3D Visualization of an Electron Wave Packet in an Intense Laser Field

Ryan Sandberg

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Justin Peatross, Advisor

Department of Physics and Astronomy

Brigham Young University

June 2013

ABSTRACT

3D Visualization of an Electron Wave Packet in an Intense Laser Field

Ryan Sandberg
Department of Physics and Astronomy
Bachelor of Science

We present the 3D visualization of an electron wave packet in an intense laser field. Our visualization is based on a virtual camera model. The camera can be rotated around the wave packet, allowing the wave packet to be seen from any angle. The camera program integrates probability density along the line of sight and displays the calculated wave packet as a cloud. The probability density relies on a formula for calculating the wave packet in an intense laser field that was developed previously. To improve the 3D perspective of the images, we display coordinate lines in the wave packet visualization. We apply our camera model to an electron in an intense laser field and show frames from the movies we made. Features of note include the helical path of the wave packet in circularly polarized light and the multi-peaked structure that develops when the wave packet reaches the size of a laser wavelength.

This work was supported by the National Science Foundation (Grant No. PHY-0970065).

Keywords: [3D visualization, orthographic projection, Klein-Gordon equation]

ACKNOWLEDGMENTS

I want to thank my advisor, Dr. Peatross, for giving me a project when I didn't know what I was doing or what I wanted to do. This project was his idea. His guidance and encouragement were most helpful in overcoming the difficulties that came up. This has been my first experience as a professional scientist and Dr. Peatross has been a great mentor throughout the journey.

I am grateful to Dr. Ware for his help in improving the movies and images presented in this thesis. His feedback and instruction were beneficial as I wrote this thesis, and his work on the department website let me finally submit this thesis.

Chapter 1

Introduction

Our lab group works with high intensity lasers. We achieve intensities on the order of 10^{18} W/cm^2. The field strength at this intensity is enough to ionize atoms and accelerate the ionized electrons to relativistic velocities. An accelerating charged particle like an electron radiates electromagnetic energy. The predicted radiation depends on the theoretical treatment of the particle. A classical point particle, for example, radiates differently than a charge distribution. Our research group is investigating, both experimentally and theoretically, how a single electron radiates.

The theoretical models explore what an electron does in an intense laser. They predict how the electron moves and how it radiates. The full treatment of an electron in this situation requires advanced analysis beyond the scope of this thesis, but interested readers can find such treatment in papers by John Corson of our group [1], [2]. However, we obtain a good understanding of the motion of the wave packet from the Klein-Gordon equation.

1.1 Klein-Gordon Equation

An electron moving at relativistic speeds can be modeled by the Klein-Gordon equation,

$$(i\hbar\frac{\partial}{\partial t})^2\Psi = (-i\hbar\nabla - q\vec{A})^2c^2\Psi + m^2c^4\Psi. \tag{1.1}$$

Here, Ψ is the wave function and $\vec{A}$ is the vector potential that represents the laser field. The Klein-Gordon equation is similar to the Schrodinger equation, but it accounts for relativistic effects if one is willing to neglect spin [3]. Solutions to the Klein-Gordon equation can be expressed in terms of basis wave functions. A convenient basis for solutions to the Klein-Gordon equation consists of plane waves called Volkov states [3]. Each Volkov state $\Psi^V_{\vec{p}}(\vec{r},t)$ corresponds to a particular momentum $\vec{p}$. Each state has the form:

$$\Psi^V_{\vec{p}}(\vec{r},t) = \sqrt{\frac{mc^2}{(2\pi\hbar)^3E}}exp\left\{i\frac{(\vec{p}\cdot\vec{r}-Et)}{\hbar} + \frac{i}{\hbar(p_z-E/c)}\int_{-\infty}^{z-ct}\left[q\vec{p}\cdot\vec{A}(l) - q^2A^2(l)/2\right]\mathrm{d}l\right\}. \tag{1.2}$$

$E = \sqrt{p^2c^2 - m^2c^4}$ is the energy of the electron and $\vec{A}$ is the vector potential representing the laser field. In a unidirectional electromagnetic wave, any solution to the Klein-Gordon equation can be expressed as a linear superposition of Volkov states. We construct a wave packet with a superposition of Volkov states,

$$\Psi(\vec{r},t) = \int a(\vec{p})\ \Psi^V_{\vec{p}}(\vec{r},t)\ d^3p, \tag{1.3}$$

where $a(\vec{p})$ is the coefficient that determines the shape of the wave packet.

The above integration is difficult to perform. Authors involved in such calculations say as much in their publications [4]. As seen from the superposition formula, we must integrate over all momenta in three dimensions. Because $E = \sqrt{p^2c^2 + m^2c^4}$, the integral over momentum requires integration of two $p^{-1/2}$ expressions. Further, the exponential in the integrand has a complex argument, and it oscillates rapidly. To perform such an oscillatory integral numerically requires a very fine grid. Because of this difficulty, numerical solutions are often found in 2D space instead of 3D space, but even these can be computationally intensive. Other researchers have found clever ways to speed up these integrals and to display their answers, but largely they are still in 2D space [5], [6].

1.2 Analytic Formula

In 2007, J. Peatross et al. developed an analytic formula for the wave packet superposition integrals in the special case of a Gaussian distribution [3]. The formula relies on some strategic approximations akin to the Fresnel approximations used in optical diffraction. To derive the formula, first we choose to make a Gaussian wave packet. This means that in Eq. (1.3), we let

$$a(\vec{p}) = \frac{1}{(p_w\sqrt{\pi})^{3/2}} \exp\left\{-\frac{p^2}{2p_w^2}\right\}, \tag{1.4}$$

where p_w is the initial spread in momentum of the electron.

We can write out the superposition integrals in Eq.(1.3) but can't yet perform them. Recall that we integrate over the Volkov states (Eq.(1.2)), each of which has $\vec{p}$ appearing under radicals in denominators. To simplify these expressions so we can integrate, we expand E in p as follows:

$$E = mc^2\sqrt{1+\frac{p^2}{m^2c^2}} \cong mc^2\left[1+\frac{p_x^2+p_y^2+p_z^2}{2m^2c^2} - \frac{(p_x^2+p_y^2+p_z^2)^2}{8m^4c^4}\dots\right]. \tag{1.5}$$

Then to zeroth order in p we can say $1/\sqrt{E} \cong 1/\sqrt{mc^2}$. We also have

$$\frac{1}{E-cp_z} \cong \frac{1}{mc^2}\left[1+\frac{p_z}{mc} - \frac{p_x^2+p_y^2-p_z^2}{2m^2}c^2 - \frac{p_x^2p_z+p_y^2p_z}{m^3c^3}\dots\right]. \tag{1.6}$$

These expansions are valid as long as the initial spread in momentum p_w is small.

We insert the above expansions into the superposition integral, evaluate, and simplify what we can. After the dust settles, we arrive at the following formula for the probability density ρ:

$$\rho = b^2 \left(\frac{p'_w}{\sqrt{\pi}}\right)^3 \frac{1}{|\alpha|^2|\beta|} e^{-{p'_w}^2 \mathrm{Re}\left(\frac{\eta_x^2+\eta_y^2}{\alpha}+\frac{\eta^2}{\beta}\right)}$$
$$\times \left[\dot{\eta}_t^- + \mathrm{Im}\left\{\frac{\dot{\alpha}}{\alpha}\left(1-{p'_w}^2\frac{\eta_x^2+\eta_y^2}{2\alpha}\right)+\frac{\dot{\beta}}{2\beta}\left(1-{p'_w}^2\frac{\eta^2}{\beta}\right)+{p'_w}^2\left(\frac{\eta_x\dot{\eta}_x+\eta_y\dot{\eta}_y}{\alpha}+\frac{\eta\dot{\eta}}{\beta}\right)\right\}\right], \tag{1.7}$$

where

$$b = \frac{mc\lambda}{\hbar}, \quad p'_w = \frac{p_w}{mc},$$
$$\dot{\eta} \equiv \dot{\eta}_z - i{p'_0}^2\frac{\dot{\alpha}}{\alpha^2}(f_x\eta_x+f_y\eta_y)+i\frac{{p'_0}^2}{\alpha}(\dot{f}_x\eta_x+f_x\dot{\eta}_x+\dot{f}_y\eta_y+f_y\dot{\eta}_y),$$
$$\dot{\alpha} \equiv ip{-_0}^2\dot{\eta}_t^+, \quad \dot{\beta} \equiv i{p'_0}^2\dot{\eta}_t^- + 2{p'_0}^4\frac{f_x\dot{f}_x+f_y\dot{f}_y}{\alpha} - {p'_0}^4\frac{f_x^2+f_y^2}{\alpha^2}\dot{\alpha},$$
$$f_x = b\int_{-\infty}^{z'-t'} A'_x \mathrm{d}l', \quad f_y = b\int_{-\infty}^{z'-t'} A'_y \mathrm{d}l', \quad f = \frac{b}{2}\int_{-\infty}^{z'-t'} {A'}^2 \mathrm{d}l',$$
$$\eta_x = bx'+f_x, \ \eta_y = by'+f_y, \ \eta_z = bz'+f, \text{ and } \eta_t^{\pm} = bt' \pm f. \tag{1.8}$$

In the images presented in this paper, λ is set to 800 nm, giving a b value of 2.07×10^6. The parameter p'_w is set to .005.

There are several levels of nested expressions, making it difficult to glean insight directly from the formula. However, evaluating this formula doesn't require the computer to do costly oscillatory integrals; the formula can be evaluated much faster than numerical solutions can be determined.

1.3 Visualization Options

When the above formula was first developed in 2007, it was displayed as a 2D slice through the 3D wave packet [3]. As we see in Fig. 1.1, this slice shows the growth of the wave packet and the structure imposed by the intense laser. However, there is more structure to the wave packet in 3D space than can be seen in this 2D slice.

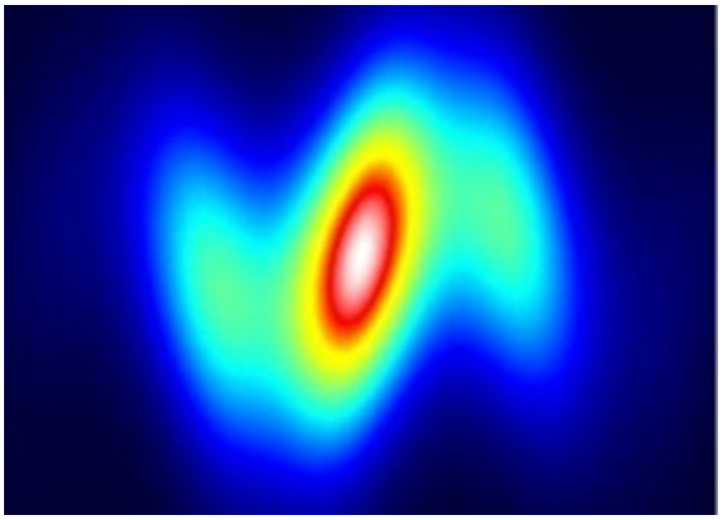


Figure 1.1 Our first view of the formula in Eq. 1.7 is a 2D slice through the 3D wave packet. This slice is in the xz plane. The image is colored by probability density, blue being the least significant density and white being the largest density values.

There are many approaches to displaying quantum waves beyond taking slices. Common techniques include making contour plots of probability density and displaying "clouds" of probability density. The contour plot displays the wave packet as a three-dimensional solid. The interpretation is that there is some percent chance of finding the particle within the volume of the solid. Fig. 1.2 shows a 45% contour plot of a hydrogen orbital, meaning that for an electron in this state of the hydrogen atom, there is a 45% chance of finding the electron in this volume. The cloud visualization displays the wave packet as a cloud. For example, Fig. 1.3 from an online applet [8] displays a hydrogen orbital as a cloud of probability density, with indeterminate boundaries. There isn't a well-defined volume containing the wave packet, just as the wave packet of a free particle isn't confined to a finite volume. Each visualization approach gives a different perspective to the quantum wave.

In discussion with my advisor, we were inspired by the hydrogen orbital cloud (Fig. 1.3). In this figure, there is a sense of looking into space, through the electron cloud. We decided to emulate that visual effect in the display of our analytic formula. That became my project - visualizing the

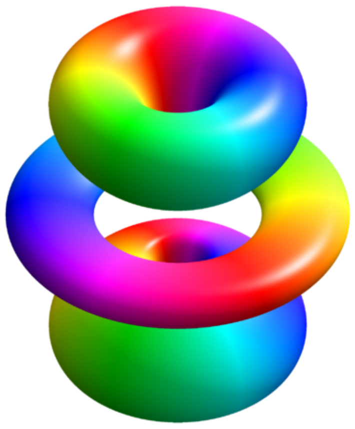


Figure 1.2 This is an example of probability densities displayed as contours. This particular image is a 45% contour plot of the $n = 4$, $l = 3$, $m = 1$ hydrogen orbital borrowed from Wikipedia [7]. The color indicates the phase of the wave packet and the shape indicates where there is a 45% or more chance of finding the electron.

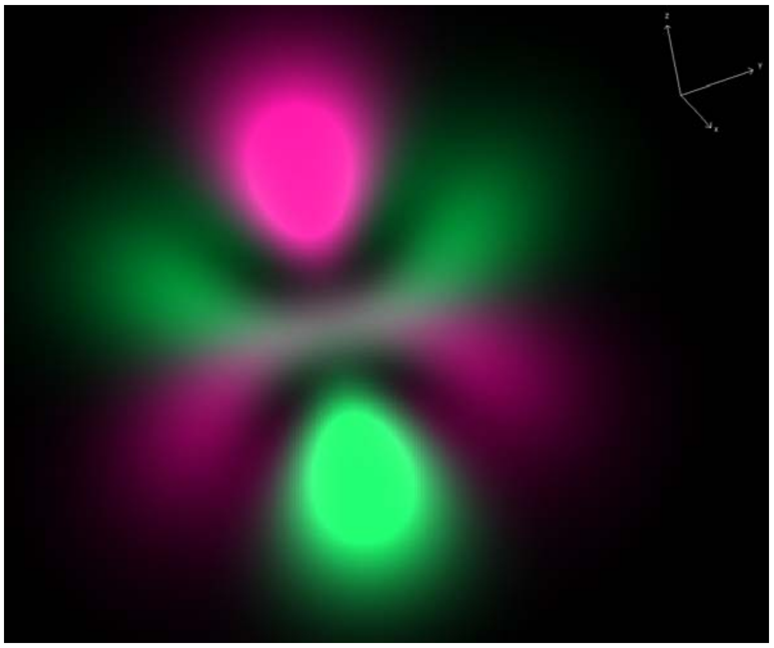


Figure 1.3 This is the $n = 4$, $l = 3$ hydrogen orbital. It is also an example of a probability density displayed as a cloud. Made by Paul Falstad [8].

electron wave packet as a 3D probability density cloud.

1.4 Overview

For my thesis project, I wrote a program to display a cloud-like electron wave packet. The program uses the formula discussed earlier to calculate the probability density in a region of space and creates an image of the probability density. Essentially, the program is a virtual wave packet camera.

In chapter 2, we discuss in detail our virtual camera model. We describe the components necessary to create the virtual pinhole camera and examine how the camera concept produces an image.

In chapter 3, we discuss the display of coordinate grid lines and their contribution to our virtual camera's 3D perspective. In particular, we show how the grid lines are calculated and displayed so they interact with the visualization of the wave packet.

Finally, in chapter 4 we use these visualization tools to explore dynamics of an electron wave packet in an intense laser field. The program generates a sequence of images which we combine to make movies. Chapter 4 has several images from animations produced by the program. These images are chosen to highlight the capabilities of the wave packet viewer. We can see the Lorentz drift, circular motion in the case of elliptical polarization, multiple peaks in the wave packet, and other features of electron wave packet motion in an intense laser.

Chapter 2

Visualization: Making a Virtual Camera

To view the wave packet as a cloud instead of merely as a 2D slice as in Fig. 1.1, we need a new framework to see the wave packet. The cloud comes from different amounts of probability density in different regions in space. To create the image of the cloud from different angles, we create a virtual camera that takes an image of the probability density at a certain instant in time. Fig. 2.1 shows the spatial relationship between the wave packet and our camera. In this chapter we explain our camera model.

2.1 Pinhole Camera: the Basics

We base our model on a pinhole camera. In the graphics world you won't find references to pinhole cameras; most authors refer to the process as a projection of some kind. Slater et al. discuss two kinds of projections: parallel and perspective projections [9]. The parallel projection is an orthographic projection. It defines a viewing window and a rectangular viewing area. Everything in the viewing area is projected along straight lines that are perpendicular to the viewing window. The orthographic projection gives no depth perception - it is like looking at everything from infinitely far away with an infinite zoom.

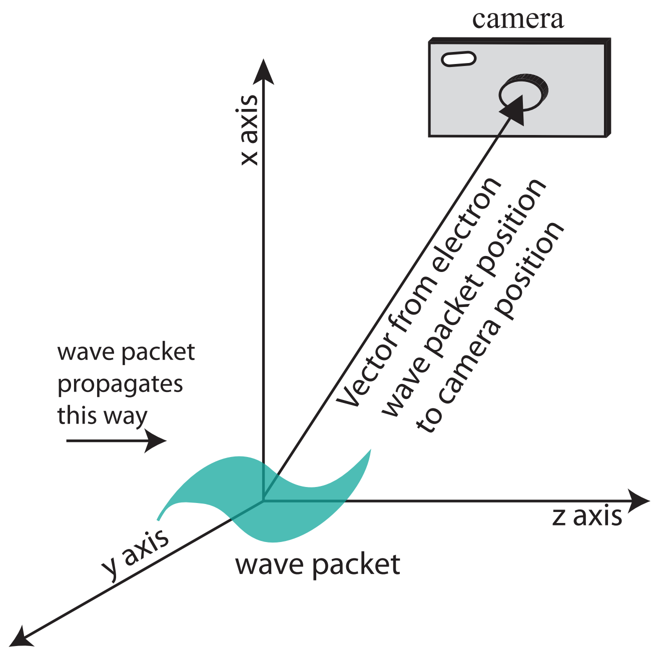

camera
x axis
Vector from electron wave packet position to camera position
wave packet propagates this way
z axis
y axis
wave packet

Figure 2.1 This is the basic setup for our camera. We want to be able to follow the wave packet as it moves along the z axis and we want to see the wave packet from any angle.

On the other hand, the perspective projection provides some sense of depth by altering the angular size of everything imaged, based on distance from the viewing window. It projects the image along lines that aren't parallel and aren't perpendicular to the viewing window. This makes closer objects appear larger and further objects appear smaller. This also results in some distortion of particularly close objects, as Newman discusses in his text [10]. It is hard to know how much distortion is present in the animations developed, because we don't see probability density distributions and don't have any reference for the images we produce.

Considering these two imaging models, we chose the perspective projection for the depth it creates. The model we use here applies the perspective projection process outlined in introductory texts on computer graphics [9], [10], [11], [12], but I refer to the imaging model as a pinhole camera in this thesis because I think the term is more illustrative than perspective projection. A pinhole camera is a simple physical example of a perspective projection. If you poke a small hole in a container and put some film in the container behind the hole, you have a pinhole camera and you get images made by perspective projection. In this chapter we discuss the application of a perspective projection, or a pinhole camera model, to the electron wave packet.

2.2 Pinhole Camera: the Virtual Parts

Here we outline the assembly of the virtual camera. The benefit of using a pinhole camera model is that the virtual camera parts are patterned after the real parts. The real camera needs a pinhole, film, and an object to image. Similarly, the virtual model needs a pinhole, film, and an object to image. To implement these ideas in our program, we introduce two coordinate frames. The coordinate systems provide a framework whereby the camera position relative to the wave packet is defined. This allows the program to calculate where the wave packet is and produce an image of the wave packet.

2.2.1 Coordinate Frames

The two coordinate frames we use are depicted in Fig. 2.2. One system, the unprimed system in Fig. 2.2, is defined by the initial position of the wave packet and the direction of the laser beam. The other system, the primed system in Fig. 2.2, is the coordinate system determined by the pinhole camera.

The formula in Section 1.2 defines the first coordinate frame. The electron wave packet is initially centered at the origin. The laser beam propagates in the z direction and is polarized in the xy plane. Since the wave packet is calculated using this formula, these coordinates define the space that our virtual camera views.

The second coordinate frame has the position of the virtual pinhole as its origin. The pinhole of a pinhole camera determines what information reaches the camera. In our virtual camera, the

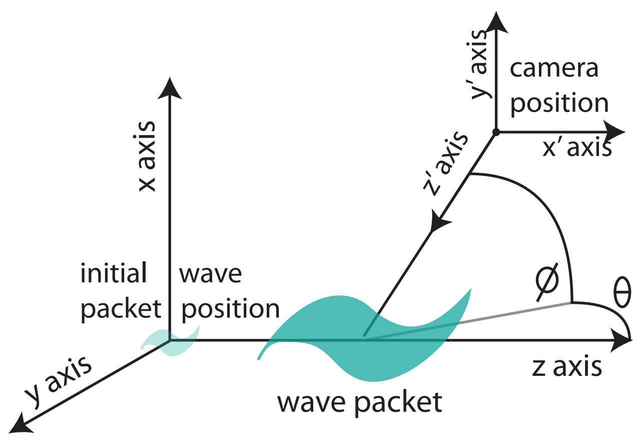


Figure 2.2 There are 2 coordinate frames used in this program. They are the unprimed and primed frames labeled here. The unprimed frame is the original frame, with the origin in the initial position of the wave packet. The primed frame is the camera coordinate frame.

virtual pinhole is a point in space, represented by a vector, that determines the position of the camera. Everything the camera sees is on a ray that goes through the position of the pinhole, so we use the pinhole position vector to define a new coordinate system. The position of the pinhole in relation to the camera is shown in Fig. 2.3.

2.2.2 Pinhole Vector

Since the pinhole determines what the camera sees, its position is determined by the position of the wave packet. To keep the wave packet in view, the camera has to do two things: first, it must move along the z axis with the wave packet, and second, it must be able to look at the wave packet from an arbitrary angle relative to the laser directions.

For the virtual camera to move with the electron wave packet, we need to know, at least approximately, how the wave packet moves. The laser beam is strong enough that while the electron oscillates in a direction parallel to the xy plane, it experiences a Lorentz drift in the z direction. Through experimentation with the camera model, we find that we can approximate the motion of the center of the wave packet with a constant speed in the z direction. In the program the constant

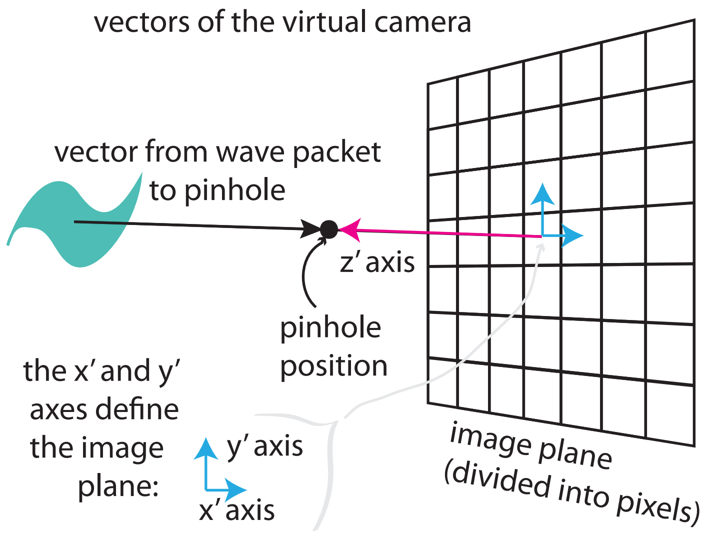

vectors of the virtual camera
vector from wave packet to pinhole
z′ axis
pinhole position
the x′ and y′ axes define the image plane:
y′ axis
x′ axis
image plane (divided into pixels)

Figure 2.3 The position of the camera is defined in terms of the position of the pinhole. The program makes a vector from the wave packet center to the pinhole. Continuing along that line a little further, the program defines the image plane. This plane is perpendicular to the pinhole vector and is divided into pixels.

v_e is used to denote the wave packet speed. At the parameters used for the first movie, v_e is 2. In a time t the wave packet has moved an approximate distance $v_e\, t$ along the z axis.

The position of the pinhole is a vector, which we will denote $\overrightarrow{r_p}$. To determine $\overrightarrow{r_p}$, we first define the vector $\overrightarrow{r_{mid}} = (0,0,v_e\, t)$ as the position of the wave packet center. Next we need a vector from the wave packet to the pinhole, which we will call $\overrightarrow{r_0}$. We set $\overrightarrow{r_0}$ equal to $\overrightarrow{r_{mid}}$. Next we make it a unit vector, $\overrightarrow{r_0} = \frac{1}{\|\overrightarrow{r_0}\|}\overrightarrow{r_0}$. Then we scale to a desired length, determined before the program is compiled. The length is typically set to 1, but can be varied.

The final step in the calculation of $\overrightarrow{r_0}$ is a rotation. The program rotates $\overrightarrow{r_0}$ angles set as parameters before the program is executed. Considering $\overrightarrow{r_0}$ as a vector from the origin, we first rotate it around the x axis by an angle θ. Then we rotate around the y axis by an angle ϕ. The rotation is computed as a matrix multiplication:

$$\overrightarrow{r_0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{pmatrix} \overrightarrow{r_0}. \tag{2.1}$$

The vector $\overrightarrow{r_p}$ giving the position of the pinhole is calculated from the wave packet position vector $\overrightarrow{r_{mid}}$ and from this vector $\overrightarrow{r_0}$ going from the wave packet to the pinhole by a simple addition: $\overrightarrow{r_p} = \overrightarrow{r_{mid}} + \overrightarrow{r_0}$.

We use the vector for the pinhole position as the origin of the camera coordinate frame. The z' direction of this frame goes from the pinhole to the camera, so it is represented using the opposite of the vector $\overrightarrow{r_0}$: $\hat{z}' = -\frac{1}{\|\overrightarrow{r_0}\|}\overrightarrow{r_0}$. The x' and y' directions are rotations of the x and y directions, given by rotating the vectors $\hat{x} = (1,0,0)$ and $\hat{y} = (0,1,0)$ by the angles θ and ϕ.. The camera coordinate frame, determined by the primed vectors, is depicted in Fig. 2.2.

2.2.3 Image Plane

Now that the coordinate frames are defined and the pinhole position is determined, we can introduce the last part of the virtual camera: the virtual film. The virtual film has a position in the coordinate system to help define the viewing rays and a corresponding place in memory to store the wave packet image. The imaging program defines an plane parallel to the $x'y'$ plane, a specified distance c from the pinhole position. This is the image plane.

To calculate and store information about the wave packet, we divide the image plane into pixels. The pixel positions are determined in terms of the x' and y' directions from the camera coordinate frame (Fig. 2.2): for an $n \times m$ pixel image, the (i, j) pixel is $i - \frac{m-1}{2}$ pixels away from the center of the image plane in the x direction, and $j - \frac{n-1}{2}$ pixels away in the y direction (this formula requires an odd number of pixels, so n and m must be odd). Each pixel has a corresponding spot in an array in the computer memory that stores part of the wave packet image. The position of the pixels in the plane and in memory preserve the images the camera takes.

The vectors that define the coordinate systems — the position of the wave packet, the position of the pinhole, and the two vectors that define the virtual film plane — give the framework we need to make a virtual camera. As the wave packet moves, the vectors change so that the camera tracks the electron wave packet.

2.3 Pinhole Camera: the Virtual Process

Now that the coordinate system for the virtual camera is outlined, we next turn our attention to computing the image that it displays. The purpose of the camera is to convert probability density information into some visual representation of the electron wave packet. This means that our camera program must calculate probability density in a volume of space, convert it to probability, and project the calculation onto the image plane. The volume is divided into different regions that are projected onto individual pixels of the image plane. This is accomplished by creating a ray that starts at a pixel position, goes through the pinhole position, and continues into space. This ray, and the subsequent imaging process, is diagrammed in Fig. 2.4.

The virtual camera looks along rays that start at regular points in a rectangle of the image plane and go through the pinhole. This creates a pyramidal volume through which the camera looks. We truncate the pyramid, so the actual volume through which the camera looks is a 3D relative of the trapezoid. This shape is called a viewing frustum in the computer graphics world [12]. We provide in Fig. 2.5 a sketch of a frustum.

The frustum is subdivided into a volume for each viewing ray. Each pixel will ultimately be associated with the probability of finding the electron in the volume of its corresponding ray. Since each ray has a rectangle of the image plane associated with it, each ray traces out its own frus-

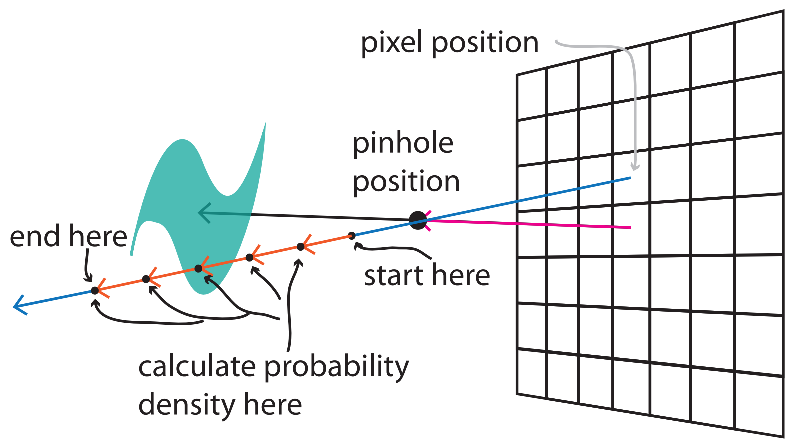


Figure 2.4 For each pixel, the camera constructs a ray through space from the pixel position to the pinhole. Along the ray, the camera calculates and sums probability densities and stores the value in an array.

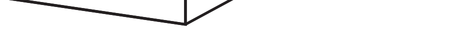


Figure 2.5 Our camera is defined to look through a certain volume of space determined from the position of the pinhole and the size of the image plane. This volume is pyramidal. To avoid excessive distortion, we truncate the viewing pyramid. The name for the truncated pyramid the camera looks through is frustum.

tum. The volume for each ray is further subdivided into volumes for each point on the ray where probability density is calculated using the formula explained in section 1.2. To convert probability density values into probabilities, the program must determine the volume of each frustum subsection.

Since a frustum is a section of a square pyramid, we can use the volume of a pyramid to determine the volume of a frustum section. The volume of a pyramid is $1/3Bh$ where B is the base of the pyramid and h is the height of the pyramid, the distance from the vertex to the base. We can determine the volume of a frustum section by subtracting the pyramid volume of the larger base from the volume corresponding to the smaller base.

The probability of finding an electron in a given volume is the probability density times the volume. Hence at each step along a viewing ray, the program evaluates the probability density and multiplies by the volume of the frustum section at that point, as determined above. The program steps along the ray, summing the evaluated probabilities along the ray to get the probability that the electron appears in the frustum volume of the ray. At a specified distance from the pinhole, the program stops summing probability densities and stores the sum in the memory array position corresponding to that ray. The values of the start and stop distances from the pinhole were adjusted to ensure that most of the wave packet is calculated.

Proceeding in this manner through every pixel, the camera ends with an array of probability values in memory. The brightness of the pixel is proportional to the value of the probability sum stored in the pixel. The larger the value assigned to a pixel, the brighter it appears. This information is used to make a "picture" of the probability density distribution. The picture is made by creating an $m \times n \times 3$ array of color vectors from the probability array. A color vector is a vector whose components lie in the interval $[0,1]$. The components of the color vector correspond to the brightness of maroon, cyan, and yellow. An $n \times m \times 3$ array of color vectors tells the screen to make an image of $n \times m$ pixels, each showing the combination of maroon, cyan, and yellow specified by

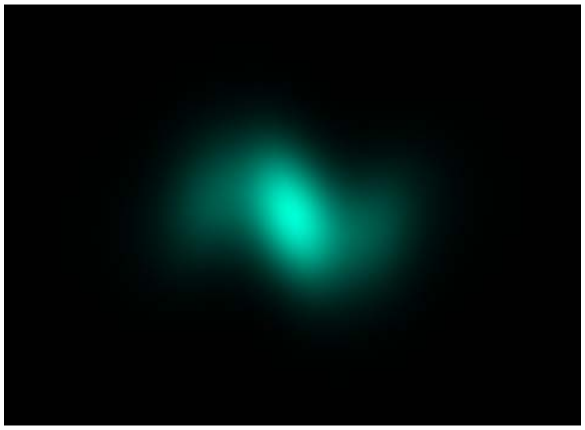


Figure 2.6 This is an image of the electron wave packet generated by the virtual camera. The brightest blue pixels correspond to the areas with the largest probability of finding the electron. The important parameters for this image are: $t = 45$, $\theta = \pi/2$, $\phi = \pi/40$, $Ao = 1$, $a = 2.1 \times 10^6$, $p_w = 1 \times 10^{-6}$.

the corresponding color vectors.

For our movies, each frame starts with an $m \times n \times 3$ array where each pixel is the same turquoise color vector $(1, .8, 0)$. The i, j color vector is calculated by multiplying the base color vector $(1, .8, 0)$ by the probability value in the i, j position of the probability array. Thus, array positions with the largest probability have the brightest turquoise, while the places with least probability appear dark. One such image is given in Fig. 2.6.

The process of generating a single image is repeated at every desired point of time and each picture stored as one frame of a movie. We then combine the frames to animate the wave packet.

Chapter 3

Coordinate Grids - Generating Perspective

3.1 Need for Perspective

The virtual camera we created makes a good image of an electron cloud moving in a laser field, but more can be done to give it a 3D perspective. Various methods for enhancing perspective include: juxtaposing a schematic showing where the camera is in relation to the wave packet with the image produced by the camera; shading parts of the wave packet differently with distance from the camera; or adding coordinate grids to the camera-generated picture. Of these, the coordinate grid lines proved most effective.

In this chapter we present our method for calculating and displaying the coordinate grid lines. The wave packet is described in a cartesian coordinate system, so we draw lines parallel to the coordinate axes at regularly spaced intervals. For example, the line $(1,1,t)$ goes parallel to the z axis, one wavelength in the $+x$ direction and one wavelength in the $+y$ direction from the origin. If the camera is looking through a region of space intersecting the line $(1,1,t)$, then we wish to see this line in our image of the wave packet. To this effect, we discuss the geometry of the grid lines, how to project these lines onto the image plane, and how we shade the lines to enhance the

3D perspective the lines offer.

3.2 Generating Perspective - Projective Geometry

Our approach to displaying the coordinate grid lines uses the geometry of lines to project them onto the image plane. A line is defined by a point $\vec{p}_0$ on the line and a vector $\vec{d}$ describing the direction of the line: $\vec{r}(t) = \vec{p}_0 + t\vec{d}$. For each value of t, $\vec{r}(t)$ is a point on the line. We can use the vectors $\vec{p}_0$ and $\vec{d}$ of any line to project the line onto a plane. The coordinate grid lines have an especially simple description. Each coordinate line runs parallel to one of the coordinate planes, so x lines will have the form $\vec{r}(t) = (t, y_0, z_0)$, y lines have the form $\vec{r}(t) = (x_0, t, z_0)$, and z lines have the form $\vec{r}(t) = (x_0, y_0, t)$. In this section we explain how the program projects the coordinate grid lines onto the image plane.

The program images the coordinate grid lines that intersect the viewing frustum. Since the frustum has a finite volume, only finitely many lines intersect the frustum. The process for imaging the coordinate grid lines involves several steps, outlined in the following subsections. First, the virtual camera determines which coordinate grid lines intersect the viewing frustum, as discussed in subsection 3.2.1, and then determines the coordinates of the intersection points, as discussed in subsection 3.2.2. Since each line that goes through the viewing space intersects two faces of the frustum, each line intersecting the frustum becomes a pair of points. As explained in subsection 3.2.3, to draw the coordinate grid lines, the program must take each pair of intersection points, project them onto the image plane, and draw a line between them.

3.2.1 Which lines to draw? Defining the problem

First, the program must specifically define the volume the camera is looking through to decide which lines to display. Recall from section 2.3 that the camera looks through a truncated rect-

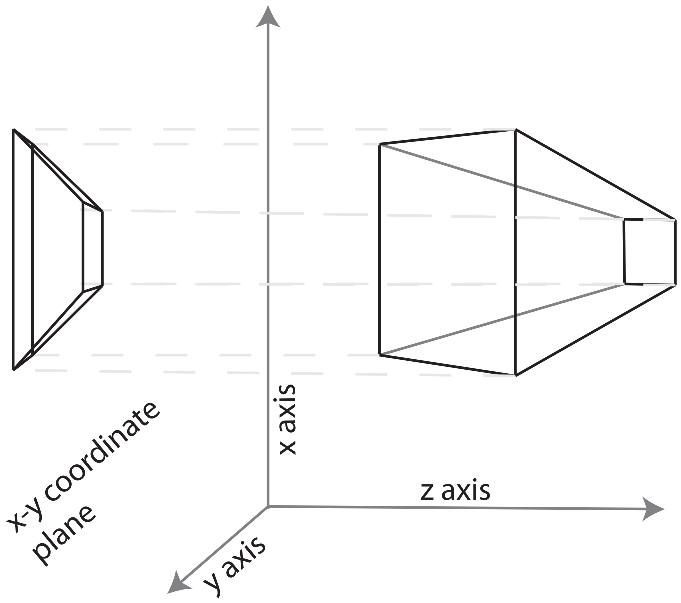


Figure 3.1 In order to determine which coordinate grid lines pass through the frustum, we project the frustum onto each of the coordinate planes.

angular pyramid, with its apex at the position of the pinhole. This truncated pyramid is called a viewing frustum. If we examine the frustum in Fig. 2.5, we see that it has six faces, eight vertices, and twelve edges. Using these faces, vertices, and edges, we can define the frustum and determine which coordinate lines needs to be imaged.

The program checks each face of the frustum for coordinate lines that pass through the face. The faces are checked one at a time by projection onto each of the xy, yz, and xz coordinate planes, as shown in Fig. 3.1. To project a face, we actually project the four vertices and the four edges of the face onto the coordinate plane. The four edges make a quadrilateral on the coordinate plane. For a coordinate line to pass through the frustum, it intersects at exactly one point on each of two faces of the frustum. To understand what this means, suppose the coordinate line (x_0, y_0, t) intersects the frustum. When we project this line onto the xy plane, we get the point (x_0, y_0). Since the line (x_0, y_0, t) intersects the frustum, there are two faces of the frustum whose projections onto the xy planes (which are quadrilaterals) contain the point (x_0, y_0).

To find the points corresponding to grid lines that are inside a given quadrilateral, the program first draws a rectangle through the vertices of the quadrilateral. This rectangle contains the quadrilateral and makes it easy for the computer to step through (x_0, y_0) points representing possible grid line intersection points. To determine whether a point lies inside a quadrilateral, the program represents the quadrilateral as a set of four lines. If each equation is written as an inequality, then a point is either on or below the line, or is above than the line. The intersection of the four inequalities defines the interior of the quadrilateral. That is, a point has to satisfy all four of the inequalities for a quadrilateral to be inside the quadrilateral.

3.2.2 Which lines to draw? Calculation

Now we examine how the program constructs the inequalities that determine whether a grid line intersects the viewing frustum, and determines the coordinates of the points of intersection.

A line can be written as $(y-y_1)=m(x-x_1)$ where x and y are the x and y coordinates of any point on the line, x_1 and y_1 are the coordinates of one particular point on the line, and m is the slope of the line. In our case, we only have two points on each line. We use these points to calculate m; the slope is $\frac{\text{rise}}{\text{run}}$ or $\frac{\Delta y}{\Delta x}$. For the first and second points, $m=(y_2-y_1)/(x_2-x_1)$. The line equation is, after substituting for m,

$$(y-y_1)=\frac{y_2-y_1}{x_2-x_1}(x-x_2).$$

If we multiply by the denominator, we arrive at $(x_2-x_1)(y-y_1)=(y_2-y_1)(x-x_2)$. The corresponding inequality is either

$$(y_2-y_1)(x-x_2)\geq(x_2-x_1)(y-y_1)$$

or

$$(y_2-y_1)(x-x_2)\leq(x_2-x_1)(y-y_1).$$

The first inequality describes all points above the line and the second gives all points below the line.

The program makes a set of four inequalities corresponding to the four edges of the quadrilateral. Let the four vertices of the quadrilateral be denoted $(x_1,y_1),(x_2,y_2),(x_3,y_3)$, and (x_4,y_4). We assume here that we've already ordered the points so that, when connected in a loop, they describe a quadrilateral. Here is a sample set of four inequalities describing the inside of the quadrilateral, for a projection onto the *xy* plane:

$$\begin{aligned}
(y_2-y_1)(x_0-x_1) &\geq (y_0-y_1)(x_2-x_1)\\
(y_3-y_2)(x_0-x_2) &\geq (y_0-y_2)(x_3-x_2)\\
(y_4-y_3)(x_0-x_3) &\geq (y_0-y_3)(x_4-x_3)\\
(y_1-y_4)(x_0-x_4) &\geq (y_0-y_4)(x_1-x_4)
\end{aligned} \tag{3.1}$$

The x_i and y_i, where $i = \{1,2,3,4\}$, are the x and y coordinates of the four vertices of the face of the frustum. The values x_0 and y_0 are the coordinates of a possible coordinate grid line. The inequalities represent the four edges of the projected face, the four sides of a quadrilateral. If the point (x_0,y_0) satisfies all four inequalities for a given face of the frustum, then we know that a coordinate grid line intersects that face and passes through the volume of interest. Figure 3.2 is a diagram of the process.

After the program has found a coordinate grid line that passes through the volume of interest, it knows that the line intersects the surface of the frustum at two points. The program has two of the three coordinates for each point of intersection - they are the two parts of the (x_0,y_0) point found in the preceding paragraph. We recall that the point (x_0,y_0) was found by projecting one of the faces of the frustum onto a coordinate plane. The face of the frustum is itself a plane. The equation of this plane is:

$$\begin{gathered}
n_x(x-x_1)+n_y(y-y_1)+n_z(z-z_1)=0,\\
\vec{n}=(n_x,n_y,n_z),
\end{gathered} \tag{3.2}$$

where $\vec{n}$ is the normal vector of the plane and (x_1,y_1,z_1) is a known point on the plane.

Using the plane equation corresponding to the particular face of the frustum, and plugging in

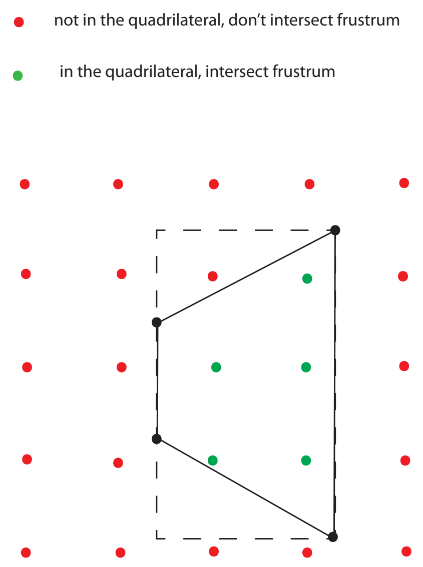

not in the quadrilateral, don't intersect frustrum
in the quadrilateral, intersect frustrum

Figure 3.2 Calculating lines through the frustum. After the edges of the frustum are projected onto a coordinate plane, the program tests evenly spaced points in the plane to see if they are inside any of the projected faces of the frustum.

the two coordinates already determined, the value of the third coordinate can be determined:

$$x_0 = x_1 - \frac{n_y}{n_x}(y_0 - y_1) - \frac{n_z}{n_x}(z_0 - z_1). \tag{3.3}$$

The program saves the ordered triplet (x_0, y_0, z_0) as the position of one of the endpoints.

The distances in the analytic formula developed in section 1.2 are in units of laser wavelengths. We chose to have each pair of adjacent grid lines separated by one laser wavelength. This means the program looks for all whole number points inside the projection of a face, i.e., all points such as (1,1), (1,2), and so on that lie inside the quadrilateral corresponding to a face. The program repeats the process of the last three paragraphs for the other faces of the frustum to determine the position of the other endpoint. Each grid line becomes a pair of points in space.

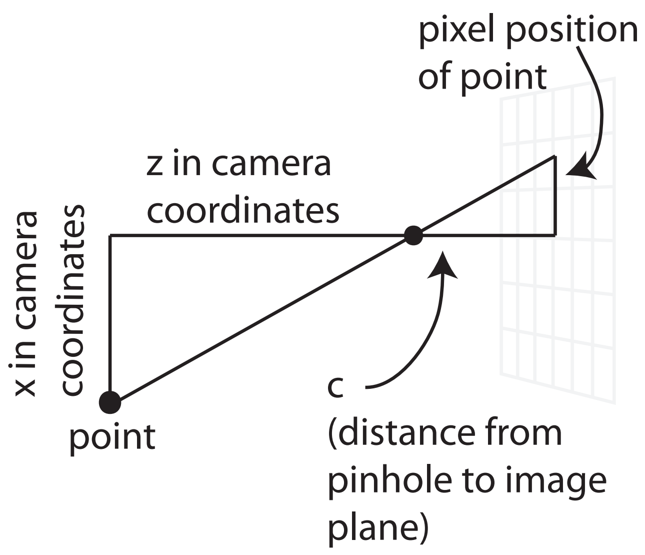


Figure 3.3 In order to draw the coordinate grid lines, we need to draw their endpoints. To draw the endpoints, we need to convert their positions in the camera coordinate frame to pixel positions. We do that using the similar triangles the positions create along the z' axis.

3.2.3 Drawing the lines

In order to draw the endpoints on the image, we need to determine their coordinates in the coordinate system of the image plane. As calculated in the process described above, the coordinates of the endpoints are expressed in the initial coordinate frame. In section 2.2 we discussed two coordinate frames: the initial frame and the camera frame. The program uses the transformations between the frames as expressed in section 2.2 (specifically, in (Eq. 2.1)) to get the coordinates of the intersection points in the camera reference frame.

Lastly, these 3D coordinates need to be converted to 2D pixel positions. The x pixel position is related to the x' coordinate value by the similar triangles shown in Fig. 3.3. The two positions are related by the ratio equation $\frac{x}{z} = \frac{\text{pixel position}}{c}$. We can find the pixel by rounding the value cx/z to the nearest integer value. Then we add $(m-1)/2$, and we have a pixel position the computer can store as the position of the endpoints of the coordinate grid line. We go through a similar process to calculate the y pixel position of the endpoints.

With the pixel coordinates of the grid line endpoints determined, the program can draw the grid line on the image. An array in memory is set aside for the image of the grid lines. Each point in the array is initially assigned zero. Whichever pixels should have the line are changed from zero to one. We use a simple line-drawing algorithm to connect the grid line endpoints in the image. Line-drawing algorithms are well-developed, and there are descriptions and motivations for the various algorithms in almost any graphics text books [11], [12]. The specific form of the line-drawing algorithm we used is an optimized form of Bresenham's algorithm that we copied from Wikipedia [13] (see appendix for code). Our line drawing program assigns 1 to every pixel on the line segment between grid line endpoints.

Once the array for the grid lines is generated, it is converted to an array of color vectors. The process is similar to that for the wave packet image, explained at the end of section 2.3. We make our grid lines orange-yellow, so we use a color vector $(0,1,.833)$. To make the $m \times n \times 3$ color vector array, we multiply the base vector $(0,1,.833)$ by the value in the i, j position of the array with the grid line information. Then the i, j pixel is either $(0,0,0)$ or $(0,1,.833)$, depending on whether a line goes through the pixel.

3.3 Shading the Coordinate Grid

We also shade the coordinate grid lines to enhance the spatial perspective. The lines are shaded with distance from the camera, to indicate depth, and as they pass through and behind the probability cloud.

When we say shade, we mean that we alter the array with the grid line information. The array stores ones wherever the lines are and zeros where there are no lines. If we make the array values with the lines slightly less than one, then the grid line appears a little dimmer in the image.

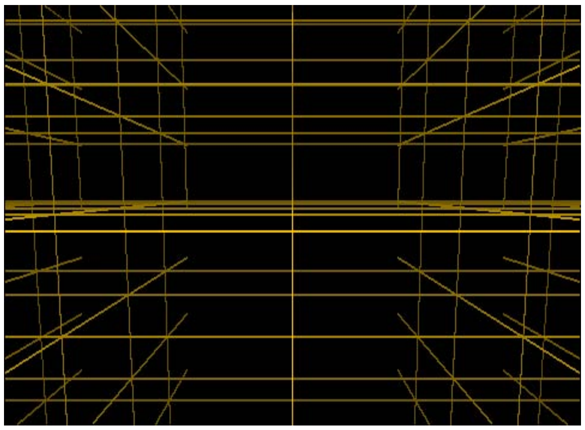


Figure 3.4 This is an image of the coordinate grid lines as seen by the virtual camera. The important parameters for this image are: $t = 45$, $\theta = \pi/2$, $\phi = \pi/40$.

First, we shade the lines with distance from the pinhole. When the program draws a line, it keeps track of spatial coordinates of end points. The values at the endpoints are decreased slightly from one, with greater decrease as distance from the pinhole increases. Shading varies linearly from one endpoint to the other.

Next we shade the alter the appearance of the grid lines and the wave packet as the grid lines pass through the points in space where the wave packet has the highest probability density values. The process we use to display the grid lines doesn't actually track the grid lines, so it doesn't actually know when the lines 'pass through' the wave packet. Rather, it makes an educated guess as to when the lines should appear to go through the wave packet. For each i, j pixel in image array, we compare grid line array and wave packet image array. The relation between the values in the grid line array at a certain pixel and the values in the wave packet image array at the same pixel determine how the color vector at that pixel is calculated.

The program compares the values in the wave packet and grid line arrays to a set of parameters in a series of if/then statements. Depending on the values at each pixel, the pixel is displayed in one of the following ways: it displays the wave packet information if there is no substantial grid or the wave should be well in front of the grid; it displays the grid if there is no substantial wave

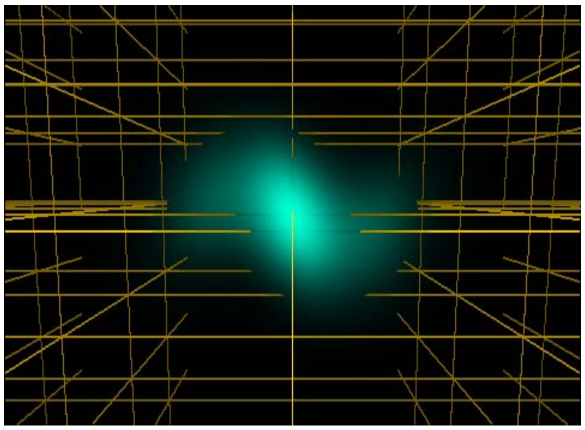


Figure 3.5 This is an image of the electron wave packet generated by the virtual camera, with the coordinate grid lines displayed to enhance the 3D perspective. The important parameters for this image are: $t = 45$, $\theta = \pi/2$, $\phi = \pi/40$, $Ao = 1$, $a = 2.1 \times 10^6$, $p_w = 1 \times 10^{-6}$.

or if the grid should be well in front of the wave; it displays the wave packet, dimmed slightly to indicate the presence of a grid line; or it displays the grid, dimmed slightly to indicate the grid going through the wave packet.

Chapter 4

Results

In this chapter, we explore wave-packet animations produced using the 3D tools developed in this thesis. We present frame sequences from two movies and discuss the notable features revealed by the camera model. The first movie shows an electron wave packet in linearly polarized light, meaning a laser beam whose transverse field oscillates only in one direction. The laser field is expressed in the formula (Eq. 1.7) by the vector potential $\vec{A}(t) = A_0 \cos \omega t \, \hat{x}$. This gives a field that oscillates sinusoidally in the x direction only.

The frames from the second movie show an electron wave packet in circularly polarized light, meaning a laser beam whose transverse field oscillates in a circle. To achieve this, we let $\vec{A}(t) = A_0(\cos \omega t \, \hat{x} + \sin \omega t \, \hat{y})$.

The calculations are all done in units of laser wavelengths and laser periods. The frames are labeled by time. This number indicates how many laser periods have occurred since the beginning of the animation. The distance from one line to the next is one laser wavelength. The wave packet develops interesting structure as it spreads to the size of a wavelength.

4.1 Linearly Polarized Light

The frames in this section illustrate the motion of an electron wave packet in a linearly polarized laser field. The frames in Fig. 4.1 are chosen to show the growth of the wave packet from $t = 10$ laser periods to $t = 50$ laser periods, the usual length of movies we've made so far.

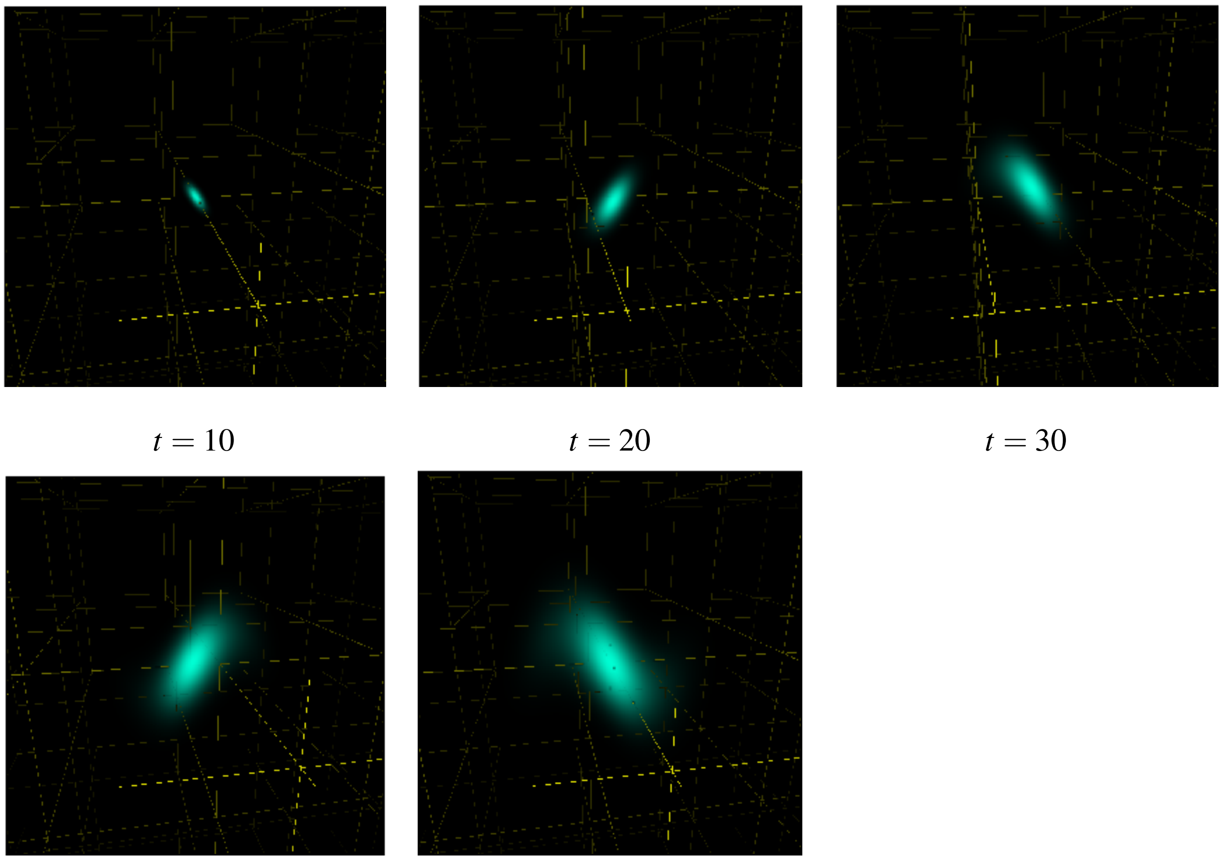
In these frames, the parameters are chosen so that the wavelength of the laser is 800 nm and the intensity is 2.1×10^{18} W/cm^2. At this intensity, the motion of the electron is relativistic. The B field is strong enough to cause a drift along the z axis. The initial size of the wave packet is determined by the dimensionless parameter p'_w, which relates to the initial momentum spread. For this movie, p'_w is set to .005.

The movie covers enough electron time to see the diffusion of the wave packet. In the beginning, the wave packet is localized to one region of space. While the wave packet is a distribution of probability density spread through all space, the probability density is very low (essentially zero) in most of regions. The wave packet is only drawn where it is significant, and the movies indicate this significant region is quite small at first. The probability distribution then diffuses over time. The volume where there is a significant chance of finding the electron increases, and the size of the wave packet grows significantly over the course of these first movie frames.

The next movie frames, in Figs. 4.2-4.3, are chosen to show the wave packet over one period of the laser, from $t = 44.4$ to $t = 45.4$.

Over one laser period, the wave packet goes through one complete oscillation in the x direction. This is a result of the force the laser exerts on the wave packet. The electric field oscillates in the x direction, which we is vertical in the image we see. This field creates a force on the electron that moves it vertically. As the wave packet reaches the size of a laser wavelength, different parts of the wave packet are being accelerated in different directions. The different accelerations of different parts of the wave packet result in a multi-peaked wave packet structure.

In the movie, the electron wave packet moves to the right. This motion, called z drift, is a result



$t = 10$ $t = 20$ $t = 30$

$t = 40$ $t = 50$

Figure 4.1 Electron wave packet in linearly polarized light over our typical movie duration

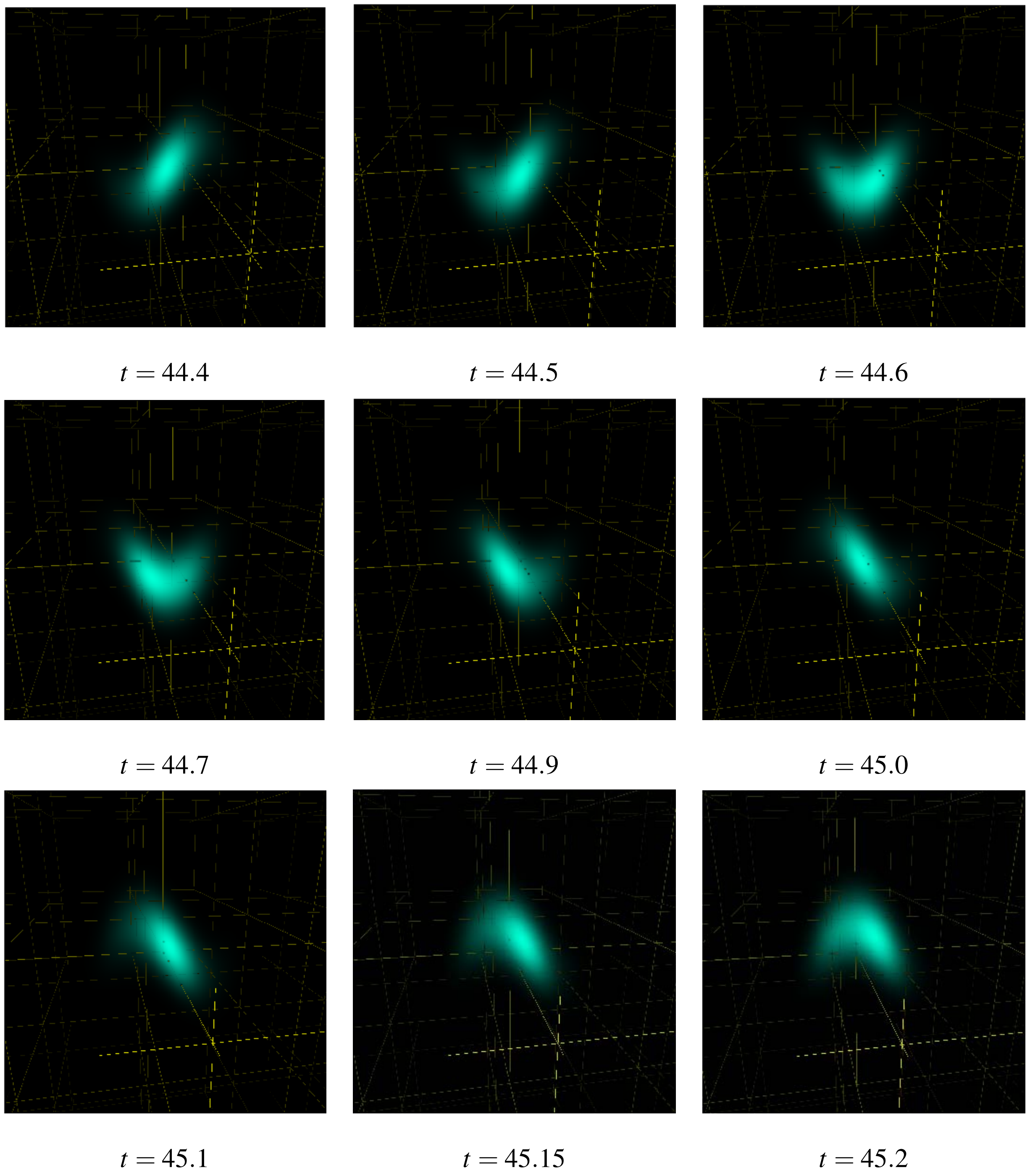


Figure 4.2 Electron wave packet in linearly polarized light over one laser period

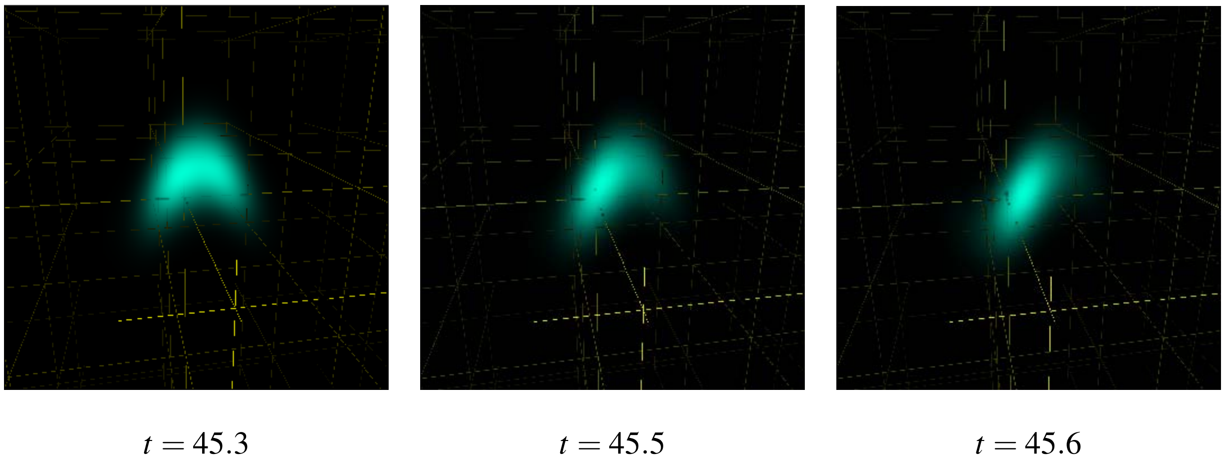


Figure 4.3 Electron wave packet in linearly polarized light over one laser period (continued)

of the Lorentz force from the magnetic field. As the electron moves up and down, it is accelerated to high velocities. The magnetic field is strong enough that the electron feels a significant magnetic Lorentz force along the z direction. Over the one period depicted in these frames, the electron has drifted slightly to the right by about a fourth of a grid, or a quarter wavelength.

4.2 Circularly Polarized Light

Next we look at an electron in a circularly polarized laser. The laser that drives the wave packet in Fig. 4.4 has an electric field that oscillates in a circle, not up and down.

As in the previous section, for these frames p'_w is set to .005 and the parameters are chosen so that the wavelength of the laser is 800 nm. The intensity doubles for similar parameters compared to the linearly polarized case, because there is now a field in the y direction as well. The intensity the electron experiences in these movies is 4.2×10^{18} W/cm^2. The motion of the electron is still relativistic and we see a drift along the z axis.

The circular field pushes the electron in a circular path. We are looking at enough frames to see

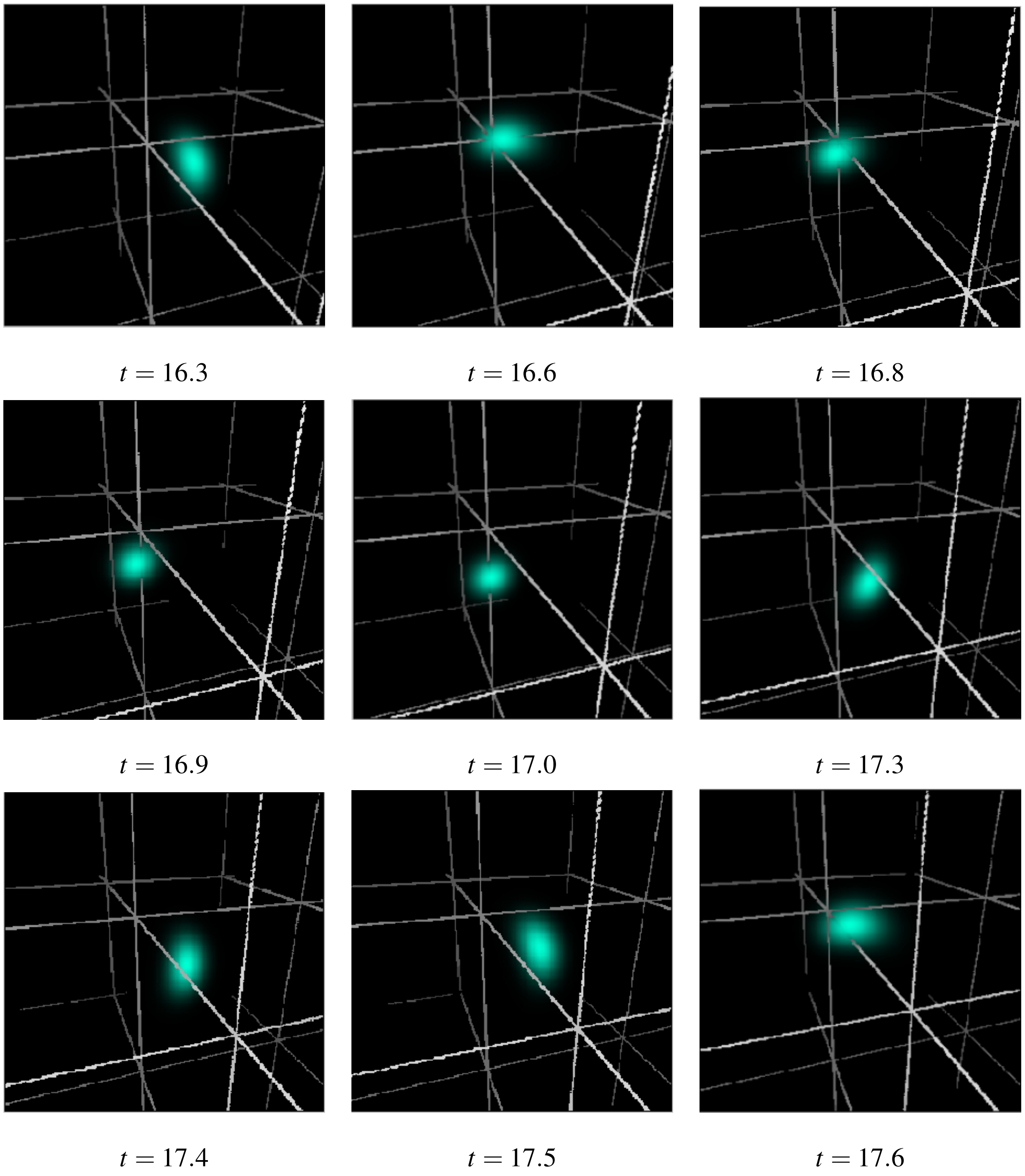


Figure 4.4 Electron wave packet in circularly polarized laser over one laser period

one circular path made by the electron, from $t = 16.3$ laser periods to $t = 17.6$ periods. Note that the electron wave packet moves with the same frequency as the laser, moving through one circle in about one laser period. The camera is positioned just above and to the left of the z axis as the laser propagates towards it, so the Lorenz force moves the wave packet toward the viewer as the wave packet rotates around the z axis.

These frames highlight another advantage to the coordinate grid lines. In addition to showing the motion of the wave packet in the direction of laser propagation, the grid lines give a reference point for understanding the circular motion. The wave packet moves around the z axis, and seeing the wave packet relative to this line helps us understand the motion. The camera model and the grid lines complement each other in improving our appreciation for the wave packet's motion.

Appendix A

Matlab Code

```
% wavePacketViewer
% Ryan Sandberg
% Summer 2012
% -------------------------------------
% This is the latest version of our electron wave packet viewer.

% Documentation:
%
% Functions:
% Rotate
% gridDraw
%   lineDraw
% calculateWave
%   Evaluate
% SeeCam
% firstMask
%--------------------------------------
clear; close all;
format long;

%!!!!!!!!!!!!!!!!!!!!Essential Movie Parameters!!!!!!!!!!!!!!!!!!!!!!!!!!!!
% Use these to craft your movie

% Don't forget to set 'nframe' to 600 or whatever is appropriate for your movie
% and uncomment the commands to save the frames as tifs (found at the end
% of the program) when you
% have the parameters you want

%Movie title, will appear in frame titles, followed by 'frame'(framenumber)
% i.e. if I set moviename to 'slimer', then the first frame will be titled
% 'slimerframe1'
% NOTE:  Don't use numbers in moviename!

moviename='paralleltest';

Ao=1; % Scaled amplitude of laser. At Ao=1, and with a=2.1e6, we have an intensity of 2.1e18 W/cm^2
a=2.1e6; % a=mc * lambda/hbar = 2.07e6 for 800 nm light
pw=5*10^-3; % scaled initial momentum spread

% Polarization of light
% Change in the function Evaluate
% There are three options already prepared: linear in x direction, linear
% in y=x direction, and circular polarization in xy plane

% Electron speeds
% ve=2;  % for Ao=1 and linearly polarized light in x direction only
ve=3.25;  % for Ao=1 and either circularly polarized light or light polarized in y=x direction

%Start and stop times
tmin=5;
tmax=50;
%number of frames
nframe=5; % 600 frames is preferred for a movie, 5 is good to check your
```

```
%           parameters and make sure you're getting the movie you want
dt=(tmax-tmin)/nframe; % Don't change this

% Number of pixels in final image  !!MUST BE ODD!! (unless you change code
%                                    everywhere)
% Image is an array, so there are mfinal rows by nfinal columns
nfinal=1281;
mfinal=1025;

%****************Viewer angle and zoom controls*************************
% viewer is set up in non-traditional cartesian coordinates.
% z is the direction of the B field; electron moves along z axis.
% z is a lateral direction, as is y
% x is the "vertical direction"

% theta is the angle of r0 around the x axis. (the azimuthal angle)
% phi is the angle of r0 up from the zy plane. (the latitudinal angle)

%-------------Setting the angles and zoom-----------------------------
% Each of these parameters can be set to a constant value,
% a range of values that increases linearly with time, or given a more
% complicated functional dependence.

% The constant values are set here.

% Start/stop values can be set here but things that change in time need to
% be evaluated in the loop.  This can be done in the part of code labeled
% ******More advanced angle/zoom control****  in the working
% section of the program

% thetaw=pi/2;
% thetaw=.125664;

theta0=1*pi/5;
thetaf=-1*pi/5;
dtheta=(thetaf-theta0)/50;

%--------angles for single instant tests-----------
% theta0=0;
% thetaf=4*2*pi;
% dtheta=(thetaf-theta0)/nframe;
% phi0=-pi/2+.01;
% phif=pi/2;
% dphi=(phif-phi0)/nframe;
%-------------------------------------------------

% phiw=pi/40;


% phi0=1*pi/3;
% phif=1*pi/3;
% dphi=(phif-phi0)/50;

% aspin is the rotation angle of the camera about the camera viewing axis
% (the vector r0).  It's better to leave aspin alone.
aspin=0;

% One way to change the zoom is to change how close the camera is to the
% wave packet.  nlr0 is a multiplier that scales this camera-wavepacket
% distance
nlr0=1;

%-------------------------------------------------------------------------
%--------------------Development of camera constants-----------------------
% You could change these, but I don't recommend it unless you know what
% you're doing.  You have already set the important parameters in the
% section above

% camera constants
rmid=[0,0,5]; % center of box (rmid means electron position later)
r0 = [0,0,2]; % initial vector from rmid to pinhole
c=1.5; % distance from pinhole to image plane

camheight=1.51;% set size of camera

aratio=nfinal/mfinal; % ratio of camera width to height

camwidth=aratio*camheight;

m=201; % number of rows of pixels !!must be odd!!
nposs=floor(m*aratio);
if( mod(nposs,2)==0)
   n=nposs+1;
else
   n=nposs;
end
% n=m; % number of columns of pixels !!must be odd!!
piw=camwidth/m;% pixel width
pih=camheight/n;% pixel height

% setting up the initial pixel vectors
camx=[piw,0,0];
camy=[0,pih,0];

criticalVal=8*10^-2; % parameter for showing wave packet
criticalVal3=1.3*10^-1; % parameter for distant gridline, dim wavepacket
criticalVal2=.4; % parameter for showing wave packet v. grid coloring

gridicalVal= .65;  % parameter for deciding if grid should appear behind wave packet
gridicalVal2=.95;  % parameter for deciding if grid appears in front of wave packet

gridcolor=[1,.8,0]; % set the colors of the wavepacket and the grid
wavecolor=[0,1,.833];

%Lateral offset of camera focus from expected center of wave packet
xo=0;
yo=0;
%------------------------------------------------------------------
```

```
%------------------Test array---------------------------------------
sums=zeros(nframe+1,1);
%-------------------------------------------------------------------
%----------Program starts working here!------------------------------

for s=1:nframe+1; % Loops through each frame of the movie
   tic
   t=tmin+(s-1)*dt;
%    t=8.08; % for single instant test

%*****************More advanced angle/zoom control*******************
   thetaw=theta0+t*dtheta;

%    phiw=phi0+t*dphi;
%    Some examples of functional phi dependence:
   phiw=pi/5*cos(2*pi*t/50-3/8)-pi/24;
%    phiw=pi/3-2*pi/3/50*t;

%    aspin=2*pi/50*t;

   nlr0=(exp(-t/2)+1/4*log(t/2+2));
%*************************************************************
%***************Angle controls for testing a single instant*****
% thetaw=theta0+(s-1)*dtheta;
% phiw=phi0+(s-1)*dphi;
%*************************************************************

  adjr0=r0*nlr0;

  epos=1*ve*t/50*rmid; % approximate position of electron wave center
  nrmid=[xo,yo,0]+epos;

  nr0=Rotate(0,thetaw,phiw,adjr0);
  ncamx = Rotate(aspin,thetaw,phiw,camx);
  ncamy = Rotate(aspin,thetaw,phiw,camy);


  dx=19/(m-1);  % Why 19?  It's completely arbitrary right now.
             % It is an artifact of earlier work.
  dy=19/(n-1);
  [Xi,Yi]=ndgrid(1:dx:20,1:dy:20);

  % grid, waveintensity
  % calculate wave packet
  [waveintensity]=calculateWave(Ao,a,pw,m,n,nr0,nrmid,ncamx,ncamy,c,t);

 %final grid calculation, finalintensity interpolation

  finalgrid=gridDraw(mfinal,nfinal,camheight,camwidth,c,ncamx*m/mfinal,ncamy*n/nfinal,nr0,nrmid);

  dxi=19/(mfinal-1);  %Deciding interpolation of final image
  dyi=19/(nfinal-1);
  [Xii,Yii]=ndgrid(1:dxi:20,1:dyi:20);
  finalintensity=interp2(Xi',Yi',waveintensity',Xii',Yii')';

  finalimage=zeros(mfinal,nfinal,3);

  % This loop makes the color image 'finalimage' from the calculated projection of the
  % wave packet in the array 'finalintensity'
  for k=1:mfinal
     for l=1:nfinal

        if finalgrid(k,l)>1
           finalgrid(k,l)=1;
        end

           if finalgrid(k,l)>0 % wherever grid is

              if finalintensity(k,l)>criticalVal % where wave is bright enough to see

                 if finalgrid(k,l)<gridicalVal  % if grid is well behind wave packet,
                    if finalintensity(k,l)<criticalVal3 % see grid faintly
                        finalimage(k,l,:)=finalgrid(k,l)*(.95-finalintensity(k,l))*gridcolor;
                    else
                       finalimage(k,l,:)=finalintensity(k,l)*wavecolor; % don't see grid
                    end

                 else
                    if finalgrid(k,l)<gridicalVal2  % if grid goes through wave packet
                       if finalintensity(k,l)>criticalVal2 % where wave is very bright/concentrated
                          finalimage(k,l,:)=.20833*(finalintensity(k,l)+3.8)*finalintensity(k,l)*wavecolor;
                       else  % grid goes through edges of wave packet
                           finalimage(k,l,:)=(1-5/7*finalintensity(k,l))*(finalgrid(k,l)-.05)*gridcolor;
                       end
                    else   % grid is well in front of wave packet
                       finalimage(k,l,:)=finalgrid(k,l)*gridcolor;
                    end

                  end
              else % where wave packet isn't
                  finalimage(k,l,:)=finalgrid(k,l)*gridcolor;
               end
           else % where grid isn't
              finalimage(k,l,:)=finalintensity(k,l)*wavecolor;
           end

     end
  end

%-----------------------------------------------------------------------

  figure% change the figure number if you want a figure window for every frame

%  These 6 lines let you see the camera position
%   subplot(1,2,1,'position',[.05,.1,.37,.37]);
%   SeeCam(ncamx,ncamy,nr0,nrmid,m,n,c,t)
%   camTitle=sprintf('Electron Wave Packet\nin a Laser');
%   text(.05,2.5,camTitle,'units','inches','fontsize',14,'fontweight','bold','color','b')
%   subTitle=sprintf('Camera position:');
%   text(.45,1.7,subTitle,'units','inches','fontsize',11,'color','k')

% %   These 7 lines let you see the image you're producing
% %   subplot(1,2,2,'position',[.3 .25 .65*aratio .65]);
  imagesc(finalimage)
  axis off;
  Title=sprintf('t=%2.4f and theta=%2.4f and phi=%2.4f',t,mod(thetaw*180/pi,360),mod(phiw*180/pi,360));
  title(Title)
  drawnow
  set(0,'DefaultImageCreateFcn','axis image')

%   These two lines save the frames as tifs
%   filename=[moviename 'frame' num2str(s) '.tif'];
%   imwrite(finalimage,filename,'tif');
%-------------------------------------------------------------------------
  toc
%   pct = fprintf('Percent completed: %2.1f%%\n',s/(nframe+1)*100);
end
```

```
function [intensity]=calculateWave(Ao,a,pw,m,n,r0,rmid,camx,camy,c,t)
stepSize=.015;
delx=norm(camx);
dely=norm(camy);

pinhole=rmid+r0; % position of the pinhole (for convenience)
intensity=zeros(m,n); % initializing image array

% r=.25+3*t/200;% approximation to size of electron cloud
r=1; % when pw is small
r2=norm(r0);  % distance from pinhole to electron
                                        % center

% integrating through the wavepacket at a specified time
parfor k=1:m
    testrow=zeros(1,n);
for l=1:n
    pinholet=pinhole;

    dvec=-1/norm(r0)*c*r0-(k-(m+1)/2)*camx...
        -(l-(n+1)/2)*camy;% dvec is vector from pixel to pinhole

        % calculate the bounds of integration:
    v1=(r2-r)/c; % approximate start and stop positions of wavepacket
    v2=(r2+r)/c;

    u1=(r2-1)/c; % bounds to the grid calculator
    u2=(r2+1.75)/c;

    U=u1:stepSize:u2;

    indexV=find((U>=v1)&(U<=v2));
    lRho=length(indexV);
    U3=indexV(1); U4=indexV(lRho); % the indices marking the approximate e- radius
    rhoL=zeros(lRho+1,1);
    phaseL=zeros(lRho,1);

    for u=U3:U4
        x=pinholet(1)+U(u)*dvec(1);
        y=pinholet(2)+U(u)*dvec(2);
        z=pinholet(3)+U(u)*dvec(3);

            v=u+2-U3;  % rhoL(1)=0,  then we add 0 to every other result
            [rhoL(v),phaseL(v)]=Evaluate(x,y,z,t,pw,Ao,a);

                % probability calculator that includes volume effects
                    dOut=c*U(u); % distance from pinhole
                    delV=1/3*delx*dely/c^2*((dOut)^3-(dOut-c*stepSize)^3);
                    % volume containing the rhoL(v) probability density
                  rhoL(v)=rhoL(v)*delV; % prob. = prob. density * volume
                Add=rhoL(v)+rhoL(v-1);

                rhoL(v)=Add;
    end
    testrow(l)=rhoL(lRho+1);  %get last value of rhoL
end
intensity(k,:)=testrow;
end

% ------------Testing validity of volume factor in calculating probability
% density-----------------------------------------------------------------
intensitysum=sum(sum(intensity));
fprintf('Time: %2.1f\t  Sum of intensity array: %2.3f\n', t,intensitysum)
% % ----------------------------------------------



  peak=max(max(abs(intensity)));
  if peak > 0
    intensity=intensity/peak;
  end
```

```
function [rho,phase]=Evaluate(x,y,z,t,pw,Ao,a)
pw2=pw^2;
pw4=pw^4;
Ao2=Ao^2;
fx=a*Ao/(2*pi)*sin(2*pi*(z-t));
fxd=-a*Ao*cos(2*pi*(z-t));
%!!!!!!!!!!!!!!!!!!!!Three possible polarizations here:!!!!!!!!!!!!!!!!!
% % % light polarized in x direction only
% fy=0;
% fyd=0;
% f=a*Ao2*((z-t)/4+sin(4*pi*(z-t))/(16*pi));
% fd=-a*Ao2*(cos(2*pi*(z-t))).^2/2;
%==================================================
% for light polarized linearly along the y=x diagonal
% fy=a*Ao/(2*pi)*sin(2*pi*(z-t));
% fyd=-a*Ao*cos(2*pi*(z-t));
% f=a*Ao2*((z-t)/2+sin(4*pi*(z-t))/(8*pi));
% fd=-a*Ao2*(cos(2*pi*(z-t))).^2;
%==================================================
% circularly polarized light
fy=a*Ao/(2*pi)*cos(2*pi*(z-t));
fyd=a*Ao*sin(2*pi*(z-t));
f=a*Ao2*(z-t)/2;
fd=-a*Ao2/2;
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ntp=a*t+f;
ntpd=a+fd;
ntm=a*t-f;
ntmd=a-fd;
nx=a*x+fx;
ny=a*y+fy;
nxd=fxd;
nyd=fyd;
nz=a*z+f;
nzd=fd;
al=1+1i*pw2*ntp;
ald=1i*pw2*ntpd;
b=1+1i*pw2*ntm+pw4*(fx.^2+fy.^2)./al;
bd=1i*pw2*ntmd+2*pw4*(fx.*fxd+fy.*fyd)./al-pw4*ald.*(fx.^2+fy.^2)/al.^2;
n=nz+1i*pw2*(fx.*nx+fy.*ny)/al;
nd=nzd-1i*pw2*ald.*(fx.*nx+fy.*ny)./(al.^2)+1i*pw2...
   *(fxd.*nx+fx.*nxd+fyd.*ny+fy.*nyd)./al;
q=real((nx.^2+ny.^2)./al+n.^2./b);
w=imag((-pw2*(nx.^2+ny.^2)./(2*al)).*ald./al+(-pw2*n.^2./b).*bd./(2*b)...
   +pw2*((nx.*nxd+ny*nyd)./al+n.*nd./b));
rho=(a*pw/sqrt(pi))^3/a*exp(-pw2*q).*(ntmd+w)./((abs(al)).^2.*abs(b));

% Checking approximations
pw3=pw^3;
pw4at=pw4*a*t;
fxpw3=pw3*a*Ao/(2*pi);
fypw3=pw3*a*Ao/(2*pi);
fpw3=pw^3*a*Ao2*t/4;
```

```
% fprintf('t = %g    pw^4*a*t = %g     pw^3*f = %g     ...
% pw^3*fx = %g     pw^3*fy = %g\n',t,pw4at,fpw3,fxpw3,fypw3)


% phase=-1i*ntm-pw2*(nx.^2+ny.^2)./(2*al)-pw2*n.^2./(2*b); % an actual phase
phase=0; % until we find a way to compute phase faster than it oscillates
```

```
function newVector = Rotate(aspin,theta,phi,vector)
%
R1=[cos(phi),0,sin(phi);0,1,0;-sin(phi),0,cos(phi)];
R2=[1,0,0;0,cos(theta),sin(theta);0,-sin(theta),cos(theta)];
R3=[cos(aspin),-sin(aspin),0;sin(aspin),cos(aspin),0;0,0,1];
newVector=(R2*R1*R3*vector')';
```

```
function SeeCam(camx,camy,r0,rmid,m,n,c,t)
% showing where camera is, for testing

% our wave packet ranges over:
% x from -1 to 1
xlength=2;
% y from -1 to 1
ylength=2;
% z from 0 to 10
zlength=10;


pinhole=rmid+r0; % pinhole position

%box outlining imaging plane
caml=zeros(m,3);
k=1;
for l=1:m
caml(l,:) = rmid+r0+1/norm(r0)*c*r0+(l-(m+1)/2)*camx...
        +(k-(n+1)/2)*camy; % pixel position
end
plot3(caml(:,3),caml(:,2),caml(:,1))
hold on;

camr=zeros(m,3);
k=n;
for l=1:m
camr(l,:) = rmid+r0+1/norm(r0)*c*r0+(l-(m+1)/2)*camx...
        +(k-(n+1)/2)*camy; % pixel position
end
plot3(camr(:,3),camr(:,2),camr(:,1))

camt=zeros(n,3);
l=1;
for k=1:n
camt(k,:) = rmid+r0+1/norm(r0)*c*r0+(l-(m+1)/2)*camx...
        +(k-(n+1)/2)*camy; % pixel position
end
plot3(camt(:,3),camt(:,2),camt(:,1))

l=m;
camb=zeros(n,3);
for k=1:n
camb(k,:) = rmid+r0+1/norm(r0)*c*r0+(l-(m+1)/2)*camx...
        +(k-(n+1)/2)*camy; % pixel position
end
plot3(camb(:,3),camb(:,2),camb(:,1))

axis([0 15 -4 4 -4 4])

% setting up the 'coordinate axes' for the viewer
numpoints=50;
deltax=4*xlength/50;
```

```
deltaz=2*zlength/50;
X=-2*xlength:deltax:2*xlength; % now the X, Y, Z arrays are all the same length
Y=-xlength*2:deltax:ylength*2;
Z=0:deltaz:2*zlength;
Blank1=zeros(1,numpoints+1);
Blank2=ones(1,numpoints+1);

plot3(Blank1,Blank1,X,'k') % plotting the x edges of the 'box'
% plot3(Blank1,-ylength/2*Blank2, X,'k')
%                 note: the wavepacket is contained entirely in the box
% plot3(zlength*Blank2,-ylength/2*Blank2,X)
% plot3(Blank1,ylength/2*Blank2,X,'k')
% plot3(zlength*Blank2,ylength/2*Blank2,X)

plot3(Blank1,Y,Blank1,'k')% plotting the y edges of the 'box'
% plot3(Blank1,Y,-xlength/2*Blank2,'k')
% plot3(zlength*Blank2,Y,-xlength/2*Blank2)
% plot3(Blank1,Y,xlength/2*Blank2,'k')
% plot3(zlength*Blank2,Y,xlength/2*Blank2)

plot3(Z,Blank1,Blank1,'k')% plotting the z edges of the 'box'
% plot3(Z,-ylength/2*Blank2,-xlength/2*Blank2)
% plot3(Z,ylength/2*Blank2,-xlength/2*Blank2)
% plot3(Z,-ylength/2*Blank2,xlength/2*Blank2)
% plot3(Z,ylength/2*Blank2,xlength/2*Blank2)

% plot3(rmid(3),rmid(2),rmid(1),'kh') % center of camera rotation
plot3(pinhole(3),pinhole(2),pinhole(1),'x') % pinhole position
plot3(0,0,0,'g*')% to denote the start position of the electron wave
plot3(rmid(3),rmid(2),rmid(1),'ro') % position of electron wave center


%the normal to the camera plane
dp=-cross(camx,camy);% note: the normal vector is only negative here because
                % our coordinate axes don't follow the right-hand rule
                % in our depiction of the camera
dp=dp/norm(dp)*c;
dl=norm(dp);

% T is a matrix whose entries reflect the time intervals for
% the line equation from the camera through the box when the camera was
% along the z axis
T=zeros(1,numpoints);
r=1*t/50;
% delt=norm(rmid-[0,0,zlength/50*t])/numpoints;
delt=2*r/numpoints;
t0=(norm(rmid-pinhole)-r)/dl;
for i=1:numpoints+1
   T(i)=t0+delt*(i-1);
end

lp=[pinhole(1)+T*dp(1);pinhole(2)+T*dp(2);pinhole(3)+T*dp(3)];
plot3(lp(3,:),lp(2,:),lp(1,:),'g')
```

```
%tracing the volume visible to camera
d1=[pinhole(1)-camb(1,1),pinhole(2)-camb(1,2),pinhole(3)-camb(1,3)];%dvector=<a,b,c>
l1=[pinhole(1)+T*d1(1);pinhole(2)+T*d1(2);pinhole(3)+T*d1(3)];
plot3(l1(3,:),l1(2,:),l1(1,:),'c')

d2=[pinhole(1)-camb(n,1),pinhole(2)-camb(n,2),pinhole(3)-camb(n,3)];%dvector=<a,b,c>
l2=[pinhole(1)+T*d2(1);pinhole(2)+T*d2(2);pinhole(3)+T*d2(3)];
plot3(l2(3,:),l2(2,:),l2(1,:),'c')

d3=[pinhole(1)-camt(1,1),pinhole(2)-camt(1,2),pinhole(3)-camt(1,3)];%dvector=<a,b,c>
l3=[pinhole(1)+T*d3(1);pinhole(2)+T*d3(2);pinhole(3)+T*d3(3)];
plot3(l3(3,:),l3(2,:),l3(1,:),'c')

d4=[pinhole(1)-camt(m,1),pinhole(2)-camt(m,2),pinhole(3)-camt(m,3)];%dvector=<a,b,c>
l4=[pinhole(1)+T*d4(1);pinhole(2)+T*d4(2);pinhole(3)+T*d4(3)];
plot3(l4(3,:),l4(2,:),l4(1,:),'c')

hold off
```

linedrawingscript.txt

```
% Drawing straight lines
% Using Bresenham's algorithm, as found on wikipedia
% http://en.wikipedia.org/wii/Bresenham's_line_algorithm
function imagearray=linedraw(xstart,ystart,shadestart,xf,yf,shadef,imagearray)

% start with a square array of length
% n=200;
% imagearray=zeros(n,n);


isSteep=abs(yf-ystart) > abs(xf-xstart);
if isSteep % swap x and y if |slope| > 1
    xi1=xstart;
    xi2=xf;
    xstart=ystart;
    xf=yf;
    ystart=xi1;
    yf=xi2;
end
if xstart>xf
    xi=xstart;
    yi=ystart;
    shadei=shadestart;
    xstart=xf;
    ystart=yf;
    shadestart=shadef;
    xf=xi;
    yf=yi;
    shadef=shadei;
end

deltax=fix(xf-xstart);
deltay=abs(fix(yf-ystart));
if deltax~=0
    deltashade=(shadef-shadestart)/deltax;
else
    deltashade=0;
end
error=0;
deltaerror=deltay/deltax;

y=ystart;
ystep=1;
if ystart>yf
    ystep=-1;
end

shadeHere=shadestart;

for x=xstart:xf

    if isSteep
        if shadeHere>imagearray(y,x)
            imagearray(y,x)=shadeHere;
        end
    else
        if shadeHere>imagearray(x,y)
            imagearray(x,y)=shadeHere;
        end
    end
    shadeHere=shadeHere+deltashade;

    error=error+deltaerror;
```

```
        if error >= .5
            y=y+ystep;
            error=error-1;
        end
    end
```

Bibliography

[1] J. P. Corson and J. Peatross, “Quantum-electrodynamic treatment of photoemission by a single-electron wave packet,” Phys. Rev. A 84 (2011).

[2] J. P. Corson, J. Peatross, C. Mueller, and K. Z. Hatsagortsyan, “Scattering of intense laser radiation by a single-electron wave packet,” Phys. Rev. A 84 (2011).

[3] J. Peatross, C. Mueller, and C. H. Keitel, “Electron wave-packet dynamics in a relativistic electromagnetic field: 3-D analytical approximation,” Opt. Express **15,** 6053–6061 (2007).

[4] G. Mocken and C. Keitel, “Quantum dynamics of relativistic electrons,” J. Comput. Phys. **199,** 558–588 (2004).

[5] G. R. Mocken and C. H. Keitel, “FFT-split-operator code for solving the Dirac equation in 2+1 dimensions,” Comput. Phys. Commun. **178,** 868–882 (2008).

[6] J. S. Roman, L. Plaja, and L. Roso, “Relativistic quantum dynamics of a localized Dirac electron driven by an intense-laser-field pulse,” Phys. Rev. A 64 (2001).

[7] Wikipedia, “Hydrogen eigenstate n4 l3 m1,” http://en.wikipedia.org/wiki/Hydrogen_atom (Accessed March 28, 2012).

[8] P. Falstad, “Hydrogen Atom Applet,” http://falstad.com/qmatom/ (Accessed January 30, 2012).

[9] M. Slater, A. Steed, and Y. Chrysanthou, *Computer Graphics and Virtual Environments: From Realism to Real-Time* (Pearson Education Limited, Essex, England, 2002), pp. 194–202.

[10] W. Newman, *Principles of Interactive Computer Graphics* (McGraw-Hill, New York, 1973), pp. 235–282.

[11] R. Kingslake, *An Introductory Course in Computer Graphics* (Chartwell-Bratt, Sweden, 1986).

[12] D. Rogers, *Mathematical Elements for Computer Graphics*, 2 ed. (McGraw-Hill, New York, 1990).

[13] Wikipedia, "Bresenham's line algorithm," http://en.wikipedia.org/wiki/Bresenham's_line_algorithm (Accessed January 24, 2012).

Index