5-14-2007

Computer Modeling and Visualization of Luminescent Crystals: The Role of Energy Transfer and Upconversion

Christopher S. Stuetzle

Computer Modeling of Energy Transfer in Crystal

Article · April 2007

CITATIONS
0

READS
20

3 authors, including:

Christopher Stuetzle
Merrimack College
21 PUBLICATIONS **34** CITATIONS

SEE PROFILE

Computer Modeling and Visualization of Luminescent Crystals:

The Role of Energy Transfer and Upconversion

BY

Christopher S. Stuetzle

of

Wheaton College

in Partial Fulfillment of the Requirements

for

Graduation with Departmental Honors

in Computer Science

Norton, Massachusetts

May 14th, 2007

Acknowledgments

This work is for my family and friends, without whom my life would be incomplete, for my teachers, without whom I would never have come this far, and for my advisers, without whose guidance and patience this work would have never happened.

Table of Contents

Chapter 1 - Introduction

The work presented in this thesis describes the development of computer software for the purpose of investigating energy transfer processes in luminescent crystals. There are two distinct components of the work. The first component is the development of the software, etVisual, a computer program designed to simulate the firing of a laser at a crystal. The second component involves using the software to demonstrate the effect of energy transfer processes on the luminescence properties of crystals and to compare the output to experimental data.

Luminescence is the spontaneous emission of light from an atom that has been excited in some way. In the systems of concern to this work, this luminescence originates from an impurity (or dopant) ion embedded in a crystal. The luminescence from impurity ions (usually transition metal ions or rare earth ions) in ionic crystals is important to lasers and lamp phosphors (phosphors used in halogen lamps), among other applications. For instance, pump lasers excite these ions within the crystal with energy before they emit photons in phase (all with the same frequency), thus creating a laser beam. In phosphor applications, the luminescence from these ions is responsible for the visible light emitted from the lamp. However, many aspects of the luminescent property of crystals are not fully understood. In particular, the luminescence properties of these crystals depend on the particular locations of the ions in the crystal. There are some processes for which certain arrangements of ions will cause the energy in the crystal to be lost, while other arrangements of ions can favor emission of a photon. This study aims to

take a step forward in understanding the effect of one such process: non-radiative energy transfer among impurity ions in the crystal. One aspect of this process that is of particular concern to us is upconversion.

Understanding the behavior of these impurity ions and how they affect the emission properties of the crystal is vital. In the laboratory, the experiment most useful for gaining insight into the energy transfer process is the examination of the kinetic properties of emission; that is, we examine the shape of the intensity time curve of the photons emitted from the crystal. In a real crystal, the arrangement of the ions is unknown, though they are generally assumed to be random for the purpose of interpreting the data. It is well known, however, that the distribution of ions is in fact not random, bringing the interpretation into question. In this work, we have created a computer program that will allow for the examination of the luminescence and energy transfer properties of the impurity ions of a crystal in situations where the ions are placed in the crystal in both a random and non-random manner.

There are several motivations for our work. Because the computer allows the user to exercise control over the distribution of ions in a crystal, the previous assumption that ions are arranged randomly can be challenged. The interpretations of experimental observations are based on the assumption of random distribution. Thus, we hope to test the validity of these interpretations when the arrangement of atoms within the crystal is non-random, as is often the case. Another motivation is to be able to conduct these investigations with different types of crystals. This ability would allow the comparison to data available in the literature. A third motivation is the application of computer

visualization to display energy transfer in a way that is easily understood by a user. A fourth motivation is to create a piece of software that will be useful in the study of energy transfer. The software will be open source, meaning that its source code will be made available to anyone to alter and use as he pleases. On a personal note, the author is passionate about the two disciplines of computer science and physics, and looks for ways to combine the two by using computers to solve physics problems, and to help users understand the physics behind the solutions.

There were two major aims of this work. The first was to produce a fully functioning computer program that could allow the virtual formation of a crystal structure and visualize the energy transfer following excitation from a virtual laser. This aim was reached with the development of etVisual, an original piece of software that creates a visualization and takes measurements of energy transfer within a crystal. In order to meet our aim of creating a fully-functional program, several goals had to be realized. For easy user input, a graphical user interface must be developed. The program also must take the user input and develop and draw a crystal. These goals were met with the development of a user-friendly graphical user interface, with inputs for various parameters needed to create a crystal lattice structure.

Once able to create and draw a crystal structure, the program must be able to perform a series of simulations on the crystal. The program needs to be able to simulate adding impurity (dopant) ions to the crystal, both randomly and non-randomly. The program also needs to be able to display a visualization of the crystal structure, as well as laser excitation, energy transfer, and upconversion within the crystal. Oftentimes, when

trying to understand a topic as complex as energy transfer in crystals, being able to see what is occurring leads to a better understanding of the process. etVisual's visualization component allows for such an understanding by showing the formation of a crystal, followed by the initial excitation by a laser, and ending with the slow decay of the system's total energy. The visualization uses coloring techniques to better convey energy transfer and upconversion, and plays a big role in the future utility of etVisual.

Finally, the program needed to be able to output the intensity time data, or the photon emission data, of the crystal. This output would be used to reach the second aim of this study: to use the program to investigate energy transfer within crystals. We aim to study emission without energy transfer processes and with energy transfer processes. The study aims to show how energy transfer, upconversion, and photon emission are affected by the dopant ion distribution, doping percentage, and excitation intensity.

This study focused on the Nd:YAG crystal: a Yttrium Aluminum Garnet with Neodymium replacing chosen Yttrium ions. This crystal has been shown to have its photon emission affected by the upconversion process. Nd:YAG is a crystal that is popular in lasers. In a Nd:YAG Pump Laser, a pump hits the Nd:YAG crystal with photons, most of which are absorbed. As the crystal emits its energy in the form of photons in phase, a laser beam is formed.

The organization of this thesis is as follows. The second chapter discusses several physical properties of crystals that etVisual uses to build various crystal structures. The chapter then outlines crystal doping, what it is, and how it is accomplished. Two major doping procedures taken into account in this study, random and non-random doping, are

discussed. The chapter then addresses energy transfer processes within crystals, explaining photon absorption, photon emission, and conveying the details of the energy transfer algorithm etVisual uses to model the processes.

Chapter three goes through a brief discussion of past studies regarding energy transfer upconversion. It also contains a brief discussion of previous visualizations of crystal structures and computer models of physical behavior in crystals.

Chapter four discusses the programming involved in the creation of etVisual. The program's code involves several different sections, one for the user interface, one for visualization and animation, and one for the random number generation. The chapter explains each of these topics, discussing coding techniques employed while creating the program, and drawbacks to methods used. A discussion of both FLTK (Fast Light Toolkit) and OpenGL (Open Graphics Library) is contained within this chapter. Both APIs were used in the programming of etVisual, and the reasons for their choosing is discussed. A brief performance test was also performed, and the results are reported.

Chapter five presents the results of the research. These results, presented in graphs, display the luminescent output of the crystals tested. These results are examined and interpreted and the questions raised in the goals above are answered.

Chapter six is a brief discussion and conclusion, including a section outlining future opportunities and goals. This chapter ties our goals into the “bigger picture” of crystal studies, and how this work can contribute to this area of study. The chapter also discusses future goals for etVisual itself, and how we imagine it can and will be used in the future.

Chapter 2 – Physics Background

Given the goals of this work presented in Chapter 1 (page 12), we now set out to discuss the relevant physics-related principles necessary for achieving those goals. This chapter will describe in detail the physical processes that etVisual computerizes. We first go about building the crystals on the computer, and describe the process of inserting the dopant ions into the crystal in both random and non-random arrangements. We then discuss the energy transfer processes that are relevant to this thesis. Understanding these processes is vital to expanding our understanding of how energy transfer processes affect the light emission from these crystals.

2.1 Crystal Formation

The program first builds a crystal structure. Several crystal formations are described in Wyckoff's *Crystal Structures* [11]. For the purposes of this study, the building of a crystal structure revolves around the formation of a unit cell containing a certain number of base points. Base points refer to the physical locations of particular atoms. These are the points on which the crystal operations described below are performed. The base points and the appropriate operations are supplied in the text by Wyckoff.

The first step in the formation of a crystal structure is to input base points within a single unit cell. These base points are then duplicated and translated based on the rules of the operations below, creating a single unit cell. The unit cell is then duplicated several times in each the x-, y-, and z- directions, as shown in Figure 1.

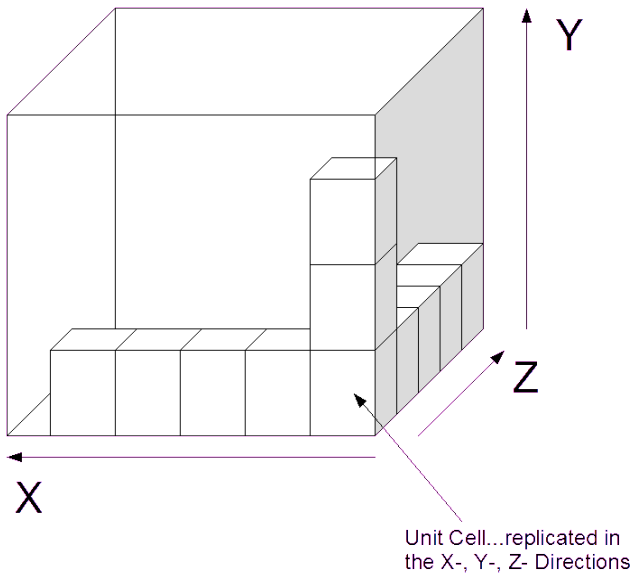

Figure 1: Diagram depicting the formation of a crystal. A unit cell is drawn and replicated in each X-, Y-, and Z-directions.

There are three basic operations described by Wyckoff that are used to form a unit cell in this study: the body center, face center, and transposition operations. Once the base points are defined, any of the above operations, or combination of the operations, can be performed on them. The original base points are left unchanged, and new points are added, so that each operation will increase the number of points in the unit cell by some factor, depending upon what the operation does. For the remainder of this section, let (x, y, z) be the coordinate representation of any base point in the set of all base points of a unit cell. The following operations, therefore, are performed on all (x, y, z) in the unit cell.

The first operation is the **body center (BC)** operation. When forming a unit cell through the body centering operation, ½ is added to each coordinate of each base point.

For example, (x, y, z) gets duplicated as (x + ½, y + ½, z + ½). If we let the base points be (0, 0, 0) and (¼, ¼, ¼), then the unit cell under the BC formation become (0, 0, 0), (¼, ¼, ¼), (½, ½, ½), and (¾, ¾, ¾). Under the BC formation, the number of points in the unit cell is twice the number of base points. An example of the Body Center formation is shown in Figure 2.

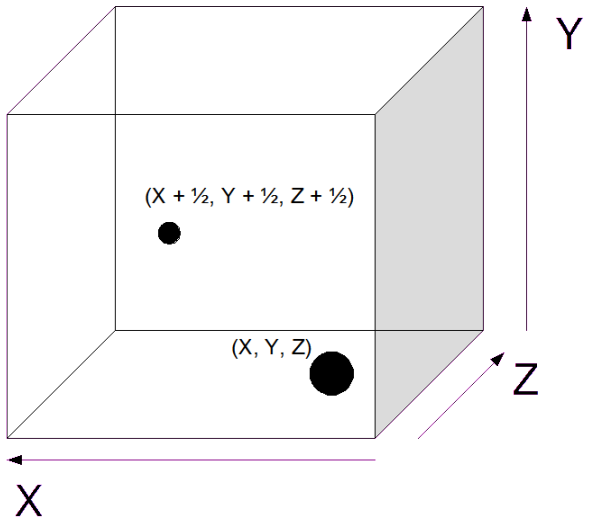

Figure 2: The Body Center formation of a crystal. Each base point, represented by (X, Y, Z) in the figure, has 1/2 added to each of its coordinates. This operation doubles the total number of points in the unit cell. The smaller point represents a larger z-coordinate, because it is farther away from the viewer.

In all figures in this section, point size represents distance from the viewer. The longer the Z coordinate, the smaller the point.

The second operation is the **face center (FC)** operation. When forming a unit cell by using face centering, three new points are formed for each base point. For the base point (x, y, z), the three new points that are created are of the form (x + ½, y + ½, z), (x

+ ½, y, z + ½), and (x, y + ½, z + ½). For instance, for the base points (0, 0, 0) and (¼, ¼, ¼), the unit cell that would result would be (0, 0, 0), (¼, ¼, ¼), (½, ½ , 0), (½ , 0, ½), (0, ½, ½), (¾, ¾, 0), (¾ , 0, ¾), and (0, ¾, ¾). This then quadruples the number of points in the unit cell. An example of the FC operation can be found in Figure 3.

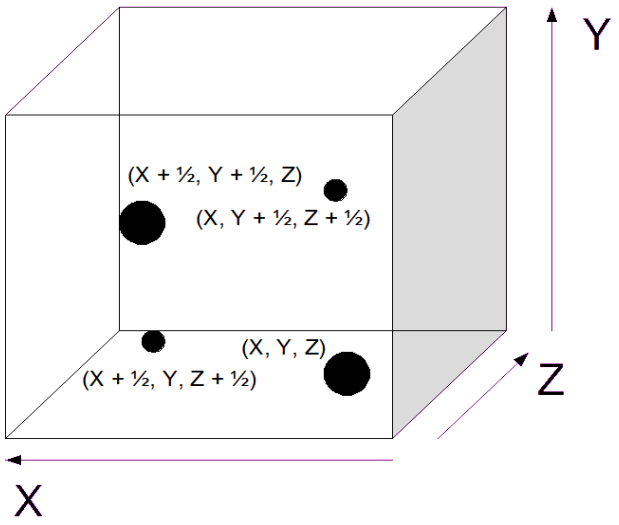

Figure 3: The Face Center formation of a crystal. Each base point, represented by (X, Y, Z) in the figure, is duplicated 3 times, with 1/2 added to its X- and Y-coordinates in one replication, 1/2 added to its X- and Z-coordinates in another replication, and 1/2 added to its Y- and Z- coordinates in the third replication. This operation quadruples the number of points in the unit cell.

The third operation is the **transposition (tr)** operation. In the tr operation, the three coordinates are rotated and the number of points in a unit cell is tripled (adding two for each base point). However, it should be noted that one base point transposes and becomes three only if the base point's x, y, and z coordinates are not all equal to one another. If the three coordinates are equal, all three transpositions of the base point will be the same, and so are only counted as one base point. For the base point (x, y, z), the

two new points that are added are (y, z, x) and (z, x, y). In this operation, the coordinates are treated like a cyclic permutation, causing the newly created points to make a triangular formation based around the X = Y = Z axis. For example, for the base points (0, ¼, ½) and (0, ¼, ⅛), the unit cell that would result from the transposition operation would be (0, ¼, ½), (¼, ½, 0), (½, 0, ¼), (0, ¼, ⅛), (¼, ⅛, 0), and (⅛, 0, ¼). An example of this operation can be seen in Figure 4.

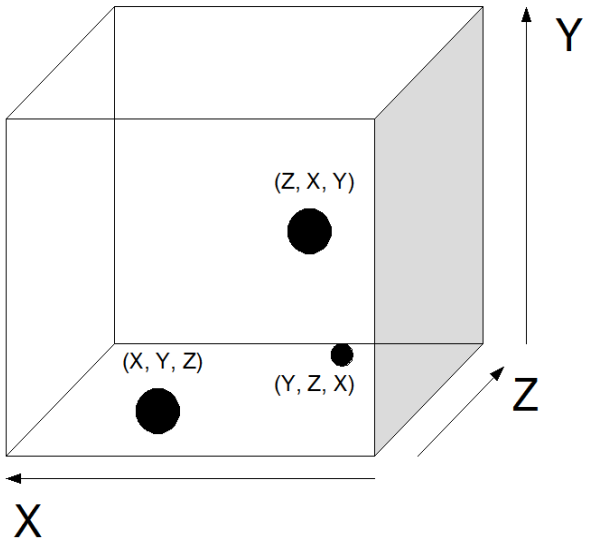

Figure 4: The tr crystal formation. Each base point, represented by (X, Y, Z) in the figure, is treated as a cyclic permutation, where the other points are permutations of the original X-, Y-, and Z- coordinates.

Any combination of the above operations can be applied to the base points to create a unit cell. For instance, if the operation is trFC (transposition and then face center), then the base points have the tr operation applied to them first, creating three times as many base points. These new points are all treated as base points and have the face center operation applied to them. In this case, the number of base points is first

tripled by the tr operation, and then this new number of base points is quadrupled by the FC operation. After the operations, the total number of points in the unit cell is 12 times the number of original base points.

Once any operation, or combination of operations, has been performed on the base points, the unit cell is complete. This unit cell is now replicated in the x-, y-, and z-directions n times, as shown in Figure 1. The resulting crystal is comprised of $n \times n \times n$ unit cells. This completes the formation of a crystal structure.

2.2 Doping the Crystal

The physical characteristics of a crystal are determined by the elements that comprise it. The predefined base points that are operated on are actually the ions that make up the crystal. These ions do not all have to be of the same element, but may be a combination of several types within one crystal. Doping a crystal is the act of substituting impurity ions (dopants) in place of a certain type of atom found within it. For example, a commonly used crystal is Ytrium Aluminum Garnet (YAG). One may remove certain ytrium ions and replace them with neodymium ions, thus making the neodymium-doped YAG crystal, commonly used for lasers.

In etVisual, we use two methods to place these dopant ions into the crystal. The first and the simpler of the two is the random method, where ions are chosen at random to be replaced with a dopant ion. Before beginning the doping process, the user must first choose a percentage of ions to dope. Then the program selects a random sample of ions to remove. Those ions are then replaced with the dopants, or the ions replacing the chosen ions, and a random doping has occurred. In this method, each ion has an equal probability

of being replaced.

2.2.1 Non-Random Doping Method

The second and considerably more complicated of the two methods is non-random doping. This method places ions in clusters (groups of doped ions) near each other when an attractive reaction is desired, or away from each other when a repulsive reaction is wanted. This method allows for the study of clustering and how it may affect energy transfer processes, which are explained in detail in section 2.3.

The non-random method uses a complex algorithm to choose which ions to dope and which to leave. For the purpose of this study, we used the method for non-random doping described by Barbosa-Garcia and Struck [1]. In their algorithm, a random ion is chosen to be doped initially. Then, every other ion is assigned a weight based on its distance to the initial ion. This weight may be high or low depending on whether an attractive distribution (closer ions get a higher weight) or a repellent distribution (closer ions get a lower weight) is desired. Once every ion has been assigned a weight, an ion is selected by taking into account each ion's weight. For each subsequent ion that is to be doped, the procedure is repeated, where the weight given to any ion is now based on its distance from all previously doped ions.

Algorithmically, the process is as follows:

```
h[numberIonsToDope]           # an array of h values
dopedIons[numberIonsToDope]   # ions to dope
H = 0                         # running some of weights

percentToDope  # % to dope out

#First, select a random ion to dope
dopedIons[0] = random doping
```

```
numDopedIons = 1

while numDopedIons < ( percentToDope * totalIons )
     for each ion i
          randomNum = GetRandomNumber01() # Between 0 & 1
          update_h( i )  # update h[i]
          H = H + h[i]   # running sum of h's
     end for

     runningTotal = 0

     for each ion j
          if(j not already doped)
               runningTotal = runningTotal + h[j]
               if(randomNum < runningTotal)
                    dopedIons[numDopedIons] = j
                    numDopedIons = numDopedIons + 1
               end if
          end if
     end for

update_h( i )
     h[i] = max ( 0 , h[i] + sumOfGs( i ) )

end update_h

sumOfGs( i )
     runningSum = 0
     for each doped ion m
          runningSum = runningSum +
                         e^( α β γ e^(-R_i,m/R_min)^2 ) - 1
     end for
end sumOfGs
```

In the algorithm, each possible doping site (undoped ion) has its own *h* value. This value represents the ion's weight. The first part of the algorithm randomly selects an ion to dope. During each subsequent round of the algorithm, all of the ions' *h* values are updated based on their distances from each previously doped ion. The function `update_h( i )` is used to update ion *i*'s *h* value. An ion's *h* value (its weight) is determined by taking the ion's current *h* value and adding to it the sum of every doped

ion's *G* value, shown below in equation (1). Called from the `update_h( i )` function is the `sumOfGs( i )` function, which totals each doped ion's *G* value according to equation (1).

$$G = e^{\alpha\beta\gamma\left(e^{-R_{i,m}/R_{min}}\right)^2} \qquad (1)$$

There are three values in this equation that must be input before the algorithm is run, usually by the choice of the user: α can take the value 1, 0, or -1, where 1 is for an attractive reaction, 0 is for a random reaction, and -1 is for a repulsive reaction. An attractive reaction will tend to place doped ions close to one another as the name would suggest, and a repulsive reaction will tend to spread them out. β is defined by 1.0 / (*B* * *T*) , where *B* is the Boltzmann Constant is 1/1.4 cm^{-1} and *T* is the annealing temperature of the ion, and γ is the strength of the attractive or repulsive interaction. This value gives a user the ability to input whether the ions being doped are attractive or repulsive to one another. Finally, $R_{i,m}$ is the distance between ions *i* and *m*, and R_{min} is the minimum possible distance between two doping sites. The `sumOfGs( i )` function runs through all ions previously doped and keeps a running sum *G*. The function will return this running sum, where `update_h( i )` uses it to determine the i^{th} ion's new *h* value by adding the returned sum to the ion's previous *h* value. If this new *h* value is negative, the ion's *h* value is set to 0 because a negative weight is the same as a weight of 0. We want to weight all the ions that have no chance of absorbing energy from another ion the same so that they all start at the same base in the next iteration of the algorithm.

As each possible doping site's *h* value is determined, *H* is incremented. *H*

represents the running sum of all possible doping sites' *h* value. After *H* is fully calculated, a random number between 0 and 1 is generated and whichever ion's *h* value corresponds to that random number is the one that is doped that round. *H* is needed so that the random number generated can fall somewhere between 0 and the total of all *h* values, thus guaranteeing that at least one ion is doped each round. This is done by multiplying *H* by the random number, thereby creating a percentage of *H* which corresponds to one possible doping site's *h* value.

2.3 Energy Transfer and Emission

2.3.1 Photon Absorption

The impurity ions inside of a crystal all have energy levels, including a base energy level (ground state). An ion's base energy level is the energy the ion has while it is not excited. The amount of energy an ion holds is restricted to a series of energy levels, as ions can only be excited from one energy level to a higher one by absorbing the amount of energy needed to move between them.

An ion will become excited when it absorbs a photon (light particle) with enough energy to raise it a higher energy level. When a laser is fired at a crystal, some of the laser beam's photons are absorbed by ions in the crystal, exciting them. A laser's beam contains photons that, for most lasers, have the same frequency. The energy of a photon is given in equation (2).

$$E_{photon} = h\nu \qquad (2)$$

where ν is the photon's frequency and h is Planck's constant (6.626068×10^{34} meters^2 kilograms per second). Because the energy of a photon is based on the photon's frequency

(Planck's constant is invariant), and all photons in a laser's beam have the same frequency, all the photons in the beam must also have the same energy. For this reason, when a laser's beam enters a crystal, all of the ions that absorb photons from the beam acquire the same amounts of energy. Therefore, all excited ions that are of the same element are excited to the same energy level. This is the manner in which the ions inside of the crystal are excited in this and similar investigations.

2.3.2 Photon Emission

After a length of time, an ion will decay from its excited state. When this occurs, the ion must release the energy that it obtained in reaching its excited state because of the law of conservation of energy. Some energy is lost due to the emission of heat, but in the systems we will be modeling most of this energy is lost through the emission of a photon.

The energy of an emitted photon is equal to the difference between the two energy levels involved. This is demonstrated in equation (3).

$$E_{photon} = E_{initial} - E_{final} \qquad (3)$$

where E_{final} is the final energy level and $E_{initial}$ is the initial energy level of the ion. E_{photon} is the energy of the photon that is released when an ion goes from its initial energy level to its final energy level.

All of the photons that are emitted from the same type of ion will have the same energy because each ion's energy levels are the same, and all ions are struck with the same laser light, as described above. Because of this fact, this study will focus on the emission of photons in terms of quantity of photons as opposed to the amount of energy the ions released. This focus will also allow us to more easily see the effect that energy

transfer has on photon emission.

Every energy level of an ion type in a crystal has an associated lifetime, that is, an average time that it spends in an excited state. This lifetime is generally labeled *t*. The number of photons, *N*, emitted from a group of ions in such an energy level goes as:

$$N = N_0 e^{-t/\tau} \qquad (4)$$

where N_0 is the initial number of ions in the excited level and τ is the lifetime of an excited atom. For the purposes of calculating whether an ion will emit a photon in the simulation, we must speak in terms of probabilities of such an event. Based on equation (4), the probability of an excited ion returning to a lower energy level and releasing a photon per change in time (Δt where $\Delta t << \tau$, defined below). It is given by the following equation:

$$P_{emission} = 1 - e^{-\Delta t/\tau} \qquad (5)$$

where $P_{emission}$ is the probability of the ion releasing a photon. This is a standard decay curve based on Δt, the time interval over which the measurements are made. Each type of ion has its own lifetime, and so each doped crystal will emit photons at a different rate, depending upon the crystal type and the impurity ions used to dope it. The value of the lifetime input by the user.

For an example of the use equation (5), let the time interval (Δt) be .00001 seconds and the lifetime (τ) be .001 seconds. Then, using equation (5), we get:

$$
\begin{aligned}
P_{emission} &= 1 - e^{-\Delta t/\tau} \\
&= 1 - e^{-.001/.00001} \\
&= 1 - e^{-.001} \\
&= 1 - 0.99005 \\
&= 0.00995017
\end{aligned}
\quad .
$$

This means that each excited ion has a .995017% chance of emitting a photon each Δt seconds. In other words, .995017% of the excited ions in the crystal will emit a photon each Δt seconds. In the next millisecond roughly 1% of those ions that are still excited in the crystal will emit a photon, thus creating an exponential decay curve. An example of this data is seen below:

Formation Type	:	Body Centered
Δ t	:	.00001 seconds
Unit Cell Number	:	6 x 6 x 6
Doping%	:	15 %
Light Absorption%	:	10 %

Time (seconds)	Photons Absorbed – Photons Emitted
0.000	239
0.001	92
0.002	36
0.003	12
0.004	5
0.005	2
0.006	1
0.007	1
0.008	0

The sample decay pattern shown in the table above describes an exponential behavior that is a common observation in doped crystals. The lifetime can vary from being long (e.g. 3 milliseconds in a ruby, Cr-doped aluminum oxide, crystal) to short e.g.

a few tens of nanoseconds in Pr-doped yttrium silicate). This process explains the colored light that emits from the crystal when its ions are excited with laser light.

2.3.3 Energy Transfer

There is a second way in which excited ions within a crystal may lose energy. Ions may transfer their energy to a nearby ion, thus returning to a lower energy level while exciting the nearby ion to a higher one. The energy released from one excited ion and the energy absorbed by another are usually equal, as demanded by the law of conservation of energy.

There are two types of energy transfer that are of concern to us in this study. The first occurs when energy transfers from an excited ion to one in its ground state, and the newly excited ion reaches the same energy level as the originally excited ion, which returns to its own ground state. This process we call ground state absorption (GSA) and is depicted in Figure 5. It is the simplest form of energy transfer within a crystal because there is no energy lost in the process, just energy that has been moved from one ion to another. Because no energy is lost during this process, the process does not change the energy output of the crystal. In other words, the same amount of energy enters the crystal as leaves it, and the same type of photons are emitted. The crystal's photon output still produces an exponential decay curve.

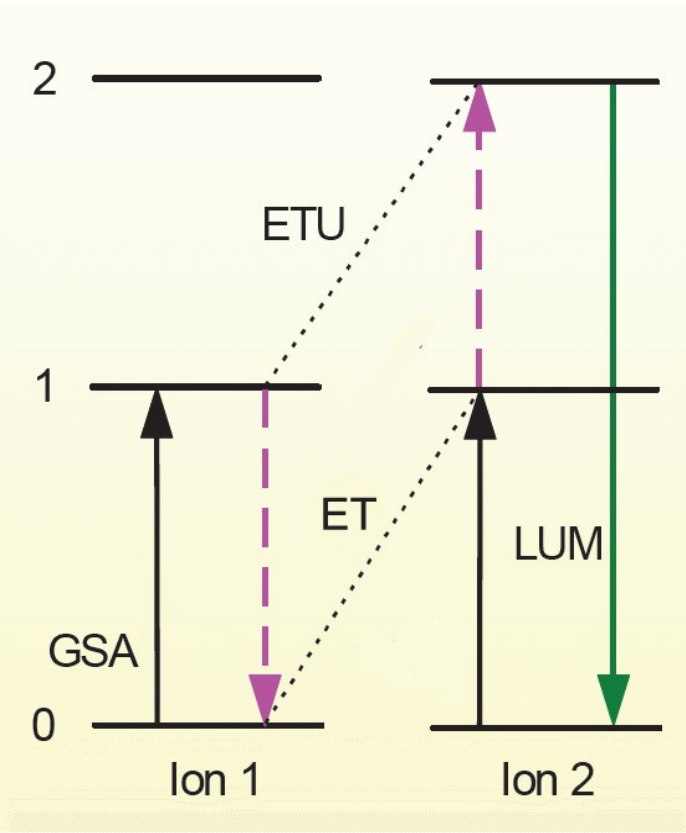

Figure 5: Energy transfer between two ions. Ground State Absorption (GSA), where ion 1 absorbs energy and becomes excited from its ground state, is the first process that takes place. Ion 2 either experiences Energy Transfer Upconversion (ETU) or energy transfer (ET), depending on whether or not it is excited before the transfer takes place. Luminescence is defined by LUM.

The second form of energy transfer in this study is Energy Transfer Upconversion (ETU), which occurs when an excited ion transfers energy to an already excited ion, and it is a main focus of this investigation. When ETU is not taken into account, measuring the emission of photons of a crystal produces an exponential decay curve as discussed above. However, upconversion can disrupt this decay behavior. Most applications of photon emission require photons to be emitted from a particular "target" energy level.

This is especially important in laser applications, as all photons released must have the same energy. However, when a photon is released from a higher energy level than the target one, the energy (and frequency) of the released photons will not match those released from the target level (level 1 in Figure 5). Some of the energy is released as a higher energy photon, or perhaps also as heat when an ion returns from a higher level to a lower one. Either process represents energy lost to the system. Thus, ETU will cause the emission of a crystal to not act as expected, disrupting laser production as well as other applications of crystal photon emission.

Although ions may have dozens of energy levels they can reach, this study focuses on only three energy states. The first is the base energy level, denoted E_0. The second is the excited state, denoted E_1. The final energy level taken into consideration is the doubly-excited state, denoted E_2. This state is reached only via upconversion. When an ion is already excited (at level E_1) and a nearby ion transfers energy to it, it enters the E_2 energy level. The focus of the study is limited to these states because the goal is to study the effect upconversion has on the luminescence (light/energy output) of a crystal. In this way, E_1 represents our "goal" energy level, the one whose output we are measuring, and the one which is responsible for the laser emission. E_2 is the energy level reached by upconversion. The specific energy level an ion reaches after upconversion is not discussed here and is ignored.

2.4 Energy Transfer Algorithm

This study is concerned with observing and measuring the effect that

upconversion has on the energy output of a crystal. With no upconversion, we know the crystal's energy output is described by an exponential decay curve. However, we wish to know the effect of upconversion on the crystal's emission. For this reason, a discrete time algorithm (an algorithm that uses set time intervals to see when events occur) was implemented in etVisual to mimic the response of the system following the firing of a laser beam at a crystal.

Before the algorithm begins, a user-defined percentage of the crystal's dopants have absorbed photons. That is, a percentage of the randomly chosen impurity ions in the crystal are already at energy level E_1, while the rest start at the base energy level E_0. The algorithm counts up by the discrete time interval Δt. For each Δt, the algorithm checks each ion that is excited (at energy level E_1) and decides whether the ion emits a photon. If an ion does emit a photon, the algorithm moves on to the next excited ion. If it does not emit, then the algorithm checks to see whether it transfers its energy to a nearby dopant. If energy is transferred between the two doped ions, then the energy level of the first is decremented while the energy level of the second is incremented. If it does not transfer energy, than nothing happens and the algorithm moves on to the next excited ion. The algorithm follows:

```
for each Δ t
     for each excited ion i
          probability = P_emission

          randomNumber = GetRandomNumber()

          if(probability < randomNumber)
               EmitPhoton()
               ion[i].absorbedPhoton = false
          else
```

```
                for each doped ion j
                     distance = getDistance(i, j)
                     if(ion[j].getEnergyLevel() == 1)
                          Kvalue = K_two
                     else
                          Kvalue = K_one
                     end if else

                     probability = probability +
                                    (Kvalue/distance^6)

                     if(probability < randomNumber)
                          ion[i].emit()
                          ion[j].absorb()
                     end if
                end for j
           end if else
      end for i
end for
```

The algorithm runs through every excited ion to decide what happens to its energy level. In order to do this, a random number is generated, and the algorithm uses the "bins" method to determine the next course of action for the energy in the current ion. The variable `probability` is responsible for creating the bins that `randomNumber` will fall into, as shown in Figure 6.

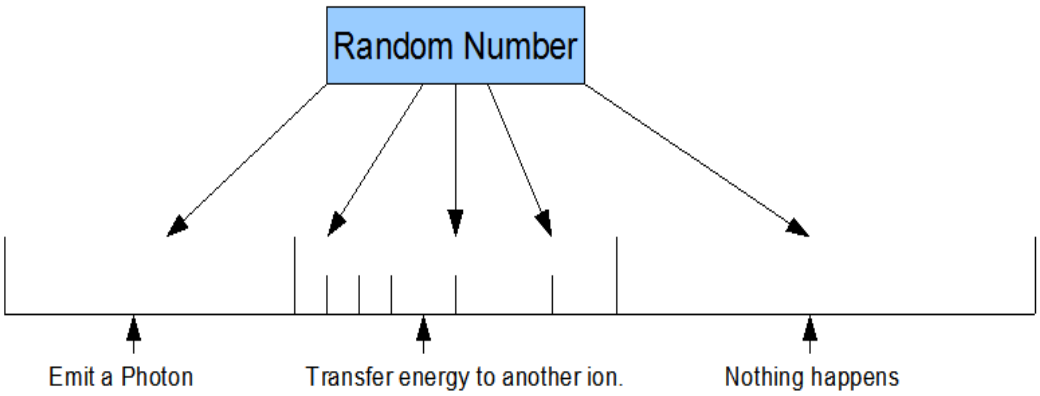

Figure 6: The "bin" strategy used to determine each excited ion's course of action. A random number is selected and the bin it falls in determines what happens next. The smaller bins inside of "Transfer Energy..." indicate to which ion the excited ion will transfer its energy.

First, `probability` is set to the probability of photon emission. This creates the “Emit a Photon” bin shown in the figure. If `randomNumber` is less than `probability`, then it “falls” into the “Emit a Photon” bin. However, if `randomNumber` does not fall into this bin, then the algorithm checks every other doped ion to see if the current excited ion transfers energy to that doped ion.

In order to check whether or not the current ion transfers energy, `Kvalue` must be found. This value is the constant associated with energy transfer. It is necessary to have the probability of energy transfer to be of the same order of magnitude as the probability of photon emission. However, `Kvalue` has a different value for each different energy transfer type (`K_one` refers to the constant of single-level transfer and `K_two` refers to the constant of upconversion). This allows for the possibility that single-level energy transfer may be less or more likely than upconversion. By multiplying each type of transfer probability by a constant, it is possible to manipulate the likelihood of

each event. Some ions may be more or less likely to experience upconversion, based on their individual properties.

In order to find `Kvalue`, or rather find `K_one` and `K_two`, a generic constant K must be found. This generic constant is the constant for all energy transfer. Its purpose is to bring the order of magnitude of energy transfer to a comparable level with the probability of photon emission. The probability of energy transfer is given by equation (6):

$$P_{ET} = \sum \frac{1}{R_{ij}} \qquad (6)$$

where P_{ET} is the probability of energy transfer and $R_{i,j}$ is the distance between the i^{th} and j^{th} ion, non-repeating. In other words, the probability of energy transfer is the sum of the reciprocals of the distances between all the ions. This makes sense, as the probability of energy transfer between the i^{th} and j^{th} ion is the reciprocal of their distance, and so the probability of energy transfer from any ion to any other is the sum of these smaller probabilities.

However, this value for P_{ET} may not be on the same order of magnitude as the probability of photon emission. It is desirable that the two processes be on similar orders of magnitude, or one may occur often while the other not at all, which is not a realistic scenario. In order to correct this, P_{ET} can be expressed as the product found in equation (7):

$$P_{ET} = K \sum \frac{1}{R_{ij}} \qquad (7)$$

where K is the constant associated with energy transfer, as described above. Equation (8)

follows from equation (8).

$$K = \frac{P_{ET}}{\sum \frac{1}{R_{ij}}} \qquad (8)$$

Now that we have an expression for K, we can use it to find K_1 and K_2, the constants of single-level transfer and upconversion discussed above. These values correspond to `K_one` and `K_two` in the algorithm above. First, however, the ratio of K_1 to K_2 must be determined. Because upconversion is the main process investigated in this study, we chose to use the variable K_{ratio} to represent the ratio of K_2 to K_1, so that it may be "turned off" (set to 0) without division by 0.

Now that K and K_{ratio} have been found, we can use them to define K_1 and K_2 based on the following equations.

$$P_{emission} = P_{ET} \qquad (9)$$

$$K = \frac{P_{emission}}{\sum \frac{1}{R_{ij}}} \qquad (10)$$

Equation (10) reflects the desire for the likelihood of energy transfer to be equal to the likelihood of photon emission. This constraint allows for easier studying of the effect energy transfer has on the output of a crystal as the constraint ensures transfer happens as often as emission.

Now that K has been established, the following equations are used to determine K_1 and K_2.

$$K_{Ratio} = \frac{K_{two}}{K_{one}} \qquad (11)$$

$$K = K_{one} + K_{two} \qquad (12)$$

$$K_{two} = K_{Ratio}\, K_{one} \qquad (13)$$

$$K_{one} = \frac{K}{1 + K_{Ratio}} \qquad (14)$$

Equation (13) follows from equation (11) and equation (14) follows from equations (12) and (13). These equations are used to calculate `K_one` and `K_two`, and so the algorithm knows these values before it begins.

In the energy transfer algorithm, each ion `j` has a probability associated with energy transfer from excited ion `i` to it. This probability is shown in equation (16).

$$P_{i,j} = \frac{K_{value}}{R_{i,j}^{6}} \qquad (15)$$

In equation (15), $P_{i,j}$ is the probability of energy transfer between ion `i` and ion `j`, and $R_{i,j}$ is the distance between ions `i` and `j`. This distance is raised to the 6th power, as is common in probability calculations. The variable `probability` in the algorithm is incremented according to the probability in equation (15). Each incrementation of `probability` creates a new bin for each ion that the i^{th} ion could transfer energy to. If `randomNumber` is less than `probability`, then it falls into whichever bin has just been created. Therefore, the excited ion that the algorithm is currently checking transfers energy to the ion that the bin represents, and the algorithm moves on to check the next excited ion.

When the algorithm is complete, every excited ion has either emitted a photon, transferred energy to a nearby ion, or remained in the same state.

Chapter 3 – Previous Research

This chapter discusses previous research in the areas of energy transfer upconversion and computer modeling and visualization of crystal structures. Both areas have been investigated for years. However, computer modeling of energy transfer processes within a crystal is a novel combination of the two areas.

3.1 ETU Research

Traditionally, physicists have studied energy transfer upconversion through decay curve analysis. The crystal in question is often pumped full of photons and the output of photons over time is recorded. From these experiments one is able to see that upconversion has an adverse effect on the output of a crystal and can therefore disrupt a smooth decay curve. These experiments do not warrant results that provide explanations regarding the energy transfer within the crystal itself, but merely provide evidence that ETU has an effect on the output.

Markus Pollnau discusses the need for the knowledge of the fraction of ions that can participate in ETU in order to test models of ion distribution in the crystal [5]. If the distribution of impurity ions within the crystal lattice is non-random, then a knowledge of this fraction of ions can supply the needed information to derive the “critical distance”, or distance outside of which ETU is not likely to occur. He discusses the importance of ETU on a non-random distribution of impurity ions. His findings reflect a method for determining the fraction of ions in a system that can participate in energy transfer upconversion. However, when using etVisual, the user defines this fraction, because all

doped ions have the possibility of participating in ETU.

There are two algorithms associated with etVisual (the energy transfer algorithm and the non-random doping algorithm). The non-random doping algorithm was extracted from Barbosa-Garcia and Struck[1], while the energy transfer algorithm was created using traditional energy transfer principles for the system we constructed.

A fairly thorough search for computer models of non-radiative energy transfer produced no results. After personal communications with Markus Pollnau, he corroborates our belief that no such computer model has ever been published.

3.2 Visualization of Crystals

There have been numerous visualizations of crystal formations created, each for a different purpose. Some visualizations strive to make conceptual understanding clearer, as the one in this paper does. Others are created to simply show an aspect of crystals that is easy to express mathematically but difficult to "see" in that way.

This latter visualization type is the kind Mehta, Hazzard, and Machiraju describe[3]. In their paper, they define a series of techniques for detecting and visualizing defects in lattice structures. The purpose of their investigation was to demonstrate the utility of using a visualization to determine the nature and position of defects with lattice structures. The datasets were constructed using Molecular Dynamics simulations. The authors feel that being able to visualize defects has enhanced their detection for physicists, and so their utility was demonstrated.

Slavin, et. al. [7] demonstrated methods for visualizing topological defects in liquid crystals. Once again, the purpose of the visualization was to make defect detection

simpler. This system of visualizations is very complex, far beyond the scope of this study, with no practical application to the energy transfer investigated in this thesis.

Tudisco [8] visualized the Nd:YAG crystal for the purpose of studying relative positions of nearby ions for their impact on the crystal's spectroscopic properties. The main idea for the visualization in etVisual was based on Tudisco's work. The unit cells building up from a single unit cell and drawn with a border can be seen in his visualization of Nd:YAG.

Several generic visualization tools are available that could have been capable of displaying a crystal. The first we looked at was IBM's Open Visualization Data Explorer (OpenDX) tool [15]. This tool takes input in the form of points on a grid. Properties of these points, such as color, can be modified. This system seemed ideal for the visualization needed for this study.

Another such visualization tool we looked at was MayaVi [16]. MayaVi uses the Visualization Toolkit (VTK), an open source system for data visualization and image processing. MayaVi works overtime of the Python programming language and is a powerful tool for displaying data points. Again, with some altering, MayaVi would have been a useful tool with which to create the visualization found in etVisual.

Due to the simplicity of the visualization needed, it would have been outside the scope of this work to learn to use either of these visualization tools. OpenGL allows for advanced graphics methods that we wished to become more familiar with before pursuing computer graphics in the future. Creating the visualization from scratch allowed more practice with the OpenGL API and so reinforced several programming tactics and

methods. This is the reason that, although other visualization tools exist, the authors chose to create the program rather than use an application to do it.

Chapter 4 – Programming etVisual

This chapter describes etVisual in detail the program, designed to produce a visualization of energy transfer in a crystal. The chapter goes into detail about several components of the program, including the code used to write it. The chapter also discusses the input/output of the program, and some problems that arose during the conception of etVisual. These programs include generating very large random numbers, and producing a visualization using the OpenGL API while coding a graphical user interface in FLTK.

4.1 etVisual

etVisual is a computer program designed to simulate the firing of a laser at a crystal. The program measures both the non-radiative energy transfer between ions and the energy output of the crystal. These measurements occur after the crystal is doped by a percentage designated by the user. Only the doped ions are focused on for the visualization because the program is designed to study the upconversion process and how it effects the photon emission of a crystal. We focus only on the impurity (doped) ions because only they will absorb and transfer energy.

The purpose of the program is to give physicists a tool by which to compare experimental data to analytical data and also to visualize the processes of energy transfer and photon emission. etVisual can also be seen as a teaching tool that may accompany experimentation. However, it is important to note that efficiency was not a primary concern during the design and building of etVisual. The program manages large datasets,

created on the fly. These datasets are built from parameters input to the program by the user and many operations are performed on them. These operations include calculating distances between several ions and traversing the large datasets of ion information. Both of these operations can become slow as the datasets become larger.

etVisual was programmed using C++, FLTK, and OpenGL. FLTK and OpenGL are Application Programming Interfaces (APIs). An API is a source code library that provides services outside a programming language's normal capabilities. OpenGL (Open Graphics Library) is a software interface to graphics hardware [OPENGLSOURCE] OpenGL provides the functionality to create low level geometric primitives (points, lines, and polygons). Separate libraries have been built on top of OpenGL, providing further functionality. OpenGL's utility library (GLU) comes standard with OpenGL releases, and provides higher level functionality for creating models. The OpenGL Utility Toolkit (GLUT) provides further functionality, allowing for the creation of three-dimensional models and advanced lighting techniques.

FLTK (Fast Light Toolkit) is a GUI (Graphical User Interface) creation tool. It uses the idea of widgets to create attractive and easy-to-use GUIs [FLTK SOURCE]. FLTK is small and modular, and works both as a statically linked library and a stored library. etVisual encompasses an FLTK window that contains a smaller OpenGL window within it (Section 4.1.5).

4.1.1 User Interface

The user interface and visualization of etVisual can be seen in Figure 7. The GUI appears on the left side of the application window, where the visualization portion

(crystal window) appears on the right side. The GUI is split into two parts: the crystal definition and the animation control, as seen in the figure. In the crystal definition section of the GUI, the user specifies the parameters used to define the crystal structure. The program requires input for the position of the base points in the primary unit cell. The user is also able to input the structure of the crystal (body center, face center, etc.), the number of unit cells needed, the percentage of ions to dope, the percent of doped ions that are excited (absorb a photon), the K_{Ratio}, the type of doped ion distribution (random vs. non-random), and the lifetime of one of the ions of the crystal. All of these parameters are input through the user crystal definition section of etVisual, on the left side of the window, as shown in Figure (7).

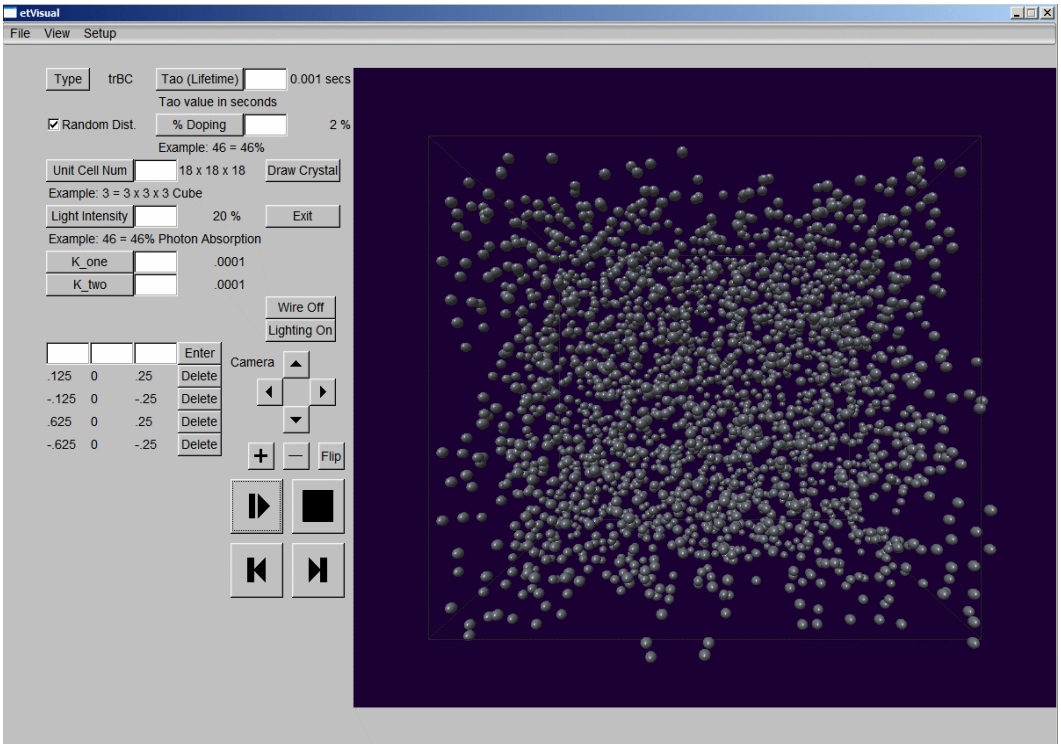

Figure 7: The full window, with a visualized crystal in the window to the right. This is an image of the crystal before the animation button in the bottom right of the crystal formation window has been clicked.

Once the user has entered the appropriate input and clicks the Draw Crystal button, the program draws the crystal's doped ions to the screen. The user may then begin an animation of energy transfer and photon emission in the crystal by clicking the play/pause button, change the camera view by using the camera direction buttons, speed up and slow down the animation with the playback speed buttons, or toggle wire frame and lighting on and off with the wire and lighting buttons. The user may make changes to the crystal's parameters in the crystal definition area of the window. Clicking the Draw Crystal button again will make the visualization window reflect any changes made to the crystal definition. With this feature, it is easy to switch between different crystals.

The animation of energy transfer will run the energy transfer algorithm and produced the program's output (Section 4.1.3). As the datasets become large, more ions are drawn on the screen and their size may shrink, and so slight changes in color may be difficult for a user to detect. To prevent this, the lighting may be turned off or the wire frame may be turned on. Both of these options will produce fewer shadows on screen and more clearly define the borders of the ions. Turing the lighting off will create a visualization where color is brightly defined and no longer dependent upon how bright the light shining on the ion is. However, enabling lighting clarifies depth and allows the user to more easily see which ions are closer and which are farther from it. When lighting is disabled, this perception of depth disappears, and so it is generally more beneficial to leave lighting on when viewing the visualization.

4.1.2 Input/Output

Another feature of etVisual is file loading and saving. When a crystal has been

defined in the crystal definition portion of the window, the user can select File/Save Crystal to save the definition to a file to be loaded later. The format of one of these files is simple to recreate. Note that all fields are separated by tab characters, but any white space separator (of any length) will work. The format is as follows:

```
<Lattice Type>  <Tao Value>     <Unit Cell Number>      <Dope Percentage>      <Light Intensity>
<Point 1 X>     <Point 1 Y>     <Point 1 Z>
<Point 2 X>     <Point 2 Y>     <Point 2 Z>
...
```

For example, the Nd:YAG crystal used in this study is stored in a file as follows:

```
2       0.001   12      2       10
.125    0       .25
-.125   0       -.25
.625    0       .25
-.625   0       -.25
```

In this example, the lattice type is represented by 2 (which translates to trBC), a lifetime of .001 seconds, a 12 by 12 by 12 unit cell number, doped at 2% with 10% laser excitation. There are four base points to the Nd:YAG crystal formation. This simple format makes it easy to create the files outside of the program, as well as being able to load them and save them easily. When etVisual saves files in this format, each field is tab delimited.

When a file is read in, the GUI is updated to reflect the current crystal settings. If a user wishes to make updates to a file that he has already created using etVisual, but doesn't want to manually change the file, the user may load the file into the program and make his changes. The user can then save the new parameters as a new file, or overwrite the old file. In this manner, is it easy to quickly create several crystal files.

FLTK contains a file chooser class which is used to define an FLTK File Chooser.

FLTK's File Chooser is a separate window that opens on top of the FLTK window. This window is shown in Figure 8. This chooser includes a file navigator, a preview pane (the right side containing question mark in Figure 8), an area to navigate favorite files and directories (the "Favorites" drop down menu), and a file name output.

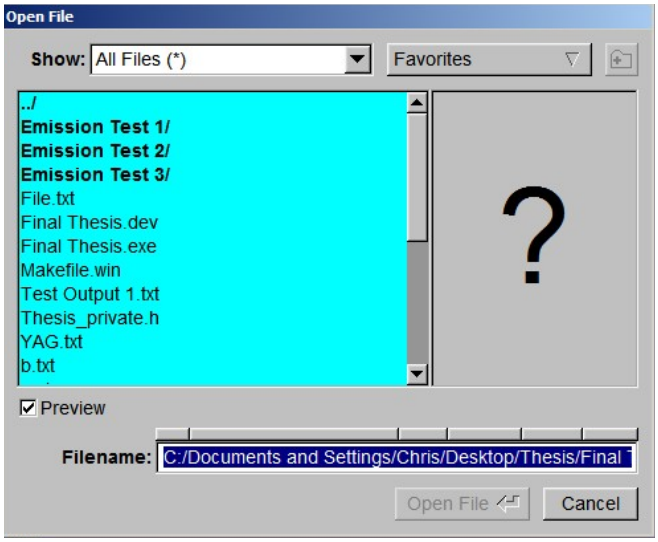

Figure 8: FLTK's File Chooser class defines the above File Chooser. The chooser includes a file navigator, a preview pane (the right side), an area to navigate favorite files and directories (the "Favorites" drop down menu), and a file name output.

When instantiated, a File Chooser object will appear (after its `show()` method is called) as a file navigation window, which is depicted in Figure 8. The file chooser constructor has four parameters: the default directory the file chooser starts in, the file type that the chooser works with (in etVisual, all types are supported in the file chooser), the type of file chooser (the save file chooser is of type `CREATE` whereas the load file chooser is of type `SINGLE`, meaning a single file may be selected), and the name that appears in the title bar of the file chooser. In etVisual, the save file chooser instantiation

looks like

```
SaveFileChooser = new Fl_File_Chooser(".", "*",
                        Fl_File_Chooser::CREATE, "Save File");
```

This file chooser declaration and the definition that follows it appear in the callback function for the “Save Crystal” menu option. The file chooser is created and an `FL::Wait()` is called while the file chooser is still shown to force the program to wait and not take more input.

The program outputs photon emission data in a text file (extension .txt). This data is simply a table of times and energy contained within the crystal at the designated time. The program records this data by counting the number of ions that have been emitted. At each time increment, etVisual outputs the current time and the photons that have not yet been emitted. These output fields are tab delimited for easy conversion into Microsoft Excel or another spreadsheet program.

4.1.3 User Interface Code

The code for the GUI defines its look and operation, and is written using calls to FLTK. FLTK defines GUIs by allowing the programmer to define widgets. Widgets are simple components of the GUI. Callback functions may be assigned to widgets to give them functionality. A callback function is a one defined by the programmer that is run when the state of a widget is changed. For instance, if the widget in question is a button that is “pressed” when the user clicks on it, then the callback function that has been assigned to that button is called when the state of the button is changed.

All widget types are objects that inherit from the `Fl_Widget` superclass. In fact,

FLTK was built using an extensive hierarchical system of inheritance. The FLTK inheritance hierarchy is based on the superclass `Fl_Widget`, which is inherited by all different widget types, including buttons, boxes, and labels. For instance, if a programmer uses a floating point input box (a box that allows for the input of floating point values), he would create an instance of the `Fl_Float_Input` class. This class inherits from the `Fl_Input` class, which inherits from the `Fl_Widget` class. This inheritance hierarchy makes FLTK easy to learn, as all widgets have similar constructors and methods.

When programming using FLTK, a GUI is produced by creating an instance of the class `Fl_Window`. The `begin()` and `end()` methods of the Fl_Window class are used to define the scope of the GUI code. Any widgets defined after `begin()` and before `end()` will appear in the FLTK window when `Fl_Window.show()` is called.

Placing buttons in FLTK is done through a coordinate system that starts at 0,0 in the top left of the GUI window and continues to the window's width and height. When a widget is declared, the constructor for the widget's class takes an x- and y- coordinate, and a width and height as parameters. This defines the widget's area in the GUI. For some widgets, the constructor also takes a string for the widget's label. For instance, the code to create the Enter button appears below. Figure 9 shows the button as it appears on the GUI.

```
1     EnterButton = new Fl_Button(200, 250, 50, 25, "Enter");
2     EnterButton -> type(FL_NORMAL_BUTTON);
3     EnterButton -> box(FL_UP_BOX);
4     EnterButton -> shortcut(FL_Enter);
5     EnterButton -> callback(fillPoints);
```

```
6     Point_InputX = new Fl_Input(50, 350, 50, 25, "");
7     Point_InputY = new Fl_Input(100, 350, 50, 25, "");
8     Point_InputZ = new Fl_Input(150, 350, 50, 25, "");
```

Figure 9: The Enter button as it appears in etVisual's GUI.

Line 1 of the code defines the constructor for the Fl_Button class, which takes five arguments. The first two arguments are the x and y coordinates of the top left corner of the rectangle that defines the button. The third and fourth arguments are the height and length of the rectangle, respectively. The final argument is the label of the button.

Line 2 defines the type of button. Other button types include radio buttons and check boxes, among others. Line 3 of the code segment defines the type of box. The enter button is an "up box", as it appears raised off of the screen. Line 4 describes a keystroke shortcut for the button. This means that pressing "Enter" on the keyboard is the same as clicking the enter button on the screen. Line 5 is the callback function associated with the Enter button's state change. When the button is pressed, the function `fillPoints()` is called. In etVisual, this method takes the values currently in the three input boxes seen in Figure 9 and creates a new point (creates a new instance of the "point" class, discussed later). Finally, the definition of the three input buttons seen in Figure 9 are the last three lines of the code segment above. These input boxes allow the user to input the coordinates of base points that will be entered when the Enter button is clicked. Notice that the constructor for the `Fl_Input` class is the same as the `Fl_Button` class. This demonstrates the usefulness of FLTK's inheritance hierarchy, as a programmer must only

remember the parameter needed for one constructor in order to make buttons of several different types.

4.1.4 The Visualization

When the Draw Crystal button is clicked, etVisual goes through a series of steps to produce a visualization of the crystal and draws it in the visualization pane on the right side of the program window. In order to accomplish this, every piece of information input by the user in the crystal definition portion of the GUI is stored. The base points are all made into `point` objects, where `point` is a C++ class that holds each ion's information, including its color (which is important during the animation), whether or not it can emit energy, whether or not it can absorb energy, and whether or not it is doped.

Once the base points are made into objects, they are duplicated according to the lattice type (BC, FC, tr, etc.) that was selected by the user. This new list of points is placed into yet another object, a `unitCell`. This is a class that contains information regarding a single unit cell, such as the list of base points that are contained therein, and the number of points in a single unit cell.

Once a unit cell is created, it is duplicated *n* times in each of the x-, y-, and z-directions, where *n* is the unit cell number defined by the user. etVisual keeps a three-dimensional unit cell array, where each object in the array is a single unit cell. This is the fashion in which all of the data regarding the ions inside the crystal are stored.

If the random flag is turned on, the program then randomly chooses a percentage of points to dope, which entails simply setting the ion's `isDoped` flag to `true`. For instance, if a random distribution of doping is chosen by the user, the program first

chooses a series of random integers that index a unit cell. This requires the acquisition of three separate random integers, one for each the x-, y-, and z-coordinates of the unit cell being selected. Finally a random point is selected from within the chosen unit cell. This process is repeated until the user-defined doping percentage has been met. If the random flag is turned off, then the program runs the non-random doping algorithm (Section 2.2.1). Once all of the ions have been doped, the program moves on to the next stage of the visualization.

etVisual uses the OpenGL API to draw the doped ions in the visualization section on the right side of the window. One aspect that was especially difficult to program was the visualization window. FLTK has the capability to work with OpenGL to create windows used specifically for the API's code. However, the desired effect for etVisual was to have an OpenGL window, or graphics window, inside of the larger FLTK window. Normally, an OpenGL window is initialized outside the FLTK window, and when the former is created, it appears as a second window, separate from the original. However, by defining a class that inherits from FLTK's `Fl_Gl_Window` class, which handles all OpenGL code, the graphics window can be created inside the original FLTK window. By creating an instance of the class, the programmer may use OpenGL function calls inside of the larger FLTK window, or simply create a separate OpenGL window. Inheriting from the `Fl_Gl_Window` class is my preferred method for infusing OpenGL with FLTK. By initializing the OpenGL window inside of the FLTK window, and calling the OpenGL's `show()` function (which produces the window on the screen), the OpenGL window is created inside of the larger FLTK window, as is seen in Figure 7 on the right side of the

program's window.

The `GlWindow` class is an object that must be programmed and must inherit from the `Fl_Gl_Window` class. It has several methods, including a `resize()` method to handle resizing the window if necessary, and a `draw()` method that contains all information the class needs to draw in the OpenGL window. `Fl_Gl_Window`'s `draw()` method is roughly the equivalent of OpenGL's `display()` method, as both are called automatically and anything the programmer wants to be drawn to the screen must be defined in these methods. This means that in the `GlWindow` class, all objects that are to be drawn on the OpenGL window must be created inside the `draw()` method. Below is a code segment demonstrating the programmer-defined `GlWindow` class.

```
class GlWindow : public Fl_Gl_Window{
private:
      void draw();
      void resize(int X, int Y, int height, int width);
      static void FLTKWindow :: Timeout (void* data);
public:
      MyWindow():Fl_Gl_Window();
}
```

In order to make the OpenGL window appear inside the GUI, it had to be initialized inside the `begin()` and `end()` methods of the FLTK window itself. An example of this appears in the code segment below.

```
Fl_Window FLTKWindow;
FLTKWindow.begin();
...
Fl_Gl_Window* GlWindow;
GlWindow = new Fl_Gl_Window(int X_position, int Y_position, int
                                  height, int width);
...
FLTKWindow.end();
```

The OpenGL window appears inside the FLTK window as an object in the

window (similar to a widget) at position `X_position`, `Y_position` and is `height` by `width` in size. The OpenGL window appears within the GUI because it is instantiated between `Fl_Window`'s `begin()` and `end()` methods. This method for instantiation creates a more useful application than if a separate OpenGL window appeared, as users are able to see a closer attachment between what is defined in the crystal definition segment of the GUI and what appears in the OpenGL window.

4.1.5 The Animation

Once the impurity ions of the crystal have been rendered in the visualization window, the user may select the Play/Pause button and play an animation of the energy transfer within the crystal. During the animation, ions are drawn with different colors, depending on their energy levels. An ion in its ground state is drawn in a dark gray color. Those ions that have been excited to E_1 are drawn in a deep red color. Finally, those ions that have been excited to E_2 by upconversion are drawn as a light green, and are ignored for the remainder of the animation (as the energy they have is lost due mainly to the emission of heat). Colors in the animation change slowly, to demonstrate the change in the ions' energy levels over time. These colors can be seen in Figure 10 below.

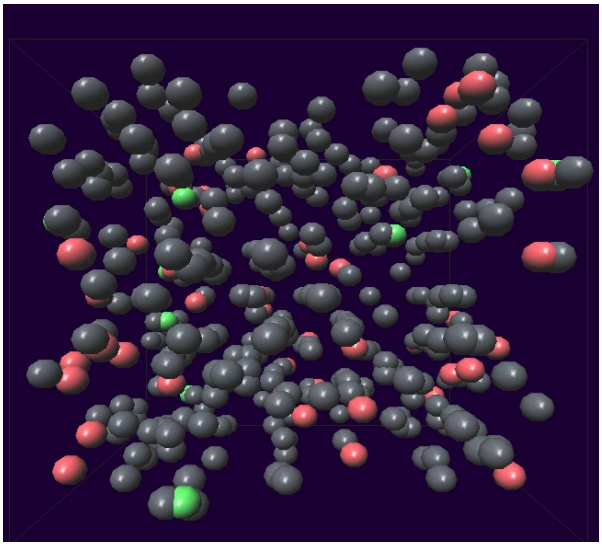

Figure 10.: An image demonstrating the ions in the crystal drawn at their various energy levels. The gray spheres are the ions in their ground state, while the red ions are excited to E_1 and the green ions have been excited to E_2.

A brief explanation of color choices seems necessary. The dark gray color was chosen for its association with cold and death. A typical winter's day in New England produces a “cold, gray” atmosphere. The gray ions are the “cold” ones, those not excited with energy. Red was chosen for its association with “heat”. Heat and energy are often interchangeable, and so those ions with energy are drawn red. Green was chosen for its contrast to red. Green and red are “opponent colors” [9], and so they stand out from each other clearly. Also, the visual vibrancy of both colors far outweighs that of the gray, “cold” ions. In other words, they both “jump” out at a viewer far more than the gray ions do.

The animation portion of the visualization is done using the `Timeout()` method of `Fl_Gl_Window`. The Timeout method is called every time a timeout is called within the program. The `Timeout()` method is passed the object that calls it (`void*` data in the

class definition above) and uses that data to create a new OpenGL window and draws the new window. This, in essence, creates a window that is constantly redrawing itself, making animation possible. It is also worth noting that FLTK's OpenGL package is double-buffered by default, and therefore the timeout works seamlessly, without needing to designate the window as double-buffered.

In order to achieve the animation, any changes that need to be made to the data occur within the timeout method. In this case, the status of which ions can and cannot absorb, and the ions' color data (all found within the `point` class) are updated in the timeout method according to the energy transfer algorithm described in section 2.4. The `Timeout()` method is shown below.

```
1     static void GlWindow:: Timeout(void *data) {
2          GlWindow *GraphicWindow = (GlWindow*)data;
3          UpdateData();
4          GraphicWindow->redraw();
5          Fl::repeat_timeout(1.0/60.0, Timeout, data);
6     }
```

This code segment is the `Timeout()` method found in the `Fl_Gl_Window` class. By adding a timeout recall to the `GlWindow` class, the `Timeout()` method is called at various intervals as determined by line 5. The `Fl::repeat_timeout()` method takes three parameters. The first is the length of time between calls. In this case, the timeout is called 60 times every second (1.0 seconds for every 60 calls). The second parameter is the timeout callback (the function acting as the timeout), and the third parameter is the arguments that the timeout callback takes.

Line 2 sets up a new OpenGL that is a copy of the one that calls `Timeout()`. This new window is the one that is actually redrawn. Line 3 is a function call to the

function that runs the energy transfer algorithm and updates the data accordingly. For instance, if the algorithm determines that ion *i* transfers energy to ion *j*, then ion *i*'s energy level is decremented and ion *j*'s energy level is incremented. Also, ion *i*'s and *j*'s colors are adjusted to reflect their new energy states. Line 4 redraws the OpenGL window.

By updating ion data 60 times in a second, or with a frequency of 60 Hz (the same as a television), a smooth animation is produced. The animation produces a visualization of the energy transfer inside of the crystal defined by the user. By showing upconversion through the change in ion colors, the animation can show a user just how many ions are affected by the process as well as how much energy is lost through heat which can significantly affect the crystal's photon emission.

4.2 Random Number Generation

A problem faced when programming etVisual was the generation of random numbers. The math library in C++ has a random number generator that can be accessed by calling the `rand()` method. However, this method returns a psuedorandom integer (signed) between 0 and `RAND_MAX`. `RAND_MAX` is typically set to be 32767 in decimal notation, because 32767 is exactly $2^{16} - 1$, or the largest number that an unsigned 2-byte integer can hold. However, etVisual requires random numbers that are much higher than `RAND_MAX`, such as in the algorithm described above. For this reason, we used Shapira's implementation of LeCuyer's 32-bit random number generator [2, 6]. This generator can produce random numbers between 0 and 1 or between 0 and `MAX`, where `MAX` is $2^{32} - 1$. Both functionalities were used at various points during the programming of etVisual. The generator was seeded using C++'s random number generator (seeded by `time(0)`), so

as to differ the output every time the program was run.

The random number generator is employed at several points in the program. For example, during the non-random doping algorithm, a random number is needed between 0 and 1 in order to find the next ion to dope. This could have been accomplished by calling `rand()` and dividing the result by `RAND_MAX`, but this is still a one-to-one function and can only produce 32767 possible values between 0 and 1. In order to give uniform distribution of several times that many values between 0 and 1, the LeCuyer generator was used to produce a floating point value in that interval. Since there are several times more values produced by the LeCuyer generator than `rand()` may output, this method is preferential over using C++'s built in `rand()` method.

4.3 Performance Testing

Brief performance tests were performed on etVisual. For all of these tests, the Nd:YAG formation was used. For every test, the maximum unit cell number allowed by the program is 25 cells in any direction. These tests were performed on a Dell Inspiron 9400, with 2 GB RAM, and a 1.73 GHz Centrino Duo Core processor.

At 2% random doping, etVisual was able to draw the crystal in under one second. However, as the unit cell numbers increased past 16, the GUI portion of the window began slowing down significantly. It would take increasingly more time for a cursor to appear in an input box after clicking on it. At 15625 unit cells (25 x 25 x 25, or the maximum), it would take nearly a second for a cursor to appear in the window. This result is due to OpenGL's drawing of the ions 60 times a second.

When the doping percentage was increased to 5%, The GUI portion of the

window began showing significant slowdown at 1000 unit cells. Once 15625 unit cells were reached, the slow down was several seconds. Drawing the crystal still took less than a second, until 8000 unit cells were reached. At the 15625 unit cell mark, the program took several seconds to draw the doped ions.

The non-random doping algorithm is very time consuming in general. At 2% doping of the Nd:YAG formation, it took etVisual 5 seconds to dope and draw 64 unit cells worth of ions. When the unit cell number was increased to 125, the program took 30.5 seconds to perform the task. As a final test at 2% doping, doping 343 unit cells was attempted. This process took etVisual 10 minutes and 18.11 seconds. Because of the small number of ions present on the screen, the GUI portion of the window did not display any slow down for any of the non-random tests.

4.4 Shortcomings of etVisual

As mentioned earlier, runtime efficiency was not the primary concern when programming etVisual. For this reason, the program has several improvements that can, and should, be made to it before release for use by other physicists.

The algorithms in etVisual are both based on distances between ions in the crystal. Measuring these distances requires cycling through all of the doped ions and measuring each one's distance to every other doped ion. In a worst case scenario (where the doping percentage is set to be 100%) this is $O(n^2)$, where n is the number of total points defining the crystal. This number n could easily reach into the tens of thousands, making the time taken by the distance measurement calculations very high.

One aspect of crystal formation is the idea of symmetry. Every unit cell defining

the crystal's structure is identical to every other cell, and so the distances between all of the points in the i^{th} and j^{th} unit cells are all the same as the distances between the points in the $(i + 1)^{st}$ and $(j + 1)^{st}$ unit cells. This symmetry between the unit cells could significantly reduce the calculation of the distances between ions. This idea has already been employed when the minimum distance must be calculated, as only the first two unit cells in any direction are taken into account. By only taking the first neighboring unit cells into consideration, every possible minimum distance is measured. This technique can be adapted, with some care and cleverness, to measuring distances as a whole.

An unfortunate trade-off when using this technique is that, in many cases, these distances will need to be stored. Although they will not need to be stored for every ion in every unit cell, a data structure will be needed to keep track of every possible distance and to which pair of ions in a unit cell those distances belong. Although this is a small price to pay for vastly increased speed and efficiency, it is a trade-off nonetheless.

Chapter 5 – Procedure and Results

This chapter will go into detail regarding the procedure used for the study and the results that were gathered. The results are analyzed and the impacts of doping percentage, light intensity (initial excitation percentage), ion doping distribution (random vs. non-random), and ETU are discussed.

5.1 Testing etVisual's Accuracy

Before tests could be run, we had to confirm that etVisual's output was consistent with the theoretical output of a crystal. For this, we ran a trial using a random ion distribution, doped at 5%, with 10% light intensity, and there was no energy transfer. The result is shown in Figure 11 and Figure 12 below.

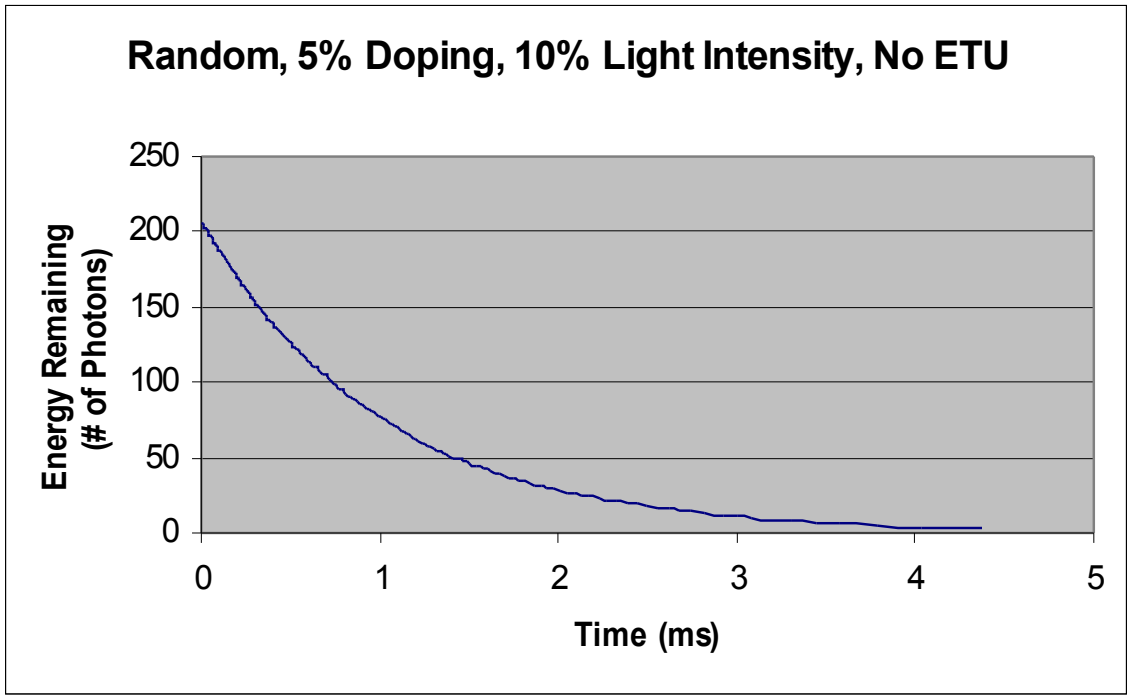

Figure 11: Graph depicting the output of an Nd:YAG crystal, randomly doped at 5% with 10% light intensity.

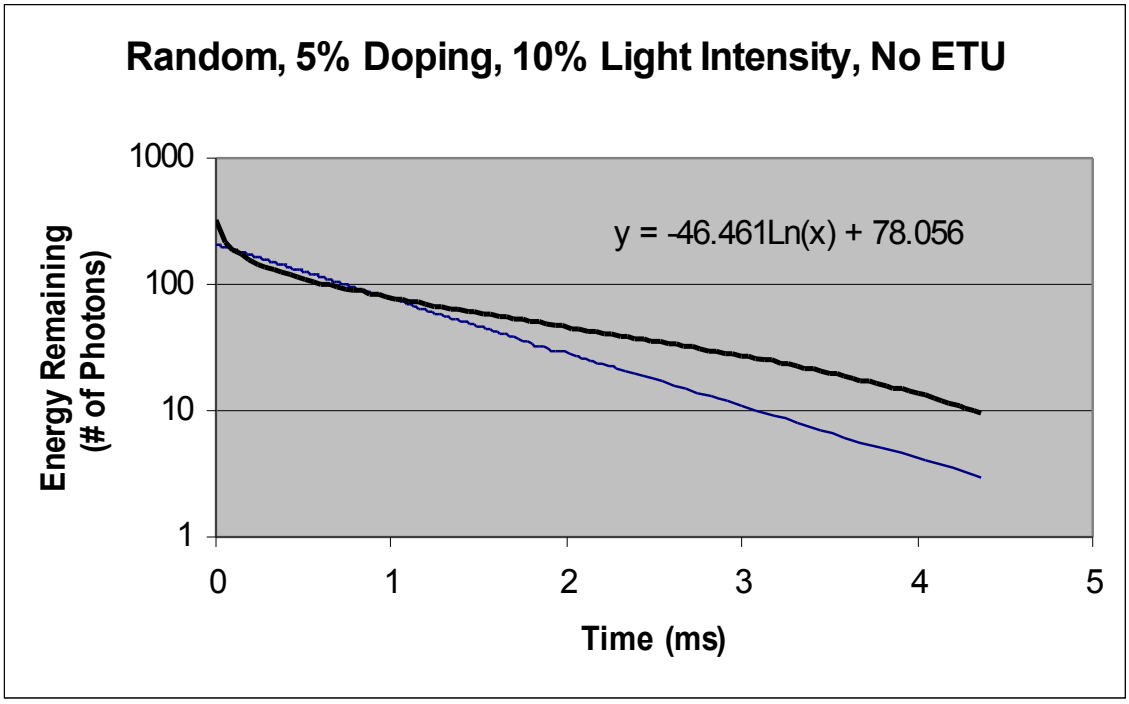

Figure 12: The Nd:YAG Output depicted in Figure 11, drawn on a logarithmic scale to demonstrate the exponentially decaying output. The bold line represents the logarithmic best fit line.

Figures 11 and 12 depict the energy output of a Nd:YAG crystal, randomly doped at 5%, with a light intensity of 10%. It is important to note the exponential decay behavior of the crystal's output. Figure 12 depicts the output of the crystal drawn on a logarithmic scale with a logarithmic trendline and equation added to the graph. The line appears to have a constant slope. The slope of the logarithmic trendline is -46.46. This exponential decay is consistent with the expected theoretical output of this crystal.

Now that we have demonstrated the etVisual is capable of producing a standard decay curve by simulating photon emission (with no energy transfer or upconversion), we are able to run the trials to meet the goals of the work.

5.2 Testing

A series of tests were performed using etVisual to output decay information of the crystal formations used. For each test run, 60 trials were taken and the mean of the resultant values (of energy output per unit time) were taken. Each trial consisted of randomly exciting a percentage of the ions in the crystal and then allowing them to decay slowly, measuring the output over time. The time interval (Δt) was 0.00001 seconds and the lifetime of the ions remained 0.001 seconds for each trial.

For each test, the formation for the Nd:YAG crystal was used. This formation is defined by yttrium atoms setup as ±(⅛, 0, ¼), ±(⅝, 0, ¼) trBC, for a total of 24 yttrium atoms in a unit cell.

Data was collected in the following way. An array was defined in which each index in the array represented a possible excitation quantity in the crystal. The index corresponding to the energy still in the crystal (ions still excited) each time interval was incremented by one. In other words, if, at the end of a time interval there were 18 photons that have not been emitted yet, then the array's 18th index is incremented. When the algorithm has finished, the array holds all of the output information of the 60 trials (how many time intervals the crystal held each level of energy). This data is then converted into an energy output vs. time curve. The curve of each test run is below.

For each graph, the time scale remains relatively unchanged (there were several abnormalities in the data that were likely the result of low sample sizes, as is discussed later) and appears along the X-axis. This axis is measured in milliseconds. We will analyze each test in order to meet the second aim of our study: to study the effect of

upconversion on the energy output of a crystal. The Y-axis reports the number of ions in the crystal that are still excited at the target energy level (level 1).

5.2.1 Testing Random Distributions

In order to compare the output of different crystal structures, a metric for measuring efficiency of the output is necessary. The metric we chose to use is the normalized area underneath the emission curve. This means that we normalized the output, meaning that at time 0 all emission curves started at 1, and measured the area underneath the curves. Figure 13 shows the results of the series of tests comparing various doping percentages and light intensities for random distributions of dopant ions.

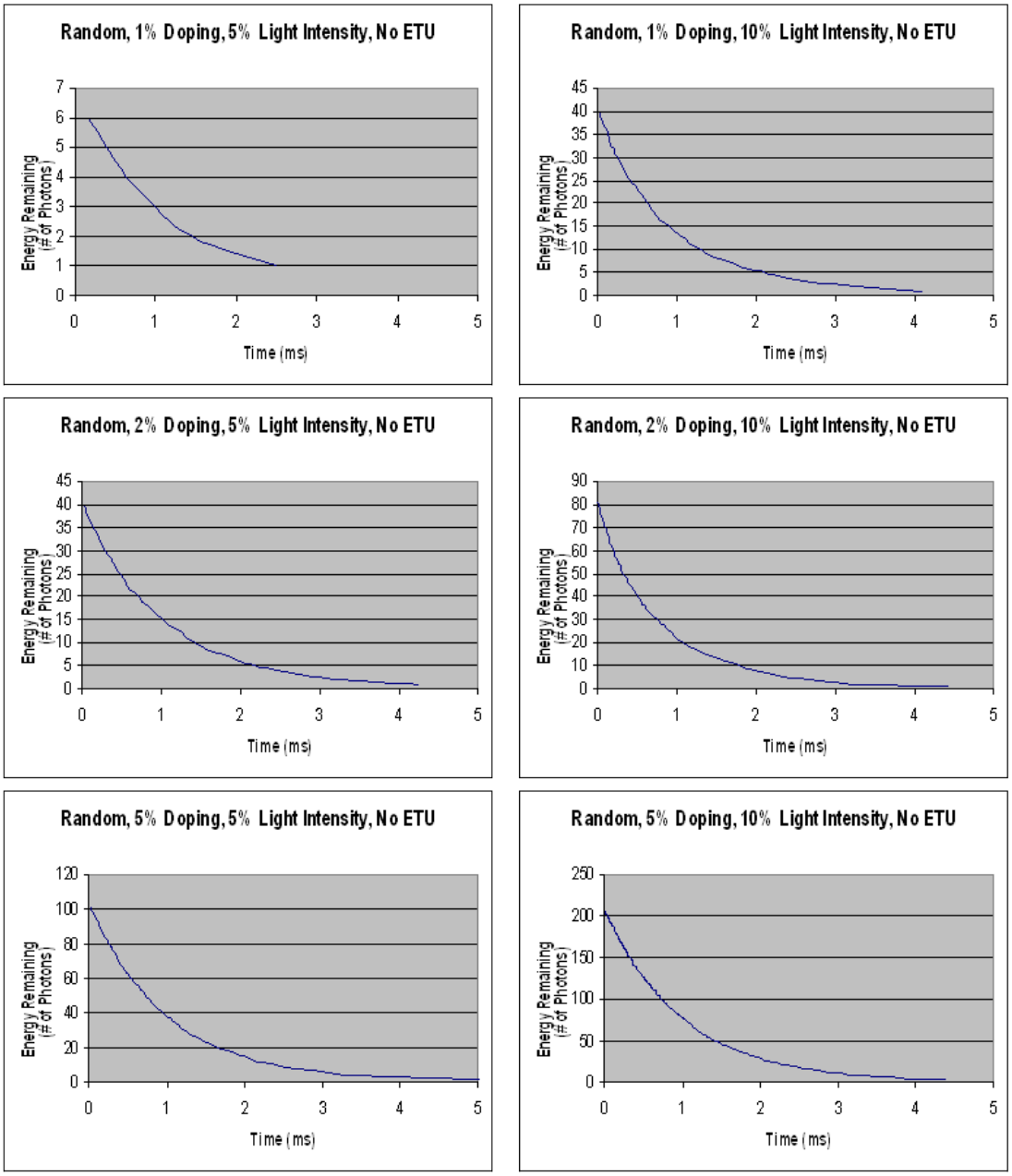

Figure 13: A series of graphs comparing 1,2, and 5% dopings with 5 and 10% light intensities. Energy transfer upconversion is not present in these tests.

Analysis of Figure 13 yields some interesting observations. The most striking observation is that the shape of the graphs do not change, and the amount of time to fully emit the crystal's ions does not change either. Each crystal emission took just over 4

milliseconds to emit, no matter how many photons were absorbed at time 0. There does exist a slight abnormality in the data, seen in the case with 1% doping and 5% light intensity. However, we believe this abnormality to be the result of too small of a sample size, as 5% of 1% of the total number of ions in the crystal is a small number (since there are 41472 possible doping sites, this number comes out to be 20.76). Calculating the normalized area underneath the curves yields the following results:

Doping Percentage	Light Intensity	Normalized Area
1%	5%	1.03 ms
2%	5%	0.98 ms
5%	5%	1.00 ms
1%	10%	0.97 ms
2%	10%	0.78 ms
5%	10%	1.00 ms

These results (without the anomaly found in the 1% doping, 10% light intensity value) describe an expected outcome. Since the areas under the normalized curve of those tests without ETU are simply the area under the curve of $e^{-t/\tau}$, as there are no external factors working on the ions in the crystal, it is not surprising that this values all fall around 1.00 ms, or the lifetime of the ions.

A second test of random distributions is found in Figure 14. This second set of graphs displays the results of a test comparing the same doping percentages and light intensities as Figure 13 above, but with ETU. For the K_{one} and K_{two} values, 0.00001 was used as it was demonstrated, through trials, to provide enough ETU to cause an impact but not so much that the decay curve was disrupted. Picking these K values allowed us to witness the impact of ETU on the emission of a crystal.

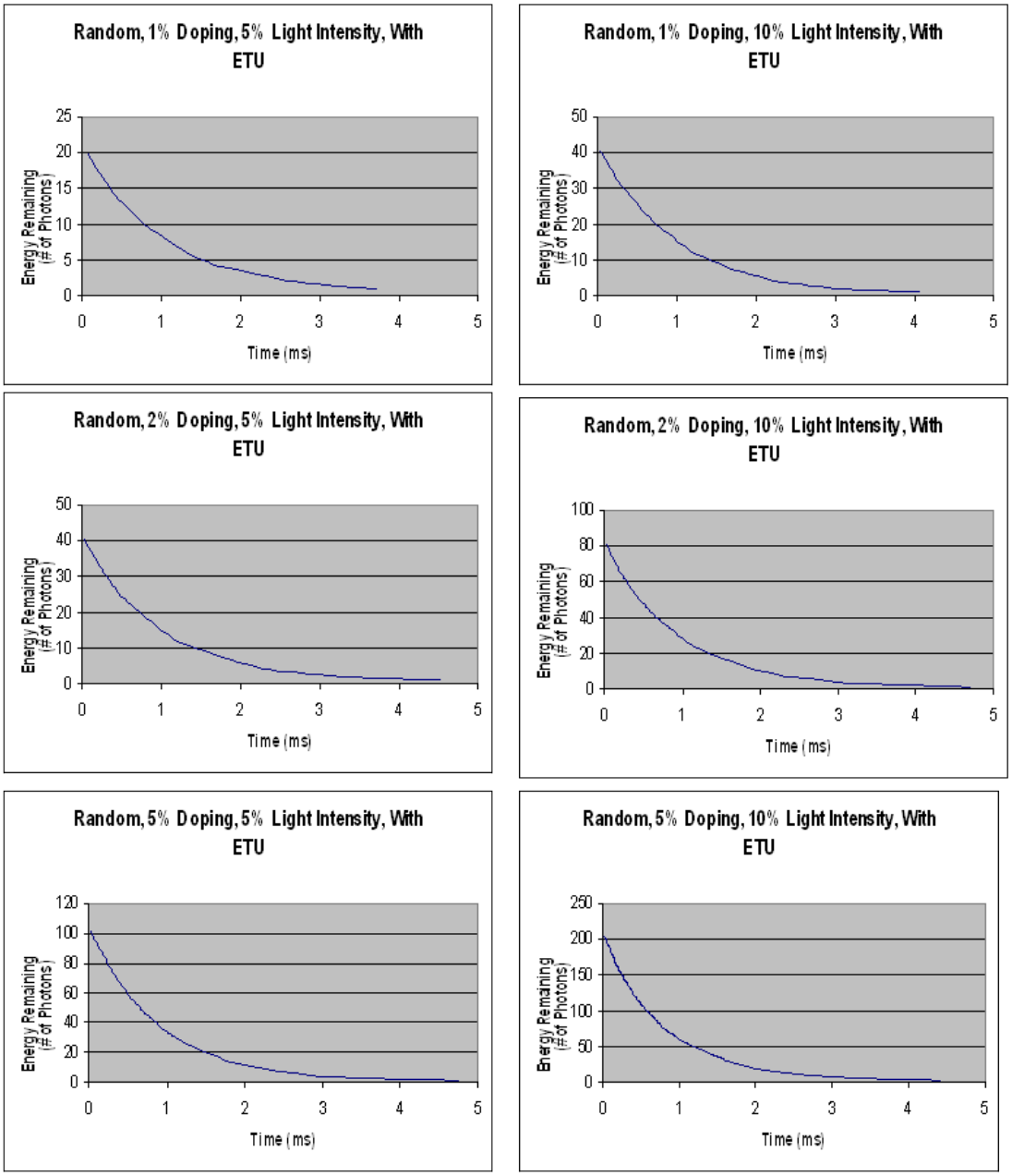

Figure 14: A series of graphs comparing 1,2, and 5% dopings with 5 and 10% light intensities. Energy transfer upconversion is present in these tests.

Careful analysis of the graphs in Figure 14 conveys the following observations. Once again, decay took the same amount of time (just over 4 milliseconds) in all cases. As to be expected, the graphs on the right start at roughly twice the level the ones on the

left do. What is very interesting about this second sampling of outputs is that the graph that had data abnormalities (1% doping at 5% light intensity) in Figure 14 above does not seem to in this group.

Once again, the areas under the curves were calculated. They appear in the table below:

Doping Percentage	Light Intensity	Normalized Area
1%	5%	1.03 ms
2%	5%	0.97 ms
5%	5%	0.90 ms
1%	10%	0.96 ms
2%	10%	0.92 ms
5%	10%	0.83 ms

The values in this table are on average below the comparable values in the previous table of results. In fact, looking at the parameters with the clearest results (5% doping with 10% light intensity), we can see from Figure 15 that ETU has a direct impact on the emission of this crystal.

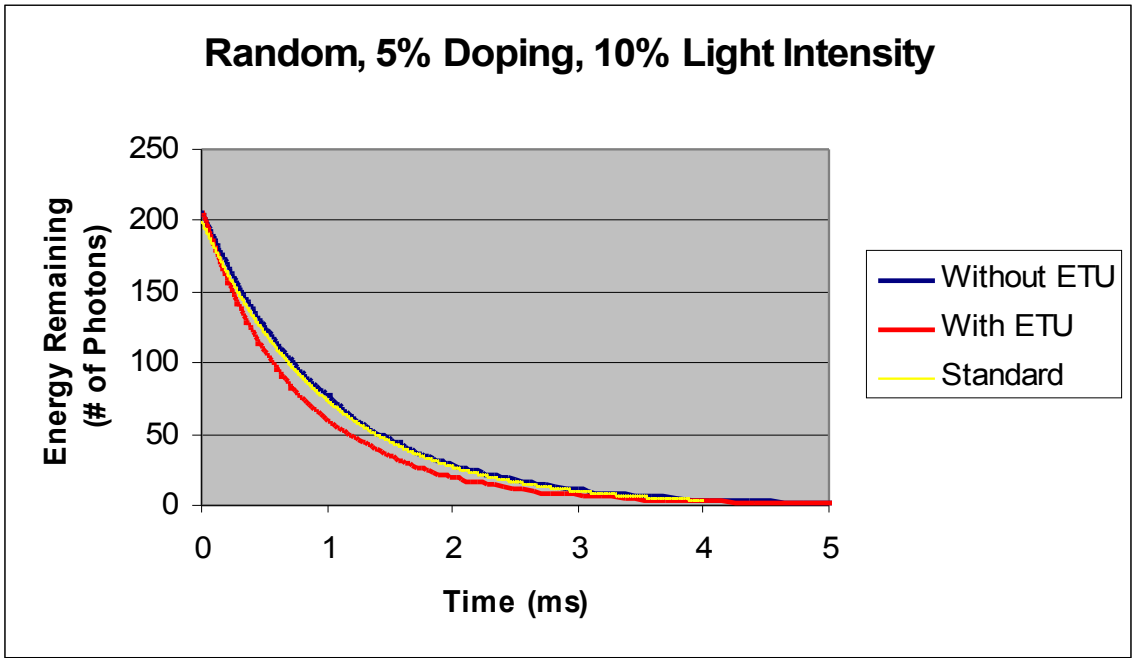

Figure 15: This graph shows a random doping distribution at 5% doping and 10% light intensity with both ETU turned on and turned off. The yellow line is the standard exponential decay curve, while the blue line is ETU turned off and the red line is ETU turned on.

In Figure 15, the yellow line represents the standard output with no external factors (the exponential decay curve). The blue line represents the results taken with ETU turned off and the red line represents the data taken when ETU were turned on. As can be seen in the figure, ETU has a dramatic impact on the decay of the crystal. In fact, the normalized area underneath the blue curve is 1.00 ms while it is 0.83 ms under the red curve, demonstrating a 17.66% reduction in the output of the crystal when ETU is turned on.

The lines in Figure 15 depict all three doping levels at 10% light intensity both with and without the ETU process impacting the results. These curves demonstrate an intuitive belief about ETU. Because excited ions now have two avenues by which to release energy (ETU or photon emission), then the ions that remain at our target energy

level of E_1 will decrease faster than those trials with no ETU. But what happens when ions are doped using a non-random ion distribution?

5.2.2 Testing Non-Random Distributions

The second series of tests that were performed show the effect of ETU on a non-random dopant distribution. Due to the nature of the small sample size necessary to run non-random testing, the light intensity was kept constant 10% during these tests in order to guarantee a large enough sample size. Due to the non-random distribution algorithm's time use, a smaller unit cell number (8) had to be used. Figure 16 demonstrates the relationships between non-random dopant distribution and energy transfer upconversion.

For the non-random distribution, the default values of 1 for α,1400 for the annealing temperature, and 1000 for γ. These values were chosen because they scale nicely and can be generalized for all ion types. Displayed in Figure 16 are the curves for both tests runs with ETU turned on and turned off. Both tests had 5% doping and 10% light intensity.

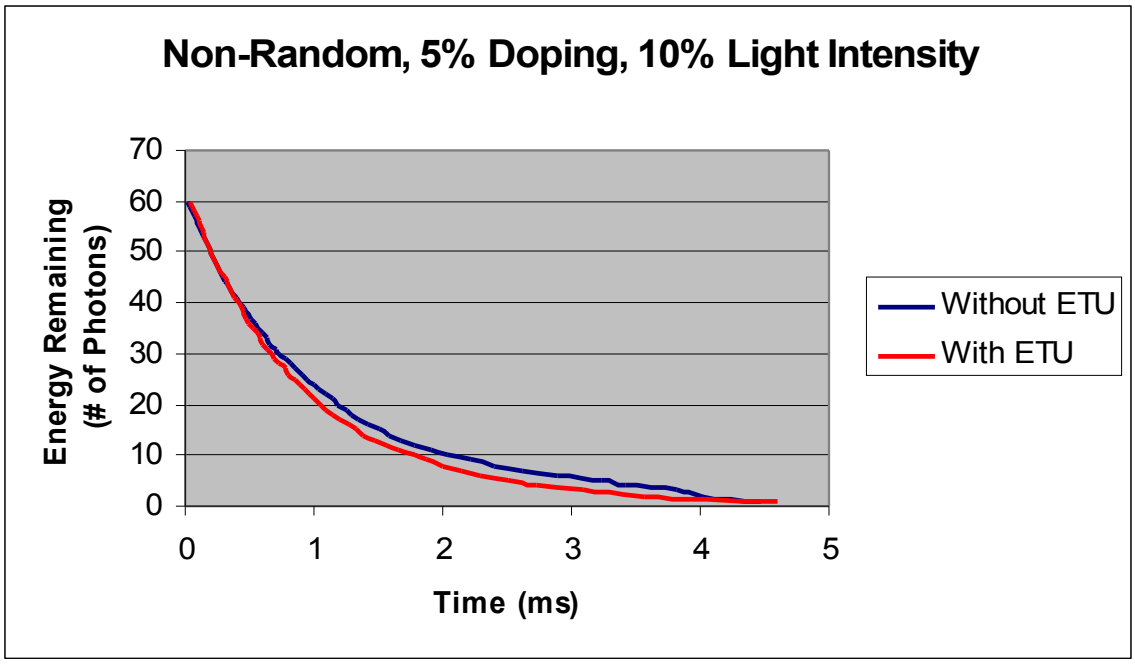

Figure 16: This graph shows a non-random doping distribution at 5% doping and 10% light intensity with both ETU turned on and turned off. The blue line is ETU turned off and the red line is ETU turned on.

At first glance, the relationships between the curves in Figure16 and those in Figure 15 above are very similar. It would appear that the presence of ETU creates a faster decay than if ETU was not present. This is in fact the case, as the normalized area under the Without ETU curve is 1.08 ms while the normalized area under the With ETU curve is 0.94 ms. This represents a 12.66% drop. This situation once again comes about because an ion has more ways in which to release energy in a system where ETU is present. All of this is true, but what effect does a non-random distribution of ions have in comparison to a random distribution?

In fact, with the particular non-random doped ion distribution used in the tests, the non-random distribution demonstrated a less severe drop-off in efficiency of the crystal output than the random distribution. Several factors may have contributed to this, as it

goes against intuition since energy transfer is based on the proximity of excited ions to other doped ones. These factors may have included a larger sample size for the random distribution, and sheer luck.

The final comparison performed was of a relatively low light intensity (10%) to a high light intensity (40%). For both tests, a random distribution at 5% doping was used. The resulting decay output is seen in Figure 17.

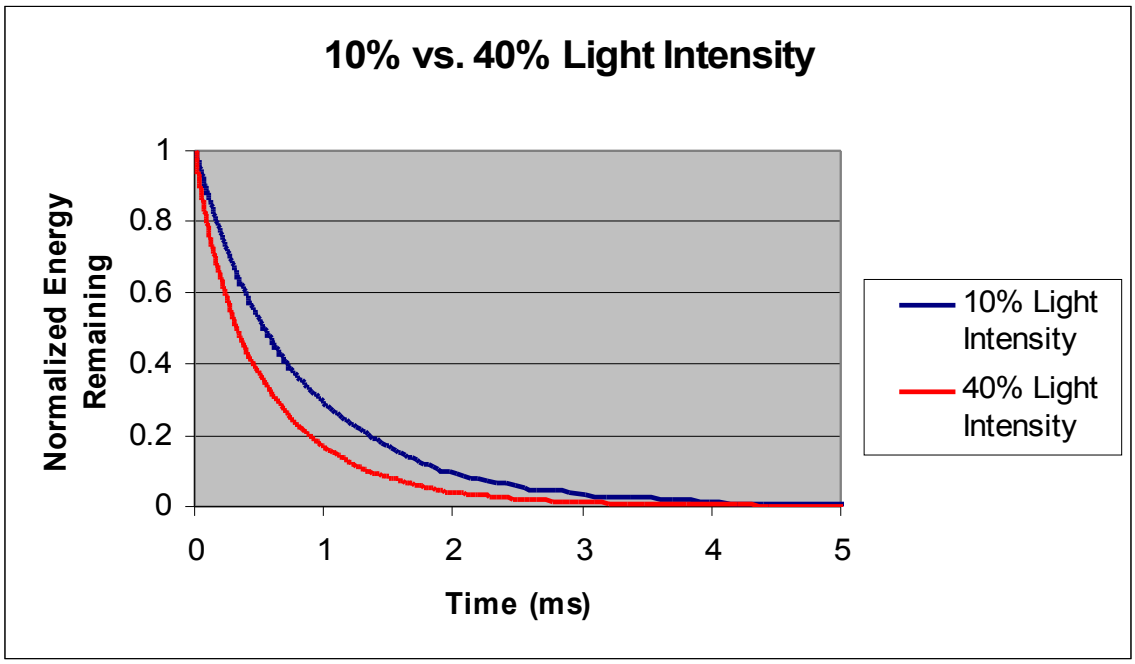

Figure 17: This graph shows a non-random doping distribution at 5% doping and ETU turned on. The blue line is 10% light intensity and the red line is 40% light intensity.

The data in Figure 17 demonstrate another intuitive idea about energy transfer upconversion. When more ions are initially excited, there is a much greater chance that, if energy transfer occurs, the ion being transferred to is already excited. This means that ETU will occur more often. This result is clear from the graph, as the normalized area under the 10% curve is 0.83 ms while the normalized area under the red curve is 0.55 ms.

This is a 33.75% drop, and the most significant effect of ETU of any of the tests run.

5.3 Visualization as a Tool

Another important result came out of this study. During the course of programming etVisual, we were able to do much of our debugging using the visualization that was already in place. For instance, when designing the energy transfer algorithm, it would have been nearly impossible to understand if the algorithm was working the way we wanted it to. However, because we could see the energy transfer as it took place, the algorithm was easily debugged. We could see when energy transfer was occurring in the algorithm because the change was reflecting in the fading colors of the ions in the visualization.

Another example of the utility of etVisual's visualization component came about when running tests. K values (K_{one} and K_{two}) were set based on the level of ETU they created. Too much ETU and the crystal would not emit enough photons, whereas too little ETU and the results would have been meaningless. Because the visualization allows the user to see the energy transfer happening, drawing the crystal to the screen was the ideal tool for choosing the correct K values for the ETU testing. One instance where this was particularly important came about when we ran a visualization and all of the upconversion was occurring on one side of the screen. It turned out that we had made K_{two} so high that the algorithm never needed to look beyond the first several dozen ions to find one to transfer energy to, thus causing all of the ions on one side of the screen to gain energy while the rest of the crystal was drained. Without the visualization component, these problems may have gone unnoticed.

Chapter 6 - Conclusions and Future Work

6.1 Conclusion

This study accomplished two aims. The first aim was to create a piece of software that would allow a user to study the energy transfer within a crystal. This was accomplished with etVisual, which allows for the definition of almost any conceivable crystal formation, as well as several other experimental parameters such as the lifetime of the ions and the initial excitation percentage.

With etVisual, our second aim was also accomplished. We studied how energy transfer upconversion affected the photon emission of a crystal. Through a series of several trials each, we were able to show evidence supporting the idea that ETU has an adverse effect on photon emission in crystals. This conclusion may severely impact laser production. The ions of a laser's crystal must emit photons from the same target energy level. If, as what happened in the trials, a significant percentage of the ions experience upconversion, the laser's efficiency suffers as fewer ions are losing energy through photon emission from the target level. An obvious goal of laser production is to create the most efficient laser possible, which means having all of the energy pumped into a crystal be released in the formation of a laser beam, and not in some other way.

Another conclusion is the fact that ETU has a greater effect on trials with a higher light intensity. This makes intuitive sense, as more excited ions in the crystal means more possible transfer sites and more sites that ETU may occur at.

Our software may benefit the future development of crystal doping formations

that allow for the maximum output efficiency. We are now able to directly compare the effects of doping methods, doping percentages, light intensities, and crystal formations on the energy output of a crystal. We foresee this software being used as a quick reference to ETU's effect when different crystal parameters are in place. In other words, it is much easier for a physicist to gather data on several different crystal formations by running several trials with etVisual than to obtain crystals with those formations and test their emissions in a laboratory setting.

6.2 Future Work

This study was done using the formation of the Nd:YAG crystal. Future work would involve using etVisual to study any of several other crystal formations, especially those commonly used in laser production and as lamp phosphors.

The program needs several improvements before release in the public domain. First, the interface must be updated to allow a user to input all of the information that was static throughout this study, such as the Δt value, as opposed to each trial using the same hard-coded value for the parameter. This would put total control in the user's hands. The visualization that the system displays can be improved by reporting more information, such as the current time interval and using a visualization method to highlight energy transference from one ion to another. Also, an implementation that allows for several output files for each run of the program to be produced will be created.

One physical process that was ignored in etVisual was the possible return of an ion at the second energy level to the first level, or the possible emission of a photon from an ion at the second energy level and return its to its ground state. If an ion releases

enough heat over a short enough period of time, or releases a photon, the ion may re-enter the pool of ions that may be excited. This possibility was ignored because its probability of occurrence is minuscule within our relatively small data sets. However, despite this small probability, the process will be incorporated into the program as future improvements are being made.

The code for etVisual will be made more efficient and cleaner overall. This must be done to facilitate future upgrades to the program by those who choose to undertake the task. Also, the algorithms must be made far more efficient. Although many tests similar in nature to the ones in this test are done using small doping percentages and light absorption percentages, it is conceivable that a user may wish to study a much larger crystal, and so the algorithms must be adjusted to handle large datasets quickly. This includes using the symmetry of the crystal to our advantage.

Another area of future research with etVisual is to compare the effect ETU has on both repulsive and attractive non-random ion distributions. For this study we focused only on attractive distributions. However, it would be interesting to see how the repulsive distribution's results differ from those of the attractive.

There are several parameters involved with energy transfer, many of which we kept constant through this study. Some of these parameters that can be experimented with include the K values and τ (the lifetime of an ion). Also, larger unit cell numbers can be experimented with, as our test only went up to 12 unit cells and the maximum that etVisual will allow is 25.

An interesting area of future work would be to investigate the impact that ETU

has on the emission of a crystal. From this study we have determined that etVisual can describe the impact that ETU has on the crystal's emission. However, more extensive studies should be done to determine to exactly what degree does ETU cause disruption in the emission of a crystal, as it is an overall poorly understood phenomenon.

Finally, a user study should be done to determine the usefulness of etVisual's visualization component in demonstrating the idea of energy transfer to individuals not necessarily in the physics community. Since we foresee this software being sued by physicists in crystal research to radically ease the tedium of their emission tests, the GUI and visualization must be accessible to these physicists. Only a true user study will determine the degree to which these components of etVisual ease the testing process.

References

1. Barbosa-Garcia, Oracio and Struck, Charles W. Monte Carlo treatment of the nonradiative energy transfer process of nonrandom placements of dopants in solids, *J. Chem. Phys*. 100 (6), March 15, 1994.
2. Lecuyer. Efficient and portable combined random number generators, *Communications of the ACM Archive,* Vol. 31, Issue 6 (June 1988) pp. 742-751.
3. Mehta, Sameep, Hazzard, Kaden, Machiraju, Raghu, Parthasarathy, Srinivasan,and Wilkins, John. Detection and Visualization of Anomalous Structures in Molecular Dynamics Simulation Data, *IEEE Visualization 2004*, October 10-15, Austin, Texas, pp. 465-472.
4. Pollnau, Markus (personal communications)
5. Pollnau, Markus. Decorrelation of luminescent decay in energy-transfer upconversion, *Journal of Alloys and Compounds* 341 (2002), pp. 51 – 55.
6. Shapira, Andrew (personal communications)
7. Slavin, Vadin A., Laidla, David H., Pelcovits, Robert, Zhang, Song, Loriot, George, Callan, Jones, Andrew. Visualization of Topological Defects in nematic Liquid Crystals Using Streamtubes, Streamsurfaces, and Ellipsoids. *Proceedings of the conference on Visualization '04*, October 2004, IEEE Computer Society, p. 598.21.
8. Tudisco, Jeremy. Graphical and statistical modeling of a doped Yttrium aluminum garnet (Y3AAI2(AIO4)3) crystal, *Journal of Computing Sciences in Colleges*,

Vol. 21, Issue 6 (June 2006), Consortium for Computing Sciences in Colleges, p. 301.

9. Ware, Colin, *Information Visualization: Perception for Design, Second Edition*, Morgan Kauffman Publishers (2004), pp. 97-144.
10. Woo, Mason, Neider, Jackie, and Davis, Tom. *OpenGL Programming Guide, Second Edition*, Addison Wesley (1997), Silicon Graphics, Inc.
11. Wyckoff, Ralph W. G. *Crystal Structures*, Vol 1., John Wiley & Sons, New York, London, 1963.
12. Wyckoff, Ralph W. G. *Crystal Structures*, Vol 3., John Wiley & Sons, New York, London, 1966.
13. C++ Reference. http://cppreference.com. Last referenced: April 14, 2007.
14. FLTK. http://fltk.org/ Last referenced: April 5, 2007.
15. Open Visualization Data Explorer. http://www.research.ibm.com/dx/ Last referenced: 4/28/07
16. Data visualization tools for Linux. http://www-128.ibm.com/developerworks/linux/library/l-datavistools/index.html Last referenced: 4/28/07