7-2012

COMPREHENSIVE CALCULATION-BASED IMRT QA USING R&V DATA, TREATMENT RECORDS, AND A SECOND TREATMENT PLANNING SYSTEM

Jared D. Ohrt
The University of Texas Graduate School of Biomedical Sciences at Houston, jaredohrt@gmail.com

Peter Balter
The University of Texas Graduate School of Biomedical Sciences at Houston

Michael Gillin
The University of Texas Graduate School of Biomedical Sciences at Houston

Court Laurence
The University of Texas Graduate School of Biomedical Sciences at Houston

Issac Rosen
Methodist Hospital

See next page for additional authors

COMPREHENSIVE CALCULATION-BASED IMRT QA USING R&V DATA, TREATMENT RECORDS, AND A SECOND TREATMENT PLANNING SYSTEM

By

Jared Dean Ohrt, B.S.

APPROVED:

Peter Balter, PhD
Supervisory Professor

Michael Gillin, PhD

Laurence Court, PhD

Isaac Rosen, PhD

Valen Johnson, PhD

APPROVED:

Dean, The University of Texas
Health Science Center at Houston
Graduate School of Biomedical Sciences

COMPREHENSIVE CALCULATION-BASED IMRT QA USING R&V DATA, TREATMENT RECORDS, AND A SECOND TREATMENT PLANNING SYSTEM

A

THESIS

Presented to the Faculty of

The University of Texas

Health Science Center at Houston

and

The University of Texas

M.D. Anderson Cancer Center

Graduate School of Biomedical Sciences

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

By

Jared Dean Ohrt, B.S.

Houston, TX

August, 2012

Acknowledgements

The first person I have to thank is my wife Andrea. Through the past two years of graduate school she has supported and encouraged me even when the stress, exhaustion, and time commitments made it very hard to do so. She has sacrificed a lot, and asked very little in return. The smartest thing I have ever done, or will ever do, was ask her to be my wife.

The second person that I must thank is my advisor, Dr. Peter Balter, for his guidance, dedication. Not only did he always find the time to discuss this work with me, but the exuberance he showed over the work and the results was contagious. I must also thank the rest of my committee for their ideas and insights: Dr. Michael Gillin, Dr. Laurence Court, Dr. Isaac Rosen, and Dr. Valen Johnson. Their contributions significantly improved the quality of this work.

I would also like to thank two former committee members that were not able to see this work to completion for their contributions: Dr. R. Allen White and Dr. Milos Vicic.

I must also like to thank Michael Kantor. He, along with Dr. Vicic, was a valuable source of knowledge, and without their help this work would not have been possible.

Another person I must thank is thank Dr. Rebecca Howell for the commissioning the treatment planning software which served as the verification treatment planning software in this work.

Finally I must thank the coworkers and those who gave me my start in medical physics: Dr. Rajat Kudchadker, Dr. Karl Prado, Scott LaNeave, Craig Martin, Robert Sessions, and my wife. These people greatly contributed to my knowledge and desire to pursue a career in medical physics.

COMPREHENSIVE CALCULATION-BASED IMRT QA USING R&V DATA, TREATMENT RECORDS, AND A SECOND TREATMENT PLANNING SYSTEM

Jared Dean Ohrt, BS

Supervisory Professor: Peter Balter, Ph.D.

Purpose: Traditional patient-specific IMRT QA measurements are labor intensive and consume machine time. Calculation-based IMRT QA methods typically are not comprehensive. We have developed a comprehensive calculation-based IMRT QA method to detect uncertainties introduced by the initial dose calculation, the data transfer through the Record-and-Verify (R&V) system, and various aspects of the physical delivery.

Methods: We recomputed the treatment plans in the patient geometry for 48 cases using data from the R&V, and from the delivery unit to calculate the "as-transferred" and "as-delivered" doses respectively. These data were sent to the original TPS to verify transfer and delivery or to a second TPS to verify the original calculation. For each dataset we examined the dose computed from the R&V record (RV) and from the delivery records (Tx), and the dose computed with a second verification TPS (vTPS). Each verification dose was compared to the clinical dose distribution using 3D gamma analysis and by comparison of mean dose and ROI-specific dose levels to target volumes. Plans were also compared to IMRT QA absolute and relative dose measurements.

Results: The average 3D gamma passing percentages using 3%-3mm, 2%-2mm, and 1%-1mm criteria for the RV plan were 100.0 (σ=0.0), 100.0 (σ=0.0), and 100.0 (σ=0.1); for the Tx plan they were 100.0 (σ=0.0), 100.0 (σ=0.0), and 99.0 (σ=1.4); and for the vTPS plan they were 99.3 (σ=0.6), 97.2 (σ=1.5), and 79.0 (σ=8.6). When comparing target volume doses in the RV, Tx, and vTPS plans to the clinical plans, the average ratios of ROI mean doses were 0.999 (σ=0.001), 1.001 (σ=0.002), and 0.990 (σ=0.009) and ROI-specific dose levels were 0.999 (σ=0.001), 1.001 (σ=0.002), and 0.980 (σ=0.043), respectively. Comparing the clinical, RV, TR, and vTPS calculated doses to the IMRT QA

measurements for all 48 patients, the average ratios for absolute doses were 0.999 (σ=0.013), 0.998 (σ=0.013), 0.999 σ=0.015), and 0.990 (σ=0.012), respectively, and the average 2D gamma(5%-3mm) passing percentages for relative doses for 9 patients was were 99.36 (σ=0.68), 99.50 (σ=0.49), 99.13 (σ=0.84), and 98.76 (σ=1.66), respectively.

Conclusions: Together with mechanical and dosimetric QA, our calculation-based IMRT QA method promises to minimize the need for patient-specific QA measurements by identifying outliers in need of further review.

Table of Contents

Signature Page ..... *i*

Title Page ..... *ii*

Acknowledgements ..... *iii*

Abstract ..... *iv*

Table of Contents ..... *vi*

List of Figures ..... *ix*

List of Tables ..... *x*

List of Equations ..... *xi*

Abbreviations ..... *xii*

1 Introduction *1*

1.1. Megavoltage Photon Radiotherapy **1**

1.1.i. Early Megavoltage Radiotherapy 1

1.1.ii. Treatment Planning System and CT simulation 2

1.1.iii. Dose Calculation Algorithms 3

1.1.iv. Treatment Planning Process 8

1.1.v. Beam Modulation 10

1.2. The Current Radiotherapy Treatment Process **11**

1.2.i. CT Simulation 11

1.2.ii. Treatment Planning 12

1.2.iii. The Record & Verify System 13

1.2.iv. The Treatment Unit 14

1.3. Requirements of a QA Program **15**

1.3.i. Data Transfer Verification 16

1.3.ii. Treatment Delivery Verification 16

1.3.iii. Dose Calculation and Treatment Plan Verification 17

1.4. Quality Assurance for Un-modulated Treatments **17**

1.5. Quality Assurance for IMRT Treatments **19**

1.5.i. Measurement-based IMRT QA 21

1.5.ii. Calculation-based IMRT QA 23

1.5.iii. The Need for Improvement 24

1.6. Proposed Comprehensive Calculation-Based IMRT QA 25
1.6.i. Pretreatment QA 25
1.6.ii. Post Delivery QA 27
1.6.iii. Increased Machine-Specific QA 28
1.7. Hypothesis and Specific Aims 29
2 Methods and Materials 30
2.1. The Pinnacle^3 Treatment Planning System 30
2.2. Patient Selection 31
2.3. Data Transfer Verification 31
2.4. Treatment Delivery Verification 32
2.4.i. MLC dynalog files 32
2.4.ii. Combining RT-Record and MLC dynalog Information 33
2.5. TPS Comparison 35
2.6. 3D Gamma Analysis 36
2.7. ROI Dose Comparisons 38
2.8. IMRT QA Measurement Comparisons 39
3 Results 40
3.1. Software Performance and Verification 40
3.2. 3D Gamma Analysis 41
3.3. ROI Dose Comparisons 43
3.4. IMRT QA Measurement Comparisons 46
4 Discussion 48
4.1. Software Performance and Verification 48
4.2. Delivery Instructions 48
4.3. Delivery Verification 49
4.4. TPS Comparison 49
5 Conclusions 50
6 Appendix A: Pinnacle .Script building software 52
7 Appendix B: Pinnacle Import software 97
8 Appendix C: 3D Gamma Calculation software 104
9 Appendix D: DVH Data Export software 131

10 References 143

11 Vita..... 150

List of Figures

Figure 1.1 A diagram of the flow of information in a radiotherapy treatment 13

Figure 1.2 A diagram of the Information flow in two common types of IMRT QA. (A) Measurement-based IMRT QA and (B) Typical calculation-based IMRT QA 20

Figure 1.3 A diagram of the flow of information used by (A) the proposed IMRT QA method and (B) in this study. 27

Figure 3.1 The mean percentages of voxels passing a 3D gamma comparison. 42

Figure 3.2 The mean percentage of voxels passing the 3D gamma comparisons separated by treatment site... 43

Figure 3.3 A histogram of the ROI mean dose comparisons. 45

Figure 3.4 A histogram of the ROI-specific dose level comparisons. 45

Figure 3.5 Results of the comparison of the IMRT QA absolute dose measurement 47

List of Tables

Table 3.1 Results of the 3D gamma comparisons between the RV and Clinical Plans. 41

Table 3.2 Results of the 3D gamma comparisons between the Tx and Clinical Plans 42

Table 3.3 Results of the 3D gamma comparisons between the vTPS and Clinical Plans. 42

Table 3.4 The average ratio of the RV and clinical Mean ROI Doses and type-specific ROI levels 43

Table 3.5 The average ratio of the Tx and clinical Mean ROI Doses and type-specific ROI levels 44

Table 3.6 The average ratio of the vTPS and clinical Mean ROI Doses and type-specific ROI levels 44

Table 3.7 Mean ratios of the calculated doses to the IMRT QA measured absolute dose 46

Table 3.8 Results of the 2D gamma comparisons of the IMRT QA film measurements to the dose distributions.....47

List of Equations

Equation 1.1 The Relationship between Absorbed Dose and Energy Deposited……8

Equation 2.1 Computation of the Gamma index (Γ)……37

Equation 2.2 Computation of the Quality index (γ)……37

Abbreviations

AAA	Analytic Anisotropic Algorithm
CCC	Collapsed Cone Convolution
CT	Computed Tomography
CTV	clinical treatment volume
D95	The maximum dose received by 95% of the voxels in an ROI
D99	The maximum dose received by 99% of the voxels in an ROI
D100	The maximum dose received by 100% of the voxels in an ROI
DICOM	Digital Imaging and Communications in Medicine
GTV	gross tumor volume
GUI	graphical user interface
Gy	Gray, the units of absolute dose
HU	Hounsfield Units
ICRU	International Commission on Radiation Units and Measurements
IMRT	Intensity Modulated Radiation Therapy
ITV	integrated treatment volume
LUT	Look-up Table
MLC	Multi-leaf Collimator
OAR	organs-at-risk volume
PRV	planned organs-at-risk volume
PTV	planned treatment volume
QA	Quality Assurance
R&V	Record and Verify
ROI	Region of Interest
RV	Label given to plans and dose distributions results from delivery instructions from the R&V system
TERMA	total energy released in matter
TPS	treatment planning software
Tx	Label given to plans and dose distributions results from Treatment records created by the treatment console and MLC controller
vTPS	Label given to plans and dose distributions results from the 2nd (verification) TPS

1 Introduction

One of the most important aspects of radiation therapy is ensuring the safety and accuracy of each treatment. The process of accomplishing this task is broadly defined as quality assurance (QA). Every radiotherapy facility must employ a set of systematic procedures meant to ensure that each treatment is delivered as planned. Advances in medicine and technology have increased the complexity of radiation therapy and correspondingly have increased the burden of the QA process.

In this document, the terms "radiotherapy" and "treatment" refer to the use of megavoltage x-rays only. Treatments involving lower energy photons, electrons, or heavy particles are beyond the scope of this study.

In order to gain a better understanding of QA in radiation therapy it is helpful to review the types of radiation therapy treatments, the modern radiation therapy treatment process, and the requirements of a QA program.

1.1. Megavoltage Photon Radiotherapy

1.1.i. Early Megavoltage Radiotherapy

Megavoltage radiotherapy started with simple beam arrangements, such as parallel opposed beams, with either rectangular fields or fields shaped by blocks designed based on projection x-rays combined with the radiation oncologist's knowledge of anatomy[1]. Dose calculations were performed manually or by computers using lookup table (LUT) based algorithms [2]. These algorithms started with an LUT that represented measured dose distributions in a water phantom [3, 4] and corrected them to better match the patient's treatment geometry. These simple dose calculations were adequate because of the simplicity of the treatment fields[1]. In these algorithms patients were approximated as a homogeneous mass, so a few parameters obtained from LUTs were all that were needed to calculate the dose at any point in the treatment volume. The effect of

irregularly shaped treatment fields could be accounted for using Clarkson integration [3, 4] to correct the LUT-data.

1.1.ii. Treatment Planning System and CT simulation

Incorporation of computed tomography (CT) images into the radiation dose calculation began in the late 1970's [5] but did not become widespread until the 1990's [6]. Inclusion of the CT simulation dataset was an important advancement in radiotherapy planning. The CT dataset provides the geometry of the patient, provides tissue density information, and defines a coordinate system that is used to orient the treatment plan to the patient anatomy[5, 7]. The geometry provided by the CT dataset allows for the definition of regions of interest (ROIs) that correspond to various treatment volumes, normal tissues, and/or critical anatomy[8, 9], so that the dose to these structures can be quantified.

CT datasets are composed of volume elements known as voxels. The value of each CT voxel is a CT-number or Hounsfield Unit (HU), which is predominantly a function of the average physical density of the tissue contained in the voxel[10]. At therapeutic x-ray energies (6- 25 MV), the attenuation and dose deposition in tissue is also primarily a function of the physical tissue density[3]. The tissue density information from a CT dataset can be used by a treatment planning system (TPS) to account for tissue inhomogeneities during dose calculations and optimization [11].

CT data allowed the simple LUT-based treatment planning system to apply simple corrections for the geometry of the patient and the tissue inhomogeneities. However, as the complexity of plans increased, the accuracy and flexibility of LUT-based algorithms was no longer adequate. Model-based planning systems were introduced to overcome some of the weaknesses of LUT-based systems. Unlike LUT-based methods which scale measured dose distributions, model-based calculation attempts to compute dose from first principles. In model-based systems, beam data are not used directly by the TPS, but rather are used to tune the parameters of a mathematical model

of the specific x-ray beam[12]. This method improves the accuracy of dose calculations for inhomogeneous tissues[2]. Compared to LUT-based systems, model-based methods can more accurately compute the dose delivered from complicated beam arrangements and can better account for heterogeneities inside the patient. This improved accuracy allows the creation of dose distributions with better conformality to the intended target and/or better sparing of critical structures because allowances for the uncertainties in the dose calculations can be smaller.

1.1.iii. Dose Calculation Algorithms

The goal of a TPS is to calculate the absorbed dose in a patient resulting from the radiation delivered by the treatment unit. The absorbed dose in the patient is represented by an array of discrete points and is referred to as a dose grid. Each point in the dose grid represents a volume element known as a voxel, which are independent of the CT voxels. The absorbed dose in each voxel is the energy absorbed in the voxel divided by its mass and is quantified in units of Gray (Gy), where 1 Gy = 1 J/kg. Throughout this work the term “dose” is used in place of “absolute dose”. The calculation of dose from first principles can be difficult due to the statistical nature of the radiation interaction processes. One method used to calculate dose is to simulate the transport of each individual radiation quantum from its source through the patient including all random interactions. This is known as the Monte Carlo method, which is described in detail the literature [13-15]. In general, Monte Carlo creates an independent random sample of a population using a probability function and a random number generator [15]. The probability functions used in radiotherapy applications describe interaction probabilities and dose deposition based on physical principles. For a given quantum at a given point, a random number generator is used in conjunction with probability equations to determine if an interaction occurs, the type of interaction that occurs, the direction and energy of any secondary radiation produced in an interaction, and the dose deposited locally.

The Monte Carlo simulation of a photon beam produced by a linear accelerator, or linac, begins with a model of a beam of electrons hitting an x-ray bremsstrahlung target. Electrons and their secondary radiations are then simulated one-by-one as they traverse the bremsstrahlung target, the flattening filter, the treatment head, the beam collimation devices, and the patient anatomy. This calculation requires that the shape, position, and composition of all materials irradiated by the beam be modeled [16-18]. Once the treatment head is modeled, the beam fluence exiting the treatment head can be computed for any combination of beam modifiers. This fluence then interacts with the patient as modeled by a CT-dataset [19]. This method can produce an accurate dose distribution even in the presence of interfaces between different materials. The accuracy of the dose distribution is only limited by the number of quanta tracked and the complexity of the model.

The biggest drawback to Monte Carlo methods is that the calculations are computationally expensive. Long calculation times are needed to compute dose with sufficient accuracy [20, 21], because it is necessary to simulate large numbers of radiation quanta, which each generate secondary radiation that must also be simulated. The need to decrease calculation times led to the development of other dose algorithms that borrow from Monte Carlo, but can perform dose calculations much more quickly[21].

One such class of dose calculation algorithms is convolution superposition algorithms. Convolution superposition based calculations typically divide the dose calculation into three parts: a model of the beam fluence at the accelerator head, a model of how the energy will be released into the irradiated volume, and a model of how that energy will be deposited as dose.

Beam modeling in a TPS starts with a generic model of an x-ray beam based on Monte Carlo simulation of a standard treatment unit [21]. The TPS then fits the model to detailed dose measurements in a water phantom. The user can also manually adjust certain parameters in the

model to achieve better agreement with the measured doses because compromises must be made between the quality of the fit in different regions of the dose distributions to achieve the most clinically useful models. When an acceptable model is developed, it can be used to determine beam fluence at the exit window of the treatment unit for any clinically realizable combination of beam modifiers.

The distribution of energy released to the irradiated volume by the fluence is quantified by the calculation of the total energy released in matter (TERMA). The TERMA distribution can be calculated from first principles by scaling the depth in the medium by the attenuation properties of the medium. The effect of patient geometry and attenuation on the TERMA distribution are incorporated into the calculation by considering the inhomogeneity of the medium in the depth direction[22]. To determine how the TERMA is deposited as dose, the TERMA distribution is convolved with a dose deposition kernel. Dose deposition kernels are based on Monte Carlo derived point-spread functions of relative dose deposition [23, 24]. Kernels can be derived for specific sources of energy deposition, or combined into a single kernel representing all sources. In the case of irradiation of a homogenous medium, the kernel is spatially invariant and Fourier techniques can be employed to speed up the convolutions. Although kernels are computed in a homogenous medium, tissue heterogeneities are incorporated by scaling the kernels using density information from the CT dataset [23-25]. However, when the kernels are scaled they become spatially variant and Fourier techniques can no longer be applied [22]. This means that integrals must be carried out to determine the dose deposited at each point from every other point. When the volume is divided into a dose grid, the integrals become summations that reduce the computational requirements, but convolution superposition in the inhomogeneous case is still computationally expensive [21, 22].

Slightly modified convolution superposition algorithms have been developed which decrease dose computation time by simplifying the calculations. Two such algorithms incorporated into commercial TPSs are the Collapsed Cone Convolution (CCC) and the Analytic Anisotropic Algorithm (AAA) [26].

CCC is the algorithm used by Pinnacle^3 TPS (Philips Medical Systems, Fitchburg, WI, USA) one of the common TPSs used for 3D-XRT. Pinnacle^3 models the beam from the linac by adjusting pre-computed Monte Carlo models to fit measured beam data. These models include the primary, secondary, and electron contamination sources. The electron contamination source is accounted for by adding electron dose to the dose from other sources using a modified exponential function[2]. CCC was developed to overcome the fact that the speed of a convolution superposition dose algorithm is limited by the need to continually sum over the entire volume to calculate dose to each voxel. This summation is required because energy released by the primary beam in a given voxel is nonuniformly distributed across an entire 4π solid angle defined in spherical coordinates. To increase the speed of the calculations, CCC algorithms simplify the geometry of the energy released by dividing the solid angle into discrete cones[22]. Each cone is then mathematically "collapsed" on to its axis, and all energy scattered into the cone is considered to travel along the cone's axis[22]. This energy, along with any energy released into cones that share the same axis, is transported along the axis where it is deposited and attenuated as a function of the radiological distance traveled [22, 27]. In this collapsed cone geometry there is no need to sum over the entire volume. The dose scattered to a voxel is the sum of the dose imparted by energy scattered into cones whose axes cross the voxel. The total dose to the voxel is then the sum of the scattered doses, dose from contamination electrons, and the locally deposited doses from primary interactions. Tissue inhomogeneities can be included by scaling the energy scatter kernel. For all

photon treatments, our institution uses the Pinnacle^3 implementation of CCC with CT-based inhomogeneity corrections described in detail by Papanikolaou et al. [28] and McNutt et al. [27].

AAA is a proprietary algorithm employed by the Eclipse™ TPS (Varian Medical Systems, Inc., Palo Alto, CA, USA). It models the linac head as three separate radiation sources [21, 29]: the primary photon source (bremsstrahlung target), the secondary photon source (flattening filter), and an electron contamination source (mainly Compton interactions in the linac head). As with other model-based calculations, the parameters in the model can be adjusted to match measured beam data for each linac.

The AAA method begins by calculating the total energy deposited in each voxel from each source separately. The energy deposited by the primary and secondary photon sources is computed using the same method. For each source, the beam fluence is broken into a number of beamlets that extend radially from each source called pencil beams. The fanlines of these beamlets corresponds to voxels of a divergent dose grid. For the primary source the energy and intensity is a function of radial distance from the central ray. For the secondary source the energy is assumed to be constant and the intensity is a function of radial distance from the central ray.

AAA makes the assumption that the energy deposition can be separated into longitudinal (depth) and lateral (radial) components [21, 29]. To account for inhomogeneities in the medium, the energy deposition in the longitudinal, or depth, direction is scaled by the radiologic depth traveled. The lateral scatter kernel is also scaled by the radiological distance traveled along the radius from the origin of the beamlet. Further corrections are made for changes in the lateral scatter due to inhomogeneities in the longitudinal direction. These corrections are discussed in detail in the literature[21].

The total energy deposited in a voxel due to the primary and secondary photon sources is then the superposition of the energy deposited in that voxel for all beamlets from both sources.

To determine the dose contributions from the electrons originating in the treatment head, AAA first determines the fluence of these "contamination electrons" as a convolution of the primary photon fluence with a Gaussian lateral spread function[29]. The energy deposited by the contamination electrons is given by a second convolution with a second lateral spread Gaussian and is then multiplied by a depth-dependent electron energy deposition function.

The total energy deposition distribution is obtained by superimposing the energy deposition from the three sources. Then the dose distribution, D(x,y,z), is calculated by dividing the energy deposited in each voxel, E(x,y,z), by average electron density of that voxel, $\rho(\mathrm{x,y,z})$, relative to the electron density of water, ρ_{water} :

$$D(x,y,z) = E(x,y,z) \cdot \frac{\rho_{water}}{\rho(x,y,z)} \tag{1.1}$$

1.1.iv. Treatment Planning Process

The first step in the treatment planning process is to define ROIs based on the CT dataset. The International Commission on Radiation Units and Measurements (ICRU) has defined several types of ROIs [8, 9] that are widely used in modern treatment planning. The gross tumor volume (GTV) represents all demonstrable disease, usually on CT imaging. The clinical target volume (CTV) includes the GTV plus any subclinical disease that must be treated based on the clinician's training and experience. Variations in the size, shape, position, and expected physiological movements of the CTV are incorporated by defining an internal target volume (ITV), which consists of the CTV plus a margin to account for motion and deformation of the CTV with respect to the setup surrogate. A planning treatment volumes (PTV) is then defined by adding margins to the CTV or ITV to ensure that it receives the planned dose despite patient setup uncertainties, including the uncertainty in the surrogate, skin marks or boney anatomy, position, and the mechanical uncertainties in the

delivery by the linac. The ICRU also defines normal tissue ROIs known as organs-at-risk (OARs). OARs are organs whose location and/or sensitivity to radiation may influence the goals of the treatment. The ICRU also defines the planning organ at risk volume (PRV) to be the OAR expanded to include daily setup uncertainties. At our institution, we use slightly different definitions for workflow reasons. For thoracic and gastrointestinal cases, GTV is used to refer to the envelope of motion of the ICRU GTV. This is expanded for sub-clinical disease to create a CTV which is similar to the ICRU ITV. We then add additional margins for setup uncertainty, deformation, and changes in motion to get a PTV which is the same as the ICRU PTV. For gynecological cases, we observe the ICRU definitions, and for head and neck cases, the CTV includes setup uncertainties and is thus similar to an ICRU PTV. Only the GTV, CTV, ITV and PTV target structures will be considered in this work.

After the ROIs are defined, a set of initial parameters (e.g. the number of beams, gantry angle, etc.) is defined and used to compute a dose distribution. ROIs enable the calculation of quality metrics for treatment plans such as mean, minimum or maximum dose for clinically significant volumes within the patient. After dose is calculated, the quality metrics for target volumes, OARs, and PRVs are evaluated to see if they meet predefined criteria. If an ROI does not meet clinical dose criteria, the operator adjusts the beam collimation, adjusts the beam weighting, and adds or removes beams as necessary to try and achieve these goals. This process is repeated until a dose distribution that meets the treatment goals is obtained. This method of generating a treatment plan is known as “forward” planning.

The natural extension of this was to have a computer, rather than a human operator, determine the optimal field shapes for each angle.

This is the concept of “inverse” planning, which is generally associated with intensity-modulated radiation therapy (IMRT). In IMRT there are generally a number of treatment fields. For

each field, the beam is modulated to allow the fluence to be a function of position within the 2D beam projection. This modulation allows the therapeutic doses to be delivered to the treatment targets while keeping the dose to the surrounding normal tissues at acceptable levels[30]. IMRT treatment plans are optimized through the "inverse" planning mentioned above. This process starts with an operator defining treatment goals in the form of dose criteria for specific ROIs. Given a predetermined set of treatment fields, the TPS then determines the modulation that most closely achieves the defined goals. This is not a definitive calculation and involves searching through a family of potential solutions to minimize a cost function. The cost function is a measure of how closely a treatment plan meets the planning objectives. One commonly used cost function is the sum of weighting factors multiplied by difference between each treatment goal and the achieved goal for a number of objectives [31]. If the resulting dose distribution is not satisfactory, then the operator adjusts the treatment goals and the process is repeated.

1.1.v. Beam Modulation

IMRT requires differential intensities or fluences across the radiation field. One approach to modulating the beam from a linear accelerator is to place an attenuator in the beam with a 3D shape that produces the desired intensity pattern. Such attenuators, referred to as compensators, can be designed by the TPS and then either manually constructed from blocks of high-density material or machined by computer controlled milling machines. The TPS determined the design of each compensator on a beam-by-beam basis to account for oblique skin surfaces and tissue inhomogeneities. Early compensator-based treatments were not IMRT treatments because an IMRT plan is developed by optimizing the intensities of all treatment fields simultaneously to achieve the desired dose distributions. The use of beam compensators is an expensive and laborious process because a compensator must be designed and constructed for each treatment field. In addition, for every beam of every patient currently under treatment there is a compensator that must be stored

and inventoried. Before a treatment field can be delivered the appropriate compensator must be attached to the treatment unit and verified.

An important innovation in radiotherapy was the introduction of multi-leaf collimators (MLCs). An MLC is a device consisting of a number of small leaves grouped in opposing pairs, designed to block the x-ray beam. These leaves are individually controlled and precisely positioned to create complex field shapes. MLCs were initially introduced as a replacement for custom-fabricated beam blocks. However, it was observed that by either moving the leaves during treatment (sliding window) or delivering the treatment in several segments, each with a different MLC pattern (step-and-shoot), the MLC could be used to achieve beam modulation without the downsides of compensators. This method is technically fluence modulation rather than intensity modulation but achieves the same goal. Advancements in MLC technology and IMRT techniques have led to the widespread use of MLCs for IMRT. The current generation of MLCs have 120 individual leafs. Each leaf can be in 40-100 different positions per treatment field, making manually entry and/or verification of IMRT parameters nearly impossible.

1.2. The Current Radiotherapy Treatment Process

The process for radiotherapy in many facilities (Figure 1.1) is to perform volumetric imaging, generally a CT simulation with the patient in the treatment position; transfer these images to a TPS; generate a patient-specific treatment plan; transfer the plan and images to a record and verify system (R&V); and then transfer them to a treatment console, which then delivers the treatment and transfers a treatment record back to the R&V system. For IMRT treatments, QA of each of these systems is essential to ensuring that the treatment is delivered as planned.

1.2.i. CT Simulation

The current standard for radiation therapy process starts with a CT simulation. CT simulation begins by creating a custom immobilization device for the patient. The immobilization device will

help get the patient into the same position each day and discourages and minimizes movement during the simulation and treatment. The type of device used varies depending upon the facility and treatment site.

After the patient has been immobilized, reference marks are made on the patient's skin. The reference marks are needed to orient the treatment plan and position the patient for treatment. The position of the reference marks can be chosen based on anatomic landmarks or from a pre-simulation CT image. A CT simulation room includes a laser marking system with laser crosshairs that align with the sagittal, coronal, and lateral planes of the patient. The planes defined by these lasers must be coincident and oriented to the CT coordinate system so that a point in a CT scan can be identified by moving the lasers to intersect at the corresponding point in space. In addition to marks on the patient's skin, radio opaque markers may be placed at the reference marks. These markers can be seen on the CT simulation images and will be used during treatment planning to orient the treatment plan to the reference marks. Alternatively, the coordinates of these markers can be independently sent as a DICOM (Digital Imaging and Communications in Medicine, NEMA, Rosslyn, VA, USA) object to the TPS. The CT simulation dataset is then imported into the TPS.

1.2.ii. Treatment Planning

The CT images are imported into the TPS where it provides geometric and radiological information for dose calculation and ROIs are defined as described in Section 1.1.iv. An arrangement of treatment fields is typically chosen based on experience with a given treatment site and patient anatomy. After the clinician establishes the treatment goals, the operator adjusts the treatment fields until the treatment goals are achieved. The physician then reviews the plan, and either gives approval or modifies the treatment goals. Once accepted by the physician, the treatment plan is sent to the R&V system.

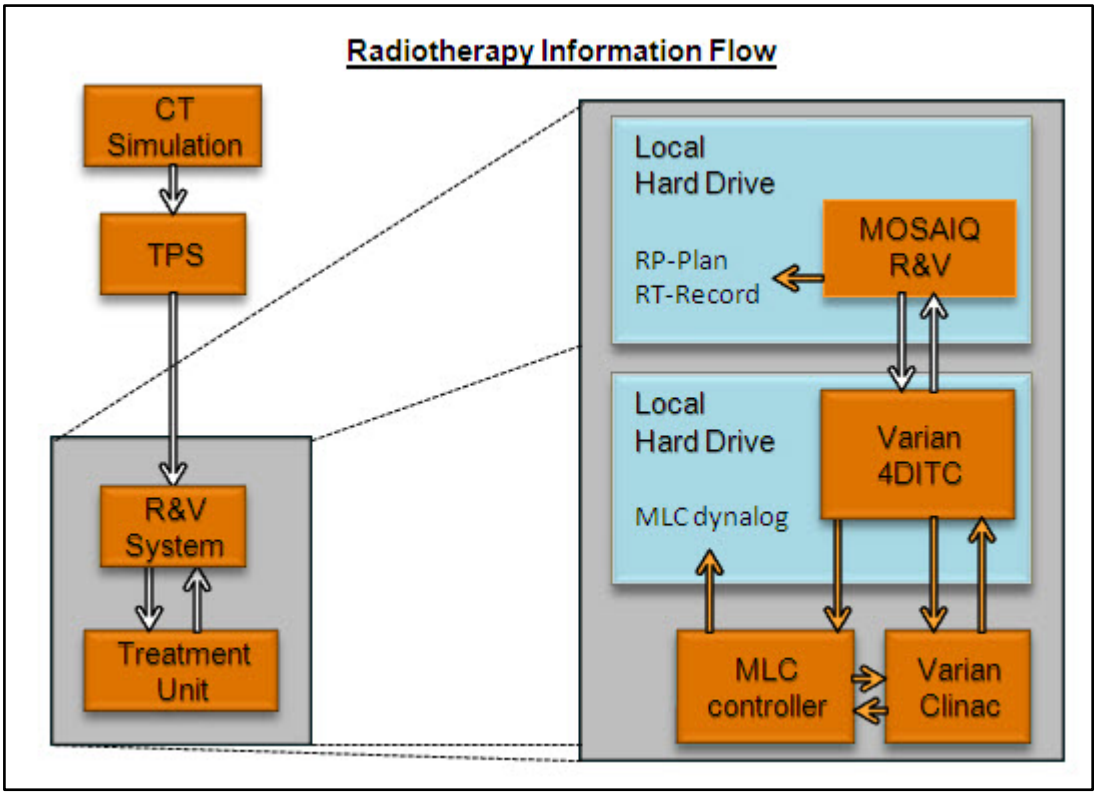

Figure 1.1 A diagram of the flow of information in a radiotherapy treatment. Inset is an exploded view of the communication between the R&V system, treatment console, and treatment unit during radiotherapy treatments at our institution. The MLC controller only creates MLC dynalog files during the delivery of modulated treatment fields.

1.2.iii. The Record & Verify System

The original purpose of R&V systems was to verify the patient setup and treatment unit settings prior to treatment. They have since taken on various other tasks, such as storing relevant treatment information, creating records of treatment deliveries, scheduling, management of clinical notes, and billing. R&V systems are a combination of a database and various task-specific interfaces. The database contains all the information needed to treat each patient and the record of each treatment event. Plan information from the TPS is sent to the R&V via a DICOM import interface and is used to populate the R&V database. A user interface is provided so that the contents of the database can be browsed and/or edited. This interface is used to verify of many parameters of the

data transfer as well as to enter parameters not generated in the TPS, such as couch co-ordinates. When a treatment is to be delivered at our institution, the R&V system uses the information stored in its database to send instructions to the treatment unit. Most of our treatment units receive this information as a DICOM message containing the relevant plan information. Our institution uses MOSAIQ® (Elekta IMPAC Medical Systems, Inc, Sunnyvale, CA, USA) as both an R&V system and to provide an electronic medical record for all radiation therapy treatments. In the process of sending the DICOM delivery instructions to the treatment unit, MOSAIQ creates a temporary copy of the message locally as a DICOM RT-Plan file (Figure 1.1).

The methods described in this study were designed to take advantage of the work flow that we employ clinically. Although these methods could be adjusted to account for different workflows, there are some combinations of older treatments units and R&V software for which these methods cannot be used.

1.2.iv. The Treatment Unit

Most of the treatment units at our institution are linear accelerators controlled by a combination of digital and analog feedback systems (Clinac®, Varian Medical Systems, Palo Alto, CA, USA). The Clinac gets setup instructions from a separate control console (4DiTV, Varian Medical Systems, Palo Alto, CA, USA) that also manages the MLC and imaging systems, which were not part of the original Clinic design. In this configuration the treatment console communicates the delivery parameters received from the R&V to the linear accelerator and monitors its mechanical performance and dosimetric output. After the treatment delivery, the treatment console communicates these parameters back to the R&V system where they are added to the patient's record. The MOSAIQ R&V system also saves a copy of these DICOM messages locally as DICOM RT-Record files. This operational model differs from the original model in which the R&V system both

monitored the delivery and recorded the resultant data without the use of an intermediate treatment console.

The 4DiTC also communicates with another dedicated computer, the MLC controller (Millennium MLC Controller, Varian Medical Systems, Palo Alto, CA, USA). This computer sets the positions of the MLC leaves with respect to either dose or gantry position depending on the type of treatment being delivered. It also logs the positions of various linear accelerator components including the MLC leaves, the gantry angle, the jaws and the collimator angle. Only a limited set of this information is communicated back to the R&V system, but the MLC controller can be made to save it as an ASCII file for each field delivered, Figure 1.1. This file is known as an MLC dynalog file.

1.3. Requirements of a QA Program

The ultimate goal of any radiotherapy QA program is to ensure the safety and effectiveness of patient treatments[32]. As a whole, a QA program must address parts of the radiotherapy treatment process from simulation to treatment, end-to-end, to minimize errors.

QA can be broken into two categories: machine-specific QA and patient-specific QA. Machine-specific QA is a regular check of the basic functionality of the components of the treatment process: the CT simulator [33-35], the TPS [36], and the treatment unit [37-39]. These regular checks are performed under controlled conditions and are meant to identify problems with machine performance. Patient-specific QA involves the verification of treatment components that can be customized on a patient-by-patient basis, including the CT dataset, dose distribution, field shapes, field orientations, beam modulation, and/or treatment goals that may be unique to a particular patient.

The relative contributions of machine-specific and patient-specific QA to a QA program vary depending upon how a QA program is implemented, and often there is overlap between them. A

detailed discussion of machine-specific QA is outside of the scope of this study. Machine-specific QA is only discussed as it relates to patient-specific QA.

Patient-specific QA must verify that no errors have been introduced during the transfer of electronic data [40, 41] ,during the treatment delivery, or in treatment planning and dose calculation [42-44]. These sources of error are discussed in this section. The methods used to perform patient-specific QA differ between un-modulated and IMRT treatments, so they will be addressed separately in the Sections 1.4 and 1.5, respectively.

1.3.i. Data Transfer Verification

Computing networks have become an important part of the treatment process because they link the various components in the process. Errors can be introduced as information is transferred and interpreted between systems. Such transfer errors can cause unexpected behaviors that may result in errors in the delivery of a treatment[45]. It is important that any QA system employ a method to verify that no errors have been introduced as data is transferred during the treatment process.

1.3.ii. Treatment Delivery Verification

Many radiotherapy treatments are delivered by a medical linear accelerator, often referred to as a linac. A linac is a complex device that relies on the integration of a number of mechanical and electrical components as well as control software. Machine-specific QA of a linac ensures that it precisely controls its components and monitors its dosimetric output, and it ensures that the linac will prevent radiation from being produced if a subsystem is not operating within specifications. However, as with any such device there are random uncertainties associated with mechanical components (e.g. MLC, Jaw, or Gantry position).These may be within specifications for operation but still affect the plan quality and should be included in the treatment verification process.

1.3.iii. Dose Calculation and Treatment Plan Verification

After a treatment plan has been deemed acceptable, a manual verification is performed by a medical physicist as a reasonableness check. This is done to make sure that the plan is based on the correct CT dataset, that the proper dose algorithms were used, that the dose calculated is reasonable, that the treatment field parameters are reasonable, and that the plan meets clinical safety guidelines. This manual verification is achievable for un-modulated treatments, but may not be for modulated fields because the inverse planning engine may develop non-intuitive solutions.

Verification of the dose calculation is an important part of the QA process, especially as dose calculations move further into the realm of computers. Modern TPSs employ sophisticated dose algorithms that model linac output and make patient-specific heterogeneity corrections. Because each treatment plan is unique to a given patient/treatment site, it is difficult to identify conditions that may cause dose calculations to have large uncertainties. This means that dose calculations should be verified on a patient-specific basis, ideally in the patient specific geometry.

1.4. Quality Assurance for Un-modulated Treatments

Un-modulated treatments usually involve one or a few treatment fields that can be described by a small number of parameters. This means that transfer of the treatment plan to the R&V system can be verified manually by simply comparing the parameters from TPS to the parameters in the R&V system.

The lack of modulation also reduces complexity of the delivery, which limits the treatment fields to relatively simple geometries. The performance of the treatment unit using these simple geometries can be verified by routine machine-specific QA. In addition to this, machine components do not move during the delivery un-modulated treatment fields. Thus verification of the position of the linac components just before delivering a field serves as a verification of the

mechanical aspects of the delivery. The dosimetric performance of the linac is ensured by machine-specific QA [37, 38] and linac safety features.

An independent verification of un-modulated dose calculations can be accomplished by a second calculation performed independent of the initial calculation either manually by a second person[43] and/or by dose verification software. Dose verification software generally uses a simpler algorithm than the TPS to calculate dose to a few reference points in the patient's dose distribution.

If a mistake is made in treatment planning or if corrupt data is used during the initial dose calculation, it is unlikely that the same problem will occur during an independent dose calculation. The accuracy of the LUT data or beam model is verified by comparison with routine machine-specific measurements [37, 43].

Verifying each treatment field by direct measurement serves as a verification of the data transfer, treatment delivery, and dose algorithm. However, measurement-based methods have their own drawbacks. Measurements usually take place in a simplified phantom which does not simulate patient inhomogeneities [43]. In this respect, they may be a poor test of the inhomogeneity corrections made by the TPS. In addition to this, these measurements also have uncertainties that may obscure real systematic problems that produce errors smaller than the level of experimental uncertainties (1%).

Patient-specific measurements are also time-consuming because each treatment must be delivered on a treatment unit at a time that does not interfere with normal patient treatments. This can incur additional staffing and maintenance expenses that add to the overall costs of radiotherapy treatment.

For un-modulated treatments, the combination of machine-specific QA and patient-specific dose calculations accomplishes the goals of a QA program and is more efficient than patient-specific measurements.

1.5. Quality Assurance for IMRT Treatments

IMRT treatment plans tend to be very complex. In general, each plan has multiple treatment fields with each field having multiple treatment segments. QA of IMRT-based treatments has the same basic requirements as QA of un-modulated treatments. The verification of the data transfer of un-modulated treatments is done manually as described above. The beam modulation in IMRT treatments increases the number of parameters per field significantly. This increase makes manual verification of each IMRT parameter impractical, so data transfer must be verified by some other means.

Verification of the treatment delivery is also more difficult for IMRT treatments because each field has multiple, intricately-shaped segments. Unlike un-modulated treatments, machine components do move during delivery of an IMRT field. IMRT treatment fields are modulated by an MLC which controls up to 120 leaves that move throughout the delivery of the field. While the initial position of the treatment unit components can be verified, there is no practical way to manually verify the position of each MLC leaf throughout the delivery. In addition to this the accuracy requirements for the position of the MLC leaves is much higher than for un-modulated fields. Radiation is partially transmitted through the ends of the MLC leaves. In un-modulated treatment fields only a small part of the dose is delivered along the aperture edges, and thus makes a smaller contribution to the dose delivered. However, in IMRT a large portion treatment field receives dose from the partial transmission through the leaf ends, so the accuracy of TPS model of leaf-end transmission is more important.

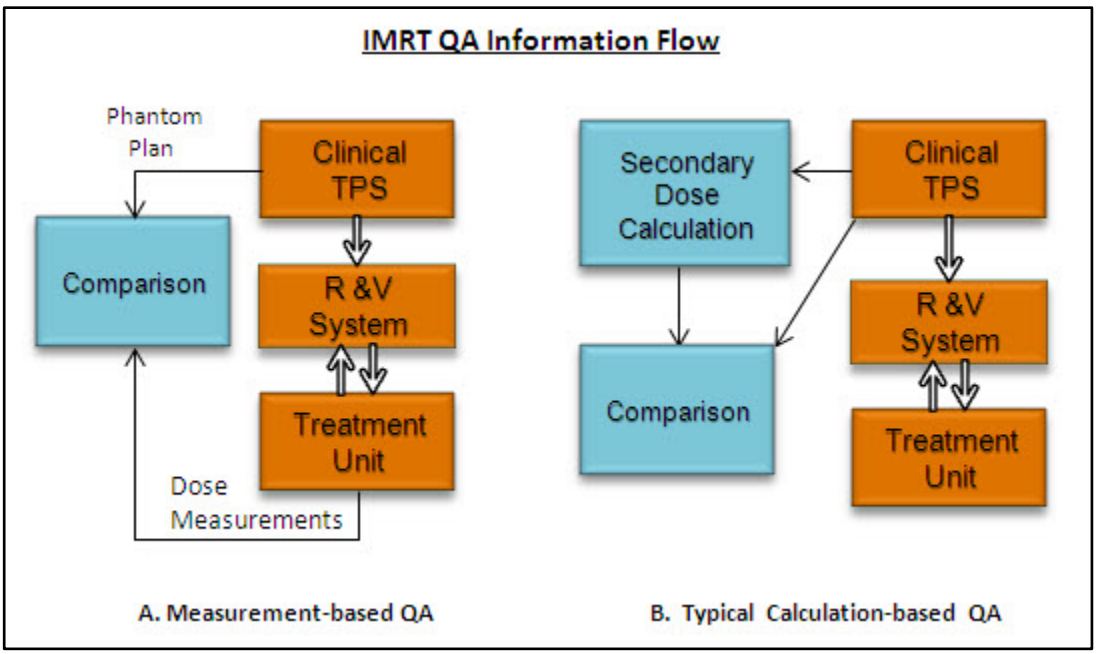

Figure 1.2 A diagram of the Information flow in two common types of IMRT QA. (A) Measurement-based IMRT QA involves making dosimetric measurements of the treatment fields using a phantom and comparing the measurements to the values predicted by a TPS dose calculation in phantom geometry. (B) Typical calculation-based IMRT QA involves sending some or all of the plan information from the Clinical TPS to a secondary dose calculation software. Then dose calculated by the second software is compared to the clinical dose calculation from the TPS.

One of the chief benefits of IMRT treatments is the ability to more closely conform dose to the patient anatomy. IMRT's ability to deliver highly conformal dose distributions has enabled the escalation of the dose delivered to the treatment target [46-48]. This also increases the risks associated

with delivery uncertainties and/or mistakes. Dose calculations in IMRT are performed entirely by computers due to the large numbers of field segments, each of which has its own weighting and complex shape. While computer computational methods are more accurate than manual calculations, there is still the possibility that the dose calculations may be inaccurate in certain situations. Every treatment is unique to a particular patient/treatment site combination. It is

difficult to identify when the accuracy of the TPS dose algorithm may be beyond an acceptable clinical level.

Any IMRT QA method employed must overcome all of the difficulties mentioned above and provide an end-to-end verification of the treatment verifying not only the calculations but also the data transfer. There are two general types of IMRT QA: measurement-based and calculation-based. In addition to describing previous calculation-based approaches, this study also proposes a new calculation-based method, which is described in Section 1.6.

1.5.i. Measurement-based IMRT QA

Measurement-based IMRT QA involves delivering the treatment to a QA device, which samples the dose distribution in at least one plane for comparison to the TPS, Figure 1.2A. This is the current standard for IMRT QA, because it checks many aspects of the IMRT process from the TPS to the mechanical delivery.

Measurement-based IMRT QA is not without its drawbacks. It generally only samples a limited part of the dose distribution and it is done in a non-patient geometry. It is also very time consuming [43]as someone has to physically be at the treatment machine to deliver all of the plans being evaluated. It also adds to the workload of the linac, which adds to the maintenance needs. The time needed to perform measurements could also be used for routine maintenance or expanded treatment hours.

At most institutions, MD Anderson included, the absolute dose is only measured in a small volume (i.e. the active volume of the ion chamber), and the film measurement is only made in a one or a few axial planes. Dosimeter arrays (ion chamber, diodes, or solid state) may also be used to make measurements, but these also only sample a small part of the treatment volume. Several vendors have released 3D arrays but these are generally 2D arrays in non-planar geometries and/or are very expensive (ArcCheck, Sun Nuclear Corporation, Melbourne, FL, USA; Delta4, ScandiDos,

Uppsala, Sweden). In addition to this, in IMRT treatments there may be no point in the dose distribution that is irradiated by every segment of every beam [49]. Thus it is possible that even though the plan agrees with the IMRT QA measurements it may differ in another part of the treatment volume not measured [40]. It can also be difficult to make precise measurements in low dose parts of the treatment volume which may contain critical structures [41].

Another drawback of measurement-based QA is that comparisons do not usually use patient-specific geometry or attenuation [40, 43]. The expected dose to the phantom is calculated by transferring the clinical treatment plan to a phantom geometry defined by a CT dataset of the phantom. This "hybrid" plan is used to calculate the phantom dose distribution [50]. The phantoms used for IMRT QA measurements are typically homogenous and therefore do not effectively test the inhomogeneity corrections made by the TPS.

Our institution uses measurement-based IMRT QA consisting of an absolute dose measurement at a point using an ion chamber and a relative dose measurement in an axial plane using radiographic film. The measurements are made in a homogenous phantom (I'mRT Phantom, IBA Dosimetry, Schwarzenbruck, Germany), and compared to the dose distribution generated by creating a hybrid plan using a CT dataset of the phantom. The measured ionization chamber dose is compared to the mean dose to the active volume of the ionization chamber as calculated in the Pinnacle^3 hybrid plan. The film measurement is compared to a dose plane exported from the Pinnacle^3 hybrid plan. The combination of the absolute dose and relative planar dose distribution measurements are used to verify that the plan was calculated, transferred, and delivered as expected, because any errors introduced will be reflected in the measurements. Of course the only calculation uncertainties that will be discovered would be in the basic beam model as the patient geometry/attenuation is not included or is the actual dose calculation used for the patient treatment.

1.5.ii. Calculation-based IMRT QA

The other general method of IMRT QA is calculation-based IMRT QA. This involves performing a second, independent dose calculation using the clinical treatment plan data, Figure 1.2B. Calculation-based methods are less labor intensive than measurement-based methods, and have the potential to be highly automated.

Unlike measurements, secondary dose calculations can be performed on the patient CT-dataset and can effectively test not only the dose algorithm, but also the inhomogeneity corrections of the TPS. This also allows the plan to be evaluated at every point in the dose distribution instead of limiting the comparison to a small volume or plane.

Traditionally, secondary dose calculations are done using secondary dose calculation software that only calculates dose at a single point per field with a simplified calculation algorithm (DIAMOND™,PTW, Freiburg, Germany),(RadCalc®,LifeLine Software, Inc, Austin, TX, USA). This often results in poor agreement between the secondary calculation and the original plan as this point maybe highly blocked for that field or maybe in an area of non-water like scattering material. It has been proposed that a second high quality dose calculation can be done to validate the entire treatment plan [51], but this was only demonstrated for a homogeneous phantom without CT-based heterogeneity corrections.

Another significant drawback to the typical calculation-based methods is that they do not verify the data transfer to the linac. In general, only the treatment plan and physical and effective depth are transferred directly to the dose verification software and this data transfer is not along the IMRT treatment chain. This means that it cannot verify data transfer errors. One published event of a radiation delivery accident was, in fact, caused by a data transfer issue[45], demonstrating the importance of the end-to-end check.

In addition, typical calculation-based methods include no means to verify the treatment delivery. Two groups have investigated overcoming this short fall using MLC dynalog files in conjunction with R&V treatment records to reconstruct the treatment fields as they were delivered[52, 53]. These reconstructed fields were then used as input for Monte Carlo based dose calculation in patient geometry. The resulting dose distributions were then compared to the clinical distribution. The methods investigated by these groups require that the treatment fields be delivered before the calculations can be made, because this is the only way to generate the MLC dynalog files. This means either QA cannot be performed until after the patient's first fraction or the treatment must be delivered in a QA capacity before the patient's first fraction. QA must be performed before the patient's first fraction, but delivering the treatments in a QA capacity makes these calculation-based methods as labor intensive as measurement-based methods. In addition, to achieve sufficient precision in these Monte-Carlo based calculations requires a large amount of computation time limiting their usefulness in busy clinics.

1.5.iii. The Need for Improvement

It is widely believed that measurement-based IMRT QA cannot be abandoned, because it is the only way to verify the data transfer and the treatment delivery [49]. The intrinsic cost of delivering an IMRT treatment is the same as an un-modulated treatment except for the costs of measurement based QA. The increased use of IMRT increases the costs and time commitments required for patient-specific QA measurements. The rising cost of health care could become a prohibitive factor in the decision to use IMRT. Bringing the costs of IMRT closer to those of un-modulated treatments could enable continued or expanded use of this technology.

As mentioned above, calculation-based methods are potentially much more efficient than measurement-based methods and can provide a more comprehensive dose comparison. Despite this fact little progress has been made to overcome the shortcomings of calculation-based methods

in an effort to move away from patient-specific measurements. It is the onus of the medical physics community to continually improve the quality and efficiency of the QA process[40]. After more than a decade of experience with IMRT, we propose to take a step toward reducing the costs of current IMRT QA methods while increasing the quality.

1.6. Proposed Comprehensive Calculation-Based IMRT QA

The three most glaring deficiencies in traditional calculation-based QA are the lack of sophistication in the secondary dose calculation, the lack of the data transfer verification, and the lack of delivery verification. In the proposed system, Figure 1.3A, the secondary dose calculation will be performed using the Eclipse TPS, a second fully commissioned TPS. The remaining two deficiencies are addressed by taking advantage of the data transfer in the IMRT Treatment process. In the proposed system, the information imported into the 2nd TPS will come from two sources: the delivery instructions created by the R&V system for the treatment machine and the treatment records from the treatment machine. Previous calculation-based methods were lacking, because the plan information is transferred directly to the verification software from the TPS and thus were not able to find data transfer issues or to identify plans that were beyond the mechanicals limits of the treatment unit.

1.6.i. Pretreatment QA

During every treatment, the R&V system communicates the delivery instructions to the treatment console via a DICOM message that is also saved locally as a DICOM RT-Plan. The proposed system will use the delivery instructions as input into the 2nd TPS where it will be used to compute a 3D dose distribution using heterogeneity corrections based on the same CT dataset used to compute the clinical dose distribution.

The 2nd TPS is the Eclipse TPS which employs AAA which matches the sophistication of the CCC algorithm employed by the Pinnacle^3 TPS. The beam models used by the two TPSs were developed

independently using each TPS's beam modeling tools. The Pinnacle^3 model was adjusted to match detailed beam measurements of one of our linear accelerators. The Eclipse model was adjusted to match standard set of beam data provided by the linac manufacturer (Varian Medical Systems, Inc., Palo Alto, CA, USA). The fact that the two beam models were developed using independent techniques and datasets ensures that the models are independent of one another. This coupled with the fact that the TPSs use different types of dose algorithms allows dose calculations by one TPS to serve as an independent verification of the other. We are not concerned with exact agreement between the two systems, but wish: 1) to look for gross errors; and 2) to use the differences between the two TPSs to indicate situations in which there may be more uncertainty in the clinical dose calculations.

The use of a 2nd TPS was suggested by the American Association of Physicists in Medicine (AAPM) [43], but it was only recommend to make a point comparison. Anjum et al. [51] investigated using a 2nd TPS for IMRT QA using more comprehensive comparisons but used phantom geometry without the use of inhomogeneity corrections. This does not accurately simulate the conditions of a clinical dose calculation. Also, their study did not address data transfer or delivery verification. A comparison of the dose distribution derived from the delivery instructions with the clinical dose distribution will serve as an independent dose verification. In addition to this, the dose comparison will verify that no errors have been introduced by data transfer or interpretation errors, because any such errors will be contained in the delivery instructions.

The R&V system can be made to generate these files without having to deliver the treatment, so this step of the proposed QA process can be done before the patient's first fraction as a part of the pretreatment plan verification. An ideal implementation of this system would have a virtual linac as a treatment device in the R&V system with identical geometries and energies to the actual linac being used to deliver the treatment.

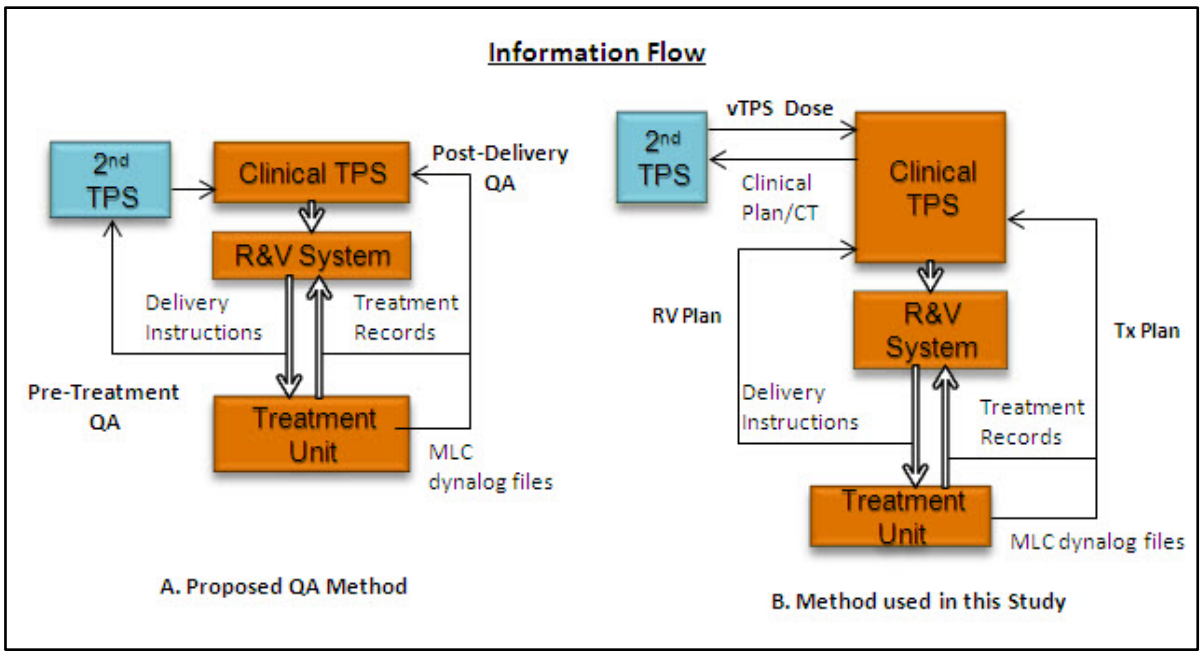

Figure 1.3 A diagram of the flow of information used by (A) the proposed IMRT QA method and (B) in this study. The proposed post-delivery QA is identical to the delivery verification (Tx Plan) portion of this study. The pre-treatment portion of the proposed QA method has been broken into two pieces for this study so that the data transfer errors (RV-plan) and the difference between the two TPSs (vTPS Dose) could be studied independently.

1.6.ii. Post Delivery QA

During the delivery of an IMRT field, the MLC controller creates a log of several linac parameters that is saved as an MLC dynalog file. For step and shoot IMRT fields this file contains a record, in 50ms intervals, of the position of the linac's mechanical components including the individual MLC leaves and fraction of the total monitor units delivered per field. This file contains the majority of the information needed to reconstruct the treatment delivery but lacks information about the total monitor units delivered and the treatment couch angle. However, these two pieces of information are recorded in the DICOM message sent from the treatment console to the R&V system after the delivery saved as a DICOM RT-Record file. Information from the MLC dynalog files

and the RT-Record files is combined to reconstruct a plan that represents each treatment as the delivery was recorded in these two files.

The delivered plan is imported into the clinical TPS where dose is recomputed on the patient CT dataset. The differences between the clinical dose distribution and the dose distribution derived from the delivered plan represent not only the data transfer errors but also the uncertainties introduced by the allowed variations in the linac performance during the delivery.

The treatment records used to construct the delivered plan are only created after the treatment has been physically delivered by the linac. It is possible to perform this QA prior to the patient's first treatment, but this involves delivering the treatment prior to the patient's first treatment. This would increase the time required to perform the QA to the level of patient-specific measurements. We propose that the for traditionally fractionated patients the post-delivery QA be performed after the patient's first treatment, for patients only be treated with a few fractions pre-treatment QA may be necessary to minimize the probability that an error or uncertainties will not have clinical consequences

An advantage of this QA system is that it is possible to QA any or all treatments delivered. This type of analysis is beyond the scope of this work.

1.6.iii. Increased Machine-Specific QA

In the proposed system the accuracy of the delivery verification depends on the accuracy treatment unit's digital readouts for dose and component positions. These readouts are verified by at least 2 treatment unit subsystems and are carefully checked periodically during machine-specific QA. Some aspects, such as dosimetric output, are checked daily. While it is reasonable to assume that the existing checks are adequate, the routine machine-specific QA may need to be increased to ensure the accuracy of the delivery record if such an IMRT QA system were to be adopted. While

this will increase the time needed to perform machine-specific QA of the treatment unit, it would still be more time efficient than making patient-specific measurements.

1.7. Hypothesis and Specific Aims

The hypothesis of this study was: A calculation- based IMRT QA system that uses the communications between components as input into a second TPS is sensitive enough to detect errors in data transfer between components as well as dose uncertainties caused by delivery uncertainties or calculations at levels similar to (<3%) or superior to current measurement-based methods.

The specific aims of this study are:

-To characterize the error contribution from data transfer and interpretation errors from the treatment planning system to the record and verify system.

- To characterize the contribution of the combination of data transfer errors and delivery uncertainty

- To characterize the differences in the calculated dose distributions due the difference between the two dose calculation algorithms and heterogeneity corrections.

-To compare reconstructed dose at each step to clinical IMRT QA measurements

2 Methods and Materials

2.1. The Pinnacle^3 Treatment Planning System

At our institution the all photon radiotherapy treatments are planned with the Pinnacle^3 v9 TPS (Philips Medical Systems, Fitchburg, WI, USA) employing the CCC algorithm. All patients undergo a CT simulations prior to treatment planning, and patient-specific CT-based inhomogeneity corrections are applied to every clinical dose calculation.

Pinnacle^3 organizes its data in a hierarchical manner. The top level is the Institution which contains a set of patient objects and a set of treatment machine definitions that characterize the beam models. Each patient object consists of CT simulations and/or diagnostic imaging dataset and one or more plan objects. Each plan object references one CT dataset for dose calculation, but other CT datasets can be used to help define targets and critical structures. Plan objects have one or more Trial objects which contain a collection of treatment fields which Pinnacle^3 labels as beams. Modulated beams are broken into beam segments each of which share jaw settings, treatment couch angle, gantry angle, and collimator angle, but have unique MLC positions. We will refer to these Pinnacle^3 object definitions throughout this work.

When a plan is opened in Pinnacle^3 a graphical user interface (GUI) is created that allows the user to easily access the trials, beams, and segments saved in the plan. The contents of the GUI are created by executing a file written in Pinnacle^3's proprietary scripting language that is created every time the plan is saved. Custom operations can also be carried out by executing files written in this language (Script files) using Pinnacle^3's HotScript interface. Pinnacle^3 scripting lacks some of the capabilities provided by other programming languages, but the Pinnacle^3 language can be used to call, pass information to, and receive instructions from code written in other languages.

For the purposes of this study a Pinnacle^3 Script files are directly to manipulate trial and beam objects as well as being used to spawn software written in the Python programming language

(Python v2.3.3, Python Software Foundation, Wolfeboro Falls, NH, USA). Python is freely downloadable software with an open source license. Python is part of the standard Solaris v10 operating system (Oracle Corporation, Redwood Shores, CA, USA) installed on the Pinnacle^3 server used in this study.

2.2. Patient Selection

This study includes 48 patients who received step-and-shoot IMRT treatment at our institution. All clinical dose calculations were performed using the Pinnacle^3 TPS employing CT-base, patient-specific inhomogeneity corrections. Measurement-based IMRT QA consisting of an ionization chamber measurement and radiographic film measurement were performed for each patient. These patients were selected from all patients who received treatment to prostate (19), head and neck (14), gynecologic (8), or thoracic (7) treatment sites. These patients were selected by capturing treatments on two separate treatment units on a given day excluding VMAT treatments.

2.3. Data Transfer Verification

To study the errors induced by the transfer of data from the TPS through the R&V system to the treatment console, the DICOM RT-Plans generated as instructions to the treatment console for treatment were used as input into the original TPS to allow dose calculations which were compared to the original dose distributions. The treatment plans and dose distributions created through this process were labeled as RV plans and RV dose distributions, respectively.

Pinnacle^3 v9 does not support the direct importation of a DICOM RT-Plan file that includes modulation, so software was developed to accomplish this. This software is a collection of Python commands that read the plan information from the RT-Plan file. This information was used to create a Pinnacle^3 Script that was executed when a patient's plan was open. This Script then copied the clinical trial so that the new trial had the same parameters (i.e. dose grid, couch removal, etc.) as the clinical trial. The Script then removed all of the beams from the new trial, and created new

beams and beam segments using the information from the RT-Plan file. The RV dose distribution was generated by calculating dose for the new trial using the same CCC dose algorithm and inhomogeneity corrections used to compute the clinical dose distribution.

Each RV dose distribution was compared to the corresponding clinical dose distribution using 3D gamma analysis (Section 2.6) and ROI dose comparisons (Section 2.7). These comparisons characterize the effect of data transfer an interpretation errors.

2.4. Treatment Delivery Verification

The actual parameters of the treatment delivery were captured in DICOM RT-Record files that were saved by the R&V system and MLC dynalog files created by the MLC controller. Together these files represent the treatment records. The RT-Record files contain patient identification information, the number of monitor units delivered, and a treatment time stamp as well as the gantry angle, collimator angle, couch angle, and relative weight of each MLC segment. A large amount of information is contained in the MLC dynalog files.

2.4.i. MLC dynalog files

For each delivered field two MLC dynalogs are created; one dynalog for each MLC carriage. These files include the values of 254 different parameters recorded every 50ms during beam delivery. The dynalog information used in this study was gantry angle, collimator angle, jaw positions, position of each MLC leaf, an MLC segment descriptor, a beam-on-state flag, and the fractional dose delivered.

The gantry and collimator angles are given in tenths of a degree. The jaw positions are given in mm project to the plane of isocenter. The positions of the MLC leaves are given in hundredths of mm in the physical plane of the MLCs. Pinnacle^3 use the MLC position projected to the plan of isocenters, so the recorded MLC positions must scaled in order to be imported into a treatment plan. Varian provides a ratio of the source-to-isocenter distance to the source-to-MLC distance of

1.96078 [54] and this value is used to scale the leaf positions from the dynalog file. This was confirmed by delivering a simple IMRT field in which the MLC leaves traveled to predefined positions. The resulting dynalog file information was then imported into a spreadsheet and the leaf positions were extracted compared to the planned position of each leaf to determine to verify that the conversion factor gave accurate positions.

The MLC dynalog records data every 50ms while the beam is being delivered. In a step and shoot MLC delivery for part of this time the beam is off and the MLC leaves are moving. This study is only concerned with the data recorded while each segment is being delivered. MLC dynalog files contain a segment descriptor, so it, along with the beam on/off flag, was used to define beam segments. The beam-on-state flag is a binary value that is equal to 1 when the beam is on and equal to 0 when the beam is off. The weight of each segment was determined from the fractional dose delivered. The fractional dose delivered is captured as an integer with values from 0 to 25,000. A value of 25,000 corresponds to 100% of the MUs delivered for that beam. A more detailed description of the MLC dynalog format is available from Varian Medical Systems [55].

2.4.ii. Combining RT-Record and MLC dynalog Information

To verify the correct delivery of the treatments, the information in the treatment records was used as input into the original TPS. The treatment plans and dose distributions created through this process were labeled as Tx plans and Tx dose distributions, respectively. There is no way to directly import these files into a TPS, so software was created to accomplish this.

A Python command is used to read all of the RT-Record files for a given patient for a given treatment to obtain data needed to tie the beam record to the dynalog data. This data includes the patient's medical record number, the patient's name, the number of beams delivered, the treatment date, and beam specific information. The beam specific information includes the gantry

angle, collimator angle, treatment couch angle, set dose rate, photon energy, jaw positions, beam name, and total monitor units delivered. The Python command reads each pair of dynalog files for a given patient for a given treatment to obtain data needed to tie the dynalog data to the beam record. This data includes the patient's last name, medical record number, and number of beams, and beam specific information. The gantry angle, collimator angle, and jaws position are also recorded in these files and are recorded at a higher precision than in the RT-record file. The values of theses parameters was taken to be the average of all values recorded in both dynalogs files for that beam to average out uncertainty in these readouts that should not change during step and shoot IMRT.

For each segment, the positions of all 120 MLC leaves are obtained by averaging the recorded position of each leaf over the segment. The segment weight can be determined by calculating the change in the dose fraction delivered from the start of the segment to the end of the segment and then dividing by 25,000. For each beam the total segment weights should equal 1. The accuracy of the segment weights determined from the MLC dynalog files was verified by making sure that the total segment for each beam was equal to 1 and by comparing to the clinical segment weights.

After the individual beam information has been identified and matched between the RT-record and dynalog files, a Pinnacle^3 Script file is created by the python script that will, when executed, create a trial corresponding to the recorded delivery of the treatment plan using the gantry angle, collimator angle, jaw positions, and segment information from the dynalog files and the couch angle, photon energy, set dose rate, MU, and beam weight information from the RT-Record files.

When the Pinnacle^3 Script file is executed with the patient's plan open, it first copies the clinical trial to maintain consistent settings of the dose grid. The Script then removes all of the beams from the new trial, and creates new beams and beam segments using the information from the treatment records. The Tx dose distribution is then generated by computing the dose for the new

trial using the same CCC dose algorithm and inhomogeneity corrections used to compute the clinical dose distribution. The Tx dose distribution was compared to the clinical dose distribution using the 3D gamma analysis (Section 2.6) and ROI dose comparisons (Section 2.7). These comparisons will characterize the effect of data transfer and interpretation errors and recorded delivery uncertainties.

2.5. TPS Comparison

To understand the differences between the Pinnacle^3 and Eclipse TPSs, the original treatment plan and patient CT dataset were transferred to the Eclipse planning system. The treatment plans and dose distributions created through this process were labeled as vTPS plans and vTPS dose distributions, respectively.

The plan parameters and CT dataset were exported from Pinnacle^3 as DICOM RT-Plan and CT files. These files were imported into Eclipse where dose was recomputed using AAA applying CT-based inhomogeneity corrections. The dose distribution was then exported from Eclipse as a DICOM RT-Dose file. Pinnacle^3 v9 has no means from importing a DICOM-RT dose distribution, so code was written to perform this task. The code consists of two parts: a program written in MATLAB (The MathWorks Inc., Natick, MA, USA) that reads the actual dose grid information from the RT-Dose file and saves it as a binary file with “Big Endian” byte order and a Python command that reads the header information from the RT-Dose file and builds a Pinnacle^3 Script that will create a new trial with a single un-modulated beam prescribed to deliver 1 MU. The Script set the dimensions, voxel size, and origins of the trial dose grid to match the dose grid defined in the RT-Dose file. The Script then computes dose to this dummy beam, and Saves the plan. When the plan is saved several binary files are saved in the patient object which corresponds to the dose distribution from each beam. The Script then replaces the binary files containing the dose computed for the dummy beam with the binary file saved by the MATLAB command, and then

closes the plan without saving it. When the plan is reopened the vTPS dose distribution will displayed as the dose delivered from the dummy beam.

The vTPS dose distribution was compared to the clinical dose using the 3D gamma analysis (Section 2.6) and ROI dose comparisons (Section 2.7) to characterize the difference between the two TPSs.

2.6. 3D Gamma Analysis

Quantifying the differences between two 3D dose distributions can be difficult. The distributions often contain sharp dose gradients, and different dose levels have different levels of importance.

One method of comparing two dose distributions is to calculate the difference in dose at corresponding points in each distribution. In smoothly varying dose distributions, dose-difference methods can be relied upon to identify unacceptable dose differences between the two distributions. However, in regions of high dose gradients small difference in the position of the gradient will result in large, but clinically insignificant, dose differences. These steep dose gradients are typical in IMRT[56] and, in fact, are one of the motivations for using IMRT.

The distance-to-agreement (DTA) is another metric for comparing two dose distributions. For each voxel in the standard dose distribution, the DTA is the minimum distance to a voxel in the measured distribution with the same dose. The DTA method performs well in regions with high dose gradients, but performs poorly in low-dose regions of the dose distribution. Just as a large dose difference in an area of a sharp dose gradient may not be clinically significant , a large DTA in a low dose region may not be clinically significant[56].

Gamma analysis was introduced as a way to combine the strengths of the dose-difference and the DTA methods while avoiding their weaknesses [57]. In determining the agreement of a given point (S) in a standard distribution, the gamma analysis evaluates the difference between that point

and the dose at points (T) in a test distribution using a dose difference criteria (dD) that decreases as a function of the distance between S and T ($\left\|\overrightarrow{\mathbf{r}_{S,T}}\right\|$) .

Gamma analysis begins by computing a gamma value ($\Gamma_{S,T}$) between every point S and every point T. $\Gamma_{S,T}$ combines the spatial distance between points S and T ($\left\|\overrightarrow{\mathbf{r}_{S,T}}\right\|$) and the differences between the dose at point S and T (D_S and D_T) by adding them in quadrature. The two values are normalized by dose distance (dr) and dose difference (dD) criteria, respectively. Low et al. [57] presented gamma analysis for n-dimensional arrays, but formulated it to compare two-dimensional dose distributions. Here it is extended to three spatial dimensions by expanding $\overrightarrow{\mathbf{r}_{S,T}}$ from a 2D vector to a 3D vector. The value of $\Gamma_{S,T}$ between a point S and a point T is given by:

$$\Gamma_{S,T} = \sqrt{\left(\frac{\left\|\overrightarrow{\mathbf{r}_{S,T}}\right\|}{dr}\right)^2 + \left(\frac{D_S - D_T}{dD}\right)^2} \tag{2.1}$$

If for a given point S, there exists at least one point T for which $\Gamma_{S,T}$ is less than or equal to 1, then point S is said to pass the gamma analysis. Otherwise, it is said to fail. This pass/fail criteria leads to the definition of a quality index (γ_S) for each point S that represents the minimum of the set of $\Gamma_{S,T}$ for all T.

$$\gamma_S = min\{\Gamma_{S,T}\}\forall\{T\} \tag{2.2}$$

The result of the gamma analysis between two distributions is reported as the percentage of analyzed points for which γ_S is pass (i.e. $\gamma_S \leq 1$).

It can be seen from Equation 2.1, above that $\Gamma_{S,T}$ cannot pass (i.e. $\Gamma_{S,T} \leq 1$) if $\left\|\overrightarrow{\mathbf{r}_{S,T}}\right\|$ is greater than dr. This property was exploited to reduce the computation time for each gamma analysis by only computing values $\Gamma_{S,T}$ between voxels for which $\left\|\overrightarrow{\mathbf{r}_{S,T}}\right\|$ is less than dr. Also, the value of $\Gamma_{S,T}$ was only

calculated for points in the standard distribution for which the test dose distribution extended at least dr in every direction along the grid axes.

For each dose distribution, a dose grid was defined using the coordinate system defined by the patient's simulation CT dataset. Dose calculations in the primary TPS took place on a 3mm x 3mm x 3mm dose grid. The dose calculations done in the secondary TPS took place on a 2.5mm x 2.5mm x 2.5mm dose grid.

Software was written to perform the 3D gamma analysis using the binary dose grid files from the Pinnacle[3] as the input. To speed up the gamma calculations the clinical dose grid was interpolated to 1mm x 1mm x 1mm spacing using a linear interpolation and maintaining the origin defined in the Pinnacle. The verification plans were also interpolated to 1mm grid spacing with an extra step required to match location of the grid points of the clinical distribution. Both distributions were then normalized to the maximum dose value in the clinical distribution. The gamma analysis was performed using 3 different sets of dD and dr criteria: 3% and 3mm, 2% and 2mm, and 1% and 1mm.

For every patient in this study, 3D gamma analysis was performed between the clinical dose distribution the RV, Tx, and vTPS dose distributions.

2.7. ROI Dose Comparisons

Part of the treatment planning process is to segment the patients planning CT into a number of ROIs. ROIs provide a means of simplifying the interpretation of 3D dose distributions by allowing the dose to be evaluated in clinically significant regions. Since ROIs are intended to represent a treatment target or normal anatomy, Section 1.1.iv, the dose deposited in the ROI volume represents the dose received by a target or normal structure during the treatment.

ROI based dose metrics can also be used to compare dose distributions. This study examined the calculated doses to 121 treatment target volumes from 48 patients. For each ROI, the mean ROI

dose and a specific ROI dose level from the RV, Tx, and vTPS dose distributions were compared to those calculated by the clinical dose distributions.

The specific ROI dose level was based on the type of treatment volume: the dose received by 100% of the voxels (D100) for GTVs, the dose received by 99% of the voxels (D99) for CTVs and ITVs, and the dose received by 95% of the voxels (D95) for PTVs.

2.8. IMRT QA Measurement Comparisons

Our institution currently uses measurement-based IMRT QA that includes an absolute dose measurement using an ionization chamber, described in Section 1.5.i . For each patient in this study, hybrid plans were created in the Pinnacle^3 TPS for the clinical, RV, and Tx plans and in Eclipse for the vTPS plan. To determine how the proposed method compares to the current measurement-based method the clinical absolute dose measurement for each patient studied were compared to the doses predicted by the clinical, RV, Tx, and vTPS plans.

Film planes from the hybrid dose distributions were also exported and compared to file measurements for a randomly selected subset of patients representing at least 2 patients from each treatment site. The dose planes from the clinical, RV, Tx, and vTPS hybrid dose distributions were compare to radiographic film measurements using the a 2D gamma comparison software (OmniPro-I'mRT, IBA Dosimetry, Schwarzenbruck, Germany). The gamma criteria for these comparisons were 5% and 3mm, which are the same criteria used to evaluate clinical IMRT QA film measurements.

3 Results

3.1. Software Performance and Verification

The software written for this study performed with minimal errors. The software used to translate information into a Pinnacle readable format did so in a matter of seconds. The 3D gamma software was able to perform 3D gamma calculations at the 3 criteria in about 10 to 30 minutes per comparison. The 3D gamma software is a Matlab function running on a 64-bit server with 8 processors and 8GB of RAM. The speed of the 3D gamma software is limited by the size of the dose grids being compared.

The software that translated the RV delivery instruction and the RT-Record DICOM files into a Pinnacle readable format were verified by performing a manual parameter check between the resulting Pinnacle plans and the original files for a few random patients. The gantry angles, collimator angles, couch angles, jaw positions, monitor units, and MLC leaf positions were compared between the R&V system and the trial created by the software. No discrepancies were observed.

The conversion of the dynalog files-into a Pinnacle trial was also verified by spot checking leaf positions for a few fields on a few patients. The segment weighting was verified by ensuring the weights added to 1 in addition to spot checking against the planned delivery.

Verification of the data transfer to the 2nd TPS was also done by spot checking. Since the 2nd TPS accepted the DICOM import with no modifications this was only done as a precaution. Verification of the RT-Dose import from the vTPS was done by computing a few simple field geometries, and ensuring that they were imported into Pinnacle in the right position with the correct orientation and values.

The 3D gamma analysis software was verified by performing 3D gamma calculations comparing a dose distribution to itself and verifying the agreement was perfect (all voxels pass gamma). It

was also verified by comparing simple dose distributions and verifying that the results were as expected.

3.2. 3D Gamma Analysis

For each patient the 3D gamma analysis was performed between the RV, Tx, and vTPS plans and the clinical plan for three sets of gamma criteria: 3%/3mm, 2%/2mm and 1%/1mm. This study examined the mean percentage of voxels passing the 3D gamma comparisons (Table 3.1,Table 3.2, and Table 3.3). The distribution of 3D gamma results (Figure 3.1Figure 3.1 Mean Percentages of Voxels passing a 3D gamma comparison of the RV, Tx, and vTPS plans and the clinical plans.) was also studied with particular attention paid to the distribution of the 1% and 1mm 3D gamma results (Figure 3.2).

3D Gamma Comparison: RV to Clinical			
Site (n)	**Mean % Voxels Passing (St.Dev.)**		
	3% - 3mm	**2% - 2mm**	**1% - 1mm**
all sites (48)	100.0 (0.0)	100.0 (0.0)	100.0 (0.1)
GU (19)	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
HN (14)	100.0 (0.0)	100.0 (0.0)	99.9 (0.1)
GYN (8)	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
THOR (7)	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)

Table 3.1 Results of the 3D gamma comparisons between the RV and Clinical Plans. The average percetage of voxels passing the three sets of gamma criteria for 48 patients.

3D Gamma Comparison: Tx to Clinical			
Site (n)	Mean % Voxels Passing (St.Dev.)		
	3% - 3mm	2% - 2mm	1% - 1mm
all sites (48)	100.0 (0.0)	100.0 (0.0)	99.0 (1.4)
GU (19)	100.0 (0.0)	100.0 (0.0)	99.6 (0.9)
HN (14)	100.0 (0.0)	100.0 (0.1)	98.3 (1.8)
GYN (8)	100.0 (0.0)	100.0 (0.0)	98.2 (1.2)
THOR (7)	100.0 (0.0)	100.0 (0.0)	99.6 (0.5)

Table 3.2 Results of the 3D gamma comparisons between the Tx and Clinical Plans. The average percetage of voxels passing the three sets of gamma criteria for 48 patients.

3D Gamma Comparison: vTPS to Clinical			
Site (n)	Mean % Voxels Passing (St.Dev.)		
	3% - 3mm	2% - 2mm	1% - 1mm
all sites (48)	99.3 (0.6)	97.2 (1.5)	79.0 (8.6)
GU (19)	99.7 (0.0)	97.8 (0.7)	79.8 (5.7)
HN (14)	98.6 (0.7)	96.0 (1.3)	78.6 (7.2)
GYN (8)	99.4 (0.3)	96.4 (1.6)	69.7 (11.0)
THOR (7)	99.7 (0.2)	98.7 (0.6)	88.2 (3.9)

Table 3.3 Results of the 3D gamma comparisons between the vTPS and Clinical Plans. The average percetage of voxels passing the three sets of gamma criteria for 48 patients.

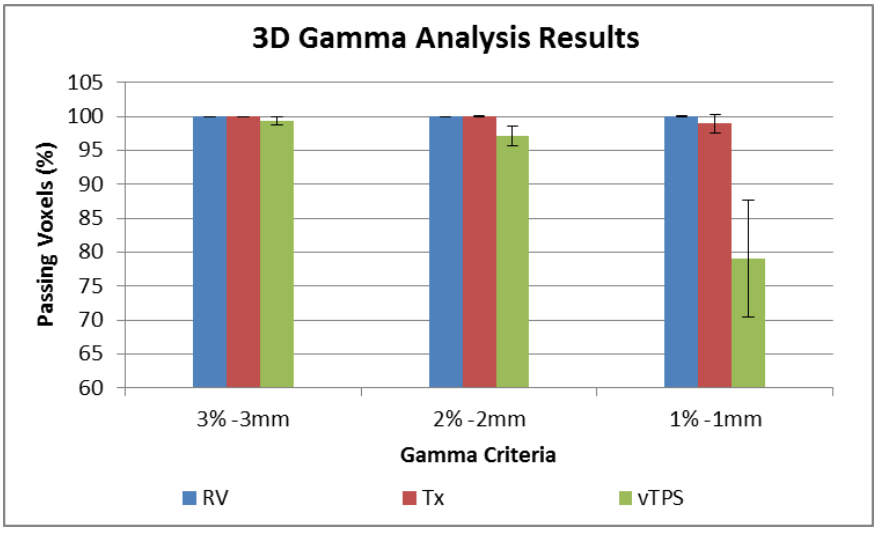

Figure 3.1 Mean Percentages of Voxels passing a 3D gamma comparison of the RV, Tx, and vTPS plans and the clinical plans. The error bars are ± 1 standard deviation.

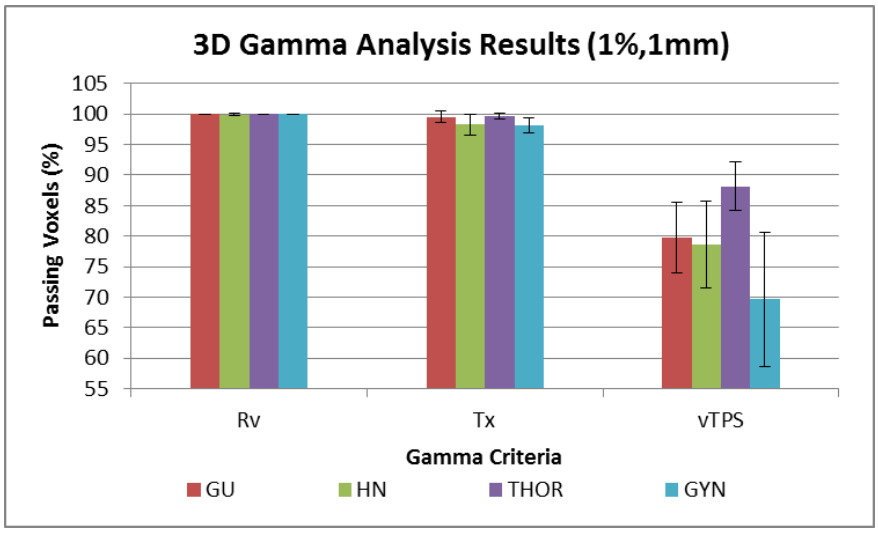

Figure 3.2 The mean percentage of voxels passing the 3D gamma comparison of the RV, Tx, and vTPS plans to the clinical plan with gamma criteria of 1% and 1mm. The data is separated by treatment site. The error bars are ± 1 standard deviation.

3.3. ROI Dose Comparisons

For each patient the mean dose and ROI-specific dose levels were evaluated for treatment volumes. The ROI-specific dose levels are discussed in Section 2.7. This study examined the average ratio of the mean dose and specific dose levels of the RV, Tx, and vTPS plans to that of the clinical plan for 48 patients (Table 3.5, Table 3.4, and Table 3.6). The distribution of the mean dose (Figure 3.3) and specific dose level (Figure 3.4) comparison results was also studied.

Treatment Volume Dose Ratio: RV / Clinical				
	Mean ROI Dose		Specific ROI Dose Levels	
Site (n)	Avg. Ratio	St. Dev.	Avg. Ratio	St. Dev.
all sites (121)	0.999	0.001	0.999	0.001
GU (27)	0.999	0.001	0.999	0.001
HN (55)	0.999	0.002	0.999	0.002
GYN (20)	0.999	0.001	1.000	0.001
THOR (19)	1.000	0.000	1.000	0.000

Table 3.4 The average ratio of the Mean ROI Doses and type-specific ROI levels is reported for 121 treatment volumes from 48 patients.

Treatment Volume Dose Ratio: Tx / Clinical				
	Mean ROI Dose		Specific ROI Dose Levels	
Site (n)	Avg. Ratio	St. Dev.	Avg. Ratio	St. Dev.
all sites (121)	1.001	0.002	1.001	0.002
GU (27)	1.001	0.001	1.001	0.001
HN (55)	1.001	0.003	1.001	0.003
GYN (20)	1.001	0.002	1.001	0.002
THOR (19)	1.001	0.000	1.001	0.001

Table 3.5 The average ratio of the Mean ROI Doses and type-specific ROI levels is reported for 121 treatment volumes from 48 patients.

Treatment Volume Dose Ratio: vTPS / Clinical				
	Mean ROI Dose		Specific ROI Dose Levels	
Site (n)	Avg. Ratio	St. Dev.	Avg. Ratio	St. Dev.
all sites (121)	0.990	0.009	0.980	0.043
GU (27)	0.985	0.005	0.975	0.007
HN (55)	0.993	0.009	0.987	0.062
GYN (20)	0.993	0.007	0.980	0.017
THOR (19)	0.983	0.007	0.969	0.020

Table 3.6 The average ratio of the Mean ROI Doses and type-specific ROI levels is reported for 121 treatment volumes from 48 patients.

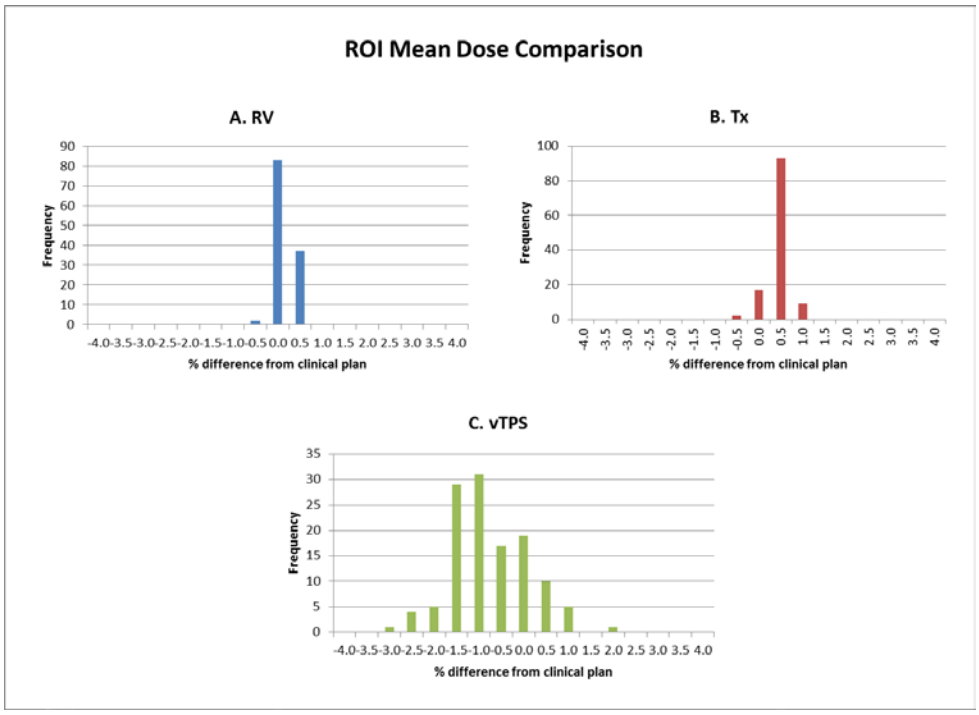

Figure 3.3. A histogram of the ROI mean dose comparisons between the (A) RV, (B) Tx, and (C) vTPS plans and the clinical plan. A total of 121 ROIs from 48 patients were analyzed.

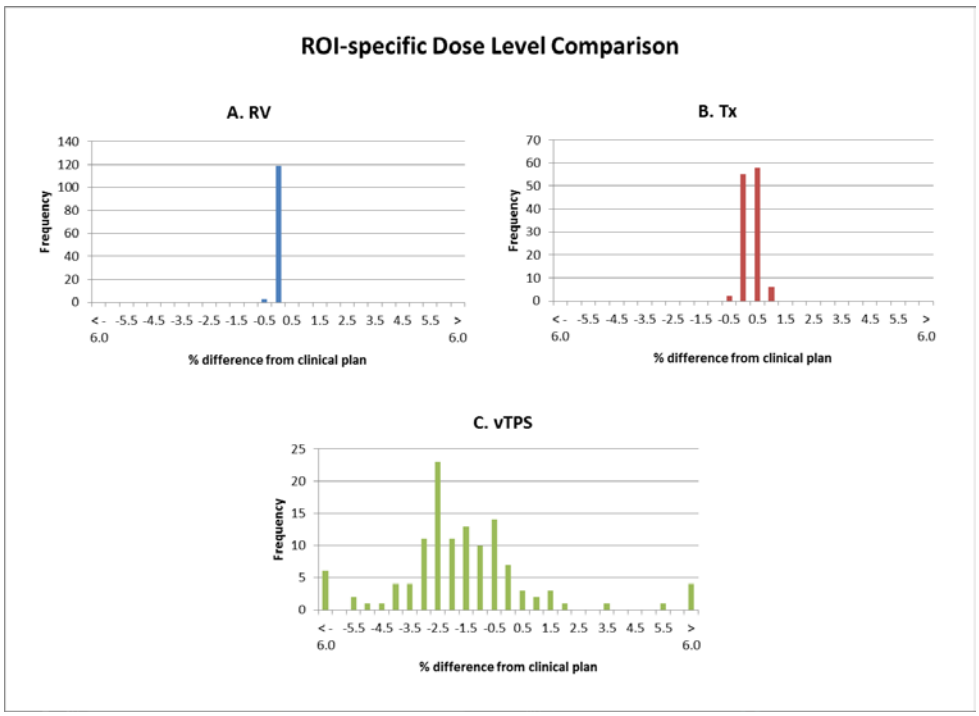

Figure 3.4 A histogram of the ROI-specific dose level comparisons between the clinical plan and the (A) RV, (B)Tx, and (C) vTPS plans and the clinical plan. A total of 121 ROIs from 48 patients were analyzed.

3.4. IMRT QA Measurement Comparisons

To compare calculated to measured absolute dose for each type of treatment plan we compared to clinical IMRT QA absolute dose measurements (ionization chamber measurements) to dose calculated with each system. This was done for 48 patients by copying each plan to a CT-dataset of the IMRT QA phantom to create hybrid plans (see Section 1.5.i). For each plan the percent difference between the measured ionization chamber dose and the average calculated dose to the ionization chamber volume was examined (Table 3.7, Figure 3.5).

To compare calculated to measured relative dose for each type of treatment plan we also compare the IMRT QA film measurement to the corresponding dose plane in the hybrid pan using a 2D gamma comparison and the clinical gamma criteria of 5% and 3mm. We randomly selected 9 patients representing all treatment sites: prostate (3), head and neck (2), gynecologic (2), and thoracic (2). This study examined the number of pixels passing the 2D gamma comparison (Table 3.8).

Site (n)	Comparison to Absolute Dose Measurement Mean Ratio to Measurement (St. Dev.)			
	Clinical	RV	Tx	vTPS
All (48)	0.999 (0.013)	0.998 (0.013)	0.999 (0.013)	0.990 (0.012)
GU (19)	0.996 (0.011)	0.996 (0.011)	0.996 (0.011)	0.988 (0.012)
HN (14)	0.998 (0.014)	0.997 (0.014)	0.998 (0.015)	0.990 (0.013)
GYN (8)	1.001 (0.011)	1.000 (0.010)	1.001 (0.011)	0.996 (0.011)
THOR (7)	1.005 (0.017)	1.005 (0.017)	1.005 (0.017)	0.992 (0.012)

Table 3.7 Mean ratios of the dose calculated by the Clinical, RV, Tx, and vTPS plans to the measured absolute dose measured during IMRT QA.

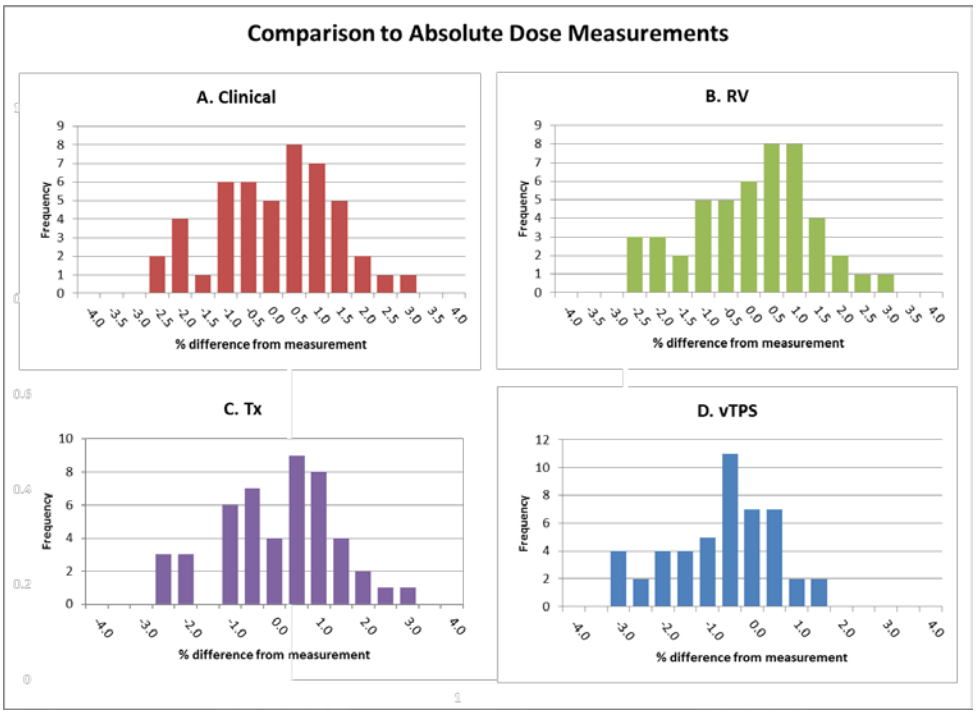

Figure 3.5 Histograms of the results of the comparison of the IMRT QA absolute dose measurement to the (A) clinical plans, (B) RV plans, (C) Tx plans, and (D) vTPS plans

2D Gamma Comparison to Film Measurement Percentage of Pixels Passing Gamma (5% 3mm)				
Patient	**Clinical**	**RV**	**Tx**	**vTPS**
GU1	99.940	99.940	97.45	99.930
GU14	99.830	99.830	99.68	99.980
GU19	98.390	98.410	99.39	97.060
GYN2	99.780	99.780	99.87	99.330
GYN8	99.210	99.440	98.41	95.130
HN10	98.080	99.070	98.96	99.710
HN11	99.520	99.520	98.67	98.520
THOR4	99.930	99.930	99.92	100.000
THOR7	99.600	99.610	99.81	99.210
Average	99.36	99.50	99.13	98.76
St. Dev.	0.68	0.49	0.84	1.66

Table 3.8 Results of the 2D gamma comparisons of the IMRT QA film measurements to the dose distribution calculated by the Clinical, RV, Tx, and vTPS plans using the clinical gamma criteria of 5% and 3mm.

4 Discussion

4.1. Software Performance and Verification

The software written for this study showed that it could perform well in the proposed IMRT QA system (Figure 1.3A) with no disruption to clinical workflow and can be largely automated. If automation of the vTPS software could be achieved then this system could potentially perform QA checks on each treatment delivery automatically and identify when a plan or treatment unit may need further evaluation.

The speed and memory requirements of the 3D gamma calculation may be a limiting factor in large facilities, but there these could be improved by moving to enterprise class hardware as well as by improving the efficiency of this software.

4.2. Delivery Instructions

Modern networks employ sophisticated tools (checksums, cyclic redundancy checks, etc.) to reduce or eliminate data loss or corruption during network communications, so differences between the RV and clinical plans should be hard to detect. However, differences were seen between the RV and clinical plans. At the strictest 3D gamma criteria of (1%, 1mm) the RV comparisons for 6 of 48 patients did not yield perfect results (passing % = 100%). When comparing the ROI-specific and ROI mean doses between 121 treatment volumes from the RV and clinical plans, the dose from the RV plans averaged 0.1% (σ=0.1%) different.

The differences seen in this study prompted further investigation that led to the discovery of the MLC positioning issue. For the purposes of this study, the clinical Pinnacle plan for each patient was copied to a Pinnacle research institution. Dose was then recomputed using a specific beam model in the research institution. It was discovered that in some cases when the beam model was changed from the clinical model to the research model some of the MLC leaf positions changed

by 0.5mm. We believe that these MLC shifts are the cause of the small differences seen between the RV and clinical plans. The fact that this system was able to detect such small differences in MLC position is promising.

When compared to the clinical plans, the RV plans show better agreement than the Tx and vTPS plans. In the proposed IMRT QA system (Figure 1.3A) the RV and vTPS arms of this study are mixed. The proposed clinical QA system would combine the RV plan and the vTPS plan, it is unlikely that compensating errors and /or uncertainties at each step would hide real problems.

4.3. Delivery Verification

If there were no uncertainty associated with treatment delivery the differences between the Tx and clinical plans would be the same as between the RV and clinical plans, because any errors in the RV plans will be propagated to the Tx plans. However, like any electro-mechanical device, the treatment unit parameters are allowed to vary within pre-defined limits defined by the vendor, published QA guidelines (e.g. TG-142 [38]), and/or in-house specifications.

Although the average ratio of the RV and Tx plan ROI doses to the clinical ROI dose were both within ±1% the standard deviation of the Tx results ($\sigma = 0.2\%$) was larger than the standard deviation of the RV results ($\sigma = 0.1\%$). The difference between the Tx and RV distributions can be attributed to variation in the treatment delivery. The good agreement between the Tx and clinical plans is an indication that the tolerances currently defined for the performance of the treatment unit are adequate.

4.4. TPS Comparison

When compared to the clinical plans the vTPS plans had larger differences than the RV or Tx plans. These differences are due to the differences between the two TPSs used in this study. As discussed in Section 1.1 the two TPSs used in this study (Pinnacle and Eclipse) used different dose

calculation algorithms (CCC and AAA, respectively) that account for tissue inhomogeneity in different ways as well as differences in the model of the linear accelerator.

The proposed system does not rely on perfect agreement between the two TPSs, but rather seeks to use the vTPS to perform an independent dose calculation. This second dose calculation will serve to verify that the calculations made by the clinical TPS are reasonable and ensure the data integrity of the clinical system. This step is not a replacement for acceptance testing of the clinical TPS but serves as an ongoing check which meets established QA guidelines for an independent calculation [42-44].

5 Conclusions

In this study, data transfer errors, treatment delivery uncertainty, and the difference between the clinical and verification TPSs were characterized. The data transfer and treatment delivery errors were studied by using information from the R&V system and the treatment console to recompute dose distributions representing the delivery instructions to the treatment console and the delivered treatment. The differences between the TPSs were studied by exporting the clinical plan information to the vTPS, computing the vTPS dose based on this information, and exporting the dose distribution back to the clinical TPS for comparison. This was performed using clinically approved treatment plans representing 19 prostate, 14 head and neck, 8 gynecologic, and 7 thoracic IMRT treatments.

The hypothesis of this study was: A calculation- based IMRT QA system that uses the communications between components as input into a second TPS is sensitive enough to detect errors in data transfer between components as well as dose uncertainties caused by delivery uncertainties or calculations at levels similar to (<3%) or superior to current measurement-based methods.

In this study we found that, with no explicit errors, the differences between the clinical and the RV and Tx plans are very small (<0.5%) allowing us to use very tight action levels for detection of actual errors. The observed change due to the 0.5 mm random MLC leave error in Pinnacle was easily picked up by this system even though it had no measurable clinical consequence. The difference between the RV and Tx plans showed the system can pick up small, even if insignificant, delivery uncertainties. This is far superior to our current, measurement based, QA system.

This system would have to be used in tandem with rigorous machine-specific QA to ensure that the recorded treatment unit parameters are consistent with the actual treatment unit performance. In particular the absolute dose rate of the machine would need to be checked on a regular basis as well as the electronic readouts for various positions as the ability of this system to verify the treatment delivery is directly affected by the accuracy of the recorded parameters.

This system has the potential to be entirely automated, and perform QA of the plan before the first treatment and after every treatment delivery. Rather than perform measurements on every patient, measurements could be reserved for those cases identified as outliers. This system would reduce the time, labor, and material costs of patient-specific IMRT QA by limiting the number of patient-specific measurements. This could lower the cost of delivering IMRT enabling this to become standard-of-care for all disease sites.

6 Appendix A: Pinnacle .Script building software

dictPinn.py

```
#!/usr/bin/env python
import os

arcsegmentlen = 4 # in degrees


# LOCAL directories
ptdir = 'C:/PinnacleImport/Patients/'
scriptdir = 'C:/Projects/Pinn-Thesis/Scripts/'
locgammadir = 'C:/Projects/Pinn-Thesis/gammatmp/'
locgammatmp = os.path.join(locgammadir,'gamma.tmp')
holdingdir = 'C:/Projects/Pinn-Thesis/datatmp/'


# LOCAL files
noBeamsLog = 'C:/Projects/Pinn-Thesis/Patients/NoBeams.log'
errorlog = 'C:/Projects/Pinn-Thesis/Patients/Errors.log'

# PINNACLE directories
pinnscriptdir = '/home/p3rtp/ohrt/PtImport/'
pinngammadir = '/home/p3rtp/ohrt/gammatmp/'


# dictionaries
manuf = { 'Pinnacle3':'pinn', 'clinical':'clinical', 'Aria
RadOnc':'eclipse','MOSAIQ':'mosaiq','2300IX':'4DTC_Tx', '2100EX':'4DTC_Tx', '2100C/D':'4DCT',
'AcQSimCT':'', 'On-Board Imager': '4DTC_Tx' }
type = { 'RTPLAN':'RP' , 'RTDOSE':'RD' , 'RTSTRUCT':'RS' , 'RTRECORD':'RT', 'RTIMAGE': 'RI' , 'CT':'CT' }
mach = {'Varian 2109':'Varian 2109: 2010-11-22 10:23:13'}
mode = { 'STATIC': 'SnS', 'DYNAMIC':'ARC'}
edom = { 'SnS':'Step & Shoot MLC' , 'ARC':'Dynamic Arc'}

#filter for RT-Record last name, to get rid of QA RT-Records
QAname = ['zzz','zzOBI']
```

DataManager.py

```
#!/usr/bin/env python

import sys
import glob
import os
import time
import ftplib
import datetime

import dicom
import fget
import PinnScribe
from PinnScribe import MismatchError

from dictPinn import ptdir

from dictPinn import holdingdir
from dictPinn import scriptdir
from dictPinn import pinnscriptdir
from dictPinn import errorlog

from dictPinn import manuf
from dictPinn import mode
from dictPinn import type
from dictPinn import QAname

wantftp=0
wantscribe=1

class PathError(Exception):
   def __init__(self, value):
      self.value = 'ERROR!!! : ' + value + ' is not a valid file or directory path'
   def __str__(self):
      return repr(self.value)

class break_try(Exception):
   def __init__(self):
      self.value = 'Just want the try statement fail to except'
   def __str__(self):
      return repr(self.value)

def go(dirorfile=holdingdir, wantftp=0, wantscribe=1):
   if (os.path.isdir(dirorfile)):
      txfiles = dir(dirorfile,'*',[],wantscribe,0,)
      if wantscribe:
```

```
            for pt in range(len(txfiles[0])):
               txscribe = txfiles[2][pt]*txfiles[3][pt]
               if txscribe:
                  try:
                     PinnScribe.txrecords(txfiles[0][pt],txfiles[1][pt])
                  except:
                     ErrorLog(txfiles[0][pt],sys.exc_info(),'TXRecords')

         if wantftp: ftp2pinn(scriptdir)

      elif (os.path.isfile(dirorfile)):
         newname , dummy = processfile(dirorfile,[], wantscribe)
         if wantftp: ftp2pinn(newname)

      else:
         raise PathError(dirorfile)


def dir(fdir=holdingdir, filter='*' ,txfilegrps=[],wantscribe=1, rmdir=0):
      filter = str(filter)
      if (str(filter).find('*') == -1):
         filter = '*' + str(filter) +'*'

      if (os.path.isdir(fdir) == 0):
         fdir = fget.dir()

      if (len(txfilegrps) == 0):
         txfilegrps = [range(0),range(0),range(0),range(0)]

      ############ move dynalogs files to patient folder
      for flie in glob.glob(os.path.join(fdir,'*.dlg')):
         f = open(flie,'r')
         data=f.read().splitlines()
         f.close()
         data = data[1].split(',')
         lastname = data[0][0].upper() + data[0][1:].lower()
         lastname = lastname.replace(' ', '')
         mrn =  str(flie[-10:-4])

         ptfolder = os.path.join(ptdir,str(flie[-10:-4] + lastname))
         filename = os.path.basename(flie)
         txfolder = 'Tx_' + filename[1:5] + '-' + filename[5:7] + '-' + filename[7:9]
         newname = os.path.join(ptfolder,txfolder,filename)


         if ((mrn in txfilegrps[0]) == 0):
            txfilegrps[0].append(mrn)
            txfilegrps[1].append(os.path.join(ptfolder,txfolder          txfilegrps[2].append(0)
```

```
            txfilegrps[3].append(1)
        else:
            for i in range(len(txfilegrps)):
                if mrn == txfilegrps[0]:
                    txfilegrps[3][i] = 1

        if (os.path.isdir(ptfolder) == 0):
            os.mkdir(ptfolder)
        ptfolder = os.path.join(ptfolder,txfolder)
        if (os.path.isdir(ptfolder) == 0):
            os.mkdir(ptfolder)
        newname = rename(flie,newname)

    ############process dicoms
    filenames = glob.glob(os.path.join(fdir,filter))
    for file in filenames:
        if os.path.isdir(file):
            txfilegrps = dir(file,'*',txfilegrps,wantscribe,1) #### will look in all subdirectories
        else:
            newname, txfilegrps = processfile(file, txfilegrps, wantscribe)
    try:
        if rmdir: os.rmdir(fdir)
    except:
        pass

    return txfilegrps

def file(file='none'):
    if (os.path.isfile(file) == 0):
        print "Not a valid file: ",file,'\n\t ASKING for new file'
        file=fget.file()
    processfile(file)

def processfile(file,filegrps=[],wantscribe='True'):
    print 'Processing: ',file
    dcm = dicom.read_file(file)
    MRN = dcm.PatientID.replace(' ','_')
    try: ## Comprehensive RTRECORD doesn;t have Instance Creation Date, (that's ok because it's
useless)
        InstanceCreationDate = dcm.InstanceCreationDate
    except:
        InstanceCreationDate = dcm.StudyDate

    LastName = dcm.PatientsName.split('^')[0]
    LastName = LastName.replace('restored', '')
    LastName = LastName.replace(' ', '')
    LastName = LastName[0].upper() + LastName[1:].lower()
```

```
    Type = type[dcm.Modality]
    #### define filename for all types
    if (Type != 'CT'):
        Creator = manuf[dcm.ManufacturersModelName]
        filename = Type + '.' + MRN + '.' + LastName + '.' + Creator + '.' + InstanceCreationDate + '.dcm'
    else:
        ctnm = file[-9:]
        ctnm = ctnm[ctnm.find('.')+1:]
        filename = Type + '.' + MRN + '.' + LastName + '.' + InstanceCreationDate + '.' + ctnm

    #### create patient folder if it doesn't exists
    ptfolder = os.path.join(ptdir,str(MRN + LastName))
    if (os.path.isdir(ptfolder) == 0):
        os.mkdir(ptfolder)

   #### define newfile name
   newname = os.path.join(ptfolder,filename)


    #### If CT slice
    if (Type =='CT'):
        ptfolder = os.path.join(ptfolder,str('CT_' + dcm.InstanceCreationDate))
        if (os.path.isdir(ptfolder) == 0):
            os.mkdir(ptfolder)
        newname = os.path.join(ptfolder,filename)
        newname = rename(file,newname)

  #### If treatment Record
    elif (Type =='RT'):
        if (Creator == 'mosaiq'):
            os.remove(file)
        elif (LastName.find('Zzzobi') != -1):
            os.remove(file)
        else:
            txfolder = 'Tx_' + dcm.TreatmentDate[0:4] + '-' + dcm.TreatmentDate[4:6] + '-' +
dcm.TreatmentDate[6:8]
            ptfolder = os.path.join(ptfolder,txfolder)
            if (os.path.isdir(ptfolder) == 0):
                os.mkdir(ptfolder)

            if ((MRN in filegrps[0]) == 0):
                filegrps[0].append(MRN)
                filegrps[1].append(ptfolder)
                filegrps[2].append(1)
                filegrps[3].append(0)
            else:
                for i in range(len(filegrps[0])):
                    if (MRN == filegrps[0][i]):
```

```
                filegrps[2][i] = 1

        try:
            bnum = 'Beam_ ' + str(dcm.TreatmentSessionBeams[0].RefdBeamNumber)
            filename = Type + '.' + MRN + '.' + LastName + '.' + dcm.TreatmentDate + '.' + bnum + '.dcm'

            newname = os.path.join(ptfolder,filename)
            newname = rename(file,newname)
        except:
            os.remove(file)
            print '\t--> deleted plan with No Beams:', file

#### If BEV image
    elif (Type == 'RI'):
        os.remove(file)
        print '\t--> deleted RI file'

 #### If Plan
    elif (Type == 'RP'):
        Mode = mode[dcm.Beams[0].BeamType]
        if (Creator == 'mosaiq'):
            newname = rename(file,newname)
            try:
                if wantscribe:
                    PinnScribe.plan(newname, Mode)

            except:
                ErrorLog(file,sys.exc_info(),'PLAN')
                newname = rename(newname,file)

        else:
            newname = rename(file,newname)

 #### If dose grid
    elif (Type == 'RD'):
        newname = rename(file,newname)
        try:
            if wantscribe:
                PinnScribe.dose(newname)

        except:
            ErrorLog(file,sys.exc_info(),'DOSE')
            newname = rename(newname,file)


    else:
        newname = rename(file,newname)
```

```
    return newname, filegrps

def ErrorLog(loc, error='UNKNOWN ERROR',info=''):
    if (len(error) != 3):
        errmess = 'UNKNOWN ERROR: '+ str(info) + ':' + str(loc) + '\n'
    else:
        errmess = str(error[0]).split('.')[-1][:-2] + '(' + str(error[1]) + '): '+ str(info) + ':' + str(loc) + '\n'

    print errmess
    err=open(errorlog,'a')
    err.write(errmess)
    err.close()

    return errmess

def rename(file,newname):
    while os.path.exists(newname):
        if ( os.path.basename(newname)[0:2] == 'CT'):
            newname='duplicate CT file was deleted'
            break
        else:
            q=newname.split('.')
            try:
                n= int(q[-2])
                if (n <= 99): ### if this is first duplicate q[-2]=Created Date(yyyymmdd)
                    q[-2] = str(n+1)
                else:
                    raise break_try()
            except:
                n = 2
                q.append(q[-1])
                q[-2] = str(n)

            newname = '.'.join(q)

    print'\t -->',newname
    if (newname == 'duplicate CT file was deleted'):
        os.remove(file)
    else:
        os.rename(file,newname)

    return newname
```

PinnScribe.py

```
#/usr/bin/env python
import os
import glob
import subprocess
import time
import shutil
import fget
import dcminterp
from dcminterp import NoBeams
from dictPinn import noBeamsLog
from BinaryRW import writePinnDose
import dynreader
import dicom


from dictPinn import ptdir
from dictPinn import scriptdir
from dictPinn import pinnscriptdir
from dictPinn import manuf
from dictPinn import edom


class MismatchError(Exception):
   def __init__(self, value):
      self.value =  'The numbers of dynalogs and treatment records do not match: ' + value
   def __str__(self):
      return repr(self.value)

class NoMode(Exception):
   def __init__(self, value):
      self.value =  'Mode improperly defined:' + value
   def __str__(self):
      return repr(self.value)

class BeamRecStruct(object):
   def __init__(self,dcmbm,dynbm):
      self.Gantry = dynbm.Gantry
      self.Collimator = dynbm.Collimator
      self.Jaws = dynbm.Jaws
      self.NumberofSegments = dynbm.NumberofSegments
      self.Segment = dynbm.Segment
      self.Name = str(dcmbm.Name)
      self.FieldID = dcmbm.FieldID
      self.Couch = dcmbm.Couch
      self.Energy = dcmbm.Energy
      self.DoseRate = dcmbm.DoseRate
```

```
        self.MU = dcmbm.MU
        self.Weight = dcmbm.Weight

class CombinedRecStruct(object):
    def __init__(self,dcm):
        self.LastName = dcm.PatientName[0]
        self.MRN = dcm.MRN
        self.Mode = dcm.Mode
        self.TxDate = dcm.TxDate
        self.NumberofFractions = dcm.NumberofFractions
        self.TotalMU = dcm.TotalMU
        self.Beam = range(dcm.NumberofBeams)

########################################################

def plan(file='none',type = 'none'):
    if (file == 'none'):
        file = fget.file()

    if (type == 'none'):
        raise NoMode(type)

    #try:
    dcm=dcminterp.readPlan(file)
    #except NoBeams:
    #    print "NO BEAMs"
    #    return "NO BEAMs

    scriptname = scriptdir + str(os.path.basename(file).replace('RP', type + 'plan')).replace('.dcm',
'.Script')
    ptcopy = os.path.join(os.path.dirname(file), os.path.basename(scriptname))
    tmp = open(scriptname,'w')


    ###############Create Isocenters
    ISOlist = {}
    for iso in range(1,len(dcm.Isocenters)):
        if (len(dcm.Isocenters[iso]) == 4):
            tmp.write('CreateNewPOI = "Add Point Of Interest";\n')
            tmp.write('PoiList .Last .Name = "ImportedISO_' + dcm.Isocenters[iso][0] + '";\n')
            tmp.write('PoiList .Last .MakeCurrent = 1;\n')
            tmp.write('PoiList .Last .PoiInterpretedType = "ISOCENTER";\n')
            tmp.write('PoiList .Last .RelativeXCoord = "' + str(dcm.Isocenters[iso][1]) + '";\n');
            tmp.write('PoiList .Last .RelativeYCoord = "' + str(dcm.Isocenters[iso][2])+ '";\n');
            tmp.write('PoiList .Last .RelativeZCoord = "' + str(dcm.Isocenters[iso][3]) + '";\n');
            ISOlist[str(iso)] = str('ImportedISO_'+ dcm.Isocenters[iso][0])
        ISOlist[str(iso)] = str('ImportedISO_'+ dcm.Isocenters[iso][0])
```

```
   tmp.write('CopyCurrentTrial = "Copy Current Trial";\n')
   tmp.write('TrialList .Last .MakeCurrent = "1";\n')
   tmp.write('TrialList.Last .BeamList.DestroyAllChildren = "";\n')
   tmp.write('TrialList.Last.PrescriptionList.DestroyAllChildren = "";\n')
   tmp.write('TrialList.Last = {\n')
   tmp.write('  Name = "Plan_Import (' + dcm.Vendor + ')";\n')

   ################ Patient Representation
   tmp.write('  PatientRepresentation ={\n')
   tmp.write('    CtToDensityTableAccepted = 1;\n')
   tmp.write('  };\n')

   ##############  Prescription
   tmp.write('  PrescriptionList ={\n')
   tmp.write('    Prescription ={\n')
   tmp.write('      Name = "ImportedRx";\n')
   tmp.write('      RequestedMonitorUnitsPerFraction = ' + str(dcm.TotalMU) + ';\n')
   tmp.write('      PrescriptionDose = 200;\n')
   tmp.write('      PrescriptionPercent = 100;\n')
   tmp.write('      NumberOfFractions = ' + str(dcm.NumberofFractions) + ';\n')
   tmp.write('      Method = "Set Monitor Units";\n')
   tmp.write('      NormalizationMethod = "Max Dose";\n')
   tmp.write('      PrescriptionPeriod = "Per Fraction";\n')
   tmp.write('      WeightsProportionalTo = "Monitor Units";\n')
   tmp.write('    };\n')
   tmp.write('  };\n')

   ###############BeamList
   tmp.write('  BeamList ={\n')
   ###############Beam
   for beam_i in range(len(dcm.Beam)):
      tmp.write('    Beam ={\n')
      tmp.write('      Name = "' + str(dcm.Beam[beam_i].Name) + '";\n')

      if (len(dcm.Isocenters[int(dcm.Beam[beam_i].IsocenterIndex)]) == 4): # if isocenter was given
         tmp.write('      IsocenterName = "'+ str(ISOlist[dcm.Beam[beam_i].IsocenterIndex]) +'";\n')
         tmp.write('      PrescriptionName = "ImportedRx";\n')
         tmp.write('      UsePoiForPrescriptionPoint = 1;\n')
         tmp.write('      PrescriptionPointName = "'+ str(ISOlist[dcm.Beam[beam_i].IsocenterIndex])
+'";\n')
         #tmp.write('      PrescriptionPointName = "ImportedISO_1";\n')
      else:
         tmp.write('      PrescriptionName = "ImportedRx";\n') # included this to keep order the same
probably could remove from loop
      tmp.write('      MachineNameAndVersion = "Varian 2110: 2011-07-15 17:01:07";\n')
      tmp.write('      Modality = "Photons";\n')
      tmp.write('      MachineEnergyName = "' + dcm.Beam[beam_i].Energy + '";\n')
      tmp.write('      SetBeamType = "' + edom[dcm.Beam[beam_i].Mode] + '";\n')
```

```
    tmp.write('     CPManager ={\n')
    tmp.write('       CPManagerObject ={\n')
    tmp.write('         IsGantryStartStopLocked = 1;\n')
    tmp.write('         IsCouchStartStopLocked = 1;\n')
    tmp.write('         IsCollimatorStartStopLocked = 1;\n')
    tmp.write('         IsLeftRightIndependent = 1;\n')
    tmp.write('         IsTopBottomIndependent = 1;\n')
    tmp.write('         NumberOfControlPoints = ' + str(dcm.Beam[beam_i].NumSegments   ) + ';\n')
    tmp.write('         ControlPointList ={\n')

    ###########Control Points
    for cp in range(int(dcm.Beam[beam_i].NumSegments)):
       tmp.write('           #' + str(cp) + ' ={\n')
       tmp.write('             Gantry = ' + str(dcm.Beam[beam_i].Segment[cp].Gantry) + ';\n')
       tmp.write('             Couch = ' + str(dcm.Beam[beam_i].Couch) + ';\n')
       tmp.write('             Collimator = ' + str(dcm.Beam[beam_i].Collimator) + ';\n')
       tmp.write('             WedgeContext ={\n')
       tmp.write('               WedgeName = "No Wedge";\n')
       tmp.write('               Orientation = "NoWedge";\n')
       tmp.write('               OffsetOrigin = "Patient Surface";\n')
       tmp.write('               OffsetDistance = -2.5;\n')
       tmp.write('               Angle = "No Wedge";\n')
       tmp.write('               MinDeliverableMU = 0;\n')
       tmp.write('               MaxDeliverableMU = 1e+30;\n')
       tmp.write('             };\n')
       tmp.write('             LeftJawPosition = ' + str(dcm.Beam[beam_i].Jaws.X1) + ';\n')
       tmp.write('             RightJawPosition = ' + str(dcm.Beam[beam_i].Jaws.X2) + ';\n')
       tmp.write('             TopJawPosition = ' + str(dcm.Beam[beam_i].Jaws.Y2) + ';\n')
       tmp.write('             BottomJawPosition = ' + str(dcm.Beam[beam_i].Jaws.Y1) + ';\n')
       tmp.write('             ModifierList ={\n')
       tmp.write('             };\n')
       tmp.write('             MLCLeafPositions ={\n')
       tmp.write('               RawData ={\n')
       tmp.write('                 NumberOfDimensions = 2;\n')
       tmp.write('                 NumberOfPoints = 60;\n')
       tmp.write('                 Points[] ={\n')

       ########### MLC Leaf Positions
       for leaf in range(60):
          tmp.write('                   ' + str(dcm.Beam[beam_i].Segment[cp].B_LeafPositions[leaf]) + ',' +
str(dcm.Beam[beam_i].Segment[cp].A_LeafPositions[leaf]) + ',\n')

       tmp.write('                 };\n')
       tmp.write('               };\n')
       tmp.write('             };\n')
       tmp.write('             Weight = ' + str(dcm.Beam[beam_i].Segment[cp].SegmentWeight) + ';\n')
       #if (dcm.Beam[beam_i].Mode != 'ARC'): tmp.write('             WeightLocked = 1;  \n')
```

```
        if (dcm.Beam[beam_i].Mode == 'ARC'): tmp.write('            PercentOfArc = ' +
str(dcm.Beam[beam_i].Segment[cp].PercentofArc) + ';\n')
        tmp.write('          };\n')

    ######### Beam continued
    tmp.write('        };\n')
    tmp.write('      };\n')
    tmp.write('    };\n')
    tmp.write('    UseMLC = 1;\n')
    tmp.write('    Display2d = 0;\n')
    tmp.write('    Display3d = 0;\n')

    ######## Dose Engine
    tmp.write('    DoseEngine ={\n')
    tmp.write('      TypeName = "CC Convolution";\n')
    tmp.write('      ConvolveHomogeneous = 0;\n')
    tmp.write('      FluenceHomogeneous = 0;\n')
    tmp.write('      FlatWaterPhantom = 0;\n')
    tmp.write('      FlatHomogeneous = 0;\n')
    tmp.write('    };\n')

    tmp.write('    DisplayList ={\n')
    tmp.write('      #0 ={\n')
    tmp.write('        Name = "Blocked Field";\n')
    tmp.write('        OnOff2d = "Off";\n')
    tmp.write('      };\n')
    tmp.write('      #2 ={\n')
    tmp.write('        Name = "Open Field";\n')
    tmp.write('        OnOff2d = "Off";\n')
    tmp.write('      };\n')
    tmp.write('      #5 ={\n')
    tmp.write('        Name = "Central Axis Crosshair";\n')
    tmp.write('        Is2dDashOn = 0;\n')
    tmp.write('        OnOff2d = "On";\n')
    tmp.write('      };\n')
    tmp.write('    };\n')


    ######### Monitor Units (at Beam level, NOT under MonitorUnitInfo)
    tmp.write('    Weight = ' + str(dcm.Beam[beam_i].Weight) + ';\n')
    #tmp.write('    IsWeightLocked = 1;\n')
    tmp.write('    FieldID = "' + str(dcm.Beam[beam_i].FieldID) + '";\n')
    tmp.write('    DoseRate = ' + str(dcm.Beam[beam_i].DoseRate) + ';\n')
    tmp.write('  };\n')

  tmp.write('  };\n')
  tmp.write('};\n')
  ######## Select new trial in DVH
```

```
   tmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .Selected = 1;\n')
   tmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .LineType = "Medium Dashed";\n')
   ####### Turn on Isodose Lines
   tmp.write('IsodoseControl .LineList .#"*" .Display2dOn = "1";\n')
   ###### Change Beam Color
   tmp.write('TrialList .Current .BeamList .ChildrenEachCurrent .#"@".TrialList .Current .BeamList
.Current. Color = "tomato";\n')
   tmp.write('TrialList .Current .ComputeUncomputedBeams = "1";\n')
   tmp.close()

   shutil.copyfile(scriptname,ptcopy)

def txrecords(ptid=[], dir='none'):
   #ptid = str(ptid)
   if len(ptid):
      ptid = str(ptid)
   if (os.path.isdir(dir) == 0):
      dir = fget.dir(ptdir)

   dcm = dcminterp.readBeamRecords(ptid, dir)
   dyn = dynreader.read(dir, ptid)


   if (dyn.NumberofBeams != dcm.NumberofBeams):
      raise MismatchError(dir)
   else:
      nbeams = dyn.NumberofBeams
   tx = CombinedRecStruct(dcm)

   dyntmp = dyn;

   for rt in range(nbeams):
      RT =
[round(dcm.Beam[rt].Gantry,0),round(dcm.Beam[rt].Collimator,0),round(dcm.Beam[rt].Jaws.X1,0),
round(dcm.Beam[rt].Jaws.X2,0),round(dcm.Beam[rt].Jaws.Y1,0),round(dcm.Beam[rt].Jaws.Y2,0)]
      if (RT[0] == 360): RT[0] = 0.0
      if (RT[1] == 360): RT[1] = 0.0
      #print 'RT[',rt,'] =',RT

      for dy in range(len(dyn.Beam)):
         DY = [round(dyn.Beam[dy].Gantry,0),
round(dyn.Beam[dy].Collimator,0),round(dyn.Beam[dy].Jaws.X1,0),
round(dyn.Beam[dy].Jaws.X2,0), round(dyn.Beam[dy].Jaws.Y1,0), round(dyn.Beam[dy].Jaws.Y2,0)]
         if (DY[0] == 360): DY[0] = 0.0
         if (DY[1] == 360): DY[1] = 0.0
         #print '\tDY[',dy,'] =',DY
         if (RT == DY):
            #print 'Match for rt =',rt,' at dy=',dy
```

```
            tx.Beam[rt] = BeamRecStruct(dcm.Beam[rt],dyn.Beam[dy])
            del dyn.Beam[dy]
            break


    mismatch = []
    for b in range(nbeams):
        if (type(tx.Beam[b]) == type(int())):
            mismatch.append(b)

    if len(mismatch):
        raise MismatchError(dir)

    scriptfile = 'TXQA.' + str(tx.MRN) + '.' + tx.LastName[0].upper() + tx.LastName[1:].lower().replace('
','') + '.' + str(tx.TxDate) + '_' + tx.Mode + '.Script'
    scriptname = os.path.join(scriptdir, scriptfile)
    ptcopy = os.path.join(os.path.dirname(dir), scriptfile)

    writeRec(tx,scriptname)

    shutil.copyfile(scriptname,ptcopy)
    return dcm, dyn, tx

def writeRec(tx,scriptname):  ####def writeARC(tx,scriptname):   <---OLD

    tmp = open(scriptname,'w')

    #######Check Image UID would be cool

    ###### Copy curent Trial set the first beams iso as current POI, then destryo all beams. This
makes
    ######  sure that the dose grid and isocenters should be the same.
    tmp.write('Store.StringAt.CnrtTrial = TrialList. Current. Name;\n')
    tmp.write('Store.FreeAt.ISO = "";\n')
    tmp.write('Store.StringAt.ISO = TrialList.Current.BeamList.Current.PrescriptionPointName;\n')
    #tmp.write('Store.StringAt.CnrtTrial = TrialList. Current. Name;\n')

    tmp.write('Store.StringAt.ISO = TrialList.Current.BeamList.Current.IsocenterName;\n')
    tmp.write('PoiList.Current = Store.At.ISO.String;\n')
    tmp.write('Store.At.newTrial.AppendString = Store.At.CnrtTrial.String;\n')
    tmp.write('CopyCurrentTrial = "Copy Current Trial";\n')
    if (tx.Mode == 'ARC'):tmp.write('TrialList .Last .BeamList .SortBy .D .Gantry .MU = "";\n') # sort
beam list by start gantry then MU(descending).
    if (tx.Mode == 'SnS'):tmp.write('TrialList.Last.BeamList.DestroyAllChildren = "";\n')
    if (tx.Mode == 'SnS'):tmp.write('TrialList.Last.PrescriptionList.DestroyAllChildren = "";\n')

    tmp.write('TrialList .Last .MakeCurrent = 1;\n')
    tmp.write('TrialList.Last = {\n')
```

```
   tmp.write('  Name = "Tx Update (' + tx.TxDate[-2:] + '-' + tx.TxDate[-4:-2] + '-' + tx.TxDate[:4] +
')";\n')

   ################ Patient Representation
   tmp.write('  PatientRepresentation ={\n')
   tmp.write('    CtToDensityTableAccepted = 1;\n')
   tmp.write('    CtToDensityTableExtended = 1;\n')
   tmp.write('  };\n')

     ##############  Prescription
   tmp.write('  PrescriptionList ={\n')
   tmp.write('    Prescription ={\n')
   tmp.write('      Name = "TxUpdate_Rx";\n')
   tmp.write('      RequestedMonitorUnitsPerFraction = ' + str(tx.TotalMU) + ';\n')
   tmp.write('      PrescriptionDose = 200;\n')
   tmp.write('      PrescriptionPercent = 100;\n')
   tmp.write('      NumberOfFractions = ' + str(tx.NumberofFractions) + ';\n')
   tmp.write('      Method = "Set Monitor Units";\n')
   tmp.write('      NormalizationMethod = "Max Dose";\n')
   tmp.write('      PrescriptionPeriod = "Per Fraction";\n')
   tmp.write('      WeightsProportionalTo = "Monitor Units";\n')
   tmp.write('    };\n')
   tmp.write('  };\n')



   ###############BeamList
   tmp.write('  BeamList ={\n')
   ###############Beam
   for beam_i in range(len(tx.Beam)):
      tmp.write('    Beam ={\n')
      tmp.write('      Name = "' + str(tx.Beam[beam_i].Name) + '";\n')
      tmp.write('      PrescriptionName = "TxUpdate_Rx";\n')
      tmp.write('      UsePoiForPrescriptionPoint = 1;\n')
      #tmp.write('      PrescriptionPointName = Store.At.ISO.String;\n')
      tmp.write('      MachineNameAndVersion = "Varian 2110: 2011-07-15 17:01:07";\n')
      tmp.write('      Modality = "Photons";\n')
      tmp.write('      MachineEnergyName = "' + tx.Beam[beam_i].Energy + '";\n')
      tmp.write('      SetBeamType = "' + edom[tx.Mode] + '";\n')
      tmp.write('      CPManager ={\n')
      tmp.write('        CPManagerObject ={\n')
      tmp.write('          IsGantryStartStopLocked = 1;\n')
      tmp.write('          IsCouchStartStopLocked = 1;\n')
      tmp.write('          IsCollimatorStartStopLocked = 1;\n')
      tmp.write('          IsLeftRightIndependent = 1;\n')
      tmp.write('          IsTopBottomIndependent = 1;\n')
      tmp.write('          NumberOfControlPoints = ' + str(tx.Beam[beam_i].NumberofSegments   ) +
';\n')
```

```
    tmp.write('        ControlPointList ={\n')

    ###########Control Points
    for cp in range(int(tx.Beam[beam_i].NumberofSegments)):
      tmp.write('          #' + str(cp) + ' ={\n')
      tmp.write('            Gantry = ' + str(tx.Beam[beam_i].Gantry) + ';\n')
      tmp.write('            Couch = ' + str(tx.Beam[beam_i].Couch) + ';\n')
      tmp.write('            Collimator = ' + str(tx.Beam[beam_i].Collimator) + ';\n')
      tmp.write('            WedgeContext ={\n')
      tmp.write('              WedgeName = "No Wedge";\n')
      tmp.write('              Orientation = "NoWedge";\n')
      tmp.write('              OffsetOrigin = "Patient Surface";\n')
      tmp.write('              OffsetDistance = -2.5;\n')
      tmp.write('              Angle = "No Wedge";\n')
      tmp.write('              MinDeliverableMU = 0;\n')
      tmp.write('              MaxDeliverableMU = 1e+30;\n')
      tmp.write('            };\n')
      tmp.write('            LeftJawPosition = ' + str(tx.Beam[beam_i].Jaws.X1) + ';\n')
      tmp.write('            RightJawPosition = ' + str(tx.Beam[beam_i].Jaws.X2) + ';\n')
      tmp.write('            TopJawPosition = ' + str(tx.Beam[beam_i].Jaws.Y2) + ';\n')
      tmp.write('            BottomJawPosition = ' + str(tx.Beam[beam_i].Jaws.Y1) + ';\n')
      tmp.write('            ModifierList ={\n')
      tmp.write('            };\n')
      tmp.write('            MLCLeafPositions ={\n')
      tmp.write('              RawData ={\n')
      tmp.write('                NumberOfDimensions = 2;\n')
      tmp.write('                NumberOfPoints = 60;\n')
      tmp.write('                Points[] ={\n')

      ############ MLC Leaf Positions
      for leaf in range(60):
        tmp.write('                  ' + str(tx.Beam[beam_i].Segment[cp].B_LeafPositions[59-leaf]) + ',' +
str(tx.Beam[beam_i].Segment[cp].A_LeafPositions[59-leaf]) + ',\n')

      tmp.write('                };\n')
      tmp.write('              };\n')
      tmp.write('            };\n')
      if (tx.Mode != 'ARC'): tmp.write('            Weight = ' +
str(tx.Beam[beam_i].Segment[cp].SegmentWeight) + ';\n') # can't get segemnt weight from ARC
dynalogs
      #tmp.write('            WeightLocked = 1;  \n')
      tmp.write('          };\n')

    ######### Beam continued
    tmp.write('        };\n')
    tmp.write('      };\n')
    tmp.write('    };\n')
    #tmp.write('    UseMLC = 1;\n')
```

```
        tmp.write('     Display2d = 0;\n')
        tmp.write('     Display3d = 0;\n')

        ######## Dose Engine
        tmp.write('     DoseEngine ={\n')
        tmp.write('       TypeName = "CC Convolution";\n')
        tmp.write('       ConvolveHomogeneous = 0;\n')
        tmp.write('       FluenceHomogeneous = 0;\n')
        tmp.write('       FlatWaterPhantom = 0;\n')
        tmp.write('       FlatHomogeneous = 0;\n')
        tmp.write('     };\n')

        tmp.write('     DisplayList ={\n')
        tmp.write('       #0 ={\n')
        tmp.write('         Name = "Blocked Field";\n')
        tmp.write('         OnOff2d = "Off";\n')
        tmp.write('       };\n')
        tmp.write('       #2 ={\n')
        tmp.write('         Name = "Open Field";\n')
        tmp.write('         OnOff2d = "Off";\n')
        tmp.write('       };\n')
        tmp.write('       #5 ={\n')
        tmp.write('         Name = "Central Axis Crosshair";\n')
        tmp.write('         Is2dDashOn = 0;\n')
        tmp.write('         OnOff2d = "On";\n')
        tmp.write('       };\n')
        tmp.write('     };\n')

        ######### Monitor Units (at Beam level, NOT under MonitorUnitInfo)
        tmp.write('     Weight = ' + str(tx.Beam[beam_i].Weight) + ';\n')
        #tmp.write('     IsWeightLocked = 1;\n')
        #if (tx.Mode != 'ARC'): tmp.write('     PercentOfArc = ' +
str(tx.Beam[beam_i].Segment[cp].PercentofArc) + ';\n')
        tmp.write('     FieldID = "' + str(tx.Beam[beam_i].FieldID) + '";\n')
        tmp.write('     DoseRate = ' + str(tx.Beam[beam_i].DoseRate) + ';\n')
        tmp.write('   };\n')

    tmp.write('  };\n')
    tmp.write('};\n')
    ####### Select new trial in DVH
    tmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .Selected = 1;\n')
    tmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .LineType = "Medium Dashed";\n')
    ####### Turn on Isodose Lines
    tmp.write('IsodoseControl .LineList .#"*" .Display2dOn = "1";\n')
    ###### Change Beam Color
    tmp.write('TrialList .Last .BeamList .ChildrenEachCurrent .#"@".TrialList .Last .BeamList .Current.
Color = "tomato";\n')
    tmp.write('TrialList .Last .ComputeUncomputedBeams = "1";\n')
```

```
    tmp.close()

def dose(file):
    file = file.replace('\\','/')
    dose=dicom.read_file(file)
    Y = dose.Rows
    X = dose.Columns
    Z = dose.NumberofFrames
    mrn = dose.PatientID
    lastname = dose.PatientsName.split('^')[0]
    lastname = lastname.replace('restored', '')
    lastname = lastname.replace(' ', '')
    lastname = lastname[0].upper() + lastname[1:].lower()

    creator = manuf[dose.ManufacturersModelName]
    dosefactor = dose.DoseGridScaling

    scriptfile = 'DoseGrid_' + str(mrn) + lastname + '_' + creator + '.Script'
    dosefile = 'DoseGrid_' + str(mrn) + lastname + '_' + creator + '.binary'


    scriptname = os.path.join(scriptdir,scriptfile)
    localdosename = os.path.join(scriptdir, dosefile)
    pinndosename =  os.path.join(pinnscriptdir, dosefile)
    ptscriptcopy = os.path.join(os.path.dirname(file), scriptfile)
    ptdosecopy = os.path.join(os.path.dirname(file), dosefile)

    writePinnDose(file,localdosename)


    ##### create and write Script
    tmp = open(scriptname,'w')

    tmp.write('TrialList .CreateChild = "Add New Trial";\n')
    tmp.write('TrialList .Last .Name = "Dose_Import ('+ creator +')";\n')
    tmp.write('TrialList .Last .MakeCurrent = "1";\n')
    tmp.write('TrialList .Current .PatientRepresentation .CtToDensityTableAccepted = "1";\n')

    #### Set Dose Grid
    tmp.write('TrialList .Current .DoseGrid .VoxelSize .X = ' + str(dose.PixelSpacing[0]/10) + ';\n')


    tmp.write('TrialList .Current .DoseGrid .VoxelSize .Y = ' + str(dose.PixelSpacing[1]/10) + ';\n')
    zspace = dose.GridFrameOffsetVector[1]- dose.GridFrameOffsetVector[0]
    tmp.write('TrialList .Current .DoseGrid .VoxelSize .Z =' + str(zspace/10)+ ';\n')
    tmp.write('TrialList .Current .DoseGrid .Dimension .X = ' + str(dose.Columns) + ';\n')
    tmp.write('TrialList .Current .DoseGrid .Dimension .Y = ' + str(dose.Rows) + ';\n')
    tmp.write('TrialList .Current .DoseGrid .Dimension .Z = ' + str(dose.NumberofFrames) + ';\n')
```

```
   if '1.2.840.113619.2.55.3.1871670999.11859.1251222934.417.1829.0.1' ==
dose.FrameofReferenceUID:
      #### then Dose is from I'mRT Body Phantom in Eclipse. Adjust dosegrid origin by the location
      ####  of the Alignment Point.  Pinn(-0.17,-50.12,51.12) = Eclipse(0,0,0)
      ###LAT = ((dose.ImagePositionPatient[0] + ((dose.Columns-1)*dose.PixelSpacing[0]))/-10)-0.17
      print 'Check LAT Settings'
      LAT = (dose.ImagePositionPatient[0]/-10)-0.17
      AP = ((dose.ImagePositionPatient[1] + ((dose.Rows-1)*dose.PixelSpacing[1]))/-10)-50.12
      tmp.write('TrialList .Current .DoseGrid .Origin .Y = ' + str(AP) + ';\n')
      SI = ((dose.ImagePositionPatient[2] + ((dose.NumberofFrames-1)*zspace))/-10)+51.12
   else:
      LAT = dose.ImagePositionPatient[0]/10
      AP = (dose.ImagePositionPatient[1] + ((dose.Rows-1)*dose.PixelSpacing[1]))/-10
      SI = (dose.ImagePositionPatient[2] + ((dose.NumberofFrames-1)*zspace))/-10


   ##tmp.write('TrialList .Current .DoseGrid .Origin .X = ' + str(dose.ImagePositionPatient[0]/10) +
';\n')
   tmp.write('TrialList .Current .DoseGrid .Origin .X = ' + str(LAT) + ';\n')
   tmp.write('TrialList .Current .DoseGrid .Origin .Y = ' + str(AP) + ';\n')
   tmp.write('TrialList .Current .DoseGrid .Origin .Z = ' + str(SI) + ';\n')
   tmp.write('TrialList .Current .DoseGrid .Display2d = 1;\n')

   #### Create Beam
   tmp.write('CreateNewBeam = "Add New Beam";\n')
   tmp.write('TrialList .Current .BeamList .Current .Name = "dummy";\n')
   tmp.write('TrialList .Current .BeamList .Current .Display2d = 0;\n')

   #### Create Prescription
   tmp.write('TrialList .Current .PrescriptionList .#"#0" .Name = "dummy";\n')
   tmp.write('TrialList .Current .PrescriptionList .Current .Method = "Set Monitor Units";\n')
   tmp.write('TrialList .Current .PrescriptionList .Current .RequestedMonitorUnitsPerFraction = "
1";\n')
   tmp.write('TrialList .Current .ComputeUncomputedBeams = "1";\n')
   tmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .Selected = 1;\n')
   tmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .LineType = "Medium Dashed";\n')
   #tmp.write('TrialList.Current.DoseGrid.LoadDataFromFile = "' + pinndosename + '";\n');

   tmp.close()


   print '\tWaiting for binary:',localdosename
   mtlabdone = os.path.isfile(localdosename)
   while (mtlabdone == 0):
      time.sleep(.5)
```

```
    mtlabdone = os.path.isfile(localdosename)
time.sleep(1) # to make sure matlab is done writing

shutil.copyfile(scriptname,ptscriptcopy)
shutil.copyfile(localdosename,ptdosecopy)
```

dcminterp.py

```
#!/usr/bin/env python

import os
import glob
import math
import dicom
import Tkinter, tkFileDialog
Tkinter.Tk().withdraw()

from dictPinn import manuf
from dictPinn import mode
from dictPinn import mach
from dictPinn import noBeamsLog

class WedgeError(Exception):
   def __init__(self, value):
      self.value = value
   def __str__(self):
      return repr(self.value)

class NoMode(Exception):
   def __init__(self, value):
      self.value =  'Mode improperly defined:' + value
   def __str__(self):
      return repr(self.value)

class NoBeams(Exception):
   def __init__(self, value):# , name,MR):
      self.value =  'Dicom has no Modulated beams:' + value
   def __str__(self):
      return repr(self.value)


class IsoStruct:
   def __init__(self):
      self.LAT = []
      self.AP = []
      self.SI = []

class SegmentStruct:
   def __init__(self):
      self.SegmentWeight = []
      self.Gantry = []
      self.PercentofArc = []
      self.A_LeafPositions = range(60)
      self.B_LeafPositions = range(60)
```

```
class JawStruct:
   def __init__(self):
      self.Y1 = []
      self.Y2 = []
      self.X1 = []
      self.X2 = []

class BeamStruct:
   def __init__(self,Nseg):
      self.Name = []
      self.FieldID = []
      self.Linac = []
      self.DoseRate = []
      self.IsocenterIndex = []
      self.Energy = []
      self.Mode = []
      self.MU = []
      self.Gantry = []
      self.Couch = []
      self.Collimator = []
      self.Jaws = JawStruct()
      self.NumSegments = []
      self.Segment = range(Nseg)

class RecordStruct:
    def __init__(self, filenames):

      #### find beam type
      temp = dicom.read_file(filenames[0])
      Mode = mode[temp.TreatmentSessionBeams[0].BeamType]

      nrecs = len(filenames)

      self.PatientName = temp.PatientsName.split('^')
      self.Beam = range(nrecs)
      self.TxDate = temp.TreatmentDate
      self.MRN = temp.PatientID
      self.NumberofBeams = nrecs
      self.NumberofFractions = temp.NumberofFractionsPlanned
      self.Mode = mode[temp.TreatmentSessionBeams[0].BeamType]
      totalMU = 0

      for f in range(nrecs):
         data = dicom.read_file(filenames[f])
         nseg = data.TreatmentSessionBeams[0].NumberofControlPoints
         self.Beam[f]= BeamStruct(nseg)
         del self.Beam[f].Segment
```

```
            del self.Beam[f].IsocenterIndex
            del self.Beam[f].Mode

            if int(data.TreatmentSessionBeams[0].NumberofWedges):
               raise WedgeError('Beam
"'+data.TreatmentSessionBeams[0].BeamName+'('+data.TreatmentSessionBeams[0].ReferencedBe
amNumber+')" includes a wedge.')
            self.Beam[f].Name = data.TreatmentSessionBeams[0].BeamName
            self.Beam[f].BeamNumber = data.TreatmentSessionBeams[0].ReferencedBeamNumber - 1
            self.Beam[f].DoseRate =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].DoseRateDelivered
            self.Beam[f].Energy =
str(int(data.TreatmentSessionBeams[0].ControlPointDeliverys[0].NominalBeamEnergy)) + ' MV'
            self.Beam[f].Gantry = data.TreatmentSessionBeams[0].ControlPointDeliverys[0].GantryAngle
            self.Beam[f].Collimator =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDeviceAngle
            self.Beam[f].Couch =
IECtoVarian("couch",data.TreatmentSessionBeams[0].ControlPointDeliverys[0].PatientSupportAngl
e)
            self.Beam[f].MU =
int(data.TreatmentSessionBeams[0].ControlPointDeliverys[1].DeliveredMeterset)
            totalMU += self.Beam[f].MU
            self.Beam[f].Jaws.Y1 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[1].LeafJaw
Positions[0]/-10
            self.Beam[f].Jaws.Y2 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[1].LeafJaw
Positions[1]/10
            self.Beam[f].Jaws.X1 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[0].LeafJaw
Positions[0]/-10
            self.Beam[f].Jaws.X2 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[0].LeafJaw
Positions[1]/10

         self.TotalMU = totalMU
         for beam_i in range(len(self.Beam)):
            self.Beam[beam_i].Weight = 100*float(self.Beam[beam_i].MU)/float(totalMU)

def IECtoVarian(device,iec):
   if (device == "gantry"):
      if iec > 180:
         return 540-iec
      else:
         return 180-iec
   elif (device == "couch"):
      if (iec == 0):
         return 0
```

```
        else:
            return 360 - iec

def IsoManager(ISOs,bmiso):
    num = len(ISOs)
    if (num == 1):
        bmisoindex = '1';
        if (len(bmiso) == 3):
            ISOs.append(['1', bmiso[0], bmiso[1], bmiso[2] ])
        else:
            ISOs.append(['1','none'])

    else:
        status = 'new'
        if (len(bmiso) == 3):
            for i in range(1,num):
                if (bmiso == ISOs[i][1:4]):
                    status = ISOs[i][0]
            if (status == 'new'):
                    ISOs.append([str(num), bmiso[0], bmiso[1], bmiso[2] ])
                    bmisoindex = str(num);
            else:
                bmisoindex = str(num-1);
        else: #if bmiso not given (bmiso='')
            for i in range(1,num):
                if (bmiso == ISOs[i][1]):
                    status = ISOs[i][0]
            if (status == 'new'):
                    ISOs.append([str(num),'none'])
                    bmisoindex = str(num);
            else:
                bmisoindex = str(num-1);

    return ISOs, bmisoindex

##############################################################################

def readPlan(filename='get'):
    if (filename == 'get'):
        import Tkinter, tkFileDialog
        Tkinter.Tk().withdraw()
        filename=tkFileDialog.askopenfilename(initialdir='C:/PinnacleImport',title='Dude, Select
DICOM file....',filetypes=[("All Files","*"),("DICOM plans","RP*"),("DICOM files","*.dcm")])
        print 'filename = ',filename
    data=dicom.read_file(filename)


    ########## first get rid of kV, port fields and OBI bullshit
```

```
destroylist = []
#destroylist.append(1)
try:
   test_n = data.FractionGroups[0].NumberofBeams
   for n in range(data.FractionGroups[0].NumberofBeams):
      if (data.Beams[n].ControlPoints[0].NominalBeamEnergy == 0):
         destroylist.append(n)
      elif (data.Beams[n].BeamDescription.find('Reference') != -1):
         destroylist.append(n)
      elif (data.Beams[n].BeamDescription.find('Ref') != -1):
         destroylist.append(n)
      elif (data.Beams[n].BeamDescription.find('ref') != -1):
         destroylist.append(n)
      elif (data.PatientsName.find('ZZZOBI') != -1):
         destroylist.append(n)
      else:
         try:
            m = len(data.Beams[n].ControlPoints[1].BLDPositions)-1
         except:
            destroylist.append(n)
except:
   l = open(noBeamsLog,'a')
   l.write(data.PatientsName.split('^')[0] + ',' + data.PatientID + ',' + filename)
   l.close
   raise NoBeams(filename) ###, str(data.PatientsName.split('^')[0] + data.PatientID))

destroylist.reverse()
for kv in destroylist:
   del data.Beams[kv]
   del data.FractionGroups[0].ReferencedBeams[kv]


if (len(data.Beams) > 0):
   #### find out if it is Step and Shoot or ARC


   Mode = mode[data.Beams[0].BeamType]
   if (Mode == 'SnS'):
      dcm = SnSplan(data)
   elif (Mode == 'ARC'):
      dcm = ARCplan(data)
   else:
      raise NoMode(Mode)
   return dcm
else:
   l = open(noBeamsLog,'a')
   l.write(data.PatientsName.split('^')[0] + ',' + data.PatientID + ',' + filename + '\n')
   l.close
```

```
        raise NoBeams(filename)

class SnSplan():
    def __init__(self, data):

            ########## Get Plan info
        nbeams = len(data.Beams)
        self.PatientName = data.PatientsName.split('^')
        self.MRN = data.PatientID
        self.Beam = range(nbeams)
        self.NumberofBeams = nbeams
        self.Machine = "Varian 2110" ##############data.Beams[b].TreatmentMachineName
        vendor = manuf[data.ManufacturersModelName]
        self.Vendor = vendor
        self.NumberofFractions = data.FractionGroups[0].NumberofFractionsPlanned
        self.Isocenters = [['Index', 'LAT', 'AP', 'SI']]

        ########## Get Beam specific info
        totalMU = 0
        for b in range(nbeams):
            if int(data.Beams[b].NumberofWedges):
                raise WedgeError('Beam '+str(b)+' includes a wedge.')

            ncp = data.Beams[b].NumberofControlPoints
            nseg = ncp/2
            self.Beam[b] = BeamStruct(nseg) ####create beam attributes
            if (len(data.Beams[b].ControlPoints[0].IsocenterPosition) == 3):
                bmiso = [data.Beams[b].ControlPoints[0].IsocenterPosition[0]/10,
data.Beams[b].ControlPoints[0].IsocenterPosition[1]/-10,
data.Beams[b].ControlPoints[0].IsocenterPosition[2]/-10]
            else:
                bmiso = 'none'

            self.Isocenters, self.Beam[b].IsocenterIndex = IsoManager(self.Isocenters, bmiso)

            self.Beam[b].Name = data.Beams[b].BeamDescription
            self.Beam[b].FieldID = data.Beams[b].BeamName
            self.Beam[b].Energy = str(int(data.Beams[b].ControlPoints[0].NominalBeamEnergy)) + ' MV'
            self.Beam[b].Mode = mode[data.Beams[b].BeamType]
            self.Beam[b].MU = data.FractionGroups[0].ReferencedBeams[b].BeamMeterset
            totalMU = totalMU + int(data.FractionGroups[0].ReferencedBeams[b].BeamMeterset)
            self.Beam[b].Weight = []
            self.Beam[b].DoseRate = data.Beams[b].ControlPoints[0].DoseRateSet
            if (vendor != 'mosaiq'):
                self.Beam[b].Couch =
IECtoVarian("couch",data.Beams[b].ControlPoints[0].PatientSupportAngle)
                self.Beam[b].Gantry = IECtoVarian("gantry",data.Beams[b].ControlPoints[0].GantryAngle)
                self.Beam[b].Collimator = data.Beams[b].ControlPoints[0].BeamLimitingDeviceAngle
```

```
        else:
            self.Beam[b].Couch =
IECtoVarian("couch",data.Beams[b].ControlPoints[0].PatientSupportAngle)
            self.Beam[b].Gantry = data.Beams[b].ControlPoints[0].GantryAngle
            self.Beam[b].Collimator = data.Beams[b].ControlPoints[0].BeamLimitingDeviceAngle
        self.Beam[b].Jaws.Y1 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[1].LeafJawPositions[0]/-10
        self.Beam[b].Jaws.Y2 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[1].LeafJawPositions[1]/10
        self.Beam[b].Jaws.X1 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[0].LeafJawPositions[0]/-10
        self.Beam[b].Jaws.X2 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[0].LeafJawPositions[1]/10
        self.Beam[b].NumSegments = nseg

        ###########Get Control Point info
        prev = 0
        finalMSW = data.Beams[b].FinalCumulativeMetersetWeight
        for c in range(1,ncp,2):
          ##Structure of ControlPoints[n].BLDPositions is different for first control point(n=0),
          ##and for STATIC treatments there are two control points per segment.
           self.Beam[b].Segment[(c-1)/2] = SegmentStruct()
           del self.Beam[b].Segment[(c-1)/2].PercentofArc
           if (vendor != 'mosaiq'):
              self.Beam[b].Segment[(c-1)/2].Gantry =
IECtoVarian("gantry",data.Beams[b].ControlPoints[0].GantryAngle)
           else:
              self.Beam[b].Segment[(c-1)/2].Gantry = data.Beams[b].ControlPoints[0].GantryAngle


           self.Beam[b].Segment[(c-1)/2].SegmentWeight =
(data.Beams[b].ControlPoints[c].CumulativeMetersetWeight - prev)/finalMSW
           prev = data.Beams[b].ControlPoints[c].CumulativeMetersetWeight

           for m in range(60):
              #print "b(",b,")", "c(",c,") ","m(",m,") ", "ncp(",ncp,")"
              bld = len(data.Beams[b].ControlPoints[c].BLDPositions)-1
              self.Beam[b].Segment[(c-1)/2].B_LeafPositions[59-m]=
data.Beams[b].ControlPoints[c].BLDPositions[bld].LeafJawPositions[m]/-10
              self.Beam[b].Segment[(c-1)/2].A_LeafPositions[59-m]=
data.Beams[b].ControlPoints[c].BLDPositions[bld].LeafJawPositions[m+60]/10

    ###### calculate Beam weights
    self.TotalMU = totalMU
    for beam_i in range(nbeams):
       self.Beam[beam_i].Weight = 100*float(self.Beam[beam_i].MU)/float(totalMU)

class ARCplan():
```

```
  def __init__(self, data):
    ########## Get Plan info
    nbeams = len(data.Beams)
    self.PatientName = data.PatientsName.split('^')
    self.MRN = data.PatientID
    self.Beam = range(nbeams)
    self.NumberofBeams = nbeams
    self.Machine = "Varian 2110" ##############data.Beams[b].TreatmentMachineName
    vendor = manuf[data.ManufacturersModelName]
    self.Vendor = vendor
    self.NumberofFractions = data.FractionGroups[0].NumberofFractionsPlanned
    self.Isocenters = [['Index', 'LAT', 'AP', 'SI']]

    ########## Get Beam specific info
    totalMU = 0
    self.Beam = range(nbeams)
    for b in range(nbeams):
      if int(data.Beams[b].NumberofWedges):
        raise WedgeError('Beam '+str(b)+' includes a wedge.')

      nseg = data.Beams[b].NumberofControlPoints
      self.Beam[b] = BeamStruct(nseg) ####create beam attributes

      if (len(data.Beams[b].ControlPoints[0].IsocenterPosition) == 3):
        bmiso = [data.Beams[b].ControlPoints[0].IsocenterPosition[0]/10,
data.Beams[b].ControlPoints[0].IsocenterPosition[1]/-10,
data.Beams[b].ControlPoints[0].IsocenterPosition[2]/-10]
      else:
        bmiso = 'none'

      self.Isocenters, self.Beam[b].IsocenterIndex = IsoManager(self.Isocenters, bmiso)

      self.Beam[b].Name = data.Beams[b].BeamDescription
      self.Beam[b].FieldID = data.Beams[b].BeamName
      self.Beam[b].Energy = str(int(data.Beams[b].ControlPoints[0].NominalBeamEnergy)) + ' MV'
      self.Beam[b].Mode = mode[data.Beams[b].BeamType]
      self.Beam[b].MU = data.FractionGroups[0].ReferencedBeams[b].BeamMeterset
      totalMU = totalMU + int(data.FractionGroups[0].ReferencedBeams[b].BeamMeterset)
      self.Beam[b].Weight = []
      self.Beam[b].DoseRate = data.Beams[b].ControlPoints[0].DoseRateSet

      if (vendor != 'mosaiq'):
        self.Beam[b].Gantry = IECtoVarian("gantry",data.Beams[b].ControlPoints[0].GantryAngle)
        self.Beam[b].Couch =
IECtoVarian("couch",data.Beams[b].ControlPoints[0].PatientSupportAngle)
        self.Beam[b].Collimator = data.Beams[b].ControlPoints[0].BeamLimitingDeviceAngle
      else:
        self.Beam[b].Gantry = data.Beams[b].ControlPoints[0].GantryAngle
```

```
            self.Beam[b].Couch =
IECtoVarian("couch",data.Beams[b].ControlPoints[0].PatientSupportAngle)
            self.Beam[b].Collimator = data.Beams[b].ControlPoints[0].BeamLimitingDeviceAngle
          self.Beam[b].Jaws.Y1 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[1].LeafJawPositions[0]/-10
          self.Beam[b].Jaws.Y2 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[1].LeafJawPositions[1]/10
          self.Beam[b].Jaws.X1 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[0].LeafJawPositions[0]/-10
          self.Beam[b].Jaws.X2 =
data.Beams[b].ControlPoints[0].BeamLimitingDevicePositions[0].LeafJawPositions[1]/10
          self.Beam[b].NumSegments = nseg

          ###########Get Control Point info
          prev = 0
          finalMSW = data.Beams[b].FinalCumulativeMetersetWeight
          for c in range(nseg):
            ###Structure of ControlPoints[n].BLDPositions is different for first control point(n=0),
            ###and there are two control points per segment.
            self.Beam[b].Segment[c] = SegmentStruct()


            #get segment beam angle and collect info for percent of arc calc
            if (vendor == 'mosaiq'): ##### mosaiq uses Varian coordinates in its DICOMs
              self.Beam[b].Segment[c].Gantry = data.Beams[b].ControlPoints[c].GantryAngle

              startangle = IECtoVarian('gantry',data.Beams[b].ControlPoints[0].GantryAngle)
              self.Beam[b].Segment[c].PercentofArc =
math.fabs(IECtoVarian('gantry',data.Beams[b].ControlPoints[c].GantryAngle) - startangle)

            else:
              self.Beam[b].Segment[c].Gantry =data.Beams[b].ControlPoints[c].GantryAngle

              startangle = data.Beams[b].ControlPoints[0].GantryAngle
              self.Beam[b].Segment[c].PercentofArc =
math.fabs(data.Beams[b].ControlPoints[c].GantryAngle - startangle)



            #get segment weight
            if ( c != nseg-1 ):
              self.Beam[b].Segment[c].SegmentWeight =
(data.Beams[b].ControlPoints[c+1].CumulativeMetersetWeight - prev)/finalMSW
              prev = data.Beams[b].ControlPoints[c+1].CumulativeMetersetWeight
            else:
              self.Beam[b].Segment[c].SegmentWeight = 0

            for m in range(60):
```

```
                #print "b(",b,")", "c(",c,") ","m(",m,") ", "nseg(",nseg,")"
                ### apparently arc dicoms give leaf position in CM at isocenter????
                bld = len(data.Beams[b].ControlPoints[c].BLDPositions)-1
                self.Beam[b].Segment[c].B_LeafPositions[59-m]=
data.Beams[b].ControlPoints[c].BLDPositions[bld].LeafJawPositions[m]/-10  ### SPD of MLCs is
50.87cm
                self.Beam[b].Segment[c].A_LeafPositions[59-m]=
data.Beams[b].ControlPoints[c].BLDPositions[bld].LeafJawPositions[m+60]/10  ### SPD of MLCs is
50.87cm

          #calc percent of arc
          for s in range(nseg):
             self.Beam[b].Segment[s].PercentofArc =
(self.Beam[b].Segment[s].PercentofArc)/(self.Beam[b].Segment[-1].PercentofArc)


       ####### calculate Beam weights
       self.TotalMU = totalMU
       for beam_i in range(nbeams):
          self.Beam[beam_i].Weight = 100*float(self.Beam[beam_i].MU)/float(totalMU)

##############################################################################

def readBeamRecords(filter="RT.*",dir='none'):
   filter = str(filter)
   if (filter == ''):
      filter = 'RT.*.dcm'
   elif(filter == '[]'):
      filter = 'RT.*.dcm'
   elif(filter == []):
      filter = 'RT.*.dcm'

   if (filter.find('*') == -1):
      if (filter.find('RT.') == -1):
         filter = "RT." + filter + "*"
   #else:
   #    filter = '*' + filter + '*.dcm'
   #print 'filter = ',filter,'\ndir = ',dir

   if (os.path.isdir(dir) == 0):
      dir = tkFileDialog.askdirectory(initialdir='C:/PinnacleImport/')

   filenames = glob.glob(os.path.join(dir,filter))

   ## get beam type
   temp = dicom.read_file(filenames[0])
   Mode = mode[temp.TreatmentSessionBeams[0].BeamType]
```

```
    if (Mode == 'SnS'):
        dcm = RecordStruct(filenames)
    elif (Mode == 'ARC'):
        dcmtmp = RecordStruct(filenames)
        ### if ARC then sort beams  by gantry then Beam Weight(MU) (decending sort)
        bmsort = []
        for j in range(len(dcmtmp.Beam)):
            bmsort.append([dcmtmp.Beam[j].Gantry,dcmtmp.Beam[j].Weight,j])
            bmsort.sort(reverse=1)

        dcm=dcmtmp
        for k in range(len(bmsort)):
            #print bmsort[k][-1]
            dcm.Beam[k] = dcmtmp.Beam[bmsort[k][-1]]
    else:
        raise NoMode(dcm.Mode)


    return dcm

class getRecords():
    def __init__(self, filenames):

        #### find beam type
        temp = dicom.read_file(filenames[0])
        Mode = mode[temp.TreatmentSessionBeams[0].BeamType]

        nrecs = len(filenames)

        self.PatientName = temp.PatientsName.split('^')
        self.Beam = range(nrecs)
        self.TxDate = temp.TreatmentDate
        self.MRN = temp.PatientID
        self.NumberofBeams = nrecs
        self.NumberofFractions = temp.NumberofFractionsPlanned
        self.Mode = mode[temp.TreatmentSessionBeams[0].BeamType]
        totalMU = 0

        for f in range(nrecs):
            data = dicom.read_file(filenames[f])
            nseg = data.TreatmentSessionBeams[0].NumberofControlPoints
            self.Beam[f]= BeamStruct(nseg)
            del self.Beam[f].Segment
            del self.Beam[f].IsocenterIndex
            del self.Beam[f].Mode
```

```
        if int(data.TreatmentSessionBeams[0].NumberofWedges):
            raise WedgeError('Beam
"'+data.TreatmentSessionBeams[0].BeamName+'('+data.TreatmentSessionBeams[0].ReferencedBe
amNumber+')" includes a wedge.')
        self.Beam[f].Name = data.TreatmentSessionBeams[0].BeamName
        self.Beam[f].BeamNumber = data.TreatmentSessionBeams[0].ReferencedBeamNumber - 1
        self.Beam[f].DoseRate =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].DoseRateDelivered
        self.Beam[f].Energy =
str(int(data.TreatmentSessionBeams[0].ControlPointDeliverys[0].NominalBeamEnergy)) + ' MV'
        self.Beam[f].Gantry = data.TreatmentSessionBeams[0].ControlPointDeliverys[0].GantryAngle
        self.Beam[f].Collimator =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDeviceAngle
        self.Beam[f].Couch =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].PatientSupportAngle
        self.Beam[f].MU =
int(data.TreatmentSessionBeams[0].ControlPointDeliverys[1].DeliveredMeterset)
        totalMU += self.Beam[f].MU
        self.Beam[f].Jaws.Y1 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[1].LeafJaw
Positions[0]/-10
        self.Beam[f].Jaws.Y2 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[1].LeafJaw
Positions[1]/10
        self.Beam[f].Jaws.X1 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[0].LeafJaw
Positions[0]/-10
        self.Beam[f].Jaws.X2 =
data.TreatmentSessionBeams[0].ControlPointDeliverys[0].BeamLimitingDevicePositions[0].LeafJaw
Positions[1]/10

    self.TotalMU = totalMU
    for beam_i in range(len(self.Beam)):
        self.Beam[beam_i].Weight = 100*float(self.Beam[beam_i].MU)/float(totalMU)
```

dynreader.py

```
#!/usr/bin/env python

import sys
import os
import glob
from pprint import pprint
import fget

from dictPinn import ptdir
from dictPinn import arcsegmentlen

class MismatchError(Exception):
   def __init__(self, value):
      self.value =  'The numbers of A-carriage and B-carriage dynalog files do not agree: ' + value
   def __str__(self):
      return repr(self.value)

def read(dir='none',ptid='none'):
   dir = str(dir)
   if (dir == 'none'): dir = fget.dir(ptdir)

   if (ptid == 'none'):
      Afilter = 'A*.dlg'
      Bfilter = 'B*.dlg'
   elif (ptid == '[]'):
      Afilter = 'A*.dlg'
      Bfilter = 'B*.dlg'
   elif (ptid == []):
      Afilter = 'A*.dlg'
      Bfilter = 'B*.dlg'
   else:
      Afilter = 'A*' + str(ptid) + '*.dlg'
      Bfilter = 'B*' + str(ptid) + '*.dlg'
   #print 'dir = ',dir,'\nAfilter = ',Afilter,'\nBfilter = ',Bfilter

   Afiles = glob.glob(os.path.join(dir,Afilter))
   Btmp = glob.glob(os.path.join(dir,Bfilter))

   Bfiles = []
   for i,A in enumerate(Afiles):
      A = 'B' + os.path.basename(A)[1:]
      for j,B in enumerate(Btmp):
         B = os.path.basename(B)
         if (A == B):
            Bfiles.append('')
            Bfiles[-1] = Btmp[j]
```

```
            del Btmp[j]
            break

    ### finout what is the lowest numbered RT file
    rtnum=[]
    for rt in glob.glob(os.path.join(dir,'RT*.dcm')):
        rtnum.append(int(os.path.basename(rt).split('.')[-2].replace('Beam_','')))

    if ( len(rtnum)> 0 ):
        rtstart = min(rtnum)-1
    else:
        rtstart=0

    nbeams = len(Afiles);
    if (len(Afiles) != len(Bfiles)):
        raise MismatchError(dir)

    type = gettype(Afiles[0])

    if (type == 'SnS'):
        dyn = readSnS(Afiles,Bfiles,rtstart)
    elif (type == 'ARC'):
        dyntmp = readARC(Afiles,Bfiles,rtstart)
        bmsort = []
        for j in range(len(dyntmp.Beam)):
            bmsort.append([dyntmp.Beam[j].Segment[0].Gantry,j])
        bmsort.sort(reverse=1)
        dyn=dyntmp
        for k in range(len(bmsort)):
            dyn.Beam[k] = dyntmp.Beam[bmsort[k][1]]

    return dyn

class readSnS():
    def __init__(self, Afiles,Bfiles,rtstart, headerlines = 6):


        a = open(Afiles[0],'r')
        Afo = a.readlines()
        a.close()

        nbeams = len(Afiles)


        self.MRN = str(Afo[1].split(',')[-1]).replace('\n','')
        self.LastName = Afo[1].split(',')[0]
        self.Mode = 'SnS'
        self.Beam = range(nbeams)
```

```
self.NumberofBeams = nbeams


for i in range(nbeams):

    #for i in range(len(Afiles)):
    print '\n',Afiles[i],
    a = open(Afiles[i],'r')

    Afo = a.readlines()
    a.close()
    b = open(Bfiles[i],'r')
    Bfo = b.readlines()
    b.close()

    bnm = int(Afo[2].split(',')[-1]) - 1 - rtstart
    coords = int(Afo[5]);

    del Afo[:headerlines]
    del Bfo[:headerlines]

    bmon,segwght = getBeamOn(Afo)

    #####nseg = len(bmon)/2
    nseg = len(segwght)
    totwt = sum(segwght)

    self.Beam[bnm] = BeamStruct(nseg)

    tmp = tmpStruct()
    nlines = len(Afo)

    for j in range(len(Afo)):
        a = Afo[j].split(',')
        tmp.Adosefr.append(float(a[0]))
        tmp.AGantry.append(float(a[6]))
        tmp.AColl.append(float(a[7]))
        tmp.AY1.append(float(a[8]))
        tmp.AY2.append(float(a[9]))
        tmp.AX1.append(float(a[10]))
        tmp.AX2.append(float(a[11]))

        b = Bfo[j].split(',')
        tmp.Bdosefr.append(float(b[0]))
        tmp.BGantry.append(float(b[6]))
        tmp.BColl.append(float(b[7]))
        tmp.BY1.append(float(b[8]))
        tmp.BY2.append(float(b[9]))
```

```
            tmp.BX1.append(float(b[10]))
            tmp.BX2.append(float(b[11]))

            ########print "TEST SETTINGS dynreader 179"
            for k in range(60):
               indx = 4*k + 15

               tmp.Aleafs[k].append(float(a[indx]))
               tmp.Bleafs[k].append(float(b[indx]))

         self.Beam[bnm].Gantry = (mean(tmp.AGantry) + mean(tmp.BGantry))/20
         self.Beam[bnm].Collimator = (mean(tmp.AColl) + mean(tmp.BColl))/20
         if coords:
            self.Beam[bnm].Gantry = IEC_Varian_convert(self.Beam[bnm].Gantry)
            self.Beam[bnm].Collimator = IEC_Varian_convert(self.Beam[bnm].Collimator)

         self.Beam[bnm].Jaws.Y1 = (mean(tmp.AY1) + mean(tmp.BY1))/20
         self.Beam[bnm].Jaws.Y2 = (mean(tmp.AY2) + mean(tmp.BY2))/20
         self.Beam[bnm].Jaws.X1 = (mean(tmp.AX1) + mean(tmp.BX1))/20
         self.Beam[bnm].Jaws.X2 = (mean(tmp.AX2) + mean(tmp.BX2))/20

         #totwt = float(0)

         for m in range(nseg):
            self.Beam[bnm].Segment[m] = SegmentStruct()

            self.Beam[bnm].Segment[m].SegmentWeight = segwght[m]
            #totwt = totwt + self.Beam[bnm].Segment[m].SegmentWeight

            for n in  range(60):
               self.Beam[bnm].Segment[m].Gantry = self.Beam[bnm].Gantry
               self.Beam[bnm].Segment[m].A_LeafPositions[n] =
mean(tmp.Aleafs[n][bmon[2*m]:bmon[2*m+1]])/510####508.7 #SSD of MLC is 50.87cm
               self.Beam[bnm].Segment[m].B_LeafPositions[n] =
mean(tmp.Bleafs[n][bmon[2*m]:bmon[2*m+1]])/510####/508.7 #SSD of MLC is 50.87cm



         if (totwt != 1):

            print '%6.4f' % (totwt),######' , ', Afiles[i]
            for p in range(nseg):
               self.Beam[bnm].Segment[p].SegmentWeight =
self.Beam[bnm].Segment[p].SegmentWeight/totwt

class readARC():
   def __init__(self, Afiles,Bfiles,rtstart, headerlines = 6):
      a = open(Afiles[0],'r')
```

```
Afo = a.readlines()
a.close()

nbeams = len(Afiles)

self.MRN = str(Afo[1].split(',')[-1]).replace('\n','')
self.LastName = Afo[1].split(',')[0]
self.Mode = 'ARC'
self.Beam = range(nbeams)
self.NumberofBeams = nbeams

for i in range(nbeams):
   a = open(Afiles[i],'r')
   Afo = a.readlines()
   a.close()
   b = open(Bfiles[i],'r')
   Bfo = b.readlines()
   b.close()

   bnm = int(Afo[2].split(',')[-1]) - 1 - rtstart
   coords = int(Afo[5]);



   del Afo[:headerlines]
   del Bfo[:headerlines]
   CPgantry, rot = getGantrypos(Afo,coords)

   nseg = len(CPgantry)

   self.Beam[bnm] = BeamStruct(nseg)
   self.Beam[bnm].Rotation = rot

   tmp = tmpStruct()
   nlines = len(Afo)
   for j in range(len(Afo)):
      a = Afo[j].split(',')
      tmp.AGantry.append(float(a[0]))
      tmp.AColl.append(float(a[7]))
      tmp.AY1.append(float(a[8]))
      tmp.AY2.append(float(a[9]))
      tmp.AX1.append(float(a[10]))
      tmp.AX2.append(float(a[11]))

      b = Bfo[j].split(',')
      tmp.BGantry.append(float(b[0]))
      tmp.BColl.append(float(b[7]))
      tmp.BY1.append(float(b[8]))
```

```
            tmp.BY2.append(float(b[9]))
            tmp.BX1.append(float(b[10]))
            tmp.BX2.append(float(b[11]))

            for k in range(60):
               indx = 4*k + 15
               tmp.Aleafs[k].append(float(a[indx]))
               tmp.Bleafs[k].append(float(b[indx]))

         self.Beam[bnm].Collimator = (mean(tmp.AColl) + mean(tmp.BColl))/20
         self.Beam[bnm].Gantry = (tmp.AGantry[0] + tmp.BGantry[0])/20
         if coords:
            self.Beam[bnm].Collimator = IEC_Varian_convert(self.Beam[bnm].Collimator)
            self.Beam[bnm].Gantry = IEC_Varian_convert(self.Beam[bnm].Gantry)

         self.Beam[bnm].Jaws.Y1 = (mean(tmp.AY1) + mean(tmp.BY1))/20
         self.Beam[bnm].Jaws.Y2 = (mean(tmp.AY2) + mean(tmp.BY2))/20
         self.Beam[bnm].Jaws.X1 = (mean(tmp.AX1) + mean(tmp.BX1))/20
         self.Beam[bnm].Jaws.X2 = (mean(tmp.AX2) + mean(tmp.BX2))/20

         for m in range(nseg):
            self.Beam[bnm].Segment[m] = SegmentStruct()
            del self.Beam[bnm].Segment[m].SegmentWeight

            self.Beam[bnm].Segment[m].Gantry = CPgantry[m][0]
            for n in  range(60):

               self.Beam[bnm].Segment[m].A_LeafPositions[n] =
mean(tmp.Aleafs[n][CPgantry[m][1][0]:CPgantry[m][1][1]])/508.7 #SSD of MLC is 50.87cm
               self.Beam[bnm].Segment[m].B_LeafPositions[n] =
mean(tmp.Bleafs[n][CPgantry[m][1][0]:CPgantry[m][1][1]])/508.7 #SSD of MLC is 50.87cm

class tmpStruct():
   def __init__(self):
      self.Adosefr = []
      self.AGantry = []
      self.AColl = []
      self.AY1 = []
      self.AY2 = []
      self.AX1 = []
      self.AX2 = []
      self.Aleafs = range(60)

      self.Bdosefr = []
      self.BGantry = []
      self.BColl = []
      self.BY1 = []
      self.BY2 = []
```

```
        self.BX1 = []
        self.BX2 = []
        self.Bleafs = range(60)

        for l in range(60):
            self.Aleafs[l] = []
            self.Bleafs[l] = []

class BeamStruct():
    def __init__(self,nseg):
        self.Gantry = []
        self.Collimator = []
        self.Jaws = JawStruct()
        self.NumberofSegments = nseg
        self.Segment = range(nseg)

class JawStruct():
    def __init__(self):
        self.Y1 = []
        self.Y2 = []
        self.X1 = []
        self.X2 = []

class SegmentStruct():
    def __init__(self):
        self.Gantry = []
        self.SegmentWeight = []
        self.A_LeafPositions = range(60)
        self.B_LeafPositions = range(60)

def getGantrypos(list,coords):
    ### this is much easier in IEC coordinates, so will convert to IEC if given Varian
    gantry = []
    for i,v in enumerate(list):
        gantry.append(float(v.split(',')[0])/10)


    #if Varian convert to IEC
    if (coords != 1):
        gantry = [IEC_Varian_convert(x) for x in gantry ]

    if gantry[0] > gantry[10]:
        rot = 'CW'
    else:
        rot = 'CCW'


    start = int(round(gantry[0],0))
```

```
    fin = int(round(gantry[-1],0))

    #print 'start/fin',start,'/',fin
    #print 'start/fin',IEC_Varian_convert(start),'/',IEC_Varian_convert(fin)

    if (rot == 'CW'):
        gantryneed = range(start,fin,-1*arcsegmentlen)
        gantryneed.append(fin)
    elif (rot == 'CCW'):
        gantryneed = range(start,fin,arcsegmentlen)
        gantryneed.append(fin)


    gantryneed = [IEC_Varian_convert(x) for x in gantryneed ]
    gantry = [IEC_Varian_convert(x) for x in gantry ]


    CPgantry = []
    for j in range(len(gantryneed)):
        tmp = [abs(x-gantryneed[j]) for x in gantry]
        cnt = tmp.count(min(tmp))
        indx = tmp.index(min(tmp))
        CPgantry.append([gantryneed[j],[indx,indx + cnt]])

return CPgantry, rot

def gantryround(num):
    t = str(num).split('.')
    if (int(t[1][:1]) >= 8):
        t = float(t[0]) + 1
    elif (int(t[1][:1]) <= 2):
        t = float(t[0])
    else:
        t = float(t[0]) + float(t[1])/(len(t[1])*10)
    return t

def mean(numberList):
    floatNums = [float(x) for x in numberList]
    return sum(floatNums) / len(numberList)

def IEC_Varian_convert(angle):
    #converts between IEC and Varian coordinates. The coversion is symmetric.
    if (angle >180):
        converted = 540 - angle
    else:
        converted = 180 - angle
    return converted
def getBeamOn(list):
```

```
    Indexes = []
    for j in list:
        Indexes.append([int(j.split(',')[1]),int(j.split(',')[3]),float(j.split(',')[4]),float(j.split(',')[5])]) ####
keep [Beam-On State , Prev Seg Dose Index, Next Seg Dose Index ]


    bmon = [0]
    segweigh = []

    prev = 0
    i = 1
    while i < len(Indexes):
        if (Indexes[i][0] != prev):

            segweigh.append((Indexes[i-2][3] -Indexes[i-2][2])/25000)
            bmon.append(i)
            prev = Indexes[i][0]
            j=i+1

            if not Indexes[j+1][1]: ### if next beam is on j is new segment
                j+=2
                while j < len(Indexes):
                    if Indexes[j][1]:
                        bmon.append(j)

                        i=j
                        break
                    j+=1
                    i=j
            else:
                bmon.append(j)
                i+=2
        i+=1

    for k in range(len(Indexes)-1,-1,-1):
        if Indexes[k][1]:
            bmon.append(len(list))
            segweigh.append((Indexes[-1][3] -Indexes[-1][2])/25000)
            break

def gettype(file, headerlines=6):
    f = open(file, 'r')
    info = f.readlines()
    f.close()
    del info[:headerlines]

    #### if step and shoot the first column is dose fraction and should end in 25000
    #### if Arc the first column is the gantry angle to a tenth of a degree in Varian coords
```

```
    if (int(info[-1].split(',')[0]) == 25000):
        type = 'SnS'
    else:
        type = 'ARC'


    return type

def dynprint(dyn_obj,fileout=''):
    beam_id = {7.0:'Center',9.0:'Right',5.0:'Left',16.0:'FarRight',-2.0:'FarLeft'}

    bm = len(dyn_obj.Beam)


    if  fileout != '':
        out = open('fileout','w')

        for bm in dyn_obj.Beam:
            out.write(beam_id[bm.Jaws.X1] + '\n')
            for sg in range(bm.NumberofSegments):
                out.write('\tSeg ' + sg + '\n')

                out.write('\t\tA_car\n')

                for lf in range(60):
                    out.write('\t\t' + bm.Segment[sg].A_LeafPositions[lf] + '\n')

                out.write('\t\tB_car\n')
                for lf in range(60):
                    out.write('\t\t' + bm.Segment[sg].B_LeafPositions[lf] + '\n')
        out.close()
    else:
        for bm in dyn_obj.Beam:
            print beam_id[bm.Jaws.X1]
            for sg in range(bm.NumberofSegments):
                print '\tSeg ',sg

                print '\t\tA_car'

                for lf in range(60):
                    print '\t',lf,'\t',bm.Segment[sg].A_LeafPositions[lf]

                print '\t\tB_car'
                for lf in range(60):
                    print '\t', lf,'\t',bm.Segment[sg].B_LeafPositions[lf]


def rep(d,fileoutA='',fileoutB=''):
```

```
beam_id = {7.0:'Center',9.0:'Right',5.0:'Left',16.0:'FarRight',-2.0:'FarLeft'}

length = 63
outA = []
outB = []
for i in range(length):
   outA.append([])
   outB.append([])

bm = len(d.Beam)

for bm in d.Beam:
   for sg in range(bm.NumberofSegments):
      outA[0].append(beam_id[bm.Jaws.X1])
      outA[1].append('Seg ' + str(sg))
      outA[2].append('A_car')

      outB[0].append(beam_id[bm.Jaws.X1])
      outB[1].append('Seg ' + str(sg))
      outB[2].append('B_car')

      for lf in range(60):
         outA[lf+3].append(int(round(bm.Segment[sg].A_LeafPositions[lf])))
         outB[lf+3].append(int(round(bm.Segment[sg].B_LeafPositions[lf])))

ofileA = open(fileoutA,'w')
for line in range(length):
   ofileA.write(str(outA[line])[1:-1] + '\n')
ofileA.close()

ofileB = open(fileoutB,'w')
for line in range(length):
   ofileB.write(str(outB[line])[1:-1] + '\n')
ofileB.close()
```

BinaryRW.py

```
#!/usr/bin/env python

import struct
import dicom

def writePinnDose(dcm_file,newfile):
    dcm = dicom.read_file(dcm_file)

    x_dim = dcm.Columns
    y_dim = dcm.Rows
    z_dim = len(dcm.GridFrameOffsetVector)
    scale = dcm.DoseGridScaling*100

    IN_format = '<%sI' % (x_dim*y_dim*z_dim) # little endian, unsigned Int
    OUT_format = '!%sf' % (x_dim*y_dim*z_dim) # big endian, float


    dose_int = struct.unpack(IN_format,dcm.PixelData)
    dose_float = map(float,dose_int)
    dose_values = map(lambda x: x*scale,dose_float)

    dose = inside_out(dose_values,x_dim*y_dim)

    dose_out = struct.pack(OUT_format,*dose)

    f=open(newfile,'wb')
    f.write(dose_out)
    f.close()

    return

def inside_out(oldlist,s_size):
    newlist = range(len(oldlist))

    nslices = len(oldlist)/s_size

    for s in range(nslices):
        n_strt = s*s_size
        n_end = n_strt + s_size

        o_end = (nslices-s)*s_size
        o_strt = o_end - s_size

        newlist[n_strt:n_end]=oldlist[o_strt:o_end]

    return newlist
```

7 Appendix B: Pinnacle Import software

SA_PlanImport.Script

```
////////////    DEBUG MODE    ////////////
Store.FloatAt.DEBUG = 0;   // debug mode on  = 1


////////////    Definitions   ////////////
//    Python Script
Store.StringAt.Pyfile = "/home/p3rtp/ohrt/StandAloneScripts/SA_PlanImport.py";

//    Temp Script
Store.StringAt.TempScript = "/home/p3rtp/ohrt/tmpScripts/";
Store.At.TempScript.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.TempScript.AppendString = "TEMP\.Script";

////////////    Import   ////////////

Store.StringAt.edosegrid = "python ";
Store.At.edosegrid.AppendString = Store.At.Pyfile.String;
Store.At.edosegrid.AppendString = " ";
Store.At.edosegrid.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.edosegrid.AppendString = " '";
Store.At.edosegrid.AppendString = PlanInfo.LastName;
Store.At.edosegrid.AppendString = "' ";
Store.At.edosegrid.AppendString = Store.At.TempScript.String;
IF.Store.FloatAt.DEBUG.THEN.WarningMessage = Store.At.edosegrid.String;
SpawnCommand = Store.At.edosegrid.String;

Script.ExecuteNow = Store.At.TempScript.String;

Store.StringAt.rm = "rm ";
Store.At.rm.AppendString = Store.At.TempScript.String;
IF.Store.FloatAt.DEBUG.THEN.Store.At.Nothing.ELSE.SpawnCommand = Store.At.rm.String;

SavePlan = "";
```

SA_PlanImport.py

```
#/usr/bin/env python

import os
import glob
import sys
sys.path.append('/home/p3rtp/ohrt')
#sys.path.append('/home/p3rtp/ohrt/PtScripts')

import fget
from dictImport import scriptdir

mrn = sys.argv[1]
lastname = str(sys.argv[2]).replace('restored','')
lastname = lastname.replace(' ','')
lastname = lastname[0].upper() + lastname[1:].lower()
tmpscript = sys.argv[3]

filter = '*plan.' + str(mrn) + '.' + lastname + '*.Script'

ptfile = glob.glob(os.path.join(scriptdir,filter))
if (len(ptfile) == 0):
   print 'No Plan files found. Asking for one.....'
   ptfile = []
   ptfile.append(fget.file(scriptdir, '*plan*.Script', str( 'Pick a Plan file for ' + mrn + ' ' + lastname)))

elif (len(ptfile) > 1):
   print 'More than one  Plan file found. Asking which one you want....'
   ptfile = []
   ptfile.append(fget.file(scriptdir, filter, str( 'Pick a Plan file for ' + mrn + ' ' + lastname)))

tmp = open(tmpscript, 'w')
tmp.write('Script.ExecuteNow = "' + str(ptfile[0]) + '";')
tmp.close()
os.system('chmod 755 ' + tmpscript )
```

SA_TxImport.Script

```
// This script calls a python file to import a dosegrid into the current plan.
///IsVerbose = "1";

////////////    DEBUG MODE    ////////////
Store.FloatAt.DEBUG = 0;   // debug mode on  = 1


////////////    Definitions   ////////////
//    Python Script
Store.StringAt.Pyfile = "/home/p3rtp/ohrt/TxUpdate.py";

//    Temp Script
Store.StringAt.TempScript = "/home/p3rtp/ohrt/tmpScripts/";
Store.At.TempScript.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.TempScript.AppendString = "TEMP1\.Script";

////////////    Import   ////////////
SavePlan = "";

Store.StringAt.edosegrid = "python ";
Store.At.edosegrid.AppendString = Store.At.Pyfile.String;
Store.At.edosegrid.AppendString = " ";
Store.At.edosegrid.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.edosegrid.AppendString = " '";
Store.At.edosegrid.AppendString = PlanInfo.LastName;
Store.At.edosegrid.AppendString = "' ";
Store.At.edosegrid.AppendString = Store.At.TempScript.String;
IF.Store.FloatAt.DEBUG.THEN.WarningMessage = Store.At.edosegrid.String;
SpawnCommand = Store.At.edosegrid.String;

Script.ExecuteNow = Store.At.TempScript.String;

Store.StringAt.rm = "rm ";
Store.At.rm.AppendString = Store.At.TempScript3.String;
IF.Store.FloatAt.DEBUG.THEN.Store.At.Nothing.ELSE.SpawnCommand = Store.At.rm.String;


SavePlan = "";
```

TxImport.py

```
#/usr/bin/env python

import os
import sys
import glob
sys.path.append('/home/p3rtp/ohrt')
sys.path.append('/home/p3rtp/ohrt/PtScripts')

import fget
from dictImport import scriptdir


mrn = sys.argv[1]
lastname = str(sys.argv[2]).replace('restored','')
lastname = lastname.replace(' ','')
lastname = lastname[0].upper() + lastname[1:].lower()

tmpscript = sys.argv[3]

filter = 'TXQA.' + str(mrn) + '.' + lastname + '*.Script'


ptfile = glob.glob(os.path.join(scriptdir,filter))
if (len(ptfile) == 0):
   print 'No Tx files found. Asking for one.....'
   ptfile = []
   ptfile.append(fget.file(scriptdir, 'TXQA.*.Script', str( 'Pick a TXQA file for ' + mrn + ' ' + lastname)))
elif (len(ptfile) > 1):
   print 'More than one Tx file found. Asking which one you want....'
   ptfile = []
   ptfile.append(fget.file(scriptdir, filter, str( 'Pick a TXQA file for ' + mrn + ' ' + lastname)))


tmp = open(tmpscript, 'w')
tmp.write('Script.ExecuteNow = "' + str(ptfile[0]) + '";')
tmp.close()
os.system('chmod 755 ' + tmpscript )
```

SA_DoseGridImport.Script

```
// This script calls a python file to import a dosegrid into the current plan.

////////////    DEBUG MODE    ////////////
Store.FloatAt.DEBUG = 0;   // debug mode on  = 1

////////////    Definitions   ////////////
//    Python Script
Store.StringAt.Pyfile = "/home/p3rtp/ohrt/StandAloneScripts/SA_DoseGridImport.py";

//    Temp Script
Store.StringAt.TempScript = "/home/p3rtp/ohrt/tmpScripts/";
Store.At.TempScript.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.TempScript.AppendString = "TEMP1\.Script";

////////////    Import   ////////////
TrialList.ChildrenEachCurrent.#"@".TrialList.Current.ComputeUncomputedBeams = "1";
SavePlan = "";

Store.StringAt.edosegrid = "python ";
Store.At.edosegrid.AppendString = Store.At.Pyfile.String;
Store.At.edosegrid.AppendString = " ";
Store.At.edosegrid.AppendString = "import";
Store.At.edosegrid.AppendString = " ";
Store.At.edosegrid.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.edosegrid.AppendString = " '";
Store.At.edosegrid.AppendString = PlanInfo.LastName;
Store.At.edosegrid.AppendString = "' ";
Store.At.edosegrid.AppendString = Store.At.TempScript.String;
IF.Store.FloatAt.DEBUG.THEN.WarningMessage = Store.At.edosegrid.String;
SpawnCommand = Store.At.edosegrid.String;

Script.ExecuteNow = Store.At.TempScript.String;

Store.StringAt.rm = "rm ";
Store.At.rm.AppendString = Store.At.TempScript.String;
IF.Store.FloatAt.DEBUG.THEN.Store.At.Nothing.ELSE.SpawnCommand = Store.At.rm.String;

//////// ADDED THIS, MIGHT BE COOL
PluginManager .PlanEvalPlugin .TrialList .#"#3" .Selected = 1;

SavePlan = ""; TempScript will Save plan, then replace binary file. Don't want to save here

//WarningMessage = "The Plan has been saved. It will now be closed, so that the Dose Grid Import
can be completed. Thank You";
IF.Store.FloatAt.DEBUG.THEN.Store.At.Nothing.ELSE.Quit = "";
```

SA_DoseGridImport.py

```
#/usr/bin/env python

import os
import glob
import sys
sys.path.append('/home/p3rtp/ohrt')

import shutil

import fget
import readTrial
from dictImport import scriptdir

action = sys.argv[1]

if (action == 'import'):
   mrn = sys.argv[2]
   lastname = str(sys.argv[3]).replace('restored','')
   lastname = lastname.replace(' ','')
   lastname = lastname[0].upper() + lastname[1:].lower()
   tmpscript = sys.argv[4]

   filter = 'DoseGrid_*' + str(mrn) + lastname + '*.Script'


   ptfile = glob.glob(os.path.join(scriptdir,filter))
   if (len(ptfile) == 0):
      print 'No DoseGrid Script files found. Asking for one.....'
      ptfile = fget.file(scriptdir, 'DoseGrid*.Script', str( 'Pick a DoseGrid Script for ' + mrn + ' ' +
lastname))
   elif (len(ptfile) > 1):
      print 'More than one DoseGrid Script file found. Asking which one you want....'
      ptfile = fget.file(scriptdir, filter, str( 'Pick a DoseGrid Script for ' + mrn + ' ' + lastname))
   else:
      ptfile = ptfile[0]

   binfile = ptfile.replace('.Script','.binary')
   #print 'ptfile: ',ptfile,'\nbinfile: ',binfile

   # build a Script that executes premade DoseGrid Script and then calls this file again with action =
setgrid
   tmp = open(tmpscript, 'w')
   tmp.write('Script.ExecuteNow = "' + ptfile + '";\n')
   tmp.write('SavePlan = "";\n')
   tmp.write('Store.StringAt.setgrid = "python '+ os.path.abspath(__file__) + ' setgrid \'' + binfile + '\'
";\n')
   tmp.write('Store.At.setgrid.AppendString = TrialList.Current.Index;\n')
```

```
    tmp.write('Store.At.setgrid.AppendString = " ";\n')
    tmp.write('Store.At.setgrid.AppendString = PlanInfo.PlanPath;\n')
    tmp.write('Store.At.setgrid.AppendString = " \'";\n')
    tmp.write('Store.At.setgrid.AppendString = PlanInfo.LastName;\n')
    tmp.write('Store.At.setgrid.AppendString = "\'";\n')
    #tmp.write('WarningMessage = Store.At.setgrid.String;\n')
    tmp.write('SpawnCommand = Store.At.setgrid.String;\n')
    tmp.write('Store.FreeAt.setgrid = "";\n')
    tmp.close()
    os.system('chmod 755 ' + tmpscript )

elif (action == 'setgrid'):
    newbinfile = sys.argv[2]
    trialindex = int(sys.argv[3])
    planpath = sys.argv[4]
    pinnpath = os.getcwd()
    oldbinpath = os.path.join(pinnpath,planpath,'plan.Trial.binary.')

    lastname = str(sys.argv[5]).replace('restored','')
    lastname = lastname.replace(' ','')
    lastname = lastname[0].upper() + lastname[1:].lower()
    print 'SETGRID: ',lastname

    TrialData =
readTrial.go(os.path.join(pinnpath,planpath,'plan.Trial'),os.path.join(scriptdir,'plan.Trial.txt'))

    #print 'trialindex=',trialindex
    #for i in range(len(TrialData)): print TrialData[i][0:3]


    binindex = str(TrialData[trialindex][6][0])
    if (len(binindex) == 1):
       oldbinfile = oldbinpath + '00' +  binindex
    elif (len(binindex) == 2):
       oldbinfile = oldbinpath + '0' +  binindex
    elif (len(binindex) == 3):
       oldbinfile = oldbinpath + binindex

    #print 'old=',oldbinfile,'new=',newbinfile

    os.remove(oldbinfile)
    shutil.copyfile(newbinfile,oldbinfile)
```

8 Appendix C: 3D Gamma Calculation software

SA_ComputeGamm.Script

```
// This script requests a 3D gamma calcuation between the dosegrids of 2 or more
// trials. The desicision whether or not to import the results back into Pinnacle
// is determined in the python command.
//      Jared "heavy" Ohrt 2011


////////////    DEBUG MODE    ////////////
Store.FloatAt.DEBUG = 0;   // debug mode on  = 1


////////////    Definitions   ////////////
//    Python Script
Store.StringAt.Pyfile = "/home/p3rtp/ohrt/StandAloneScripts/SA_ComputeGamma.py";

//    Temp Scripts
Store.StringAt.TempScript1 = "/home/p3rtp/ohrt/tmpScripts/";
Store.At.TempScript1.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.TempScript1.AppendString = "TEMP1\.Script";

Store.StringAt.TempScript2 = "/home/p3rtp/ohrt/tmpScripts/";
Store.At.TempScript2.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.TempScript2.AppendString = "TEMP2\.Script";

////////////    Import   ////////////
SavePlan = "";

Store.StringAt.gamma = "python ";
Store.At.gamma.AppendString = Store.At.Pyfile.String;
Store.At.gamma.AppendString = " gatherinfo ";
Store.At.gamma.AppendString = PlanInfo.PlanPath;
Store.At.gamma.AppendString = " ";
Store.At.gamma.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.gamma.AppendString = " '";
Store.At.gamma.AppendString = PlanInfo.LastName;
Store.At.gamma.AppendString = "' ";
Store.At.gamma.AppendString = Store.At.TempScript1.String;
Store.At.gamma.AppendString = " ";
Store.At.gamma.AppendString = Store.At.TempScript2.String;
IF.Store.FloatAt.DEBUG.THEN.WarningMessage = Store.At.gamma.String;
SpawnCommand = Store.At.gamma.String;

Script.ExecuteNow = Store.At.TempScript1.String; // TempScript2 is executed by TempScript1,
clever, no?
```

```
Store.StringAt.rm = "rm ";
Store.At.rm.AppendString = Store.At.TempScript1.String;
Store.At.rm.AppendString = " ";
Store.At.rm.AppendString = Store.At.TempScript2.String;
IF.Store.FloatAt.DEBUG.THEN.Store.At.Nothing.ELSE.SpawnCommand = Store.At.rm.String;

////////SavePlan = ""; ////TempScript will Save plan, then replace binary file. Don't want to save
here
//WarningMessage = "The Gamma Computation has been completed, and the Plan has been saved.
It will now be closed so that the dose grid import can be completed";
IF.Store.FloatAt.DEBUG.THEN.Store.At.Nothing.ELSE.Quit = "";
```

SA_ComputeGamma.py

```
#/usr/bin/env python
#   This script switches based on it's first argument. A .Script calls this function with action =
'gatherinfo'. The function
# builds gets info about the trials requested for the 3Dgamma calculation then builds a .Script that
gets needed information from
# Pinnacle, and then calls this function again with the action = 'compute'. The 3D gamma caculation
is currently doen using Matlab
# on a remote computer. This .Script creates a text file containing all of the info needed to do the
Gamma Analysis, and then
# waits for the remote script to create the resulting .Script and .binary files in a local (Pinnacle)
directory. It then writes
# a .Script that will semi-permenantly import the gamma dose grids, save, and Quit the plan
(necessary for semipermenant import).
import os
import glob
import time
from Tkinter import *
import sys
sys.path.append('/home/p3rtp/ohrt')
sys.path.append('/home/p3rtp/ohrt/PtScripts')

import readTrial

from dictImport import gammapreamb
from dictImport import gammaext
from dictImport import reportfilepreamb
from dictImport import rptext
from dictImport import dosefilepreamb
from dictImport import binext
from dictImport import trialtmp
from dictImport import storedir

########################gamma criteria
from dictImport import dDose
from dictImport import dDist
from dictImport import threshold
########################################


def TrialInfo(input):
   t = input.split('-')
   del t[-1]
   tindex = []
   totmu = []
   numfx = []
```

```
    for i,v in enumerate(t):
        t[i] = v.split('_')
        tindex.append(int(t[i][0]))
        numfx.append(int(t[i][1]))
        t[i][2] = t[i][2].split(',')
        tot = 0
        for j,w in enumerate(t[i][2]):
            tot += int(w)
            #t[i][1][j] = int(w)
        #totmu.append(sum(t[i][1]))
        totmu.append(tot)

    return tindex , totmu, numfx

def writeTrial(rptfile,dosefile,stmp,gname,tname,ptcopy=0):
    f = open(rptfile, 'r')
    info = f.readlines()
    f.close()

    del info[0:15]
    info = str(info[0].replace(' ','')).split(',')

    #### create and write Script
    stmp.write('TrialList .CreateChild = "Add New Trial";\n')
    stmp.write('TrialList .Last .Name = "Gamma('+ tname + '->' + gname +')";\n')
    stmp.write('TrialList .Last .MakeCurrent = "1";\n')
    stmp.write('TrialList .Current .PatientRepresentation .CtToDensityTableAccepted = "1";\n')

    ### Set Dose Grid
    stmp.write('TrialList .Current .DoseGrid .VoxelSize .X = ' + info[0] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .VoxelSize .Y = ' + info[1] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .VoxelSize .Z = ' + info[2] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Dimension .X = ' + info[3] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Dimension .Y = ' + info[4] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Dimension .Z = ' + info[5] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Origin .X = ' + info[6] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Origin .Y = ' + info[7] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Origin .Z = ' + info[8] + ';\n')
    stmp.write('TrialList .Current .DoseGrid .Display2d = 1;\n')

    ### Create Beam
    stmp.write('CreateNewBeam = "Add New Beam";\n')
    stmp.write('TrialList .Current .BeamList .Current .Name = "' + info[9] + '%_' + info[10] + 'mm
(thresh' + info[11] + '%)";\n')
    if (info[12].find('100.0') != -1):
        stmp.write('TrialList .Current .BeamList .Current .FieldID = "100%";\n')
    else:
        stmp.write('TrialList .Current .BeamList .Current .FieldID = "' + info[12] + '%";\n')
```

```
    stmp.write('TrialList .Current .BeamList .Current .Display2d = 0;\n')

    ### Create Prescription
    stmp.write('TrialList .Current .PrescriptionList .#"#0" .Name = "3DGamma-dummy";\n')
    stmp.write('TrialList .Current .PrescriptionList .Current .Method = "Set Monitor Units";\n')
    stmp.write('TrialList .Current .PrescriptionList .Current .RequestedMonitorUnitsPerFraction = "
1";\n')

    stmp.write('TrialList .Current .ComputeUncomputedBeams = "1";\n')
    #stmp.write('SavePlan = "";\n')
    ##stmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .Selected = 1;\n')
    ##stmp.write('PluginManager .PlanEvalPlugin .TrialList .Last .LineType = "Medium Dashed";\n')
    #stmp.write('Store.StringAt.SetDGrid = "python ' + os.path.abspath(__file__) + ' switchgrid ";\n')
    stmp.write('Store.At.Grids.AppendString = TrialList.Current.Index;\n')
    stmp.write('Store.At.Grids.AppendString = "+' + dosefile + ',";\n')
    #stmp.write('SpawnCommand = Store.At.SetDGrid.String;\n')

    return info

def waitforit(file):
    there = os.path.isfile(file)
    while (there == 0):   ###### potentially infinite loop
        time.sleep(1)
        there = os.path.isfile(file)

    sz1 = os.path.getsize(file)
    time.sleep(0.5)
    sz2 = os.path.getsize(file)

    while (sz1 != sz2):   ###### potentially infinite loop
        sz1 = sz2
        time.sleep(0.5)
        sz2 = os.path.getsize(file)

    print "\tFOUND:\t",file
    return file

def AskList(trialnames=[],MRN='', LastName=''):
    root=Tk()
    root.title('3D Gamma: '+str(MRN)+LastName)

    F1 = Frame()
    Label(F1,text='Standard').grid(row=0,column=0)
    Label(F1,text='Trial Name').grid(row=0,column=1)
    Label(F1,text='Compare To').grid(row=0,column=2)

    R = len(trialnames)
```

```
    std = IntVar()
    states = []
    for i in range(R):
      Radiobutton(F1,text='',variable=std,value=i).grid(row=i+1,column=0)
      Label(F1,text = trialnames[i]).grid(row=i+1,column=1, sticky=W)
      comp = IntVar()
      Checkbutton(F1,text='',variable=comp).grid(row=i+1,column=2)
      states.append(comp)
      g=i+2
    Label(F1,text='').grid(row=g,column=1)
    Button(F1, text='OK', command=root.destroy).grid(row=g+1,column=1)

    F1.pack(side=LEFT, fill=X)
    mainloop()
    measured = map((lambda comp: comp.get()), states)
    measured[std.get()] = 0
    return std.get(), measured

def CTGrid(planpath,pinnpath):
    ptpath = '/'.join(planpath.split('/')[:-1]) #removes plan folder from path
    planID = filter(lambda x: x.isdigit(), planpath.split('/')[-1]) # get plan id from path

    numchar = [ '1','2','3','4','5','6','7','8','9','0','.','-']


    ptfile = os.path.join(pinnpath,ptpath,'Patient')
    p = open(ptfile, 'r')
    ptinfo = p.readlines()
    p.close()

    for lnum, line in enumerate(ptinfo):
          if (line.find('PlanID = '+ planID) != -1):
             if (line.find('NextUnique') == -1):
                for ct in range(lnum,len(ptinfo)):
                   if (ptinfo[ct].find('PrimaryCTImageSetID') != -1):
                      imgID = filter(lambda x: x.isdigit(), ptinfo[ct])
                      break

    imgfile = os.path.join(pinnpath,ptpath,str('ImageSet_'+imgID+'.header'))

    #print imgfile
    i = open(imgfile, 'r')
    imginfo = i.readlines()
    i.close()

    dimx = ''.join(filter(lambda x: x in numchar, imginfo[4]))
    dimy = ''.join(filter(lambda x: x in numchar, imginfo[4]))
```

```
    dimz = ''.join(filter(lambda x: x in numchar, imginfo[6]))
    ctdim = ','.join([dimx,dimy,dimz])

    gridx = ''.join(filter(lambda x: x in numchar, imginfo[13]))
    gridy = ''.join(filter(lambda x: x in numchar, imginfo[14]))
    gridz = ''.join(filter(lambda x: x in numchar, imginfo[15]))
    ctgrid = ','.join([gridx,gridy,gridz])

    origx = ''.join(filter(lambda x: x in numchar, imginfo[17]))
    origy = ''.join(filter(lambda x: x in numchar, imginfo[18]))
    origz = ''.join(filter(lambda x: x in numchar, imginfo[19]))
    ctorig = ','.join([origx,origy,origz])

    return ctdim, ctgrid, ctorig

def GatherInfo():
    planpath = sys.argv[2]
    pinnpath = os.getcwd()
    tpath = os.path.join(pinnpath,planpath,'plan.Trial')
    binpath = os.path.join(pinnpath,planpath,'plan.Trial.binary.')


    mrn = sys.argv[3]

    lastname = str(sys.argv[4]).replace('restored','')
    lastname = lastname.replace(' ','')
    lastname = lastname[0].upper() + lastname[1:].lower()

    tmpscript1 = sys.argv[5]
    tmpscript2 = sys.argv[6]




    ###### read Trial data from plan.Trial
    TrialData = readTrial.go(tpath,trialtmp)

    trialnames = []
    for trl_i in range(len(TrialData)):
        trialnames.append(TrialData[trl_i][0])

    =gold, samp = AskList(trialnames,mrn,lastname)

    tneed = [gold]
    for s,v in enumerate(samp):
        if (v == 1):
            tneed.append(s)
```

```
    #print 'gold =',gold,'\tsamp =',samp,'\ttneed =',tneed

    tmp = open(tmpscript1,'w')
    tmp.write('Store.StringAt.GammaInfo = "";\n')

    for t in range(len(tneed)):
        tmp.write('Store.At.GammaInfo.AppendString = TrialList.#"#' + str(tneed[t]) + '".Index;\n')
        tmp.write('Store.At.GammaInfo.AppendString = "\\_";\n')
        tmp.write('TrialList.#"#' + str(trl_i) + '".PrescriptionList.Current = TrialList.#"#' + str(tneed[t]) +
'".BeamList.#"#0".PrescriptionName;\n')
        tmp.write('Store.At.GammaInfo.AppendString = TrialList.#"#' + str(tneed[t]) +
'".PrescriptionList.Current.NumberOfFractions;\n')
        tmp.write('Store.At.GammaInfo.AppendString = "\\_";\n')
        for bm in range(len(TrialData[tneed[t]][7])):
            if (bm != 0):
                tmp.write('Store.At.GammaInfo.AppendString = ",";\n')
            tmp.write('Store.At.GammaInfo.AppendString = TrialList.#"#' + str(tneed[t]) +
'".BeamList.#"#' + str(bm) + '".MonitorUnits;\n')
        tmp.write('Store.At.GammaInfo.AppendString = "-";\n')


    tmp.write('Store.StringAt.GammaComp = "python '+ os.path.abspath(__file__) + ' compute ' +
planpath + ' ' + str(mrn) + ' ' + lastname + ' ";\n')
    tmp.write('Store.At.GammaComp.AppendString = "\' ";\n')
    tmp.write('Store.At.GammaComp.AppendString = Store.At.GammaInfo.String;\n')
    tmp.write('Store.At.GammaComp.AppendString = "\' ";\n')
    tmp.write('Store.At.GammaComp.AppendString = "' + tmpscript2 + '";\n')

    #tmp.write('WarningMessage = Store.At.GammaComp.String;')
    tmp.write('SpawnCommand = Store.At.GammaComp.String;')
    tmp.write('Script.ExecuteNow = "'+ tmpscript2 + '";\n')

    tmp.close()
    os.system('chmod 755 ' + tmpscript1 )

def RequestCalculation():
    ############################ Arguments
    planpath = sys.argv[2]
    pinnpath = os.getcwd()
    tpath = os.path.join(pinnpath,planpath,'plan.Trial')
    binpath = os.path.join(pinnpath,planpath,'plan.Trial.binary.')
    isodosepath = os.path.join(pinnpath,planpath,'plan.Isodose')

    ctdim, ctgrid, ctorig = CTGrid(planpath,pinnpath)

    mrn = sys.argv[3]

    lastname = str(sys.argv[4]).replace('restored','')
```

```
    lastname = lastname.replace(' ','')
    lastname = lastname[0].upper() + lastname[1:].lower()

    #### trialinfo input [index_numfx_mu1,mu2,mu3-index2_numfx2__,mu21,mu22,mu23.....]
    tindex, totmu, numfx = TrialInfo(sys.argv[5].replace(' ',''))

    tmpscript2 = sys.argv[6]

    ###### read Trial data from plan.Trial using shell script
    TrialData = readTrial.go(tpath,trialtmp)

    ##############################################

    gammaprefix = gammapreamb + str(mrn) + lastname + '_'
    reportfileprefix = reportfilepreamb + str(mrn) + lastname + '_'
    dosefileprefix = dosefilepreamb + mrn + '.' +  lastname + '.'

    MLpreamb = str(dDose) + ',' + str(dDist) + ',' + str(threshold) + ","



    ############### delete trials not in tindex from TrialData, replace binary indices with
unimported dose file paths
    ###############where appropriate, and add full path to remaining binary indices

    for i in range(len(TrialData)-1,-1,-1): #####iterates backwards
        if (i not in tindex):
            del TrialData[i]
        else:
            for j in range(len(tindex)):
                if (i == tindex[j]):
                    TrialData[i][4] = totmu[j]
                    TrialData[i][5] = numfx[j]

            for m in range(len(TrialData[i][6])):
                try:
                    TrialData[i][6][m] = binpath + "0"*(3-len(str(TrialData[i][6][m]))) + str(TrialData[i][6][m])
                except:
                    pass

            TrialData[i][0] = str(TrialData[i][0].replace('_Import','')).replace('Update','')

    #print 'TrialData = '
    #for i in range(len(TrialData)): print TrialData[i]

    tmp = open(tmpscript2, 'w')
    tmp.write('Store.StringAt.Grids = "";\n')
```

```
    gold = TrialData[0]

    for m in range(1,len(TrialData)): ## create gtmp files to pass info to calculation
        filesuffix = str(str(gold[0]).replace('(','')).replace(')','') + '-' +
str(str(TrialData[m][0]).replace('(','')).replace(')','')
        gammatmp = gammaprefix + filesuffix + gammaext
        dosefile = dosefileprefix + filesuffix + binext
        reportfile = reportfileprefix + filesuffix  + rptext

        ####### write CT grid parameters here
        gtmp = open(gammatmp, 'w')
        gtmp.write(str(mrn) + '\n')
        gtmp.write(lastname + '\n')
        gtmp.write(dosefile + '\n')
        gtmp.write(reportfile + '\n')
        gtmp.write( ctdim + '\n')
        gtmp.write( ctgrid + '\n')
        gtmp.write( ctorig + '\n')
        gtmp.write(str(dDose) + '\n')
        gtmp.write(str(dDist) + '\n')
        gtmp.write(str(threshold) + '\n')
        gtmp.write(str(gold[0]) + '\n');
        gtmp.write(str(gold[1])[1:-1] + '\n');
        gtmp.write(str(gold[2])[1:-1] + '\n');
        gtmp.write(str(gold[3])[1:-1] + '\n');
        gtmp.write(str(gold[4]) + '\n');
        gtmp.write(str(gold[5]) + '\n');
        gtmp.write(str(gold[6])[1:-1].replace("'",'') + '\n');
        gtmp.write(str(gold[7])[1:-1] + '\n');
        gtmp.write(str(TrialData[m][0]) + '\n');
        gtmp.write(str(TrialData[m][1])[1:-1] + '\n');
        gtmp.write(str(TrialData[m][2])[1:-1] + '\n');
        gtmp.write(str(TrialData[m][3])[1:-1] + '\n');
        gtmp.write(str(TrialData[m][4]) + '\n');
        gtmp.write(str(TrialData[m][5]) + '\n');
        gtmp.write(str(TrialData[m][6])[1:-1].replace("'",'') + '\n');
        gtmp.write(str(TrialData[m][7])[1:-1] + '\n');
        gtmp.close()
        print '\tSAVED: ',gammatmp

def SwitchDoseGrid():
    import shutil
    planpath = sys.argv[2]
    pinnpath = os.getcwd()
    tpath = os.path.join(pinnpath,planpath,'plan.Trial')
    binpath = os.path.join(pinnpath,planpath,'plan.Trial.binary.')
```

```
    Grids = str(sys.argv[3].replace(' ','')).split(',')
    del Grids[-1] # extra comma at end of string
    #print 'Grids = ',Grids

    TrialData = readTrial.go(tpath,'/home/p3rtp/ohrt/GammaTrialInfo.txt')

    for i in range(len(Grids)):
       Grids[i] = Grids[i].split('+')
       n = str(int(TrialData[int(Grids[i][0])][6][0]))
       if (len(n) == 1):
          Grids[i].append(binpath + '00' + str(n))
       if (len(n) == 2):
          Grids[i].append(binpath + '0' + str(n))
       if (len(n) == 3):
          Grids[i].append(binpath + str(n))
       print '\tReplacing \n\t\t',Grids[i][2],' \n\t\twith \n\t\t',Grids[i][1]
       shutil.move(Grids[i][1],Grids[i][2])
       #shutil.copyfile(Grids[i][1],Grids[i][2])
       #os.system(str('rm ' + Grids[i][1]))
       #os.rename(Grids[i][1],Grids[i][2]) # doesn't work bc dir is mounted to Tinkerbell, i think
    #print 'Grids = ',Grids
    print 'DONE\n'



def NoOpen():
    ######## try to compute gamma without opening plan
    ptpath = sys.argv[1]
    planpath = os.path.join(ptpath, sys.argv[2])
    tpath = os.path.join(pinnpath,planpath,'plan.Trial')
    binpath = os.path.join(pinnpath,planpath,'plan.Trial.binary.')


    mrn = sys.argv[3]

    lastname = str(sys.argv[4]).replace('restored','')
    lastname = lastname.replace(' ','')
    lastname = lastname[0].upper() + lastname[1:].lower()

    tmpscript1 = sys.argv[5]
    tmpscript2 = sys.argv[6]




    ###### read Trial data from plan.Trial
    TrialData = readTrial.go(tpath,trialtmp)
```

```
    trialnames = []
    for trl_i in range(len(TrialData)):
        trialnames.append(TrialData[trl_i][0])

    gold, samp = AskList(trialnames,mrn,lastname)

    tneed = [gold]
    for s,v in enumerate(samp):
        if (v == 1):
            tneed.append(s)



action = sys.argv[1]
if (action == 'gatherinfo'):
    print '\n3D GAMMA\nGathering Information....'
    GatherInfo()
elif (action == 'compute'):
    print 'Requesting Calculation...'
    RequestCalculation()
elif (action == 'setgrid'):
    print 'DONE'
    #print 'Setting Dose grids...'
    #SwitchDoseGrid()
elif os.path.isdir(action):
    NoOpen()
```

dictPinnGamm.py (remote server)

```
#!/usr/bin/env python
import os

# LOCAL directories
ptdir = '/home/johrt/Patients'
locgammadir = '/home/johrt/gammatools/gammatmp/'
locgammatmp = os.path.join(locgammadir,'gamma.tmp')



# Shared Directories
localmount = '/home/johrt/raptormount'

# LOCAL files
statusfile = '/home/johrt/gammatools/calc.status'
matlog = '/home/johrt/gammatools/MatlabLog.log'

# PINNACLE directories
pinnscriptdir = '/home/p3rtp/ohrt/PtImport/'
pinnstoredir = '/home/p3rtp/ohrt/PtStore/'
pinngammadir = '/home/p3rtp/ohrt/gammatmp/'

# Log Files
demonlog = '/home/johrt/gammatools/Demonlog.log'

# FTP settings
defaulthost = 'raptor'
user = 'p3rtp'
password = 'p3plan'


# dictionaries
manuf = { 'Pinnacle3':'pinn', 'clinical':'clinical', 'Aria
RadOnc':'eclipse','MOSAIQ':'mosaiq','2300IX':'4DTC_Tx', '2100EX':'4DTC_Tx', '2100C/D':'4DCT',
'AcQSimCT':'' }
type = { 'RTPLAN':'RP' , 'RTDOSE':'RD' , 'RTSTRUCT':'RS' , 'RTRECORD':'RT', 'RTIMAGE': 'RI' , 'CT':'CT' }
mach = {'Varian 2109':'Varian 2109: 2010-11-22 10:23:13'}
mode = { 'STATIC': 'SnS', 'DYNAMIC':'ARC'}
edom = { 'SnS':'Step & Shoot MLC' , 'ARC':'Dynamic Arc'}
```

GammaDemon.bash (remote server)

```
#!/bin/bash


watchdir='/home/johrt/raptormount'


#if [ $# -eq 1 ]; then
#    watchdir=$1
#else
#
#fi

echo -e '\nGammaDemon is awake and monitoring '$watchdir

njobs=0

while [ $njobs = 0 ]; do
   sleep 1
   njobs=`ls $watchdir| grep '.gtmp' | wc -l`
   if [ $njobs != 0 ]; then
      echo -e '\tFOUND: '$njobs' job(s)'
      worklist=`ls $watchdir| grep '.gtmp'`
      sleep 1 #to make sure gtmp file is finished writing
      pycmd='python /home/johrt/gammatools/GammaPrep.py'
      for line in $worklist; do
         tmpfile=$watchdir'/'$line
         pycmd=$pycmd" "$tmpfile
      done

      $pycmd # launches GammaPrep.py with file arguments
      #
      #for line in $worklist; do #rm processed gtmp files
      #    rm $watchdir'/'$line
      #done
      njobs=0 # start loop again
      echo -e "\nGammaDemon's gaze oncemore falls upon "$watchdir
   fi


done
```

GammaPrep.py (remote server)

```
#!/usr/bin/env python
import sys
sys.path.append('/home/johrt/gammatools')
import os
import glob
import time
import ftplib
#import ftpmirror
#import pinnftp

from dictPinnGamma import pinngammadir
from dictPinnGamma import locgammadir
from dictPinnGamma import locgammatmp
from dictPinnGamma import demonlog
from dictPinnGamma import statusfile
from dictPinnGamma import matlog
from dictPinnGamma import localmount

from dictPinnGamma import defaulthost
from dictPinnGamma import user
from dictPinnGamma import password

###IsVerbose = True
IsVerbose = False


def getbinaryfiles(binfiles,hostname=defaulthost, username=user, pw= password):
   if IsVerbose: print 'Get Binary files',

   f = ftplib.FTP(hostname,username,pw)
   binstr='';
   for i in range(len(binfiles)):
      print binfiles[i]
      binfiles[i] = binfiles[i].replace(' ','')
      if (i == len(binfiles[i])-1):
         binfiles[i] = binfiles[i][:-1] #stupid endline character
      #print 'binfiles[',i,'] = ',binfiles[i][-2:]
      newdata = []

      f.retrbinary('RETR '+ binfiles[i],newdata.append)
      newfile = os.path.join(locgammadir,os.path.basename(binfiles[i]))

      if IsVerbose: print 'Got binary'
      n = open(newfile,'wb')
      n.writelines(newdata)
      n.close()
```

```
        binfiles[i] = binfiles[i].replace(str(os.path.dirname(binfiles[i])+'/'),locgammadir)
        binstr = binstr + str(binfiles[i]) + ','
    #binfiles =str(str(binfiles)[1:-1].replace("'","")).replace('\n','')
    binstr = binstr[:-1] + '\n'# don't want last comma
    f.close()
    return binstr

def waitformatlab(statusfile):
    if IsVerbose: print 'Wait For Matlab'

    starttime = os.stat(statusfile).st_mtime
    time.sleep(30);
    crnttime = os.stat(statusfile).st_mtime
    while starttime == crnttime:
        time.sleep(30);
        crnttime = os.stat(statusfile).st_mtime


    s = open(statusfile,'r')
    state = s.readline()
    s.close()

    if (state[0].find('ERROR') != -1):
        errmsg = state[0].split(':'[1])
    else:
        errmsg = 'none'

    #matlab will delete the gamma.tmp file when it's done

    return state, errmsg

def RequestCalc(filelist):
    if IsVerbose: print 'Request Calc'

    for file_i in filelist:
        zeit = time.ctime()
        print zeit,'  PROCESSING: ',os.path.basename(file_i)
        try:
            L=open(demonlog,'a')
            L.write(zeit + '  PROCESSING: ' + os.path.basename(file_i) + '\n')
            L.close()
        except:
            pass
        ####### read in .gtmp file
        #print file_i
        ftmp = open(file_i,'r')
        info = ftmp.readlines()
```

```
        ftmp.close()

        pinndose = info[2]
        localdose = os.path.join(localmount,os.path.basename(pinndose))
        info[2] = localdose

        pinnrpt = info[3]
        localrpt = os.path.join(localmount,os.path.basename(pinnrpt))
        info[3] = localrpt

        #info[13] = getbinaryfiles( info[13][:-1].split(','))
        #info[21] = getbinaryfiles( info[21][:-1].split(','))
        info[16] = getbinaryfiles( info[16][:-1].split(','))
        info[24] = getbinaryfiles( info[24][:-1].split(','))

        if IsVerbose: print 'Got ALL binarys'

        ltmp = open(locgammatmp, 'w')
        for i,v in enumerate(info):
           ltmp.write(v )    #+ '\n')
        ltmp.close()
        #time.sleep(1)

        matlabcmd = "matlab -nodesktop -nosplash -r \"PinnGamma3d('" + locgammatmp + "')\" -
logfile \"" + matlog + "\""
        os.system(matlabcmd)
        #waitformatlab(locgammatmp)
        state, errmsg = getCalcState(statusfile)
        if (state == 'ERROR'):
           try:
              L=open(demonlog,'a')
              L.write('\tERROR: ' + errmsg + '\n')
              L.close()
              print '\tERROR: ',errmsg
           except:
              pass
        else:
           print '\tSAVED: ',localdose
           print '\tSAVED: ',localrpt

        for j,w in enumerate(glob.glob(os.path.join(locgammadir,'*'))):
           os.remove(w)
        os.system(str('rm '+ file_i))
     #os.unsetenv('MATGAM')

     return filelist

def getCalcState(statusfile):
```

```
    if IsVerbose: print 'Get Calc State'

    s = open(statusfile,'r')
    state = s.readline()
    s.close()

    if (state[0].find('ERROR') != -1):
        errmsg = state[0].split(':'[1])
    else:
        errmsg = 'none'


    return state, errmsg

def demon(watchdir=locgammadir, getdir=pinngammadir, hostname=defaulthost, username=user,
pw= password):
    print 'Demon Awake and Monitoring: ',hostname,' ',getdir,'\n'

    f = ftplib.FTP(hostname,username,pw)
    f.cwd(getdir)

    listing = []

    while (len(listing) == 0):
        time.sleep(1)
        listing = []
        try:
            f.retrlines('NLST', listing.append)
        except ftplib.error_perm:
            pass

        if (len(listing) > 0):
            print "\nFOUND: ",listing[0]
            time.sleep(1)

            crntfile = listing[0]
            info = []
            ####### read in gtmp file
            f.retrlines('RETR ' + crntfile,info.append)

            pinndose = info[2]
            localdose = os.path.join(locgammadir,os.path.basename(pinndose))
            info[2] = localdose

            pinnrpt = info[3]
            localrpt = os.path.join(locgammadir,os.path.basename(pinnrpt))
            info[3] = localrpt
```

```
        info[13] = getbinaryfiles(f, info[13].split(','))
        info[21] = getbinaryfiles(f, info[21].split(','))


        ltmp = open(locgammatmp, 'w')
        for i,v in enumerate(info):
           ltmp.write(v + '\n')
        ltmp.close()
        time.sleep(1)

        matlabcmd = "matlab -automation -r PinnGamma3d('" + locgammatmp + "')"
        os.system(matlabcmd)

        waitforem(localdose,localrpt)
        locdose = open(localdose, 'rb')
        f.storbinary('STOR ' + pinndose,locdose)
        locdose.close()
        print '\tSENT: ',localdose
        locrpt = open(localrpt, 'rb')
        f.storlines('STOR ' + pinnrpt,locrpt)
        locrpt.close()
        print '\tSENT: ',localrpt

        #for j,w in enumerate(glob.glob(os.path.join(locgammadir,'*'))):
        #    os.remove(w)

        listing = []
        #try:
        f.retrlines('NLST', listing.append)
        #except ftplib.error_perm:
           #pass

        while (crntfile in listing):
              time.sleep(1)
              listing = []
              try:
                 f.retrlines('NLST', listing.append)
              except ftplib.error_perm:
                 pass
        listing = []
    return listing


files=[]
for arg_i in range(1,len(sys.argv)):
    files.append(str(sys.argv[arg_i]))

RequestCalc(files)
```

PinnGamma3D.m (remote server Matlab)

```
function [results,gold,sample]=PinnGamma3d(infofile)
%tic;
addpath( '/home/johrt/gammatools')
%newgrid = [0.2,0.2,0.2]; % grid are interpolated to this size


%%%%%% read in variables from infofile
f=fopen(infofile, 'r');

MRN = fgetl(f);
LastName = fgetl(f);
dosefile = fgetl(f);
reportfile = fgetl(f);
CTsize = cellfun(@str2double,regexp(fgetl(f),',','split'));
CTgrid = cellfun(@str2double,regexp(fgetl(f),',','split')); % will interpolate to this resolution after
calc
newgrid = [CTgrid(1),CTgrid(2),CTgrid(2)]; % want uniform interpolation for gamma calc
CTorig = cellfun(@str2double,regexp(fgetl(f),',','split'));
CTorig = convertOrigin(CTorig,CTsize,newgrid);

dD = str2double(fgetl(f)); %decimal form
dr = str2double(fgetl(f)); %in cm
multi_crit = true;
dD = [0.03, 0.02, 0.01];%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% manual override
dr = [0.3, 0.2, 0.1];%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% manual override

cutoff = str2double(fgetl(f));

cutoff = 0;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
newgrid = [0.1,0.1,0.1];%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%% read in gold data
gold.Name = fgetl(f);
gold.grid = cellfun(@str2double,regexp(fgetl(f),',','split'));
gold.size = cellfun(@str2double,regexp(fgetl(f),',','split'));
gold.orig = cellfun(@str2double,regexp(fgetl(f),',','split'));
gold.orig = convertOrigin(gold.orig, gold.size, gold.grid);
gold.totalMU = str2double(fgetl(f));
gold.NumFx = str2double(fgetl(f));
gold.filelist = strrep(regexp(fgetl(f),',','split'),' ','');
gold.beamweights = cellfun(@str2double,regexp(fgetl(f),',','split'));
gold.dose = getTrialDoseGrid(gold.filelist, gold.totalMU, gold.NumFx, gold.size, gold.beamweights);
gold.maxdose = max(gold.dose(:));
```

```
%%%%%%% read in sample data
sample.Name = fgetl(f);
sample.grid = cellfun(@str2double,regexp(fgetl(f),',','split'));
sample.size = cellfun(@str2double,regexp(fgetl(f),',','split'));
sample.orig = cellfun(@str2double,regexp(fgetl(f),',','split'));
 sample.orig = convertOrigin(sample.orig, sample.size, sample.grid);
sample.totalMU = str2double(fgetl(f));
sample.NumFx = str2double(fgetl(f));
sample.filelist = strrep(regexp(fgetl(f),',','split'),' ','');
sample.beamweights = cellfun(@str2double,regexp(fgetl(f),',','split'));
sample.dose = getTrialDoseGrid(sample.filelist, sample.totalMU, sample.NumFx, sample.size,
sample.beamweights);
sample.maxdose = max(sample.dose(:));

fclose(f);

%%%%%%% make grids the same (sample has overlap)

gold.min = [1,1,1];
gold.max = gold.size;
[gold]=interpgrid(gold,newgrid);


GindexSorig = ceil(((sample.orig - gold.orig)./gold.grid)+1);
sample.min = ((gold.orig - sample.orig + (GindexSorig - 1).*gold.grid)./sample.grid)+1;

samplemaxpos = sample.orig + (sample.size-1).*sample.grid;


GindexSmax = floor(((samplemaxpos - gold.orig)./gold.grid)+1);
sample.max = ((gold.orig - sample.orig + (GindexSmax - 1).*gold.grid)./sample.grid)+1;

[sample]=interpgrid(sample,newgrid);


pad = ceil(max(dr(:))./newgrid).*newgrid; % in distance

minpos = max(gold.orig,sample.orig + pad);
gold.min = uint32(((minpos - gold.orig)./gold.grid)+1);
sample.min = uint32(((minpos - pad - sample.orig)./sample.grid)+1);

maxgold = gold.orig + (gold.size-1).*gold.grid;
maxsample = sample.orig + (sample.size-1).*sample.grid;

maxpos = min(maxgold,maxsample-pad);
gold.max = uint32(((maxpos - gold.orig)./gold.grid)+1);
sample.max = uint32(((maxpos + pad - sample.orig)./sample.grid)+1);
```

```
%clear('maxgold','maxsample','maxpos','minpos');

gold.dose = gold.dose(gold.min(1):gold.max(1),gold.min(2):gold.max(2),gold.min(3):gold.max(3));
gold.min = double(gold.min);
gold.max = double(gold.max);
gold.orig = gold.orig + (gold.min - 1).*gold.grid;
sample.dose =
sample.dose(sample.min(1):sample.max(1),sample.min(2):sample.max(2),sample.min(3):sample.m
ax(3));
sample.min = double(sample.min);
sample.max = double(sample.max);
sample.orig = sample.orig + (sample.min - 1).*sample.grid;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% compute the gamma -dD in %,dr
in mm;

if multi_crit
   for c=1:length(dD)
      display(sprintf('%u%%-%umm',uint8(dD(c)*100),uint8(dr(c)*10))); %display "dD%-dRmm"

      [results]=gamuh(dD(c),dr(c),cutoff,gold,sample);
       %display('GAMMA done');
       %toc

      results.MRN = MRN;
      results.LastName = LastName;
      results.gold.Name = gold.Name;
      results.gold.max = max(gold.dose(:));
      results.sample.Name = sample.Name;
      results.sample.max = max(sample.dose(:));

      reportname = strcat(reportfile(1:end-4),'.',num2str(dD(c)*100),'%-
',num2str(dr(c)*10),'mm',reportfile(end-3:end));
      ReportResults(results,reportname);
      %writebinarydosegrid(results.gamma,'pinn', dosefile, results.size(1), results.size(2),
results.size(3))
   end
else
      [results]=gamuh(dD,dr,cutoff,gold,sample);
       %display('GAMMA done');
       %toc

      results.MRN = MRN;
      results.LastName = LastName;
      results.gold.Name = gold.Name;
      results.gold.max = max(gold.dose(:));
```

```
      results.sample.Name = sample.Name;
      results.sample.max = max(sample.dose(:));

      reportname = strcat(reportfile(1:end-4),'.',num2str(dD*100),'%-
',num2str(dr*10),'mm',reportfile(end-3:end));
      ReportResults(results,reportname);
      %writebinarydosegrid(results.gamma,'pinn', dosefile, results.size(1), results.size(2),
results.size(3))
end

 delete(infofile);
 %display('Done');
 %beep
 %toc
 exit
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
function orig = convertOrigin(orig,dim,grid)
% Converts origin from pinnacle to dicom coordinates and back. The
% conversion is symmetric. X is not effected.
   %orig(2)=-1*(orig(2) + (dim(2)-1)*grid(2));
   %orig(3)=-1*(orig(3) + (dim(3)-1)*grid(3));

   orig(2)= -1*(orig(2) + (dim(2)-1)*grid(2));
%     orig(3)= orig(3) + (dim(3)-1)*grid(3);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
function [OUT]=interpgrid(IN,neugrid)
   %tic;
   OUT=IN;
   OUT.dose = [];
   OUT.grid=neugrid;
   OUT.orig=IN.orig+(IN.min-1).*IN.grid;


   %%%%%%%%%% 3D, memory issues mean this can't interp as fine as 2D X 2
   %%%%%%%%use meshgrid to get vectors for interpolation

%[y,x,z]=meshgrid(IN.min(2):(neugrid(2)/IN.grid(2)):IN.max(2),IN.min(1):(neugrid(1)/IN.grid(1)):IN.
max(1),IN.min(3):(neugrid(3)/IN.grid(3)):IN.max(3));
   %v=[x(:),y(:),z(:)];

   %OUT.dose=zeros(length(unique(x)),length(unique(y)),length(unique(z)));
   %OUT.dose(:)=interp3(IN.dose,permute(v(:,2),[2 1]),permute(v(:,1),[2 1]),permute(v(:,3),[2 1]));
```

```
   %OUT.size=size(OUT.dose);



   %%%%%%%%%%% 2D x 2   more efficient memory usage, allows finer grid
   %%%%%%%%%%% interp
   x = IN.min(1):neugrid(1)/IN.grid(1):IN.max(1);
   y = IN.min(2):neugrid(2)/IN.grid(2):IN.max(2);
   z = size(IN.dose,3);
   step1 = zeros(length(x),length(y),z);

   for Z=1:z
      step1(:,:,Z) = interp2(IN.dose(:,:,Z),y(:)',x(:));
   end

   x = size(step1,1);
   y = 1:size(step1,2);
   z = IN.min(3):neugrid(3)/IN.grid(3):IN.max(3);

   clear IN;

   step1 = permute(step1,[3 2 1]);
   OUT.dose = zeros(length(z),length(y), x);

   for X=1:x
      OUT.dose(:,:,X) = interp2(step1(:,:,X),y(:)',z(:));
   end

   clear ('step1')
   OUT.dose = permute(OUT.dose, [3 2 1]);
   OUT.size = size(OUT.dose);

   %display(sprintf('Interp Done'));
   %toc
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
function [OUT]=interpZgrid(IN,neugrid,CTorig)
   %tic;
   OUT=IN;
   OUT.gamma = [];

   CTgoldmin = ceil(((OUT.orig - CTorig)./CTgrid)+1);
   OUT.min = ((CTorig - OUT.orig +(CTgoldmin - 1).*CTgrid)./OUT.grid)+1;

   CTgoldmax = floor(((OUT.orig - CTorig + (OUT.size - 1).*OUT.grid)./CTgrid)+1);
   OUT.max = ((CTorig - OUT.orig +(CTgoldmax - 1).*CTgrid)./OUT.grid)+1;
```

```
    OUT.orig = OUT.orig + (Out.min -1).*OUT.grid;
    OUT.grid=neugrid;


    x = size(IN.gamma,1);
    y = 1:size(IN.gamma,2);
    z = OUT.min:neugrid(3)/IN.grid(3):OUT.max;
    data = permute(IN.gamma,[3 2 1]);
    OUT.gamma = zeros(length(z),length(y), x);

    for X=1:x
       OUT.gamma(:,:,X) = interp2(data(:,:,X),y(:)',z(:));
    end

    OUT.gamma = permute(OUT.gamma, [3 2 1]);
    OUT.size = size(OUT.gamma);
    %display(sprintf('Z Interp Done'));
    %toc
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
function [G]=gamuh(dD,dr,cutoff,C,M)
%display('Matlab is Computing Gamma');
%toc
%tic;
dist=floor(dr./C.grid);
offset=(((C.orig-M.orig)./C.grid)+1)-1;

%%%%%%%calculate r
r=zeros((dist.*2)+1);
%[ry,rx,rz]=meshgrid(-1*dist:1:dist);
[ry,rx,rz]=meshgrid(-1*dist(2):dist(2),-1*dist(1):dist(1),-1*dist(3):dist(3));
r=sqrt((rx*C.grid(1)).^2+(ry*C.grid(2)).^2+(rz*C.grid(3)).^2);
r=(r./dr).^2;

%%% Normalized to x% maximum dose of C
norm = 1.00*max(C.dose(:));

M.dose = M.dose./norm;
C.dose = C.dose./norm;

G.gamma=zeros(size(C.dose));
%display(sprintf('nk = %u',size(C.dose,3)));
for k=1:size(C.dose,3)
     %display(sprintf('k = %u',k));
    for j=1:size(C.dose,2)
       for i=1:size(C.dose,1)
            if (C.dose(i,j,k)/C.maxdose) > cutoff
```

```
            mmin=uint32([i,j,k]+offset-dist);           mmax=uint32([i,j,k]+offset+dist);
            gamma=sqrt(((M.dose(mmin(1):mmax(1),mmin(2):mmax(2),mmin(3):mmax(3))-
C.dose(i,j,k))./dD).^2+(r));
            %gamma=sqrt(((M.dose(mmin(1):mmax(1),mmin(2):mmax(2),mmin(3):mmax(3))-
C.dose(i,j,k))./C.dose(i,j,k)./dD).^2+(r));

            G.gamma(i,j,k)=min(gamma(:));
            else
            G.gamma(i,j,k)=NaN;
            end
        end
    end
    %display(sprintf('k=%u',k));
%    toc
end
G.deltaDose = dD;
G.deltaDist = dr;
G.cutoff = cutoff;
G.grid=C.grid;
G.orig=C.orig;
G.size=size(G.gamma);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
function ReportResults(R, rptfile)

%%%%%%%%%%%%%%%%%%%%%%%%%get gamma stats
R.max=max(R.gamma(:));
R.min=min(R.gamma(:));
%R.mean=R.gamma(isfinite(R.gamma));
%R.mean=mean(R.gamma(:));
R.pass=numel(R.gamma(R.gamma<=1));
R.all=numel(R.gamma(~isnan(R.gamma)));
R.percentage=100*R.pass/R.all;
%R.mean=sum(R.mean(:))/R.all;
R.mean=sum(R.gamma(~isnan(R.gamma)))/R.all;
R.gamma(isnan(R.gamma))=0;
pcnt='%';


%display(sprintf('Patient= %s - %s \n------------------\n%s Max Dose (Gy)=%3.1f \n%s Max Dose
(Gy)=%3.1f \n\r\nGamma: %2.1f%s - %2.1fmm (cutoff %3.1f %s) \n------------------
\nPass=%u\r\nTotal=%u\r\nGamma Percentage= %3.1f %s\r\nMax= %3.1f \nMin= %3.1f \nMean =
%4.2f \nSize= %u X %u X %u\r\nVoxel Size = %3.3f X %4.3f X %4.3f\r\n
',R.MRN,R.LastName,R.gold.Name,R.gold.max,R.sample.Name,R.sample.max,R.deltaDose*100,pcnt
,R.deltaDist,R.cutoff,pcnt,R.pass,R.all,R.percentage,pcnt,R.max,R.min,R.mean,R.size(1),R.size(2),R.si
ze(3),R.grid(1),R.grid(2),R.grid(3)));
```

```
try
   fid = fopen(rptfile,'w');
   fprintf(fid,'Patient= %s - %s \r\n----------\r\n%s Max Dose (Gy)=%3.1f \r\n%s Max Dose
(Gy)=%3.1f \r\n\r\nGamma: %2.1f%s - %2.1fmm (cutoff %3.1f %s) \r\n----------
\r\nPass=%u\r\nTotal=%u\r\nGamma Percentage= %3.1f %s\r\nMax= %3.1f \r\nMin= %3.1f
\r\nMean = %4.2f \r\nSize= %u X %u X %u\r\nVoxel Size = %3.1f X %3.1f X %3.1f\r\n
',R.MRN,R.LastName,R.gold.Name,R.gold.max,R.sample.Name,R.sample.max,R.deltaDose*100,pcnt
,R.deltaDist,R.cutoff,pcnt,R.pass,R.all,R.percentage,pcnt,R.max,R.min,R.mean,R.size(1),R.size(2),R.si
ze(3),R.grid(1),R.grid(2),R.grid(3));
   %
fprintf(fid,'%2.2f,%2.1f,%1.2f,%u,%u,%3.1f,%u,%u,%u,%1.2f,%1.2f,%1.2f,%f,%f,%f',R.deltaDose,R.d
eltaDist,R.cutoff,R.pass,R.all,R.percentage,R.size(1),R.size(2),R.size(3),R.grid(1),R.grid(2),R.grid(3),R.
orig(1),R.orig(2),R.orig(3));
   fprintf(fid,'%f,%f,%f,%u,%u,%u,%f,%f,%f,%u,%u,%u,%3.1f,-
',R.grid(1),R.grid(2),R.grid(3),R.size(1),R.size(2),R.size(3),R.orig(1),R.orig(2),R.orig(3),R.deltaDose*10
0,R.deltaDist*10,R.cutoff*100,R.percentage);
   fclose(fid);
catch exception
   display(sprintf('report file not saved\n'));
end




end
```

9 Appendix D: DVH Data Export software

GetDVH.Script

```
// ScriptDir
Store.StringAt.ScriptDir = "/home/p3rtp/ohrt/DVHgetter/";

// SaveDir
Store.StringAt.SaveDir = "/home/p3rtp/ohrt/DVHgetter/datatmp/";

// InfoLoop
Store.StringAt.InfoLoop = Store.StringAt.ScriptDir;
Store.At.InfoLoop.AppendString = "loop_DVHdata.Script";

//Set 10cGy bin size for all DVHS
DVHList .Current .AutoComputeBinSize = 0;
DVHList .Current .BinSize = " 10";
DVHList .#"*" .AutoComputeBinSize = DVHList .Current .AutoComputeBinSize;
DVHList .#"*" .BinSize = DVHList .Current .BinSize;
DVHList .#"*" .NumberOfBins = DVHList .Current .NumberOfBins;
DVHList .#"*" .ColumnsPerRow = DVHList .Current .ColumnsPerRow;



//Save info files
DVHList.ChildrenEachCurrent.#"@".Script.ExecuteNow = Store.StringAt.InfoLoop;

Store.StringAt.calc = "python ";
Store.At.calc.AppendString = Store.StringAt.ScriptDir;
Store.At.calc.AppendString = "DVHsave.py ";
Store.At.calc.AppendString = """;
Store.At.calc.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.calc.AppendString = "' '";
Store.At.calc.AppendString = PlanInfo.LastName;
Store.At.calc.AppendString = "' '";
Store.At.calc.AppendString = PlanInfo.MiddleName;
Store.At.calc.AppendString = """;
//WarningMessage = Store.StringAt.calc;
SpawnCommand = Store.StringAt.calc;
QuitWithSave = "";
```

loopDVHdata.Script

```
// create DVH Data filename
Store.StringAt.DataFile = Store.StringAt.SaveDir;
Store.At.DataFile.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.DataFile.AppendString = "_dvhData_";
Store.At.DataFile.AppendString = DVHList.CurrentIndex;
Store.At.DataFile.AppendString = ".txt";


// Save DVH Index, ROI Name, and TrialName
Store.StringAt.Labelcmd = "python /home/p3rtp/ohrt/DVHgetter/LabelSaver.py ";
Store.At.Labelcmd.AppendString = """;
Store.At.Labelcmd.AppendString = PlanInfo.MedicalRecordNumber;
Store.At.Labelcmd.AppendString = "' '";
Store.At.Labelcmd.AppendString = DVHList.Current.Index;
Store.At.Labelcmd.AppendString = "' '";
Store.At.Labelcmd.AppendString = DVHList.Current.RegionOfInterestName;
Store.At.Labelcmd.AppendString = "' '";
Store.At.Labelcmd.AppendString = DVHList.Current.Trial.Name;
Store.At.Labelcmd.AppendString = """;
//WarningMessage = Store.StringAt.Labelcmd;
SpawnCommand = Store.StringAt.Labelcmd;

//Store.StringAt.Label.Save = Store.StringAt.LabelFile


// save DVH data
DVHList.Current.Data.Save = Store.StringAt.DataFile;
```

LabelSaver.py

```
import os
import sys
sys.path.append('/home/p3rtp/ohrt')

from dictImport import DVHdir

mrn = sys.argv[1]
trialindex = sys.argv[2]
roiname = sys.argv[3]
trialname = sys.argv[4]

filename = os.path.join(DVHdir,str(mrn + '_dvhLabel_' + trialindex + '.txt'))

f = open(filename, 'w')

f.write(trialindex.replace(' ','') + ',')
f.write(roiname + ',')
f.write(trialname)

f.close
```

DVHSave.py

```
import os
import glob
import math
import sys
sys.path.append('/home/p3rtp/ohrt')
from Tkinter import *
from dictImport import DVHdir
from dictImport import storedir
from dictImport import dvhconfile

sitedict = { 'thor':'LUNG','gu':'GU','hn':'HN','gyn':'GYN'} # used when gettind site from middle name

rmslabels = { 1:'clin-rv',2:'clin-tx',3:'clin-eclp'}

mrn = sys.argv[1]
lastname = str(sys.argv[2]).replace('restored','')
lastname = lastname.replace(' ','')
lastname = lastname[0].upper() + lastname[1:].lower()

site = sitedict[filter(lambda x: x.isalpha(), sys.argv[3]).lower()]
print 'site: ',site


def go():
   ######## Pinnacle saves DVH data and Information separately. This part combines them into one
file.
   # One file is created for each trial-DVH pair.
   Labels = glob.glob(os.path.join(DVHdir,str(mrn + '_dvhLabel_*.txt')))

   ###### This was written specifically for plans with 4 specific trials. Here
   ###### the lists are created that will be used to report the results.
   clinical = range(len(Labels))
   rv = range(len(Labels))
   tx = range(len(Labels))
   eclp = range(len(Labels))

   dvhlist = []
   trialist = ['Clinical','Mosaiq','TxRecords','Eclipse']
   roilist = []
   ######


   #### Open the pairs of data and label files and combine them
   for t in range(len(Labels)):
      labfile = Labels[t]
      datfile = Labels[t].replace('Label','Data')
```

```
f = open(labfile, 'r')
ID = f.readlines()
f.close

ID = ID[0].split(',')

dvhindex = int(ID[0])
roi = ID[1]

#### create list of rois
if roi not in roilist:
   roilist.append(roi)
trial = ID[2]
#if trial not in triallist:
#    triallist.append(trial)
if (trial.find('Plan_Import') != -1): trial = 'Mosaiq'
elif (trial.find('Dose_Import') != -1): trial= 'Eclipse'
elif (trial.find('Tx Update') != -1): trial = 'TxRecords'
else: trial = 'Clinical'


#### open and read new file
f = open(datfile, 'r')
DAT = f.readlines()
f.close
del DAT[-3:]

#### check to make sure all of the crap is trimmed from the end of DVH info
# and that the last endline is preceeded by a comma
datchk=1
while datchk:
   num = filter(lambda x: x.isdigit(),DAT[-1])
   if len(num) == 0:
      del DAT[-1]
   else:
      if len(DAT[-1].split(',')) == 2:
         DAT[-1]=DAT[-1].replace('\n',',\n')
      datchk = 0

#### write new dvh file that has pt,trial name, and roi name in header
newfilename = mrn + '_' + lastname + '_' + roi.replace( ' ','') + '_' + trial.replace(' ','') + '.dvh'
newfile = os.path.join(DVHdir,newfilename)

out=open(newfile,'w')
out.write(mrn + ' - ' + lastname + '\n')
out.write(roi + '\n')
out.write(trial + '\n')
```

```
        out.writelines(DAT)
        out.close()

        ## add new file to dvh list and delete original data and label files.
        dvhlist.append(newfile)
        os.remove(labfile)
        os.remove(datfile)


    site = Report(dvhlist,trialist, roilist)
    print '\n SITE: ',site,' - DVHs exported for: ', lastname,'-',mrn,'\n'

def Report(filelist,trialist, roilist):
    ########### generate ASCII file containing results for this group of DVH

    #site = AskSite() ###site determined by middle name
    roicons = getConstraints(site)

    results = range(len(roilist))
    for r in range(len(results)):
        results[r] = [roilist[r]]
    ##results[r].append(range(len(trialist)+1))

    ###########open each DVH file
    for file_i in filelist:
        f=open(file_i,'r')
        info = f.readlines()
        f.close()

        ####get roi name and trial info from header and then delete header
        roi = info[1][:-1]
        roiindex = roilist.index(roi)
        roi = roi.lower()
        trial = info[2][:-1]
        trlindex = trialist.index(trial)
        del info[0:6]

        #### split dvh data on commas
        for pnt in range(len(info)-1,-1,-1):
            #print 'pnt=',pnt,'\t',info[pnt]
            info[pnt] = info[pnt].split(',')
            info[pnt][0] = float(info[pnt][0])
            info[pnt][1] = float(info[pnt][1])

            #### add a column to data that is cumulative DVH
            #try:
            #print 'file=',file_i,'\npnt=',pnt,'\t'
            if pnt != len(info)-1:
```

```
      info[pnt][2] = info[pnt+1][2] + info[pnt][1]
   else:
      info[pnt][2] = float(info[pnt][1])

   info[pnt][2] = float(info[pnt][2])
   #except:
      #print 'file=',file_i,'\npnt=',pnt,'\t'
      #for h in enumerate(info[pnt]): print h
      #q=5
      #print 'BONK'


#### loop through DVH contraint masks to see if one matches roi name
#    if so then calculates consstraint values
identified =0
for r in roicons:
   for s in r[:-1]:
      if roi.lower().find(s) != -1:
         #### match found
         #### get default calculation below.... on 2nd thought, don't
         #type, ans = calc(info,'maxdose',0)
         #results = BuildResults(results,roiindex,r,trialist,trlindex,'maxdose','-',ans)
         #
         #type, ans = calc(info,'meandose',0)
         #results = BuildResults(results,roiindex,r,trialist,trlindex,'meandose','-',ans)
         #
         #### get roi specific calculations
         identified =1
         for t in range(len(r[-1])):
            type = r[-1][t][0]
            level = r[-1][t][1]
            limit = r[-1][t][2]
            type, ans = calc(info,type,level)

            results = BuildResults(results,roiindex,r,trialist,trlindex,type,limit,ans)
         break
   if identified:
      ## roi has been identified so break the loop
      break

if identified != 1:
   ##roi has not been identified so run default calculations below
   type, ans = calc(info,'maxdose',0)
   results = BuildResults(results,roiindex,'',trialist,trlindex,'maxdose','-',ans)

   type, ans = calc(info,'meandose',0)
   results = BuildResults(results,roiindex,'',trialist,trlindex,'meandose','-',ans)
```

```
    ###### write the report file
    crap = ['[',']','\'']
    report = os.path.join(DVHdir,str('DVH_' + mrn + '_' + lastname + '.rpt'))

    rpt = open(report,'w')
    rpt.write( mrn + ',' + lastname + ',Site: ' + site + '\n')
    rpt.write(',,,ROI,Class,Constraint,Limit,' + filter(lambda x: x not in crap,str(trialist)) + '\n')
    rpt.write(mrn + ',' + lastname + '\n')
    for roi in results:
        rpt.write(',,,' + filter(lambda x: x not in crap,str(roi[:3])) + '\n')
        for cons in range(3,len(roi)):
            rpt.write(',,,,,' + filter(lambda x: x not in crap,str(roi[cons])) + '\n')

    rpt.close()
    #plan title might be good here, so would a cold beer
    return site

def BuildResults(results,roiindex,r,trialist,trlindex,type,limit,ans):
    #### see if this is the 1st calulation for this roi (then add to list),
    # or if it has already been calulated for another trial (append to list)
    found = 0
    for cons in range(2,len(results[roiindex])):
        if (results[roiindex][cons][0].find(type)!= -1):
            ## roi has been seen before
            results[roiindex][cons][trlindex+2] = ans
            found = 1;
            break
    ### 1st time roi has been seen
    if not found:
        if len(results[roiindex]) < 2:
            results[roiindex].append(' '.join(r[:-1]))  #name of constraint
        results[roiindex].append(range(len(trialist)+2))
        results[roiindex][-1][0] = type           #type of constraint
        results[roiindex][-1][1] = limit          #constraint limit
        results[roiindex][-1][trlindex+2] = ans     #value

    return results

def calc(data,type,level):
    if type == 'maxdose':
        for i in range(len(data)-1,-1,-1):
            if data[i][1] != 0:
                ans = data[i][0]
                break
    elif type == 'meandose':
        cGy_cc = 0
        for i in range(len(data)):
```

```
            cGy_cc += data[i][0]*data[i][1]
        cc = data[0][2]
        ans = cGy_cc/cc
    elif type == 'vol':

        type = 'V' + str(level/100)
        if data[-1][0] < level:
            ans = 0
        else:
            for i in range(len(data)):
                if data[i][0] == level:
                    ans = 100*data[i][2]/data[0][2]
                    break

    elif type == 'cover':
        type = 'cover' + str(level)
        volcon = level*data[0][2]
        #print 'volcon = ',volcon,'\ttot = ',data[0][2]
        tmp = []
        for i in range(len(data)):
            tmp.append(math.fabs(data[i][2]- volcon))


        min_diff = min(tmp)
        for ln in range(len(tmp)-1,-1,-1):
            if tmp[ln] == min_diff:
                min_index = ln     # have to assign min index this way bc, tmp.index(min)won;t work if two
dose levels have same value
                break

        ans = data[min_index][0]
        #print 'diff =',min_diff,';  index =',min_index,'  dose =',ans

    else:
        print 'INVALID TYPE -> ',type
        ans = '-'
    #try:
    #    print ans
    #except:
    #    print 'type/level = ',type, level

    return type,ans

def AskSite(MRN=mrn, LastName=lastname):
    #### read the DVH constraints file and ask the user to select the treatment site
    root=Tk()
    root.title('Select Treatment Site: '+str(MRN)+LastName)
```

```
    F1 = Frame()
    #Label(F1,text='Standard').grid(row=0,column=0)
    Label(F1,text='Standard').grid(row=0,column=0)

    sitelist = getSites()

    std = IntVar()
    for i in range(len(sitelist)):
      Radiobutton(F1,text='',variable=std,value=i).grid(row=i+1,column=0)
      Label(F1,text = sitelist[i]).grid(row=i+1,column=1, sticky=W)
      g=i+2
    Button(F1, text='OK', command=root.destroy).grid(row=g+1,column=1)

    F1.pack(side=LEFT, fill=X)
    mainloop()

    return sitelist[std.get()]

def getSites():

    clutter = ['[',']','\'','\n']
    f=open(dvhconfile,'r')
    data = f.readlines()
    f.close

    sights = []
    line = 0
    while line < len(data):
       if data[line].find('site: ') != -1:
          filter(lambda x: x not in clutter, str(sights.append(data[line].split(':')[-1][1:-1])))
          line += 1
       else:
          line += 1
    #print sights
    return sights

def getConstraints(site):
    f=open(dvhconfile,'r')
    data = f.readlines()
    f.close
    line = 0
    #roicons = [['PTV', 'ptv',[['cover',0.950,'-']]],['CTV', 'ctv',[['cover',0.990,'-']]],['GTV',
'gtv',[['cover',1.000,'-']]]]
    roicons = [['ptv',[['cover',0.950,'-']]],['ctv',[['cover',0.990,'-']]],['gtv',[['cover',1.000,'-
']]],['itv',[['cover',0.990,'-']]]]
    while line < len(data):
       if data[line].find('site:') != -1:
          if data[line].find(site) != -1:
```

```
line += 1

while line < len(data):
   if data[line].find('roi:') != -1:
      roi = data[line][:-1].split(': ')[-1] #split on colon-space
      roi = roi.split(' ') #SPLIT roi name into separate words
      #if (roi[0] == ''):
      #    del roi[0]
      #    if (roi[0] == ''):
      #        del roi[0]
      line += 1

      criteria = []
      while line < len(data):
         if data[line].find('maxdose') != -1:
            level = int(data[line].split(':')[-1].split(',')[0])
            limit = level
            criteria.append(['maxdose',level,limit])

         elif data[line].find('meandose') != -1:
            level = int(data[line].split(':')[-1].split(',')[0])
            limit = level
            criteria.append(['meandose',level,limit])

         elif data[line].find('volume') != -1:
            level = 100*int(data[line].split('<')[0].replace('volume',''))
            limit = int(100*float(data[line].split('< ')[1].split('[')[0]))
            criteria.append(['vol',level,limit])

         elif data[line].find('plan:') != -1:
            break

         elif data[line].find('site:') != -1:
            break

         elif data[line].find('roi:') != -1:
            break

         line += 1

      roi.append(criteria)
      roicons.append(roi)

      if data[line].find('site:') != -1:
         line = len(data)
         break
      elif data[line].find('plan:') != -1:
         break
```

```
                else:
                    line += 1
            else:
                line += 1
        else:
            line += 1

    #for i,v in enumerate(roicons):print i,v
    #print site,' ---> ',roicons

    return roicons


def ReportOnlyPrep():
    filelist = glob.glob(os.path.join(DVHdir,str('*' + lastname + '*.dvh')))
    trialist = ['Clinical','Mosaiq','TxRecords','Eclipse']
    roilist = []
    for f in filelist:
        roiname = f.split('_')[2]

        if roiname not in roilist:
            roilist.append(roiname)
    print roilist
    Report(filelist,trialist,roilist)


go()
#ReportOnlyPrep()
```

10 References

1. Fletcher, Gilbert H., *Textbook of radiotherapy*, 3d ed. Lea & Febiger, Philadelphia, 1980.
2. *Pinnacle^3 Physics Guide*. ADAC Laboratories, P/N 9201-2050A Rev. C (Ver. 5.2), 2000.
3. Johns, H. E. and Cunningham, J. R., *The Physics of Radiology*, 4th ed. Charles C. Thomas, Springfield, Ill., U.S.A., 1983.
4. Khan, F.M., *The Physics of Radiation Therapy*, 4th ed. Lippincott Williams & Wilkins, Philadelphia, PA, 2010.
5. Kijewski, P. K. and Bjarngard, B. E., *The use of computed tomography data for radiotherapy dose calculations*, Int J Radiat Oncol Biol Phys 4 (5-6), 429-435 (1978).
6. Van Dyk, J., Barnett, R. B., Cygler, J. E. and Shragge, P. C., *Commissioning and quality assurance of treatment planning computers*, Int J Radiat Oncol Biol Phys 26 (2), 261-273 (1993).
7. Van Houtte, Paul, Piron, Auguste, Lustman-Maréchal, Jacqueline, Osteaux, Michel and Henry, Jacques, *Computed axial tomography (CAT) contribution for dosimetry and treatment evaluation in lung cancer*, International Journal of Radiation Oncology*Biology*Physics 6 (8), 995-1000 (1980).
8. International Commission on Radiation Units and Measurements., *Prescribing, recording, and reporting photon beam therapy ICRU report 50*. International Commission on Radiation Units and Measurements, Bethesda, MD, 1993.
9. International Commission on Radiation Units and Measurements., *Prescribing, recording, and reporting photon beam therapy ICRU report 62*. International Commission on Radiation Units and Measurements, Bethesda, Md., 1999.

10. Bushberg, Jerrold T., *The essential physics of medical imaging*, 3rd ed. Wolters Kluwer Health/Lippincott Williams & Wilkins, Philadelphia,

11. Papanikolaou, A., Battista, A., Boyer, A., Kappas, C., Klein, E., Mackie, T.R., Sharpe, M. and Van Dyk, J., *Tissue Inhomogeneity Corrections for Megavoltage Photon Beams*, Report No 85, Task Group No 65 of the Radiation Therapy Committee of the American Association of Physicists in Medicine, Medical Physics Publishing, Madison, WI 2004

12. Sharpe, M., *Commissioning and Quality Assurance for IMRT Treatment Planning* in Intensity-Modulated Radiation Therapy :The State of the Art, edited by J. R. Palta and T. R. Mackie Medical Physics Pub., Madison, WI, 2003, pp. 449-473.

13. Andreo, P., *Monte Carlo techniques in medical radiation physics*, Phys Med Biol 36 (7), 861-920 (1991).

14. Turner, J. E., Wright, H. A. and Hamm, R. N., *A Monte Carlo primer for health physicists*, Health Phys 48 (6), 717-733 (1985).

15. Raeside, D. E., *Monte Carlo principles and applications*, Phys Med Biol 21 (2), 181-197 (1976).

16. Mohan, R., Chui, C. and Lidofsky, L., *Energy and angular distributions of photons from medical linear accelerators*, Med Phys 12 (5), 592-597 (1985).

17. Lovelock, D. M., Chui, C. S. and Mohan, R., *A Monte Carlo model of photon beams used in radiation therapy*, Med Phys 22 (9), 1387-1394 (1995).

18. Liu, H. H., Mackie, T. R. and McCullough, E. C., *A dual source photon beam model used in convolution/superposition dose calculations for clinical megavoltage x-ray beams*, Med Phys 24 (12), 1960-1974 (1997).

19. DeMarco, J. J., Solberg, T. D. and Smathers, J. B., *A CT-based Monte Carlo simulation tool for dosimetry planning and analysis*, Med Phys 25 (1), 1-11 (1998).

20. Ulmer, W., Pyyry, J. and Kaissl, W., *A 3D photon superposition/convolution algorithm and its foundation on results of Monte Carlo calculations*, Phys Med Biol 50 (8), 1767-1790 (2005).

21. Sievinen, J., Ulmer, W. and Kaissi, W., *AAA Photon Dose Calculation Model in Eclipse™*. Varian Medical Systems, RAD#7170B, 2005.

22. Ahnesjo, A., *Collapsed cone convolution of radiant energy for photon dose calculation in heterogeneous media*, Med Phys 16 (4), 577-592 (1989).

23. Ahnesjo, A., Andreo, P. and Brahme, A., *Calculation and application of point spread functions for treatment planning with high energy photon beams*, Acta Oncol 26 (1), 49-56 (1987).

24. Mohan, R., Chui, C. and Lidofsky, L., *Differential pencil beam dose computation model for photons*, Med Phys 13 (1), 64-73 (1986).

25. Mackie, T. R., Scrimger, J. W. and Battista, J. J., *A convolution method of calculating dose for 15-MV x rays*, Med Phys 12 (2), 188-196 (1985).

26. Fogliata, A., Vanetti, E., Albers, D., Brink, C., Clivio, A., Knoos, T., Nicolini, G. and Cozzi, L., *On the dosimetric behaviour of photon dose calculation algorithms in the presence of simple geometric heterogeneities: comparison with Monte Carlo calculations*, Phys Med Biol 52 (5), 1363-1385 (2007).

27. McNutt, T. R., Mackie, T. R., Reckwerdt, P., Papanikolaou, N. and Paliwal, B. R., *Calculation of portal dose using the convolution/superposition method*, Med Phys 23 (4), 527-535 (1996).

28. Papanikolaou, N., Mackie, T. R., Meger-Wells, C., Gehring, M. and Reckwerdt, P., *Investigation of the convolution method for polyenergetic spectra*, Med Phys 20 (5), 1327-1336 (1993).

29. Van Esch, A., Tillikainen, L., Pyykkonen, J., Tenhunen, M., Helminen, H., Siljamaki, S., Alakuijala, J., Paiusco, M., Lori, M. and Huyskens, D. P., *Testing of the analytical anisotropic algorithm for photon dose calculation*, Med Phys 33 (11), 4130-4148 (2006).

30. Oh, C. E., Antes, K., Darby, M., Song, S. and Starkschall, G., *Comparison of 2D conventional, 3D conformal, and intensity-modulated treatment planning techniques for patients with prostate cancer with regard to target-dose homogeneity and dose to critical, uninvolved structures*, Med Dosim 24 (4), 255-263 (1999).

31. *P^3IMRT Instructions For Use Release 9*. Philips Medical Systems, 4535 604 45421 A 2009.

32. Inter-Society Council for Radiation Oncology, *Radiation Oncology in Integrated Cancer Management: Report of the Inter-Society Council for Radiation Oncology*. American College of Radiology, Reston, VA, 1991.

33. Mutic, S., Palta, J. R., Butker, E. K., Das, I. J., Huq, M. S., Loo, L. N., Salter, B. J., McCollough, C. H. and Van Dyk, J., *Quality assurance for computed-tomography simulators and the computed-tomography-simulation process: report of the AAPM Radiation Therapy Committee Task Group No. 66*, Med Phys 30 (10), 2762-2792 (2003).

34. McCollough, C, Cody, D, Edyvean, S, Geise, R, Gould, B, Keat, N, Huda, W, Judy, P, Kalender, W, McNitt-Gray, M, Morin, R, Payne, T, Stern, S and Rothenburg, L, *AAPM report No. 96, The Measurement, Reporting, and Management of Radiation Dose in CT*. AAPM, College Park, MD, 2008.

35. Texas Administrative Code 25, *Texas Regulations for Control of Radiation §289.227 Use of Radiation Machines in the Healing Arts and Veterinary Medicine*, 2004

36. Ezzell, G. A., Burmeister, J. W., Dogan, N., LoSasso, T. J., Mechalakos, J. G., Mihailidis, D., Molineu, A., Palta, J. R., Ramsey, C. R., Salter, B. J., Shi, J., Xia, P., Yue, N. J. and Xiao, Y.,

IMRT commissioning: multiple institution planning and dosimetry comparisons, a report from AAPM Task Group 119, Med Phys 36 (11), 5359-5373 (2009).

37. Almond, P. R., Biggs, P. J., Coursey, B. M., Hanson, W. F., Huq, M. S., Nath, R. and Rogers, D. W., *AAPM's TG-51 protocol for clinical reference dosimetry of high-energy photon and electron beams*, Med Phys 26 (9), 1847-1870 (1999).

38. Klein, E. E., Hanley, J., Bayouth, J., Yin, F. F., Simon, W., Dresser, S., Serago, C., Aguirre, F., Ma, L., Arjomandy, B., Liu, C., Sandin, C. and Holmes, T., *Task Group 142 report: quality assurance of medical accelerators*, Med Phys 36 (9), 4197-4212 (2009).

39. Boyer, A., Biggs, P., Galvin, J., Klein, E., LaSasso, T., Low, D., Mah, K. and Yu, C., *Basic Applications of Multileaf Collimators: Report of Task Group No. 50 Radiation Therapy Committee*, American Association of Physicists in Medicine Report No. 72 ed. Medical Physics Publishing, Madison, WI, 2001.

40. Ezzell, G., *Quality Assurance: When And What Is Enough For IMRT?* in Intensity-Modulated Radiation Therapy :The State of the Art:, edited by J. R. Palta and T. R. Mackie Medical Physics Pub., Madison, WI, 2003, pp. 613-616.

41. Xia, P. and Chuang, C., *Patient-Specific Quality Assurance in IMRT* in Intensity-Modulated Radiation Therapy :The State of the Art, edited by J. R. Palta and T. R. Mackie Medical Physics Pub., Madison, WI, 2003, pp. 495-514.

42. Kutcher, G. J., Coia, L., Gillin, M., Hanson, W. F., Leibel, S., Morton, R. J., Palta, J. R., Purdy, J. A., Reinstein, L. E., Svensson, G. K. and et al., *Comprehensive QA for radiation oncology: report of AAPM Radiation Therapy Committee Task Group 40*, Med Phys 21 (4), 581-618 (1994).

43. Stern, R. L., Heaton, R., Fraser, M. W., Goddu, S. M., Kirby, T. H., Lam, K. L., Molineu, A. and Zhu, T. C., *Verification of monitor unit calculations for non-IMRT clinical radiotherapy: report of AAPM Task Group 114*, Med Phys 38 (1), 504-530.

44. American College of Radiology, ACR TECHNICAL STANDARD FOR THE PERFORMANCE OF RADIATION ONCOLOGY PHYSICS FOR EXTERNAL BEAM THERAPY, http://www.acr.org/SecondaryMainMenuCategories/quality_safety/guidelines/med_phys/physics_external_beam_therapy.aspx, Accessed: August 1,2011

45. Bogdanich, W., "Radiation Offers New Cures, and Ways to Do Harm", *The New York Times*,New York Edition, Jan 24, 2010, A1

46. Wu, Q., Manning, M., Schmidt-Ullrich, R. and Mohan, R., *The potential for sparing of parotids and escalation of biologically effective dose with intensity-modulated radiation treatments of head and neck cancers: a treatment design study*, Int J Radiat Oncol Biol Phys 46 (1), 195-205 (2000).

47. Grills, I. S., Yan, D., Martinez, A. A., Vicini, F. A., Wong, J. W. and Kestin, L. L., *Potential for reduced toxicity and dose escalation in the treatment of inoperable non-small-cell lung cancer: a comparison of intensity-modulated radiation therapy (IMRT), 3D conformal radiation, and elective nodal irradiation*, Int J Radiat Oncol Biol Phys 57 (3), 875-890 (2003).

48. Pollack, A., Hanlon, A., Horwitz, E. M., Feigenberg, S., Uzzo, R. G. and Price, R. A., *Radiation therapy dose escalation for prostate cancer: a rationale for IMRT*, World J Urol 21 (4), 200-208 (2003).

49. Bakai, A., Alber, M. and Nusslin, F., *A revision of the gamma-evaluation concept for the comparison of dose distributions*, Phys Med Biol 48 (21), 3543-3553 (2003).

50. Dong, L., Antolak, J., Salehpour, M., Forster, K., O'Neill, L., Kendall, R. and Rosen, I., *Patient-specific point dose measurement for IMRT monitor unit verification*, Int J Radiat Oncol Biol Phys 56 (3), 867-877 (2003).

51. Anjum, M. N., Parker, W., Ruo, R., Aldahlawi, I. and Afzal, M., *IMRT quality assurance using a second treatment planning system*, Med Dosim 35 (4), 274-279.

52. Luo, W., Li, J., Price, R. A., Jr., Chen, L., Yang, J., Fan, J., Chen, Z., McNeeley, S., Xu, X. and Ma, C. M., *Monte Carlo based IMRT dose verification using MLC log files and R/V outputs*, Med Phys 33 (7), 2557-2564 (2006).

53. Teke, T., Bergman, A. M., Kwa, W., Gill, B., Duzenli, C. and Popescu, I. A., *Monte Carlo based, patient-specific RapidArc QA using Linac log files*, Med Phys 37 (1), 116-123.

54. Varian Medical Systems *Millennium MLC System and Maintenance Guide*. Varian Medical Systems P/N 100027395-04, 2011.

55. Varian Medical Systems *Dynalog File Viewer Reference Guide*. Varian Medical Systems P/N 100013698-04, 2007.

56. Harms Sr, William B., Low, Daniel A., Wong, John W. and Purdy, James A., *A software tool for the quantitative evaluation of 3D dose calculation algorithms*, Medical Physics 25 (10), 1830-1836 (1998).

57. Low, D. A., Harms, W. B., Mutic, S. and Purdy, J. A., *A technique for the quantitative evaluation of dose distributions*, Med Phys 25 (5), 656-661 (1998).

11 Vita

Jared Dean Ohrt was born in Victoria, TX on July 28th, 1981 to Patricia and Dean Ohrt. After graduation from Stroman High School, Victoria, TX in 1999, he enrolled at the University of Texas at Austin where he received a Bachelor of Science degree in physics in May 2004. On Valentine's Day 2005 he began working as a QA Dosimetrist/Physics Assistant at U.T. M.D. Anderson Cancer Center. In 2008, he began taking medical physics courses at The University of Texas Health Science Center at Houston Graduate School of Biomedical Sciences as a Texas Medical Center employee. In a four week period starting in August 2010 he left his job at U.T. M.D. Anderson, married the most beautiful women he'd ever seen, went on a 2 week honeymoon in Greece, and enrolled as a full-time student in the Specialized Masters of Medical Physics program at The University of Texas Health Science Center at Houston Graduate School of Biomedical Sciences.

Permanent Address:
902 Norfolk Dr.
Pearland, TX 77584