

TECHNISCHE UNIVERSITÄT DRESDEN

FACULTY OF COMPUTER SCIENCE

INSTITUTE OF SOFTWARE- AND MULTIMEDIA TECHNOLOGY

CHAIR OF COMPUTER GRAPHICS AND VISUALIZATION

PROF. DR. STEFAN GUMHOLD

Student thesis

Simulation and visualisation of the electro-magnetic
field around a stimulated electron

Annett Ungethüm

(Mat.-No.: 3116073)

Tutor: Prof. Dr. Stefan Gumhold
Dr. Michael Bussmann

Dresden, October 30, 2012

Abstract

A mathematical method introduced by T. Shintake has been adapted and enhanced to compute and visualize electro-magnetic fields around moving charges interactively and in real-time. It is based on two conditions: Firstly, the waves travel with the speed of light. Secondly, their initial radial emission is affected by the charges velocity. Additionally some field quantities, e.g. the wavelength and vectorial dimensions, are computed and visualised. Furthermore alternative ways for a field representation have been implemented.

Contents

| | |
|--|-----------|
| List of symbols | 1 |
| 1 Introduction | 3 |
| 2 The Shintake concept and its further development | 5 |
| 2.1 The basic idea | 5 |
| 2.2 The 3D expansion | 9 |
| 2.3 Post-Shintake: Non-iterative field calculation | 10 |
| 3 Derived field quantities | 15 |
| 3.1 Wavelength and field strength | 15 |
| 3.2 Vectorial dimensions | 16 |
| 3.3 Trajectories | 18 |
| 4 Graphics | 21 |
| 4.1 Colour and alpha mapping | 21 |
| 4.2 The environment | 25 |
| 4.3 Field particles | 25 |
| 4.4 The wave plane | 27 |
| 4.5 Illumination | 28 |
| 5 Implementation | 31 |
| 5.1 The used framework: Ogre | 31 |
| 5.2 Program structure | 31 |
| 5.3 Control flow | 33 |
| 5.4 Specifics in the implementation | 35 |
| 5.5 Conflicts between theory and software layout | 37 |
| 6 Results and discussion | 39 |
| 6.1 Results | 39 |
| 6.2 Discussion | 46 |
| 7 Outlook | 49 |
| Appendix | 51 |
| A User guide | 51 |
| B Blinn-Phong shader in GLSL | 60 |
| C Mathematica scripts and sample points | 61 |
| D Benchmark | 63 |
| References | 65 |

List of symbols

| Symbol | | Symbol | |
|--------------|---|-------------------------|---|
| a | Index, specifies a certain step of the computation | m_e | Rest mass of an electron |
| a | Multiplicative constant, varies the amplitude of the sine trajectory | M | Centre of a wave front |
| B, \vec{B} | Magnetic field, magnitude and vectorial dimension Colour value in RGB space: blue The observer's position | \vec{n} | A normal |
| c | Speed of light | P | A position on a field line A certain particle |
| d | Distance | q | Auxiliary variable for converting form HSV to RGB space |
| \vec{d} | Vector from a point to a light source | q | Constant controlling the particle fading |
| e | Elementary charge Used as index: refers to the actual status of the electron | Q | Charge |
| \vec{e} | Vector from a point to an observer | r | Radius |
| E, \vec{E} | Electrical field, magnitude and vectorial dimension An event The electron's position at a certain event | R | Colour value in RGB space: red |
| f | Frequency Auxiliary variable for converting from HSV to RGB space Flag declaring a position as part of a wave front | s | Distance in space |
| G | Colour value in RGB space: green | S | Position in space Colour value in HSV space: saturation |
| h | Auxiliary variable for converting form HSV to RGB space | t | Time Auxiliary variable for converting form HSV to RGB space |
| h | Planck constant | v | Velocity |
| \vec{h} | Halfway vector | V | Colour value in HSV space: value |
| H | Colour value in HSV space: hue | W | Position on a wave front |
| i | Index specifying a certain position on a field line/a certain step of the computation | α | Angle Alpha value for colours |
| j | Index specifying a certain field line | ε | Permittivity Break condition for numerical algorithms |
| K | Parameter describing the undulator | ϑ | Parameter for sampling a sphere |
| \vec{k} | Wave vector | λ | Wave length |
| | | φ | Parameter for sampling a sphere |
| | | ω | Varies the wave length of the undulator trajectory |
| | | \vec{g} | Vector |
| | | $\hat{g} = \ \vec{g}\ $ | Normalised vector |
| | | $ g $ | The length of a vector |

1 Introduction

Electrodynamics is not rumoured to be very intuitive. Charges which cannot be seen emit radiation which mostly cannot be seen. Furthermore relativistic effects occur when charges move very fast. Therefore models are needed in order to make this topic comprehensible. There are many mathematical models of which some are only applicable for certain cases. But there are only a few graphical models. Traditional representations of electromagnetic fields are static 2D pictures of static fields. They show the electric field as lines. Starting at the charge positions they diverge radially outwards while the magnetic field is pictured by a collection of circles orthogonal to the field lines as shown in fig. 1.1a and 1.1b. As long as the charges move significantly slower than the speed of light this is a sufficient approximation. Changing this condition leads to two major problems: First, the field is not time-invariant anymore. This can hardly be pictured by static images. Second, a third dimension of space is necessary to satisfy the human visual perception. The combination of these two aspects results in a 3D real-time simulation. This is a challenging task which has rarely been undertaken yet. But the range of possibilities for a visualisation is wide. Since electro-magnetic waves cannot be seen by the human eye, reality sets no limits.

In [22] Shintake presents a numerical method for creating real-time simulations of electro-magnetic fields. As the according implementation, “*Radiation 2D*”, is still a 2D visualisation, the basic idea was reused in a practical course of the TU Dresden and the Helmholtz-Zentrum Dresden-Rossendorf (HZDR) to develop a 3D version, “*Radiation 3D*”[15], which is part of a project visualising light-matter-collision experiments performed at the research centre HZDR. This is the starting point for this work.

Previous works

There are several techniques for imaging vector fields in general. *Stream surfaces*[16], *-lines*[25], *-polygons*[20] and *-tubes*[4] are just a selection of techniques which use geometrical elements to visualise vectors directly. Other approaches like Cabral’s and Leedom’s *Line Integral Convolution*[3] or de Leeuw’s and van Wijk’s *Spot Noise*[5] modify textures to map a vector field.

An implementation which uses *Spot Noise* and *Stream Tubes* is *Vapor*[2]. Despite that *Vapor*’s field of application is not simulating electro-magnetic fields but ocean, atmosphere, and solar research, it suffers from some restrictions. Both techniques cannot be animated simultaneously and the *Spot Noise* is reduced to a slice. Besides the FAQs on the homepage support a conclusion that there are difficulties concerning the performance on older systems.

Radiation2D[15] simulates an electro-magnetic field but hardly uses any other visualisation method than simple line strips and circles in 2D space.

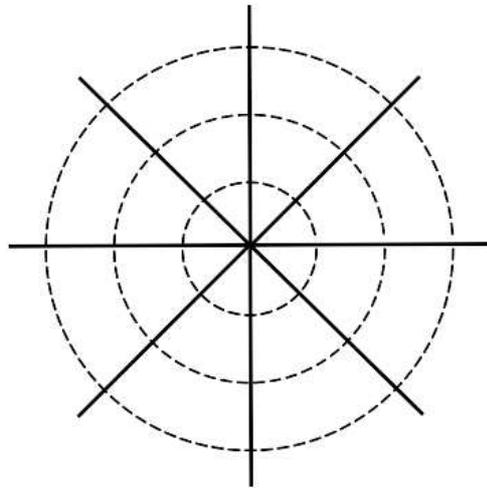
Another program for the simulation of electro-magnetic fields is *FIELDS*[23]. It introduces several potentials and generates a variety of graphical outputs including vector arrows, colour encoded potentials and field strengths and isolines. Even if the functionality is extensive, the user interface is a command line which is far from intuitive, the graphical output is mostly only in 2D and not animated. Moreover, there is no consistent evaluation method for the different field problems, of which only the static dimension can be solved. Depending on the exact problem, the Poisson, Laplace or the vector potential equation is solved.

D. Fleisch presents some selected field representations for educational purposes on his website[9]. They are neither animated nor do they treat a moving charge but they are available as 3D models in *VRML*¹.

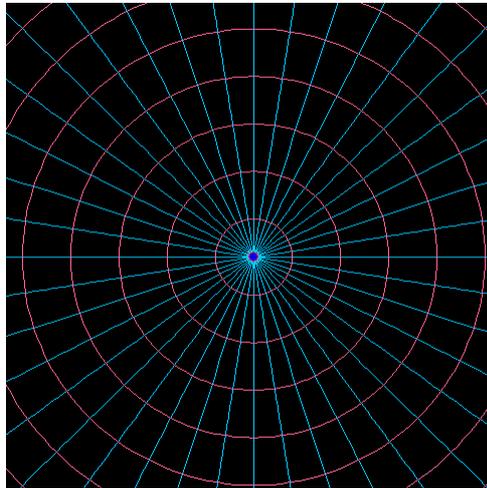
It seems there is no other simulation of fields around moving charges which generates a graphical output in 3D and in real time. The first version of *Radiation3D*[15] already contains some of these points but is still missing some information, e. g. a representation of the field strength and is imprecise in some calculations, e. g. the trajectories.

This work improves and extends *Radiation3D* significantly by adding new functionality and more physical correctness by explaining the used main concept and adding new information to the visualisation. Additional representations which are different from field lines and wave fronts are introduced.

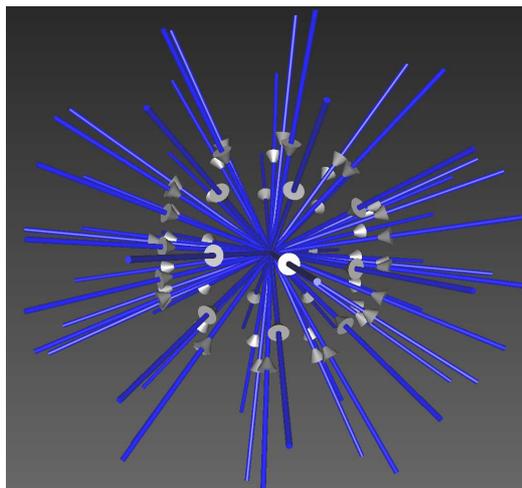
¹ Virtual Reality Modeling Language, a predecessor to X3D



(a) A classical representation of an electromagnetic field around a point charge



(b) A point charge. Representation by T. Shintake[22].



(c) A negative point charge. Representation by D. Fleisch[9].

Fig. 1.1: Different representations of a static field

2 The Shintake concept and its further development

Working with models

For making physical phenomena comprehensible and explainable a variety of models is used. They are no exact images of reality but rather outline the main idea of what we understand about reality. Just like a solid sphere is a very idealised model of the earth, field lines are a coarse grained model for describing electro-magnetic fields. Of course the field also exists where there are no field lines. Wave fronts are also just a model. Waves are continuous but defining certain positions on them, like wave fronts, eases the explanation of different effects like oscillating fields. This work uses the idea of field lines, waves and wave fronts for modeling a comprehensible simulation and visualisation.

2.1 The basic idea

A real time simulation of non static electro-magnetic fields requires the recalculation of the whole field for every frame, ideally more than twenty times per second. Therefore solving Maxwell's equations explicitly is not optimal. They either require solving integrals around closed loops or differential equations for every desired position in space which leads to a massive resource load, if it is done numerically and in a high frequency.

Another approach is introduced in [22]. It treats charges which are separated from each other and therefore do not interact. Furthermore there are two assumptions which are necessary for this concept. The first one is that the information emitted by an electron moves with the speed of light in every frame of reference. Therefore all vectors describing the emitting direction of information have the same length which corresponds to the second postulate of relativity. The second one is that the direction of a vector describing an emitting direction is changed using the aberration of light effect. It is important that only the direction is computed this way since the length is defined by the speed of light. For this purpose the moving vector of the electron \vec{v}_e in respect to the frame of reference of the observer is needed due to the first postulate of relativity.

The basic idea can be pictured as tracing the information propagated by certain events. Since this information moves outwards with the speed of light, all information caused by the same event build a perfect sphere. Hence, the fronts of the emitted waves also build a sphere expanding with the speed of light c and starting at the position of the charge at the time it was emitted. In the implementation [22] this was treated as a 2D problem.

Fig. 2.3 on page 7 shows a possible scenario where the charge follows a sine curve. For the following explanations the position vectors $O\vec{P}_i$ are renamed as P_i . The detail shows several selected field lines and discrete points P_i on it. The spheres W_i represent the information related to the same event. W_3 and W_6 are also wavefronts since they belong to events happening at the local extrema of the curve. Chapter 2.2 will explain how to find these extrema.

For computing field lines, these discrete positions P_i on the spheres must be found. To get some initial values, the actual information is emitted radially in a user-defined resolution if the charge does not move, which builds a unit circle in 2D. Otherwise this direction is changed by adding the actual moving vector of the charge \vec{v}_e . For also obtaining the distance to where the information is emitted, the resulting vector, the so-called wave vector \vec{k} , is normalised and multiplied with c and the elapsed time Δt since the according event was recognised. This follows trivially from $v = \Delta s / \Delta t$. Since the speed v equals c , the covered distance is derived by $\Delta s = c \cdot \Delta t$.

Fig. 2.1 graphically shows a transformation of the vector direction. It illustrates the transformation of the direction being nothing more than a vector addition between the moving vector \vec{v} and the transformed vector \vec{k}' weighted with a factor.

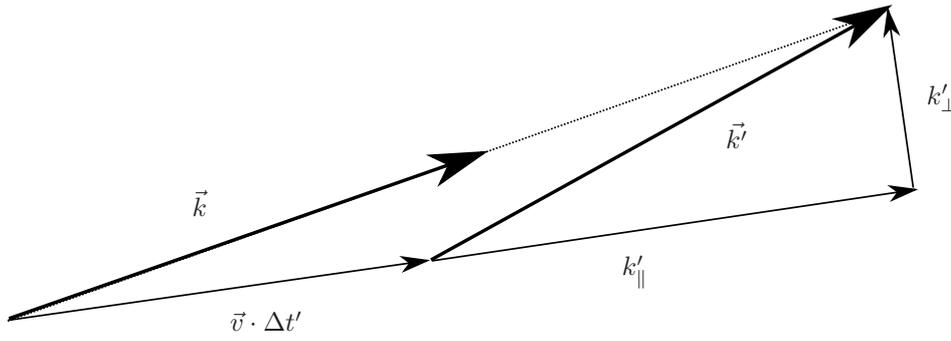


Fig. 2.1: A transformed vector

The Lorentz transformations for the vector's parallel and perpendicular part in respect to the moving vector are

$$\begin{aligned} k_{\perp} &= k'_{\perp} \\ k_{\parallel} &= \gamma(k'_{\parallel} + v \cdot t') \end{aligned}$$

with

$$\gamma = \frac{1}{\sqrt{1 - v^2/c^2}}$$

Obviously the parallel part needs to be weighted with γ to make the transformation Lorentz compliant. As long as the magnitude of the velocity stays the same, γ is a constant. Hence this issue is reduced to a matter of scaling.

At every newly recognised event, the emitted field is computed. It travels forward into the direction of the wave vector, so that a single fieldline can be evaluated iteratively by

$$P_i = P_{i-1} + c \cdot \Delta t \cdot \vec{k}_{i-1} \quad (2.1)$$

Where i is an iterator for the positions on the fieldline, P_i the positions themselves and \vec{k}_i the normalised wave vectors. P_0 is the actual position of the charge and

$$\vec{k}_0 = \|(\cos \varphi, \sin \varphi)^T + \vec{v}_e\|$$

φ is the angle at which the field line is emitted during its initialisation.

Assuming that this has to be done for all field lines, another index j is introduced to count them.

$$P_{i,j} = P_{i-1,j} + c \cdot \Delta t \cdot \vec{k}_{i-1,j}$$

Fig. 2.2 illustrates an example of the field of a resting electron with eight field lines, each of them containing their origin and three more positions.

Fig. 2.4 shows the time dependent evolution of a field line. The electron's velocity changes distinctly every time to illustrate a curved line with only a few steps. For the explanation every step is divided into two parts. The first one is emitting the part of the field into the direction \vec{k}_i . The second one is moving the electron forward into its moving direction \vec{v}_e while shifting \vec{k}_i one step forward and generating a new \vec{k}_0 . While there is only a single field line treated, fig. 2.3 shows the field of a charge that has already moved along a sine trajectory, with several field lines and wave fronts.

Fig 2.5 shows a resulting fieldline and indicates the shifting of the wave vectors \vec{k}_i .

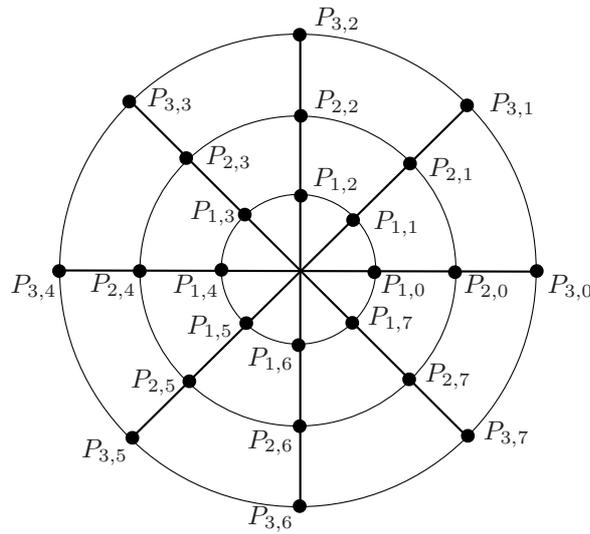
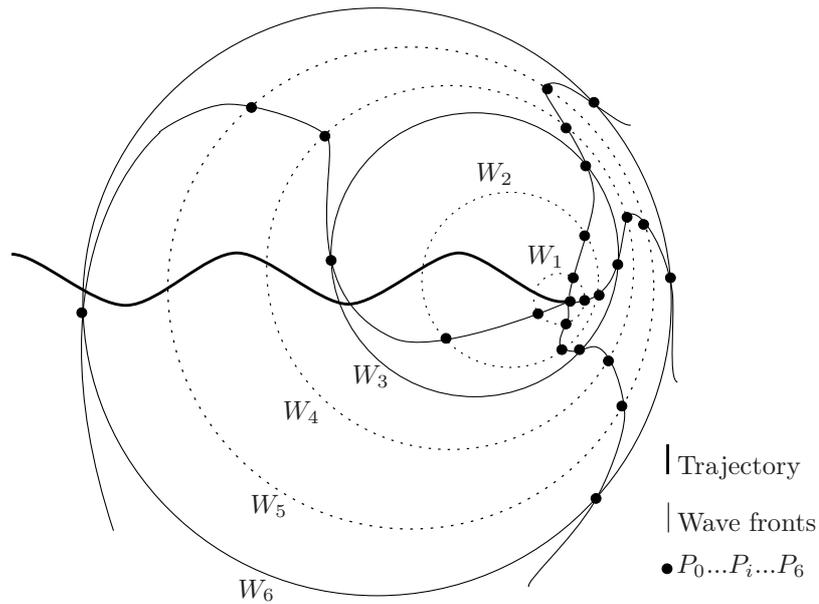


Fig. 2.2: A simple illustrative example: a resting electron represented by eight field lines. At the electron's position, the index i is 0 for every field line.



(a) The field of a charge which is following a sine curve, showing field lines, wave fronts and the trajectory.



(b) A detail of fig a) showing some selected positions to where the field is emitted and the spheres W_i connecting the information P_i which are related to the same event.

Fig. 2.3: An example of a field around a charge which is following a sine curve.

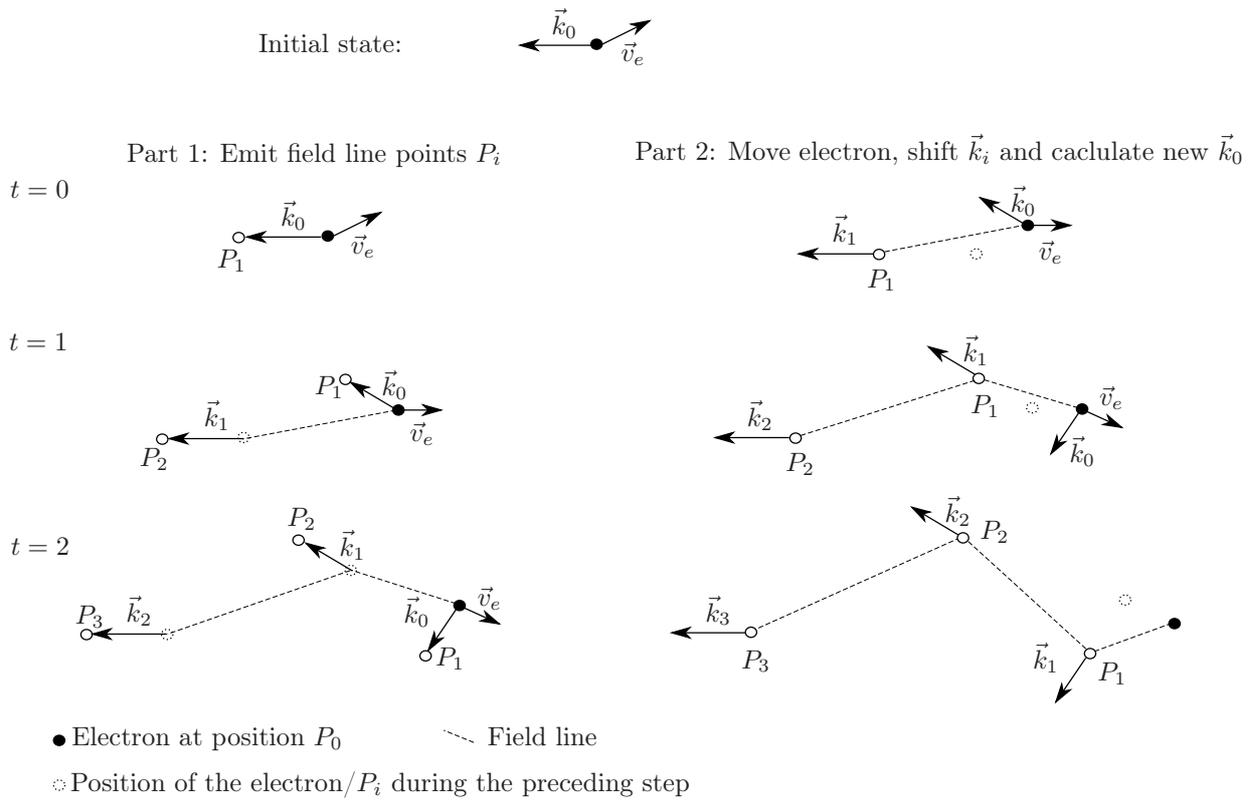


Fig. 2.4: The development of a single field line

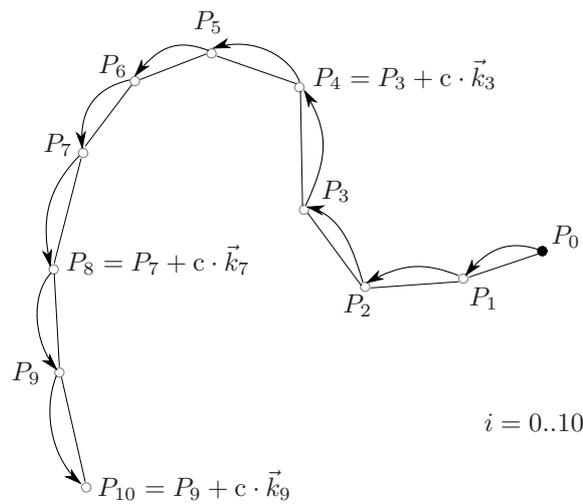


Fig. 2.5: A single field line computed using the Shintake method. The wave vector \vec{k} is being shifted forward in every step.

2.2 The 3D expansion

Generating the wave vectors

The described concept requires some extensions to work in 3D. The basic idea remains the same. However, as the circle providing the initial wave vectors is insufficient for the problem because it is a 2D object, the initial emitting directions \vec{k}_0 in 3D space are derived from a unit sphere, which can be defined by polar coordinates.

$$\vec{k}_0 = \left\| \begin{pmatrix} \sin \varphi \cos \vartheta \\ \sin \varphi \sin \vartheta \\ \cos \varphi \end{pmatrix} + \vec{v}_e \right\| \quad \begin{array}{l} \varphi[0, \pi] \\ \vartheta[0, 2\pi] \end{array} \quad (2.2)$$

Where φ and ϑ implement the resolution in horizontal and vertical direction.

Recognising wave fronts

Recognising local extrema to find the emitted wave fronts might be done analytically if the whole trajectory of the charge is known. If it is unknown, e. g. created by user interaction, this is not possible. And even numerical methods like *Newton*, *Gauss* or *Levenberg-Marquardt* are likely to fail, because not enough information is being provided at the time a wavefront is emitted. For a real-time visualization it is essential to recognise these extrema shortly after they have been passed. The mentioned methods require some knowledge about the curve at both sides of the extremum. But this information is not existent since only the part that was already passed is known. Besides, these algorithms would cause additional resources.

For the 3D implementation a fast and simple but adequate solution is being used. A wave front is being emitted when the moving direction of the charge changes the octant, which equals a change of the slope of the curve. The following example illustrates this method in 2D, using quadrants instead of octants.

$f(x) = (x - 2)^2 + 1$ has a local minimum at $(2, 1)^T$. Assuming that the charge moves along this curve and is recognised at $x_1 = 1.69$, $x_2 = 1.96$ and $x_3 = 2.25$, the resulting vectors are

$$\begin{aligned} \vec{v}_1 &= \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1.96 \\ 1.0016 \end{pmatrix} - \begin{pmatrix} 1.69 \\ 1.0961 \end{pmatrix} = \begin{pmatrix} 0.27 \\ -0.0945 \end{pmatrix} \\ \vec{v}_2 &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 2.25 \\ 1.0625 \end{pmatrix} - \begin{pmatrix} 1.96 \\ 1.0016 \end{pmatrix} = \begin{pmatrix} 0.29 \\ 0.0609 \end{pmatrix} \end{aligned}$$

Thus, \vec{v}_1 is in quadrant IV and \vec{v}_2 in quadrant I, like shown in fig. 2.6. This implies that a wavefront was emitted near $x_2 = 1.96$, which is indeed close to the analytically found solution $x_0 = 2$.

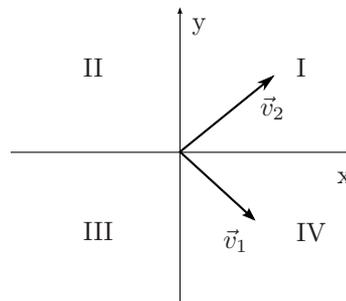


Fig. 2.6: Moving vectors in different quadrants

This comparison is done for every iterative step and can easily be adapted in 3D. There is always the sign of the actual (\vec{v}_a) and the preceding moving direction (\vec{v}_{a-1}) used. Hence, the flag of an emitted wave front is set when the following term is true.

$$\text{sgn}(\vec{v}_{a,x}) \neq \text{sgn}(\vec{v}_{a-1,x}) \vee \text{sgn}(\vec{v}_{a,y}) \neq \text{sgn}(\vec{v}_{a-1,y}) \vee \text{sgn}(\vec{v}_{a,z}) \neq \text{sgn}(\vec{v}_{a-1,z})$$

with

$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x = 0 \\ \frac{x}{|x|}, & \text{if } x \neq 0 \end{cases}$$

This flag f is added to the position P_0 of any fieldline j and iterated in the same way as the wave vector \vec{k} . Additionally the actual position of the charge is cached when f is true, because it is situated in the middle M of the sphere representing the corresponding wave front. The case in which the charge follows an axis of the coordinate system and then changes into any axis' positive direction is not treated due to its rare occurrence. In these cases more wavefronts than necessary are emitted. The change into a negative direction is treated by the definition.

The vertices $W_{i,\varphi,\vartheta}$ for the wave front drawn are then computed in the following way.

$$W_{i,\varphi,\vartheta} = |\overrightarrow{P_{f,i,j}M_i}| \cdot \begin{pmatrix} \sin \varphi \cos \vartheta \\ \sin \varphi \sin \vartheta \\ \cos \varphi \end{pmatrix} \quad \begin{matrix} \varphi[0, \pi] \\ \vartheta[0, 2\pi] \end{matrix} \text{ for } f = \text{true}$$

Where j is the index of the fieldline in which the flag f is stored.

2.3 Post-Shintake: Non-iterative field calculation

The Shintake approach is a fast way for computing the visualisation but it produces some restrictions. If the number of field lines is changed, every new line has to be built from the very beginning. There is no possibility of computing the whole field at a certain time, only the parts which already exist can be shown.

There might be situations when a user is interested in a special region of the field which therefore should be drawn in a high resolution but not the remaining field. This is hardly practicable if the region of interest changes during one simulation.

Assuming the latest movements of the charge are known, these limitations are not mandatory. The directions \vec{k} into which the information is spread, e. g. the wave vectors, always stay the same for the same event. In the iterative method this direction is multiplied by the time that has passed since the last step and the speed of light. Afterwards it is added to the corresponding position on the field line. This way, the next position on the field line is calculated. At the end of the iterative step, the direction used for deriving the new position is associated with it. If the directions and amounts of elapsed time as well as all the positions of the charge were cached, the position $P_{a,t}$ after the iterative step t is given by

$$P_{a,t} = \sum_{i=0}^a \Delta t_{t-a+i} \cdot c \cdot \vec{k}_{i,t-a+i} + P_{i,t-a+i} \quad ,$$

where a is the index for the position on a given field line. Thus, P_a is a point P on a given field line during the iterative step a . This simply means that (2.1) on page 6 is repeated and sums up for all the passed detected times and positions since the time a certain event has happened:

$$\begin{aligned} P_{a,t} &= (\Delta t_{t-a+0} \cdot c \cdot \vec{k}_{0,t-a+0} + P_{0,t-a+0}) \\ &+ (\Delta t_{t-a+1} \cdot c \cdot \vec{k}_{1,t-a+1} + (\Delta t_{t-a+0} \cdot c \cdot \vec{k}_{0,t-a+0} + P_{0,t-a+0})) \\ &+ (\Delta t_{t-a+2} \cdot c \cdot \vec{k}_{2,t-a+2} + (\Delta t_{t-a+1} \cdot c \cdot \vec{k}_{1,t-a+1} + (\Delta t_{t-a+0} \cdot c \cdot \vec{k}_{0,t-a+0} + P_{0,t-a+0}))) \\ &+ \dots \end{aligned}$$

Because of the iteration, for \vec{k} follows

$$\vec{k}_{i,t-a+i} = \vec{k}_{i-1,t-a}$$

Therefore the term above reduces to

$$P_{a,t} = c \cdot \vec{k}_{0,t-a+0} \cdot (\Delta t_{t-a+0} + \Delta t_{t-a+1} + \Delta t_{t-a+2} + \dots) + P_{0,t-a+0}$$

with

$$\Delta t_{t-a+0} + \Delta t_{t-a+1} + \Delta t_{t-a+2} + \dots = \sum_{i=0}^a \Delta t_{t-a+i}$$

which is the time Δt_{t-a} passed since the moment the corresponding part of the field has been emitted. Hence, every position is given by

$$P_{a,t} = c \cdot \vec{k}_{0,t-a} \cdot \Delta t_{t-a} + P_{0,t-a} \quad (2.3)$$

As $P_{0,t-a}$ is the position of the electron after $t-a$ steps and as there are several field lines, the equation transforms to

$$P_{a,t,j} = c \cdot \vec{k}_{0,t-a,j} \cdot \Delta t_{t-a} + E_{t-a} \quad (2.4)$$

where j is the index counting the field lines and E_{t-a} is the position of the electron at step $t-a$.

\vec{k}_0 can be computed by applying (2.2).

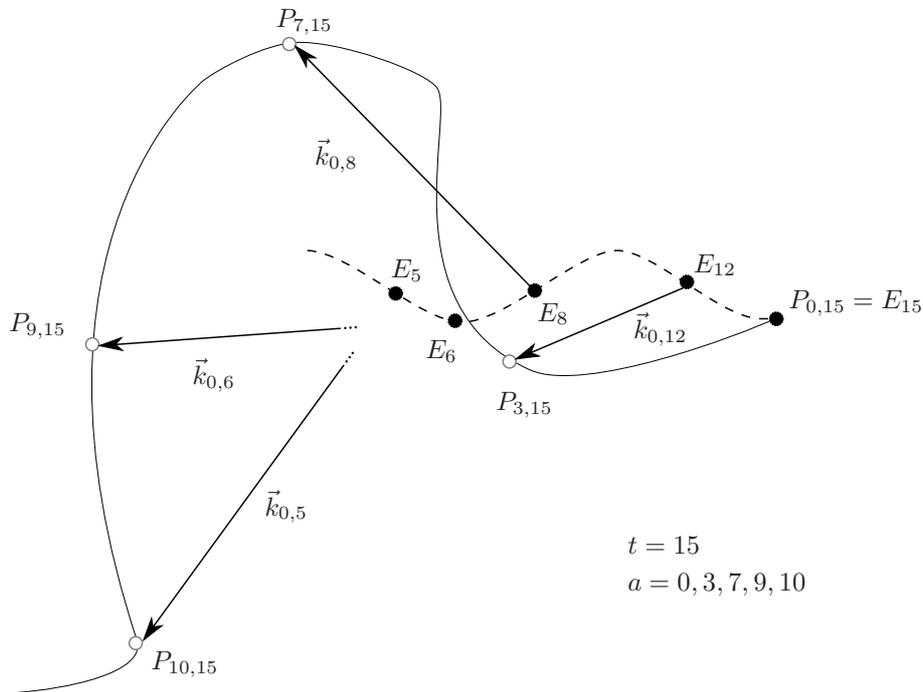


Fig. 2.7: A single field line computed using the Post-Shintake method. In this case the total amount of cached steps is 15. Only selected positions are illustrated.

The method can also be explained graphically. Laying several successive steps of a Shintake calculation on top of each other, it becomes obvious that the wave vector \vec{k}_0 is just a path line and the field lines are streak lines. Fig. 2.8 illustrates this using the steps from fig. 2.4.

The following algorithm returns as many fieldlines as desired.

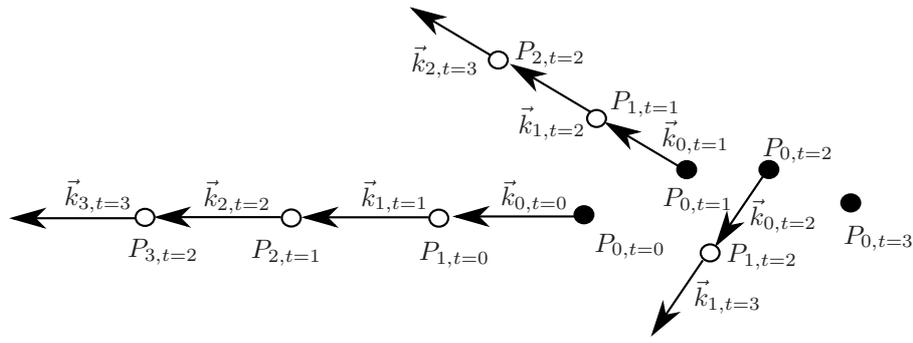


Fig. 2.8: All wave vectors \vec{k}_i and positions P_i from fig. 2.4 on page 8

Algorithm 1 Calculate fieldlines

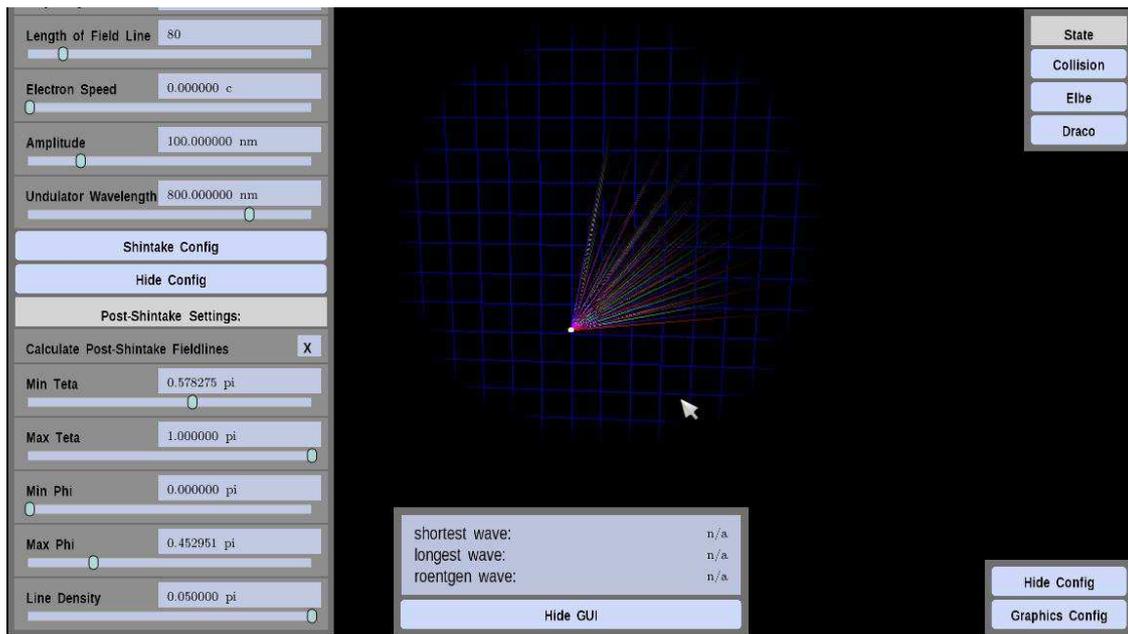
```

fieldlineIterator=0
fieldlines.clear
for  $\theta_{\min}$  to  $\theta_{\max}$  do
   $k_0.z = \cos(\theta)$ 
  for  $\varphi_{\min}$  to  $\varphi_{\max}$  do
     $k_0.x = \sin(\theta) \cdot \cos(\varphi)$ 
     $k_0.y = \sin(\theta) \cdot \sin(\varphi)$ 
     $a = 0$ 
    while fieldlines[fieldlineIterator].size <  $t$  do
      fieldlines[fieldlineIterator].pushback( $c \cdot \Delta t_t \cdot \|(\vec{k}_0 + direction_{t-a+1})\| + P_{t-a+1}$ )
       $a = a + 1$ 
    end while
    fieldlineIterator=fieldlineIterator + 1
  end for
end for
return fieldlines

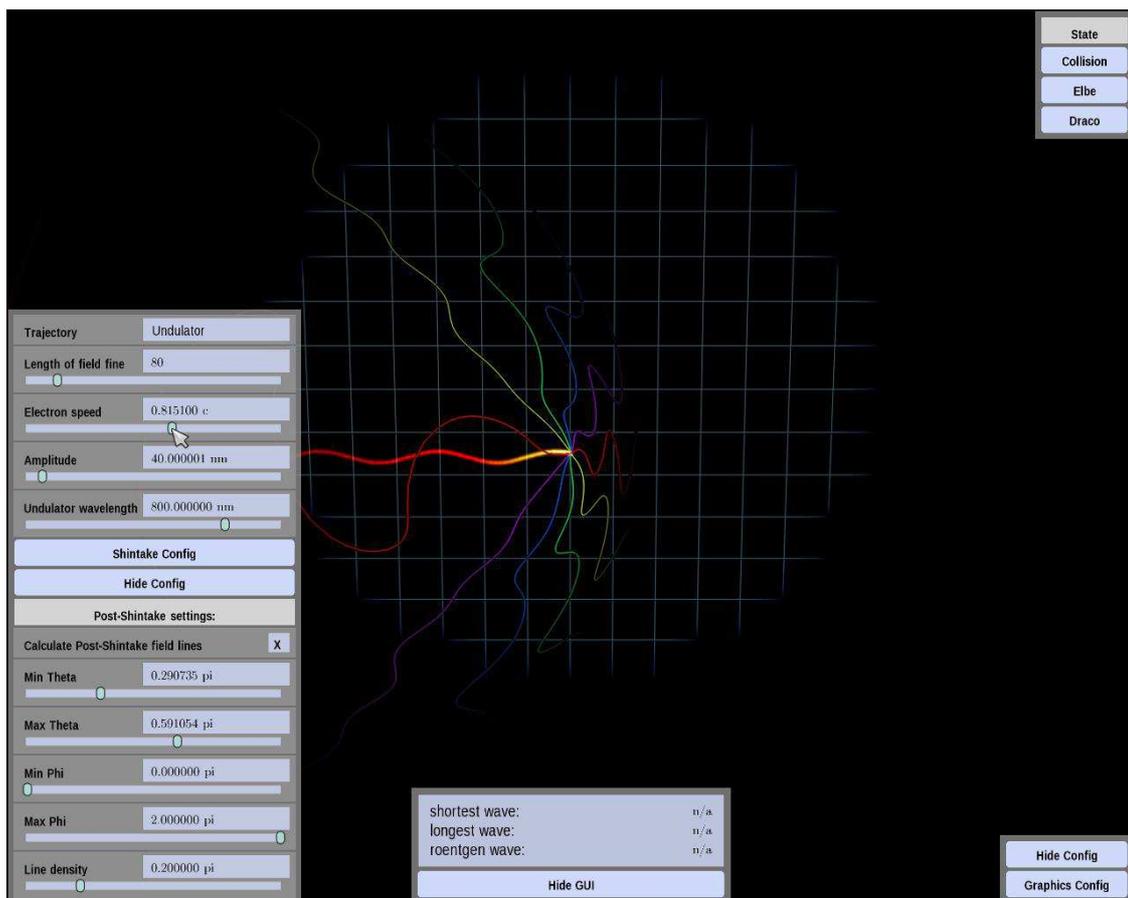
```

Fieldlines is a vector containing another vector for each fieldline.

This iterative version needs more memory for storing all positions and directions for every field line while the non-iterative version only stores the positions on the trajectory. The directions \vec{v}_e and times t_i can either be stored, too or calculated using the array size of the positions and the vectors between them. However, the amount of calculated values is the same in both methods. Nevertheless the Post-Shintake algorithm offers the opportunity to change the resolution and to select the drawn section in real-time (see fig. 2.9). Repeating this algorithm with different constraints even allows to show different sections in a different resolution, e. g. in respect to the regions of interest of the field.



(a)



(b)

Fig. 2.9: The post shintake method enables the user to choose a certain sector or even a certain plane to be drawn.

3 Derived field quantities

3.1 Wavelength and field strength

Revisiting the example of the charge moving along the sine trajectory (fig. 2.3), it becomes obvious that the detected wavelength depends on the position of the observer in relation to the trajectory. For instance, if the observer is within an area the electron is approaching, the wavefronts will occur in a higher frequency than for an observer on the opposite site. The time period between two wavefronts and therefore their distance to each other can easily be calculated. It equals the difference between the time both wavefronts need to reach the observer plus the passed time between their emission.

Assuming that the wavefronts are emitted at positions P_1 and P_2 and times t_1 and t_2 and B is the position of the fixed observer, the time difference Δt is calculated from

$$\frac{|\overrightarrow{P_2 B}|}{c} + (t_2 - t_1) - \frac{|\overrightarrow{P_1 B}|}{c} = \Delta t \quad (3.1)$$

From $v = s/t$ follows

$$\Delta s = c \cdot \Delta t \quad (3.2)$$

Where Δs is the distance between the wavefronts detected at B. Because Δs is a half period, from (3.1) and (3.2) follows

$$\begin{aligned} \left(\frac{|\overrightarrow{P_2 B}|}{c} + (t_2 - t_1) - \frac{|\overrightarrow{P_1 B}|}{c} \right) \cdot c &= \frac{\lambda}{2} \\ \Rightarrow |\overrightarrow{P_2 B}| + c(t_2 - t_1) - |\overrightarrow{P_1 B}| &= \frac{\lambda}{2} \end{aligned}$$

Fig. 3.1 and fig. 3.2 illustrate the ambiguity of other geometrical methods for deriving the wave length as there are different possibilities where to measure the distance between wave fronts. While fig. 3.1 shows two different ways for measuring the distance on the line from the position of an event to the observer, fig. 3.2 illustrates the distances between the intersection of several wave fronts and a field line. In the first case the distances d_1 and d_2 are obviously not identical and in the latter, the position of the observer is even completely disregarded.

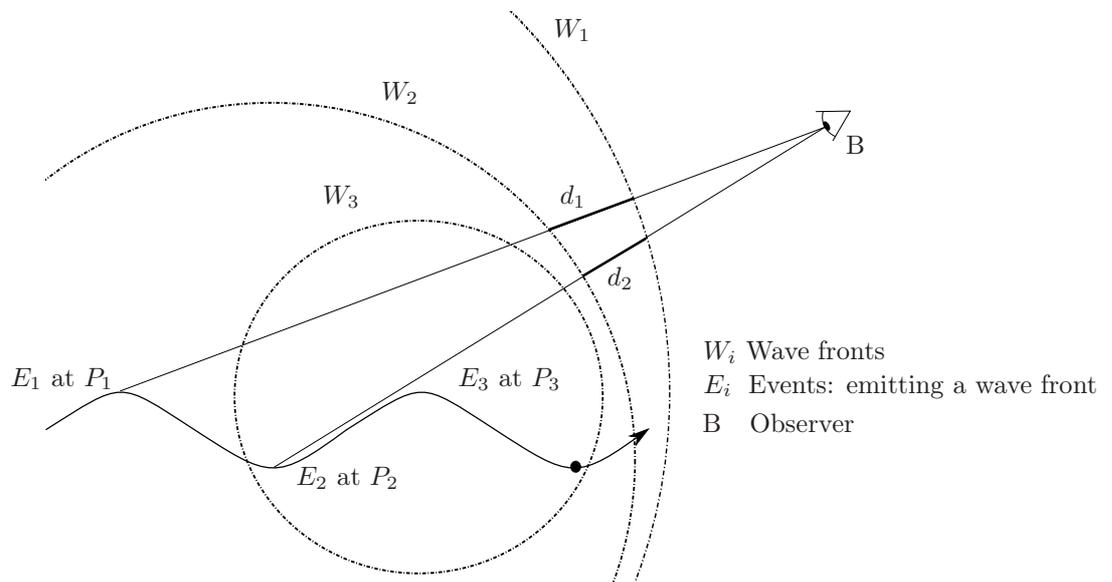


Fig. 3.1: This figure shows why the recognised wave length is not simply the distance between two wave fronts.

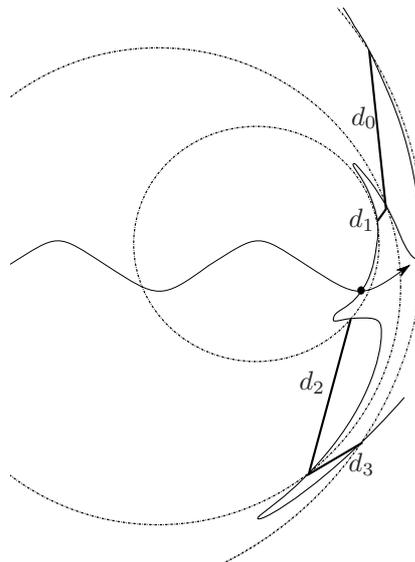


Fig. 3.2: By only measuring the distances between the intersections of wave fronts and field lines, the observer's position is disregarded.

It should be mentioned that this is a theoretical problem. In reality the observer is always situated in the far zone where a defined viewing direction exists. Additionally, in real experiments a whole spectrum of frequencies is observed. In the simulation this spectrum can be derived by recording the observed frequencies during a defined time period.

The field strength E in a distance r around a resting point charge is given by Coulomb's law:

$$E = \frac{Q}{4\pi\epsilon_0 r^2} = \frac{e}{4\pi\epsilon_0 r^2}$$

where ϵ_0 is the vacuum permittivity. As only a single electron is regarded, the charge Q equals the elementary charge e .

Since the field lines are generated using (2.2) and the resolution for φ and ϑ are equal in the implementation, they are equidistant as long as the charge is not moving and therefore their distance d from each other can trivially be calculated.

A way for computing the electric field strength's magnitude of a moving charge explicitly is looking at the distances between the electric field lines. Their density in relation to the total amount of field lines describes E . In the implementation the total amount of field lines is defined by the user and therefore it is a known variable. This provides the needed values for solving the equation

$$\frac{d_2}{d_1} = \frac{E_2}{E_1}$$

where E_1 is the field strength around the resting charge in a defined distance r to the charge Q and d_1 is the distance between the field lines for this case. d_2 is any measured distance between two field lines.

d_2 is always measured between points of the same iterative step like shown in fig. 3.3. Either the arc length or the secant can be used as they are proportional to each other.

This procedure provides a safe way for calculating the electric field strength.

3.2 Vectorial dimensions

The field is not only characterised by a magnitude but also by the direction of the electric (\vec{E}) and magnetic field (\vec{B}). \vec{B} , \vec{E} and the wave vector \vec{k} build a right-handed trihedron. \vec{B} and \vec{E} are perpendicular to each other and since \vec{k} points into the same direction as the resulting poynting vector which represents the energy flow[17], it is computed by

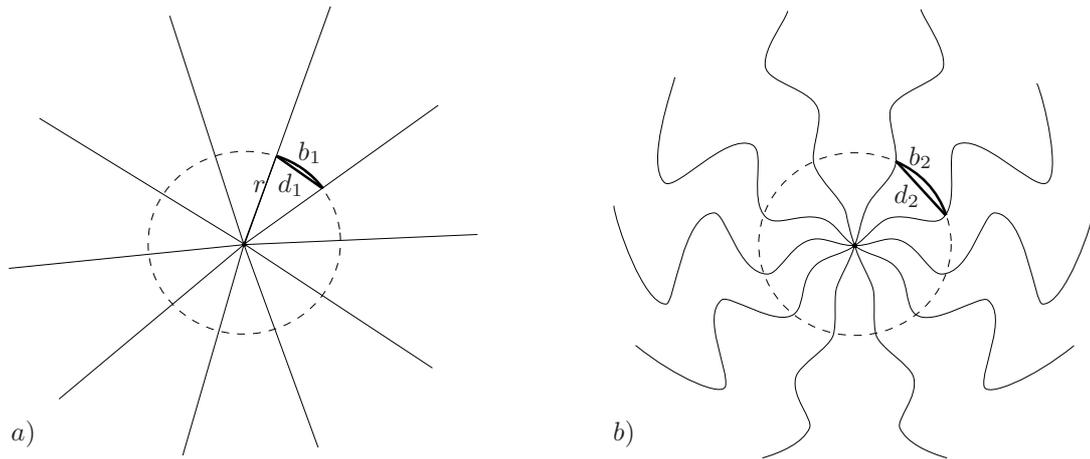


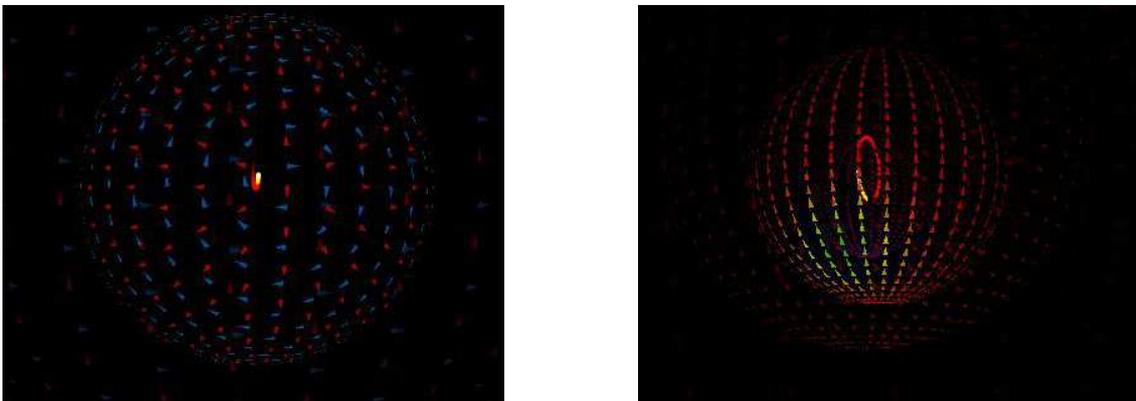
Fig. 3.3: Fig. a) shows a resting electron, fig. b) an electron which has already moved along a dipol trajectory. Both cases must be represented by the same amount of fieldlines. d_i represent the linear connection between two vertices resulting from the same iterative step, b_i represent the corresponding arc length.

$$\|\vec{E} \times \vec{B}\| = \hat{k}$$

Assuming \vec{E} is demanded at a certain point on a field line, \hat{k} and \vec{B} are necessary. The wave vector is computed using the Post-Shintake-Method (chapter 2.3). The magnitude E is also known and its computation explained in the previous subchapter. The direction of \vec{B} is still unknown but building the perpendicular on \hat{k} and the corresponding moving vector of the electron \vec{v}_e results in a vector pointing into the direction of \vec{B} . Thus the electric field and the direction of the magnetic field follow from

$$\begin{aligned}\hat{B} &= \|(\hat{k} \times \vec{v}_e)\| \\ \vec{E} &= E \cdot \|(\hat{k} \times \hat{B})\|\end{aligned}$$

A necessary condition for the applicability of \vec{k} as the emitting direction is that \vec{k} must have the same length as for moving vectors with the speed of light c , which is one unit of length in the implementation.



(a) A wave front emitted by an electron on a circle trajectory. (b) If only the electrical or magnetic field is drawn on the wavefronts, the wavelength is mapped via colour. See 4.1 for a description of the colour mapping. The blue arrows refer to the directions of the B-field while the red arrows refer to the E-field.

Fig. 3.4: The directions of the fields mapped on the wave fronts

3.3 Trajectories

The trajectories for this implementation are defined explicitly, not indirectly via the forces affecting the electron. While this is trivial for a linear and a dipol trajectory, the undulator and circle trajectories pose a challenge. Therefore, they are explained in more detail.

Undulator

When a moving electron crosses an alternating magnetic field, it starts oscillating. Depending on the actually emitted synchrotron radiation this is either called an undulator or a wiggler. To differentiate between them, mainly the dimensionless parameter K is used in literature.

$$K = \frac{eB\lambda}{2\pi m_e c} \quad ,$$

where e is the particle charge, B the magnetic field strength, λ the undulator's period, m_e the electron rest mass and c the speed of light. The description "wiggler" corresponds to $K > 1$ and "undulator" to $K \leq 1$. The latter deflects the electron less than the first. If $K < 1$ the energy of an electron in the undulator is not significantly higher than its rest energy. This causes the electron to move on a sine-function defined by

$$f(x) = a \cdot \sin(\omega x)$$

For reasons of efficiency the function is defined in the x-y-plane. If $K > 1$ the energy of the electron in the undulator is much higher than the rest energy of an electron. Hence, it does not keep the harmonic trajectory but follows a trajectory containing various harmonics.

k may be a sufficient value for describing an undulator but instead, for the sake of comprehensibility to non-specialists, the period, amplitude and speed of the trajectory are adjustable in the implementation. The constant a describes the amplitude, the angular frequency ω follows from the undulator's period λ with

$$\omega = \frac{2\pi}{\lambda}$$

Hence, the challenge consists of finding x for a given speed so that the distance passed remains constant for $v \cdot \Delta t$. The analytical solution would require a derivate of the sine function

$$\int_{x_i}^{x_{i+1}} \sqrt{1 + (f'(x))^2} dx = s = v \cdot \Delta t \quad ,$$

where v is the speed of the electron, Δt the time between the computational steps, x_i the x -component of the electron's actual position and x_{i+1} the x -component of the position which is to be computed. Solving the integral results in the very unhandy term

$$v \cdot \Delta t = \left[\frac{\sqrt{a^2 \omega^2 \cos(2\omega x) + a^2 \omega^2 + 2 E(\omega x) \frac{a^2 \omega^2}{a^2 \omega^2 + 1}}}{b \sqrt{\frac{a^2 \omega^2 \cos(2\omega x) + a^2 \omega^2 + 2}{a^2 \omega^2 + 1}}} \right]_{x_i}^{x_{i+1}} \quad ,$$

where E describes an elliptical function.

Obviously this will not lead to an applicable equation. Therefore the new electron position is found numerically.

The bisection method is a simple but adequately fast and robust method for this purpose. Fig. 3.5 shows an exaggeratedly coarse grained approximation of the curve. The boundaries are easily found by regarding the gradients of the function. Their absolute value is always 0 in the extremas. At these positions the new Δx is calculated by $\Delta x = v \cdot \Delta t$ since the electron moves straight forward into the x -direction. This is the upper limit for the first step of the bisection. In the roots the magnitude of the gradient m is at a maximum of

$$\frac{\Delta y}{\Delta x} = m = f'(0) = a \omega \cdot \cos(\omega \cdot 0) = a \omega$$

Thus the lower limit follows from

$$\begin{aligned}\sqrt{\Delta x^2 + \Delta y^2} &= v \cdot \Delta t \\ \sqrt{\Delta x^2 + \Delta x^2 a^2 \omega^2} &= v \cdot \Delta t \\ \Delta x = \sqrt{\frac{(v \cdot \Delta t)^2}{1 + a^2 \omega^2}} &= \frac{v \cdot \Delta t}{\sqrt{1 + a^2 \omega^2}}\end{aligned}$$

Fig. 3.6 illustrates the first two steps. The actual bisection is done as follows

Algorithm 2 Calculate next position on a sin-function

```

 $x_{\text{left}} = x_i + (v \cdot \Delta t) / \sqrt{1 + a^2 \omega^2}$ 
 $x_{\text{right}} = x_i + v \cdot \Delta t$ 
while  $|v \cdot \Delta t - s| > \varepsilon$  do
   $x_m = (x_{\text{left}} + x_{\text{right}}) / 2$ 
   $y_m = f(x_m)$ 
   $s = \sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}$ 
  if  $s < v \cdot \Delta t$  then
     $x_{\text{left}} = x_m$ 
  else
     $x_{\text{right}} = x_m$ 
  end if
end while
return  $x_m$ 

```

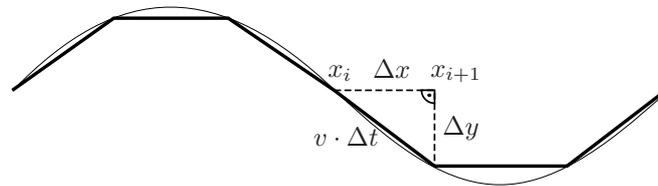


Fig. 3.5: An exaggeratedly coarse grained approximation of a sine curve

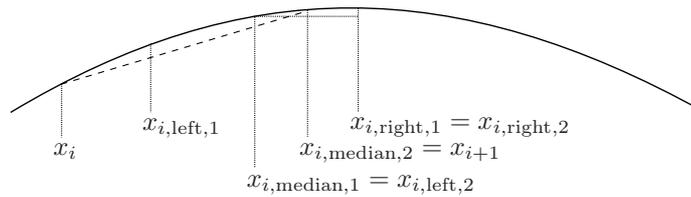


Fig. 3.6: Approximating the sine trajectory

Nevertheless it is an interesting question what happens if this adjustment is not applied, such that only the forward movement of the charge is set to a value close to c . In this case in the simulation the resulting speed of the electron may become larger than the speed of light. Hence, it travels faster than its emitted field causing loops in the fieldlines as seen in fig 3.7. The same effect occurs when an electron moving along a dipole trajectory is faster than light.

Obviously here is a conflict with Maxwell's equations. They are only applicable if the vector field representing \vec{E} is well-defined at any point. But for instance, at position S the field points into different directions without containing a charge, e.g. a source or a sink for field lines. While this is no issue of interest in reality, it is very well possible in a simulation and needs to be avoided.

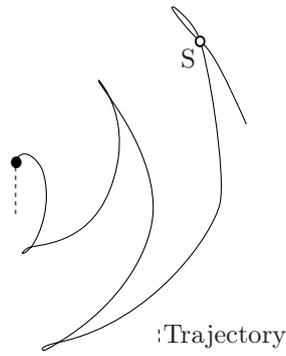


Fig. 3.7: An electron moving along an undulator or dipole trajectory emits looped field lines if it is accelerated faster than the speed of light.

Circle

The circuit trajectory is approximated by very short secants. Again the covered distance is $v \cdot \Delta t$. Hence, the according central angle α_{a+1} is calculated by

$$v \cdot \Delta t = 2r \sin\left(\frac{\alpha_{a+1} - \alpha_a}{2}\right)$$

$$\Rightarrow \alpha_{a+1} = \alpha_a + 2 \arcsin\left(\frac{v \cdot \Delta t}{2r}\right)$$

Where r is the radius of a specified circle. It is adjustable in the implementation at runtime. The according position of the electron is now

$$\vec{0P} = \begin{pmatrix} r \cos \alpha \\ r \sin \alpha \\ 0 \end{pmatrix}$$

For reasons of simplicity the circle is defined in the x-y-plane. If instead only the x-component is computed via $v \cdot \Delta t$, the resulting speed for the simulation is faster than light. This way the electron would leave the field it emits completely behind resulting in an area within the circle where there is no field at all.

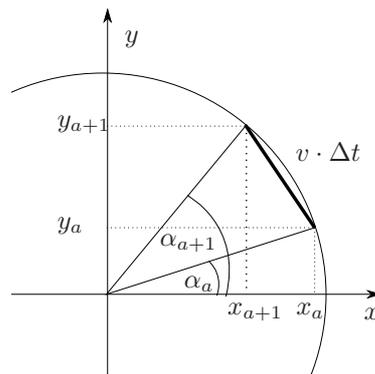


Fig. 3.8: Approximating the circle trajectory

4 Graphics

4.1 Colour and alpha mapping

Saturation, hue, value and opacity are excellent tools for visualising the computed data from the predecesing chapters not only for the sake of providing an enjoyable appearance but also for illustrating physical quantities in a comprehensible and vivid way. The following explanations describe the approach of using colour and opacity in this visualisation.

Field lines

The raw field lines either end at the frame of the window or when the predefined maximum of computation steps is reached. The first alternative leads to serious problems in a 3D-implementation as the z-axis is not bound by a frame. A solution could be built up by a bounding box. However, in both versions the fieldlines end abruptly. This could imply that the field also ends at this point. In contrast, by altering the opacity along the field line length the impression is conveyed, that the field strength decreases continuously. The following formula decreases the alpha values linearly:

$$\alpha_{1,i} = 1 - \frac{i}{i_{\max}}$$

Where i is an index indicating the position on the field line.

Furthermore the alpha values provide an opportunity to show that the field is not static. For this purpose small imaginary particles traversing the field lines are used. In [25] an exponential law for a disappearing particle is introduced:

$$\alpha_{2,i} = \alpha_0 q^{i_{\max} - i}$$

Where q controls how fast the particle mark fades away. For more than one particle on a field line the formula expands to

$$\alpha_{2,i} = \alpha_0 q^{(i_{\max} - i) \bmod d}$$

Where d is the distance between the imaginary particles.

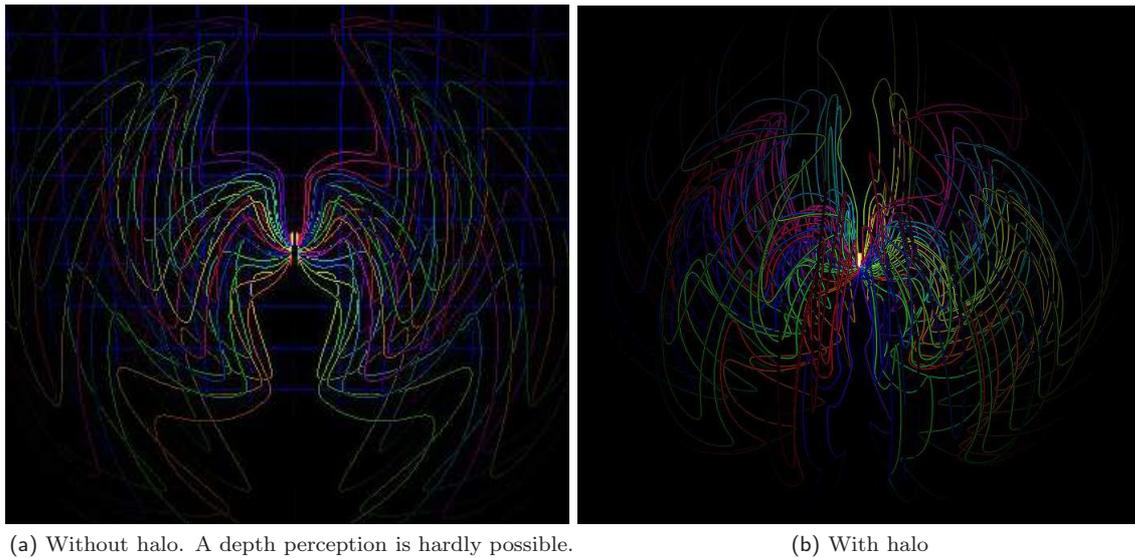
Multiplying these alpha values regards both effects and results in

$$\alpha_i = \alpha_{1,i} \cdot \alpha_{2,i}$$

Drawing field lines in 3D poses additional challenges. As they are not volumetric and therefore can not generate any shades, the depth perception is affected. Furthermore a distinction between the field lines, especially if there are many of them, is difficult. The latter may be solved by using different colours for the different field lines. The other points require a different solution.

One is presented in [4]. The authors propose to render not only a line but tubes. This way a volumetric figure which can receive shadows and have a shade is formed. Although generating the required vertices for a polygonal tube mesh on the CPU is possible in real time for a small amount of lines, a graphics processor able to run geometry shaders is required to implement a performant version of this suggestion². However, as this is not yet canon for personal computers, another solution has been implemented. It uses a halo effect around the field lines to generate a spatial impression (see fig. 4.2) similar to the approach used in [7]. Fig. 4.1 shows a comparison between enabled and disabled halo using an electron on a dipol trajectory.

² See chapter 5.4 for a very short sketch of shaders and their usage in the implementation



(a) Without halo. A depth perception is hardly possible.

(b) With halo

Fig. 4.1: A halo effect simplifies the depth perception.

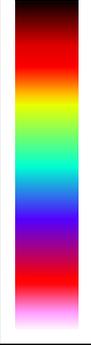


Fig. 4.2: A close up of the field lines with halo

Wave fronts

Only part of the electromagnetic spectrum is visible to the human eye. On a colour-hue-scale these frequencies can directly be mapped to the corresponding observed colour. For the invisible parts a different representation must be found. A passable solution for higher frequencies is decreasing the saturation. This causes the colour to bleach and in extreme cases to whiten. On the other hand the value is decreased for lower frequencies, causing the colour to darken and finally turn to black. Table 4.1 shows the interpolation points for the colour mapping which uses the HSV colour representation.

Since the wavelength changes in respect to the position of the observer, there are two different modes implemented. In the single observer mode the observer is situated at the position of the camera. Thus, the user controls the position of the observer at runtime by changing his own position. In the multiple observer mode there are several imaginary observers predefined, e. g. one at every vertex of the wavefront.

| | λ in nm | Hue (H) | Saturation (S) | Value (V) | Description of the electromagnetic wave |
|---|-----------------|---------|----------------|-----------|---|
|  | 2500 | 0 | 1 | 0 | infrared radiation |
| | 770 | 1 | 1 | 1 | red light |
| | 640 | 20 | 1 | 1 | orange light |
| | 600 | 45 | 1 | 1 | yellow light |
| | 570 | 65 | 1 | 1 | green light |
| | 490 | 170 | 1 | 1 | blue light |
| | 430 | 260 | 1 | 1 | violet light |
| | 390 | 360 | 1 | 1 | ultraviolet radiation |
| | 10 | 360 | 0.4 | 1 | X-radiation, ultraviolet radiation |
| | 5 | 360 | 0.2 | 1 | X-radiation |
| | 0 | 360 | 0 | 1 | - |

Tab. 4.1: Interpolation points for colour mapping on the wave fronts

Wavelengths longer than 2500 nm will appear black. They are not treated separately as they hardly arise in the simulation.

Now transfer functions are required to interpolate the intermediate values. The value for $\lambda[770; 2500]$ is interpolated linearly with

$$V = -\frac{1}{1730}\lambda + \frac{250}{173}$$

The saturation for $\lambda]0; 390[$ is interpolated by two linear functions:

$$S = \begin{cases} \frac{3}{1900}\lambda + \frac{73}{190} & \text{for } \lambda]10; 390[\\ \frac{1}{2}\lambda - 2.3 & \text{for } \lambda]0; 10[\end{cases}$$

A regression analysis gives an exponential function for H with $\lambda[390; 770]$:

$$H = 15067 \cdot 0.990506992^\lambda$$

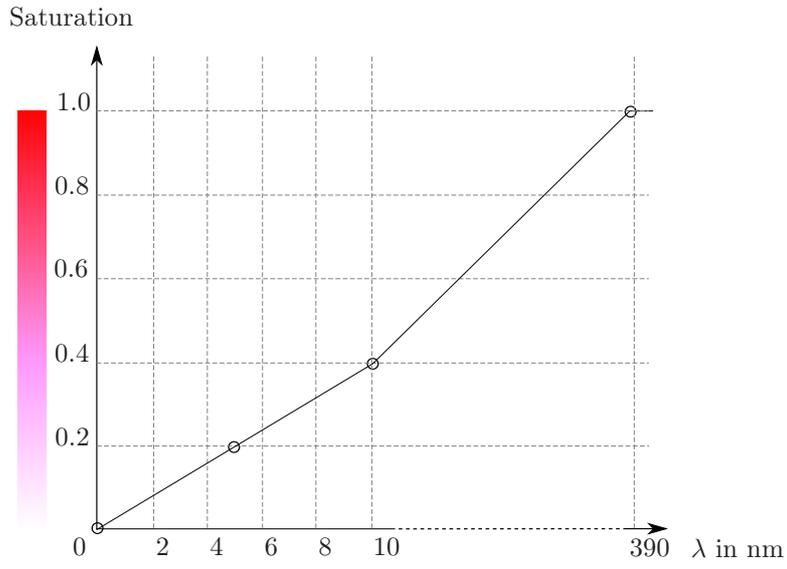
Fig. 4.3 on page 4.3 shows these functions graphically.

Then the HSV-values are converted into the corresponding RGB-values for drawing them. This is done using the method introduced in [11] resulting in RGB-values between 0 and 1.

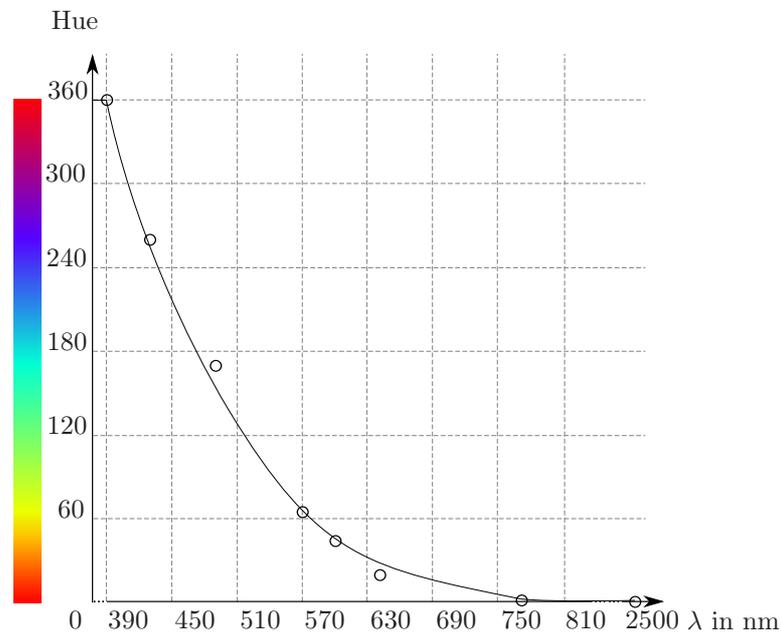
$$\begin{aligned} h &= \left\lfloor \frac{H}{60^\circ} \right\rfloor \\ f &= \left(\frac{H}{60^\circ} - h \right) \\ p &= V \cdot (1 - S) \\ q &= V \cdot (1 - S \cdot f) \\ t &= V \cdot (1 - S \cdot (1 - f)) \end{aligned}$$

$$(R, G, B) \begin{cases} (V, t, p) & \text{for } h = 0 \text{ and } h = 6 \\ (q, V, p) & \text{for } h = 1 \\ (p, V, t) & \text{for } h = 2 \\ (p, q, V) & \text{for } h = 3 \\ (t, p, V) & \text{for } h = 4 \\ (V, p, q) & \text{for } h = 5 \end{cases}$$

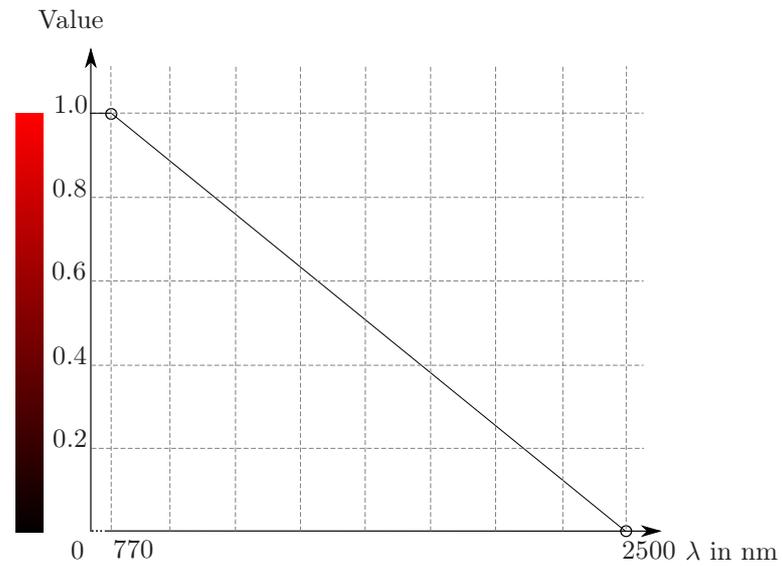
Additionally the opacity of the wavefront decreases the larger it grows using the alpha-value α_l which was introduced in the last subchapter.



(a) Saturation: Ultraviolet and X-radiation



(b) Hue: Visible light



(c) Value: Infrared radiation

Fig. 4.3: Colour mapping. The marks correspond to the values in the table.

4.2 The environment

Not every trajectory causes the electron to stay in a defined bounding box. Especially an electron on an undulator trajectory will run out of the screen if the camera, e.g. the observer is not following it. This is why the camera is attached to the electron. However, to create the imagination of a forward velocity, a moving grid is drawn which simulates a background moving contrary to the electron.

Additionally the trajectory itself is drawn using particles which are emitted at the electron's position. Fig. 4.4 shows an electron on an undulator trajectory with enabled grid and trajectory view.

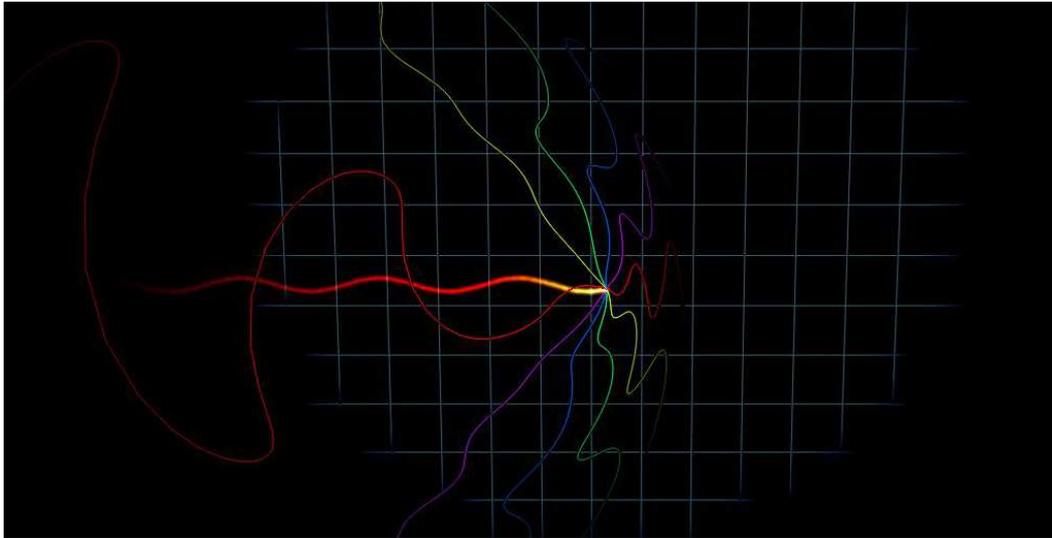


Fig. 4.4: The grid moves along the negative x-direction. The trajectory is drawn using yellow particles which fade to red and finally disappear.

4.3 Field particles

Some effects, especially those resulting from the circle trajectory, can hardly be recognised by field lines and wave fronts. Therefore an approach is implemented which treats the radiation emitted by a charge as particles moving along the wave vector \vec{k} .

Thus the computation of the wave vector and the discrete positions on the field lines respectively wave fronts is done the way described in chapter 2. But now the resulting positions are not connected via line strips. Instead billboards are placed at the position of the vertices. Billboards are small pictures representing particles which are oriented to always appear perpendicular to the viewing direction.

This simulates the distribution of density in the field showing some effects like the spiralforn field for the circle trajectory.

Fig. 4.6 shows synchrotron radiation represented by particles from different points of view.

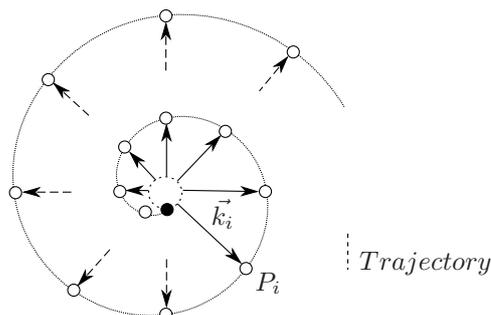
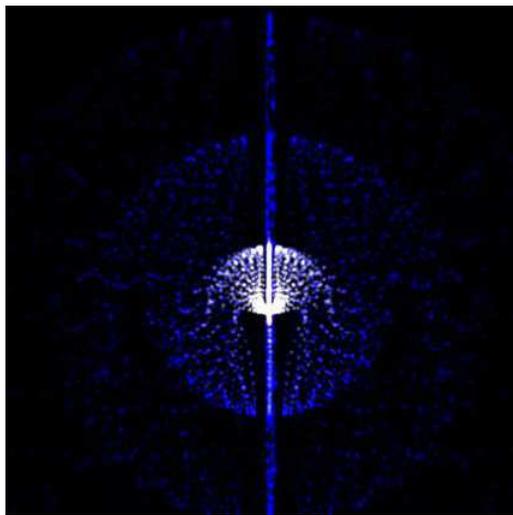
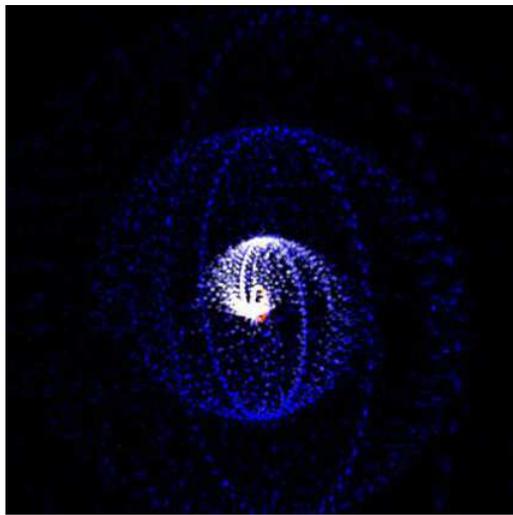
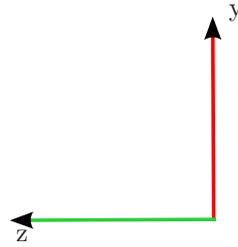


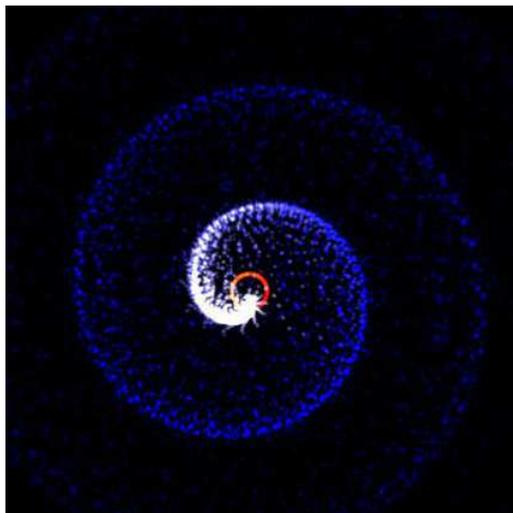
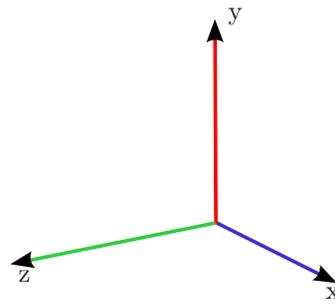
Fig. 4.5: The particles P_i , forming the field around an electron on a circle trajectory, are arranged spirally.



(a)



(b)



(c)

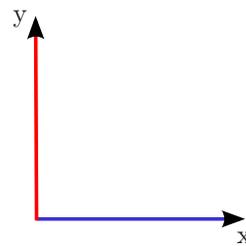


Fig. 4.6: Radiation caused by a circle trajectory from different angles. The trajectory is drawn in orange.

4.4 The wave plane

Working in 3D space offers a new way for visualising electro magnetic waves. The computed positions have been connected to build wave fronts and field lines. But this is far from how waves are commonly imagined, e.g. the surface of waves in liquids looks different from what is visualised so far.

Although this may be due to the fact that there is no geometrical shape for electro-magnetic waves visible to the human eye, it is possible to reconstruct a shape which corresponds to this image of a wave. For this purpose all field lines in a chosen plane are selected and connected to each other, such that the positions $P_{i,j}$ and $P_{i,j+1}$ on two adjacent field lines and their successors $P_{i+1,j}$ and $P_{i+1,j+1}$ build a quad. Repeating this for every field line and position on this plane results in the wave plane. When the electron moves, every quad remains connected to the corresponding part of the field line. This way the desired effect occurs.

Fig. 4.7 shows a wave plane in its initial state and while the electron moves. Both states are shown in a smooth and in an approximated version. In the implementation the wave plane is approximated in a higher resolution (fig. 4.8).

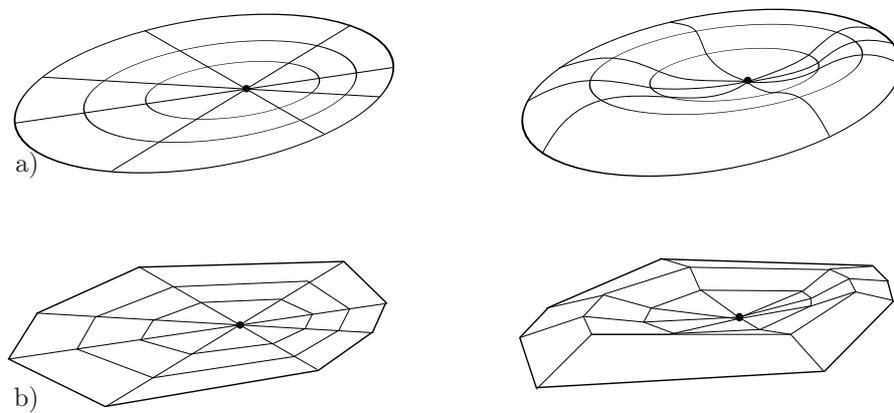
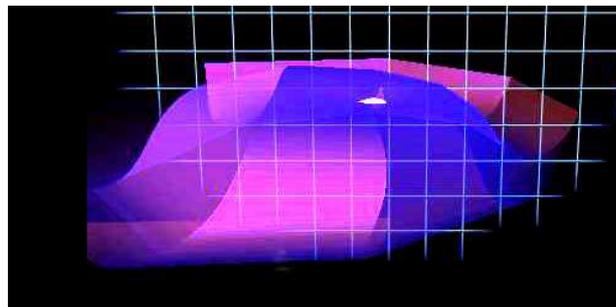
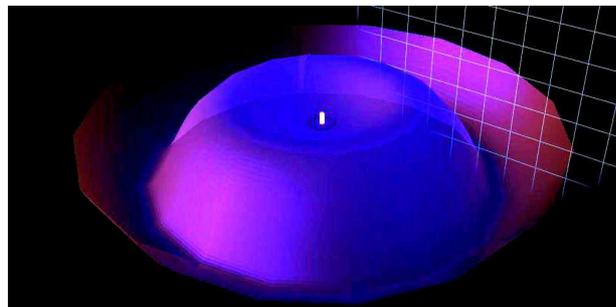


Fig. 4.7: a) The desired result b) A very coarse grained approximation



(a) Quad normals



(b) Interpolated normals

Fig. 4.8: A comparison between a normally illuminated wave plane and an illumination using interpolated normals. In (a) the segments between the field lines can be identified easily. The illumination is explained in chapter 4.5.

4.5 Illumination

The Blinn-Phong model

The Blinn-Phong model is used for illumination. This is the default shading model in OpenGL's fixed function pipeline³. But since it processes shading only per vertex, a fragment program for a per pixel lighting still has to be written by the developer. For this reason the blinn-phong model will be shortly reviewed at this place.

The resulting light intensity is the sum of ambient, diffuse and specular light intensities like shown in fig. 4.10. Ambient light is reflected equally all over the object and thus colours it in a single shade.

The diffuse reflection is controlled by the normal of the surface of the object \vec{n} and the incoming light direction \vec{d} (see fig. 4.9). The smaller the angle between those two vectors, the higher the reflection becomes. This is realised with the dot product, which reaches a maximum at parallelism and zero at perpendicularity. Even greater angles describe surfaces, which are not hit by the light and therefore are cropped to zero.

The specular reflection sets highlights whose position depends on the normal of the surface \vec{n} , the light direction \vec{d} and the viewing direction \vec{e} . Their size is set by the exponent σ . Using the normal and the light direction, a halfway vector is determined. The dot product between this vector and the normal, cropped to values greater than zero, to the σ define the part of reflected specular light.

Hence, the final reflected light intensity is computed by

$$\begin{aligned} I &= I_{amb} + I_{diff} + I_{spec} \\ &= r_{amb} \otimes L_{amb} + r_{diff} \otimes L_{diff} (\hat{n} \cdot \hat{d})_+ + r_{spec} \otimes L_{spec} (\hat{n} \cdot \hat{h})_+^\sigma \end{aligned}$$

with

$$\hat{h} = \frac{\hat{e} + \hat{d}}{|\hat{e} + \hat{d}|}$$

Where r_i are the object's attributes and L_i the light's attributes. They all consist of a red, green and blue part, such that $r = (red, green, blue)^\top$.

An example for a Blinn-Phong shader written in GLSL can be found in the appendix. Fig. 4.11 shows the illuminated wave fronts.

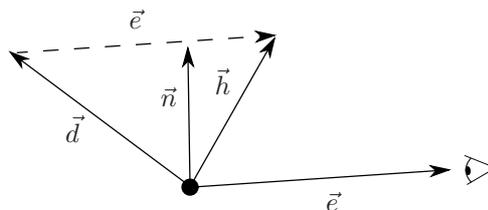


Fig. 4.9: The vectors used for calculating the reflection

³ Although the fixed function pipeline concept has been totally reworked since OpenGL 3.0, shaders are still an irreplaceable tool and have even been extended.

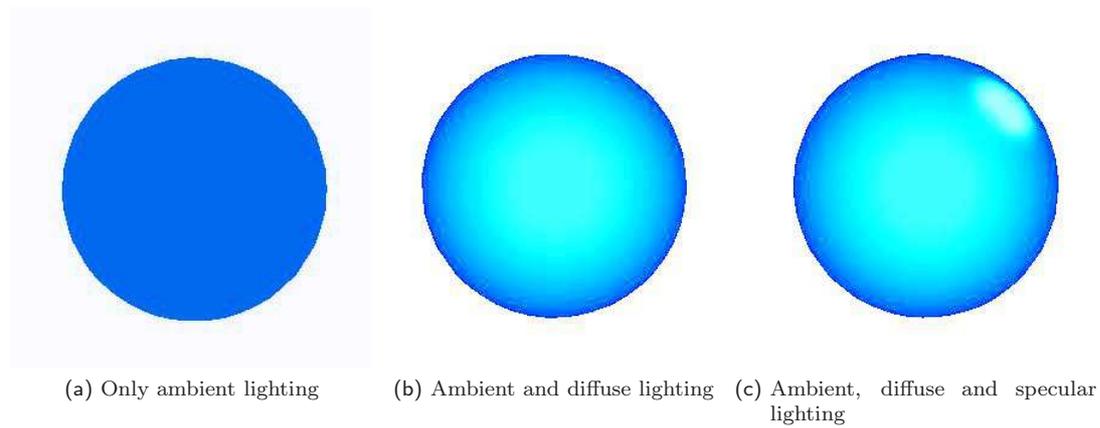
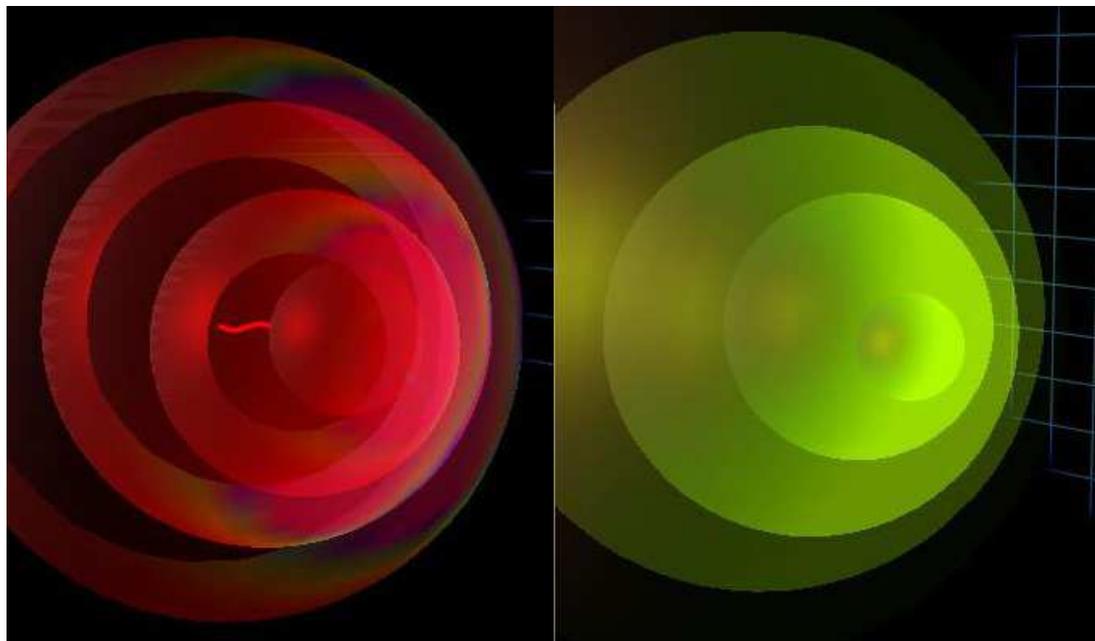


Fig. 4.10: A sphere illuminated step by step



(a) Wave fronts in the multiple observer mode, additionally the clipping distance is set shortly in front of the electron. The trajectory can be seen where the spheres are cut. (b) Wave fronts in single observer mode

Fig. 4.11: Illuminated wave fronts

The normals

In most cases per vertex shading looks acceptable. However, the normals of the object are essential as specular and diffuse reflection depend on them. Computing them is trivial for a sphere. In that case the normal of a vertex is the vector between this vertex and the middle of the sphere.

On the other hand, a segment of a field line has an infinite amount of possible normals, namely every vector which is part of the plane perpendicular to the line, the normal space E_n . This issue has already been treated in [25]. The authors span another plane $E_{d,l}$ through the light direction and the line segment. The intersection line between this plane and the normal space is the desired normal.

Therefore the direction of the normal is given by

$$\vec{n} = \vec{l} \times (\vec{d} \times \vec{l})$$

Where \vec{l} is the line segment described as a vector between P_i and P_{i+1} . This is shown in fig. 4.12. Fig. 4.13 shows the illuminated field lines.

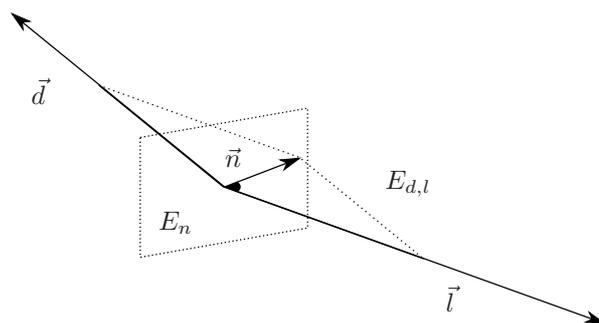


Fig. 4.12: The normal of a line is derived by building the intersection between the normal space E_n and the plane $E_{d,l}$ spanned by \vec{l} and \vec{d} .

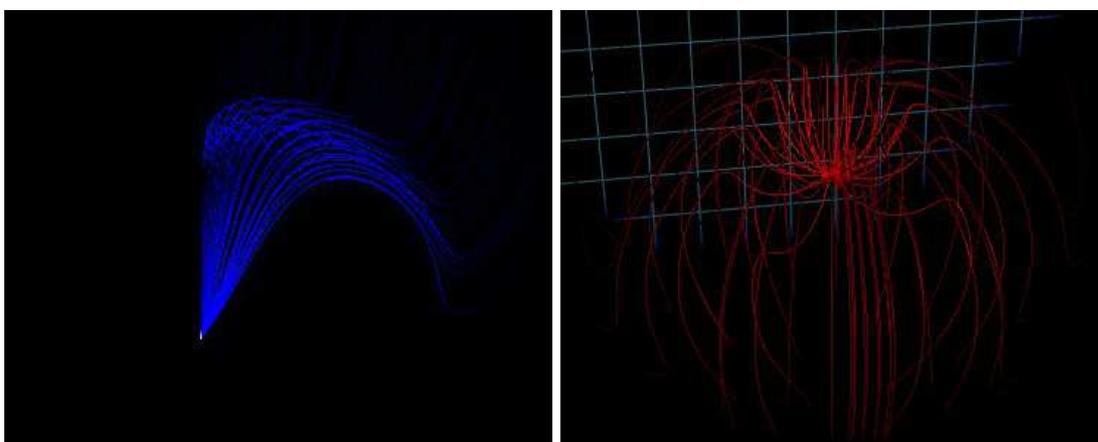


Fig. 4.13: Illuminated field lines

5 Implementation

5.1 The used framework: Ogre

The first version of *Radiation3D* was the result of a practical course at the TU Dresden and the HZDR⁴. In this course a rendering engine called *Ogre* ('Object-Oriented Graphics Rendering Engine') was used. This has several reasons. Firstly, it wraps the *OpenGL* functionality and therefore eases its usage by supplying a more intuitive interface than the underlying libraries do. And secondly, it supplies some useful pre-implemented features like particle systems and scene managers. Apart from that, the framework already proved itself in previous practical courses like the visualisation of Elbe and Draco⁵. Thus, using the same framework eases the combination of those projects significantly.

The above mentioned course did not only result in a first version of *Radiation3D* but in an integration of the results of the the previous course in a common environment using the same interfaces. In the following this environment is called *LMC* ("light matter collision") since this is the working title of the program. The subprojects, of which *Radiation3D* is one, are called states. The states concept is adopted from one of the various *Ogre* tutorials⁶.

5.2 Program structure

Fig. 5.1 provides an overview of the most important classes and their connection to *Ogre*. The green packages represent the content of the framework, the orange classes are classes written for and used by all *LMC* subprojects and the yellow classes represent the code specifically written for *Radiation3D*.⁷ This shows that the direct extension of the framework via inheritance is done by an interface which is used by *Radiation3D* as well as by the other states, too, such that only the classes in yellow need to be exchanged. Fig. 5.2 shows a detail of this diagram which is slightly modified. The classes in blue belong to the COLLISION state where ELECTRONVIS is not directly associated with the APPSTATE child but with COLLISIONBUNCH which also holds pointers to other objects necessary for this state.

Amongst others, these diagrams are missing the classes APPSTATEMANAGER and LMCAPP. They are responsible for managing the different states which are organized on a stack. As they are only used for this purpose and not for the visualisation or the model computation itself, they are not discussed here in detail.

The overall structure separates the model (implemented by the ELECTRON class) from the view and the controller (implemented by the ELECTRONVIS and BASEAPPLICATION/OGER class). Thus, the ELECTRON class does not use the engine. It is only responsible for calculating the model and stores the results in data structures which are especially defined for this case and only need the common *C++* libraries like MATH.H and VECTOR.

For the visualisation a type called MANUALOBJECT is used. This is a construct of the engine which is optimised for being updated very often. As there are many, partially also complex, objects of this kind, their construction is separated from the representation. When OGER initialises ELECTRONVIS, neither ELECTRONVIS nor OGER care for the initialisation of the MANUALOBJECTS. An instance of FIELDGENERATOR initialises them instead by defining their render operation and material as well as their number of vertices. However, the representation is still done by ELECTRONVIS. This basically implements the idea of the popular *builder pattern*[12].

Additionally OGER takes the role of a listener to mouse and keyboard events. As a big part of the necessary functionality is pre-implemented by *Ogre* and *OIS* (Object Oriented Input System), OGER extends, via inheritance from BASEAPPLICATION, which itself inherits from APPSTATE, the KEYLISTNER, TRAYLISTENER and MOUSELISTENER classes.

⁴ Helmholtz-Zentrum Dresden-Rossendorf

⁵ Elbe: Electron Linac for beams with high Brilliance and low Emittance. Draco: Dresden laser acceleration source. Both radiation sources are installed and run at the HZDR

⁶ The mentioned tutorial can be found at <http://www.ogre3d.org/tikiwiki/Game+State+Manager>.

⁷ Originally "Oger" was the name of a tutorial class which was used when this project was still a practical course. But instead of abolishing it and building a new class from scratch, it was extended and ended up in being one of the core classes. It has just never been renamed and as a tribute to the engine and because it is more pleasing being asked if the ogre is fine than how work is going, it has kept the name.

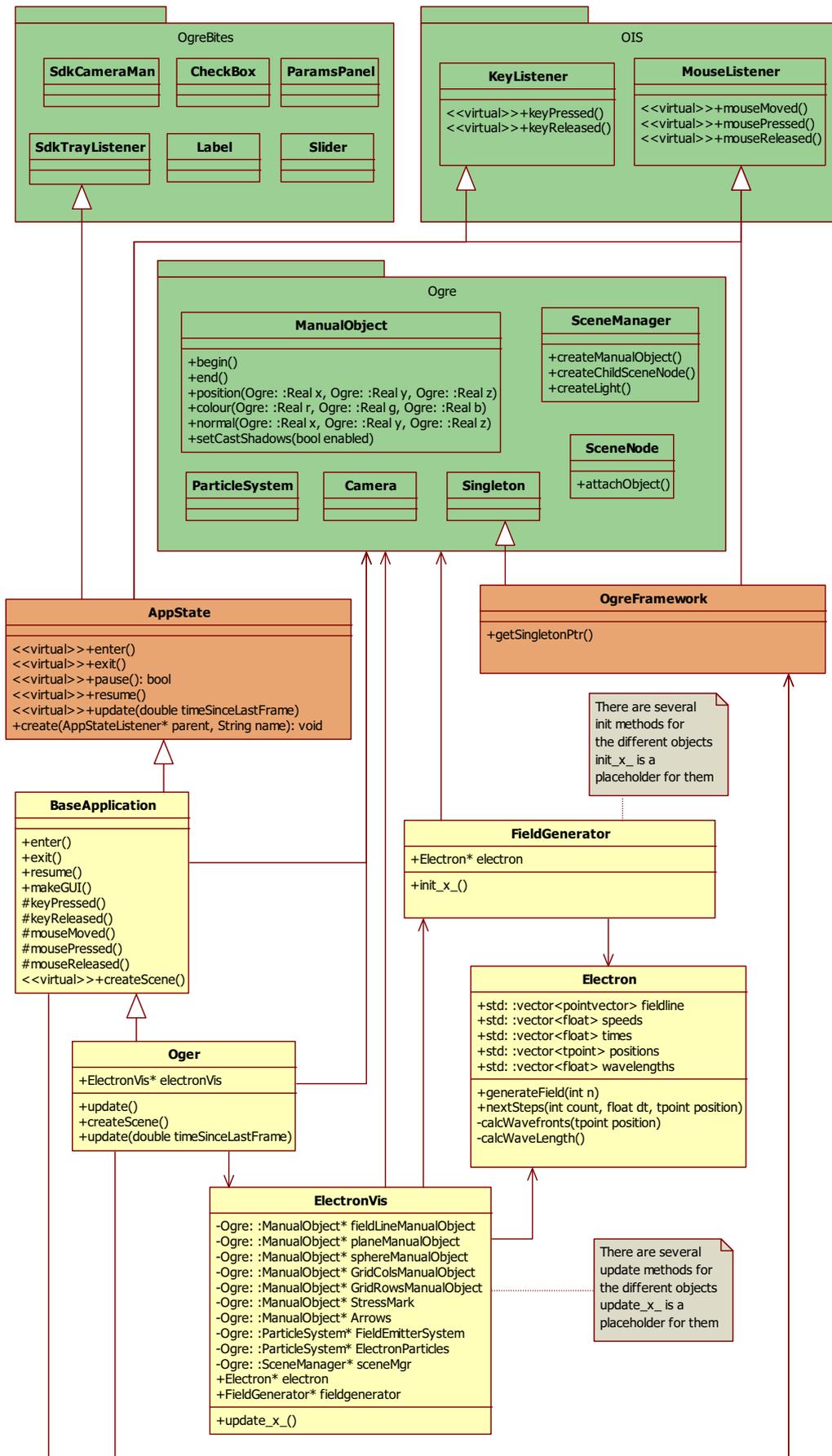


Fig. 5.1: A class diagram showing the most important classes, members and functions. This is not a complete listing but outlines the basic points. The green packages belong to the Ogre framework. The orange classes are written for and used by all states. The yellow classes are specifically written for Radiation3D and the Collision state.

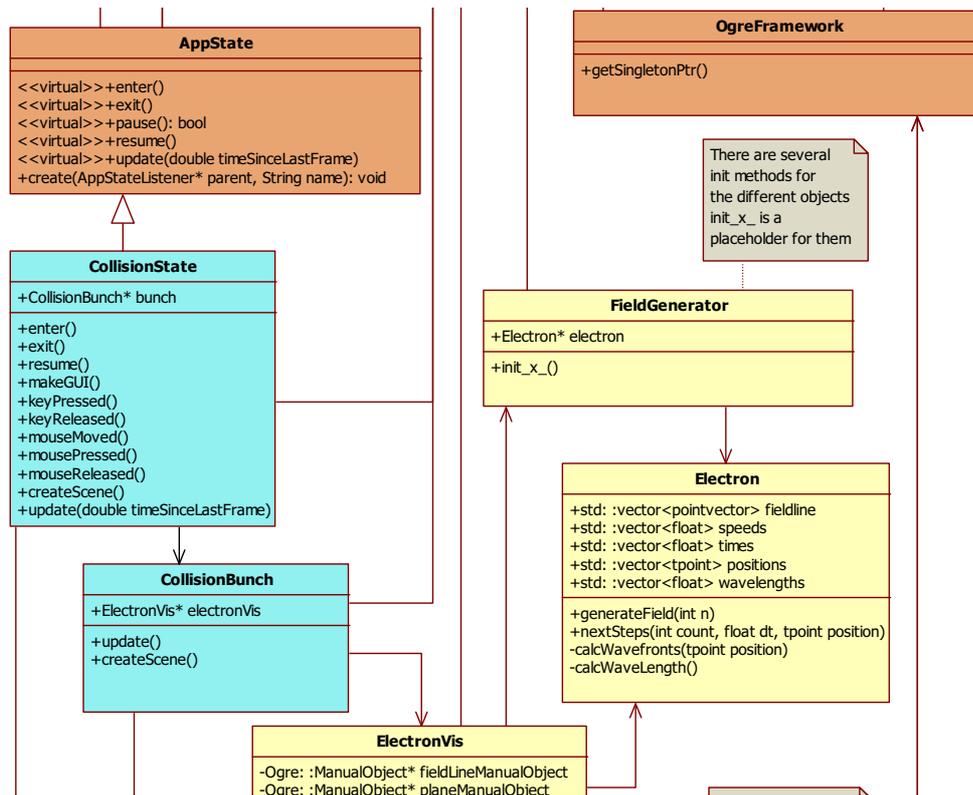


Fig. 5.2: A section of fig. 5.1 with the classes changed for the Collision state (blue).

5.3 Control flow

Fig. 5.3 and fig. 5.4 show a simplified sequence diagram. They start at the moment the corresponding state has been invoked and show the sequence of the creation of the objects. After all objects have been initialised and added to the scene graph, the update methods of the state are repeated in a loop.

These update methods call the recalculation of the model and the corresponding geometry and vertex colours. Then these values are assigned to the corresponding MANUALOBJECTS. Every section of these MANUALOBJECTS keeps a pointer to a MATERIAL definition. The way the object is rendered is defined in this MATERIAL, e. g. if the lighting is on, if and which shaders are used and how the objects are blended. MATERIALS can be defined dynamically or in an extern file like in lst. 2 and lst. 1.

What cannot be seen in the diagrams is that an additional query for the time since the last frame is necessary to ensure that the actual field calculation is done only one time per frame. A higher frequency would not falsify the results but slow the whole program down noticeably.

Listing 1: A material with two passes

```

1 material Wavefronts {
2   technique {
3     pass {
4       lighting off
5       colour_write on
6       scene_blend alpha_blend
7     }
8     pass {
9       lighting on
10      colour_write on
11      shading phong
12      separate_scene_blend src_colour one_minus_src_colour src_alpha zero
13    }
14  }
15 }

```

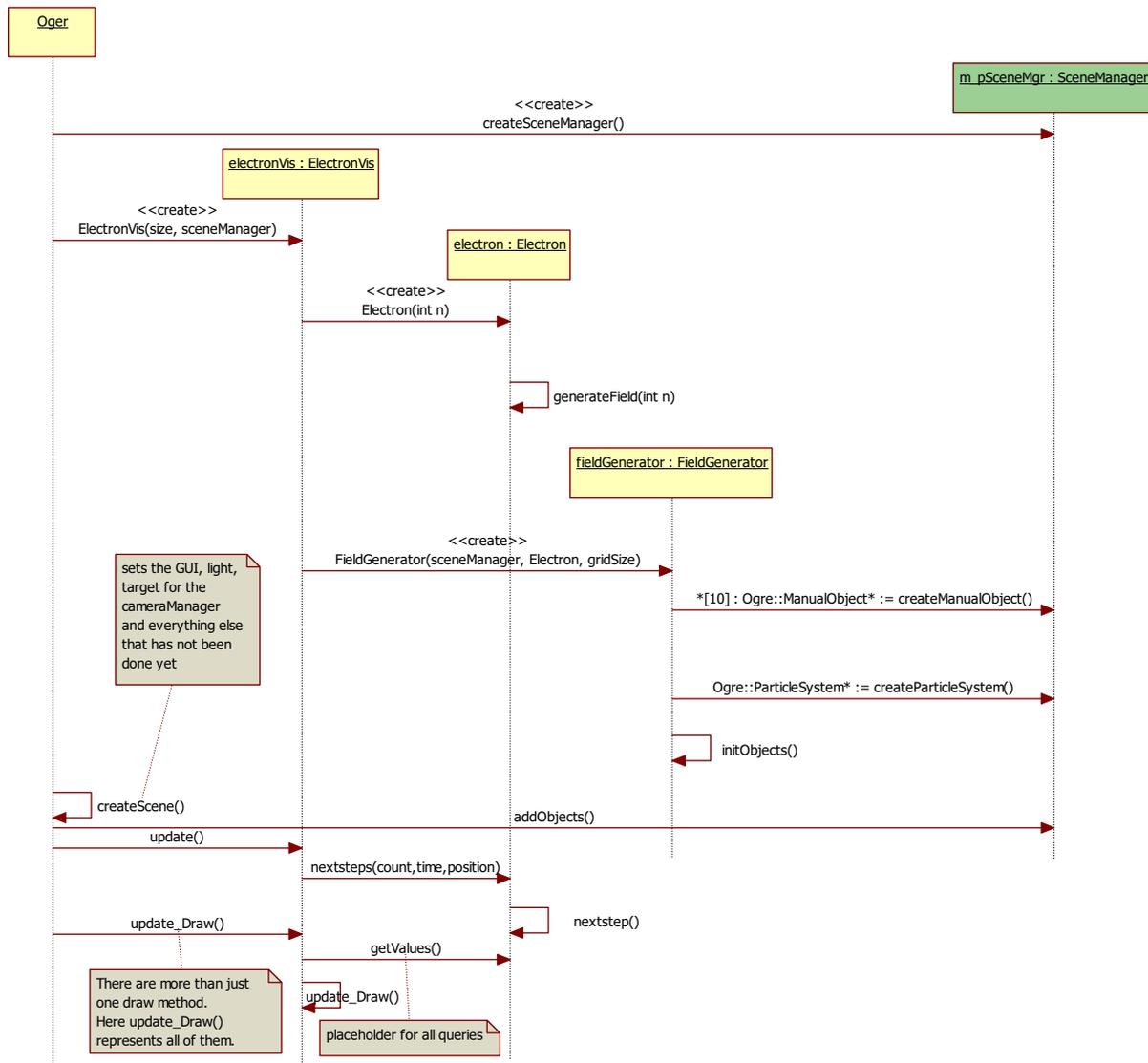


Fig. 5.3: A sequence diagram showing the most important interactions when Radiation3D is started.

Listing 2: An example for a material script using a vertex and a fragment shader

```

1  vertex_program phongvp_gls1 glsl
2  {
3      source phongvp.gls1
4  }
5  fragment_program phongfp_gls1 glsl
6  {
7      source phongfp.gls1
8  }

10 material shadertest{
11 technique {
12     pass {
13         vertex_program_ref phongvp_gls1
14         {
15             param_named_auto EyePosition camera_position_object_space
16             param_named_auto LightPosition light_position_object_space 0
17         }
18         fragment_program_ref phongfp_gls1
19         {
20             param_named_auto amb ambient_light_colour 0

```

```

21         param_named_auto spec light_specular_colour 0
22         param_named_auto diff light_diffuse_colour 0
23         param_named blinnphong int 0
24         param_named_auto shin surface_shininess
25     }
26 }
27 }
28 }

```

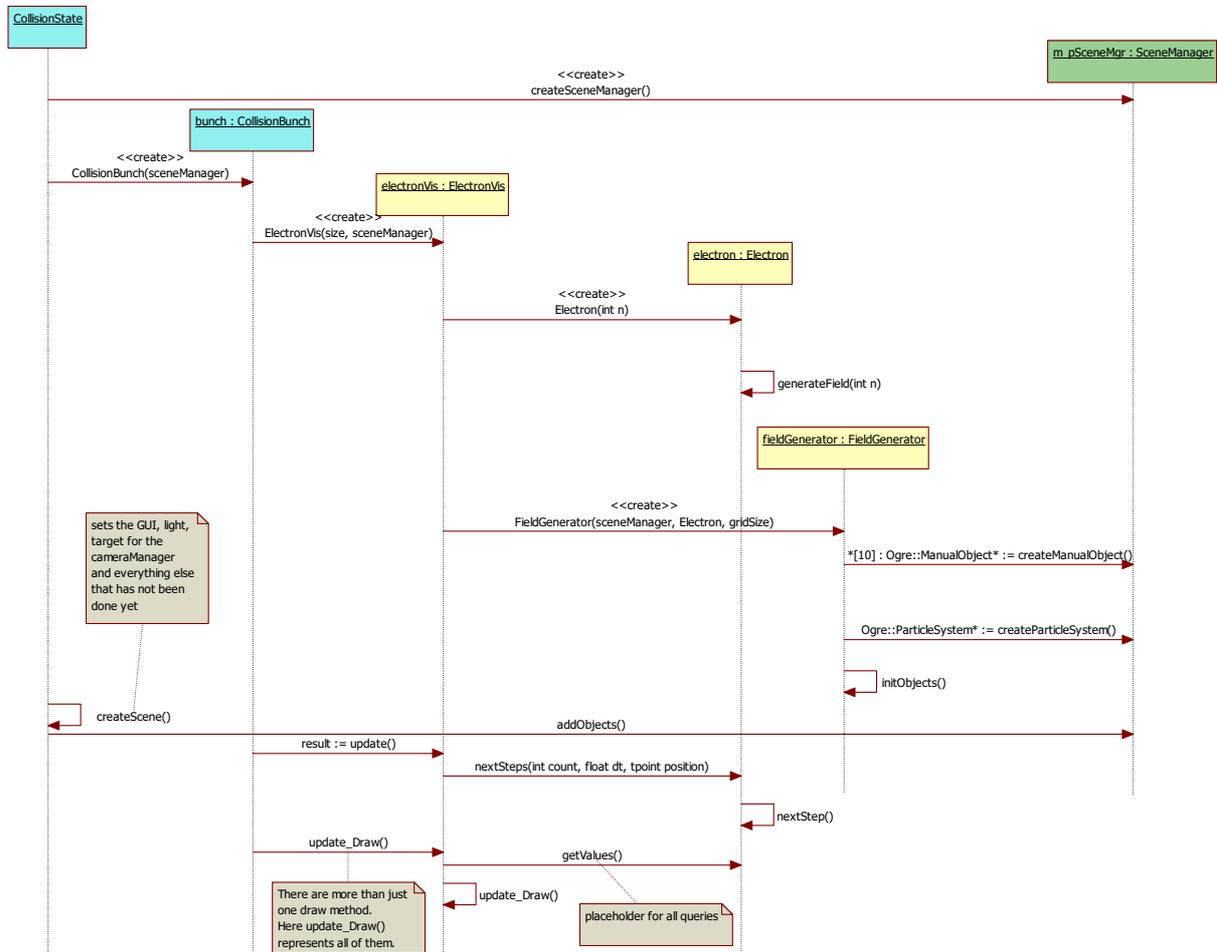


Fig. 5.4: A modified version of fig. 5.3 showing the startup routines for the collision state.

5.4 Specifics in the implementation

Using a framework usually inflicts some constraints since it does not always implement all the desired functionality. But using the lower level libraries the framework is wrapping and mixing this code up with code using the framework is no good programming style because it increases the efforts which need to be done to keep consistency enormously. Although *Ogre* is an open source project that makes it possible to know what the framework code does, this implementation hardly uses this way of adding missing functionality. In most cases work arounds which use the provided functionality are found instead.

The most problematic example for missing functionality in *Ogre* is the lack of possibilities to render quads. While *OpenGL* itself knows quads and quad strips until *OpenGL* 3.1, *Ogre* implements neither. Therefore quads are composed of two triangles. For being able to reuse this solution, it is encapsulated in a function taking the four vertices of the quad and a pointer to the `MANUALOBJECT` it should be assigned to as arguments.

Another difficulty leads from the nature of *OpenGL* to render per vertex and not natively per fragment which leads to angular shapes especially for the wavefronts. To solve this issue there are two approaches. One is choosing a resolution which is high enough to keep the differences to a minimum. For a sphere a resolution of 11.25° for φ and ϑ serves this purpose. This corresponds to a resolution of 32 steps in horizontal and vertical direction. The other approach is using a fragment shader.

Shaders replace parts of *OpenGL*'s fixed function pipeline. The sequence of their execution is shown in fig. 5.5. The shader defining the final pixel colour is the fragment shader. It gets the interpolated output of the previous shaders and optionally some user defined data as input. This is where the Blinn-Phong model described in chapter 4.5 is implemented using *GLSL* (OpenGL shading language). For integrating shaders into the program, they have to be declared in the material script which contains the definitions for the used MATERIALS. After this has been done they can be referenced in any MATERIAL and the necessary variables can be passed like shown in lst. 2.

Furthermore, the framework does not originally render illuminated objects and keep their transparency. This may be solved by either using an own shader again or by rendering the object twice. For rendering a scene more than once, two or more passes have to be added to the used material script. The first pass renders the object without any lighting enabled. The second pass renders it with lighting. How the combined output of these two passes looks in the end is now controlled by the defined blending operation. For the implementation a separate blending for colour and α values is used. For the α values, the second pass is ignored. The colour values are combined based on their brightness. Lst. 1 shows an example of a material using two passes to render with colour and transparency. The final blending is defined in the last line of the second pass. The general format is

```
separate_scene_blend <colour_src_factor> <colour_dest_factor> <alpha_src_factor>
                        <alpha_dest_factor>
```

and the final value for colour and α value is then derived by

$$value = sourceFactor \cdot source + destFactor \cdot dest$$

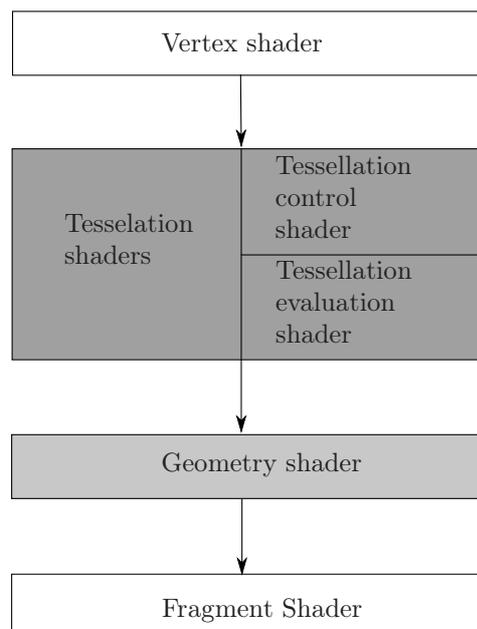


Fig. 5.5: While the “classical” shaders, specifically vertex and fragment shader, are available since OpenGL 2.0, geometry shaders have been introduced in OpenGL 3.2 and tessellation shaders even in OpenGL 4.0.

5.5 Conflicts between theory and software layout

Although the model is separated from the viewer wherever possible, there are some situations in which parts of the model depend on members of the viewer, e.g. the recognised wave length depends on the position of the observer (the camera). To keep the model independent from the viewer this issue is managed by the `ELECTRONVIS` class. If required it computes this missing model data, resulting in the situation that a part of the model is weaved into the viewer class. A clean software design would introduce a mediator between `ELECTRONVIS` and `ELECTRON`. This prevents the viewer from having to perform calculations the model is responsible for. And the model still does not need to have any knowledge about the viewer. However, as the dependencies are still manageable, this has not been applied, yet.

While it is possible to generate an amount of vertices which is only limited by the resources of the computer, it is not trivially possible to change the amount of actually drawn vertices for every frame. A `MANUALOBJECT` is initialised with a certain amount of vertices which cannot be changed without clearing and reinitialising it. This cannot be done every frame because it produces many memory writing accesses which are too slow for a real-time application. Thus, the number of vertices used for representing a field line is not changed dynamically. Even though this produces some corners in the regions where they are far away from each other, it is the fastest solution. In a subsequent version this could be optimised using geometry shaders.

An aspect of the visualisation which needs some closer examination is the spatial scaling. An electron and its minimal trajectories cannot be seen by the human eye. However, it is the purpose of a visualisation to make it visible. Consequently, the dimensions are extremely exaggerated. The computation is done using the usual SI units. However, for the representation in *Radiation3D* 1.0 nm corresponds to two picture points. In `COLLISION` the conditions are different. There are surrounding objects which should appear much bigger than the electron. This is solved using the three different zoom steps of which only the closest one shows the single electron. But having reached this closeness, the electron appears larger than the screen. This is why it is scaled down using a simple vertex shader, such that one point represents 0.025 nm.

6 Results and discussion

6.1 Results

Many aspects of electrodynamics have been simplified for this work. The trajectories are not computed by the forces affecting the electron. Wavefronts, which are necessary to visualise the vectorial dimensions of the field, are only emitted when the direction of the velocity changes. However, many effects can now be shown in a new, vivid way. Some representations, e.g. the wave plane, are observable for the first time since they are not possible in a 2D space. Also the computation of the wavelength depending on the position of an observer who can freely move in space is new.

The computations are reduced to what is necessary for drawing the field which makes the visualisation able to run in real-time, even if it runs on an ordinary office computer.

In the following, the fields for the implemented trajectories are viewed and compared to other approaches for calculating the fields.

Resting electron

The field is emitted radially like in the other visualisations mentioned in chapter 1. The lack of equipotential lines shows a contrast to the classical representation like the one in fig. 1.1a.

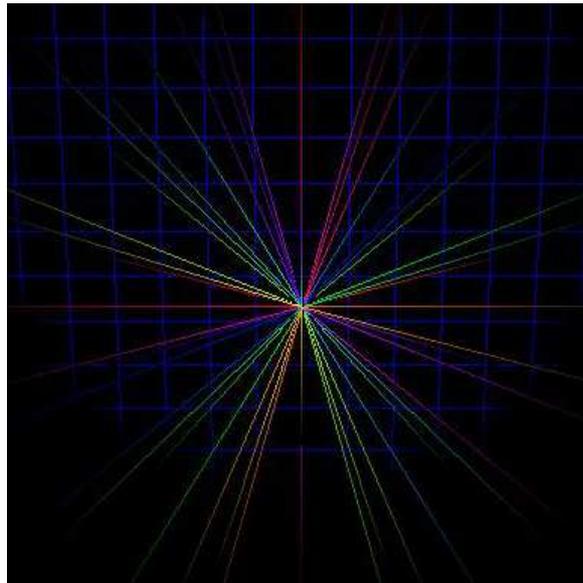


Fig. 6.1: The field lines of a resting electron. Each field line is drawn in a different colour for an easier distinction between them.

Linear trajectory

The field lines of a linearly moving charge accumulate in the direction perpendicular to the moving direction (see fig. 6.2). This effect has already been explained in [17] and can clearly be seen in the simulation. Due to the shintake algorithm which considers the finite speed of light, the field lines that point towards the moving direction are shorter than the ones on the opposite side. The distances between the computed positions is smaller there. There are no wave fronts and no equipotential lines or spheres. Hence, although there is a magnetic field, it is not shown. Solving this issue is a task for future works.

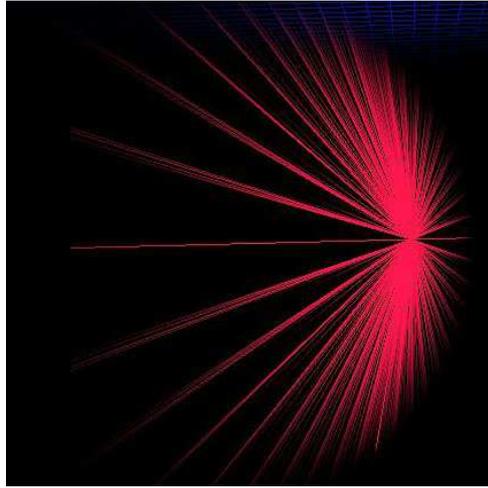


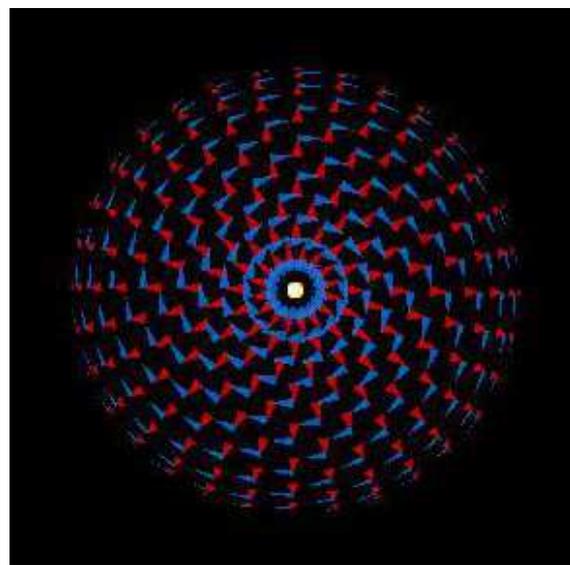
Fig. 6.2: An electron on a linear trajectory. The field lines are not cut because the field strength was too low there but because the fixed number of vertices is already reached.

Dipol trajectory

An oscillating electron emits a typical Hertzian dipole field. For small frequencies the wave fronts become almost concentric. The electric and magnetic fields change their direction depending on the moving direction of the electron, more precisely if the electron moves into positive or negative y -direction.



(a) The sector drawn is reduced to the x - y -plane.



(b) The E - and B -field in the x - z -plane (looking into negative y -direction). The red arrows show the direction of the electric field while the blue ones show the direction of the magnetic field.

Fig. 6.3: The field around an electron which is moving on a dipol trajectory

The density of the field lines provides a way for drawing conclusions about the electric field strength. Hence, for a closer examination the distances of two random field lines have been measured at every iterative step. [17] introduces a formula for calculating the electrical field

$$E = k(n \times p) \times n \frac{e^{ikr}}{r} + (3n(n \cdot p) - p) \left(\frac{1}{r^3} - \frac{ik}{r^2} \right) e^{ikr}$$

Thus, for small r , e.g. in the near zone, the formula can be simplified to

$$E = (3n(n \cdot p) - p) \frac{1}{r^3}$$

and for larger r , e.g. in the far zone, it can be simplified to

$$E = (k^2(n \times p) \frac{e^{ikr}}{r}) \times n$$

Therefore, the electrical field strength decreases with $1/r^3$ in the near zone and with $1/r$ in the far zone. Thus, the distances between the field lines increase cubically in the near zone and linearly in the far zone.

Fig. 6.4a shows the reciprocal of the distances between the first 20 positions on the field lines, which are all situated in the near and intermediate zone. The curves have been found doing a regression analysis and show clearly that the cubic relation does exist in the simulation for the near zone. They differ in the weight the first values have. Their formulas are

$$\begin{aligned} f_{red}(i) &= \frac{800}{(i + 3.4)^3} + 1 \\ f_{blue}(i) &= \frac{600}{(i + 2.8)^3} + 0.9 \end{aligned}$$

There are only few values for the near zone. Hence, the curve fits the first few sample points best before it falls off less rapidly than the values.

For the far zone, only these distances are measured, where a wave front is recognised. Fig. 6.4b shows them.⁸ The scale has been changed because the values are approaching zero and therefore decrease less rapidly. Obviously the cubical curves do not match anymore. Instead of that the values alternate around a rational function of the form

$$f(i) = \frac{a}{i + b} + c$$

If all values are given the same weight, the function

$$f(i) = \frac{13.05}{i + 0.84}$$

can be approximated. Fig. 6.4b shows the graph and the sample points.

It is also possible to find a function which is suitable for all measured values. It should have the form

$$f(i) = \frac{a}{(i + b)^3} + \frac{d}{i + c} + e$$

A regression returns

$$f(i) = \frac{122}{(x + 1.52)^3} + \frac{14.39}{x + 2.56}$$

Fig. 6.5 shows this result graphically in different levels of detail. The graphics show the high correspondence of the simulated data to the expected results. Thus, the Shintake-algorithm works very precisely.

⁸ For the regressions a mathematica script is used, which can be found in the appendix.

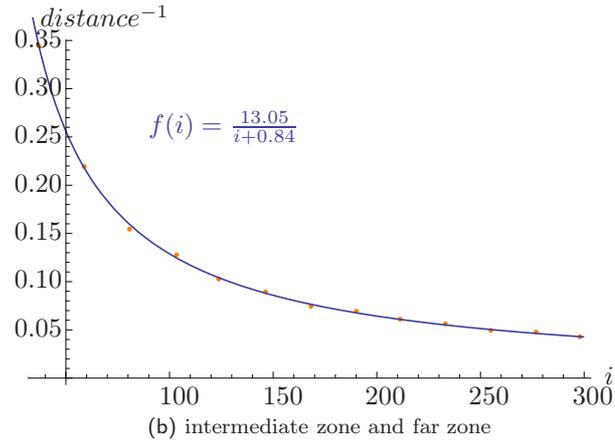
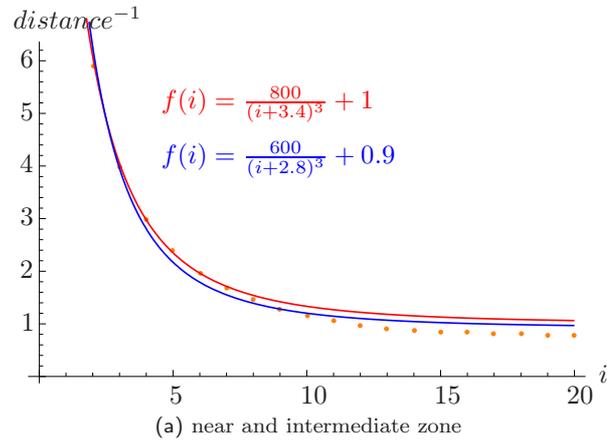


Fig. 6.4: Distances between two fieldlines, different approximation for near and far zone

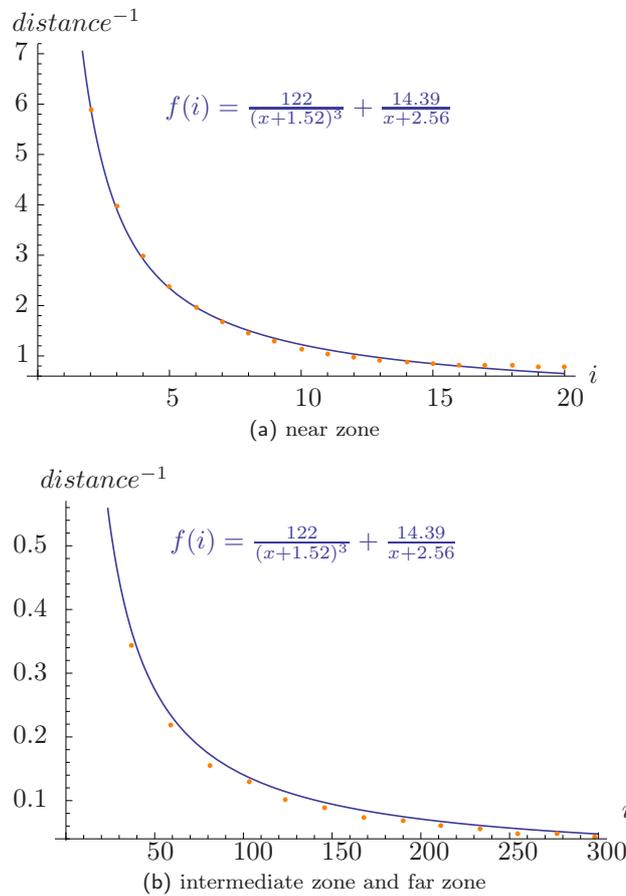


Fig. 6.5: Distances between two fieldlines, one approximation for all sample points

Circle trajectory

An observer situated in the plane of the circle sees an oscillating field but if the viewing direction is perpendicular to this plane, a spiral is visible. This effect can be seen best by viewing the “particles”. It depicts the synchrotron radiation. In practice a charge which is affected by a bending magnet does a quarter circle movement. The electro-magnetic fields for this scenario have already been analysed before, e.g. in [6] p.10. It shows a simulation of the field very similar to the one produced with Radiation3D.

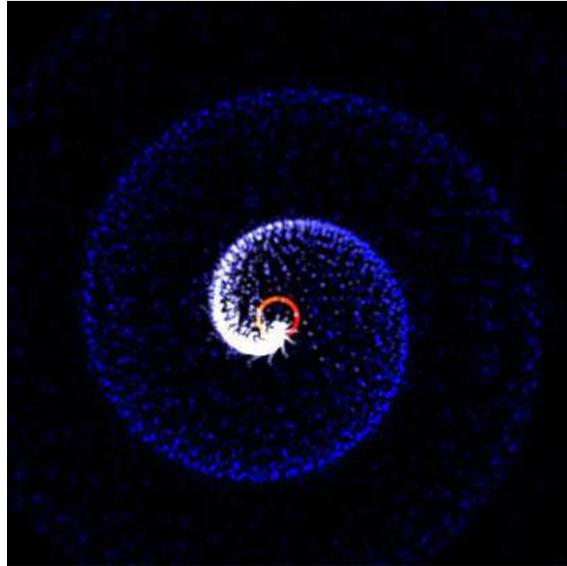


Fig. 6.6: Synchrotron radiation caused by an electron on a circular trajectory

There is a script written by Richard Pausch, who is working on this topic at the HZDR, which precalculates the radiation occurring in experiments with synchrotron radiation.⁹ Fig. 6.7 shows the result of one of those calculations for a velocity of $v = 0.903c$ and a trajectory radius of 400nm. The observer is situated in the trajectory plane. The graph shows the highest expected circular frequencies with a peak at $1.17981 \cdot 10^{16}$ Hz which corresponds to a wavelength of 159.657nm.

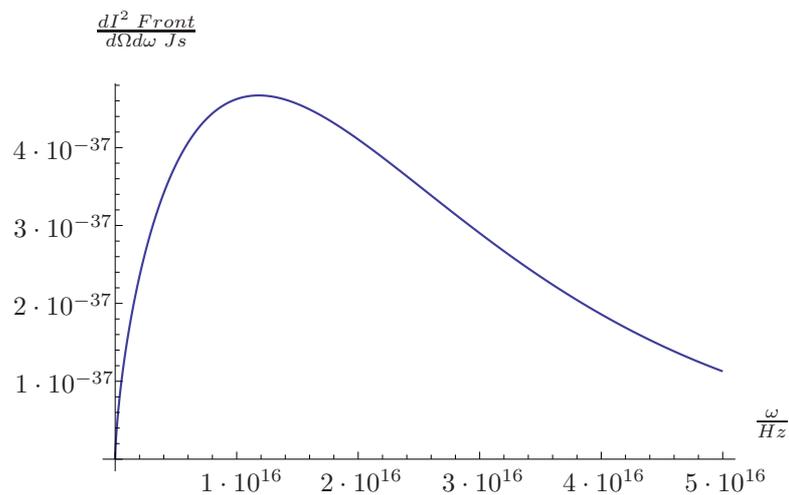


Fig. 6.7: Simulation of the highest frequency for a circular trajectory. $v = 0.9c$, $r = 400$ nm

If this simulation is repeated in Radiation3D all the values should be between these extrema. For this case the camera observer mode is chosen in order to observe the occurring frequencies from the trajectory plane and to calculate the extrema of the wavelengths with respect to the time and not with respect to the position of the observer. The actually measured shortest wavelength varies between 170 and 180nm which is close to the expected value of approx. 160nm (fig. 6.8).

⁹ One version of the script calculates the wavelengths. It can be found in the appendix.

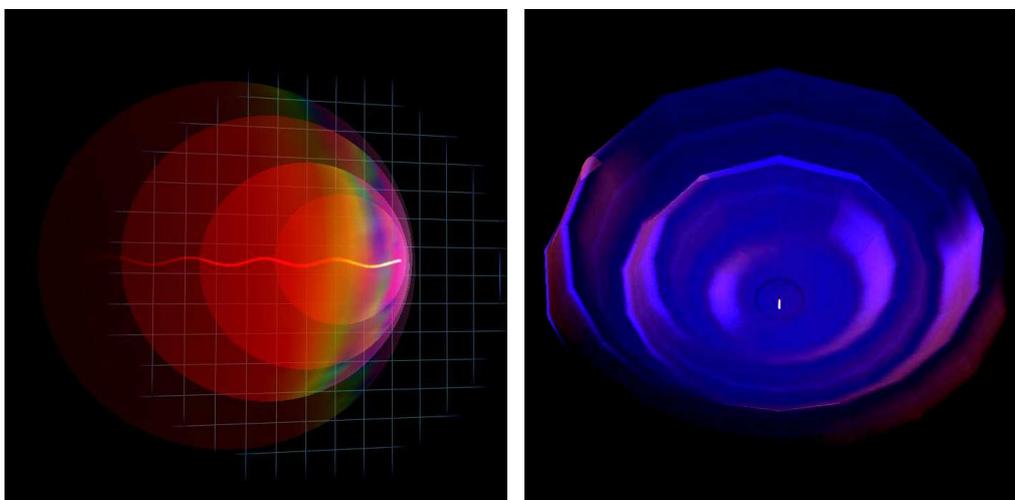
Besides the low resolution of the grid there are other error sources in Radiation3D. A circle has no local extremum. Therefore the model in which the electron emits waves at a local extremum of the trajectory cannot be applied. However, for getting any results the wave fronts are emitted like described in chapter 2.2. This generates a new wave front after every fourth of a cycle what captures only very few of the actually emitted spectrum but the repetition of shorter and longer wavelengths when the electron is approaching respectively departing can be observed.



Fig. 6.8: Circular trajectory in camera observer mode. The observer is situated in the trajectory plane.
 $v = 0.9c$, $r = 400\text{nm}$

Undulator trajectory

A charge on an undulator trajectory generates very short wave lengths towards the moving direction and very long wavelengths into the opposite direction. The simulation shows that frequencies are reached which conform to X-radiation.



(a) The wave front representation with multi observer colour mapping

(b) The wave plane representation

Fig. 6.9: The field around an electron which is moving on a sine trajectory

The script used for the circular trajectory does also calculate the wavelengths for undulators. This provides the opportunity to validate the results of this simulation. The used parameters are $v = 0.99c$ for the electron and $\lambda = 800\text{nm}$ for the undulator. There is a peak at a wavelength of 7.96nm . Starting the simulation with the same values should generate wavelengths of the same order. Fig. 6.10 shows a screenshot from Radiation3D while using the same values. There is not always a vertex which is exactly on the x-axis. In those cases the next best one is used. This way some minor variations are generated causing the wavelength to jump in an interval between 7 and 10nm . Nevertheless the difference between the calculation and the simulation is very small, e.g. around 1nm , and the results are of the same order.

The expected wavelength can also be calculated by using a simplified form of eq. 2.12 from [6].

$$\lambda = \frac{\lambda_u}{\gamma^2(1 + v/c)} \quad , \quad (6.1)$$

where λ_u is the period of the undulator and $\gamma = 1/\sqrt{1-(v/c)^2}$.

for $v \approx 1$ it reduces to

$$\lambda = \frac{\lambda_u}{2\gamma^2} \quad (6.2)$$

Thus, for an electron speed of $0.99c$ the expected wavelength emitted into the main moving direction of the electron is

$$\lambda = \frac{800\text{nm}}{2 \cdot (1/\sqrt{1-0.99^2})^2} = 7.96\text{nm}$$

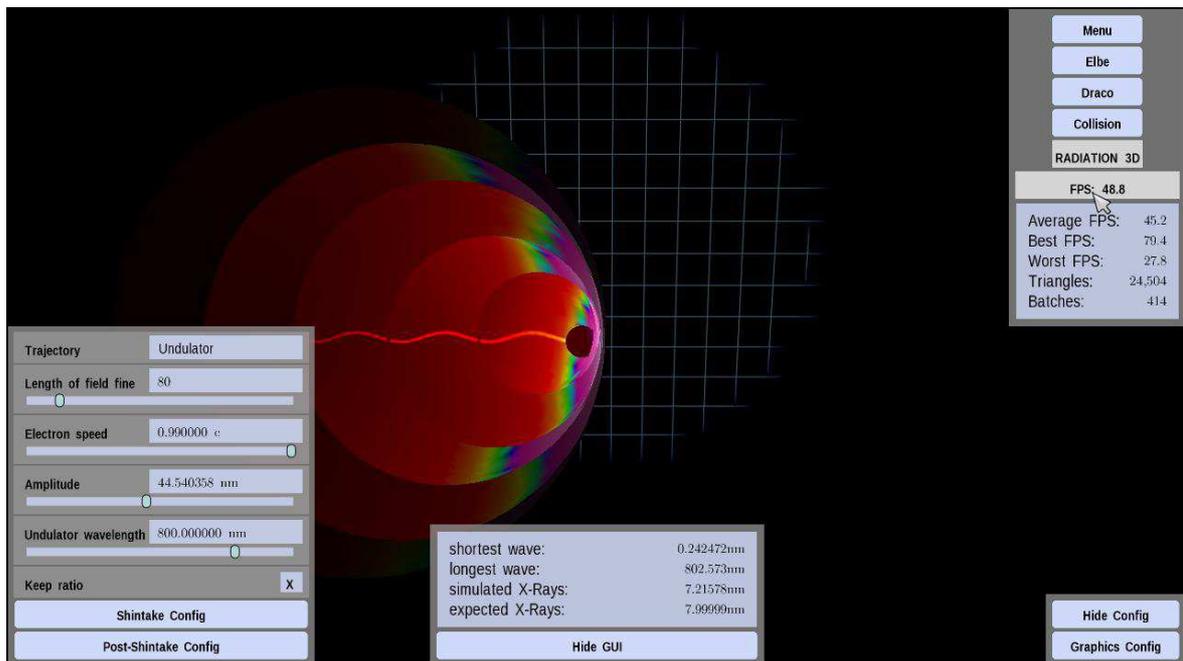


Fig. 6.10: Simulating X-Rays. “expected X-Rays” is computed by applying equation 6.1.

The wavelength in the opposite direction should be as long as the undulator period. In this example (fig. 6.10) this is 800nm . Fig. 6.10 shows the wave fronts and the trajectory as well as the longest wavelength which is measured at the back side. It shows that the longest simulated wave length is approximately 800nm .

Fig. 6.11 shows a close-up view of the wavefronts. The distances between the wave fronts vary for angles unequal to 0° and 180° . This effect has already been investigated in [6]. Fig. 6.12 shows that several frequencies are observed for the same angle. The higher this angle is, the narrower the spectrum does become. This can also be observed in fig. 6.11 and fig. 6.10.

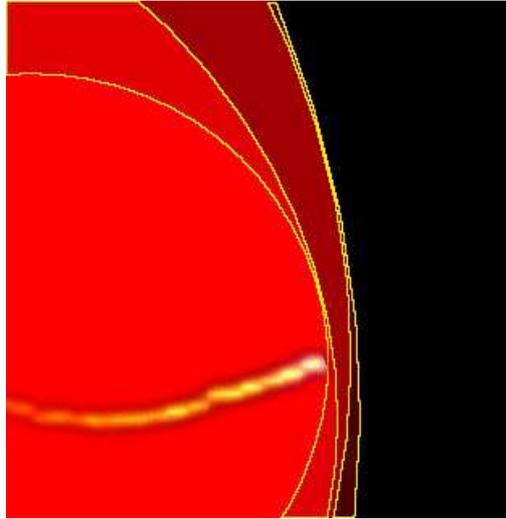


Fig. 6.11: Close-up view of the wavefronts during the undulator trajectory. There are different wavelengths for the same angle. They are visualised by different distances between the wave fronts. Additionally, the wavefronts are traced by yellow lines in this picture in order to simplify their identification.

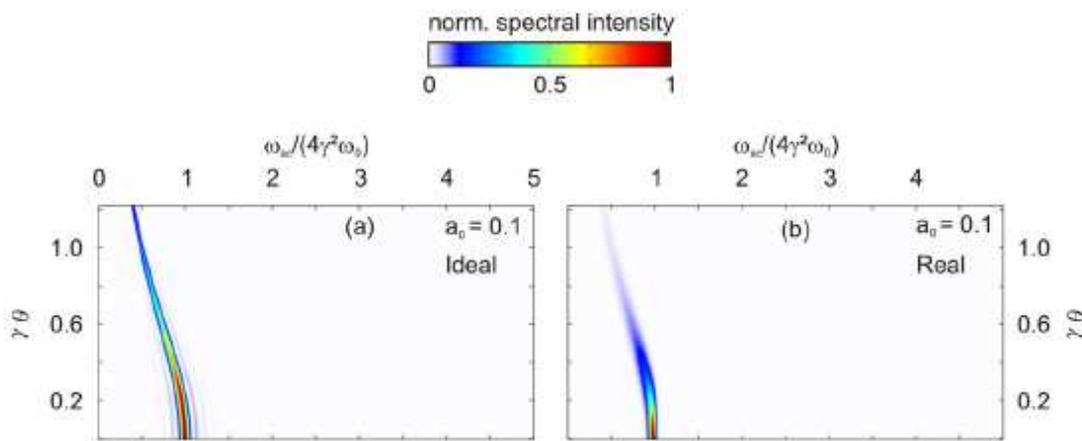


Fig. 6.12: A close-up of fig. 4.4 from [6]: “(a)-(e) show angular resolved spectral energy distributions for a range of Thomson backscattering scenarios featuring various laser strengths a_0 i.e. laser intensities and temporal laser profiles. There are three different models for laser and electrons: an ideal scenario contains a zero emittance electron bunch and a spatially flat laser profile with a rectangular temporal pulse shape [...] The scale is normalized to the maximum value for each plot.” ([6] p. 86)

6.2 Discussion

Different models and different aspects

"The theory of interaction of light with matter is called quantum electrodynamics. The subject is made to appear more difficult than it actually is by the very many equivalent methods by which it may be formulated."

Richard P. Feynman[8]

There are many methods for computing electro-magnetic fields. The challenge is not only to find and implement one which is fast enough to run in real-time but also to find representations that visualise the results in an adequate and vivid manner. After all, field lines, wave fronts, vector fields and all the other structures and algorithms are just models facilitating the comprehension of what really happens. A model

will never be able to show all aspects of reality because it is just a representation of the actual occurrence. And even with this knowledge there are several models.

One of the main differences between the implemented and most other models is, that there is no fixed grid at whose nodes the field quantities are computed. The grid which is not necessarily equidistant is built by the algorithm instead.

A challenge of this work was to present accurate information without flooding the viewer. This is one of the reasons why many different possibilities for visualising the field have been introduced, which can be activated at once, e.g. the fieldlines and the wave plane can be shown simultaneously and at every time the whole view can be clipped, so that the front parts do not hide the backmost parts. Anyway, not always all computed information can be shown. For instance, if the directions of the electrical and magnetic field are shown at the same time, they need to be distinguishable from each other. This is done using the colours red and blue. But this takes away the possibility to map the field strength via colour on the arrows.

Another issue was the question of how to show information. An electro-magnetic field cannot be seen in reality unless it is in the small spectrum of visible light, and even this spectrum depends on the velocity of the observer in respect to the observed object. Nevertheless it was a wish to show multiple observations at the same time, and for this reason the imaginary observers on the wave fronts have been introduced (chapter 4.1).

Physics and computer science

"The extension to 3D is straight forward."

Tsumoru Shintake[22]

In the version of [22] from 2002 this sentence is still existent. Nevertheless this extension was not implemented until winter 2010/11 although the first version of *Radiation2D* has already been written in 1984 [21]. One of the problems that already occurred in the practical course is the fact that it needs not only physicists for reaching this goal but also computer scientists who are well versed in modern graphics programming. This includes object-oriented programming, the knowledge of the graphics hardware abilities and their interfaces as well as the theoretical foundations of computer graphics and some experience in optimisation. While, according to the author, the first version was written in *HP-Basic*[21], a contemporary representation in 3D demands more complex techniques. Thus, the communication between physicists and computer scientists is essential which requires patience and work from both sides.

The target audience

"This looks like a jellyfish!"

various people

For the major part of its existence *Radiation3D* has only been tested as a "toy" for physicists who are familiar with the subject. But there are some occasions, like the day of the open lab at the HZDR, when it is shown to people who are not competent in this subject area where one of the purposes of this project is to make the processes of electrodynamics more perspicuous. Thus, the question how people perceive the representations is essential. Experience has shown that the concept of field lines is well adopted and usually already known. The concept of particles representing a field is not common but well adopted, too. The remaining representations needed some explanations. Especially the wave plane has rather been interpreted as a "jellyfish" than an electromagnetic wave by several visitors. Including some more explanation into the program itself could be a task for future works.

7 Outlook

Although *Radiation3D* has improved, there is still a manifold of possibilities for extending it. In addition, growing hardware resources steadily offer a wider range of potential and tools.

The trajectories are still limited. A possible scenario is not only adding fixed trajectories but making them freely defineable either by fixed positions between which a curve is interpolated, or by the forces that affect the charge. Another project would be the implementation for stereoscopic displays and different input devices, e.g. the Kinect¹⁰ or multi-touch displays. Moreover an expansion to several charges affecting each other is a logical step forward. This could accompany the transfer of many computations from CPU to GPU. Since most factors are vectors anyway, this would increase the performance noticeably if modern graphic cards are used. As already mentioned in chapter 6.2, an integration of some explanatory material is considered, too.

Besides, there is no physically established visualisation of a whole accelerated electron bunch with millions or at least thousands of electrons yet. Although this project is already in progress.

All things considered, the intention of this effort and cooperating works is making electrodynamics more than formulas and providing a way for a more intuitive understanding.

¹⁰ The Kinect is an input device which allows inputs via gestures. It was developed by Microsoft and PrimeSense and is actually intended for gaming with the Xbox 360. But there are several research projects which use it for different purposes.

Appendix

A User guide

Getting started

As this program is written using the Ogre framework, it runs under *DirectX* as well as under *OpenGL*. It has been tested under *DirectX 9* and *OpenGL 3.3*. However, using the latter is recommended as this runs distinctly faster on the systems used for testing and some *GLSL* shaders are used which only work with *OpenGL*. The program is started by running *LMC.exe*. A window opens which enables the user to choose the desired render system in a drop down menu and to adjust the configuration (fig. A.1). It should be mentioned that the default values will open the program in full screen. Pressing OK starts the main menu. (fig. A.2)

This work covers only a part of the presented subprograms, here called states, namely *Radiation3D* and parts of Collision. The remaining ones are refined by other works.¹¹

In every state there is a help opened by pressing F1 and the GUI can be hidden by pressing F3. ESC lets the user leave the state or the whole program. By choosing “Enter Radiation3D” the subprogram is started which makes the main part of this work. As no settings have been chosen yet, the electron rests (fig. A.3).

Radiation3D

The navigation

The user may rotate around the electron by holding the left mouse button while moving the mouse. It is also possible to zoom in and out by using the mouse wheel or holding the right mouse button while moving the mouse.

The main menu

In every state there is the possibility of moving to another state. For this purpose there are buttons in the upper right corner. If for any reason only the panel at the bottom of the window should be shown, the ‘Hide GUI’-Button hides all other widgets.

The panel at the bottom of the window is only updated for the undulator trajectory while the wavefronts are calculated.

For changing anything related to the trajectory of the electron, the method for the computation of the field or its dimensions like size and density, there is a menu activated by clicking ‘Charge Config’. For the visual settings, there is another menu activated by clicking ‘Graphics Config’.

The same buttons for activating a menu can be used to deactivate them.

The charge menu

For letting the electron move, the charge menu must be activated. Here the desired trajectory with its amplitude and, if necessary, its wavelength, the speed of the electron and the length of the field lines can be chosen. This menu has two submenus, ‘Shintake Config’ and ‘Post-Shintake Config’, of which only one can be active at a time. If there is one of them active and the other one is activated, the first one will be closed automatically (fig. A.9).

¹¹ When this work was written Alexander Matthes and Tino Winkler were currently working on the refinement of the Draco and Elbe part as well as on a physically correct visualisation of the whole electron bunch.

| Widget | Values[unit], default value | Remarks |
|-------------------------------------|---|---|
| Charge Config Menu | | |
| Trajectory | Dipole oscillation (default), Circle, Undulator, Linear | More trajectories like a wiggler and a helix are projected |
| Length of field line | 50-300 (the number of computational steps), 80 | Long field lines cause a lot of resource load. Thus, changing this preset is only recommended for faster systems. |
| Electron speed | 0-1[c], 0 | This value must be changed for letting the electron do anything |
| Amplitude | 10-500[nm], 100 | For the circle trajectory this is the radius of the circle. |
| Undulator wavelength | 100-1000[nm], 800 | This only has effect when the undulator trajectory is chosen. |
| Shintake Config | - | Opens a submenu for adjusting the settings corresponding to the Shintake calculation |
| Post-Shintake Config | - | Opens a submenu for adjusting the settings corresponding to the Post-Shintake calculation |
| Shintake Config Submenu | | |
| Calculate Shintake field lines | true/false, true | Enable/disable the calculation using the Shintake algorithm |
| Number of fieldlines | 8-16, 8 | The square of this value makes the number of the calculated fieldlines |
| Post-Shintake Config Submenu | | |
| Calculate Post-Shintake field lines | true/false, false | Enable/disable the calculation using the Post-Shintake algorithm |
| Min Theta | 0-1 [$\pi \cdot rad$], 0 | The initial field lines are calculated using the spherical coordinates of a unit sphere with parameters θ and φ . These settings allow to narrow the calculated section and to adjust the density of the lines (the resolution for θ and φ). |
| Max Theta | 0-1 [$\pi \cdot rad$], 1 | |
| Min Phi | 0-1 [$\pi \cdot rad$], 0 | |
| Max Phi | 0-1 [$\pi \cdot rad$], 1 | |
| Line density | 1-0.05 [$\pi \cdot rad$], 0.2 | |

Tab. A.1: The charge menu

The graphics menu

This menu enables the user to change everything that is not related to the calculation itself but the looks of the visualisation. (fig. A.4 and fig. A.5)

| Widget | Values, default value | Remarks |
|------------------------|--|--|
| Show | | |
| Field lines | true/false, true | Disabling the fieldlines detaches all field lines, no matter if they have been calculated using the Shintake or Post-Shintake method. Enabling the field lines this way always enables fieldlines using the Shintake method. |
| Particles | true/false, false | This enables a representation made out of particles following their wave vector and works even if the field lines are disabled. |
| Trajectory | true/false, true | Enabling the trajectory representation eases its tracing. |
| Wavefronts | true/false, false | Enabling the wavefronts offers more opportunities for visualising the field. |
| Waveplane | true/false, false | This enables a plane connecting the fieldlines that are initially part of the x-z-plane. |
| Grid | true/false, true | Especially when the undulator trajectory is enabled the grid illustrates the movement of the electron in space. Unlike the electron it moves on the screen when necessary. |
| Visualisation Settings | | |
| Cut view | true/false, false | This sets the near clipping distance to a value slightly smaller than the distance between observer and electron. |
| Fieldline colour mode | multi-coloured/single-coloured/ gray, multi-coloured | If single-coloured is enabled and the Post-Shintake as well as the Shintake method are used for the calculation, the fieldlines corresponding to the two methods are drawn in different colours. |
| Wavefront colour mode | multi-observer/camera-observer, multi-observer | Multi-observer colours the wavefronts corresponding to the frequencies several imaginary observers situated on the wavefronts would recognize. Camera-observer colours them corresponding to the frequency an oberver, who is situated at the position of the camera, would recognise. |
| Wavefront mapping | off/E-Field/B-Field, off | Maps the texture on the wavefronts corresponding to the vectorial dimensions of the magnetic or electric field. |
| Illumination | off/on, off | Enables the user to switch the illumination on or off. |

Tab. A.2: The graphics menu

Collision

The navigation

As this state is edited by several developers, the navigation works differently. While the rotation of the camera is done pressing the right mouse button and moving the mouse, the translation is done with the WASD keys on the keyboard. Besides there are different viewing modes the user can choose from. This is explained in the next subchapter.

| Key | Effect |
|---------------|--|
| W | Move forward |
| A | Move left |
| S | Move back |
| D | Move right |
| Shift+W/A/S/D | Doubles the speed of the camera's movement |

Tab. A.3: Keys to be used for the navigation in Collision

The menu

There is only one menu in this state so far. The buttons for changing the state are in the upper right corner again. In the lower right corner there is a slider which enables the user to adjust the speed of the animation. All other widgets, of which there are only buttons in this state, are on the right side of the window.

| Button | Effect |
|---------------------|--|
| Play/Pause | Plays/pauses the animation. |
| Stop | Stops the animation. To restart the animation it is necessary to stop the animation first. |
| Auto SlowMo on/off | If auto slowmo is enabled the animation speed decreases automatically shortly before the collision takes place . |
| Free view | Enables the user to fly around using the WASD keys. |
| Fixed Height on/off | Limits the free view by defining a fixed height in which the camera is situated |
| Follow electron | Attaches the camera to the electron. (fig. A.7, A.8) |
| Show/Hide cover | The cover of the experimental rig is shown by default. For seeing the test arrangement without zooming in, it needs to be hidden. (fig. A.6) |
| Show/Hide path | The paths that the laser and the electron bunch follow is shown by default. This can be disabled. |

Tab. A.4: The menu of Collision

Trouble shooting

Although this program has been tested on different systems with different configurations, there still is a small but existing chance that errors may occur preventing the program from running. Here are some solutions presented for the problems most likely to appear.

| Error message | Possible Reason | Solution |
|--|---|---|
| Could not load dynamic library .\RenderSystem[.] | A dll for at least one render system is missing, most likely because DirectX10 or DirectX11 is not installed. | There are two ways for solving this problem. The easiest way is disabling the render system by opening plugins.cfg and adding a '#' in front of the name of the corresponding plugin. The other solution is installing the render system and copying the dll into the release folder. |
| <ul style="list-style-type: none"> • Cannot find required template 'X'[.] • [..]error whilst opening archive: Unable to read zip file in 'X'[.] • Cannot locate resource 'X'[.] | One or more resources could not be found. | Most likely the 'media' folder is not where resources.cfg tells it is. This problem should be solved by checking the paths in resources.cfg and either moving the 'media' folder or changing the paths. |
| LMC.exe has stopped working | Practically everything can cause this error but usually there are only some resources not found. | If this has happened when the program was running, it can simply be restarted. If this error occurs during the start of the program, it is recommended to check the paths in resources.cfg as missing resources can cause this error, too. |

Tab. A.5: Known error messages



Fig. A.1: Choosing the render system and its configuration

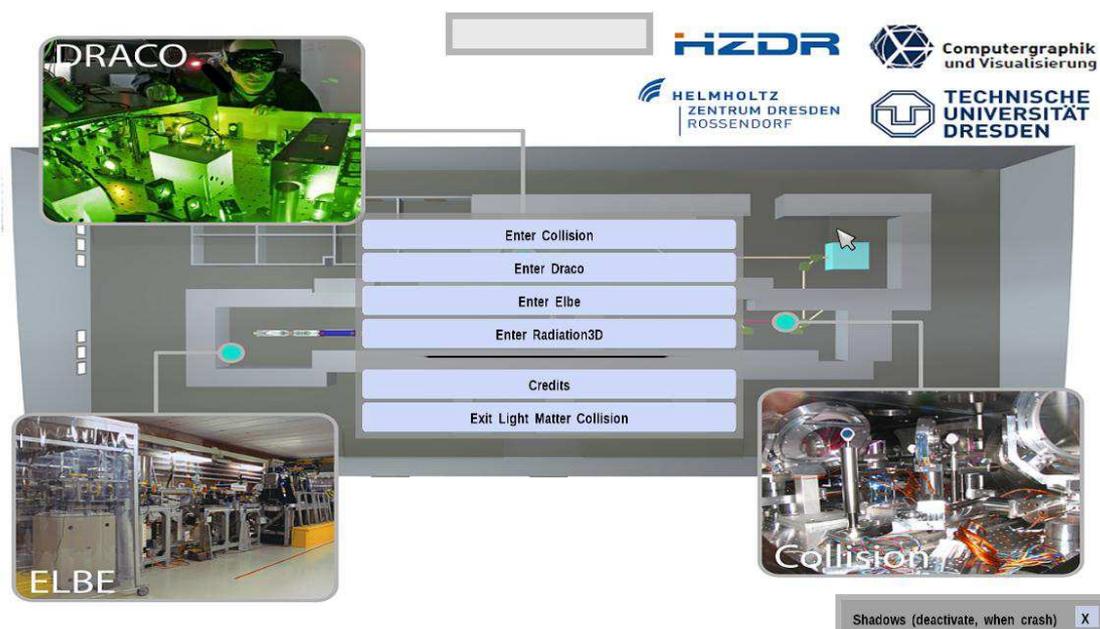


Fig. A.2: The start screen

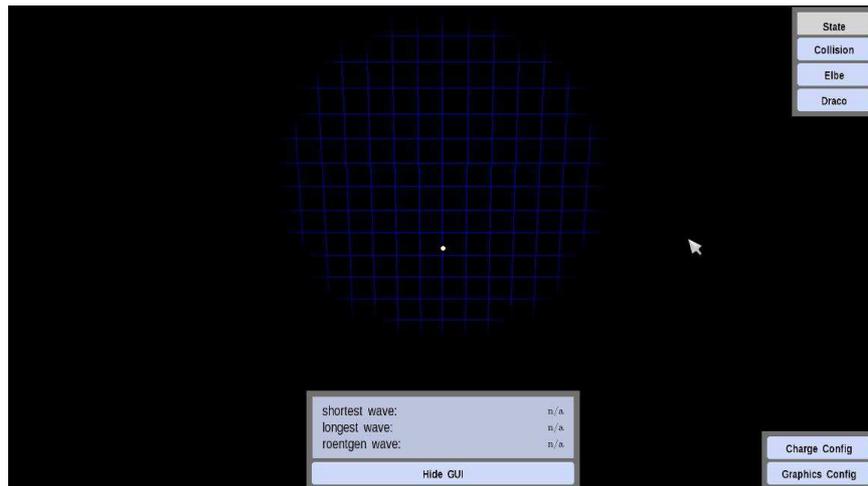


Fig. A.3: This is how Radiation3D welcomes the user

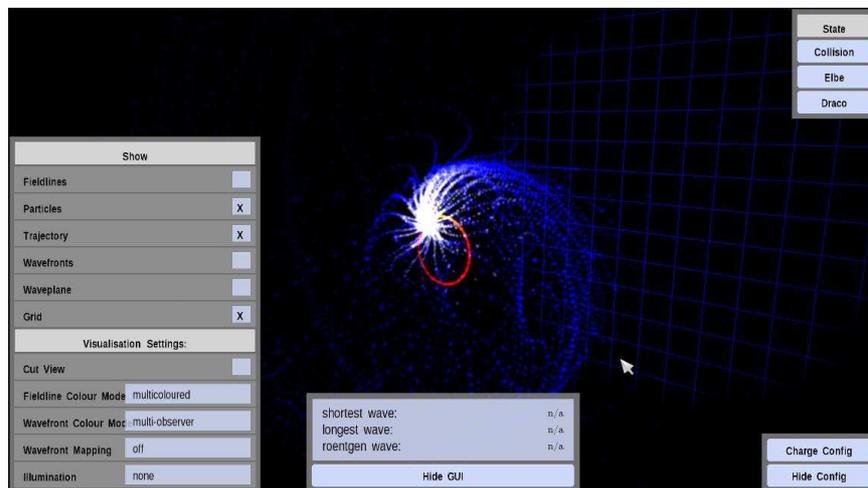


Fig. A.4: The graphics menu

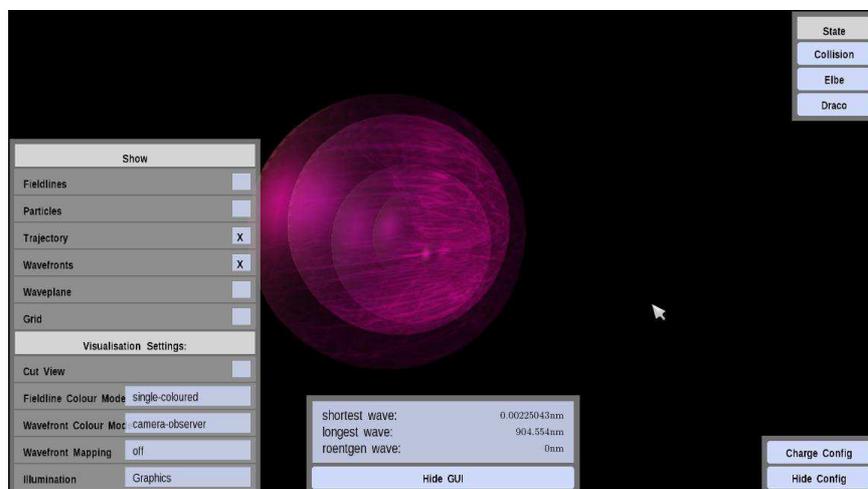


Fig. A.5: Wavefronts in camera-observer mode with lightning enabled

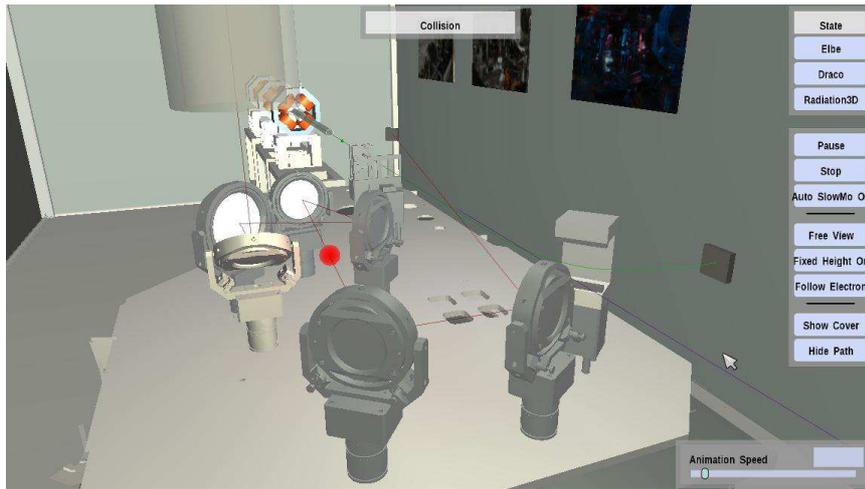


Fig. A.6: The experimental rig without the cover

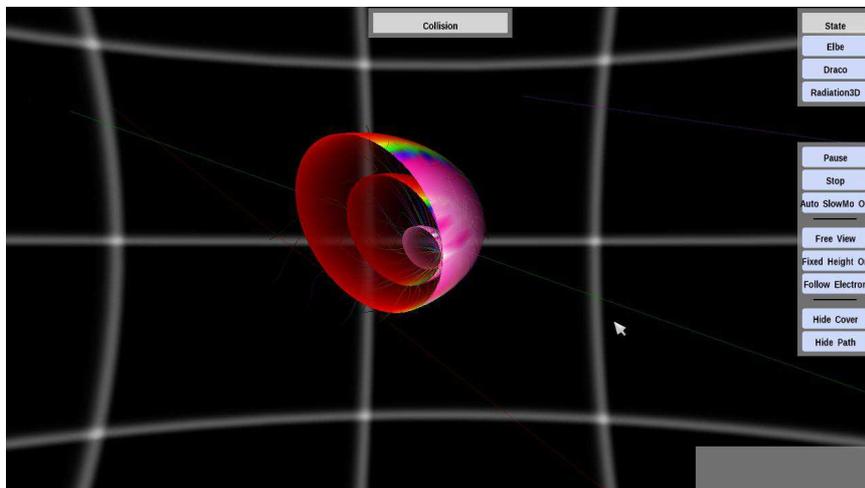


Fig. A.7: The wavefronts during the collision

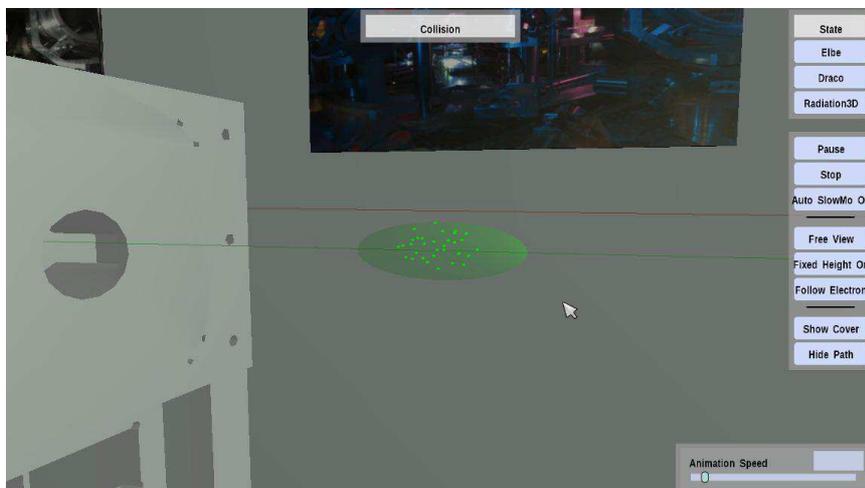


Fig. A.8: If enabled the camera follows the electron. The visibility of the whole bunch or just a single electron depends on the distance to the bunch.

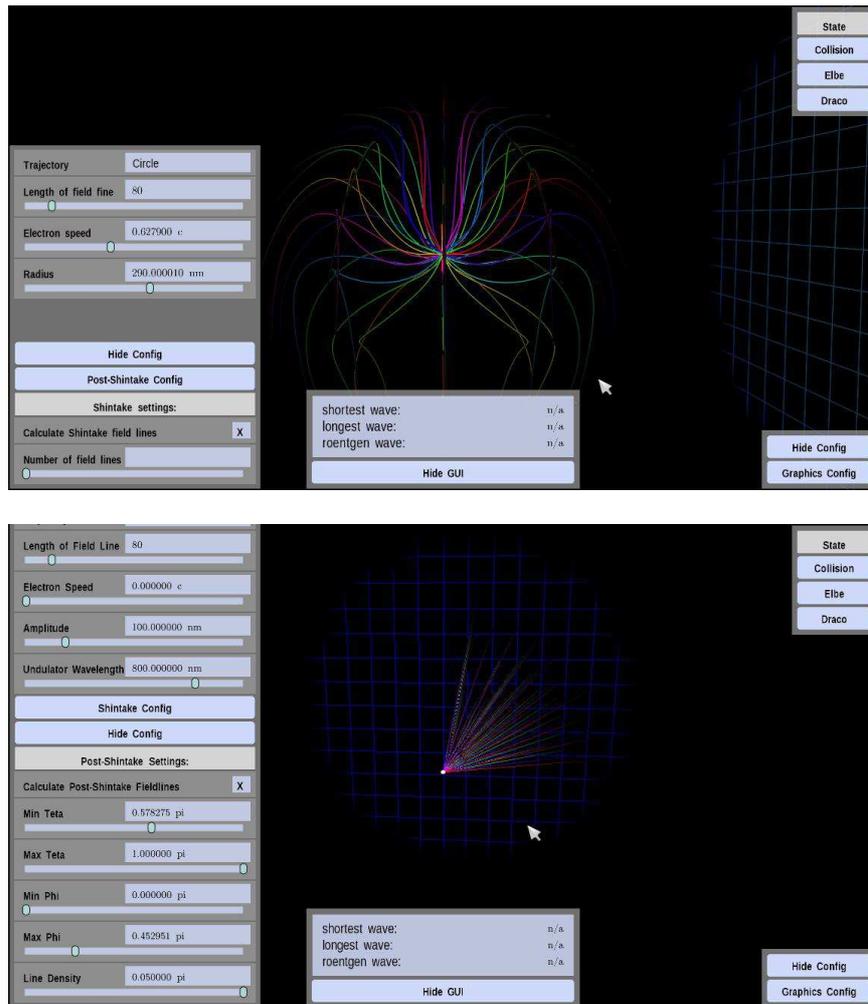


Fig. A.9: The charge menu and the Post-Shintake submenu

B Blinn-Phong shader in GLSL

Listing 3: The vertex program

```

uniform vec3 LightPosition;
2 uniform vec3 EyePosition;
varying vec3 ViewDirection;
varying vec3 LightDirection;
varying vec3 Normal;
varying vec3 color;

void main( void ) {
    vec4 ObjectPosition = gl_ModelViewMatrix * gl_Vertex;
    ViewDirection      = EyePosition - ObjectPosition.xyz;
    LightDirection     = LightPosition - ObjectPosition.xyz;
12 Normal              = gl_Normal;
    color               = gl_Color;
    gl_Position        = ftransform();
}

```

Listing 4: The fragment program

```

uniform vec4 amb;
uniform vec4 spec;
uniform vec4 diff;
uniform int blinnphong;
5 uniform float shin;
varying vec3 ViewDirection;
varying vec3 LightDirection;
varying vec3 color;
varying vec3 Normal;

void main( void ) {
    vec3 LightDir = normalize(LightDirection);
    vec3 Norm     = normalize(Normal);
    float NDotL   = dot(Norm, LightDir);
15 vec3 Reflection = normalize(((2.0 * Norm) * NDotL) - LightDir);
    vec3 ViewDirection = normalize(ViewDirection);
    float RDotV     = max(0.0, dot(Reflection, ViewDirection));
    vec4 TotalAmbient = amb * color;
    vec4 TotalDiffuse = diff * NDotL * color;
20 vec4 TotalSpecular;
    vec3 Halfvector   = (ViewDirection + LightDir)/normalize(ViewDirection +
        LightDir);
    float NDotH       = max(0.0, dot(Norm, normalize(Halfvector)));
    if (blinnphong == 1){
        TotalSpecular = spec * color * (pow(NDotH, shin)) ;
25 }
    else{
        TotalSpecular = spec * color * (pow(RDotV, shin)) ;
    }
30 gl_FragColor      = (TotalAmbient + TotalDiffuse + TotalSpecular);
}

```

C Mathematica scripts and sample points

Listing 5: Undulator script by Richard Pausch

```

Undulator
λ0 = 800*10-9 ;(*Undulator length [m] *)
clight = 2.9979*108 ;(* speed of light [m/s] *)
mass = 9.11*10-31; (* electron mass [kg] *)
5 echarge = 1.6022*10-19; (* electron charge [C]*)
ψ[a_] := 2*π*a/λ0 (* fist derivative of trajectory [] *)
Kvalue1[a_, γ_] := γ*ψ[ a] (* K value using ψ [] *)
Kvalue2[B_] := echarge*B*λ0/(2*π*mass*clight2) (* K value using magnetic field [] *)
GammaCalc[β_] := Sqrt[1/(1 - β2)] (*calculated gamma from β=v/c [] *)
10 LambdaCalc[omega_] := clight*2*π/(omega)
(* calculates wavelength from circular frequency ω [m] *)
ω[θ_, γ_] := 2*γ2/(1 + γ2*θ2)*(2*π* clight/λ0)
(* main radiation peak of undulator [1/s]*)
Kvalue1[10-15, GammaCalc[ 0.99]] < 0.001
15 (* checks if assumption for a Undulator are correct *)
Solve[Kvalue1[10-15, GammaCalc[0.99]] == Kvalue2[myB], myB]
(* B field caused by a=10-15m and β=0.99 [T] *)
LambdaCalc[ω[0.0, GammaCalc[0.99]]] (* wavelength of main radiation peak [m] *)

20 B_field
clight = 2.99792458*8;(* speed of light [m/s] *)
echarge = 1.6022*10-19; (* electron charge [C]*)
mass = 9.11*10-31; (* electron mass [kg] *)
ε0 = 8.85418781762*-12; (* [As/Vm] *)
25 ω[f_] := 2*π*f (* circular frequency [1/s] *)
f[T_] := 1/T (* frequency [1/s] *)
LambdaCalc[omega_] := clight*2*π/(omega)
GammaCalc[β_] := Sqrt[1/(1 - β2)] (*calculated gamma from β=v/c [] *)
BetaCalc[gamma_] := 1.0 - 1.0/gamma2
30 circumference[r_] := 2*π*r (* circumferency of circle [m] *)
T[r_, β_] := circumference[ r]/(β*clight) (* period of circular movement [s] *)
CyclotronOmega[B_] := echarge*B/mass (* cyclotron frequency [1/s] *)
CyclotronRadius[β_, B_] := clight*β/CyclotronOmega[B] (* cyclotron radius [m] *)
Radius = 0.4*10-6; (* [m] *)
35 betaStart = 0.903;(* [] *)
Bfield = x/.Solve[Radius == CyclotronRadius[betaStart, x], x][[1]]
(* B filed needed [T] *)
LambdaCalc[CyclotronOmega[Bfield]] (* cyclotron repetition as wavelength [m] *)

40 ξ[ω_, θ_, ρ_, γ_] := ω * ρ / (3*clight) * (1/γ2 + θ2)(3/2)
dWdΩdω[ω_, θ_, ρ_, γ_] := 1.0/(4*π*ε0) * echarge2/(3*π2*clight)*(ω*ρ/clight)2 *
(1/γ2+θ2)2 * ((BesselK[2/3,ξ[ω, θ, ρ, γ]])2 + θ2/(1/γ2 + θ2) * (BesselK[1/3, ξ[ω,
θ, ρ, γ]])2)

Plot[dWdΩdω[omega, 0, Radius, GammaCalc[betaStart]], {omega, 0, 5*16}, PlotRange -> All,
AxesLabel -> {ω/Hz, dI2/(dΩdω*Js) "Front"}, PlotPoints -> 400]
Plot[dWdΩdω[omega, π, Radius, GammaCalc[betaStart]], {omega, 0, 2*14}, PlotRange -> All,
AxesLabel -> {ω/Hz, dI2/(dΩdω*Js) "Back"}, PlotPoints -> 400]

LambdaCalc[myomega/. FindMaximum[{dWdΩdω[myomega, 0, Radius, GammaCalc[betaStart]] },
{myomega, 0.5*1016, 2*1016}]][[2]]]
LambdaCalc[myomega/. FindMaximum[{dWdΩdω[myomega, π, Radius, GammaCalc[betaStart]] },
{myomega, 0.2*1014, 1*1014}]][[2]]]

```

Listing 6: Dipol - distance between field lines in near zone and far zone

```

<< NonlinearRegression (*Load regression package*)
(*List of distances between two field lines, format:{i,1/distance}*)
3 far = {{37, 0.34378946058402265}, {59, 0.21984259709733026}, {81, 0.1542036453433352},
{103, 0.12851384151904388}, {124, 0.1027340193637189}, {146, 0.08870286518237797},

```

```

    {168, 0.0744504957807415}, {190, 0.06955553526079604}, {211, 0.060463951996459236},
    {233, 0.05549610997242343}, {255, 0.04911768430158533}, {277, 0.04763285207322465},
    {298, 0.04287563634411626}} (*intermediate and far zone*)
near = {1/0.085586, 1/0.169505, 1/0.252335, 1/0.335249, 1/0.419599, 1/0.506465,
1/0.596227, 1/0.688219, 1/0.780582, 1/0.870394, 1/0.954162, 1/1.028565, 1/1.091220,
1/1.141196, 1/1.179101, 1/1.206788, 1/1.226852, 1/1.242147, 1/1.255415, 1/1.269094};
(*near zone*)
allpoints = {{1, 1/0.085586}, {2, 1/0.169505}, {3, 1/0.252335}, {4, 1/0.335249}, {5,
1/0.419599}, {6, 1/0.506465}, {7, 1/0.596227}, {8, 1/0.688219}, {9, 1/0.780582}, {10,
1/0.870394}, {11, 1/0.954162}, {12, 1/1.028565}, {13, 1/1.091220}, {14, 1/1.141196},
{15, 1/1.179101}, {16, 1/1.206788}, {17, 1/1.226852}, {18, 1/1.242147}, {19,
1/1.255415}, {20, 1/1.269094}, {37, 0.34378946058402265}, {59, 0.21984259709733026},
{81, 0.1542036453433352}, {103, 0.12851384151904388}, {124, 0.1027340193637189}, {146,
0.08870286518237797}, {168, 0.0744504957807415}, {190, 0.06955553526079604}, {211,
0.060463951996459236}, {233, 0.05549610997242343}, {255, 0.04911768430158533}, {277,
0.04763285207322465}, {298, 0.04287563634411626}} (*near && far*)
(*Perform nonlinear regression for the near zone*)
8 g3 = NonlinearRegress[near,  $\frac{a}{(e+b)^3} + c$ , {a, b, c}, e, Weights -> {10, 4, 4, 4, 4, 4, 4, 2, 2,
2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1}]
(*Perform nonlinear regression for the far zone*)
g4 = NonlinearRegress[far,  $\frac{a}{(e+b)^3} + c$ , {a, b, c}, e]
(*Perform nonlinear regression for all values*)
g5 = NonlinearRegress[allpoints,  $\frac{a}{(e+b)^3} + \frac{d}{(e+f)^3} + c$ , {a, b, c, d, f}, e]
13 (*Plot data and results*)
Show[ListPlot[near, Joined -> False, PlotStyle -> Orange], Plot[ $\frac{800}{(x+3.4)^3} + 1$ , {x, 0, 20},
PlotStyle -> Red], Plot[ $\frac{600}{(x+2.8)^3} + 0.9$ , {x, 0, 20}, PlotStyle -> Blue], AxesLabel -> {"i",
"1/distance"}]
Show[Plot[ $\frac{13.05}{x+0.84}$ , {x, 0, 300}], ListPlot[allpoints, Joined -> False, PlotStyle -> Orange],
AxesLabel -> {"i", "1/distance"}]
Show[Plot[ $\frac{122}{(x+1.52)^3} + \frac{14.39}{x+2.56}$ , {x, 0, 20}], ListPlot[allpoints, Joined -> False, PlotStyle ->
Orange], AxesLabel -> {"i", "1/distance"}]
Show[Plot[ $\frac{122}{(x+1.52)^3} + \frac{14.39}{x+2.56}$ , {x, 0, 300}], ListPlot[allpoints, Joined -> False, PlotStyle ->
Orange], AxesLabel -> {"i", "1/distance"}]

```

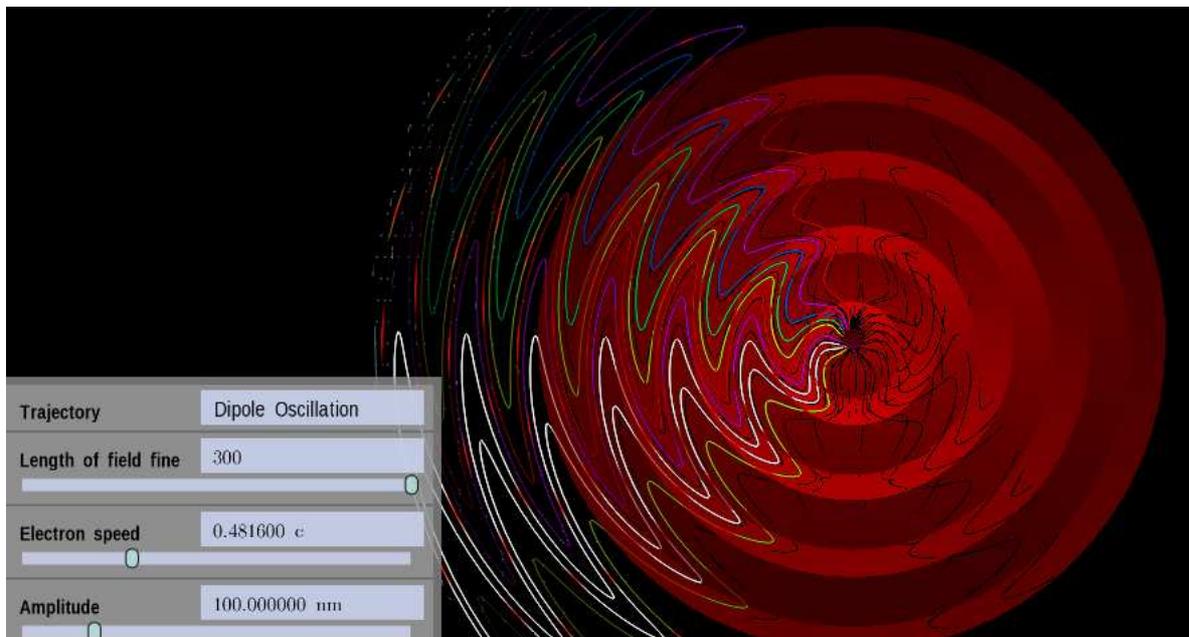


Fig. C.1: This screenshot was made for visualising the sample points. The white field lines are used for measuring the distances. The red spots on these field lines beginning from $i=20$ are the positions between which the distance is measured.

D Benchmark

For the benchmark an electron speed of $0.73c$ has been chosen. All other values are the default values. Therefore, the line length is 80 and the chosen trajectory is the dipol trajectory what produces about four simultaneously visible wave fronts. The last column shows the framerate when the update routines for the drawn objects are not called and only the physical computation is executed.

| drawn objects | fps | triangles | batches | fps without drawing |
|----------------------------|-----|-----------|---------|---------------------|
| shintake field lines | 88 | 48,042 | 653 | 189 |
| post-shintake field lines | 101 | 20,290 | 273 | 200 |
| wave fronts (plain) | 62 | 24,556 | 428 | 196 |
| wave fronts (vector field) | 33 | 14,332 | 428 | 193 |
| particles | 53 | 29,262 | 419 | 72 |
| wave plane | 44 | 8,494 | 434 | 188 |

Tab. D.1: Benchmark of *Radiation3D* on: Windows7 64-bit, 4 GB RAM, AMD Phenom II 1035T 2.6 GHz, ATI Radeon HD 4200 (onboard with HyperMemory)

The memory usage stays between 90 KB and 100 KB and the CPU usage between 16% and 17%. Since all threads are executed at the same core this core is operated at full capacity. These values are only marginally affected by the chosen drawn objects, e. g. less than 5 KB difference in the memory use und commit.

References

- [1] Ogre wiki, <http://www.ogre3d.org/tikiwiki/>.
- [2] Computational Information Systems Laboratory at the National Center for Atmospheric Research. Vapor: Visualization and analysis platform for ocean, atmosphere, and solar researchers.
- [3] B. Cabral and L. C. Leedom. Imaging Vector Fields Using Line Integral Convolution. *Proceedings of ACM SIGGRAPH '93*, 1993.
- [4] D. Darmofal and R. Haimes. Visualization of 3-D Vector Fields: Variations on a Stream. *Proceedings AIAA 30 th Aerospace Science Meeting and Exhibit*, 1992.
- [5] W. C. de Leeuw and J. J. van Wijk. Enhanced spot noise for vector field visualization. *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 1991.
- [6] Alexander Debus. *Brilliant radiation sources by laser-plasma accelerators and optical undulators*. PhD thesis, 2012.
- [7] Maarten H. Everts, Henk Bekker, Jos B. T. M. Roerdink, and Tobias Isenberg. Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1299–1306, 2009.
- [8] Richard P. Feynman. *Quantum electrodynamics*, chapter 1, page 13. 1962.
- [9] D. Fleisch. <http://www4.wittenberg.edu/maxwell/>.
- [10] D. Fleisch. *A Student's Guide to Maxwell's Equations*. 2008.
- [11] A. Ford and A. Roberts. *Colour Space Conversions*. b, 1998.
- [12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. 1995.
- [13] S. Gumhold. Computergraphik I-II, lecture notes, 2008-2012.
- [14] S. Gumhold. Wissenschaftliche Visualisierung, lecture notes, 2011-2012.
- [15] S. Gumhold, N. v. Festenberg, M. Bussmann, F. Röser, A. Matthes, A. Ungethüm, B. Schneider, J. Völker, S. Wagner, and T. Helmig. *Komplexpraktikum Computergraphik*, 2010-2011.
- [16] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. *Proceedings of the 3rd conference on Visualization '92*, 1992.
- [17] J. D. Jackson. *Klassische Elektrodynamik*. 4th edition.
- [18] R. Jacobs. Numerische Feldberechnung, lecture notes, 2011-2012.
- [19] Richard Pausch. Mathematica script for radiation calculation, 2012.
- [20] W. J. Schroeder, C. R. Volpe, and W. E. Lorensen. The Stream Polygon: A Technique for 3D Vector Field Visualization. *Proceedings of the 2nd conference on Visualization '91*, 1991.
- [21] T. Shintake. <http://www.shintakelab.com>.
- [22] T. Shintake. New real-time simulation technique for synchrotron and undulator radiations. *Proceedings of LINAC2002, Gyeongju, Korea*, 2002.
- [23] R. Stenzel and W. Klix. *Fields*, 1992-2008.
- [24] H. Störrle. *UML 2 für Studenten*. 2005.
- [25] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Interactive Visualization of 3d-vector fields using illuminated stream lines. *Proceedings of the 7th conference on Visualization '96, IEEE Computer Society Press Los Alamitos, CA, USA*, pages 107–113, 1996.