixes are sent to ViewNum01.

The Model also sets the sliders default values, either when the applet is run for the first time or when reset button is pressed. At this point, it makes sense that if the Model sets the default values, it also has to read the new slider values entered by the user and it has to update the data matrixes (dataE and dataH), so the view system can repaint the graphs using the new slider values. Some of these values are dependent to each other, that is the reason why the Model has to recognize which slider has changed its value, and to apply the parameter dependencies if necessary.

To sum up, the main performances of Model class are listed below:

- It computes all mathematical equations of the E and H waves.
- Sets default values when the simulation is run for the first time or reset button is pressed.
- It eventually sends to the view system new user events (i.e. drawing style has changed). The ViewNum01 and the ControlUI do not have listeners to

be in touch to each other, so the Model must notify to the ViewNum01 some user changes.

There are some important vectors and matrixes declared in Model class and that must be taken into account to better understand how the applet works:

- The **modelParams** array is used to set the parameters of the scrollbars (also the default parameters). Using these parameters, which are stored in modelParams array, the electromagnetic equations are computed. The 3D points coming from this computing are stored in dataE and dataH matrixes.

- **dataE** and **dataH** are two matrixes containing the points used to draw the electric and the magnetic waves on the applet. They are eventually sent to ViewNum01 class, which will eventually draw the two waves using the information stored in the two matrixes. There are 3 points in each row (x_i, y_i, z_i) and there are as many rows as the number of points that want to be drawn.

  For further information about the Model class calculation check chapter 6.

Since it is not trivial to understand the Model class duties and its relations with all the other classes, there will be listed next all the user events that can occur on the simulation, and they will be explained with the aid of figures:

1. ***Start up*:** Model sets all parameters default values (realTime, sliders, Drawing style), and sends them to ControlUI class. ControlUI updates the values on the scrollbars. ViewNum01 receives the computed data coming from the Model, and after some later data process (unit conversion and projection to 2D) it plots the 2D projected waves.

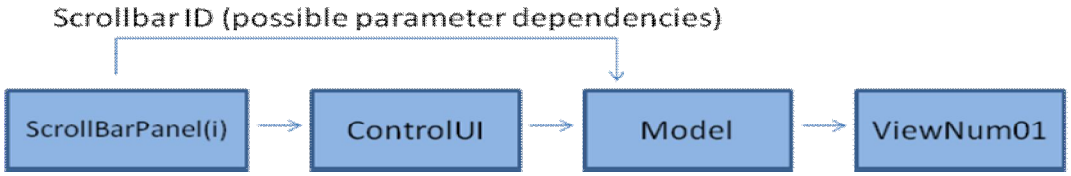
5.3 sequence of data conveying

2. ***User changes the value of a Scrollbar*:** It can be done either by dragging the slider using the mouse or by inputting a number in the text fields. Two different situation can happen:
   a. The slider changed does not have parameter dependencies, so although its value has changed, it does not affect any other scrollbar.

   In this case the sequence is as follows:

   When ScrollBarPanel class detects an event has been thrown, (user enters a new value in the text field or, what is the same thing, drags the slider) it sends directly to the Model the identifier (ID) of this scrollbar (there are 12 scrollbars and are numbered from 1 to 12). Once ID is sent, Model gets the current value of all the scrollbars from the ControlUI and checks whether this identifier obliges to apply any parameter dependencies or not. If we assume at the present time that there are no parameter dependencies, then the Model computes the equations using the new scrollbar values and

sends them to ViewNum01 by means of dataE and dataH matrixes, so the two waves can be updated according to the new computed values. Notice that Model does not send any value to ControlUI because there are no updates to be done on the scrollbars.



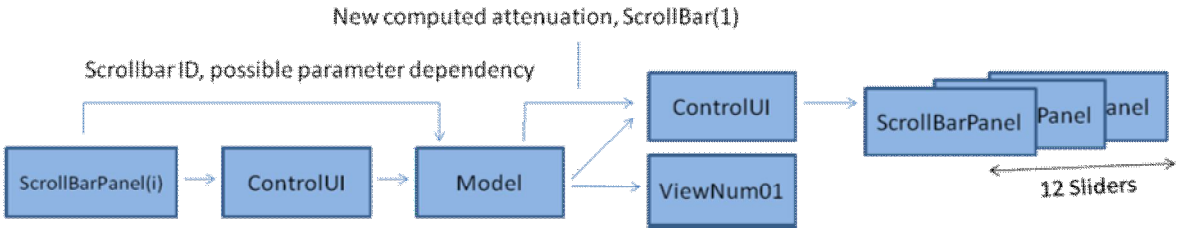
5.3 sequence without parameter dependencies

b. Model checks if any parameter dependencies must be applied and the ID sent from one out of the nine ScrollBarPanel class demonstrates it has to. That is, the changed scrollbar must be one amongst:

- Angular frequency $\omega[rad/s]$.
- Relative dielectricity ε_r.
- Relative permeability μ_r.
- Conductivity $\sigma[s/m]$.

These parameters are characteristic of every material, and are the ones that affect the attenuation $\alpha[Np/m]$ and the phase number $\beta[rad/m]$ of the plotted Transversal wave. Therefore, not only the steps of point number 2 (*User changes the value of a Scrollbar*) must be followed, but also the Model has to compute the new attenuation and phase number because one of the dependent scrollbars changed its value.

So, to summarize it, what Model does different in case B than in case A, is that has to re-compute the new attenuation and phase number (attenuation corresponds to scrollbar with ID number 1 and phase number to ID number 2) and to send the results to ControlUI by using specific methods programmed for doing this job.
ControlUI receives the new attenuation value and continues with the sequence shown in point number 1 (*Start up*). The figure below shows that all scrollbars are updated, even if only for example scrollbar(ID=1) must be updated. It is done this way in order to save unnecessary java code writing and does not affect the simulation performance in any case.



5.4 sequence with scrollbar parameter dependencies

For further information about parameter dependencies check chapter 6.2.

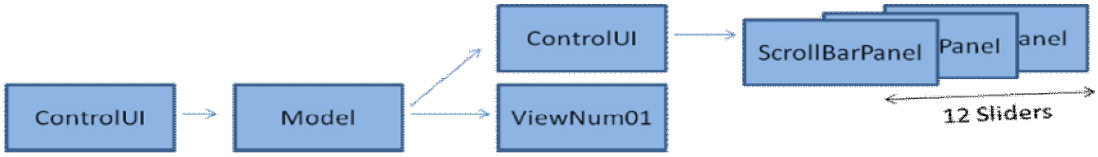
3. ***User changes the view mode of waves graphs:***

   The user can change the view mode of the wave in an interactively way. He can choose whether the two waves are shown at the same time or only one of them. Also the E and H waves can be displayed in to modes: in arrows or in curves style.
   The default value of the two existing combo box is 0, which means both waves are plot together and in curves style. These features do not change unless the user changes the default values by using the combo box.

   A JComboBox is a component of java GUI Swing, same thing happens with buttons, text fields, labels...
   It is added in ControlUI class. As soon as the graph mode changes, an action event is thrown within ControlUI class, and the new value is put into Model class. The role of the Model is to send the new value to ViewNum01 class, so it can repaint the graph using the new user chosen style.

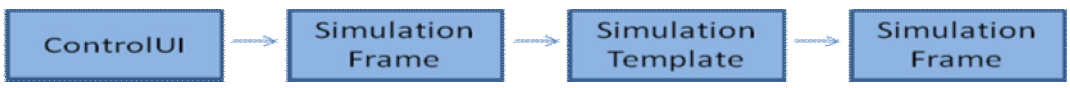
5.5 sequence of events when the view mode of drawing the waves is changed

4. ***User presses Reset button*****:** The procedure is very similar to the one followed in point number 3 (*User changes the view mode of waves graphs*). When pressing the button, an action event is thrown in ControlUI class. It is notified to Model listeners. The simulation default parameters are stored in Model class, so it can start the simulation applet for the first time or it can restart it whenever reset button is pressed, which is the situation we now assume.
   So the sequence that follows once the Model class receives the event of reset- button-pressed, is the same than in point number 1 (*start up*).



5.6 sequence of events when the reset button is pressed

5. ***User presses Exit button*****:** When so, in ControlUI an action event is thrown. Right after this, the method DestroyAll() in SimulationFrame is called, which stops the thread located in Model class and calls destroy() method in SimulationTemplate class. Destroy() method belongs to the Applet class, and its duty is to destroy any resources that the applet program has allocated. For instance, in SimulationTemplate, destroy() method calls dispose() method for SimulationFrame class. Dispose() method releases all native screen resources of JFrame contained in SimulationFrame, and all its subcomponents.



5.7 sequence of events when the exit button is pressed

5.1.4. View system features

It is time now to focus on the ViewUI class and on other related generalities.
ViewUI class makes up the MVC concept together with the Model and the ControlUI. In simulation template applet, the main tasks of the ViewUI are:

- To add the ViewNum01 and the Model listeners.
- To create and to set the dimensions of the ViewNum01 panel, where the 2D projected E and H waves will be drawn.

An overview of ViewNum01 class is going to be done:

As said before, ViewNum01 receives the data from ViewUI to set and to locate the size of its own panel. In addition, it receives straight from the Model the matrixes dataE and dataH. Using these matrixes is capable of drawing the E and H waves and the (x, y, z) axis as well as to plot the wave either in vectors or curves styles. To do so, however, viewNum01 needs to process the points contained in dataE and dataH before drawing the waves.

What does this processing consists in? It consists in converting the real units of the wave points into pixels. So E field points are converted from Volts/meter units to pixels, and the H field points are converted from Ampere/meter to pixels. This change on the units of the 3D points computed in the Model class is needed to draw the waves on the panel with their appropriate dimensions, because everything drawn on the panel is always in pixels units.

Once the wave points are in pixels units, the ViewNum01 projects them from 3D to 2D points. A projection to 2 dimensions is needed to draw on the computer screen. Thus, everything drawn on the panel (wave lines, wave vectors, axis, tics of the axis) is projected 2D points in pixels units.

For further information with regard to 3D to 2D projection, check chapter 6.3.2.

Last but not least, the ViewNum01 receives data only from the ViewUI and the Model, not from the ControlUI although the MVC concept says the Control must listen to both the View and the Model classes. So what we have so far is a minor change in MVC concept. We can affirm it is an unimportant fact because there are no processing speed issues and the bases of MVC concept are respected.

The figure below shows the particular used MVC concept. The Model class computes the points from the time domain E and H wave equations, and input them on dataH and dataE matrixes, which are sent directly to ViewNum01, where this data is processed to be eventually drawn on the screen.

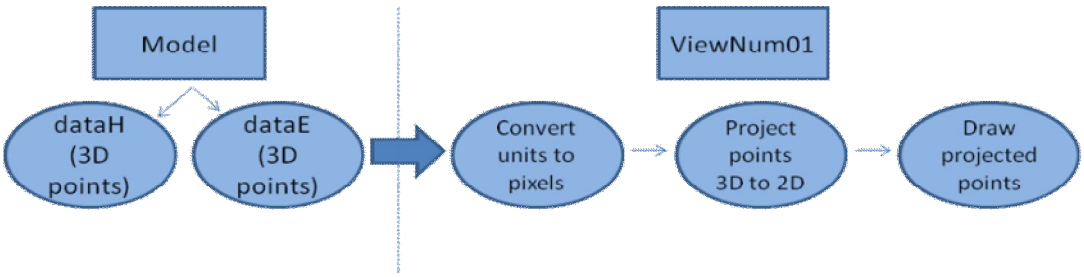


Figure 5.8 Steps to draw projected points

5.1.5. ControlUI features

ControlUI major duties can be listed in several points:

- Implements MVC control GUI.
- Adds all user controls, such as sliders, buttons and combo boxes.
- Adds the I/O methods, to join the ControlUI class to all the other classes.
- Adds model listeners.

ControlUI sets data from or to the Model and from or to the ScrollbarPanel class and listens or notifies to the Model and to the ScrollbarPanel class any change occurred on the scrollbars, on buttons…

One of the aims of ControlUI is to create and to add the buttons, as well as to add the scrollbars displayed on the applet. Actually, the scrollbars are created, painted and located in ScrollBarPanel class as well as the text fields where users can input new wave parameters. ControlUI listens to the Model and to the ScrollBarPanel classes.

ScrollbarPanel class simply throws "*action performed*" or "*adjustment value changed*" events to the Model and to the ControlUI every time a text field or a scrollbar is changed. Furthermore, it receives from ControlUI the current scrollbar values and updates them, when the applet is run for the first time or when reset button is pressed.

ScrollBarPanel class is used by ControlUI class to create the 12 scrollbars that appear in the simulation applet.

5.1.6. Properties files

Applet SimulationTemplate includes a properties files application. This application is meant to be a fast applet translator. It consists of:

- Some .properties files. These sorts of files are used to store strings for localization; these are known as Property Resource Bundles. Each parameter is stored as a pair of strings, one storing the name of the parameter (called the key), and the other storing the value. At SimulationTemplate applet, the keys correspond to all the strings to be translated, and the value contained in the .properties files correspond to the translated strings in the chosen language.

- A .conf file containing all the links of the applet to the properties files, so when the language of the applet wants to be changed, it is needed to open util.conf file and to choose among the available languages.

To run properties files it is needed to include *java.util.Properties* class. The objects of this class are created in SimulationTemplate class, and the needed *.conf* file is loaded there as well. There must be a specified path to search automatically at the location where the file is.

Check point [5] for more information about loading properties files (and file-based configurations in general).

5.1.7. Javadoc comments

Javadoc is a tool for generating API documentation in HTML format from comments in source code. It is a tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields.

In SimulationTemplate the comments emphasize on the most important public methods of the classes, and they have been created to guide the user throw the code and to make easier to understand the whole class structure of the applet.

Check point [6] to visit reference pages listing all *javadoc* tags and command-line options.

5.1.8. HotEqn

SimulationTemplate applet incorporates a version of *HotEqn* application. *HotEqn* is an AWT-based Java applet to view and display mathematical equations on the Web, using HTML language. The applet uses the familiar LaTeX notation to code its equations. The added version of *HotEqn* is a variant of *CHotEqn* application.

The original *CHotEqn* applet supported all LaTeX features, but did not include methods for equation editing, variable substitution, and for more sophisticated interactive elements. This applet was preferably for only showing mathematics statically.

In SimulationTemplate, *CHotEqn* has been slightly modified in order to simplify it as much as possible, and currently SimulationTemplates version is meant to be a simple displayer of mathematical equations on the applet. Some original features have been removed, such as mouse operations for debugging purposes.

It consists of three java classes, *CHotEqn*, *EqnScanner* and *EqToken*, as well as of 10 .gif extension archives.

The main features and duties of each one of them are:

- **CHotEqn:** It contains the majority of the methods running the application. *CHotEqn* objects are added in SimulationTemplate class. For each desired equation to be written down, a new CHotEqn constructor must be added in

SimulationTemplate. Basically it receives from the SimulationTemplate class the LaTeX equation code to be included on the applet, and it converts this LaTeX code into the appropriate mathematical way, by scanning and parsing the received LaTeX strings. An equation image is created out of them, and they are sent to the desired frame where are plotted.

- **EqnScanner:** Provides the detection methods to detect the elements (tokens) on an equation.

- **EqnToken:** Contains the list of all the scanner tokens to be recognized and parsed.

  .GIF images: Are 10 files containing all the mathematic symbols. When one symbol is required to be drawn, for instance a Greek letter, cHotEqn class loads it from the .gif files.

$$E_{0x}=e^{\alpha z}k_r\cos(\Phi)\cos(\omega t-\beta z)-k_i\cos(\Phi+\psi)\sin(\omega t-\beta z)$$
$$E_{0y}=e^{-\alpha z}k_r\sin(\Phi)\cos(\omega t-\beta z)-k_i\sin(\Phi+\psi)\sin(\omega t-\beta z)$$

Figure 5.9 Example showing electric waves phasors equations captured from the ViewNum01 panel and written using hot equations.

Check point [7], hot equations website, which includes a tutorial on LaTeX writing.

6. Simulated models

6.1. Analytic model of a homogeneous TEM electromagnetic wave in a LIH media

The waves radiate spherically, but at a remote distance away from the source they resemble *uniform plane waves*. The $\vec{E}$ (electric) and $\vec{H}$ (magnetic) fields are always orthogonal to the direction of propagation, which will always be considered in this paper work as $\vec{1z}$.

The waves drawn in this applet are assumed to be plane, in a *linear, isotropic and homogenous (LIH)* media. In this chapter, there are going to be shortly discussed all the consequences of dealing with LIH media, and will be listed the LIH explicit waves equations, which will be the ones to be plotted in the applet. All these LIH equations are the results of solving Maxwell equations with some specific values due to the LIH media.

Starting from the beginning, that is Maxwell equations, it is possible to arrive at Helmhottz equation for time-harmonic electric fields:

$$\nabla^2 \vec{E}_{(\vec{r})} = j\omega\mu(\sigma + j\omega\varepsilon)\vec{E}_{(\vec{r})} \qquad (6.1)$$

where: ω: Angular frequency [Rad/s] μ: Magnetic permeability
σ: Conductivity [S/m] ε: Electric dielectricity
$\vec{E}_{(\vec{r})}$: is a complex vector: (Ex, Ey, Ez)

To reach the equation (6.1), it is necessary to work on Maxwell's equation but within the *isotropic* TEM waves. When all physical properties are the same in different directions it is call *Isotropic* media. Those Maxwell's reduced equations due to *isotropic* media are:

$$\nabla \vec{E} = 0 \qquad \nabla \vec{H} = 0$$

$$\nabla \times \vec{H} = \sigma \vec{E} + \varepsilon \frac{\partial \vec{E}}{\partial t} \qquad \nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t} \qquad (6.2)$$

The next equations are the result of solving Helmhottz equation using (6.2) equations:

$$\vec{E}_{(\vec{r})} = \vec{E}_0 e^{-\vec{\gamma}\vec{r}} \qquad (6.3)$$

$$\vec{H}_{(\vec{r})} = \frac{\vec{1}_v \times \vec{E}_0}{Z} e^{-\vec{\gamma}\vec{r}} \qquad (6.4)$$

$$\vec{H}_0 = \frac{\vec{1}_v \times \vec{E}_0}{Z} \qquad (6.5)$$

$\vec{1}_v$: Direction of propagation

$$Z = \frac{j\omega\mu}{\gamma} \quad \text{Wave impedance} \qquad (6.6)$$

$\vec{\gamma}$: Propagation vector

$\vec{\gamma}$ is used to describe the behavior of an electromagnetic transmission wave.
Its mathematical expressions are:

$$\gamma = \sqrt{j\omega\mu(\sigma + j\omega\varepsilon)} \qquad \text{propagation constant} \qquad (6.7)$$

or $\vec{\gamma} = \vec{\alpha} + j\vec{\beta}$ (6.8)

Where: $\vec{\alpha}$: attenuation vector [Np/m]
$\vec{\beta}$: phase vector [Rad/m]

The real part of the propagation constant is the attenuation constant and is denoted. It causes the signal amplitude to decrease.

The phase constant adds the imaginary component to the propagation constant. It determines the sinusoidal phase of the signal.

To better understand the meaning of attenuation constant α and phase number β, it is next added a graph showing how the wave parameters affect the wave propagation:

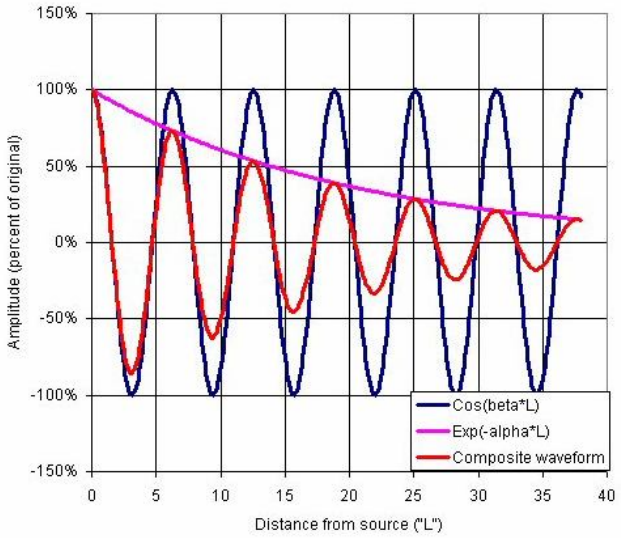


Figure 6.1 Amplitude of an electromagnetic wave versus distance when time is frozen. For further information check Reference [8].

If it is assumed that we are describing homogeneous wave, then $\vec{\alpha} \parallel \vec{\beta}$ and consequently $\vec{\alpha} = \alpha\vec{1_v}$ and $\vec{\beta} = \beta\vec{1_v}$ so it can be affirmed:

$$\vec{\gamma}\vec{r} = (\vec{\alpha} + j\vec{\beta})\vec{r} = (\alpha\vec{1_v} + j\beta\vec{1_v})\vec{r} = (\alpha + j\beta)z$$

According to the construction of wave equations, the direction of the wave propagation is the direction of the phase vector $\vec{1_v} = \frac{\vec{\beta}}{\beta}$. Without any loss of generality we can assume $1_z = \vec{1_v} = (0,0,1)$.

As a result of it, the equation (6.3) can be converted into $\vec{E}_{(\vec{r})} = \vec{E}_0 e^{-\alpha z} e^{-j\beta z}$ (6.9)

Now it is a matter of going from the complex domain to the time domain and to get the real part of it, taking into account every complex number can be split into its real and its imaginary part: $\vec{E}_0 = \vec{E}_{0r} + j\vec{E}_{0i}$

$$\vec{E}_{(z,t)} = \Re\left\{\vec{E}_{(z)}\, e^{jwt}\right\} = \Re\left\{(\vec{E}_{0r} + j\vec{E}_{0i}) e^{-\alpha z} e^{-j\beta z}\, e^{jwt}\right\}$$

Finally, the equation of the electric field is obtained after computing the equation above:

$$\vec{E}_{(z,t)} = e^{-\alpha z}(\vec{E}_{0r}\cos(\omega t - \beta z) - \vec{E}_{0i}\sin(\omega t - \beta z)) \quad (6.10)$$

The procedure to reach the magnetic field equation is very similar:

Starting within the complex domain, the general equation for the magnetic field is the equation (6.4)

Though from now on, it is assumed that the direction of propagation is (z), in the same way as in the magnetic field case. Also the magnetic field is split in real and imaginary components and the wave impedance Z written with the equations (6.6) and (6.8):

$$\vec{H}_{(\vec{r})} = \frac{\vec{H}_{0r} + j\vec{H}_{0i}}{\frac{j\omega\mu}{\alpha + j\beta}} e^{-\alpha z} e^{-j\beta z}$$

Back to time domain, and after some steps the next equation is derived:

$$\vec{H}_{(z,t)} = \Re\left\{\vec{H}_{(z)}\, e^{jwt}\right\} = \ldots = e^{-\alpha z}\left(\vec{H}_{0r}\cos(\omega t - \beta z) - \vec{H}_{0i}\sin(\omega t - \beta z)\right) \quad (6.11)$$

The equations (6.10) and (6.11) are the ones that the Model class of the applet will compute, and eventually they will be drawn. However, they are not completed yet.

Amongst all the variables that appear on the equations, there are some that must be first computed and written in the appropriate way before being computed in JAVA.

These are the cases of the phasors components $\vec{E}_{0r}$, $\vec{E}_{0i}$, $\vec{H}_{0r}$ and $\vec{H}_{0i}$. To compute them in JAVA language, it is necessary to arrange them.

The roles of these phasors on the equations are to control the polarization of the electric and magnetic waves. First we introduce the orthogonal basis of transversal plane. The transversal plane is the plane orthogonal to the wave direction that is orthogonal to the phase vector β. We choose the orthogonal base of transversal plane $\{1_\varphi, 1_\gamma\}$ so that $1_\varphi \perp 1_\gamma$ and $1_\varphi \times 1_\gamma = +1_v$.

The wave polarization is the orientation of the lines of electric flux in an electromagnetic field. Since in this paper work only *Transverse Electromagnetic (TEM)* waves are studied, the electric and the magnetic fields are both at 90^0 of the direction of propagation (direction z).

Therefore, the real and the imaginary components of phasors $\vec{E}_0$ and $\vec{H}_0$ will always be within the (x,y) plane and shifted 90^0 degrees to each other, as shown below:

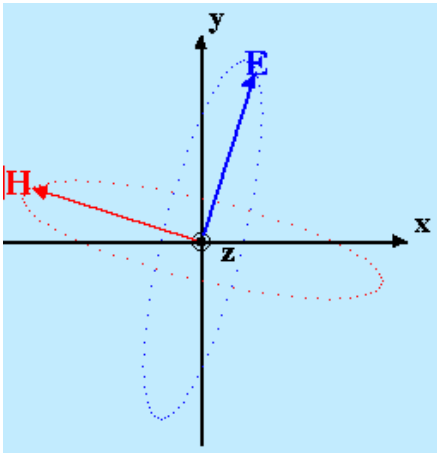


Figure 6.2 Applet program showing an elliptical polarization of $\vec{E}_0$ and $\vec{H}_0$ of a TEM. Interactive simulation is available at [9].

When time is running the vectors change their position, describing an elliptical curve. The direction of propagation is z axis, which is where the view point of this applet is located.

There are three possible states of polarization, depending on the phasors components values.

$\vec{E}_0 = \vec{E}_{0r} + j\vec{E}_{0i}$ or what is the same $\vec{H}_0 = \vec{H}_{0r} + j\vec{H}_{0i}$

1. *Linear polarization:* $\vec{E}_{0r} \parallel \vec{E}_{0i}$ where $\vec{E}_{0r} = E_{0r}\vec{1_E}$ and $\vec{E}_{0i} = E_{0i}\vec{1_E}$
   Components $E_{0r}, E_{0i} \in \mathbb{R}$.

   In Linear case, the two perpendicular components are in phase, so the direction of the electric vector $\vec{E}_0$ is constant and traces out a line in the plane.

2. *Circular polarization:* $\vec{E}_{0r} \perp \vec{E}_{0i}$ and $\left\|\vec{E}_{0r}\right\| = \left\|\vec{E}_{0i}\right\| = \|K\|$
   Where: $\vec{E}_{0r} = K\vec{1_\varphi}$ and $\vec{E}_{0i} = K\vec{1_\gamma}$
   The two orthogonal components have exactly the same amplitude and are exactly 90^0 out of phase. In this case one component is zero when the other component is at maximum or minimum amplitude. In this special case the electric vector traces out a circle in the plane.

3. *Elliptical polarization:* $\vec{E}_{0r} = K_1\vec{1_\varphi}$ and $\vec{E}_{0i} = K_2\vec{1_\gamma}$

Comprises all cases where the two components are not in phase and either do not have the same amplitude and/or are not ninety degrees out of phase. The electric vector traces out an ellipse in the plane.

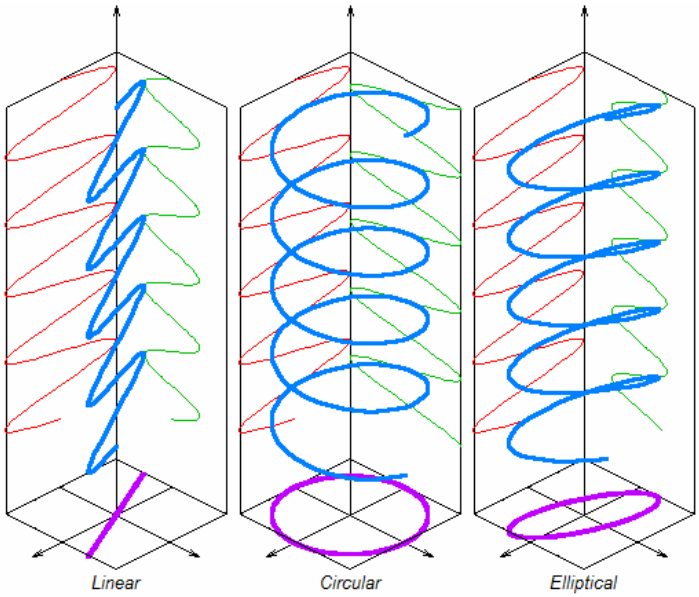


Figure 6.3 Draw showing the three possible states of polarization of a TEM wave in time evolution. Blue curves are the sum of the two perpendicular components of the wave vectors in (x,y) plane. Violet curves are the projections on a plane

The most general case of TEM wave components is:

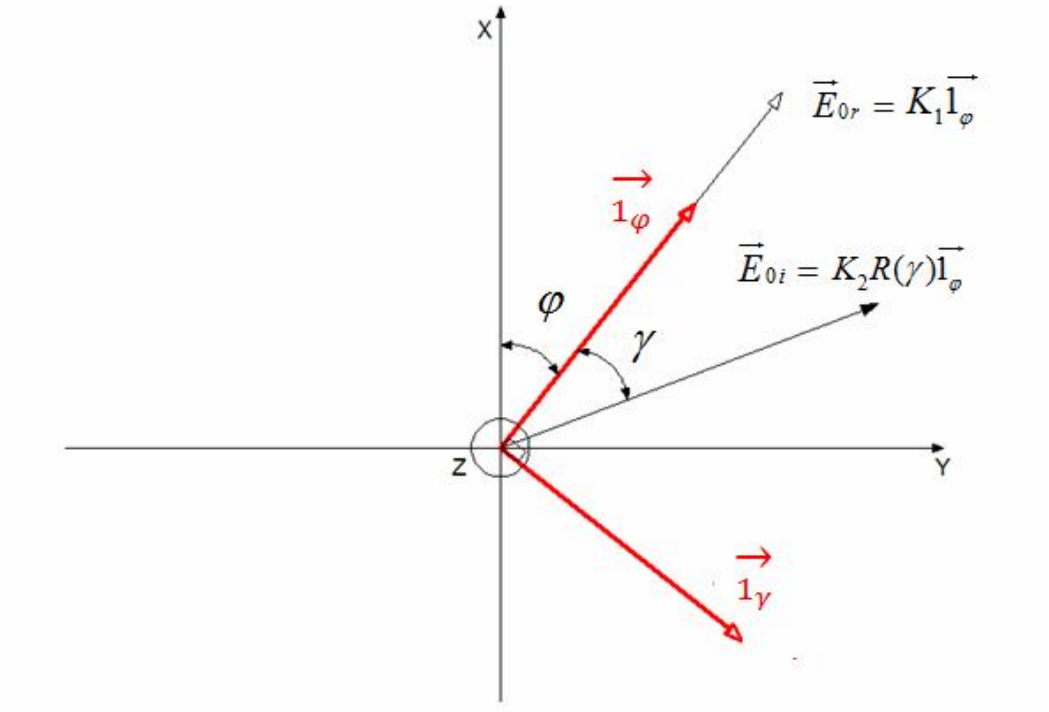


Figure 6.4 Shows the orthogonal base and the E vector components

From where it is deduced the following things:

The chosen perpendicular base is $\{1_\varphi, 1_\gamma\}$

$$\vec{E}_{0r} = K_1(\cos(\phi), \sin(\phi), 0) = K_1\vec{1_\varphi}$$
$$\vec{E}_{0i} = K_2(\cos(\phi+\varphi), \sin(\phi+\varphi), 0) = K_2 R(\gamma)\vec{1_\varphi}$$

Finally the two equations of the phasors that Model class computes have been deduced. Replacing the vectors on the electric field equation (6.10):

$$\vec{E}_{(z,t)} = e^{-\alpha z}(K_1(\cos(\phi), \sin(\phi), 0)\cos(\omega t - \beta z) - K_2(\cos(\phi+\varphi), \sin(\phi+\varphi), 0)\sin(\omega t - \beta z)) \quad (6.12)$$

Model class constructs two matrixes called **dataE[3][numOfPoints]** and **dataH[3][numOfPoints]** in order to send the computed values to the View class in an appropriate way. Then, the three components belonging to every vector are placed at dataE and dataH rows, and there are as many vectors as numOfPoints that want to be drawn. If N = NumOfPoints, the structure of the two matrixes, either dataE or dataH is:

$$\begin{pmatrix} dataE[x_0][0] & dataE[y_0][0] & dataE[z_0][0] \\ dataE[x_1][1] & dataE[y_1][1] & dataE[z_1][1] \\ ..... & ..... & ..... \\ dataE[x_N][N] & dataE[y_N][N] & dataE[z_N][N] \end{pmatrix}$$

For instance, at fixed point i, a vector lays out in a matrix row the following three components (x, y, z):

$$\begin{aligned} x &= dataE[0][i] = e^{-\alpha z}K_1\cos(\phi)\cos(\omega t - \beta z) - e^{-\alpha z}K_2\cos(\phi+\varphi)\sin(\omega t - \beta z) \\ y &= dataE[1][i] = e^{-\alpha z}K_1\sin(\phi)\cos(\omega t - \beta z) - e^{-\alpha z}K_2\sin(\phi+\varphi)\sin(\omega t - \beta z) \\ z &= dataE[2][i] = \Delta z \square i \end{aligned} \quad (6.13)$$

Note that the component z is not null, because the vector is placed according to the desired number of points to be drawn and multiplied by a constant. This point is widely explained in chapter 6.3.1.

As said before, the Model class also defines a matrix, called dataH[3][numOfPoints], that is intended to do the same than dataE[3][numOfPoints], but this time the values introduced will correspond to those of the magnetic field:

Firstly, it is needed to calculate the real and the imaginary parts of the magnetic phasor : $\vec{H}_0 = \vec{H}_{0r} + \vec{H}_{0i} = \frac{\vec{1}_z \times \vec{E}_0}{\vec{Z}}$

Since the electric phasor ($\vec{E}_0$) is known, the real and the imaginary parts of the magnetic phasor are also know:

$$\vec{H}_{0r} = \frac{\beta}{\omega\mu}K1(-\sin(\phi), \cos(\phi), 0) + \frac{\alpha}{\omega\mu}K2(-\sin(\phi+\varphi), \cos(\phi+\varphi), 0) \quad (6.14)$$

$$\vec{H}_{0i} = \frac{\beta}{\omega\mu} K2\left(-\sin(\phi+\varphi), \cos(\phi+\varphi), 0\right) - \frac{\alpha}{\omega\mu} K1\left(-\sin(\phi), \cos(\phi), 0\right) \quad (6.15)$$

Incorporating the equations (6.14) and (6.15) into the equation (6.11) and if the three components are separated to match them into the dataH matrix:

$$dataH[0][i] = \frac{e^{-\alpha z}}{\omega\mu}\left(-\left(\beta K1\sin(\phi)+\alpha K2\sin(\phi+\varphi)\right)\cos(\omega t-\beta z)+\left(\beta K2\sin(\phi+\varphi)-\alpha K1\sin(\phi)\right)\sin(\omega t-\beta z)\right)$$

$$dataH[1][i] = \frac{e^{-\alpha z}}{\omega\mu}\left(\left(\beta K1\cos(\phi)+\alpha K2\cos(\phi+\varphi)\right)\cos(\omega t-\beta z)-\left(\beta K2\cos(\phi+\varphi)-\alpha K1\cos(\phi)\right)\sin(\omega t-\beta z)\right)$$

$$dataH[2][i] = \Delta z \cdot i \quad (6.16)$$

Eventually, it has been obtained the equations (6.13) and (6.16), which are the ones computed by the Model class and sent to the ViewNum01 class so they can be drawn.

Looking the equations thoroughly, it is possible to find out what the variables of the equations are. These variables correspond to the scrollbars that appears on the applet and which can be changed by the user interactively by sliding the scrollbars:

Name of the variable	Scrollbar identifier on applet [1..12]
Attenuation constant [Np/m] (α)	1
Phase number [Rad/m] (β)	2
Angular frequency [Rad/s] (ω)	3
Relative dielectricity (ε_{rel})	4
Relative permeability (μ_{rel})	5
Conductivity [S/m] (σ)	6
Real time flow [ns]	7
Refresh time [ms]	8
Real amplitude [V/m] ($K1$)	9
Imaginary amplitude [V/m] ($K2$)	10
Angle or real component [Rad] (ϕ)	11
Angle of imaginary component [Rad] (φ)	12

Figure 6.5 Table matching the scrollbars on the applet with their identifiers

6.2. Parameters dependencies

There are some variables in *figure 6.5* that are not on these equations but that appear on the expressions to calculate α and β. These parameters are characteristic of every material. The following equation is important to find any dependency between the equation parameters (left part of the equation) and the media parameters (right part). Note that this equation is valid as long as the wave is homogeneous ($\vec{\alpha} \parallel \vec{\beta}$):

$$\alpha^2 - \beta^2 - 2j\alpha\beta = -\omega^2\mu\varepsilon + j\omega\mu\sigma \tag{6.17}$$

After some computing, these four equations are found:

$$\varepsilon_{rel} = \frac{\beta^2 - \alpha^2}{\omega^2 \mu_0 \mu_{rel} \varepsilon_0}$$

$$\sigma = \frac{2\alpha\beta}{\omega\mu}$$

$$\alpha = \omega\sqrt{\frac{\mu_0\mu_{rel}\varepsilon_0\varepsilon_{rel}}{2}\left(\sqrt{1+\left(\frac{\sigma}{\omega\varepsilon_0\varepsilon_{rel}}\right)^2} - 1\right)} \tag{6.18}$$

$$\beta = \omega\sqrt{\frac{\mu_0\mu_{rel}\varepsilon_0\varepsilon_{rel}}{2}\left(\sqrt{1+\left(\frac{\sigma}{\omega\varepsilon_0\varepsilon_{rel}}\right)^2} + 1\right)}$$

The values with a subscript 0 mean the media is free space, while these with a subscript *rel* are the ratios of an specific media to the free space media, therefore it is possible to replace the values for an specific material:

$$\mu_{rel} = \frac{\mu}{\mu_0} \qquad \varepsilon_{rel} = \frac{\varepsilon}{\varepsilon_0}$$

The applet puts into practice these parameters dependencies. That is, the equations (6.18) are inputted in a way that the right side of the equal sign is dependent on the parameter of the left side. So when some of the parameters of the left side change, the parameter on the left side must be re-computed.

6.3. Conversion of field units, projection 3D to 2D and sizing waves with constants

When the ViewNum01 class receives the dataE and dataH matrixes, which contains the 3D points of the waves, must do some processing of this data before drawing it onto the screen. This chapter is intended to overview this procedure.

6.3.1. E and H Units conversions

Firstly, a unit conversion is needed. All the elements drawn on the screen are in pixels units (axis, tics on the axis) and the panel on ViewNum01 is sized in pixels, as well. It makes sense, then, that the 3D points that will be used to draw the E and H waves must also be in pixels. Otherwise the drawn wave would be wrong; units of all plotted elements can not be mixed.

To convert the units into pixels, it is only necessary to multiply by a constant value the 3D data incoming from Model class by means of dataE and dataH.

Let's understand the important variables involved in the conversion procedure:

$numOfPeriods = 4;$ //Fixes the number of periods that will appear on the screen.
$ref\ \beta = 20;$ // Reference value of the wave number [Rad/s].

Now, how many meters are needed to draw 4 periods on the screen?

As it is already known, the waves are modeled with sines and cosines, for instance:

$\cos(\omega t - \beta z)$ Thus, it is possible to impose 4 periods (2Π is one period) to be the maximum length of the drawn wave in meters:

$$ref\ \beta[rad/m]Z\max[m] = (2\Pi \cdot numOfPeriods)Z\max;$$
$$Z\max = 1.257[m]$$

This is the "real units" distance in meters that 4 periods will take up on the screen. Since the panel is measured in pixels (600×660) pixels, it has been set that the direction of propagation (Z axe) will take up 500 pixels, leaving nearly 50 free pixels on every side. As a result of it, we are in conditions now to demonstrate the constant that will convert meters into pixels:

$$pixPerMeter = 500[pix] \Big/ Z\max[m] = 398.88[{}^{pix}\!/_{m}]$$

The number of points which draw the 4 periods is also known if previously the distance in pixels between every point computed in the Model class is decided.

$dPix = 6[pix];$ //distance of pixels left in between every point to be drawn.
$$numOfPo\text{int}s = {}^{500[pix]}\!/_{6[pix]} = 83$$

So it is come to the conclusion that the Model class computes the matrixes dataE and dataH, each one of them taking 83 3D points to the ViewNum01 class. And the distance between each computed point in meters is:

$$disZ = \frac{Z\max[m]}{NumOfPo\text{int}s} = 0,015[m] \qquad (6.19)$$

The explanation of X and Y conversion constants is similar to the one of z dimension although the units now are not meters, but in Volts per meter $[V/m]$ and in Amperes per meter $[A/m]$ respectively.

To set the constants, it is needed to know the average amplitude of the wave (considering the attenuation does not affect on it) and the number of pixels that want to be taken up. For instance, considering now the E wave:

$NumOfPixels = 150[pix];$

$$refEampl = \sqrt{Kr^2 + K_i^2} = 2.83$$

The number of pixels taken to draw the waves on the screen have been chosen to match them with the length in pixels of the positive part of X and Y axis, whereas the average amplitude ($refEampl$) is the result of fixing both reference amplitude constants (real and imaginary parts) $Kr = Ki = 2[V/m]$. So the constant is already set:

$$pixPerVpM[\frac{pix}{V/m}] = \frac{numOfPixelsX}{refEampl} = 53[pix] \tag{6.20}$$

When the constants $pixPerVpM$ and $pixPerMeter$ have been calculated, it is a matter of multiplying these constants by the values out coming from the Model class in dataE matrix as shown beneath:

$$\begin{pmatrix} dataE[x_0][0] & dataE[y_0][0] & dataE[z_0][0] \\ dataE[x_1][1] & dataE[y_1][1] & dataE[z_1][1] \\ ..... & ..... & ..... \\ dataE[x_N][N] & dataE[y_N][N] & dataE[z_N][N] \end{pmatrix} \tag{6.21}$$

$$\Downarrow$$

$$\begin{pmatrix} dataE[x_0][0]\cdot pixPerVpM & dataE[y_0][0]\cdot pixPerVpM & dataE[z_0][0]\cdot pixPerMeter \\ dataE[x_1][1]\cdot pixPerVpM & dataE[y_1][1]\cdot pixPerVpM & dataE[z_1][1]\cdot pixPerMeter \\ ..... & ..... & ..... \\ dataE[x_N][N]\cdot pixPerVpM & dataE[y_N][N]\cdot pixPerVpM & dataE[z_N][N]\cdot pixPerMeter \end{pmatrix}$$

Note that the first two columns are the (x, y) points of the 3D electric wave, whereas the third component is the location of the wave vector in the (z) axis (direction of propagation). So the units of (x, y) are $[V/m]$ and the units of (z) are $[m]$.
Let's check out what is the information of (z):

- Locates the vector through the wave's direction of propagation. So it is composed by an index increasing one unit every point, and which multiplies the constant given in equation (6.19). Then, the points stored in (z) component are such as:

$$dataE[z_0][0] = 1 * distZ$$
$$dataE[z_0][1] = 2 * distZ$$
……
$$dataE[z_0][NumOfPo\,int\,s] = NumOfPo\,int\,s * distZ$$

Where $distZ$ is the constant giving the distance in between every two computed points.

- Is introduced in the equations of (x, y) wave components because is the radius vector of the wave: $\vec{r} = (0, 0, z)$.

Similar steps have been followed in order to find the most appropriate constant for H wave:

We are currently dealing with $[Amperes/m]$ units, and it is known from equation (6.5) that the H field amplitude is proportional to the E field amplitude divided by the wave impedance as shown beneath:

$$H_{x,y} \in [0, \frac{E_{x,y}}{Z_{in}}]$$

The following conclusion is taken from this equation: points computed in dataH matrix are approximately Z_{in} times bigger than the ones in dataE matrix. At the time of drawing the waves, it would be an issue to have such a big amplitude difference between the H and the E waves, therefore it is needed to multiply H points with a constant that will put the amplitudes in a comparable size.

This constant is the reference value of: $\left|Z_{in\,Ref}\right| = \frac{\left|\omega_{ref}\,\mu_{ref}\right|}{\sqrt{\alpha_{ref}{}^2 + \beta_{ref}{}^2}} = C$

And since this is true: $H_{x,y} \approx \frac{E_{x,y}}{C}$

So: $$pixPerApm\left[\frac{pix}{A/m}\right] = \frac{numOfPix[pix]}{H_{x,y}} = C \cdot \frac{numOfPix[pix]}{E_{x,y}} = 3.34 \qquad (6.22)$$

The waves E and H will be approximately the same size as long as we multiply the dataE and dataH points with the factors found in equations (6.20) and (6.22).

Finally it is only needed to multiply the components of dataH matrix by $pixPerApm$ constant, exactly in the same way it is done in equation (6.21).

6.3.2. E and H 3D to 2D projection

It is necessary to project the 3D dataE and dataH points to 2D points, because the draw of the waves will be on the computer's screen, which is a 2D plane.

A 3D point is projected as seen from a given viewpoint υ. This viewpoint is represented by any point in the original 3D space. This projection will lie on the imagined 2D plane, which will be the computer's screen.

The equations giving the 2D points out of the 3D points are:

$$CompX2D = \frac{\left\|\vec{\upsilon}\right\|^2}{\left\|\vec{\upsilon}\right\|^2 - \vec{r}\vec{\upsilon}} \frac{1}{\sqrt{\upsilon_x{}^2 + \upsilon_y{}^2}} \left(-\upsilon_y x + \upsilon_x y\right) \qquad (6.23)$$

$$CompY2D = \frac{\left\|\vec{\upsilon}\right\|}{\left\|\vec{\upsilon}\right\|^2 - \vec{r}\vec{\upsilon}} \frac{1}{\sqrt{\upsilon_x{}^2 + \upsilon_y{}^2}} \left(-\upsilon_x \upsilon_z x - \upsilon_y \upsilon_z y + \left(\upsilon_x{}^2 + \upsilon_y{}^2\right) z\right) \qquad (6.24)$$

Where:

$\vec{r} = (x, y, z)$ is the 3D point.
$\vec{\upsilon} = (\upsilon_x, \upsilon_y, \upsilon_z)$ is the 3D view point.

The result of projecting all the points on the screen (points from the coordinate axes, points of the waves) is a 3D graph drawn in 2D.

For instance, axis (z) in 3D occupies pixels from $(0,0,0) \rightarrow (0,0,500)$.
After the projection is done, new points are $(0,0) \rightarrow (321,103)$.

This is the reason why the axis (z) in 2D appears to be turned. And this is how the sense of 3D is reached in 2D.

7. Simulation implementation tutorial

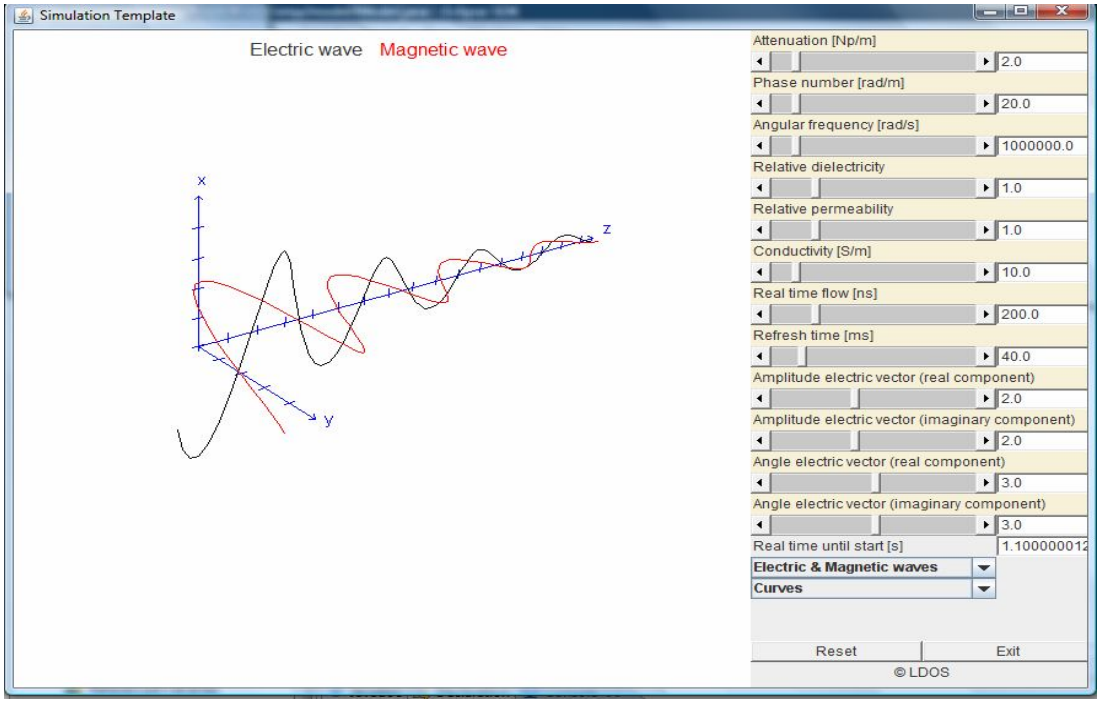
7.1. Introduction

This tutorial wants to provide the reader only with the basic knowledge needed to understand how this application is programmed and how it is run.
It means this tutorial is written either for users that simply want to work on this application or for java programmers that want to have this applet as an example to develop their own application based on the Simulation Template example.
Requirements
Basically, the following tools are needed:

- JDK from java installed in our computer.
- JRE 1.6.0 System Library to compile the code.
- An IDE to allow a quick java development and a project management (such as Eclipse).

7.2. Getting started

First of all it is needed to import the whole program package into the Eclipse environment, and run it from the class extending the JAVA applet class, which is the Simulation Template class.
The following frame will appear onto the screen:



Now let see what roles these elements play in the application:

1. The black lines is the electric wave and it is drawn from the following equations:

$$x = dataE[0][i] = e^{-\alpha z} K_1 \cos(\phi)\cos(\omega t - \beta z) - e^{-\alpha z} K_2 \cos(\phi + \varphi)\sin(\omega t - \beta z)$$
$$y = dataE[1][i] = e^{-\alpha z} K_1 \sin(\phi)\cos(\omega t - \beta z) - e^{-\alpha z} K_2 \sin(\phi + \varphi)\sin(\omega t - \beta z)$$
$$z = dataE[2][i] = \Delta z \cdot i$$

2. In the same way, magnetic wave is drawn in red from the following equation:

$$dataH[0][i] = \frac{e^{-\alpha z}}{\omega\mu}\left(-\left(\beta K1\sin(\phi) + \alpha K2\sin(\phi+\varphi)\right)\cos(\omega t - \beta z) + \left(\beta K2\sin(\phi+\varphi) - \alpha K1\sin(\phi)\right)\sin(\omega t - \beta z)\right)$$
$$dataH[1][i] = \frac{e^{-\alpha z}}{\omega\mu}\left(\left(\beta K1\cos(\phi) + \alpha K2\cos(\phi+\varphi)\right)\cos(\omega t - \beta z) - \left(\beta K2\cos(\phi+\varphi) - \alpha K1\cos(\phi)\right)\sin(\omega t - \beta z)\right)$$
$$dataH[2][i] = \Delta z \cdot i$$

Observations:

1. The discrete 3D points for drawing the electric (E) and magnetic (H) waves are computed in the Model class, stored in dataE and dataH matrixes and transferred to the ViewNum01 class where the graphs are drawn, after some further data processing. This data processing consists in converting the units E and H into pixels and in projecting to 2D the points stored in dataE and dataH.
2. The user can change the scrollbar values, so the graph will be repainted according to the new input values.
3. There are parameters dependencies between some scrollbars. If any of angular frequency (ω), relative dielectricity (ε_{rel}), relative permeability (μ_{rel}) or conductivity (σ) scrollbars is changed, it affects to both the attenuation (α) and the phase number (β) scrollbars.
4. It is possible to draw the E and H waves in two different ways by choosing either the Curves or Arrays modes. It can also be displayed only E wave, H wave or both of them at the same time.

3. The scrollbars allow the users to change interactively the parameters of the equations used to draw the two waves on the frame.

   The users can change the values either by scrolling the sliders up and down with the mouse or by entering the numeric values in the text fields located at the right side of the scrollbars.

   Observations:

   The scrollbar and the text field patterns are created in the ScrollBar class, but they are added and located in the ControlUI class together with the JAVA swing components (buttons, combo-boxes).

There are going to be shortly described all the scrollbars used in SimulationTemplate applet: Reset button: changes the current values of the scrollbars to the default values, which are stored in the Model class.

1. **Attenuation** $\alpha[Np/m]$: the reduction in nepers of wave's amplitude per meter.
2. **Phase number** $\beta[rad/m]$: The rotation rate in radians of the wave every meter.
3. **Angular frequency** $\omega[rad/s]$: How fast the wave is rotating.
4. **Relative dielectricity** ε_{rel} : Materials in which electrical charges appear at both ends are called dielectric substances. Relative dielectricity is characteristic of every material.
5. **Relative permeability** μ_{rel}: Is the degree of magnetization of a material that responds linearly to an applied magnetic field. Relative permeability is characteristic of every material.
6. **Conductivity** $\sigma[S/m]$: Is a property that allows electricity to flow through a material, measured in Siemens (electric conductance unit) per meter. Conductivity is characteristic of every material.
7. **Real time flow**$[ns]$: The time gone by since the simulation started.
8. **Refresh time** $[ns]$: How many times a second the thread runs to refresh the draw. The more the refresh time increases, the more the thread ceases the execution (sleep method is called to stop running the program for as much time as the value of refresh time).
9. **Amplitude electric vector (Real component)** $\vec{E}_{or}[V/m]$**:** The amplitude of the real component of the electric wave in time domain.
10. **Amplitude electric vector (Imaginary component)** $\vec{E}_{oi}[V/m]$**:** The amplitude of the imaginary component of the electric wave in time domain.
11. **Angle of electric vector (real component)** $\phi[rad]$**:** The angle of $\vec{E}_{or}[V/m]$ on the base $\{1_\varphi, 1_\gamma\}$
12. **Angle of electric vector (imaginary component)** $\varphi[rad]$**:** The angle of $\vec{E}_{oi}[V/m]$ on the base $\{1_\varphi, 1_\gamma\}$
13. **Reset button:** changes the current values of the scrollbars to the default values, which are stored in the Model class.
14. **Exit button:** stops running the applet.

7.3. Other features and properties

- The Simulation Template uses "property files" in order to translate very easily the language comments that appear in the applet. To change the current language, open the *util.conf* archive and set the desired language.

- There is available LaTeX writing for math equations, using *HotEqn* classes. Enter the equation using LaTeX encoding on SimulationTemplate class, and locate where desired the image created from LaTeX code. The image created is on ViewNum01 class and can be located wherever within the ViewNum01 panel.
- The main structure of the program is based on Model-View-Controller, that basically means:
  - Model class computes all the mathematical equations needed to draw the graphs.
  - ViewUI class creates ViewNum01 (creates its JAVA panel and transfers its size).
  - ControlUI class adds all the AWT Components (buttons, scrollbars,..) and listens the Model and the ScrollbarPanel classes.

7.4. Step by step instructions

In order to facilitate the programming of a new interactive simulation applet, the main points to be followed by the programmer are going to be detailed underneath:

1. Create a separate copy of the Java project in Eclipse.

2. Build the new mathematical model equations in Model class, by means of replacing the computed equations. New 3D points will be computed.

3. Since ViewNum01 class receives the new computed data from the Model class, change the displayed waves in there. Notice there is a unit conversion of E and H units into pixels, and a 3D to 2D projection of both the waves points and the axis (units of axis are always in pixels, and their length is fixed in 3D points on ViewNum01 array called graphPars).

4. To add new panels, containing new elements to be displayed on the applet (i.e. a third graph), create and add them on viewUI class. After this is done, a new class (i.e.) ViewNum02) can be added on the applet. It may be necessary to resize the whole Frame in SimulationFrame class (where the ViewUI is created and where the variables containing the sizes of the panel Views are set).

5. In ControlUI add and locate the new buttons, text fields, labels and all the new visual JAVA components desired. If a new ScrollBar wants to be set, add it in ScrollBar array, which is already created and contains Objects from the ScrollBarPanel class.

6. To change the language of the applet, copy and paste a *anylanguage.properties* file and translate the sentences into the desired language. After it is done, open the util.conf archive and change the current language by this one:

#languageFile=org/lor/simTemp/anylanguage.properties
The current language of the applet is the one that does not have the # symbol.
NOTE: the Unicode characters for Slovenian languages must be entered like shown: \u0XXX. The list of correspondence between Unicode symbols and characters is available at point [10].

7. To change or to add a mathematical equation written in LaTeX language on ViewNum01 panel, add the LaTeX code on the equation variable in SimulationTemplate class, which is waiting a string input as follows:
   //equation=" ";
   Every variable, (i.e. Greek letters) must be preceded by "\\" character).
   To get to a new line down, enter "\\\\" in a row.
   Go to ViewNum01 class and locate wherever it is wanted the new image coming from the LaTeX string equation.

8. Simulation results and evaluations

The result of this work is an interactive simulation allowing the users to check, in a graphic way, the mathematical equations of LIH electromagnetic waves.

The fact of dealing with an *interactive* simulation, simplifies how to focus on the most interesting things. For instance, if the user just wants to see how the attenuation $\alpha[nep/m]$ affects the wave's draw, it is possible to adjust the scrollbars and buttons and to observe only the desired phenomena:

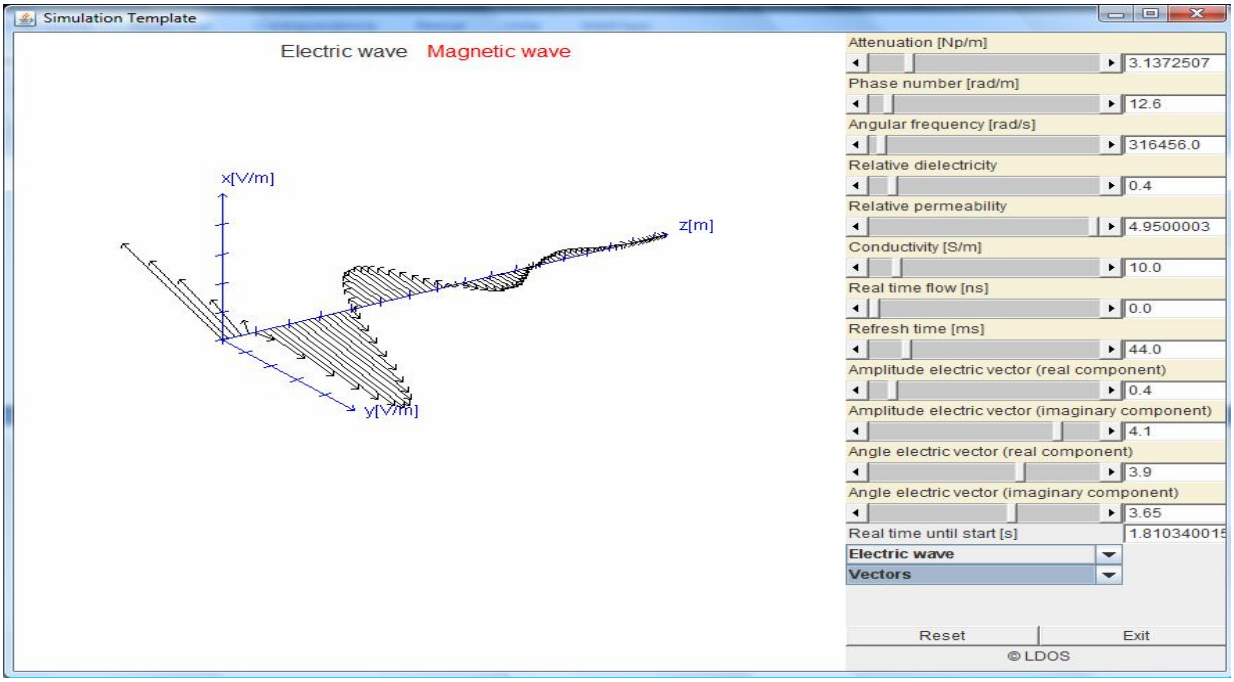


Figure 8.1 Best view for observing how $\alpha[nep/m]$ attenuates the electric wave in vectors style.

Another important and easily observed phenomenon is the relation between the E and H waves in time domain. Although the phasors vectors $\vec{H}_0$ and $\vec{E}_0$ are always perpendicular in complex domain, in time domain they are not perpendicular anymore.

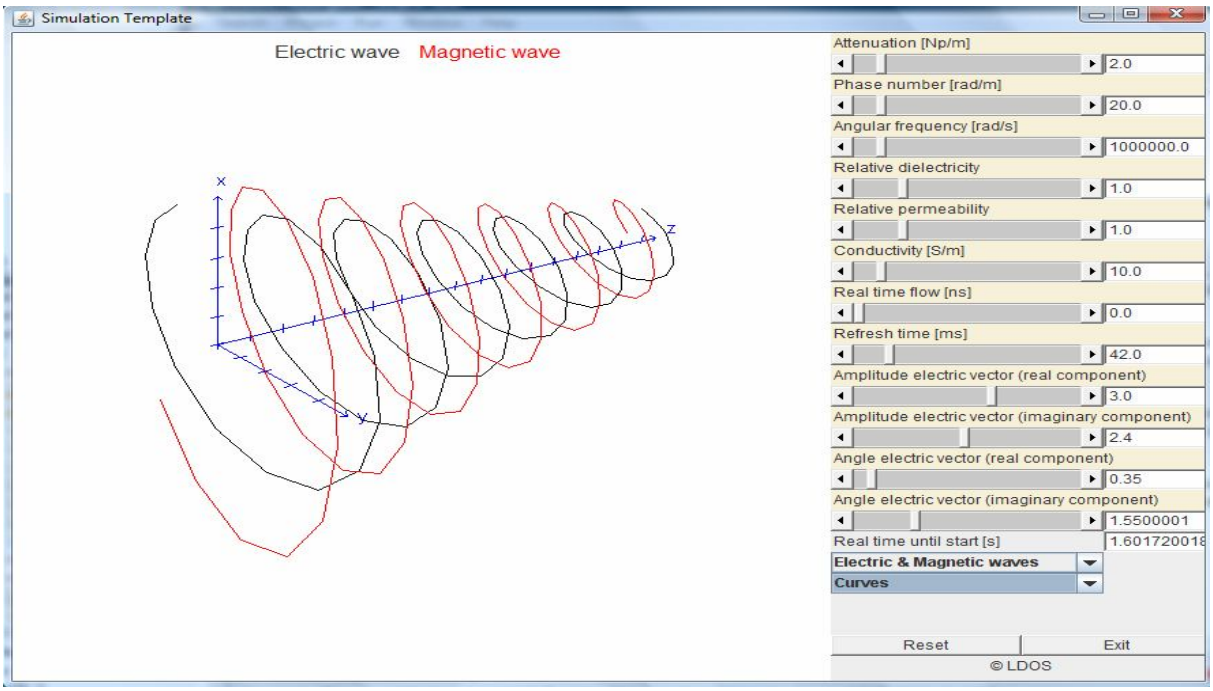


Figure 8.2 $\vec{E}(z,t)$ and $\vec{H}(z,t)$ are not necessarily perpendicular to each other

9. Conclusion and further work

The aim of this applet was to show to students a graphic example of a 3D TEM electric and magnetic wave in time space. It was intended to be an interactive tool so every student could change the equation parameters and check by himself how these changes affected to the draw of the waves. Sometimes it is hard to understand what the parameters of the equations mean. There is no better way to understand them than observing a graphic example.

The previous target was successfully reached. Nevertheless, all programs can be always improved. This applet is not an exception. Some open issues have already been reviewed in chapter 3. Furthermore, some new ideas to improve this program can come up to anybody at any time. For instance, it is used a thread to run continuously the applet and to avoid holding the processor. The use of threads could be avoided by using some programming tricks. In fact, it is certain that the code of this applet could be more efficient, but so far no processor speed issues have appeared on this SimulationTemplate version.

The first version of this applet was programmed to draw a 2D sine. Using the same base structure (MVC concept), a draw of electric and magnetic 3D waves was developed. To achieve this purpose was needed to work basically on:

- The mathematical model equations.
- The draw of the waves and axes on the panel.

Some other features were added to the applet. For instance, *properties files* or *Hot Equations.*

The code of this applet has been widely commented and explained. It can be a perfect point of departure for whoever is interested in simulating interactive mathematical models.

It is important to emphasize that the basic MVC structure is valid for a countless number of simulations. Therefore the programmer can take advantage of it and use the MVC concept as a new applet template.

The *parameter dependencies* feature enhances the applet's importance. It has been programmed a structure of JAVA listeners allowing the interconnection of parameters on sliders. This applet is meant to be a reliable example of real electric and magnetic waves. To get it, it is necessary to apply parameter dependencies. For example, if the slider of dielectricity (of a given material) changes, it must affect the wave number (β) and the attenuation (α) of the waves. This is a specific and very interesting feature available in this applet.

Bibliography

[1] Article Java Advantages & Disadvantages. Arizona community [online]. [Visited 9th January 2009] Available in:
<http://arizonacommunity.com/articles/java_32001.shtml>

[2] Sun Microsystems. Code samples and Applets [online]. [Visited 15th January 2009]. Available in: <http://java.sun.com/applets/>

[3] Sun Microsystems. Java SE Application with MVC [online]. [Visited 12th January 2009]. Available in: < http://java.sun.com/developer/technicalArticles/javase/mvc/>

[4] WENTWORTH, STUART M. (2005). Fundamentals of electromagnetics with engineering applications. USA: John Wiley & Sons, 2005.

[5] The Jakarta Project. Properties files. [online]. [Visited 23rd January 2009]. Available in: <http://commons.apache.org/configuration/howto_filebased.html>

[6] The Jakarta Project. The Java API Documentation Generator. [online]. [Visited 23rd January 2009]. Available in:
<http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/javadoc.html#synopsis>

[7] HotEqn. LaTeX writing Tutorial. [online]. [Visited 22nd January 2009]. Available at:
<http://www.atp.ruhr-uni-bochum.de/VCLab/software/HotEqn/HotEqn.html>

[8] Microwave encyclopedia. Propagation, attenuation and phase constants [online]. [Visited 25th January 2009]. Available at:
<http://www.microwaves101.com/encyclopedia/propagation.cfm**>**

[9] Wave polarization. Example of elliptical polarization [online]. [Visited 25th January 2009]. Available at:
<http://www.ece.byu.edu/em/embook/ch7/demo7.4.html>

[10] Unicode correspondences. Slovenian characters [Online].[Visited 14th February 2009]. Available at: <http://www.geocities.com/click2speak/unicode/chars_sl.html>