University of Alberta

Evaluation and Modification of a Differential SAR Dose Calculation Algorithm

by

John Richard Kollar

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

in

Medical Physics

Department of Physics

Edmonton, Alberta

Spring 1996

ISBN 0-612-10723-X

University of Alberta

Library Release Form

Name of Author: John Richard Kollar

Title of Thesis: Evaluation and Modification of a Differential SAR Dose Calculation Algorithm

Degree: Master of Science

Year this Degree Granted: 1996

John R. Kollar

7403 Fountain Road, SE
Calgary, Alberta, T2H-0W9

February 8, 1996

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Evaluation and Modification of a Differential SAR Dose Calculation Algorithm submitted by John Richard Kollar in partial fulfillment of the requirements for the degree of Master of Science in Medical Physics.

Brendan McClean

Ron Sloboda

Nathan Rodning

Stuart Jackson

Feb. 6, 1996
Date of Approval

DEDICATION

I would like to dedicate this thesis to my father, Anton Kollar, who passed away on July 25, 1995. Without doubt, his pride in my academic achievements was one of the two most important motivations behind my pursuits in the sciences, the other simply being my interest in these areas. I deeply regret that he will not be around to see the completion of this project and thesis.

ABSTRACT

"George's RAdiotherapy Treatment desIgn System" (GRATIS), a treatment planning system designed at the University of North Carolina by George Sherouse and his colleagues, offers a flexible and fast algorithm for the calculation of dose in 3D conformal radiotherapy. It's calculation routine is a generalization of the CBEAM algorithm created by John Cunningham, which calculates the dose value at a point in phantom by separately evaluating the components of dose that are due to primary and scattered radiation. The calculation algorithm, although fairly general in design, currently makes a number of simplifying assumptions to reduce calculation time, and the corresponding tradeoff is a potential loss of accuracy.

We have attempted to remove these simplifications and test the limits of accuracy of the algorithm for symmetric and asymmetric collimator configurations. Isodoses have been obtained on GRATIS, on a Varian 2300CD Linac, and on the "Alberta Treatment Planning system" (ATP) so that the performance of GRATIS can be evaluated. A modification to the algorithm has been investigated that involves a redefinition of the "pencils" used in the "scatter integration". This modification makes the integration more consistent with the measured data which is used by the algorithm and increases its accuracy. Limitations of this type of calculation have also been demonstrated for off axis cases where the energy spectrum of the accelerator is significantly different from the central axis spectrum.

ACKNOWLEDGMENTS

During the three years that it took to carry out this project I have met and become friends with a great number of people (too many to list here), and have many more friends back in my home city of Calgary and elsewhere. All have been very supportive of my work and I would like to thank them for making my stay in Edmonton an enjoyable one. My family has also shown great support, encouragement, patience and pride during my graduate education, as well as financial support to help alleviate the Spartan life that graduate students often endure. My sincerest thanks to them. Also, my thanks go to my committee members and the members of the Department of Medical Physics at the Cross Cancer Institute for the wealth of knowledge that they provided so that this thesis could be possible.

I would like to give special thanks to my supervisor Dr. Brendan McClean, who showed confidence in me when I presented my work at two conferences (very stressful experiences), provided endless insights into problems that would otherwise have stumped me for unacceptably long stretches of time, and mostly for showing extreme patience during the three years I have been his graduate student.

And finally I would like to give thanks to my committee members and anyone else who has endured reading this thesis. I hope they had a large pot of coffee at the time.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

During the past few decades radiotherapy treatment of cancer has progressed rapidly with the introduction of new technologies. New methods have been developed for the diagnosis of cancer, the production of ionizing radiation, and the control of radiation exposure. External beam radiotherapy, for instance, is a method that uses ionizing radiation from a source outside the patient to destroy cancer cells. An accelerator (figure 1.1) [Karzmark 1984] is often used to produce a beam of high energy electrons or photons that is directed through the patient to the site of the cancerous tissue. The strategy is to direct several such beams of high energy radiation through the tumor from various directions, causing dose to accumulate at the intersection of all beams (i.e. at the tumor). The intensity of each component beam must be kept as low as possible to minimize the dose deposited in tissue other than the tumor, but must also be high enough that the cumulative dose at the intersection volume is enough to destroy the tumor cells. This strategy requires that a number of problems be tackled, three of which are listed below.

- The optimal number and directions of beams must be chosen, taking into consideration the distribution of dose that will result, the vital organs that may be affected, and that more complicated beam arrangements will increase the difficulty in actually setting up and implementing the treatment.
- The cross-sectional shape of each beam must be tailored to conform to the tumor shape as much as possible, again taking into consideration the vital organs and the dose distribution.
- The dose distribution itself depends upon the materials through which the radiation passes and off which it scatters (i.e. soft tissue, bone, air, metal filters, etc.).

These three problems already entail the analysis of a large amount of data, and also the implementation of various optimization strategies. The complexity of the treatment planning problem thus necessitates the use of computers in the analysis and optimization.

The GRATIS[†] computer treatment planning system, for instance,

- allows three dimensional CT-data to be displayed,
- allows beam arrangements to be defined in a computer simulation of the patient and treatment system,
- calculates the distribution of dose based on CT-data and the chosen beam arrangements,
- and allows the resulting dose distribution to be viewed and analyzed.

[†] George's RAdiotherapy Treatment desIgn System.

Using this (or a similar) system, the treatment of a patient under an extensive variety of beam configurations can be simulated to see if the results will be acceptable in terms of sufficiently irradiating the tumor and sparing healthy tissue. Once a treatment plan has produced satisfactory simulation results, it can then be implemented with the real accelerator and patient.

Figure 1.1. A linear accelerator used in external beam radiotherapy.

The complexity of the problem, however, includes a large number of error sources, and accurate prediction of dose is of paramount importance to the treatment planning procedure. Thus, in-depth analyses and extensive testing must be carried out on the calculation routines to ensure that the limits of accuracy are well known.

The work presented in this thesis investigates the limits of accuracy of the dose calculation routines in the GRATIS treatment planning system, and discusses some modifications to these routines that improve their performance. Section 2 begins by discussing three approaches to dose calculation, and then section 3 moves on to discuss the calculation approach used in GRATIS. In section 4 the test cases and modifications implemented in GRATIS are discussed, and section 5 concludes the thesis with a summary of the results and a discussion of the limits of accuracy of the calculation method as implemented in GRATIS.

2. Historical Overview

Various methods of dose calculation have been proposed and implemented in treatment planning systems of the past, all with their own merits and downfalls. The basic variables that distinguish these algorithms are speed, accuracy and generality, which unfortunately cannot all be independently optimized. "Accuracy" refers to the ability of a dose calculation algorithm to reproduce the actual dose that results in a treatment. "Generality" refers to how complex the treatment configurations can get before the reliability of the algorithm is in question. This is related to "accuracy" in that the accuracy of the algorithm usually drops as the treatment configurations become more complex. The "speed" refers to the time it takes to calculate a typical treatment plan. Obviously, one wishes to avoid algorithms that take several hours or days to complete a dose calculation, but reducing the calculation time requires a loss of accuracy or generality. There is an unavoidable tradeoff when optimizing one of these variables over the other two, and this is the fundamental problem of dose algorithm design.

Monte Carlo methods constitute a class of algorithms that optimize accuracy and generality, but sacrifice speed efficiency. Semi-empirical methods constitute a class that attempts to optimize speed and accuracy, with an unfortunate loss of generality. Ultimately there are constraints that can be placed on how much can be sacrificed in each of the three variables, and these constraints should indicate which dose calculation method is preferred. All algorithms, for instance, must require a certain minimum level of accuracy, even if they are not attempting to optimize that variable.

In this chapter some of the tradeoffs of three popular calculation methods are reviewed. The intent is to establish the context for the analysis of the dSAR algorithm in chapter 3, where the limits of the accuracy and generality of this approach are investigated.

2.1. Monte Carlo methods

The most accurate of the dose computation algorithms are the Monte Carlo methods [Andreo 1991, Webb et. al. 1978]. They attempt to model the physics at the particle interaction level; tracking a single source particle as it deposits its energy in a random walk of interactions within the target or patient. (Figure 2.1.)

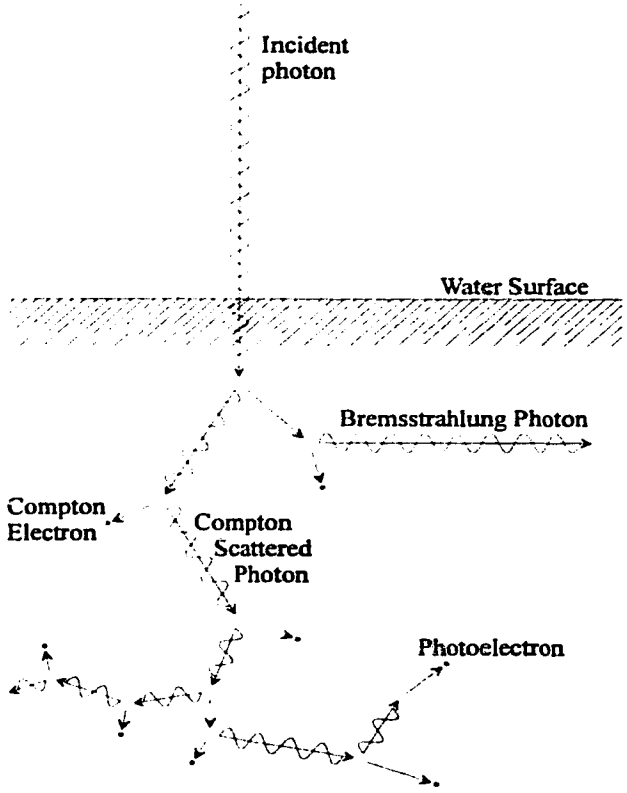


Figure 2.1. Interactions of a photon within a medium.

For high energy X-rays, a shower of a large number of photons is generated using a source model and each photon is processed using a strategy such as figure 2.2. The products it generates are also processed in this way, thus leading to an exponential increase in the number of particles that must be processed. Fortunately the particles must eventually satisfy a termination condition such as the energy dropping below a threshold, which prevents the exponential increase from continuing without bound.

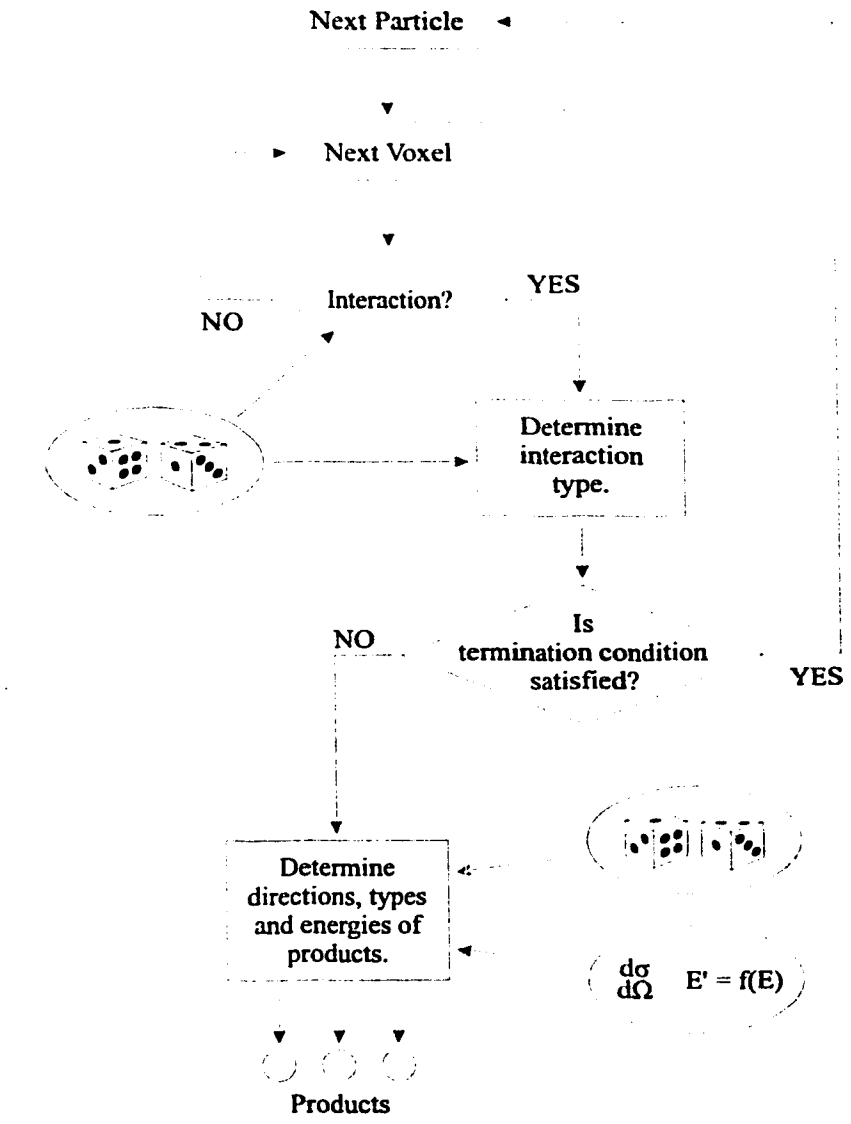


Figure 2.2. Flow diagram of a simple Monte Carlo strategy.

The strategy in figure 2.2 partitions space into voxels and simulates the movement of a particle by jumping along its projected trajectory from one voxel to the next. Within each voxel, a random number generator is used to determine if a reaction takes place, the type of reaction that has occurred (if any), and the properties of all products that are generated. If the interaction type is one that satisfies a termination condition then the kinetic energy of that particle is added to the total energy deposited in the voxel so far. Thus, at the end of the Monte Carlo simulation, each voxel will have a total of the energy that was deposited in it.

The random events that are generated depend upon the properties of the material within the current calculation voxel, and by giving each voxel its own properties (i.e. quantities describing the interaction of radiation with matter) the Monte Carlo method naturally accounts for inhomogeneities in the phantom material. One can thus follow a particle along a random walk through tissue of various densities, as well as through air, metal compensators, scattering off collimators, etc.

The method described above represents only one of many possible Monte Carlo approaches to the simulation of the random walk, and in fact is not very speed efficient for practical purposes. However, it is a useful (albeit simplified) version for the purposes of describing the essential features of the Monte Carlo strategy.
An improved algorithm might, for instance, not generate a random event in every voxel along a path, since much of the path may consist of long straight line segments (i.e. no interactions except at the two endpoints of the segment). We can, instead, randomly generate a length and direction for the straight line segment, and hence determine where the endpoint voxel will be (i.e. where the next interaction will occur) [Nelson 1985].

The Monte Carlo method, since it simulates the physics at the particle interaction level, makes assumptions only at this small scale. That is, assumptions about larger scale concepts such as electron equilibrium need not be made. The method is thus the most generally applicable and accurate one that is available. It suffers, however, from being very calculation intensive. Several hours or even days may be required to adequately simulate a treatment plan, depending upon the complexity of the plan and the accuracy desired. Thus, until affordable computing technology can handle such calculations in a reasonable time (i.e. minutes or even seconds), the Monte Carlo strategy will have to remain a research tool or be used in conjunction with other methods.

2.2. dSAR method

An alternative approach to the simulation of each particle as it interacts with matter is to calculate dose using "bulk" radiation interaction characteristics that have been obtained through measurement [Cunningham 1972, Johns et. al. 1983, Haider et. al. 1995, Khan et. al. 1986, Khan et. al. 1980, Ayyangar et. al. 1993]. That is, the transmission and scattering properties of large volumes (compared to the voxels in the Monte Carlo approach) of phantom material are obtained for a variety of measurement configurations, and these results are used to construct the dose in more complex treatment configurations. An example of such a semi-empirical algorithm is the dSAR (differential scatter-air ratio) method [Cunningham 1972, Johns et. al. 1983].

The dSAR method essentially involves calculating the dose D to an arbitrary point in phantom by an application of a generalized definition of the TAR (Tissue-air ratio). (See equation 2.1 and figure 2.4.) That is, the dose D in phantom is calculated by determining the dose D_{air} to that same point when in air, and multiplying D_{air} by the TAR for the given field shape and calculation point position.

The in-air dose for an arbitrary calculation point is constructed from the measured in-air dose at a calibration point (usually the isocenter of the accelerator). This requires measuring an in-air profile across the largest open field so that the variation of the in-air dose relative to the central axis is known. Also, an inverse square distance correction is required to relate the dose at the source-to-calibration-point distance to the dose at a different source-to-point distance.

$$D_{air}(fs,\bar{x},E) = D_{air}(fs_{max},\bar{c},E_c)\cdot\frac{D_{air}(fs_{max},\bar{x}_c,E)}{D_{air}(fs_{max},\bar{c},E_c)}\cdot\frac{D_{air}(fs_{max},\bar{x},E)}{D_{air}(fs_{max},\bar{x}_c,E)}\cdot\frac{D_{air}(fs,\bar{x},E)}{D_{air}(fs_{max},\bar{x},E)}$$

$$\approx D_{air}(fs_{max},\bar{c},E_c)\cdot\Phi(\bar{x},E,\bar{x}_c,E_c)\cdot\frac{\|\bar{x}_c\|^2}{\|\bar{x}\|^2}$$

where

$$\frac{D_{air}(fs_{max},\bar{x}_c,E)}{D_{air}(fs_{max},\bar{c},E_c)} \equiv \Phi(\bar{x},E,\bar{x}_c,E_c)$$

$$\frac{D_{air}(fs_{max},\bar{x},E)}{D_{air}(fs_{max},\bar{x}_c,E)} \approx \frac{\|\bar{x}_c\|^2}{\|\bar{x}\|^2}$$

$$\frac{D_{air}(fs,\bar{x},E)}{D_{air}(fs_{max},\bar{x},E)} \approx 1$$

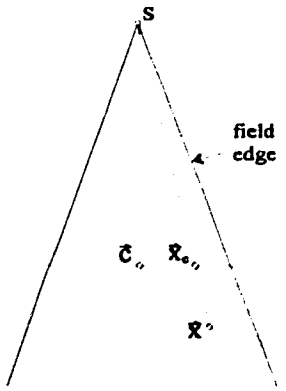


Figure 2.3. Variable definitions.

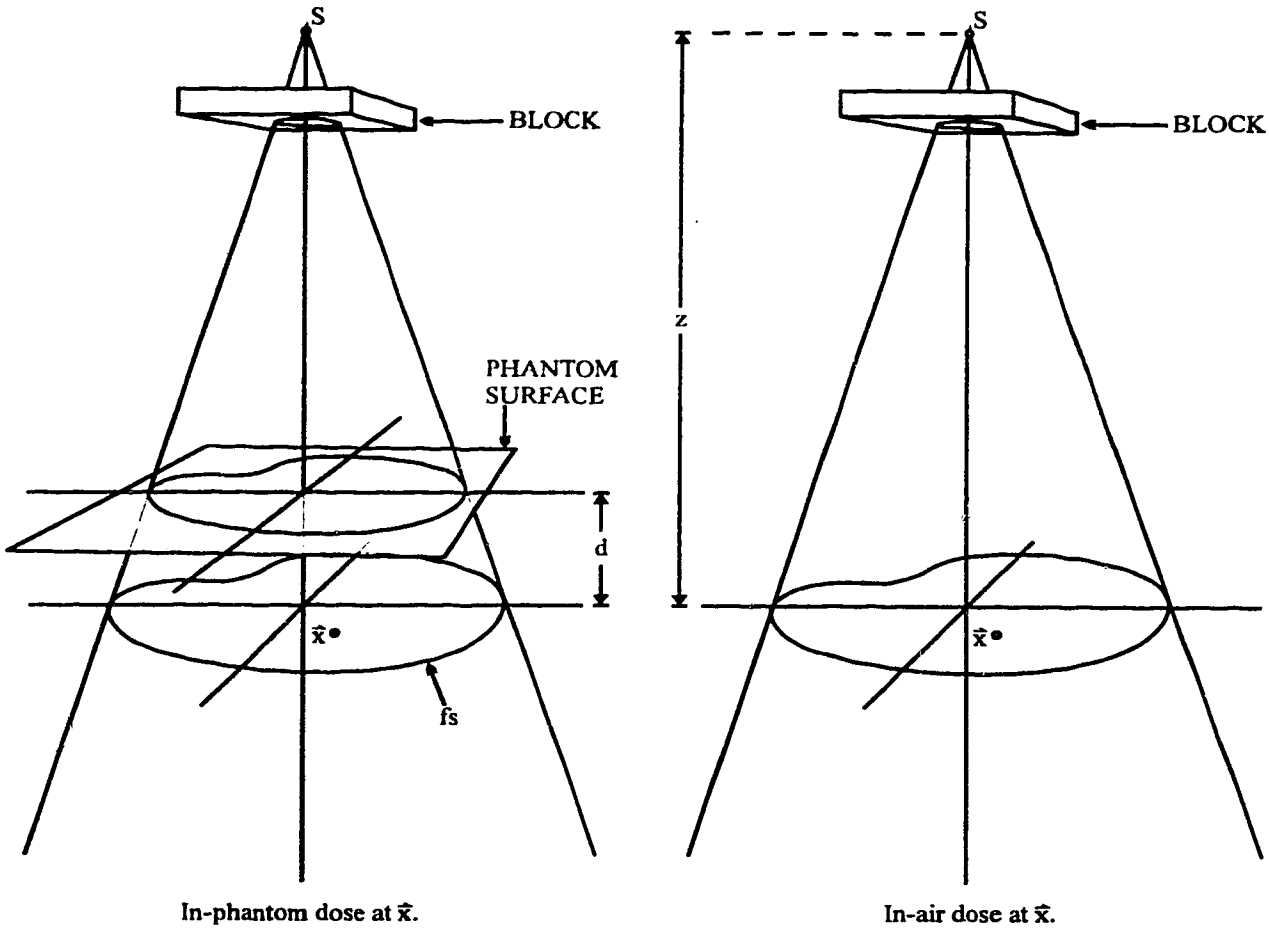


Figure 2.4. Definition of TAR for an arbitrary shaped field.

$$TAR(d, fs, \bar{x}) = \frac{D(d, fs, \bar{x}, E)}{D_{air}(fs, \bar{x}, E)} \qquad [2.1]$$

d = depth of water above measurement point,
fs = shape of field at the measurement point,
$\bar{x}$ = position of measurement point relative to source,
E = energy (spectrum) of beam,
TAR = tissue - air ratio,
D = measured dose in phantom,
D_{air} = measured dose in air.

Note that in this definition the TAR is not dependent upon changes in the beam energy spectrum E with position in the phantom. This is not true in general, but it is assumed that the change in energy spectrum across a radiation field is such that it does not significantly change the TAR. (In section 4.3 the validity of this assumption will be discussed.)

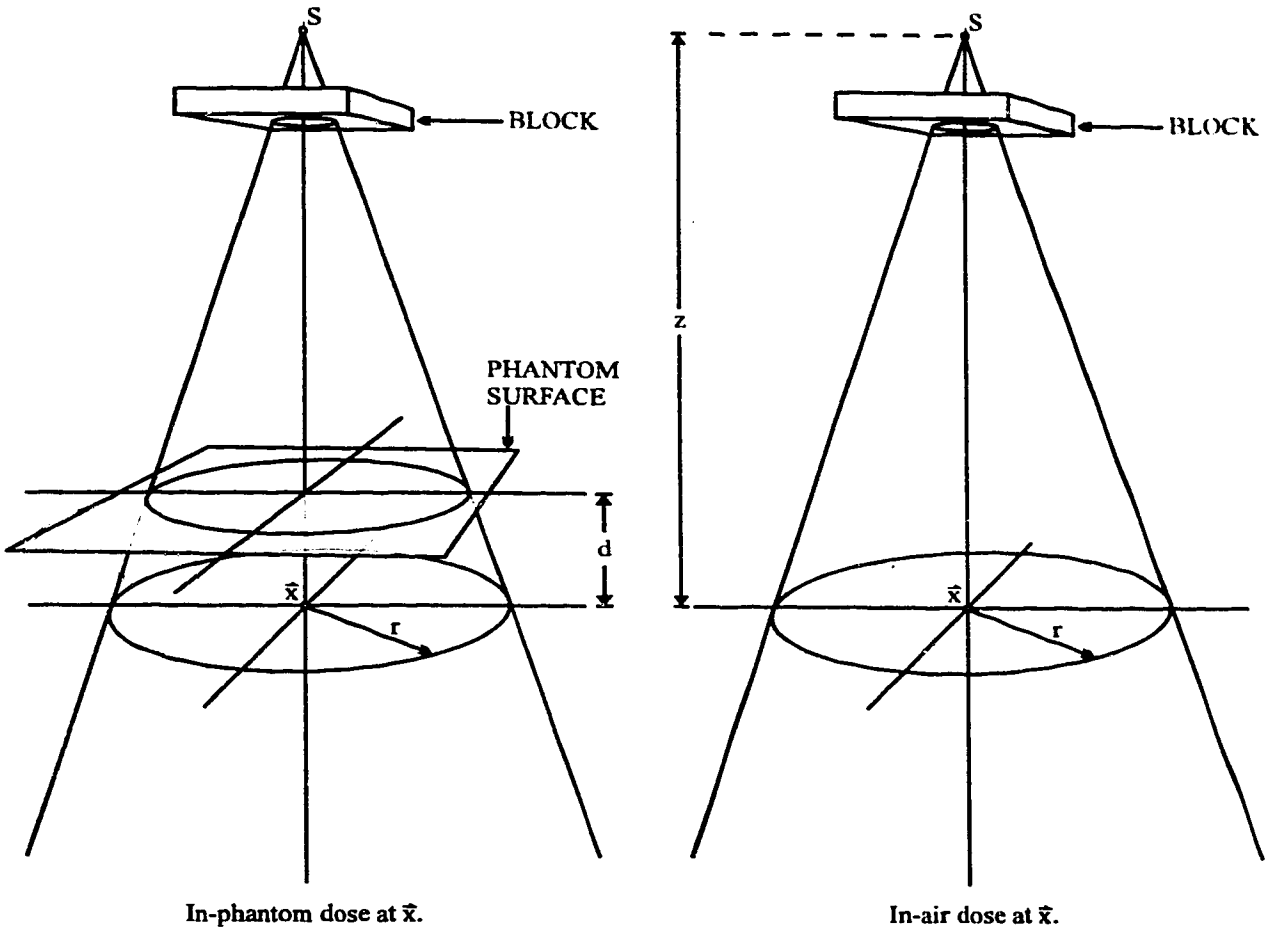


Figure 2.5. Definition of TAR for a circular field on the central axis.

$$TAR(d,r) = \frac{D(d,r,z,E)}{D_{air}(r,z,E)} \qquad [2.2]$$

d = depth of water above measurement point,
r = radius of field at the measurement point,
z = distance of measurement point from source,
E = energy (spectrum) of beam,
TAR = tissue - air ratio,
D = measured dose in phantom,
D_{air} = measured dose in air.

Note in equation 2.2 that TAR is not dependent upon the distance of the measurement point from the source. This is an assumption that will fail over a wide range of distances z, but for the range of distances that a phantom might produce under the accelerator head the assumption is quite accurate. Thus, with these assumptions the TAR only needs to be tabulated for a full range of depths and field radii.

The TAR for a general field shape and calculation point position is constructed from measured TARs for circular fields centered on the central axis (equation 2.2.). This is accomplished by approximating the field shape with sectors of a polar coordinate system that is centered on the calculation point (figure 2.6.). Each sector represents a known contribution to the TAR of the corresponding circular field of the same radius [Johns et. al. 1983]. That is,

$$dTAR(d,r)=\frac{TAR(d,r)}{N} \qquad [2.3]$$

where dTAR (differential tissue-air ratio) is the contribution from one sector to the total TAR, and N is the number of sectors.

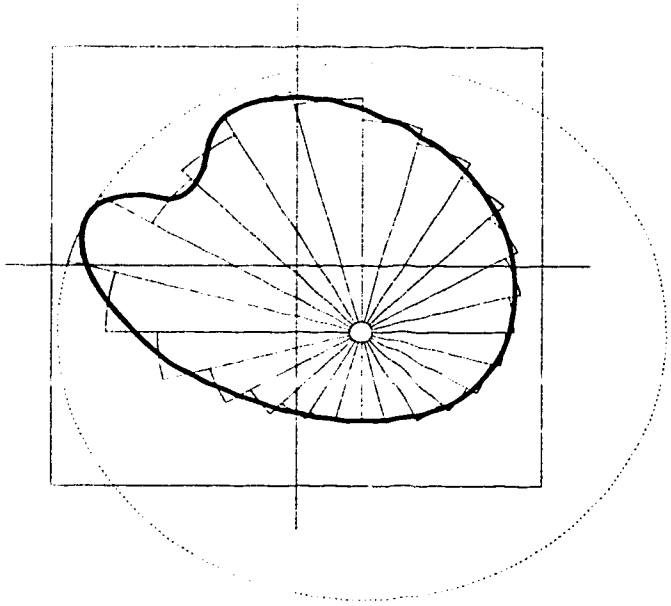


Figure 2.6. Construction TAR for an arbitrary shaped field.

The total TAR at the calculation point is then given by

$$TAR(d,fs,\bar{x})=\sum_{i=1}^{N} dTAR(d,r_i \qquad [2.4]$$

This method of calculation still suffers from some assumptions. It is assumed that the fluence incident on the phantom surface is uniform over a given sector. This is obviously false in general, as can easily be seen by introducing a wedge filter into the beam path. It is also assumed that the depth of water (or other phantom material) is constant for an entire sector, which fails for all but a flat water tank. The beam shape is also not as general as it could be, since shapes like that in figure 2.7 cannot be dealt with adequately. As well, the measured data contains assumptions concerning charged particle equilibrium.

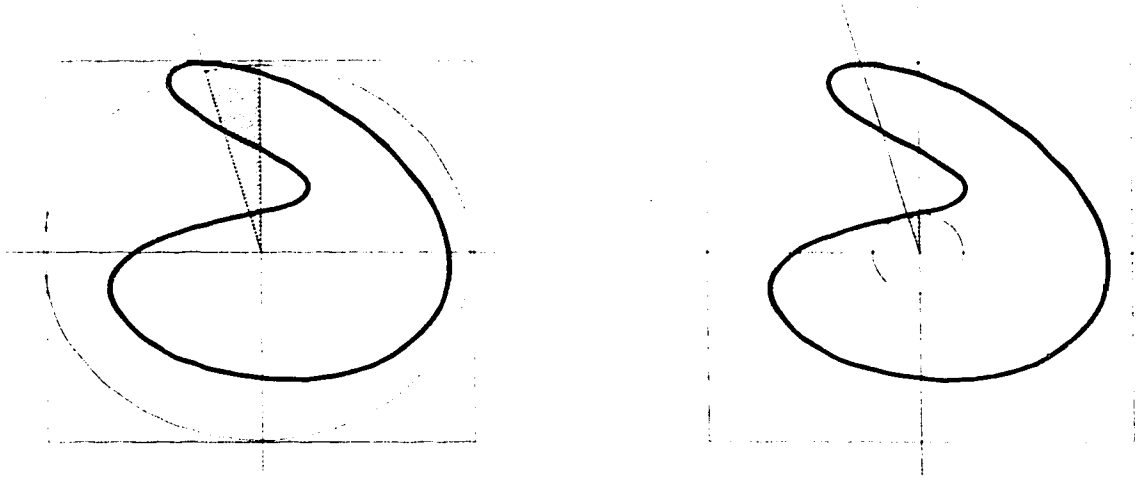


Figure 2.7. An inadequately handled beam shape.

Many of these assumptions can be eliminated by further partitioning the dTAR sectors radially. That is, partition the total TAR by both sector-cuts and annular-cuts, thereby creating smaller "pencil" contributions to the total TAR. (See figure 2.8.) Since these contributions only account for the scattered radiation from a given pencil to the calculation point (and not the primary dose due to unscattered radiation), they are referred to as "differential SCATTER-air ratios" (dSAR's).

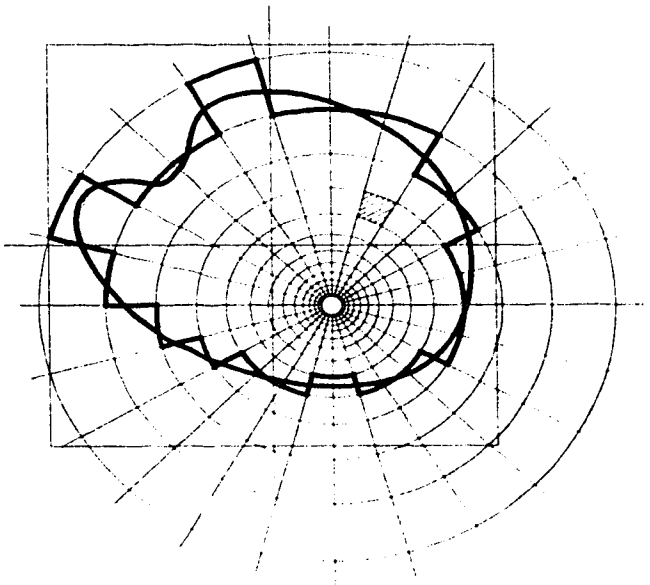


Figure 2.8. Partitioning of the phantom into "pencils" instead of sectors.

The exact method by which the dSAR contribution from each pencil is calculated will be discussed in section 3.2. It need only be noted that the dose to a calculation point is now calculated using the following formulae, where $dTAR_o$ is the contribution of primary radiation to the total dTAR for a given sector.

$$dTAR = \sum_r dSAR + dTAR_o \qquad [2.5a]$$

$$TAR = \sum_\theta dTAR \qquad [2.5b]$$

$$TAR = \frac{D}{D_{air}} \qquad [2.5c]$$

from which we obtain,

$$\begin{aligned} D &= D_{air} \cdot TAR \\ &= D_{air} \cdot \sum_\theta dTAR \\ &= D_{air} \cdot \sum_\theta \left[dTAR_o + \sum_r dSAR \right] \\ &= D_{air} \cdot \left[\sum_\theta dTAR_o + \sum_\theta \sum_r dSAR \right] \\ &= D_{air} \cdot \left[TAR_o + \sum_\theta \sum_r dSAR \right] \end{aligned} \qquad [2.6]$$

Each dSAR is separately evaluated with its own fluence and depth information, thus eliminating two of the assumptions mentioned previously. The problem with field shapes like the one shown in figure 2.7 is now solved as well. (See figure 2.9.)

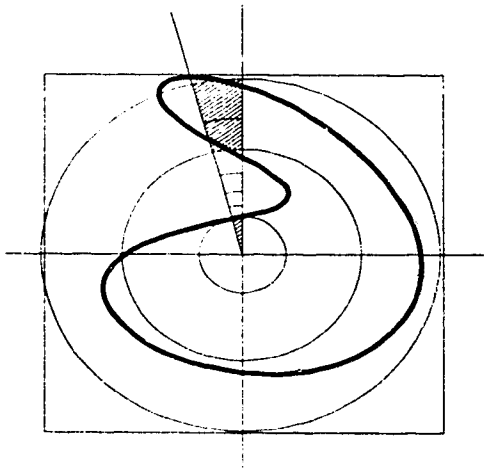


Figure 2.9. Solution to the beam shape problem.

The dSAR method is much faster than the Monte Carlo approach, since the particle-level physics is already "built into" the measured data used by the dSAR methods. That is, it need not be reconstructed through simulation as it is in Monte Carlo. Calculations with a semi-empirical approach of the dSAR type can be done in clinically acceptable time frames (minutes or even seconds), and thus are well suited for use in treatment planning software. However, the measurement process for the required quantities, and the use of the measured data in calculating dose to a point, still involves assumptions that may fail for more complex treatment configurations. For example, the measurements required for use by the dSAR algorithm are made with a diode or ion chamber, and these require electronic equilibrium in order that their output signal be linearly related to dose [Attix 1986]. Thus, for situations where the probe is close to the phantom surface or close to a field edge the measured data may not be reliable. Likewise, calculation of dose to a point close to the surface or a field edge involves either using unreliable measured data, or data that has been extrapolated from the reliable data set. In either case the calculation of dose to such a point is suspicious. (Recall that Monte Carlo did not make any assumptions about electronic equilibrium. The tradeoff in going back to Monte Carlo to eliminate the above problem is that of losing speed efficiency. A method that attempts to find the best of both worlds is discussed in section 2.3.)

The calculation volume is also assumed to be homogeneous since the empirical data being used is typically measured using a homogeneous phantom (i.e. a water tank). Inhomogeneities are usually taken into account using a correction algorithm, only after the dSAR calculation has generated a dose matrix [Wong et. al. 1990, Thomas 1991, El-Khatib et. al. 1984].

2.3. Convolution methods

A plausible compromise between a Monte Carlo method and a semi-empirical approach is to use Monte Carlo to generate the pre-stored bulk radiation interaction data that is used by a method such as dSAR. One could, for instance, generate the TAR data used by the dSAR method, rather than obtaining the TAR through measurements. This would remove any assumptions inherent in the measurement process and improve the dSAR approach.

A more popular method, however, is convolution [Boyer et. al. 1986, Boyer et. al. 1985, Ahnesjö et. al. 1992, Iwasaki 1992, Iwasaki 1990, Mackie et. al. 1985, Mackie et. al. 1984, Bourland et. al. 1992, Chui et. al. 1988]. This method also uses pre-generated data (using Monte Carlo) to calculate dose, but this data takes a different form than the TAR data of section 2.2. Essentially, a very narrow pencil-beam (or ray) of radiation, incident upon a homogeneous phantom, is simulated using Monte Carlo. A 3-dimensional distribution of dose deposition is obtained from this simulation, and this distribution (or dose kernel) is used in the calculation of dose to a point in a general treatment situation. By considering a typical treatment beam to be made up of these thin rays of radiation, the calculation of dose to a point then involves summing up (integrating) the contribution from each of these rays. (See figure 2.10.)

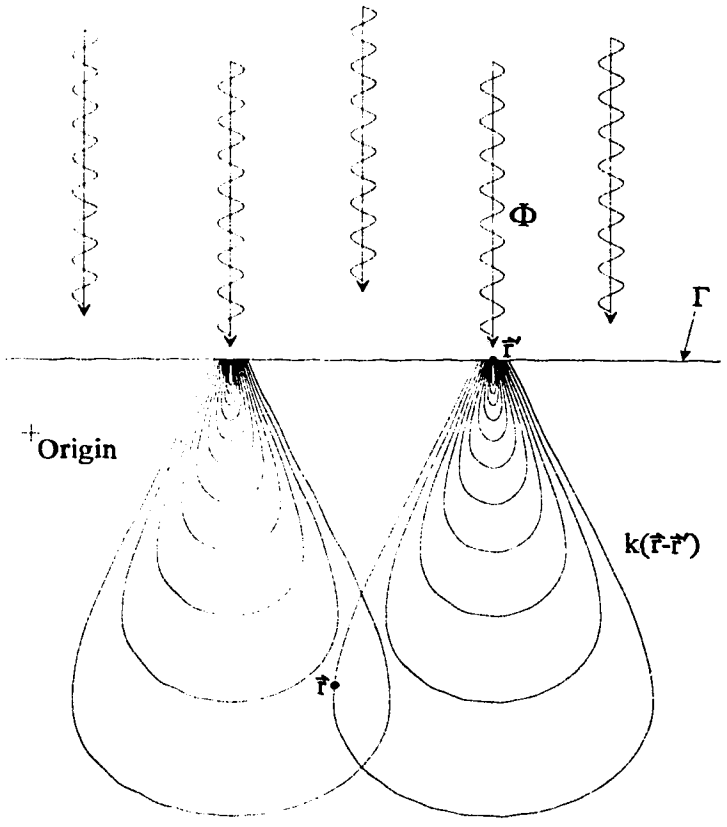


Figure 2.10. Dose calculation using the convolution kernel.

The integration takes the form shown in equation 2.7. It is basically a weighted sum of dose contributions k, where Φ is the weighting factor.

$$D(\bar{r}) = \iint_{\Gamma} \Phi(\bar{r}') \cdot k(\bar{r} - \bar{r}') \cdot d\bar{r}' \qquad [2.7]$$

$\Phi(\bar{r}')$	is the energy fluence incident upon the phantom surface at the point $\bar{r}'$.
$\bar{r}' = \bar{r}'(u, v)$	describes the phantom surface; parametrized by two variables u and v.
Γ	represents the "patch" of the phantom surface that is being irradiated, and hence the integration must be performed over Γ.

This version of the convolution assumes that the beam is monoenergetic, since $k(\bar{r} - \bar{r}')$ will be different for different energies. Thus, provided that the spectrum of the beam is known at all parts of the field, the convolution integral should also include an integration over energy.

$$D(\bar{r}) = \int \iint_{\Gamma} \Phi(\bar{r}', E) \cdot k(\bar{r} - \bar{r}', E) \cdot d\bar{r}' \cdot dE \qquad [2.8]$$

This is already a large improvement over the semi-empirical dSAR method of section 2.2 since that method does not take into account the change in energy spectrum across a field.

The integration in equation 2.7 is usually performed using a Fourier Transform, which allows a convolution to be expressed as a product.

$$\left\{ \begin{array}{l} h(x) = \int_{-\infty}^{+\infty} f(y) \cdot g(x - y) \cdot dy \\ F(k) = \mathcal{F}[f(y)] = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(y) \cdot e^{-iky} \cdot dy \\ G(k) = \mathcal{F}[g(z)] = \frac{1}{2\pi} \int_{-\infty}^{+\infty} g(z) \cdot e^{-ikz} \cdot dz = \frac{1}{2\pi} \int_{-\infty}^{+\infty} g(x - y) \cdot e^{-ik(x-y)} \cdot dx \\ H(k) = \mathcal{F}[h(x)] = \frac{1}{2\pi} \int_{-\infty}^{+\infty} h(x) \cdot e^{-ikx} \cdot dx \end{array} \right\}$$

$$\Rightarrow \quad H(k) = 2\pi \cdot F(k) \cdot G(k) \qquad [2.9]$$

The "Fast Fourier Transform" is a software implementation of this procedure that is quite efficient in terms of calculation speed. Thus the convolution method, although still slower than the dSAR approach, is significantly faster than Monte Carlo.

The convolution method still makes a number of assumptions that affect the accuracy of the dose calculation [Field et. al. 1987]. For instance, in order for the algorithm to contain a true convolution, and hence allow the use of the properties of the Fourier Transform (c.f. equation 2.9), it must be assumed that the incident pencil beams are perpendicular to the phantom surface. A treatment beam is typically diverging from a small source, and hence the kernel in the integral of equation 2.7 should be "tipped" as a function of distance from the central axis. (See figure 2.11.)

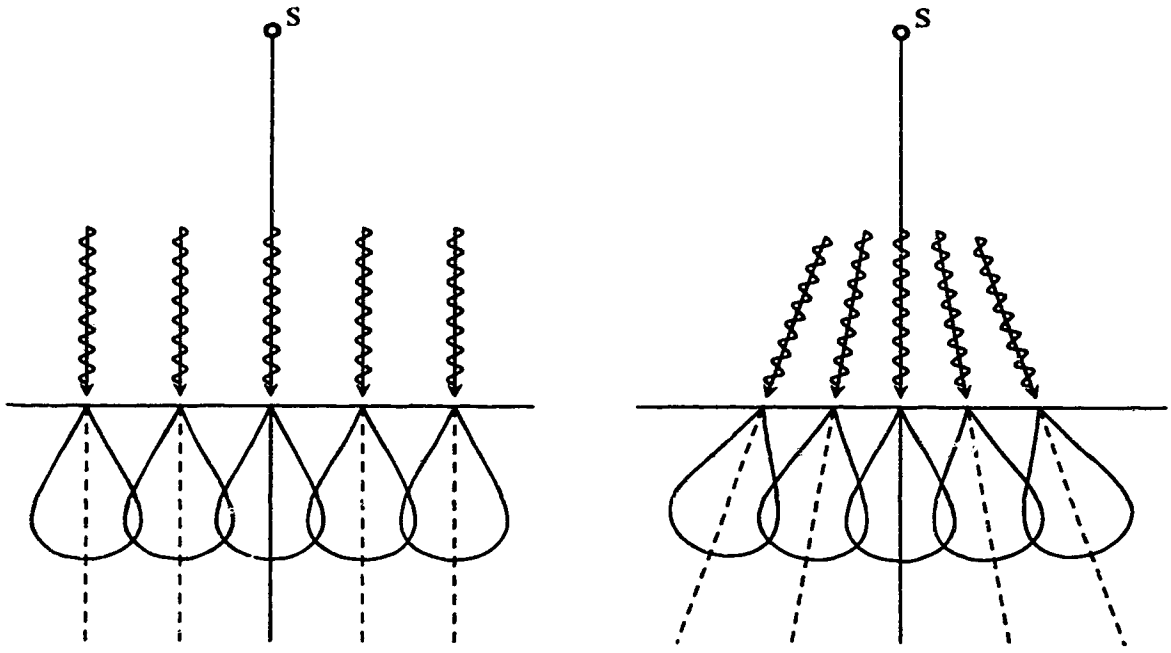


Figure 2.11. Correcting the parallel incidence assumption.

This tipping of the kernels, however, will change the integral of equation 2.7 into a superposition integral, denying the use of the Fourier Transform property outlined in equation 2.9. The resulting integral must then be evaluated using very time consuming methods.

The convolution method also assumes a homogeneous phantom, and modification of the algorithm to remove this assumption will again increase calculation time significantly.

Convolution approaches will still require inconvenient amounts of time to calculate treatment plans until faster computing technology becomes commonplace. It is reasonable to ask whether the trade off of accuracy versus speed between convolution methods and semi-empirical methods is necessary. Perhaps the dSAR approach can be made to have acceptable accuracy for most treatment planning situations and still be reasonably fast.

It is upon this premise that this thesis is based, i.e. that the dSAR approach should be investigated and enhanced to see what its limits of accuracy are. Provided that the limits of accuracy are well defined, the dSAR approach could still be the algorithm of choice for many treatment planning situations. Beyond these limits, a combination of algorithms like dSAR and convolution, or the use of some other strategy altogether, might be in order [Mohan et. al. 1991, Wong et. al. 1983, Marinello et. al. 1992, Desobry et. al. 1994, Nizin 1991].

3. What is GRATIS?

3.1. Description of the GRATIS system

GRATIS† is a treatment planning package created by Dr. George Sherouse at the University of North Carolina. It consists of a number of separate programs for the simulation of the treatment planning situation, display of CT data, calculation of dose based on the simulation, analysis of the calculation results, and graphical display of information. The entire package is written in C-language for a UNIX system with X-windows display software, and the research for this thesis was performed using the source code for GRATIS Release 4 (July 26, 1992).

A typical treatment planning procedure using GRATIS is shown in the flow chart in figure 3.1. It first involves the virtual simulation software, which is a single UNIX script called `simulate` that invokes all the necessary GRATIS utilities for creating, viewing and modifying data that might otherwise be gathered at an actual simulator. Execution is performed with the command

```
simulate name
```

where `name` is the name of a virtual patient. The virtual simulator presents the user with a variety of graphical windows that display CT images, three dimensional representations of the patient anatomy, and panels for controlling patient orientation, beam configuration, graphical display characteristics, and the placement of beam filters. An example of the graphical interface is shown in figure 3.2.

The user can create a new beam, or modify an old beam from a stored list. The collimator positions can be modified, beam modifying devices (i.e. wedges and compensators) can be added and removed, irregular beam cross sections can be created, and the virtual patient can be rotated and translated relative to the central axis. This allows the user to simulate a general treatment configuration and, when satisfied with the geometry, the treatment description can be entered back into the beam list. All beams that appear in this beam list will be displayed in the graphical interface, so that a three dimensional representation of a multiple beam configuration can be visually analyzed. This 3D view can be rotated, translated and magnified. As well, reference points in the patient can be specified for the purpose of prescribing dose when the calculation routine is executed.

† **Georges RAdiotherapy Treatment desIgn System.**

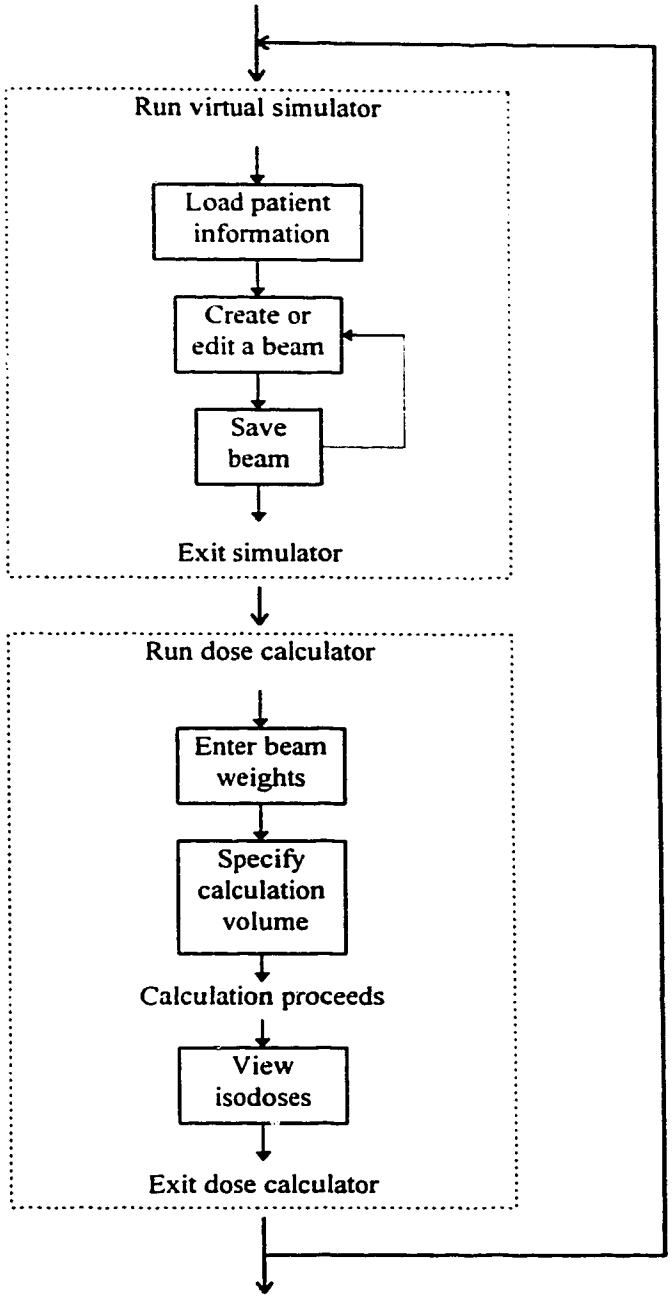


Figure 3.1. The treatment planning process.

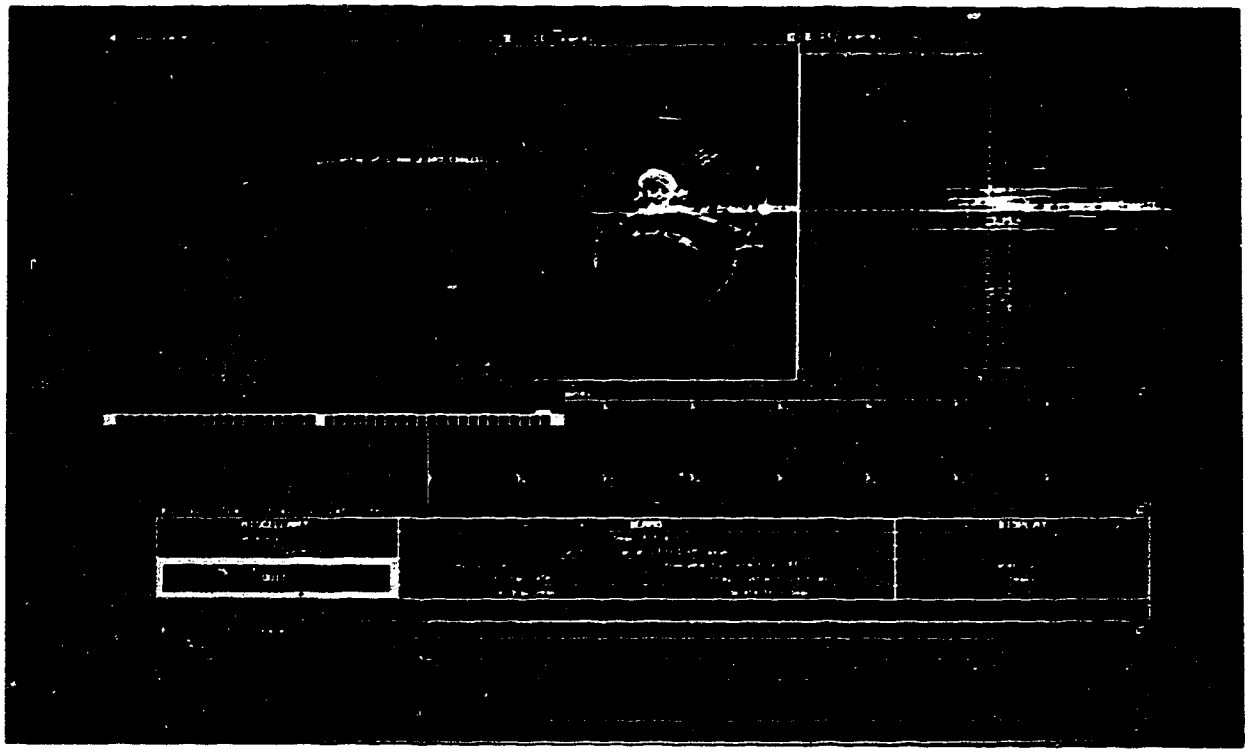


Figure 3.2. The virtual simulator.

Once the user is satisfied with the beam configurations that have been entered (i.e. the target volume is sufficiently covered, critical tissues are spared, etc.), the virtual simulator is exited and a second GRATIS script file is executed by the user with the command

```
extbm name
```

This script first runs the calculation routine of GRATIS to obtain the dose at the reference points specified during the virtual simulation. A spreadsheet of beam weights and reference point doses is then presented to the user so that dose can be prescribed and beam weights adjusted (figure 3.3).

		beams		
	dose	1	2	3
rel. weight -->		2.00	2.00	1.00
beam norm. -->		1.00	1.00	1.00
plan norm. -->		47.61	47.61	47.61
fractions -->		1	1	1
Spine	17.49	3	3	95
L Lung	66.15	60	39	2
R Front	83.98	1	45	54
R Side	62.53	38	60	2
L Front	75.54	39	2	59
Top	78.56	10	41	49
Bottom	97.05	29	32	39
Isocenter of L f	100.02	30	31	39
Isocenter of R f	100.00	30	31	39
Isocenter of Fro	100.00	30	31	39

Referenced to: open fields

Figure 3.3. The dose prescription routine.

Exiting the dose prescription spreadsheet will automatically cause a graphical utility to be executed that allows the user to define the position, size and resolution of the three dimensional grid of points at which dose is to be calculated. The rectangular volume enclosing this dose grid is displayed along with the defined beams and the patient anatomy so that the user can identify the region of interest and restrict the computation to dose points in that region (figure 3.4). This reduces calculation time by calculating dose only for points around the regions of interest (i.e. the tumor and critical organs).

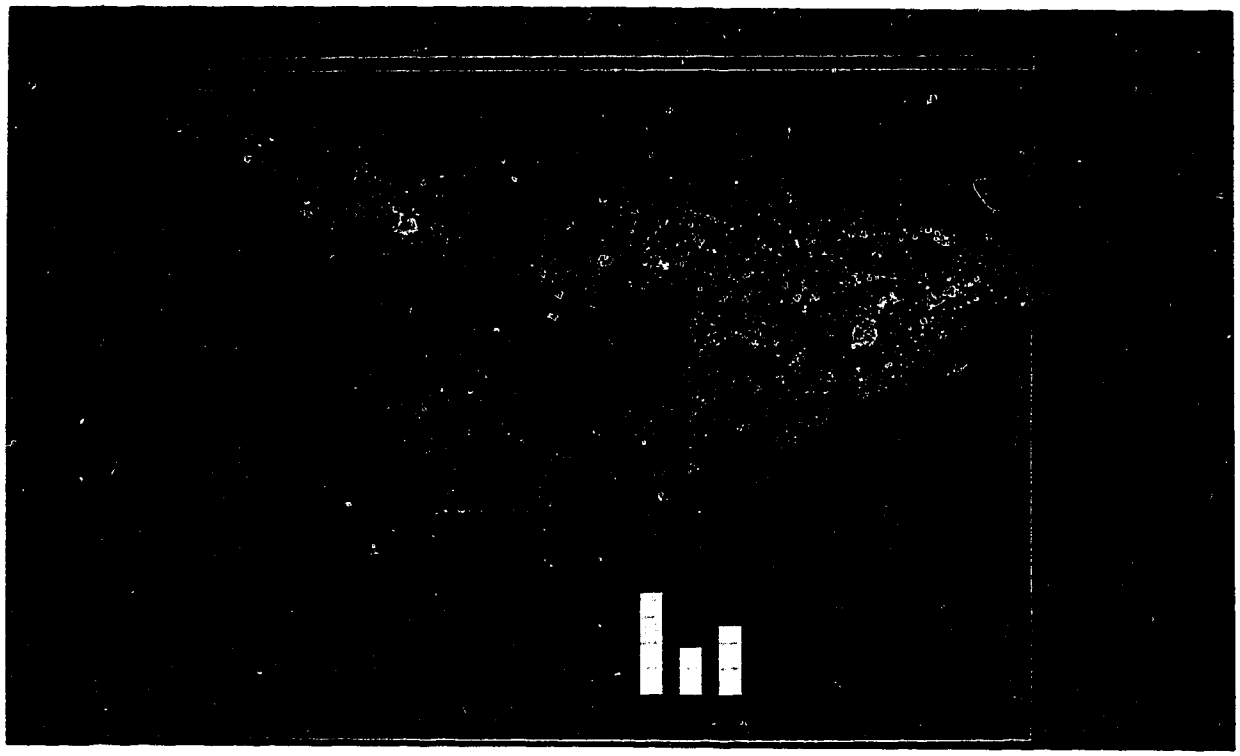


Figure 3.4. The grid volume editing utility.

After defining the grid volume the user exits this utility and routines begin calculating the dose for points on the grid. Completion of this calculation is indicated by the activation of a graphical utility for viewing isodoses (figure 3.5).

The isodose display utility overlays isodose lines on the CT image data that was presented in the virtual simulator. The user can change the magnification, move the image around, and select which isodose curves are to be displayed. Exiting this utility then presents the user with the final option of printing the displayed isodoses.

If the displayed isodoses do not meet the user's criteria for an acceptable treatment plan then the beam configuration is again edited using the virtual simulator and the above process is repeated.

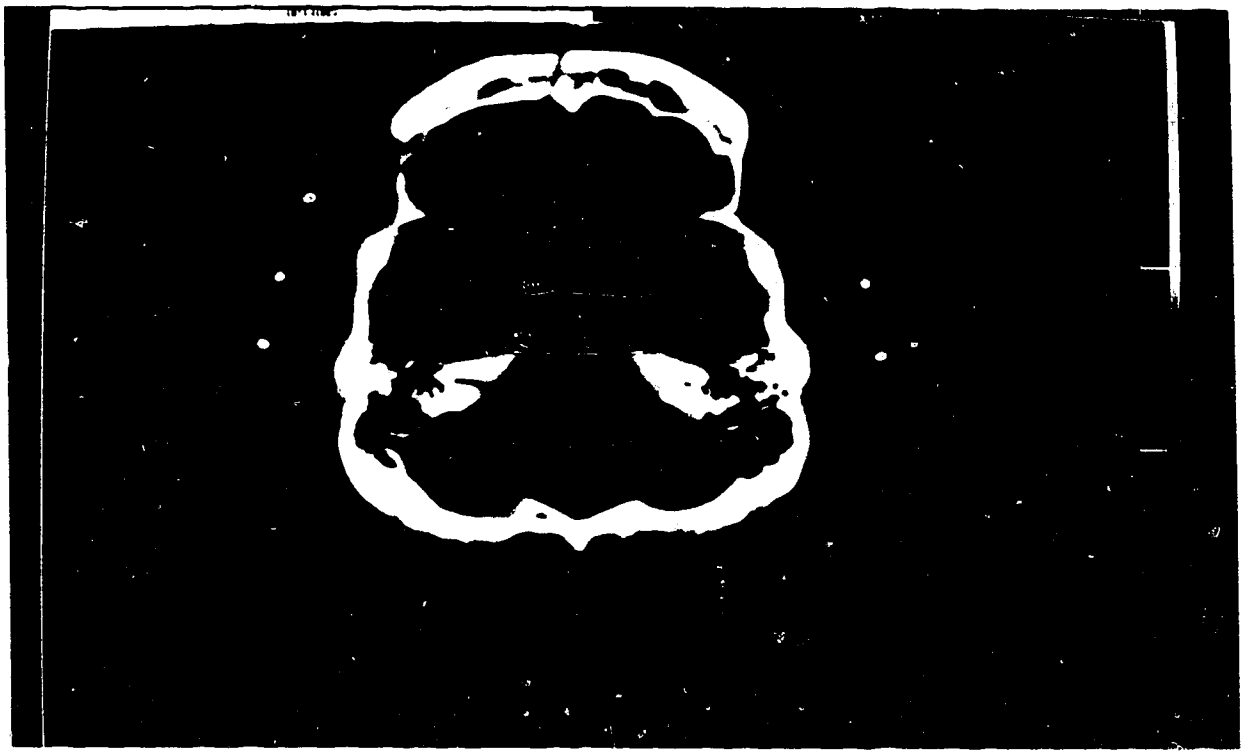


Figure 3.5. The isodose display utility.

3.2. Description of the c_photon algorithm

The dose computation program executed by the `extbm` script is called `c_photon`, which is a generalization of the algorithm discussed by John Cunningham [Cunningham 1972, Johns et. al. 1983]. This approach calculates the absorbed dose to a point in an irradiated medium by separately evaluating the contributions due to primary and scattered radiation. Primary radiation is incident upon the calculation point with no scattering in the phantom, whereas scattered radiation reaches the calculation point only after scattering by the phantom material. The basic calculation can be summarized by the following equation (c.f. equation 2.6).

$$\begin{aligned} D &= D_{air} \cdot TAR \\ &= D_{air}(x,y,z) \cdot \{TAR_o(x,y,z) + SAR(x,y,z)\} \\ &= D_{air}(x,y,z) \cdot \left\{ TAR_o(x,y,z) + \sum_{r,\theta} dSAR(r,\theta,d) \right\} \end{aligned} \qquad [3.1]$$

TAR_o represents the primary dose contribution and SAR is the contribution from scattered photons. The summation over r and θ represents how SAR is constructed by summing up the scatter contributions from small volume elements (referred to as dose 'pencils') within the irradiated medium. One such element is shown in figure 3.6, as well as the geometry for partitioning the irradiated medium. This partitioning is the result of using a polar grid and will be referred to as the "dartboard geometry" (for obvious reasons). Alternatively, the irradiated medium could be partitioned using a rectangular grid (referred to as the "cartesian geometry"). The dartboard (cartesian grid) is imagined to be drawn on a plane that passes through the calculation point, and also centered on the calculation point (figure 3.7).

The scattering properties of each pencil, as a function of distance 'r' from the calculation point and tissue depth 'd' of the pencil above the calculation point, are stored in GRATIS in the form of a differential scatter-air ratio (dSAR). The dSAR accounts for the contribution to dose due to scattered radiation from that pencil only. Summing up the differential scatter-air ratios over all pencils within the irradiated medium will reconstruct the total SAR.

One advantage of partitioning the irradiated medium into pencil-shaped volume elements is that each pencil can be separately weighted by its own beam intensity, relative to a reference intensity, thus allowing for an arbitrary intensity distribution over the phantom, such as might be generated by filters and compensators. There is also the advantage that each pencil can have its own height 'd', independent of the other pencils, which allows for a reasonable approximation to the changing contour of an arbitrary patient surface.

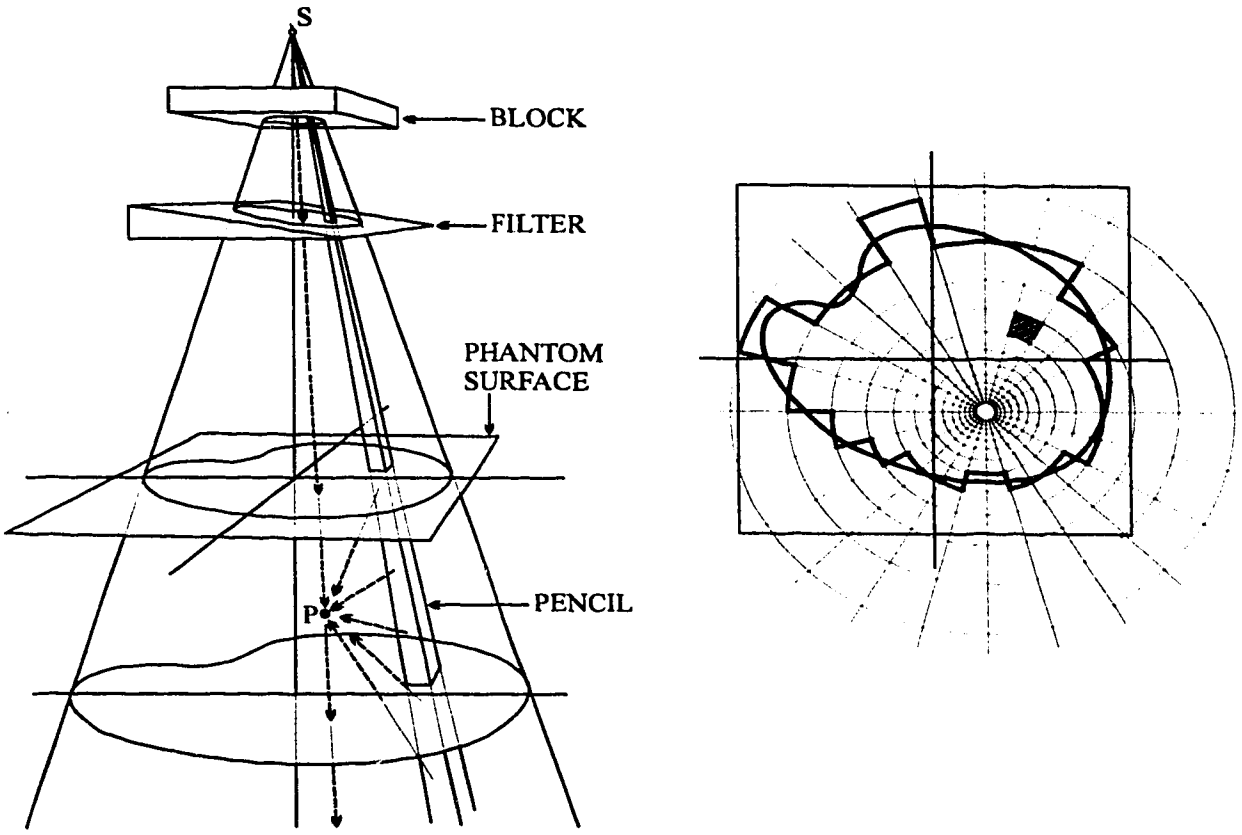


Figure 3.6. The geometry of the dose calculation.

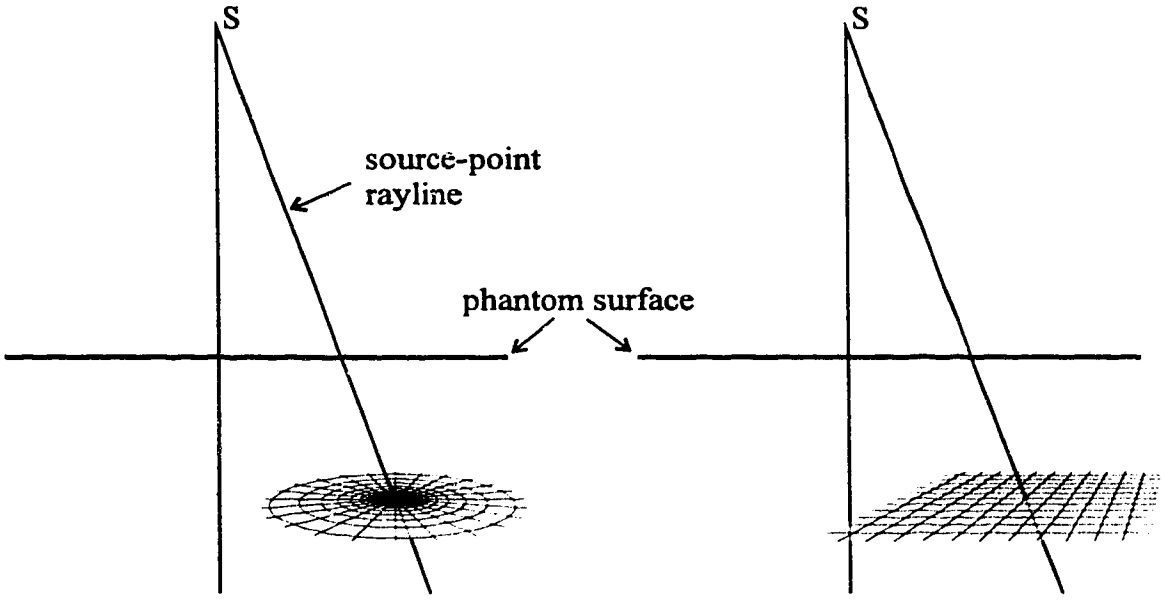


Figure 3.7. Interpretation of dartboard and cartesian grid partitioning.

The dSAR in equation 3.1 can then be expressed as follows.

$$\begin{aligned} dSAR(r,\theta) &= dSAR(r,\theta,x,y,z) \\ &= rel\_intensity(r,\theta,x,y)\cdot \Delta SAR(r,d(r,\theta,x,y,z)) \end{aligned} \qquad [3.2]$$

The function rel_intensity(r,θ,x,y) describes the variation of the energy fluence across the field, and d(r,θ,x,y,z) describes the variation of the patient surface relative to the calculation point.

The implementation of this calculation in c_photon actually takes a form similar to the following.

$$P = \sqrt{x_p^2 + y_p^2 + z_p^2}$$

$SAR = 0$

for $r = 0$ to r_{max} do

  for $\theta = 0$ to $360°$ do

$$(x_c, y_c, \phi) = F(x_p, y_p, z_p, r, \theta) \qquad [3.3a]$$

$$d = P - SSD(x_c, y_c)\cdot\cos(\phi) \qquad [3.3b]$$

$$SAR = SAR + rel\_intensity(x_c, y_c)\cdot \Delta SAR(r,d) \qquad [3.3c]$$

  end for

end for

- (x_p,y_p,z_p) are the coordinates of the calculation point relative to the source,
- P is the source-to-calculation-point distance,
- r and θ are the polar coordinates of the pencil relative to the calculation point,
- φ is the angle subtending the source-point rayline and the source-pencil rayline,
- d is the "depth" or height of the pencil above the calculation point,
- (x_c,y_c) are the coordinates of the top of the pencil, projected back to a reference plane,
- and F represents the necessary coordinate transforms to obtain (x_c,y_c) and φ.

The functions $SSD(x_c,y_c)$, $rel\_intensity(x_c,y_c)$ and $\Delta SAR(r,d)$ are expressed in terms of coordinates that allow these functions to be prestored in c_photon before the calculation takes place. That is, $SSD(x_c,y_c)$ and $rel\_intensity(x_c,y_c)$ are stored in terms of coordinates on a reference plane, which makes them independent of where the calculation point and pencil are. Likewise, $\Delta SAR(r,d)$ is stored in terms of coordinates that are relative to the calculation point, thus removing the dependence upon the absolute coordinates (i.e. relative to the patient and source) of the calculation point.

Alternatively, one might view (x_p,y_p,z_p,r,θ) as the coordinates (at calculation time) for a calculation point in an arbitrary beam geometry, and (x_c,y_c,ϕ,r,d) as coordinates in the specific measurement configuration when the input data to GRATIS was first acquired. The function F then represents the mapping of a calculation point in an arbitrary situation to an equivalent point in a measurement configuration.

Premeasured data is required by c_photon to carry out the scatter integration. This data consists of the following.

- An in-air profile is measured across the diagonal of the largest possible field on the treatment unit. This consists of scanning an ion chamber with buildup cap across the field at 100 cm from the source.
- Percent depth doses are taken along the central axis for a full range of sizes of symmetric square fields. Each percent depth dose is obtained by scanning a diode along the central axis through all depths from 0 to 30 cm, and then renormalizing all readings so that they are expressed as a percentage of the maximum value.
- Ion chamber readings are taken in-air at isocenter for a full range of field sizes.
- Ion chamber readings are taken at isocenter in a water phantom at the depth of maximum dose.
- Data that characterize the mechanical properties of the accelerator and treatment couch are recorded, such as the range of couch motions, the source to isocenter distance, etc.

The rel_intensity function introduced in equation 3.2 was constructed from the in-air profile by fitting the in-air profile to a smooth function (figure 3.8), which in this case consists of quartic and linear functions joined smoothly at 6 cm distance from the central axis. The linear component fits the data well between 6 cm and 19 cm from the central axis and allows the "roll off" of the profile close to the field edge (penumbra effect) to be eliminated by linear extrapolation. The quartic component was chosen since it is a simple function (i.e. a low order polynomial) that fits the data below 6 cm from the central axis, and it serves merely as a method of smoothing the data in this region. This quartic polynomial was constrained to match both value and slope with the linear fit at 6 cm off-axis, and also to have zero slope at isocenter to prevent a cusp from developing there. A cubic polynomial could also have been used to satisfy these conditions, but then it is so heavily constrained that it does not fit the data well. Thus the quartic polynomial was found to be the lowest order polynomial that was sufficient to fit the data. Finally the profile was normalized so that it has the value 1.00 at isocenter.

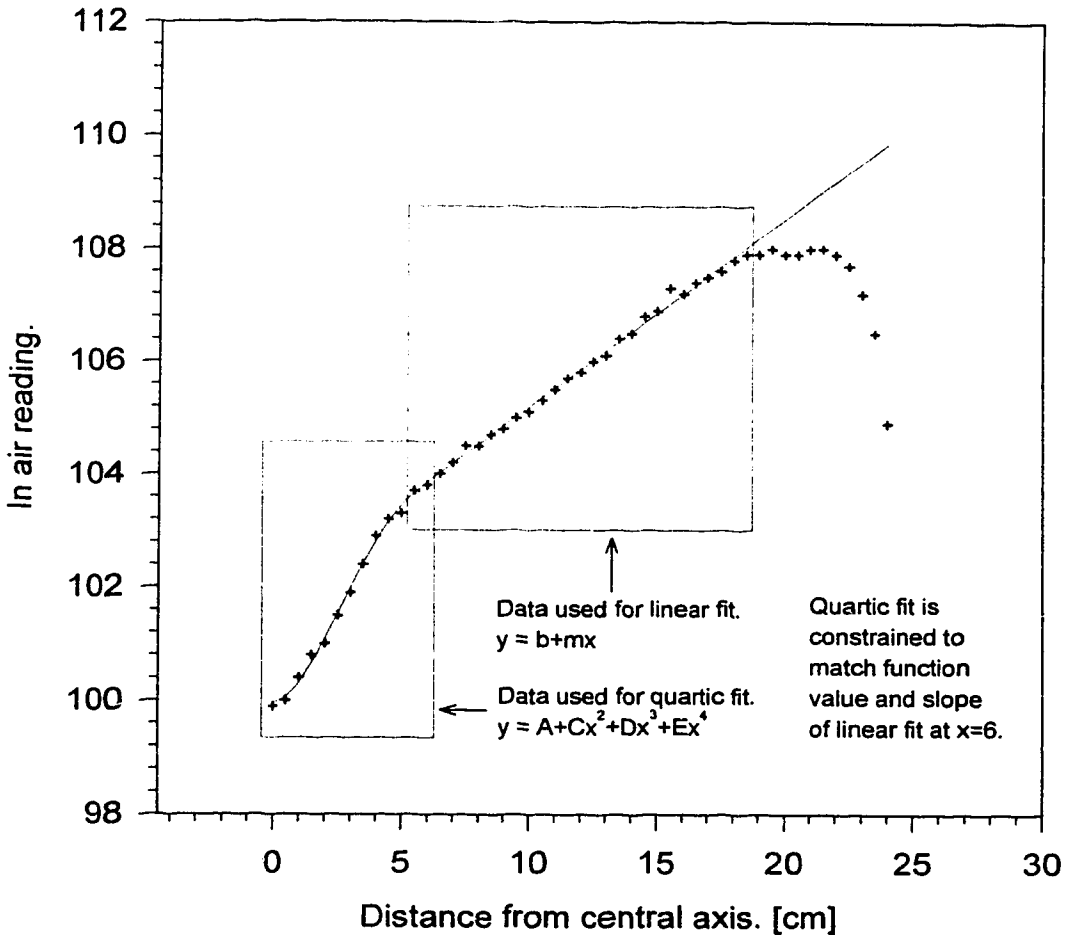


Figure 3.8. Construction of the rel_intensity distribution from the in-air profile.

The peak scatter factors (PSFs) are constructed from the ion chamber measurements at isocenter by taking the ratios of the in-phantom over the in-air measurements. A fit of these ratios versus field size to equation 3.4 allows extrapolation to field sizes less than 3 cm, and the PSFs are obtained by renormalizing these ratios to 1.0 at zero field size (figure 3.9).

$$y = P + S \cdot (1 - e^{-\mu \cdot x}) \qquad [3.4]$$

where P, S, and μ are fitting parameters, y is the ratio of ion chamber readings, and x is the field size.

Equation 3.4 was chosen simply because it is a relatively simple equation, and also because it has some properties that are also expected in the data. For instance, as the field size increases the PSF is also expected to increase, but not without bound. Thus the fitting function must asymptotically approach a maximum PSF value. The PSF is also expected to have some positive non-zero value for a field size of zero, representing primary contribution only. The fitting function thus must have a positive y-intercept.

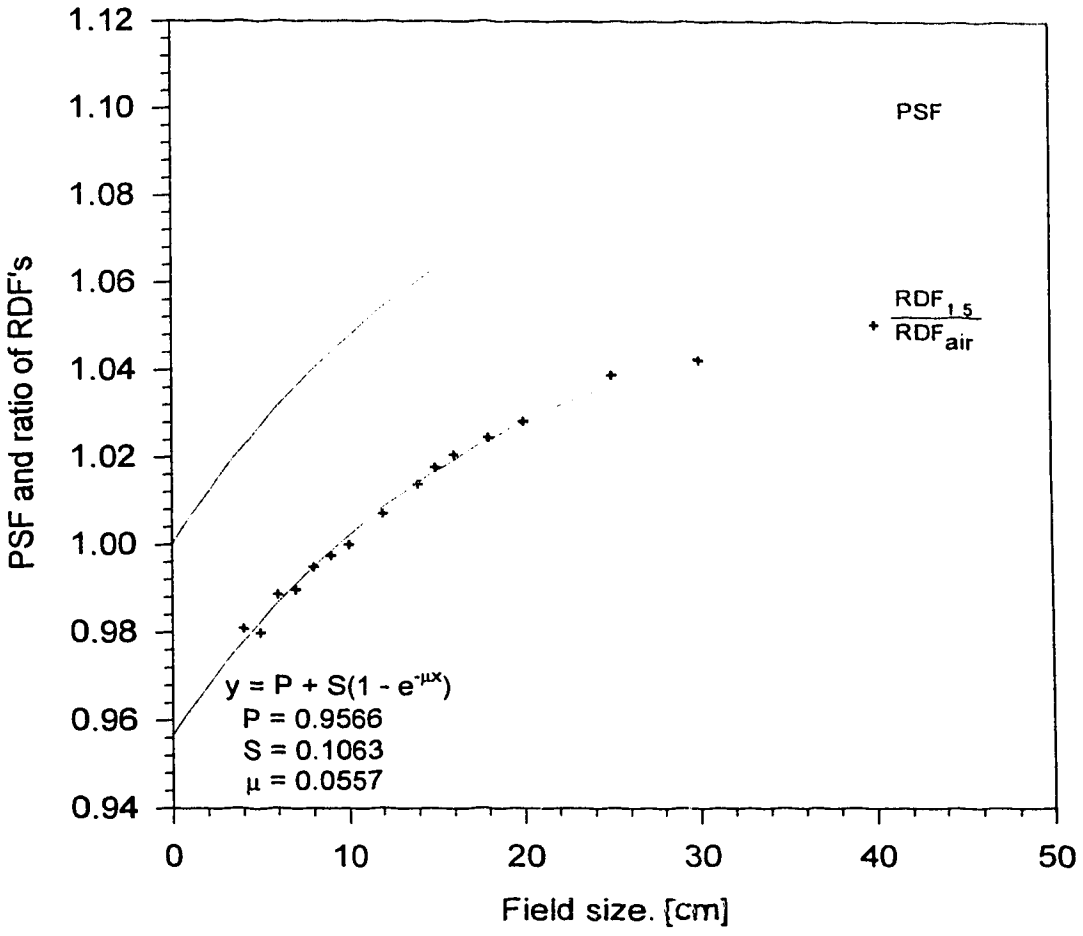


Figure 3.9. Construction of the peak scatter factors.

The percent depth doses (PDDs) were measured for square field sizes from 3 × 3 cm^2 (SSD = 100 cm), to the maximum allowable field size of 40 × 40 cm^2, in 1 cm increments. For field sizes less than 3 cm the PDD values were obtained once again by extrapolation using the fitting function in equation 3.4. In this case, y represents the PDD values at a fixed depth. A separate (independent) fit was performed for each depth from 0 cm to 3 cm in increments of 0.1 cm, and from 3 cm to 30 cm in increments of 0.5 cm. Examples of this extrapolation for depths of 10 to 15 cm is shown in figure 3.10.

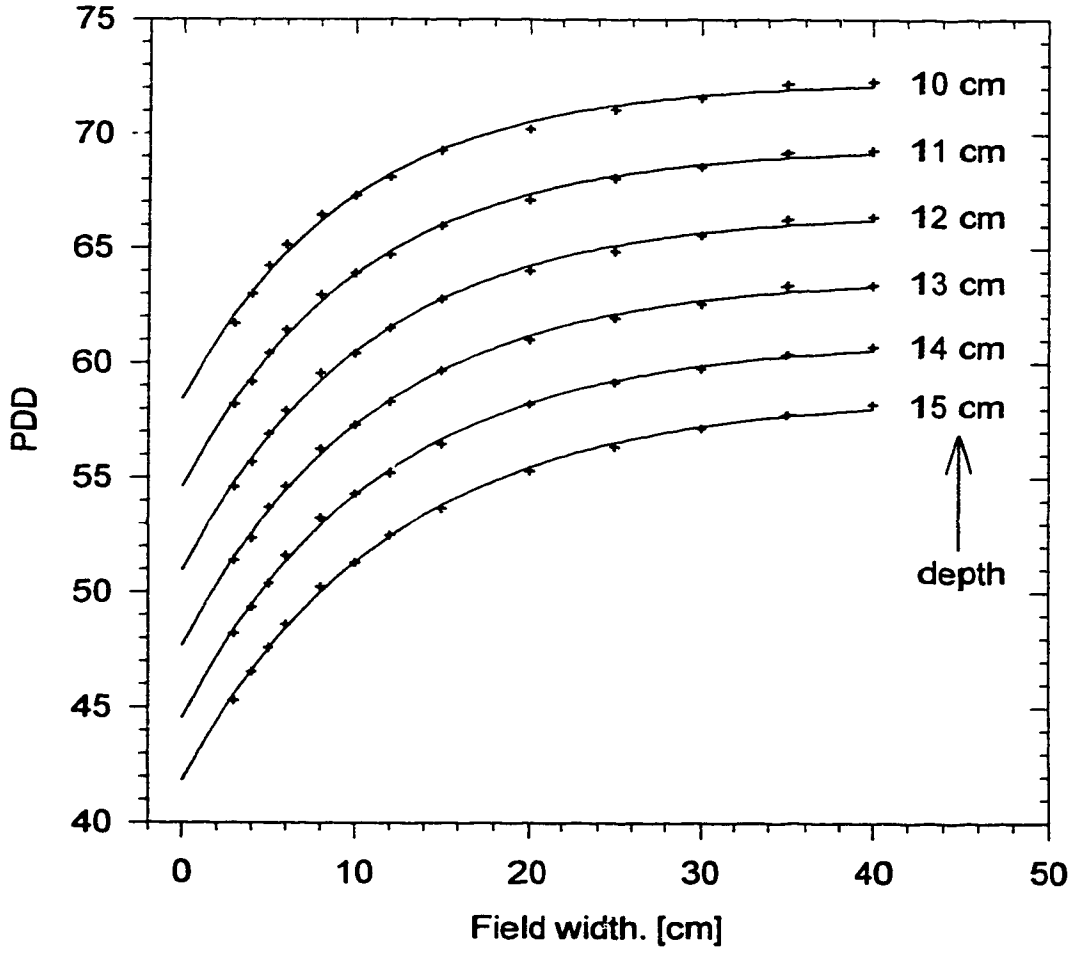


Figure 3.10. Extrapolation of the PDD data.

The parameters P and S that resulted from equation 3.4 were plotted as a function of depth, and a smooth curve was drawn through the points by hand. The fitting procedure was then repeated using the smoothed values for P and S to constrain equation 3.4. This procedure serves to smooth the PDD data versus depth and versus field size, without having to resort to a two variable fitting function, the form of which is unknown. Figures 3.11a through 3.11c show the final fitting parameter values versus depth.

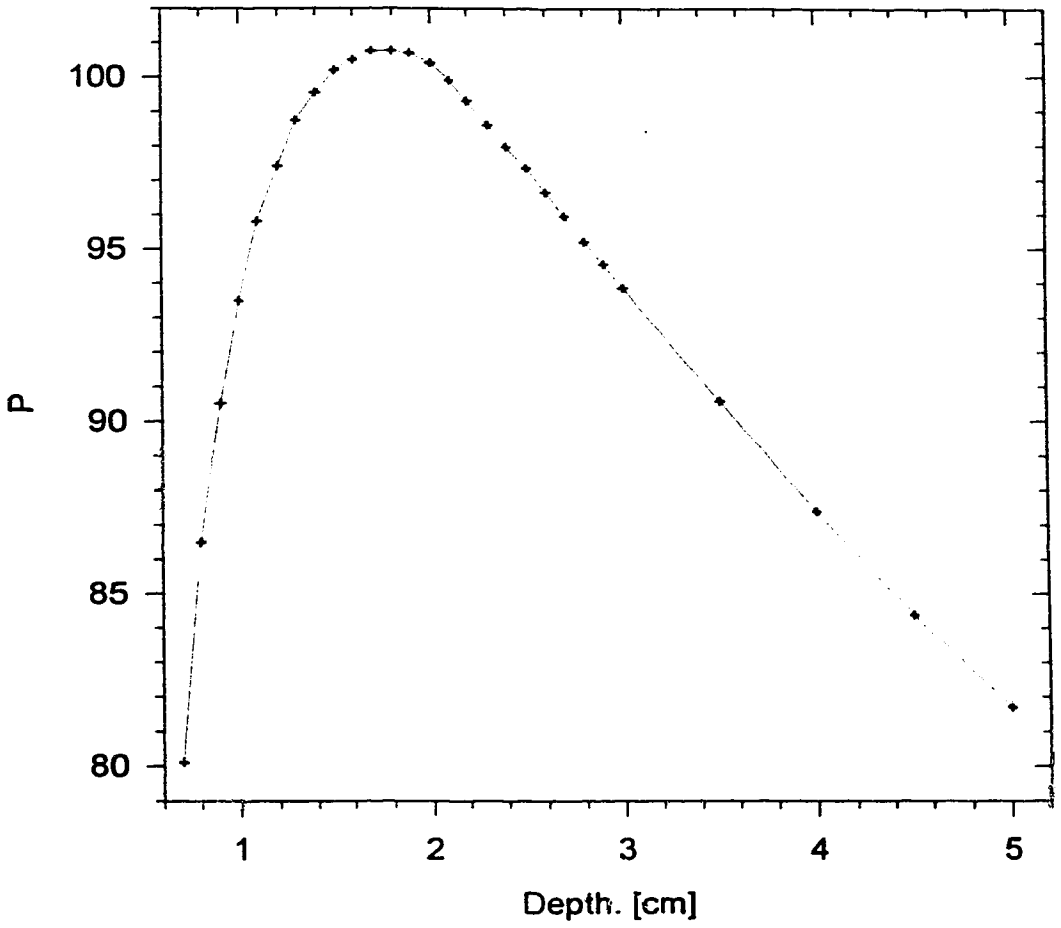


Figure 3.11a. The fitting parameter P.

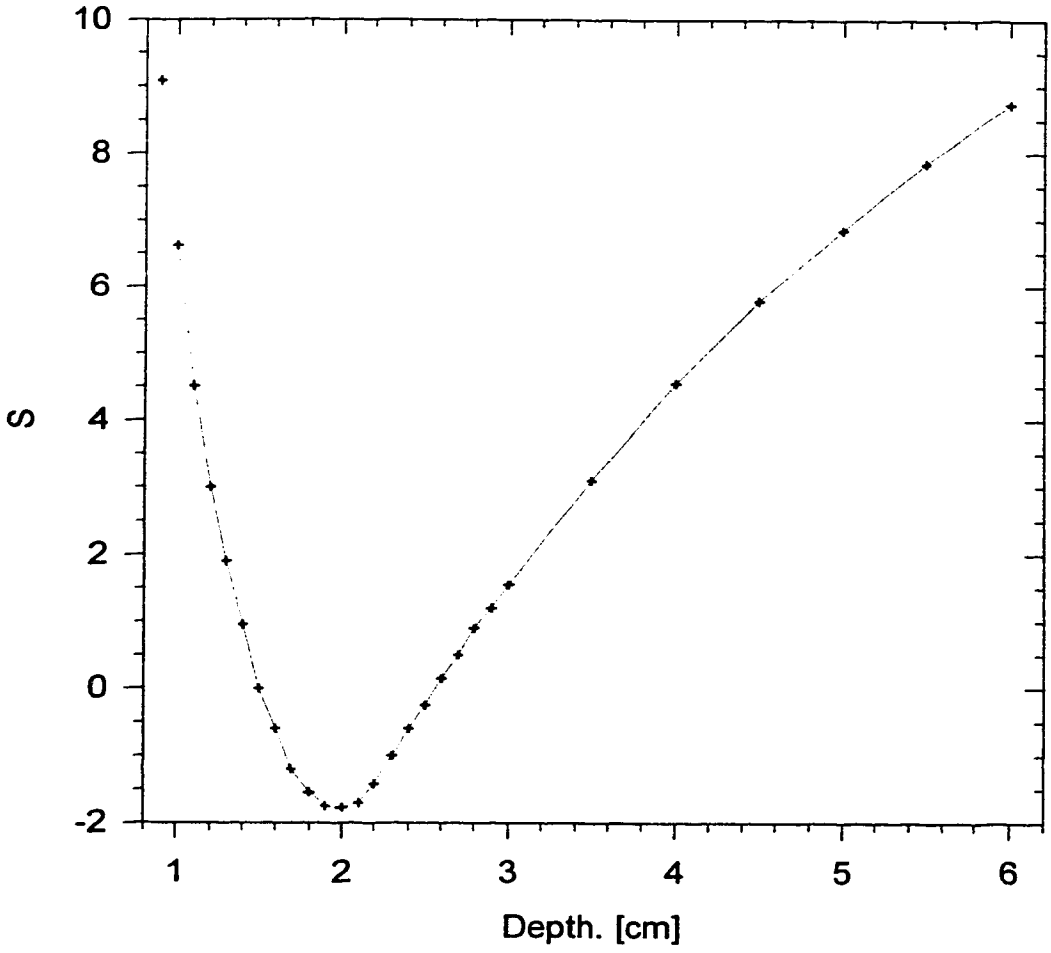


Figure 3.11b. The fitting parameter S.

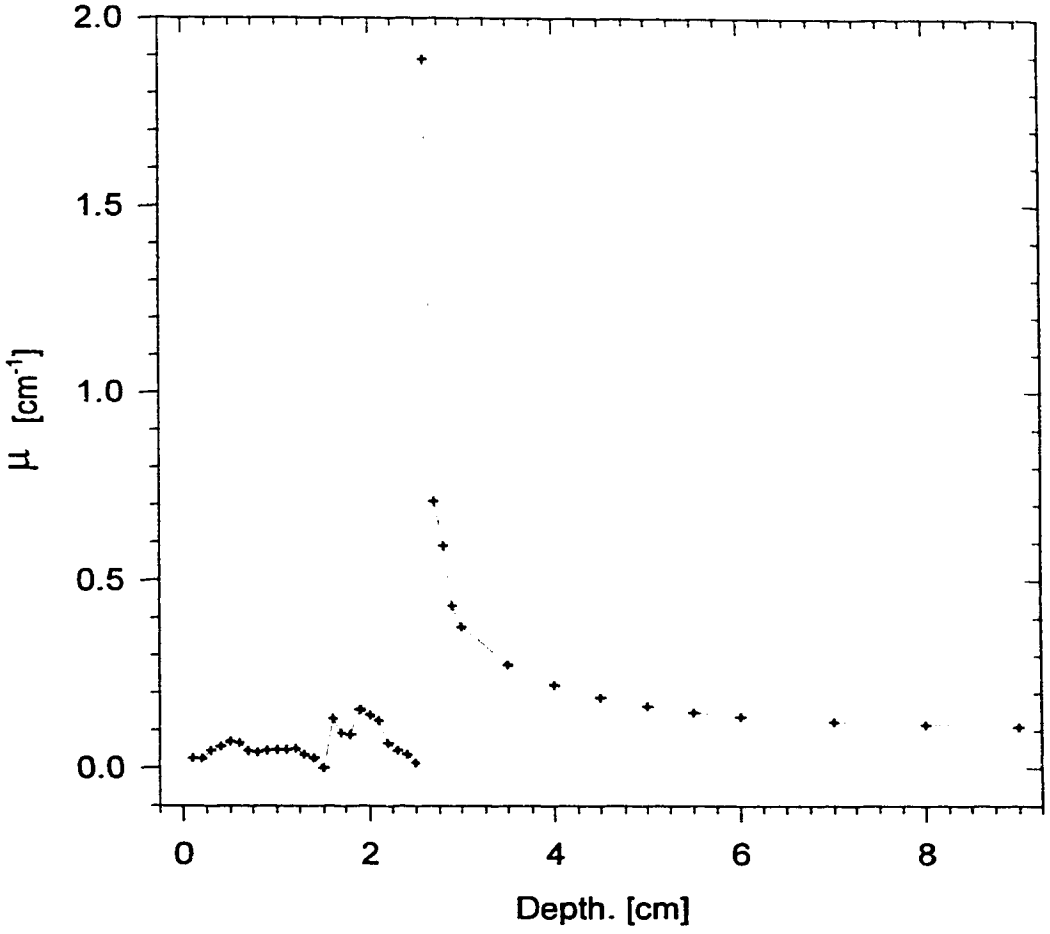


Figure 3.11c. The fitting parameter μ.

TAR data can now be constructed from the PSF and PDD data using the following formula [Burns 1983].

$$TAR(d,S) = \frac{1}{100} \cdot PDD\left(d, f, \frac{S \cdot f}{f+d}\right) \cdot PSF\left(\frac{S \cdot f}{f+d}\right) \cdot \left(\frac{f+d}{f+d_m}\right)^2 \qquad [3.5]$$

In the above formula, d is the depth of the point in question, S is the size of the square field, f is the source to surface distance, and d_m is the normalization point for the PDDs. Since the TAR is defined at the source-point distance (f+d), whereas the PDD and PSF have the field size defined at the source-surface distance f, the factor $S \cdot f/(f+d)$ is required to project the field size S back from the source-point distance to the source-surface distance.

The SARs are then constructed from the TARs using equation 3.6.

$$SAR(d,S) = TAR(d,S) - TAR(d,0) \qquad [3.6]$$

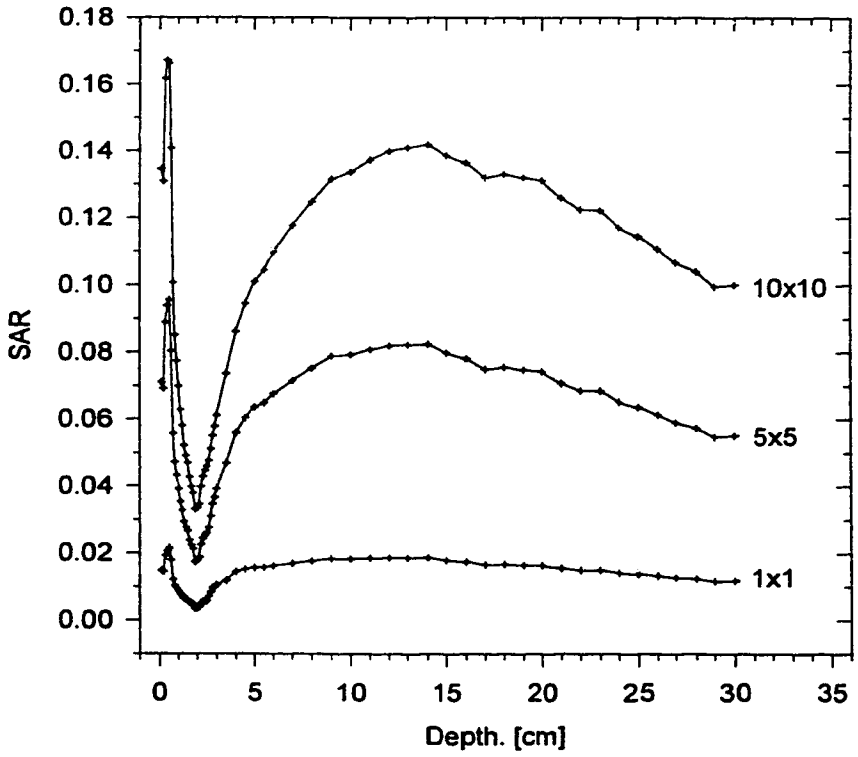


Figure 3.12. The SAR data obtained from equations 3.5 and 3.6.

Figure 3.12 shows the SAR curves for three field sizes. It shows an increase in the scatter contribution to dose from 2 cm depth up to about 13 cm depth, which is due to the fact that as the depth of the calculation point increases, the volume of forward scattering material above the calculation point also increases. As well, the beam spectrum softens with depth in the phantom, and the softer beam energies exhibit more scattering (i.e. they are less penetrating). The drop in scatter beyond 13 cm is due to the attenuation of the scattered photons that are produced at smaller depths, which now must travel through several centimeters of phantom material, as well as the fact that fewer scattered photons are produced at large depths due to this attenuation of the beam through the medium. That is, the effect of increase in scatter due to beam softening and increasing scatter volume above the calculation point eventually becomes dominated by the competing effect of decrease in scatter due to beam attenuation with depth.

The initial rise and fall of scatter below 2 cm depth is due to electron contamination in the beam. The first 0.5 cm shows a rise is scatter and is possibly due to an initial buildup of scattered electrons. The next 1.5 cm shows a drop in scatter, and this is likely due to the attenuation of the electrons with depth.

The data imported by c_photon, then, are SAR(d,S), TAR(d,0), rel_intensity(x,y) and data describing the machine characteristics. The c_photon program further constructs the function ΔSAR(r,d) using the formula:

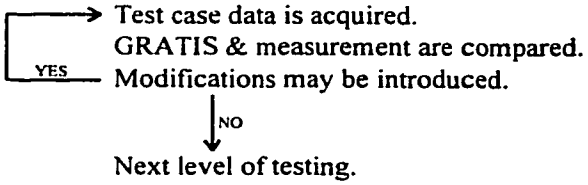
$$\Delta SAR(r_i, d) = SAR(d, S(r_i)) - SAR(d, S(r_{i-1})) \qquad [3.7]$$

Much of the data required by the algorithm is thus determined before the scatter integration begins, resulting in some savings in computation time.

A detailed description of this data and a derivation of the algorithm for dose computation is given in Appendix A.

4. Experimental evaluation of the algorithm

The strategy adopted for stress testing the c_photon program is to begin with symmetric square fields of various sizes (Section 4.1), then examine cases where one set of jaws is set asymmetrically (Section 4.2), and finally the dual asymmetric cases are studied (Section 4.3). Measurements are made for test situations on a Varian 2300CD linac[‡] and compared with simulations of these cases on GRATIS. Discrepancies between measurement and GRATIS are analyzed to uncover the shortcomings of the calculation method or the possible flaws in the implementation. In either case, if modifications to the algorithm are made then the test cases are repeated using the new code.



All in-phantom measurements were made using a 40×40×40 cm^3 water tank, equipped with a computer controlled scanning system[*] and a silicon diode[†] or an ion chamber[✱]. 6 MV photons were exclusively used, and no beam modifying devices were introduced (i.e. filters, compensators, and beam shaping blocks). Isodoses were acquired by scanning across the field at depths ranging from 0.00 cm up to 24.00 cm, with a scan spacing of 0.25 (±0.02) cm for depths smaller than 3.00 cm., and a spacing of 1.00 cm. for depths greater than 3.00 cm.

As a convention, the beam coordinate system is defined to have the source as it's origin, Y axis pointing toward the gantry and parallel to the axis of gantry rotation, Z axis pointing from the virtual source toward isocenter, and X axis pointing perpendicular to these two in such a way as to form a left handed coordinate system (See figure 4.1). This is the definition of "beam coordinates" described in the documentation for GRATIS. In this thesis any reference to the lower (upper) Y collimator refers to the Y collimator that has a negative (positive) Y position when a symmetric field is created. That is, the term "lower" will not refer to the distance of the collimators from the source, as it does in other literature. Similarly, the left (right) X collimator has a negative (positive) X position when a symmetric field is created.

---

‡ The 2300CD is computer controlled with dynamic wedge capability, 2 photon energies, 6 electron energies and a multileaf collimator.

* A Wellhöfer scanning system was used, which allows computer controlled positioning of a diode or ion chamber anywhere within the 40x40x40 cm^3 watertank.

† Scanditronix p-type silicon diode. Thickness of silicon chip = 0.45 ±0.02 mm. Detector diameter = 2.5±0.1 mm.

✱ Wellhöfer IC-10 ion chamber. Spherical volume of 0.1 cm^3.

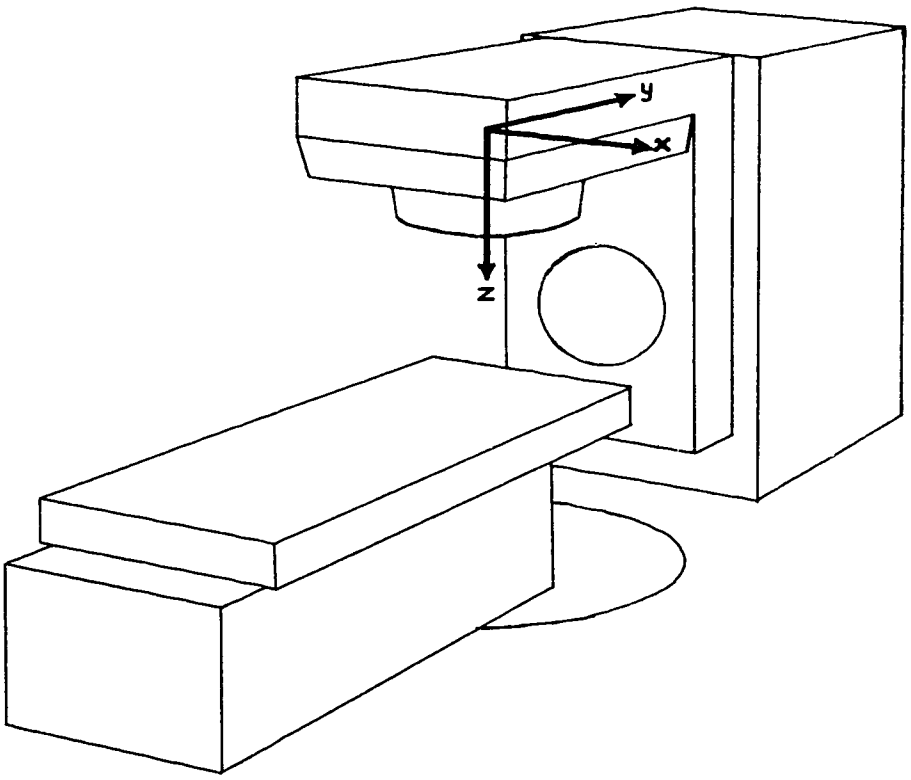


Figure 4.1. The definition of beam coordinates.

Also as a convention, an asymmetric rectangular field will be described using the following notation:

10x15, X=0 Y=2.5, YZ-iso X=5

This describes the resulting field when, starting with a symmetric 10x20 cm^2 field, the lower Y collimator is moved 5 cm toward the central axis. That is, the resulting field is now 10x15 cm^2, but is displaced in the Y direction by 2.5 cm from the symmetric 10x15 cm^2 case. The notation also indicates that isodoses were obtained in a plane that is parallel to the central axis (Z-axis) and to the axis about which the gantry rotates (Y axis), and that the plane is at a distance of 5 cm from the central axis.

The term "old GRATIS" used in the titles of figures 4.2 indicates that the isodoses were generated using the unmodified copy of GRATIS obtained from Dr. Sherouse.

4.1. Symmetric Fields

The first set of cases acquired were square fields centered on the central axis. The water surface was adjusted to be 100.0 cm from the source, and the following isodoses were acquired:

4x4, X=0 Y=0, YZ-iso X=0
10x10, X=0 Y=0, YZ-iso X=0
30x30, X=0 Y=0, YZ-iso X=0

The corresponding configurations were simulated on GRATIS, and the isodose plots for both GRATIS and measurement were overlaid[†] , as shown in figures 4.2a through 4.2c. Isodose values and dose discrepancies are expressed as percentages of the dose at a reference point, which for the fields discussed in this section is at 1.5 cm depth on the central axis.

It can be seen that the small field (figure 4.2a) agrees reasonably well along the central axis. The 95% isodose lines, for instance, represent a maximum difference of about 1% on the central axis, and this is considered an acceptable match.

The discrepancy near the field edges, however, suggests that there is a problem with the penumbra model being used by GRATIS. This model is a geometric penumbra that is calculated using an extended source distribution [Johns et. al. 1983]. The source distribution being used is a relatively simple one that does not accurately describe the radiation produced in the head of a linear accelerator, and hence the introduction of a better source distribution will improve the agreement near the field edges. A Monte Carlo analysis of the effects of the source distribution on the penumbra was performed by Chaney and Cullip [Chaney et. al. 1994], and it was found that the addition of a Gaussian term was a significant improvement to the negative exponential being used in some treatment planning systems (and currently being used in GRATIS). However, modifications to the penumbral model of GRATIS will be avoided at this point, so that a wider range of field cases can be evaluated and more immediate problems dealt with first.

Figure 4.2c immediately indicates a problem with more than just the penumbra model. The 95% isodose lines, for instance, show a discrepancy of about 2.5% on the central axis, and this discrepancy increases to 5% at 10cm away from the central axis. This is not due to the penumbra model since the penumbral effects do not extend all the way to the central axis, as can be seen in figure 4.2d where the penumbra has been "turned off" in order to see its effect on the calculated 30x30 isodoses. Rather, it seems that the dose contribution due to scattered radiation is being underestimated by the calculation algorithm, and this underestimation is field size dependent since the discrepancy gets worse the larger the field.

[†] The contour plotting features of the MATLAB software package was used to construct the plots shown in the following sections.

Figure 4.2a. 4x4, X=0 Y=0, YZ-iso X=0, old GRATIS.

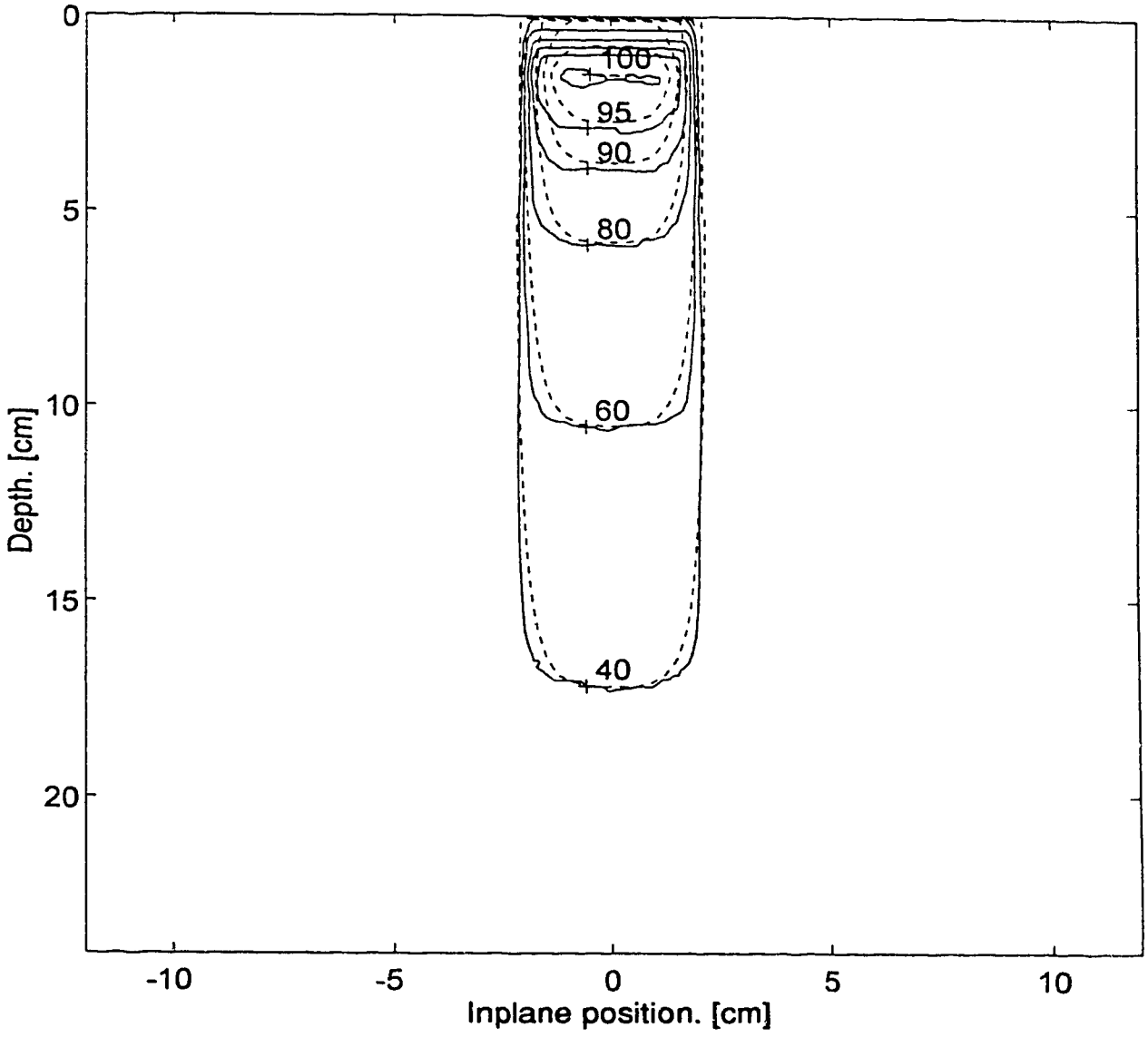


Figure 4.2b. 10x10, X=0 Y=0, YZ-iso X=0, old GRATIS.

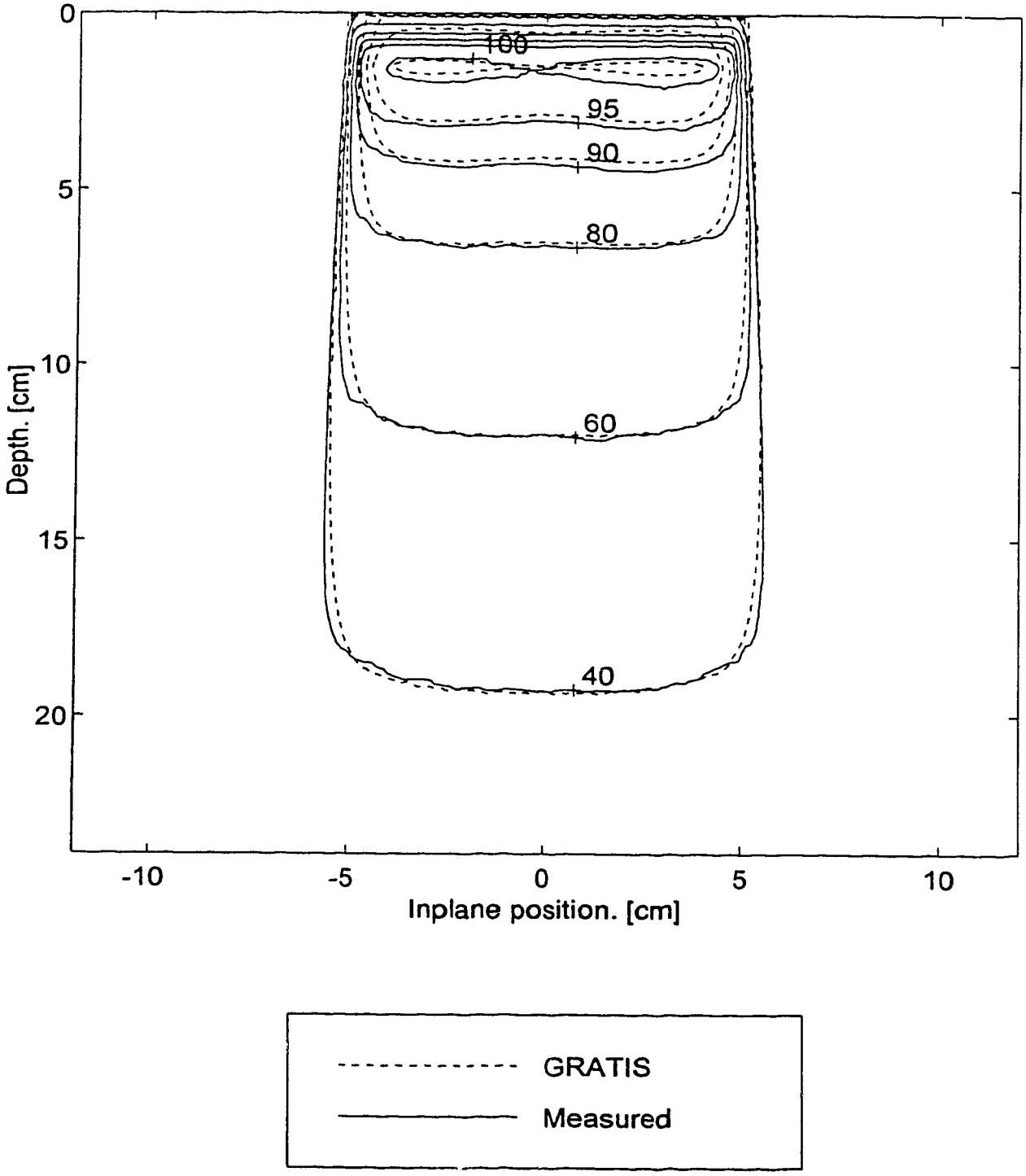


Figure 4.2c. 30x30, X=0 Y=0, YZ-iso X=0, old GRATIS.

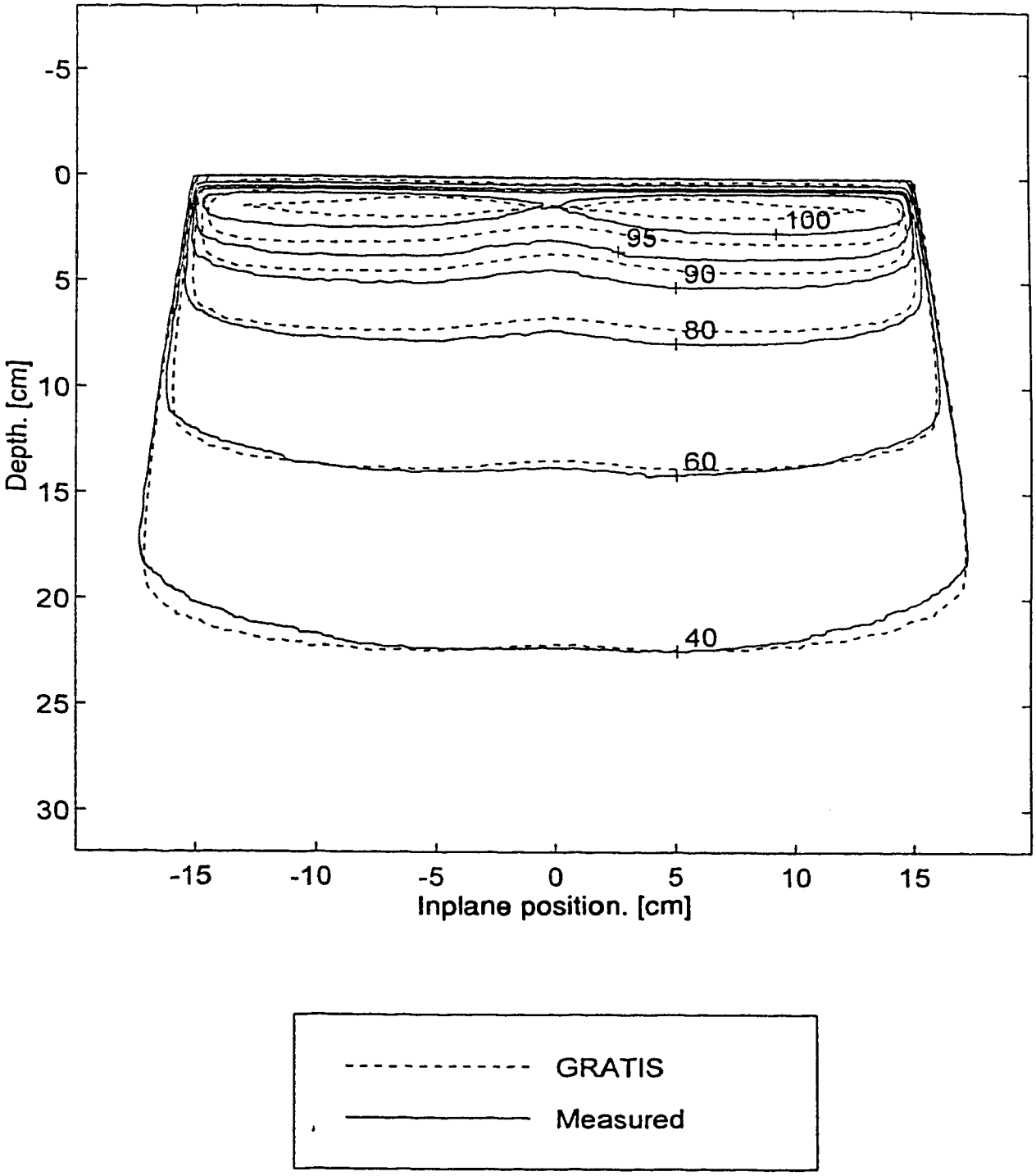
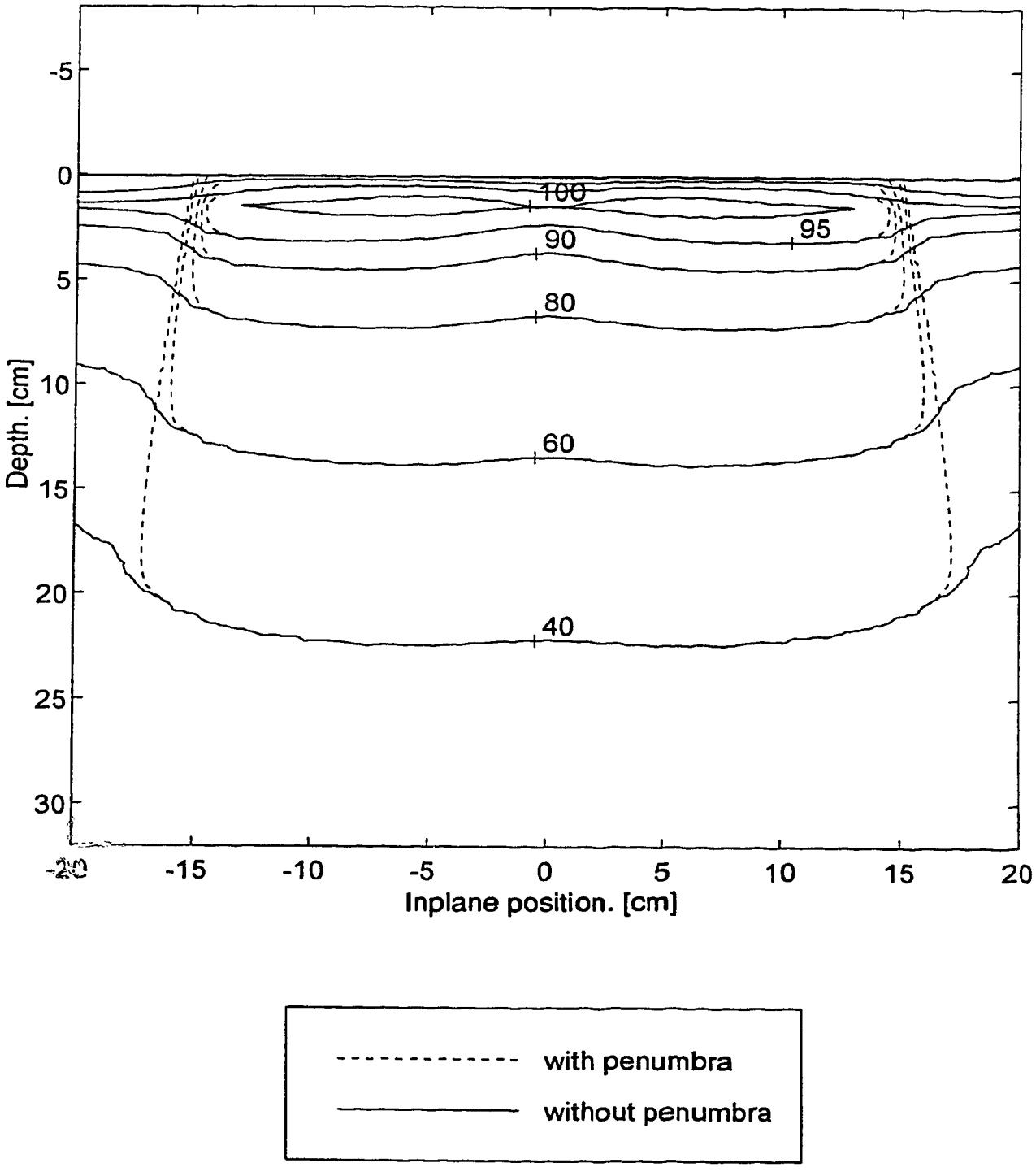


Figure 4.2d. 30x30, X=0 Y=0, YZ-iso X=0, old GRATIS, penumbra test.



An examination of the code of GRATIS revealed that the depth and radius being used to look up the dSAR's was not consistent with the way the dSAR table was actually constructed. The depth and radius that were being used in the original GRATIS are shown in figure 4.3 as depth d_1 and radius r_1. However, d_3 and r_3 are the ones that most accurately represent the parameters used in the dSAR table. d_2 and r_2 are presented in figure 4.3 because one might initially expect **them** to be the most consistent with the dSAR table, but closer examination reveals that, in fact, d_3 and r_3 are the correct parameters to use for dSAR lookup.

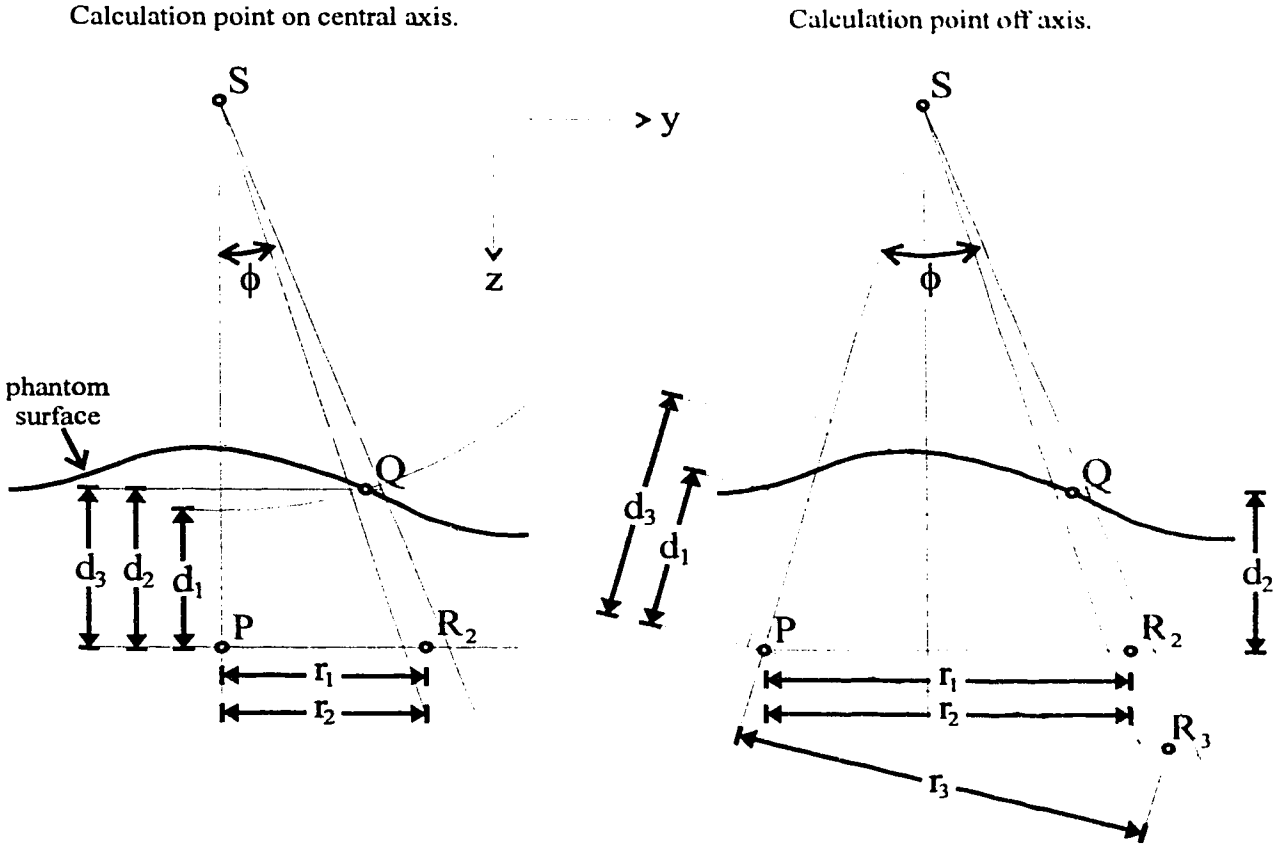


Figure 4.3. Three choices for the depth used in the dSAR lookup.

$$d_1 = \|SP\| - \|SQ\| \qquad r_1 = \|PR_2\|$$
$$d_2 = SP_z - SQ_z \qquad r_2 = \|PR_2\|$$
$$d_3 = \|SP\| - \|SQ\| \cdot \cos(\phi) \qquad r_3 = \|PR_3\|$$

To illustrate why d_3 and r_3 are the correct parameters, consider how the dSAR table (or, equivalently, the TAR table) is constructed from measured data (c.f. section 3.2). The TAR's are determined for a range of depths and for circular fields of various radii, but there is no requirement that these circular fields be centered on the central axis. In principle, the TAR's for circular fields that are centered on some other rayline (figure 4.4) could also be determined. Assuming that the energy spectrum does not change significantly with increasing distance from the central axis, the resulting TAR data would be the same as would be obtained for fields centered on the central axis[†].

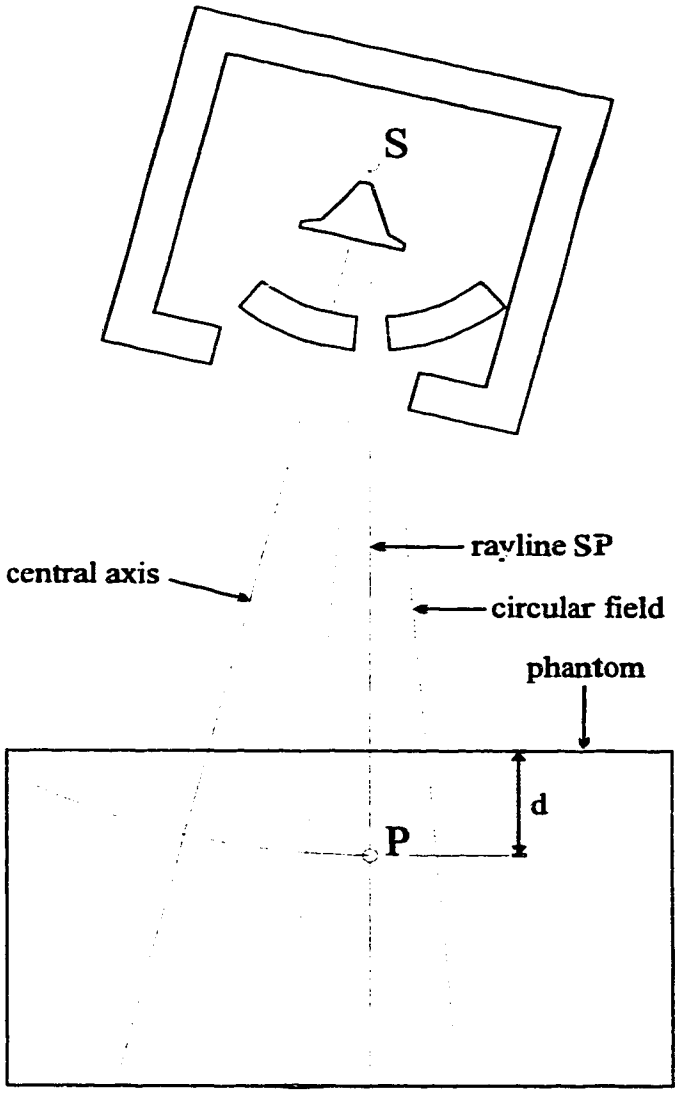


Figure 4.4. Off-axis measurement of TAR's.

[†] In section 4.3 the effect of change in energy spectrum on TAR's measured off-axis will be investigated.

It is apparent from figure 4.4 that the depth for such a TAR is not taken along the central axis, but rather it is defined along the rayline SP. Thus, for off-axis calculation points, it is this rayline depth that should be used when looking up TAR data from a stored table; or, equivalently, when looking up the dSAR data during the scatter integration. In figure 4.3 this rayline depth is just d_3.

As an alternative argument for using the d_3 and r_3 to look up the dSAR data during the scatter integration, consider what would happen if d_2 and r_2 were used to look up the dSAR values during the scatter integration.
At the time of the dSAR table construction, a pencil with depth d and radius r has the geometry shown in figure 4.5a, and also in figure 4.5b with only the relevant information drawn. This pencil follows a rayline that begins at the source, and hence the primary radiation incident upon the top of the pencil will travel down the length of the pencil. Thus, the radiation intensity at the top of the pencil can be used to weight the scatter contribution to the total dose.
However, consider the off-axis point shown in figure 4.5c. Using the same depth d and radius r (i.e. the same pencil geometry) as was used for the central axis point will result in a pencil that does not follow along a ray through the source. Thus the radiation incident upon the pencil surface does not travel down the length of the pencil, and hence cannot be used to weight the pencil.
A beam blocking device, for instance, can be used to make the surface fluence zero[†], whereas most of the pencil is in fact still being irradiated. In this case the radiation intensity at the surface of the pencil (i.e. zero) is obviously the wrong weighting to use in the scatter integration. It is unclear what weighting to use in this case.
If a different pencil geometry (i.e. a different d and r) is chosen so that the pencil is aligned along a rayline from the source (figures 4.5d through 4.5f) then this problem no longer occurs, since zero primary at the surface entails zero primary along the entire pencil. Thus it is clear that d_3 and r_3 are more suitable than d_2 and r_2.

It is not difficult to show that d_1 and r_1 are not suitable for dSAR lookup since d_1 does not measure a distance along the same rayline, and at measurement time the TAR depths are along the same rayline. In figure 4.3 for the central axis point (left diagram) it is clear that d_1 underestimates the depth (d_3) for which the dSAR table was constructed. This underestimation will be worse for scatter pencils that are further away from the central axis, and hence introduce larger errors for larger fields; and this is what is being seen in figures 4.2a through 4.2c.

[†] Actually the fluence under a blocking device will be nonzero, but the value can be made to be very small compared to the fluence in the unblocked part of the field.

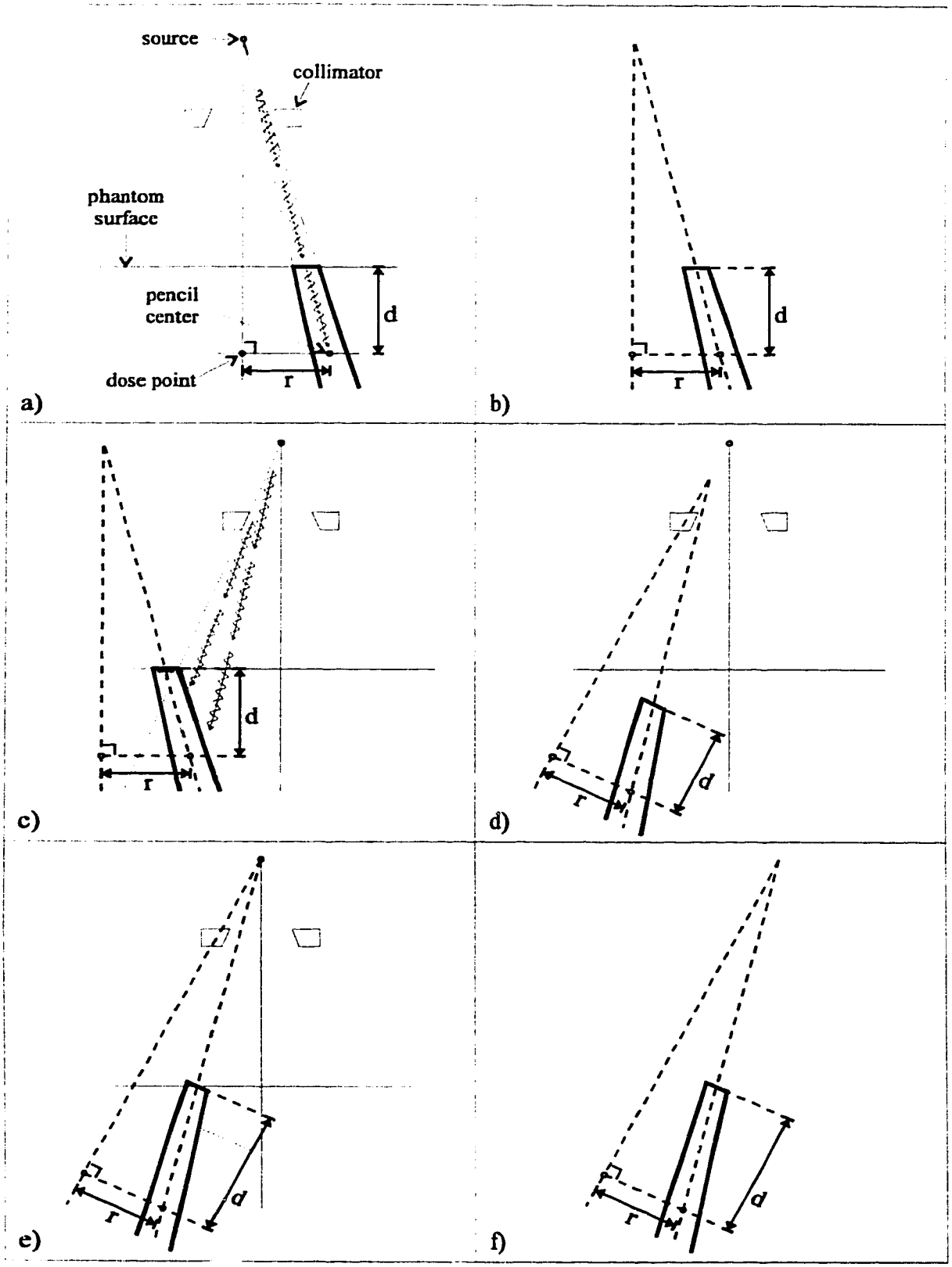


Figure 4.5. The pencil alignment problem.

Having determined why the current depth and radius may be responsible for the discrepancies on the central axis for large symmetric fields, some changes were introduced into the GRATIS code. Recall from section 3.2 the pseudocode that describes the scatter integration.

$$P = \sqrt{x_p^2 + y_p^2 + z_p^2}$$
$$SAR = 0$$
for $r = 0$ to r_{max} do
  for $\theta = 0$ to $360°$ do
$$(x_c, y_c, \phi) = F(x_p, y_p, z_p, r, \theta) \qquad [3.3a]$$
$$d = P - SSD(x_c, y_c) \cdot \cos(\phi) \qquad [3.3b]$$
$$SAR = SAR + rel\_intensity(x_c, y_c) \cdot \Delta SAR(r, d) \qquad [3.3c]$$
  end for
end for

- (x_p,y_p,z_p) are the coordinates of the calculation point relative to the source,
- P is the source-to-calculation-point distance,
- r and θ are the polar coordinates of the pencil relative to the calculation point,
- ϕ is the angle subtending the source-point rayline and the source-pencil rayline,
- d is the "depth" or height of the pencil above the calculation point,
- (x_c,y_c) are the coordinates of the top of the pencil, projected back to a reference plane,
- and F represents the necessary coordinate transforms to obtain (x_c,y_c) and ϕ.

The coordinate transform F takes the following form† in the original GRATIS.

define $(x_c, y_c, \phi) = F(x_p, y_p, z_p, r, \theta)$
begin
$$\bar{a} = \bar{p} + \bar{r} \qquad [4.1a]$$
$$\bar{c} = \bar{a} \cdot 100/z_a \qquad [4.1b]$$
$$\phi = 0 \qquad [4.1c]$$
end

- $\bar{r} = (r \cdot \cos(\theta),\ r \cdot \sin(\theta),\ 0)$ is the position of the pencil relative to $\bar{p}$.
- $\bar{a} = (x_a, y_a, z_a)$ is the center position of the pencil.
- $\bar{p} = (x_p, y_p, z_p)$ is the position of the dose point.
- $\bar{c} = (x_c, y_c, z_c)$ is the intersection of the source-pencil rayline with a reference plane at a distance z_c from the source.

† The actual C-language code is given in Appendix B.

The statement 4.1a introduces a "shifted dartboard", as shown in figure 4.6. This is inconsistent with the dSAR table construction since it uses the radius in the geometry shown in figure 4.5c (i.e. r_1 and r_2). The statement 4.1c also introduces an inconsistency since it causes the statement

$$d = P - SSD(x_c, y_c) \cdot \cos(\phi) \qquad [3.3b]$$

to become

$$d = P - SSD(x_c, y_c) \qquad [4.2]$$

and this is just the definition of d_1.

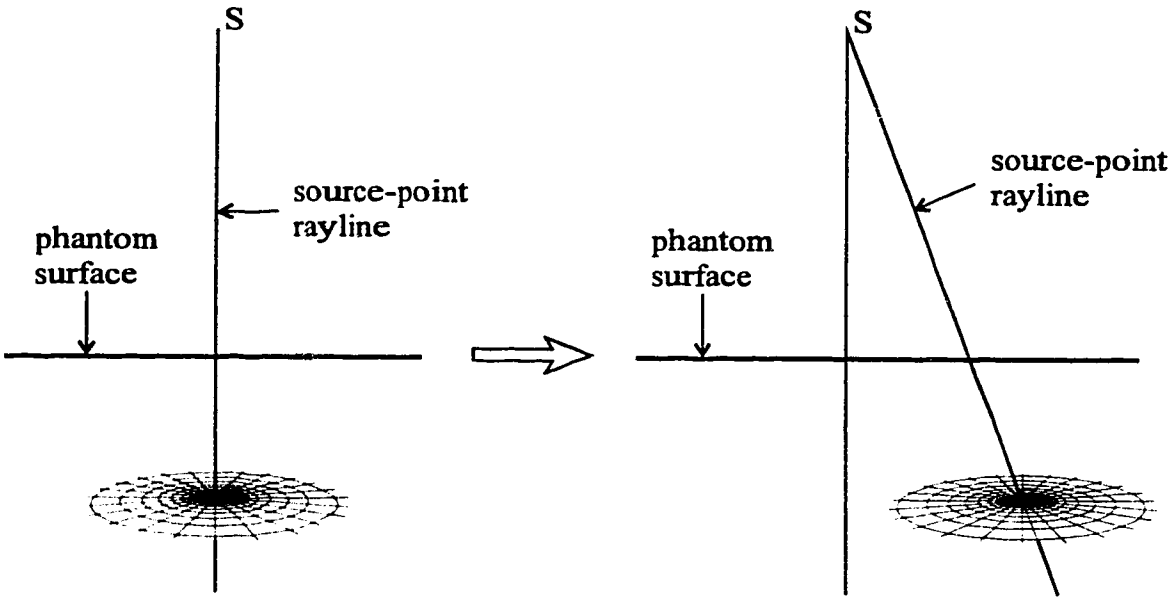


Figure 4.6. Shifting of the dartboard.

These statements (4.1a and 4.1c) must be replaced by the appropriate code† that will introduce r_3 and d_3 (c.f. equations 4.3).

define $(x_c, y_c, \phi) = F(x_p, y_p, z_p, r, \theta)$
begin

$$\alpha = \arccos(z_p / \|\bar{p}\|) \qquad [4.3a]$$

$$\beta = \arctan(y_p / x_p) \qquad [4.3b]$$

$$M = R_z(\beta) \cdot R_y(\alpha) \qquad [4.3c]$$

$$\bar{a} = \bar{p} + M \cdot \bar{r} \qquad [4.3d]$$

$$\bar{c} = \bar{a} \cdot 100 / z_a \qquad [4.3e]$$

$$\phi = \arctan(\|\bar{r}\| / \|\bar{p}\|) \qquad [4.3f]$$

end

where the following notation has been introduced.

α and β are two angles describing the position of the dose point $\bar{p}$,

M is a matrix that tilts the dartboard,

$$R_z(\beta) = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ is a positive rotation about the z-axis,

$$R_y(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$ is a positive rotation about the y-axis.

The matrix M constructed in statements 4.3a through 4.3c and used in 4.3d will create a "tilted dartboard", forcing the magnitude $\| \bar{r} \|$ to represent r_3, the desired radius. Also, the statement 4.3f will cause the depth (equation 3.3b) to be d_3. The effect of the equations 4.3 on the dartboard geometry is shown in figure 4.7 for comparison with figure 4.6.

The effect on the isodoses introduced in figures 4.2a through 4.2c can now be seen in figures 4.8a through 4.8c. Note that for the large field size (figure 4.8c) the agreement on the central axis is dramatically improved over figure 4.2c. It can also be seen, however, that the agreement off-axis has become worse for large depths. This may be due, at least in part, to a problem with the penumbra calculation, as was indicated earlier in reference to figure 4.2d. It may also partly be due to the softening of the energy spectrum with distance from the central axis. This is an effect that will be addressed in section 4.3.

† The actual C-language code is given in Appendix B.

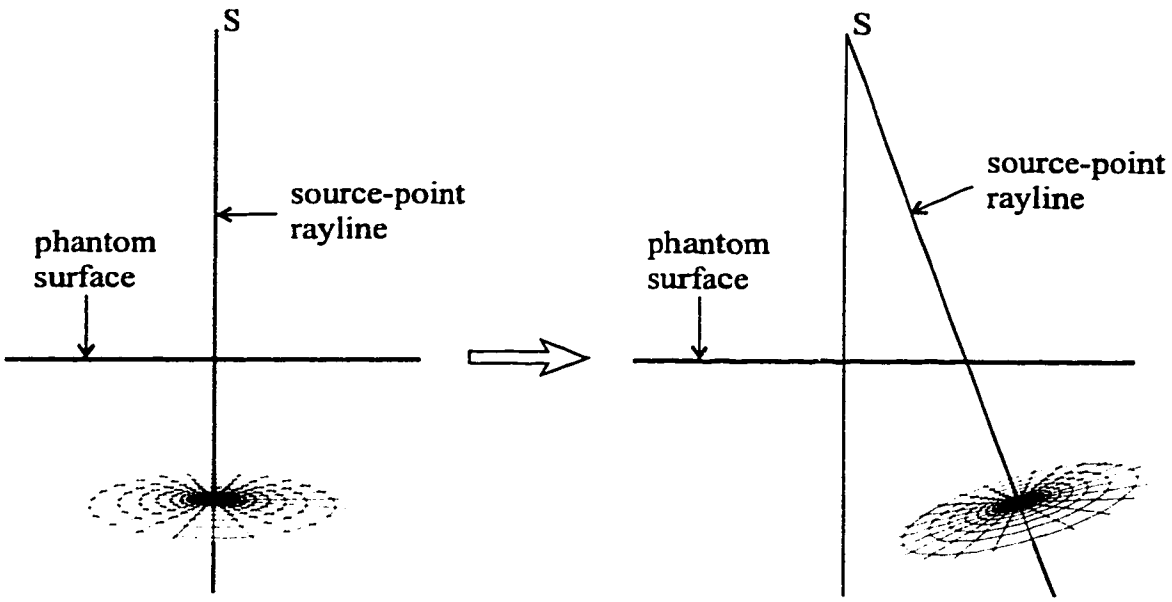


Figure 4.7. Tilting the dartboard.

Since the discrepancy between GRATIS and measurement is less than 1% in the dose region near the central axis, we conclude that, except possibly for the penumbra calculation and beam softening effects, the algorithm is performing acceptably. We thus proceed to test cases involving single asymmetric collimators.

The data acquired for off-axis fields should offer more support for (or refute) the hypothesis that beam softening has a significant effect on the isodoses. Also, the penumbra calculation is independent of the scatter integration method. A different penumbra model can be developed later, if we are satisfied that the scatter integration is performing adequately. Thus the analysis of both effects is deferred until the scatter integration has been investigated more thoroughly.

Figure 4.8a. 4x4, X=0 Y=0, YZ-iso X=0, tilted dartboard.

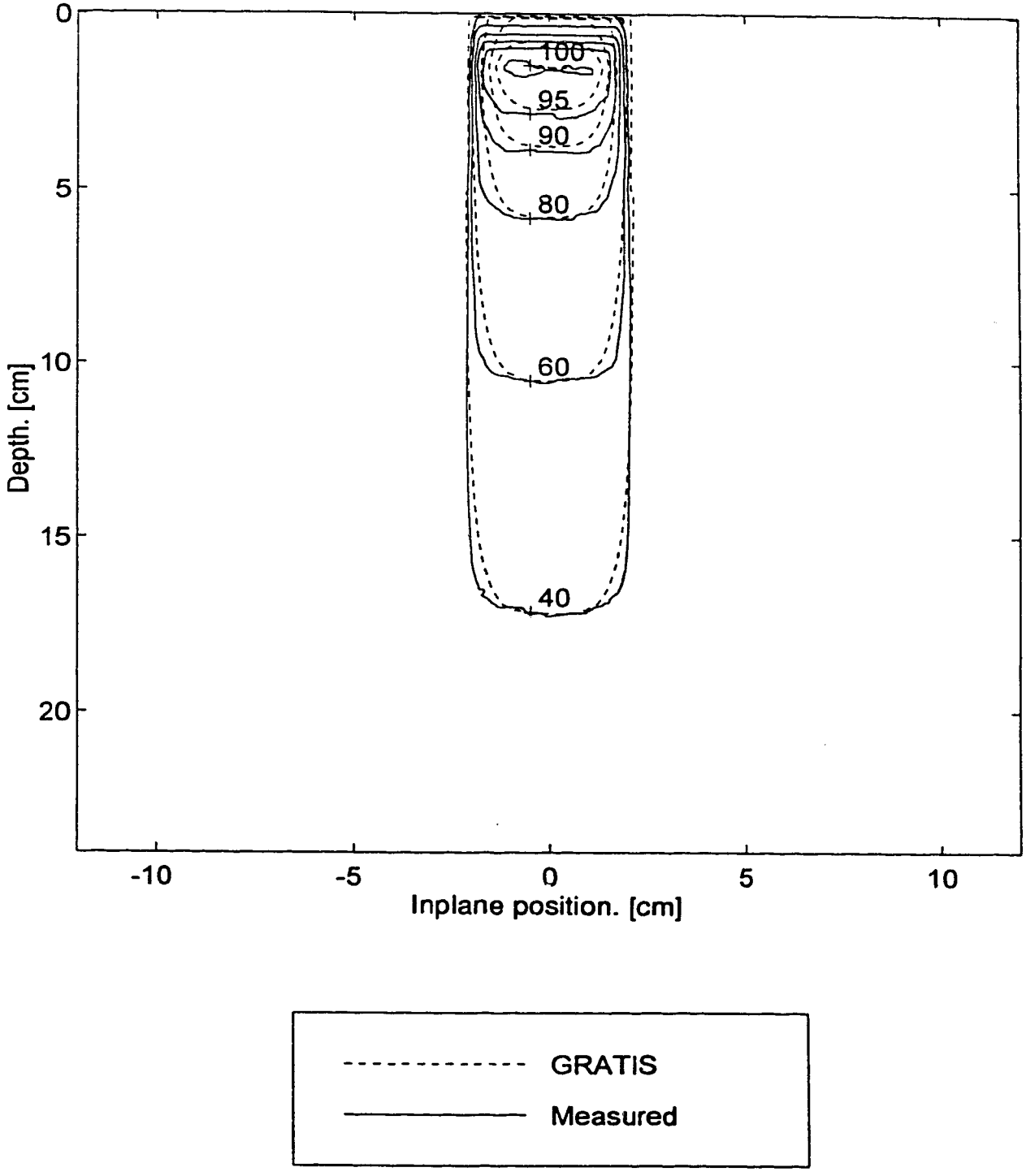


Figure 4.8b. 10x10, X=0 Y=0, YZ-iso X=0, tilted dartboard.

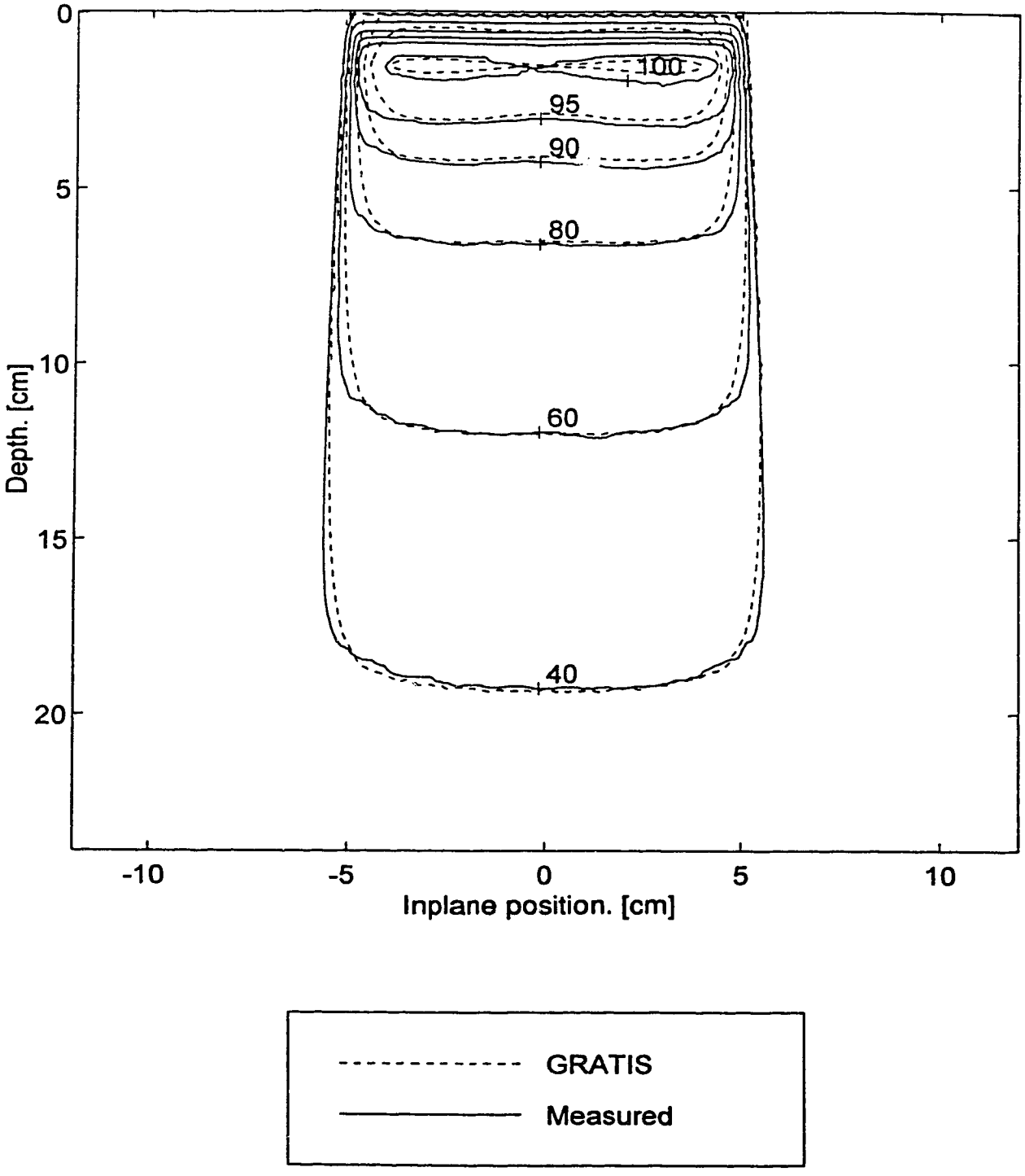
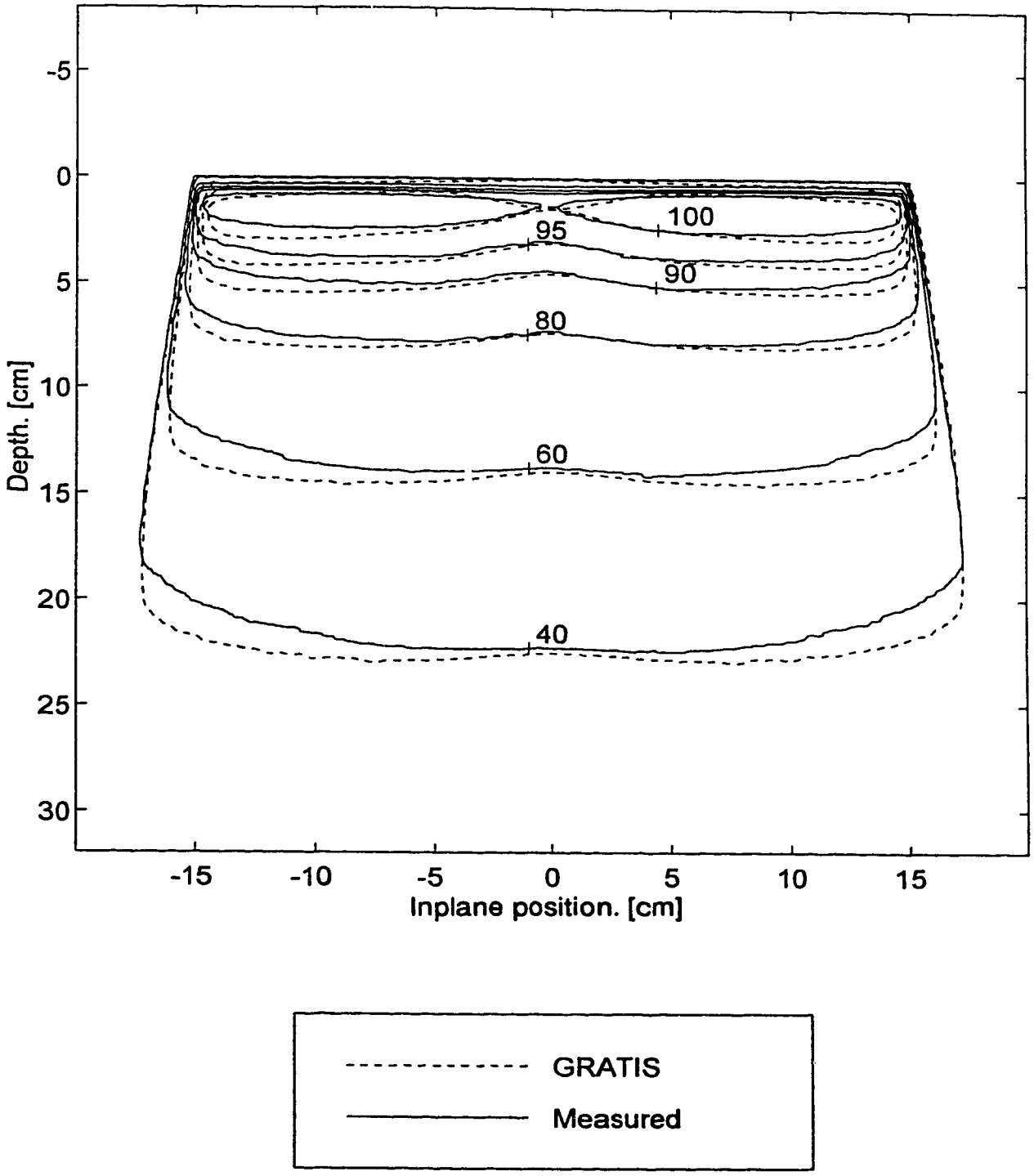


Figure 4.8c. 30x30, X=0 Y=0, YZ-iso X=0, tilted dartboard.



4.2. Single Asymmetric Fields

Single asymmetric fields were investigated by setting up a symmetric 10x20 cm^2 field (10.0 cm in the X direction and 20.0 cm in the Y direction) and then reducing the field length by moving a single Y collimator toward (and across) the central axis. By doing this the effect of one collimator on the isodose distribution can be investigated, since the other three collimators are held fixed and their scattering effect should remain constant. Isodoses were measured in the YZ plane for various positions of the Y collimator. The following isodoses are presented in figures 4.9a through 4.9g.

10x5, X=0 Y=-7.5, YZ-iso X=0
10x10, X=0 Y=-5, YZ-iso X=0
10x15, X=0 Y=-2.5, YZ-iso X=0
10x20, X=0 Y=0, YZ-iso X=0
10x15, X=0 Y=2.5, YZ-iso X=0
10x10, X=0 Y=5, YZ-iso X=0
10x5, X=0 Y=7.5, YZ-iso X=0

Each is normalized so that GRATIS agrees with the measured data at a point 7.5 cm away from the central axis in the Y-direction, and 1.5 cm below the water surface. This point was chosen since it was used to normalize the measured data for **all** fields so that 100% is at 1.5 cm depth on the central axis in the 10x20 field. That is, a common reference point inside all fields was needed in order that the dose values in one field could be related to the dose values in another field. Diode readings were accumulated for 100 monitor units at that reference point in each field configuration, and these readings served to relate the dose values from each field to the 10x20 cm^2 field.

It can readily be noticed that there is a large discrepancy between the measured and calculated isodoses in figures 4.9a and 4.9b. We would expect that the GRATIS isodoses in figure 4.9a should be mirror images of the isodoses in figure 4.9g, since the beam geometries are just mirror images of each other. Likewise for figures 4.9b and 4.9f. Also, the GRATIS isodoses in figures 4.9a and 4.9b are too "flat". They resemble the distribution which would be obtained if scatter dose was subtracted off the total, leaving only the primary dose (i.e. the TAR_o). In fact, plotting the primary isodoses produces distributions identical to the GRATIS isodoses shown in figures 4.9a and 4.9b. Plots of the scatter isodoses also verified that the scatter contribution to the total dose was zero for cases 4.9a and 4.9b.

Figure 4.9a. 10x5, X=0 Y=-7.5, YZ-iso X=0, tilted dartboard.

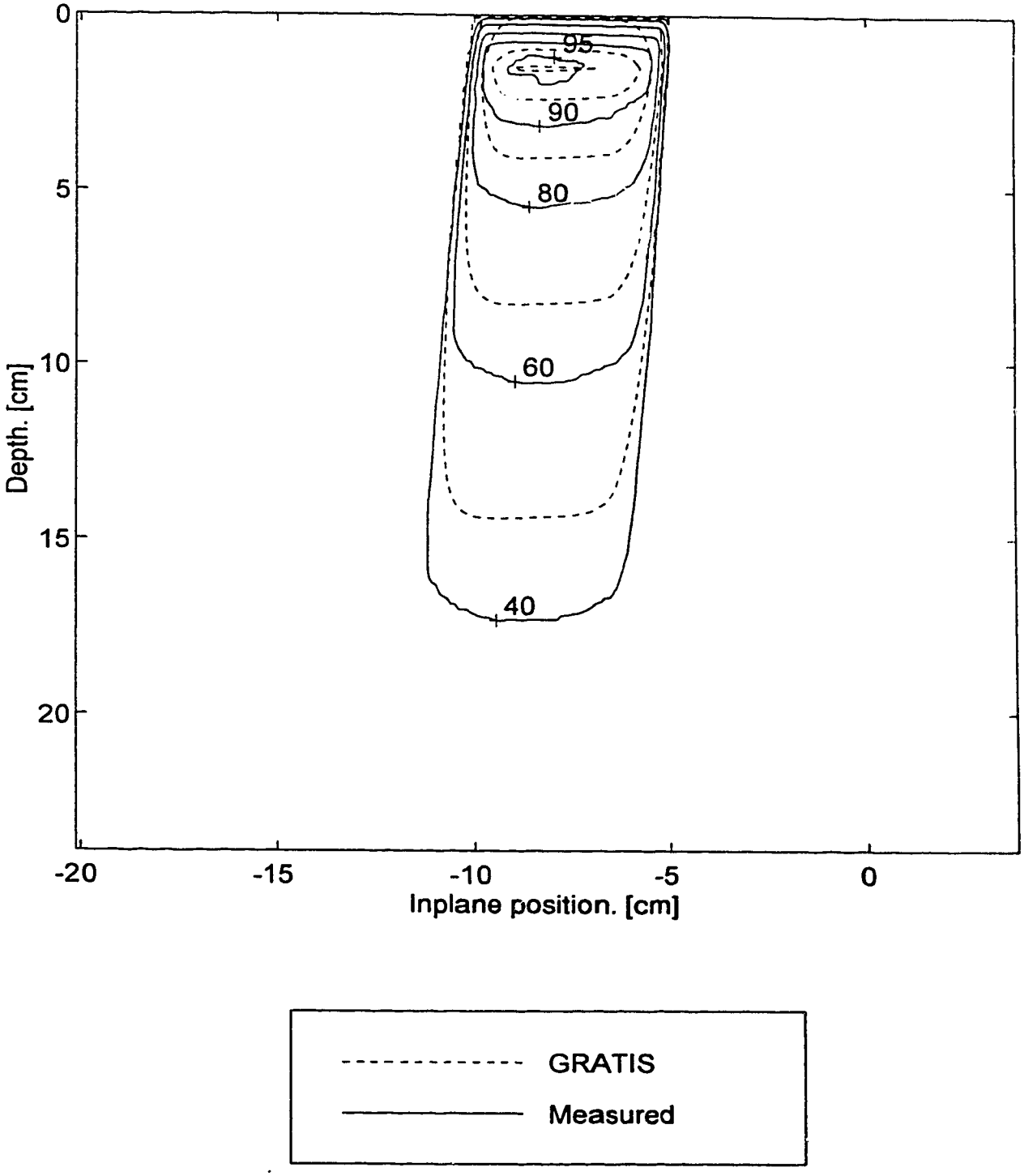


Figure 4.9b. 10x10, X=0 Y=-5, YZ-iso X=0, tilted dartboard.

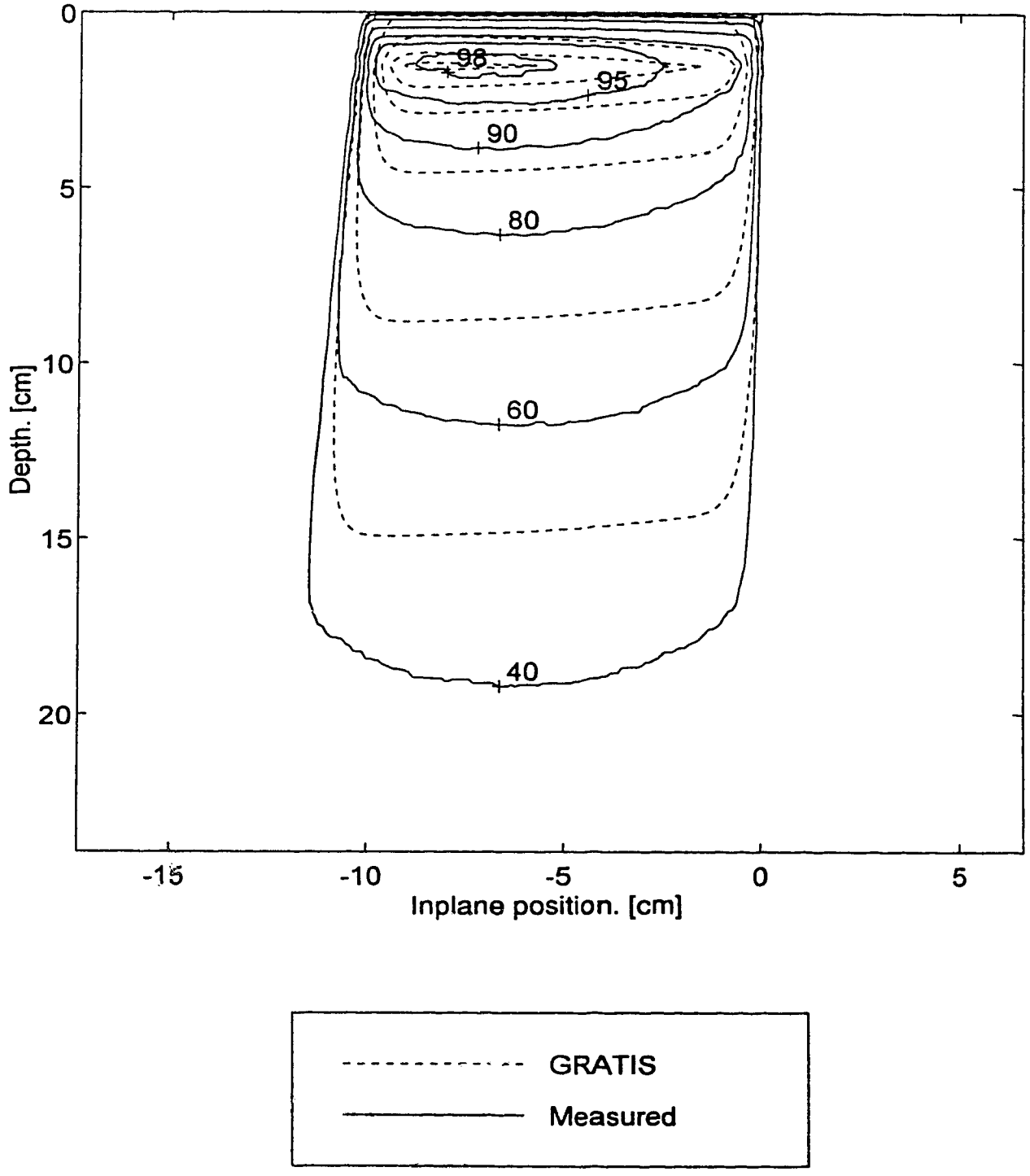


Figure 4.9c. 10x15, X=0 Y=-2.5, YZ-iso X=0, tilted dartboard.

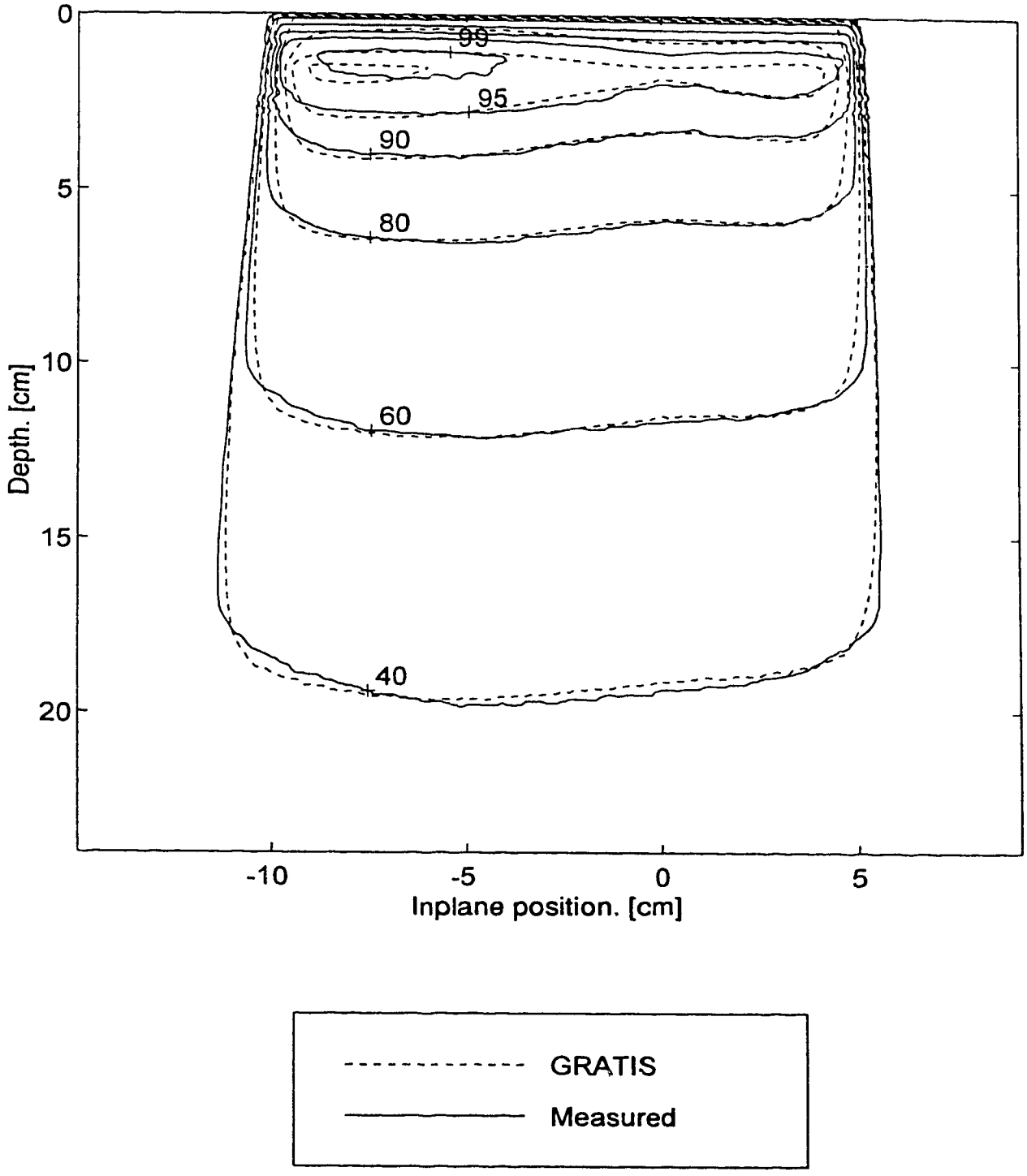


Figure 4.9d. 10x20, X=0 Y=0, YZ-iso X=0, tilted dartboard.

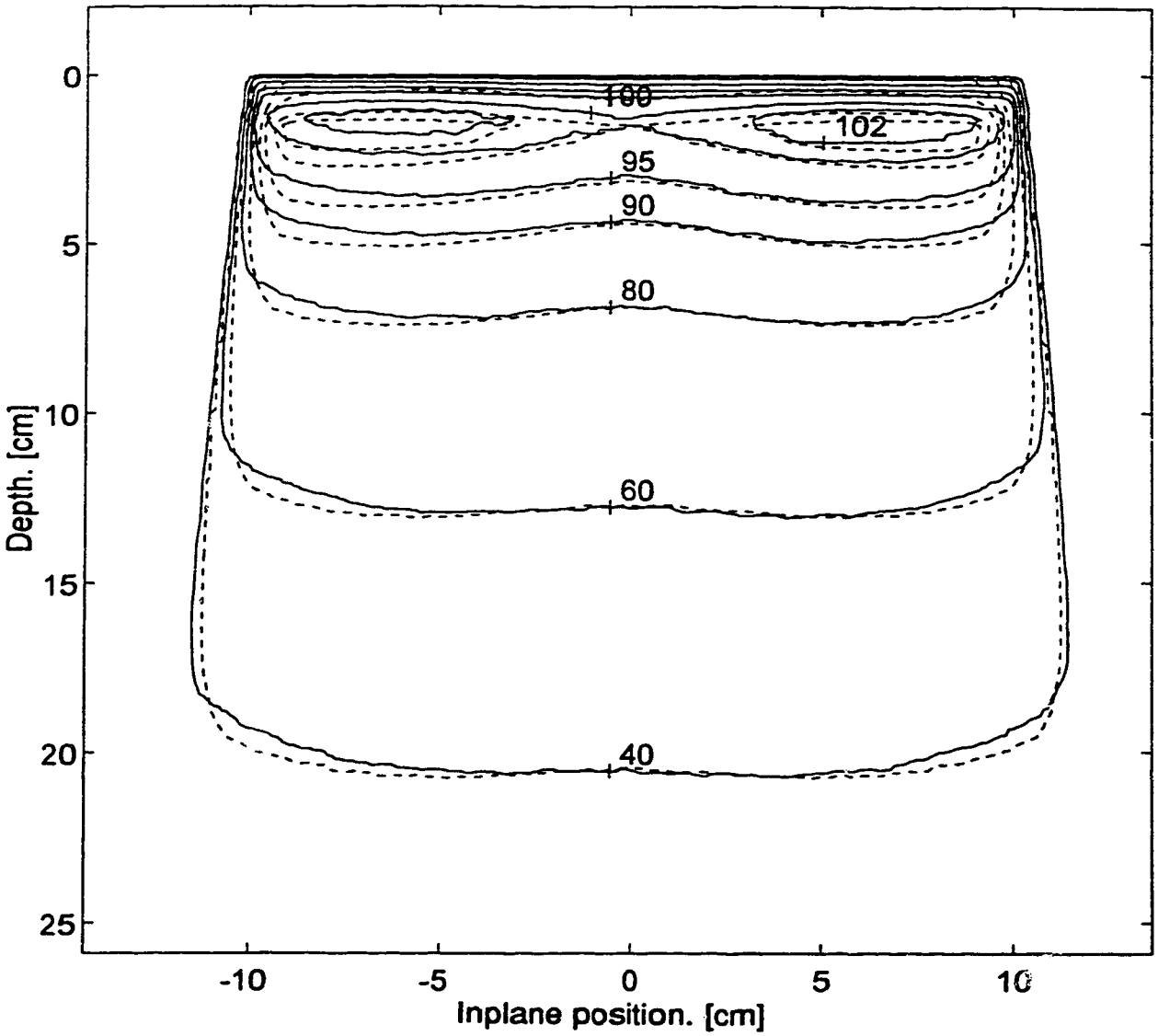


Figure 4.9e. 10x15, X=0 Y=2.5, YZ-iso X=0, tilted dartboard.

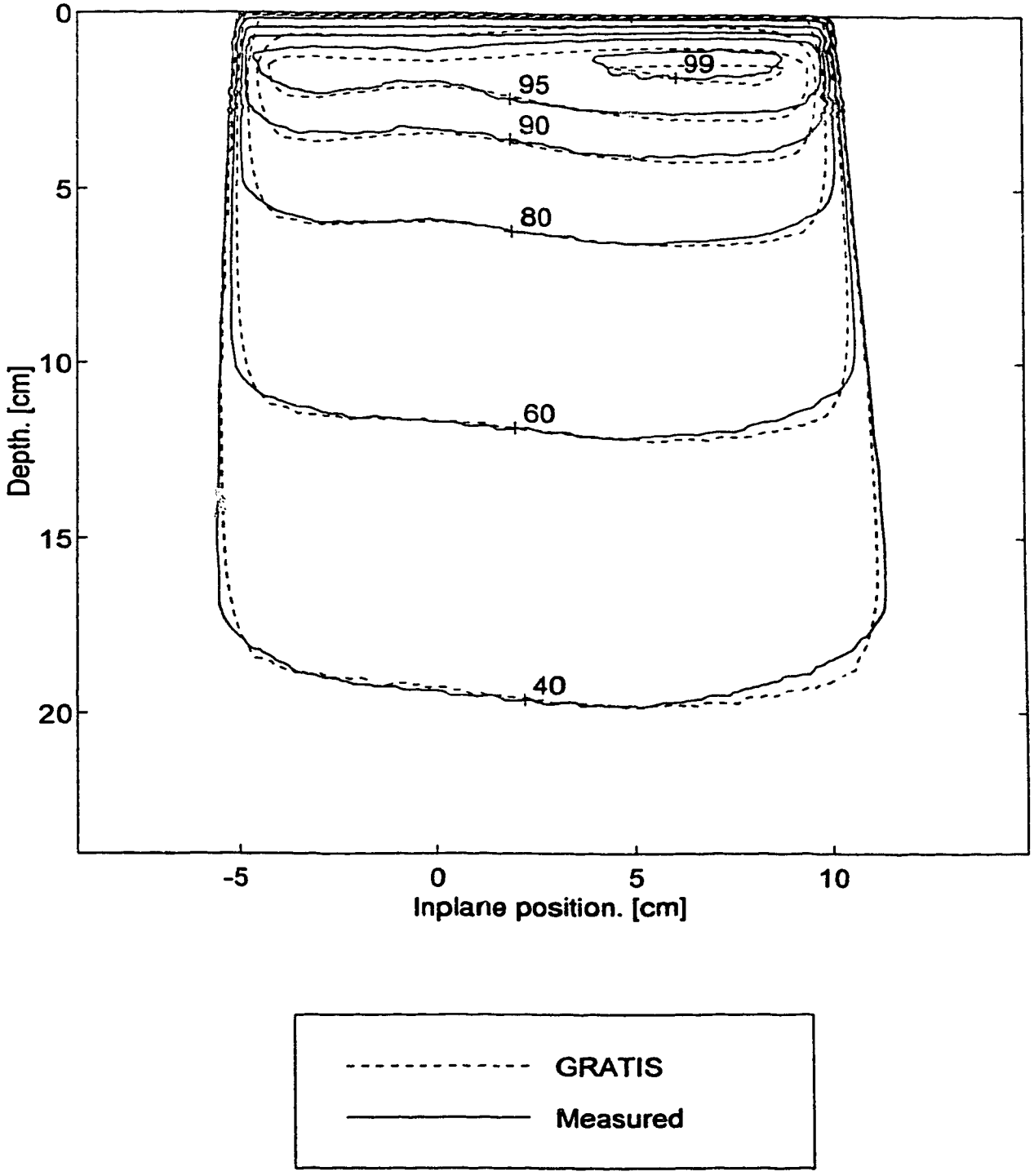


Figure 4.9f. 10x10, X=0 Y=5, YZ-iso X=0, tilted dartboard.

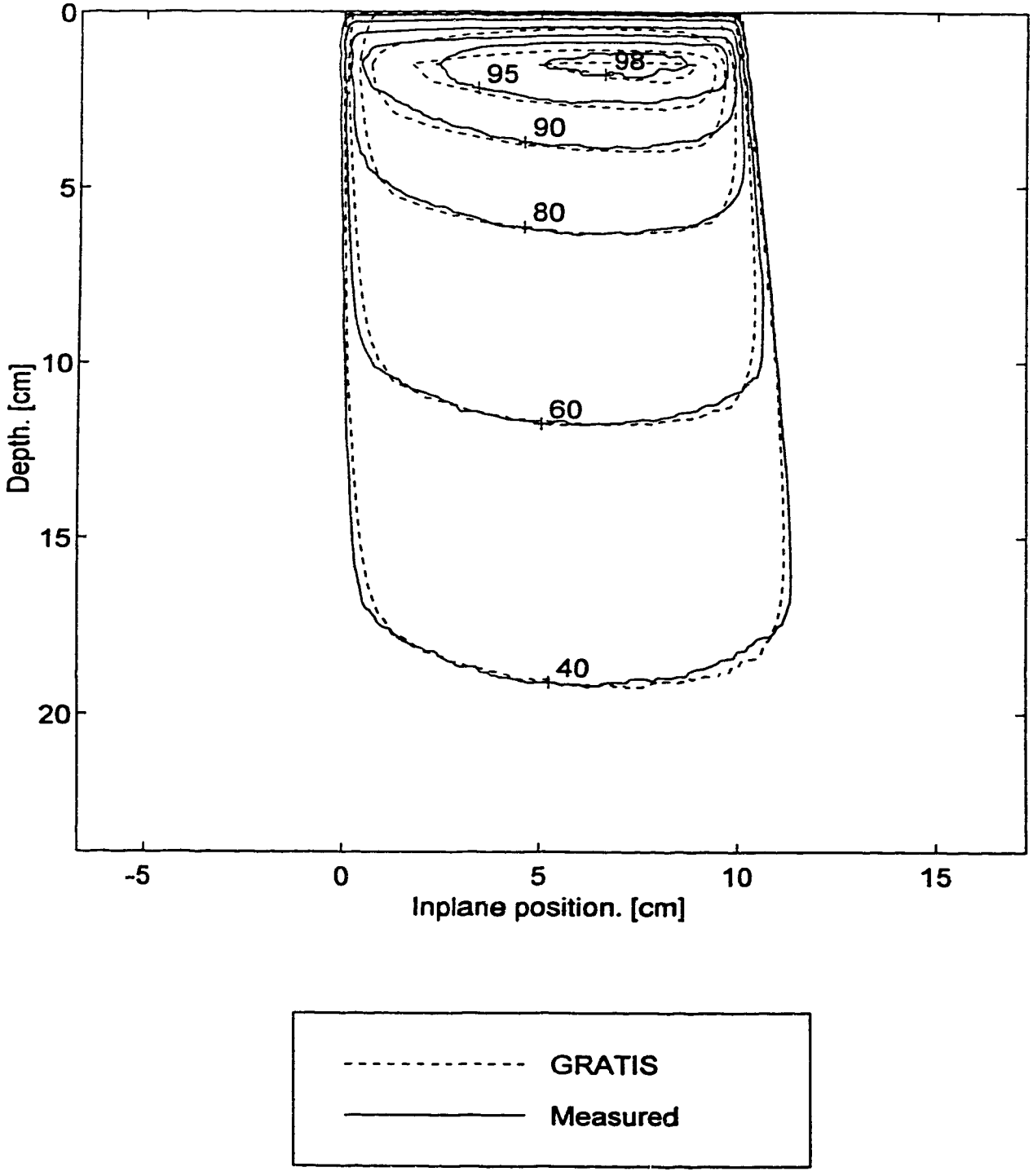
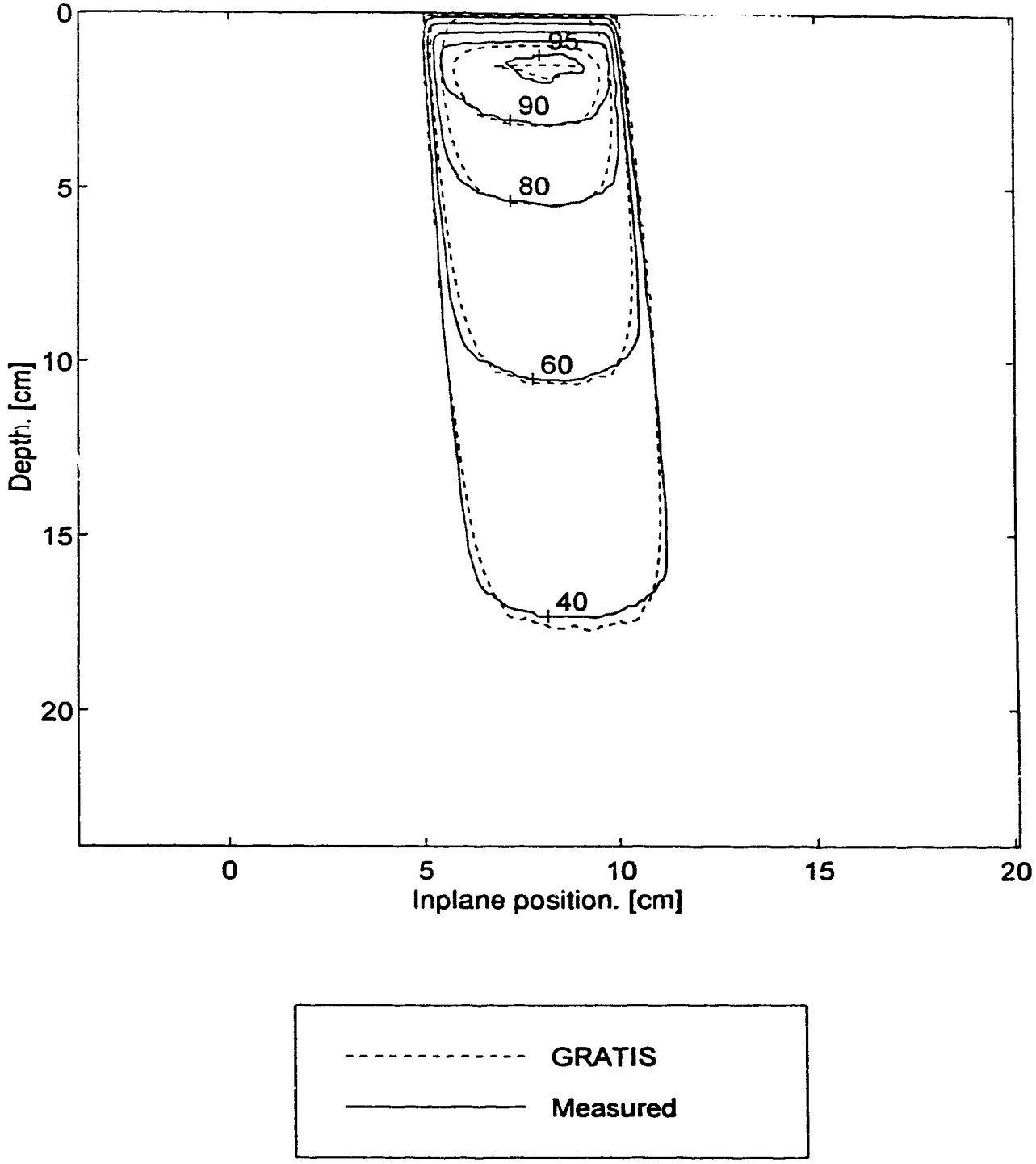


Figure 4.9g. 10x5, X=0 Y=7.5, YZ-iso X=0, tilted dartboard.



Investigation of the GRATIS code revealed that the SSD function (c.f. equation 3.3b) was zero for all calculation points in these two cases, and the GRATIS implementation of the algorithm in equation 3.3 skips the scatter integration for all calculation points with zero SSD. As a result, the scatter contribution to each calculation point was zero.

$$P = \sqrt{x_p^2 + y_p^2 + z_p^2}$$
$$SAR = 0$$
for $r = 0$ to r_{max} do
  for $\theta = 0$ to $360°$ do
$$(x_c, y_c, \phi) = F(x_p, y_p, z_p, r, \theta) \qquad [3.3a]$$
$$d = P - SSD(x_c, y_c) \cdot \cos(\phi) \qquad [3.3b]$$
$$SAR = SAR + rel\_intensity(x_c, y_c) \cdot \Delta SAR(r, d) \qquad [3.3c]$$
  end for
end for

The construction of the SSD function made the assumption that the field was symmetric; an assumption clearly documented in the original source code. This SSD function is stored in GRATIS in a two dimensional array, which represents a 50x50 cm^2 grid taken on a reference plane through isocenter and perpendicular to the central axis. The positions of the collimators are used to determine the rectangular region of the array that has non-zero values (i.e. inside the field), and patient contour data is used to determine what SSD values should be stored in this non-zero region. For most field sizes only a small fraction of the 50x50 cm^2 grid is irradiated, and hence considerable computation time is saved if the SSD values are computed for that small region only. The rest of the 50x50 cm^2 grid is simply initialized to zero.

The assumption of symmetric fields lay in the method for using the collimator positions to determine the rectangular region. Only the positions of the right and upper jaws were used to define the edges of the rectangular region. The lower and left edges were assumed to be placed symmetrically to the upper and right jaws, respectively. Thus, when the right jaw was moved up to (and over) the central axis the left jaw was assumed to do the same, resulting in a rectangular region of zero area in the contour array. The entire contour array was set to zero and hence the scatter integration was skipped for all calculation points.

The code of the calculation routine was easily modified to remove this limitation. The lower and left edges of the rectangular region were made to depend on the lower and left collimators, in the same way that the other edges depended on their respective collimators. The changes to the C-language code are given in appendicies C and D, and their effect on the isodoses in figures 4.9 is presented in figures 4.10. Figures 4.10a and 4.10b are now mirror images of 4.10g and 4.10f, as expected.

Figure 4.10a. 10x5, X=0 Y=-7.5, YZ-iso X=0, tilted db & map fix.

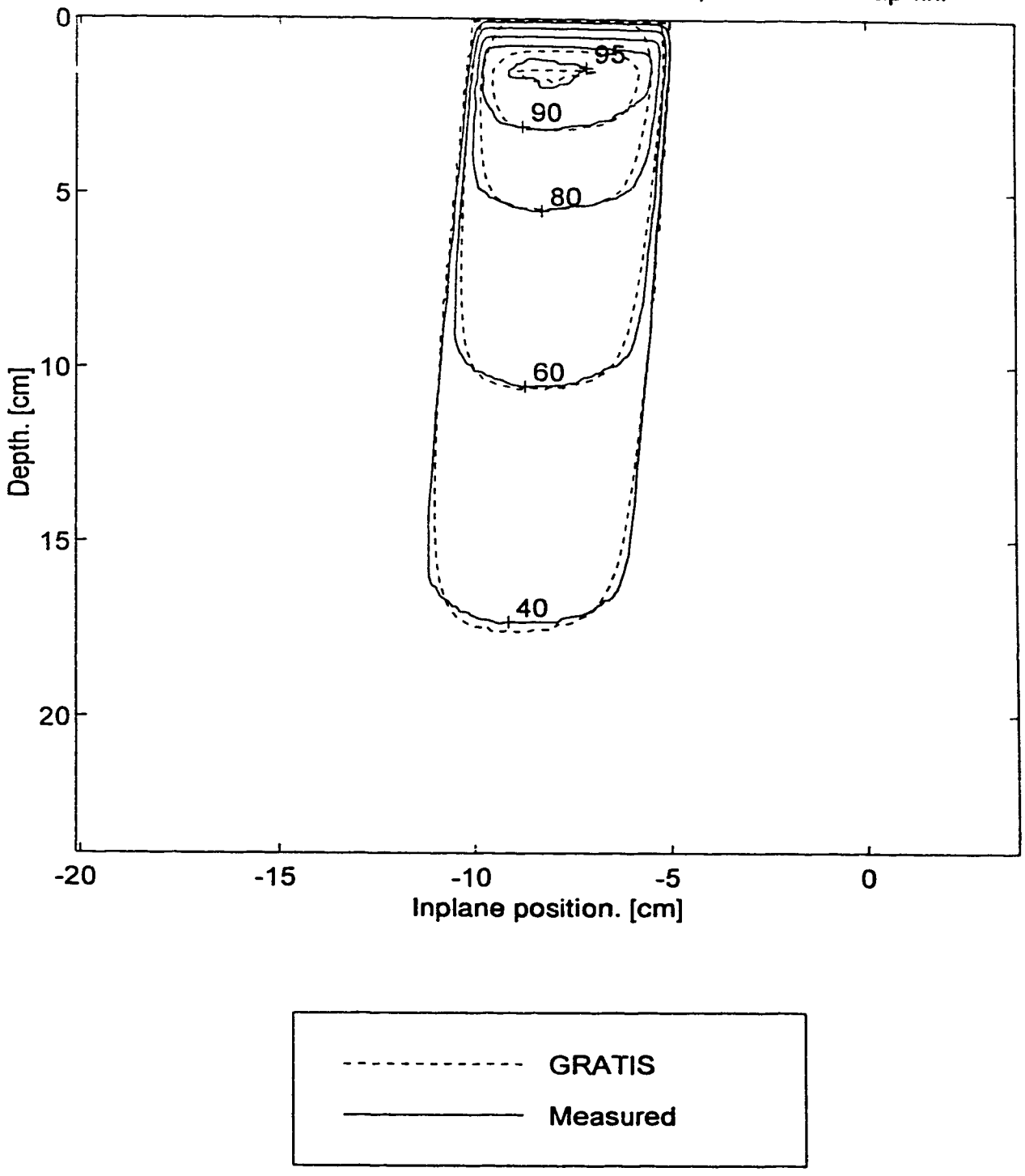


Figure 4.10b. 10x10, X=0 Y=-5, YZ-iso X=0, tilted db & map fix.

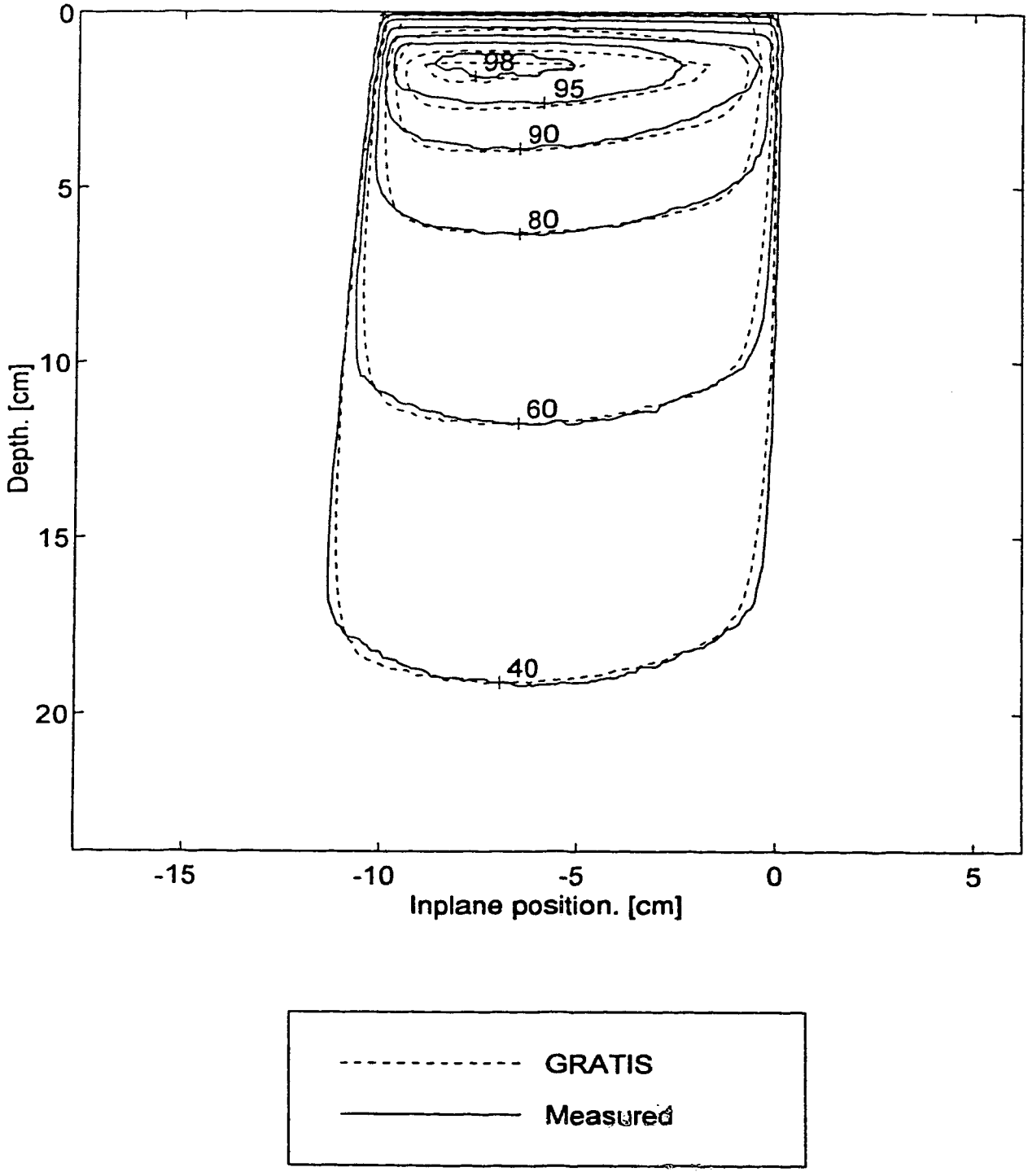


Figure 4.10c. 10x15, X=0 Y=-2.5, YZ-iso X=0, tilted db & map fix.

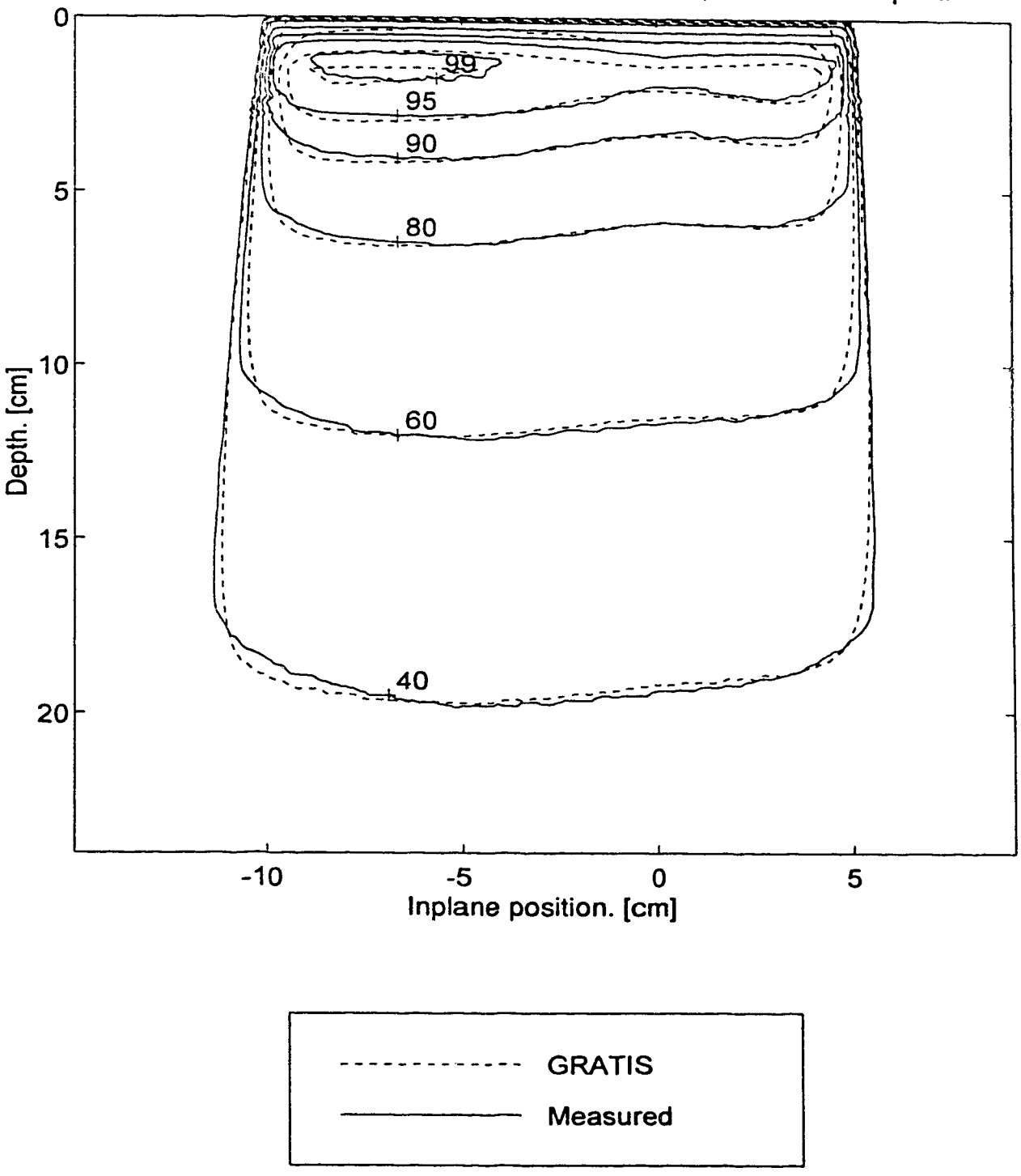


Figure 4.10d. 10x20, X=0 Y=0, YZ-iso X=0, tilted db & map fix.

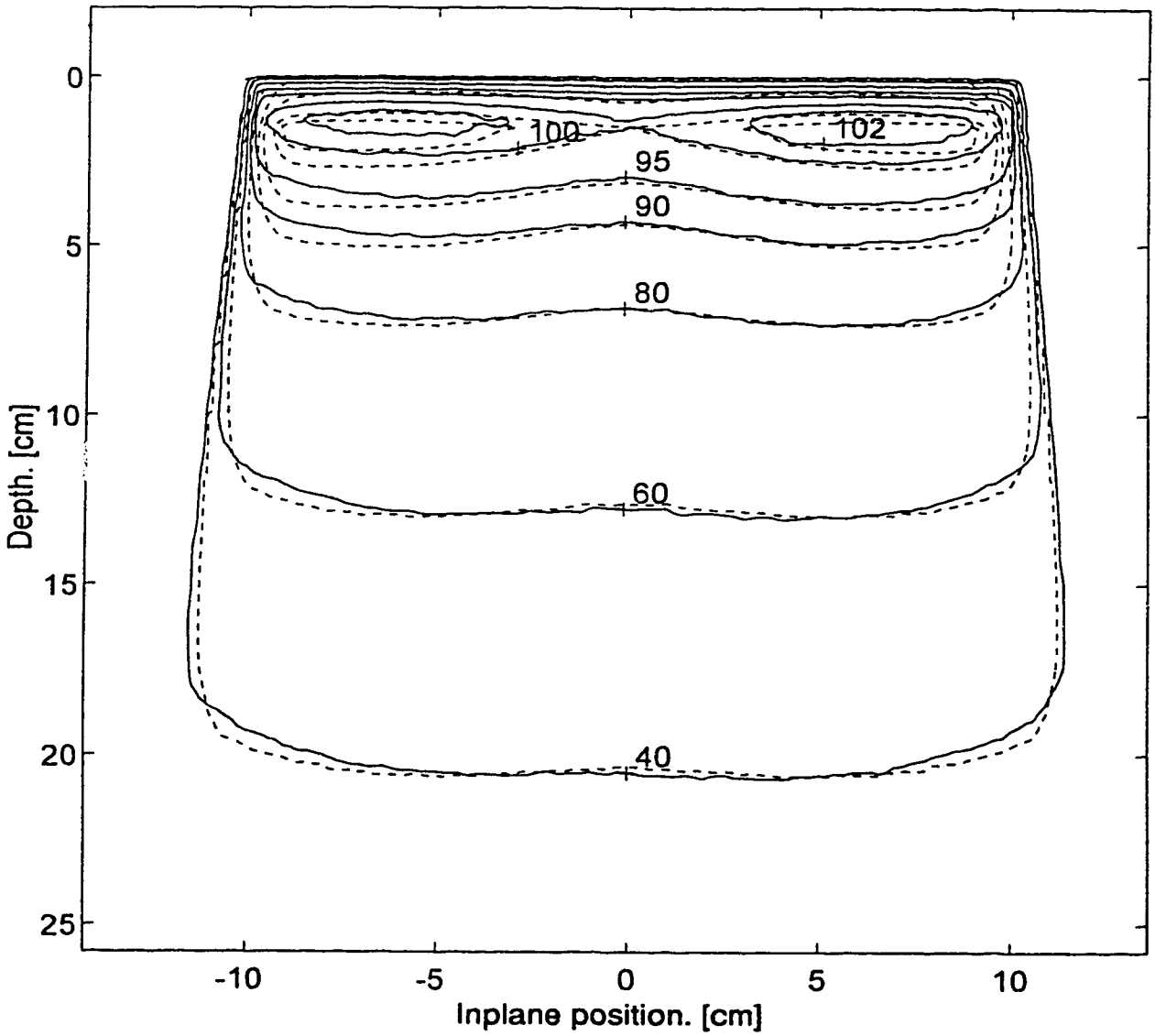


Figure 4.10e. 10x15, X=0 Y=2.5, YZ-iso X=0, tilted db & map fix.

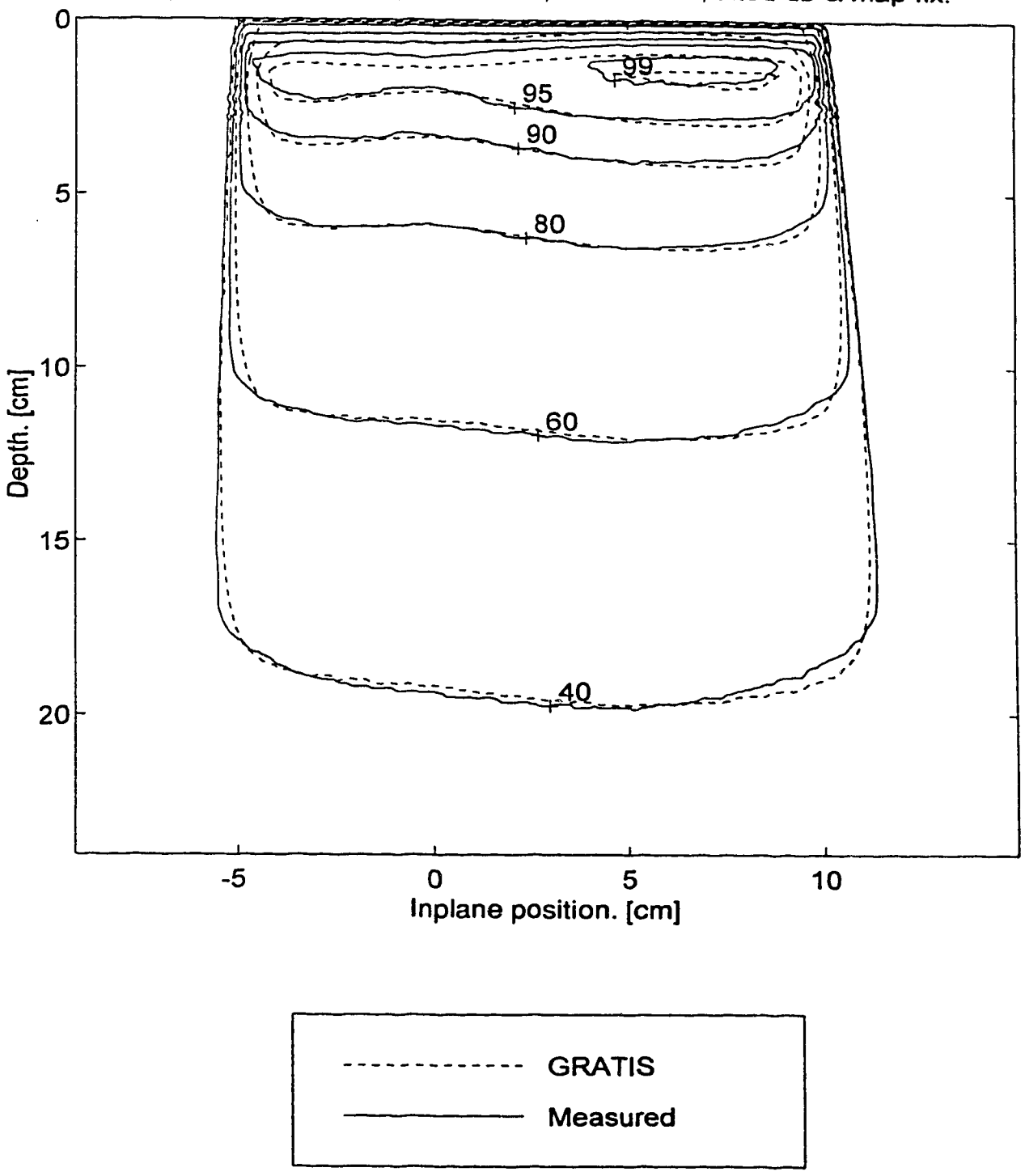


Figure 4.10f. 10x10, X=0 Y=5, YZ-iso X=0, tilted db & map fix.

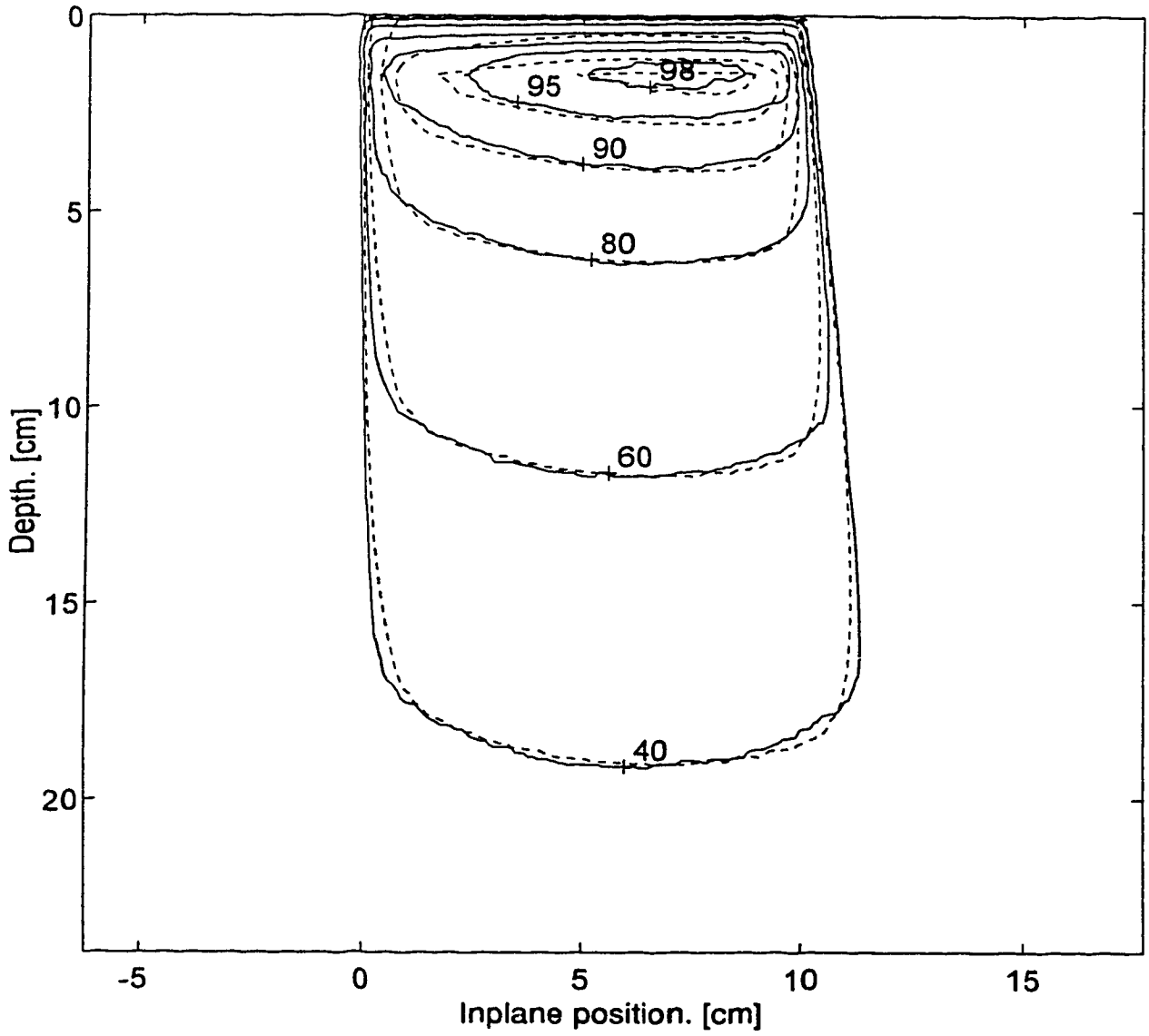


Figure 4.10g. 10x5, X=0 Y=7.5, YZ-iso X=0, tilted db & map fix.

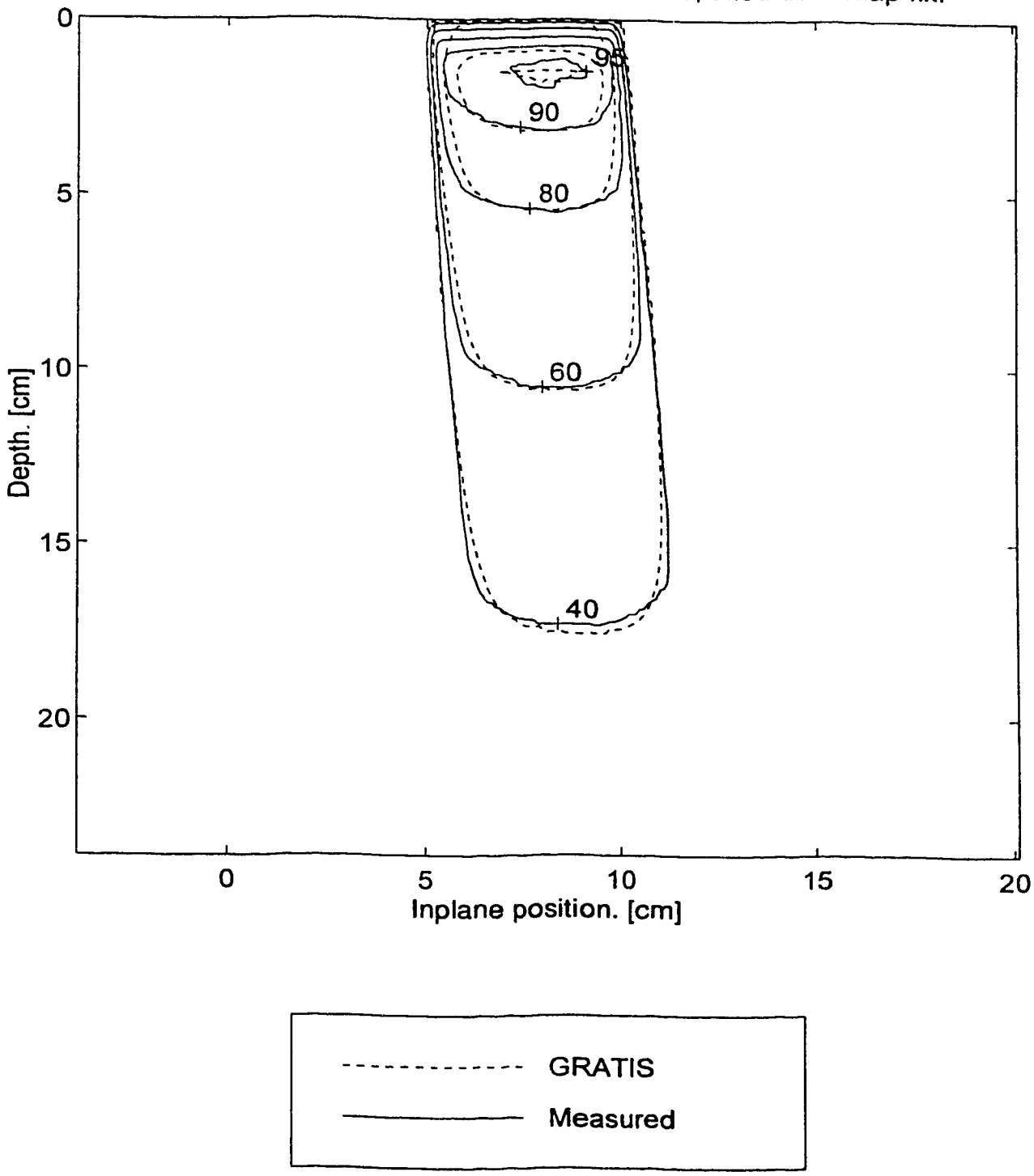


Figure 4.11a. 10x20, X=0 Y=0, YZ-iso X=0, 4% discrepancy level.

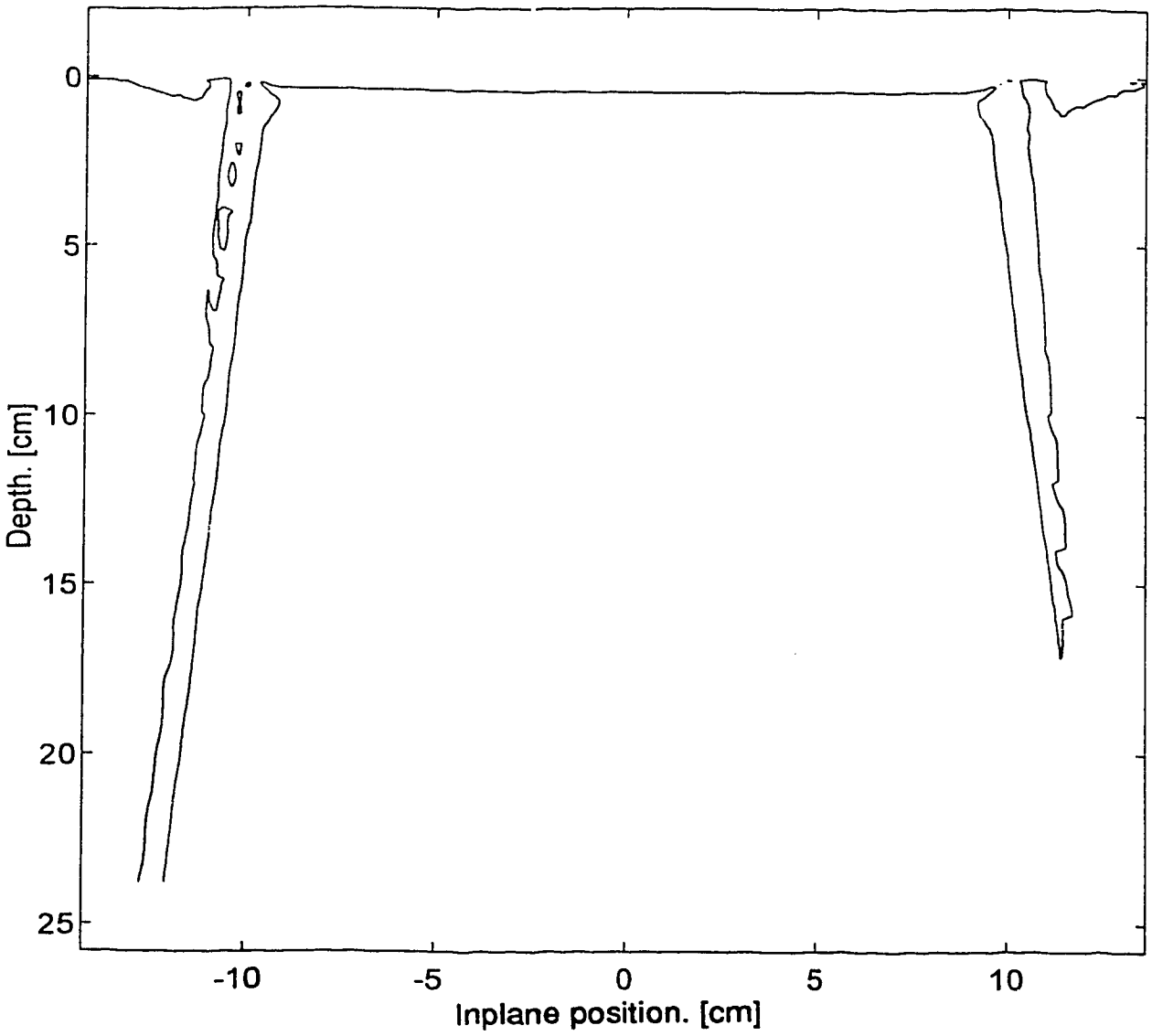


Figure 4.11b. 10x20, X=0 Y=0, YZ-iso X=0, 3% discrepancy.

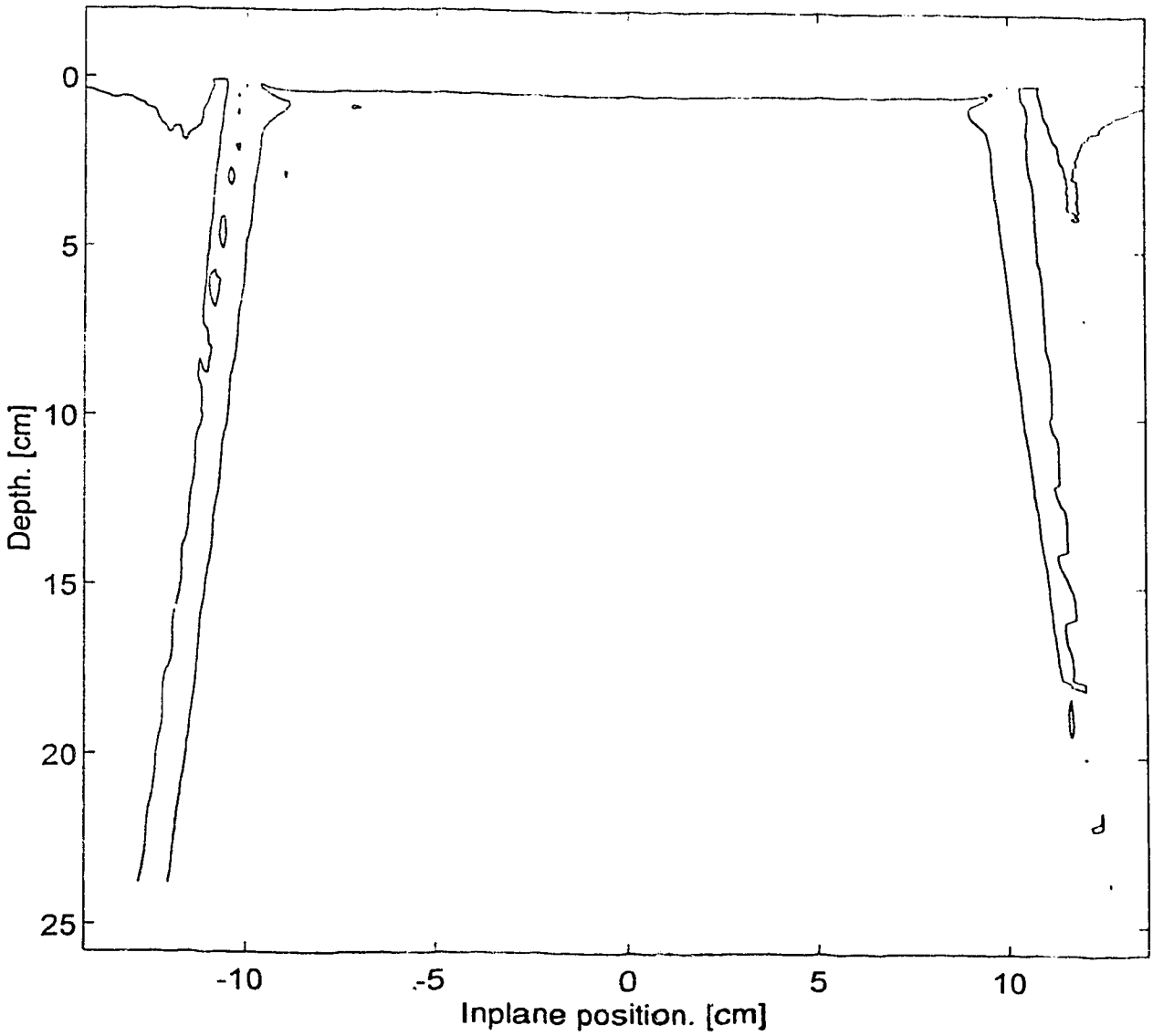


Figure 4.11c. 10x20, X=0 Y=0, YZ-iso X=0, 2% discrepancy level.

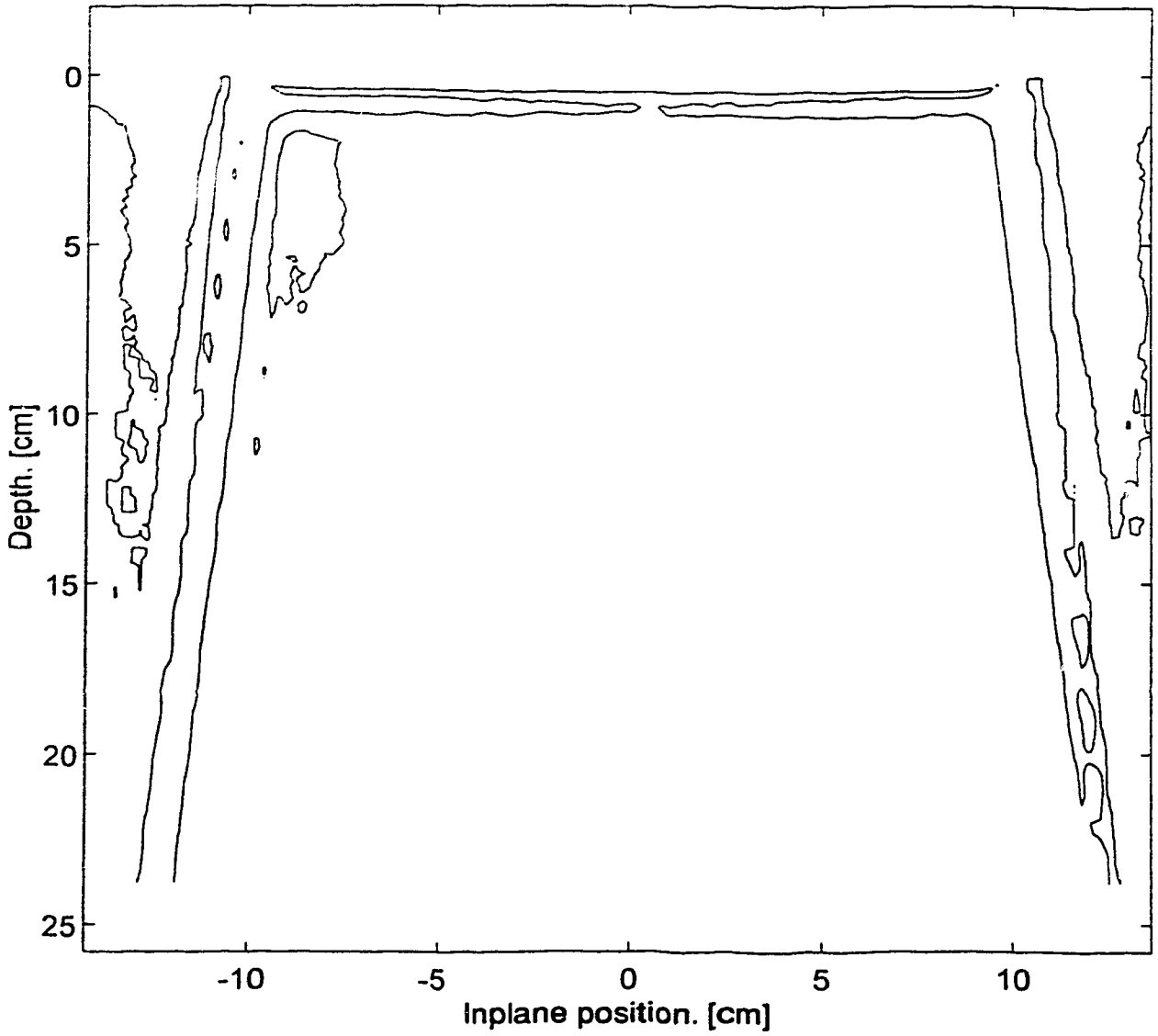
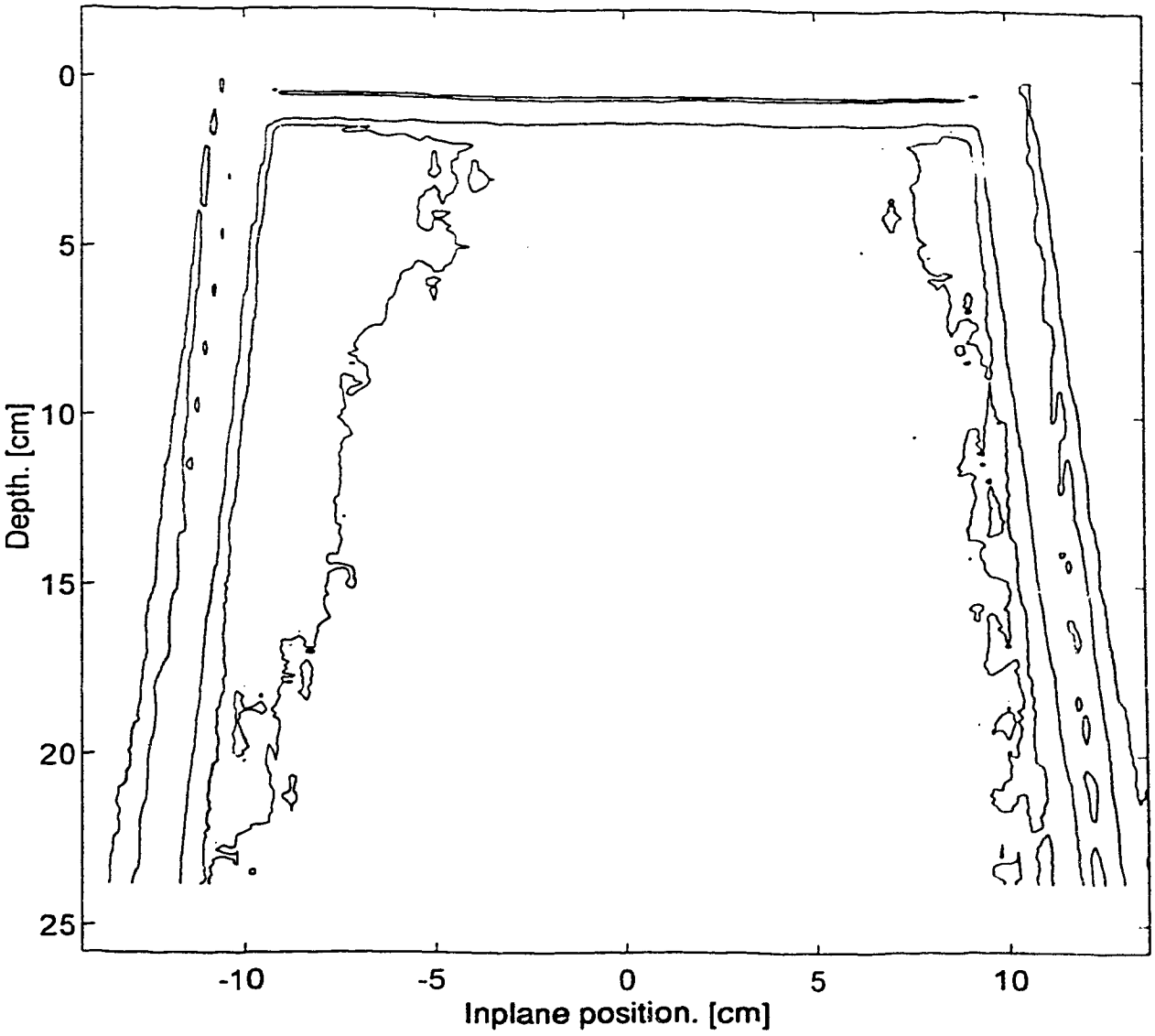


Figure 4.11d. 10x20, X=0 Y=0, YZ-iso X=0, 1% discrepancy level.



The isodoses in figures 4.10 seem to be in fairly good agreement, with the greatest discrepancy being for the largest field (figure 4.10d), as was the case for the symmetric field tests in section 4.1. Figures 4.11 quantify the comparison by plotting the absolute difference of the two data arrays that were used to produce the isodoses in figure 4.10d. The difference levels are normalized to be a percentage of the dose at 1.5 cm depth on the central axis. As can be seen in figure 4.11b, the discrepancy between measured and calculated dose is 3% or less in the region away from the surface and the penumbra. There is, however, a discrepancy exceeding 2% in the small region near the upper left corner of figure 4.11c, and a discrepancy exceeding 1% along both sides of the field and away from the penumbra. The fact that the plots are not completely symmetric can be explained as a slight asymmetry in the experimental setup during measurement. If the diode deviates by a small amount from a scan that is parallel to the water surface, then the data array will be asymmetric and produce discrepancy plots like the ones in figure 4.11c and 4.11d.

The asymmetric 10x10 field in figure 4.10f also seems to have about the same agreement as did the symmetric 10x10 field in figure 4.8b, suggesting that the calculation accuracy is not dependent on the position of the field (though it may still be dependent upon field size, field shape and other factors). Figure 4.12 shows a more extreme test of this assertion, in which a 10x10 field has been placed 11.4 cm away from the central axis. The significance of choosing 11.4 cm rather than the maximum possible displacement of 15 cm (the collimators will not allow for further displacement) is simply to allow this measurement to be reused in the analysis of beam-softening with distance from the central axis (section 4.3). In that analysis the maximum displacement is 11.4 cm, due to a rotation of the gantry.

It can be seen in figure 4.12b that a discrepancy of 4% is present in the region at 1cm depth and between 14 cm and 16 cm away from the central axis. However, in the shallow depths (i.e. less than 1.5 cm) there is a fairly high gradient in the vertical direction, which means a slight vertical misalignment of the data arrays could introduce a significant discrepancy. Thus this 4% discrepancy may not be due to the dose calculation at all, and for this reason emphasis is placed on comparisons in the region of the field that is outside of the penumbra and more than 1.5 cm deep.

Figures 4.12c and 4.12d show discrepancies of 1 to 2 percent in the lower right hand side of the field, which is small and might be considered acceptable. But the fact that it occurs over a large region that is not an exceptionally high gradient, and that it is on one side of the field might also suggest that beam-softening or some other effect is present. The 100% isodose lines in figure 4.12a also do not match up well, though the discrepancy there is 1% or less.

Figure 4.12a. 10x10, X=0 Y=11.4, YZ-iso X=0, tilted db & map fix.

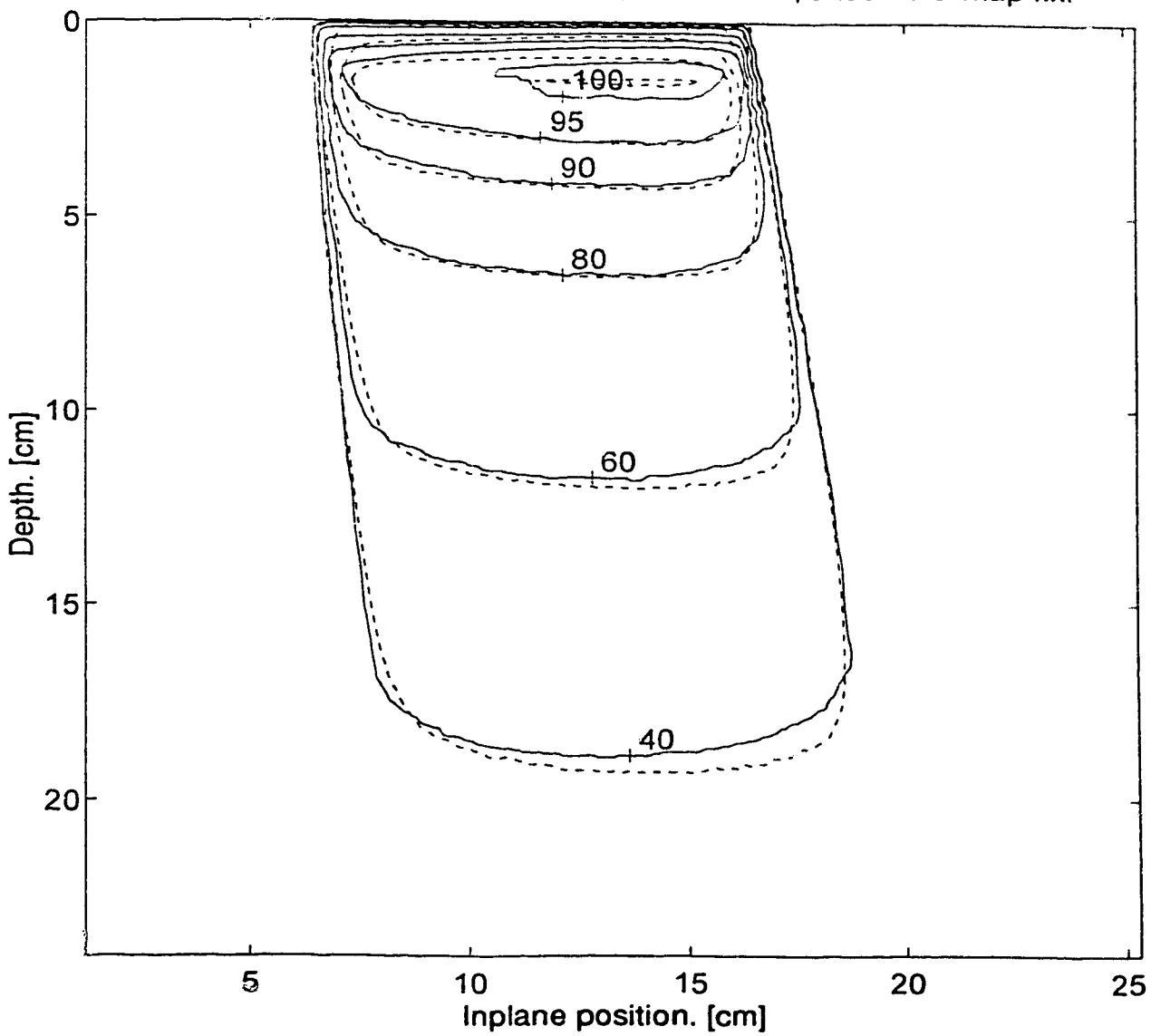


Figure 4.12b. 10x10, X=0 Y=11.4, YZ-iso X=0, 4% discrepancy level.

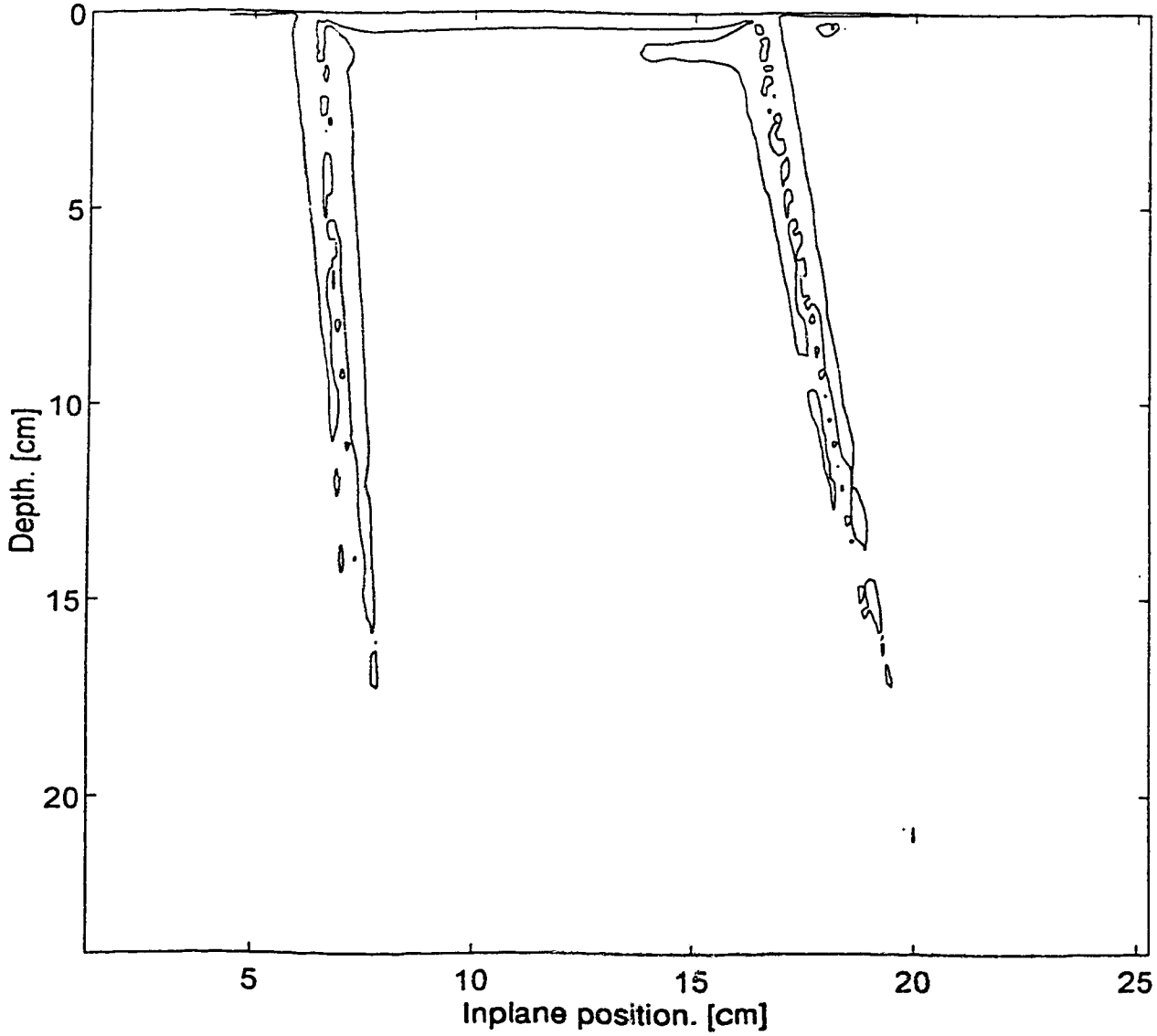


Figure 4.12c. 10x10, X=0 Y=11.4, YZ-iso X=0, 2% discrepancy level.

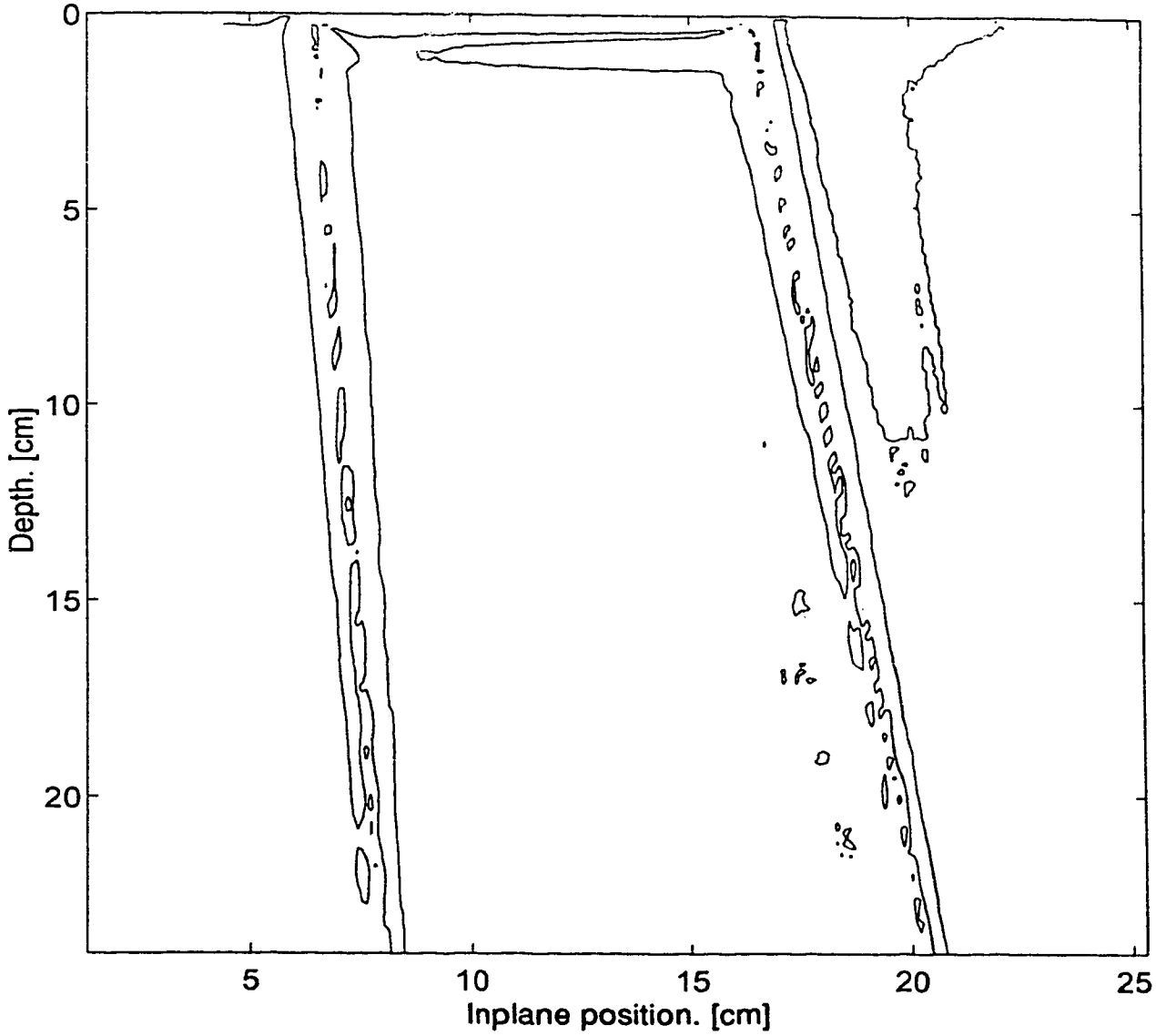
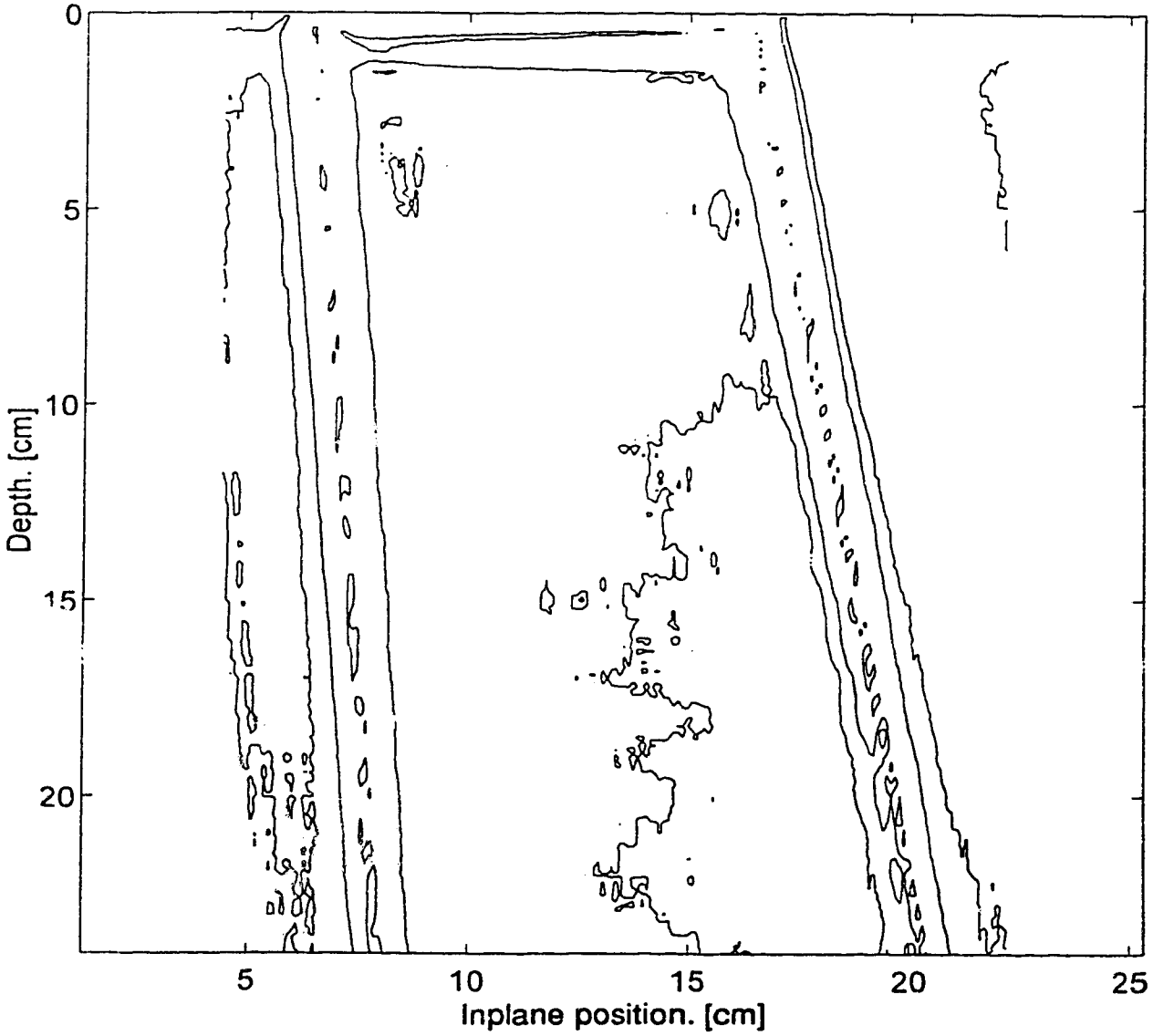


Figure 4.12d. 10x10, X=0 Y=11.4, YZ-iso X=0, 1% discrepancy level.



4.3. Dual Asymmetric Fields

Dual asymmetric fields were studied by displacing a 10x10 cm^2 field diagonally away from the central axis until the X-collimators would not allow it to be moved further (i.e. 10x10, X=7 Y=7). Displacement then continued in the Y direction until the 10x10 field was as far as possible in both the X and Y directions (i.e. the most asymmetric case). The isodoses measured are listed below, and the corresponding plots are shown in figures 4.13. These plots are all normalized so that 100% dose is at 1.5 cm depth on the central axis of the symmetric 10x10 field. That is, the 1.5 cm depth point in the symmetric field represents a common reference point for all fields, rather than each field being normalized separately.

10x10, X=0 Y=0,	YZ-iso X=0
10x10, X=3 Y=3,	YZ-iso X=3
10x10, X=5 Y=5,	YZ-iso X=5
10x10, X=7 Y=7,	YZ-iso X=7
10x10, X=7 Y=15,	YZ-iso X=7
10x10, X=7 Y=15,	XZ-iso Y=15

Figure 4.13b shows fairly large discrepancies compared with figures 4.13a, 4.13c and 4.13d, contrary to the expectation that either 4.13a or 4.13c should exaggerate any discrepancies seen in this field. However, the ragged appearance of the 100% isodose level in figure 4.13b, and the slight raising of the isodose lines on the left side of the plot suggest that an error occurred during measurement. It is likely that the reference probe[†] was above (or nearly above) the path of the scanning probe, thus interfering with the results. Normalizing both isodose sets to the same dose value at 1.5 cm depth and 3 cm crossplane position may have produced the observed discrepancy if the measured dose at that reference position was interfered with in this way. To test this idea, the measured isodoses can be renormalized to a point that is away from the affected region, such as 3 cm depth and a crossplane position of 5 cm. This renormalization is shown in figure 4.13g where it can be seen that the measured isodoses agree much better.

[†] A reference probe is placed in the beam upstream from the phantom so that temporal fluctuations in beam intensity can be normalized out by the Wellhöfer scanning system. That is, the linac does not actually produce a constant beam intensity over time.

Figure 4.13a. 10x10, X=0 Y=0, YZ-iso X=0, tilted db & map fix.

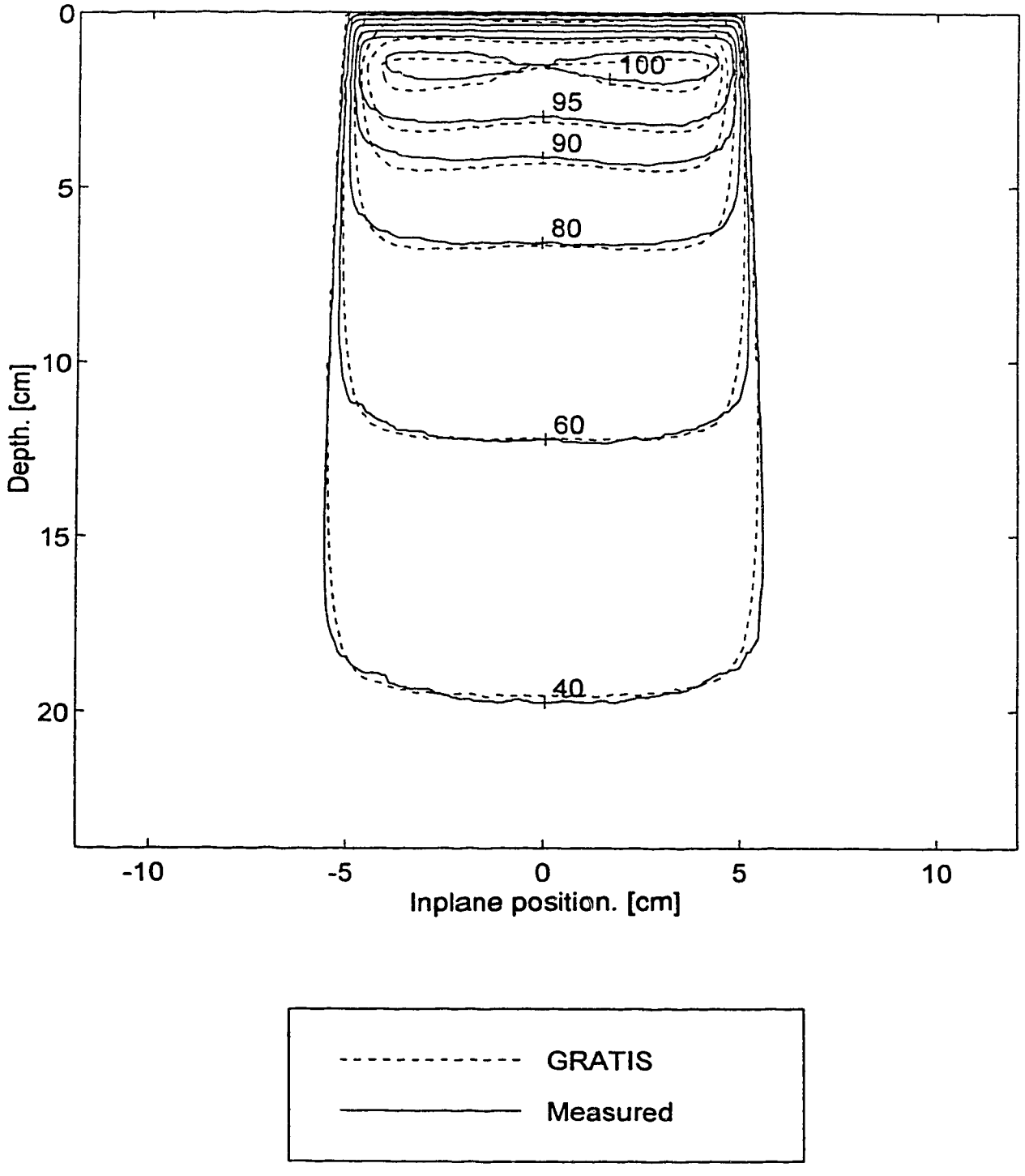


Figure 4.13b. 10x10, X=3 Y=3, YZ-iso X=3, tilted db & map fix.

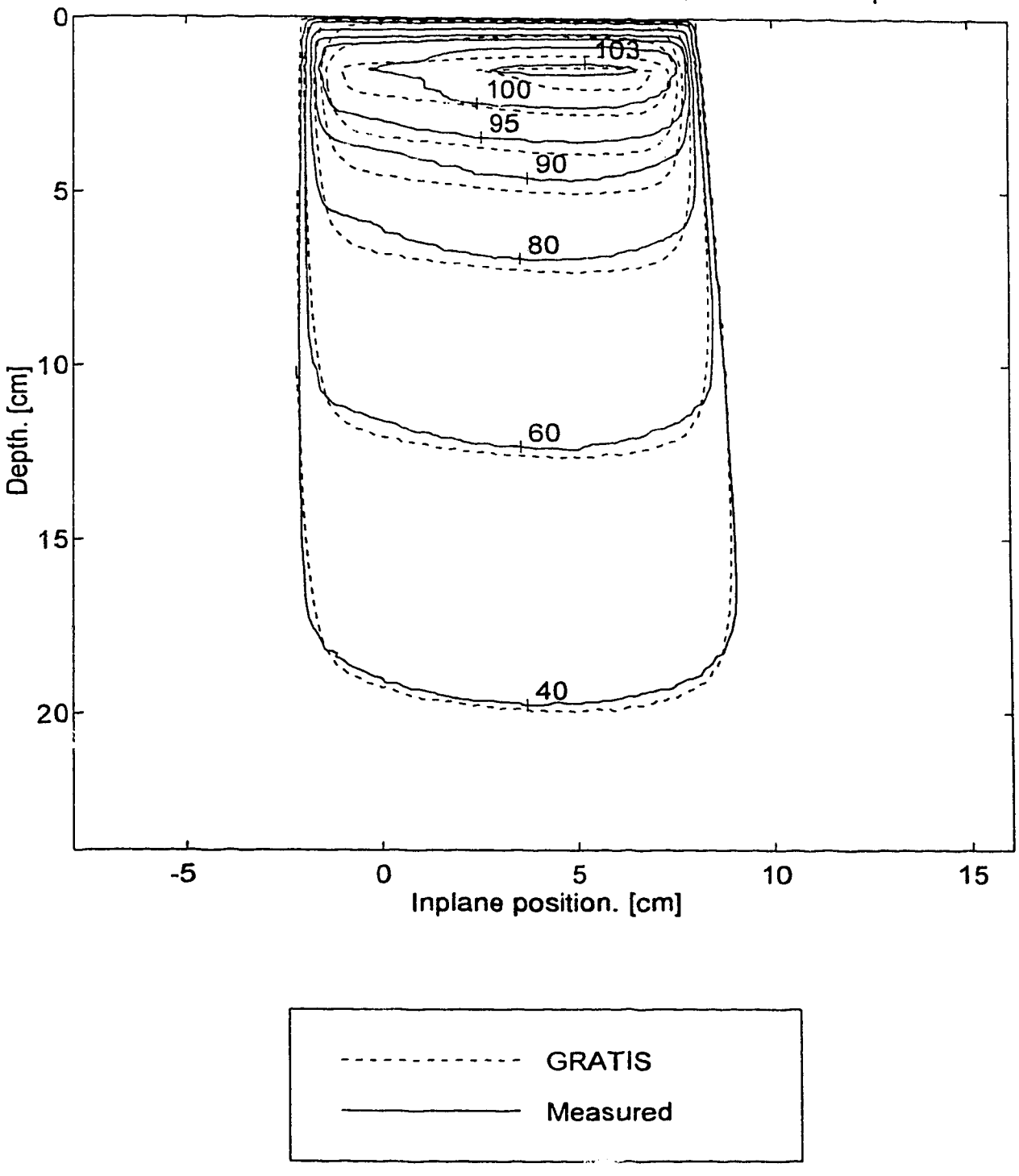


Figure 4.13c. 10x10, X=5 Y=5, YZ-iso X=5, tilted db & map fix.

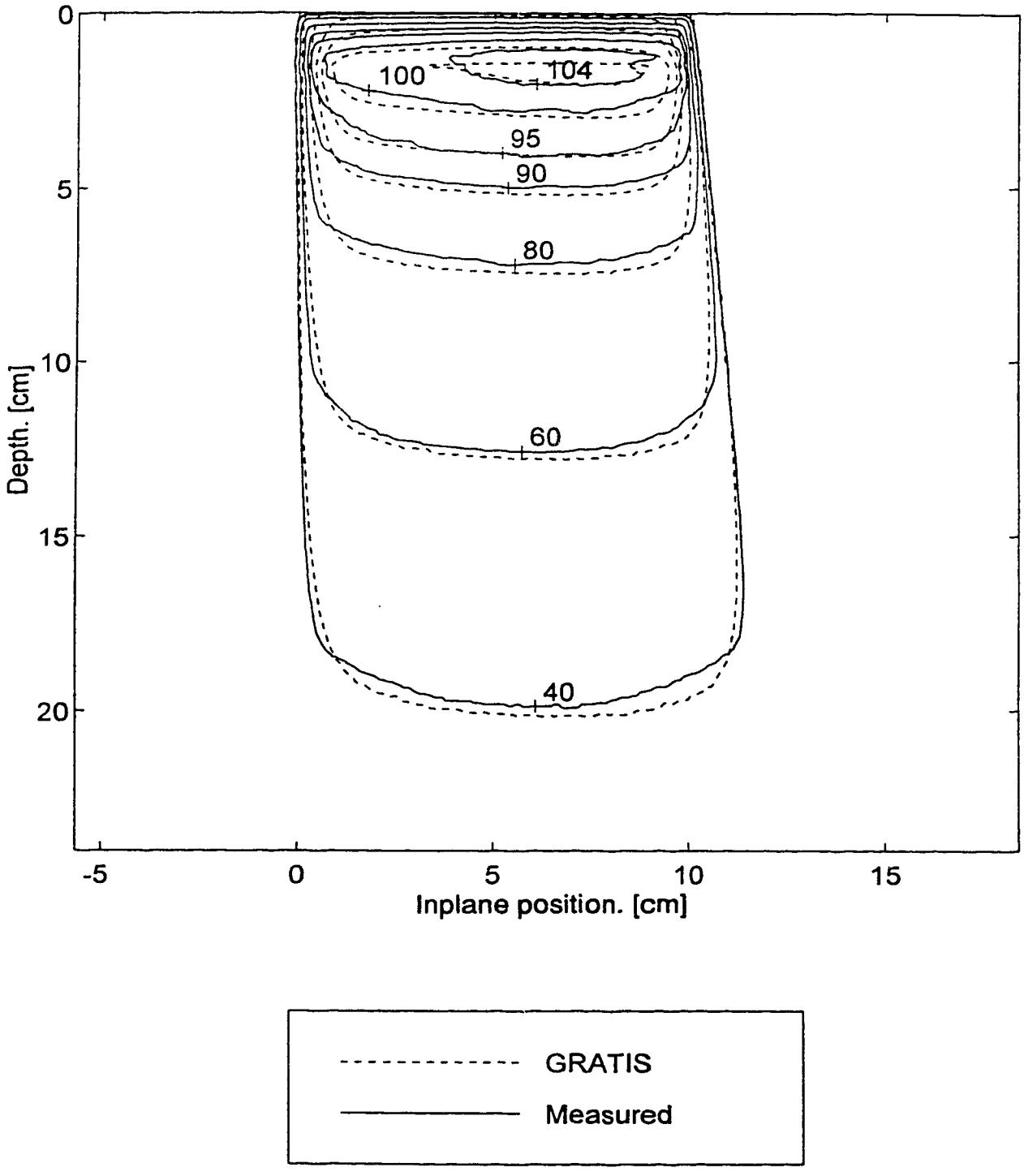


Figure 4.13d. 10x10, X=7 Y=7, YZ-iso X=7, tilted db & map fix.

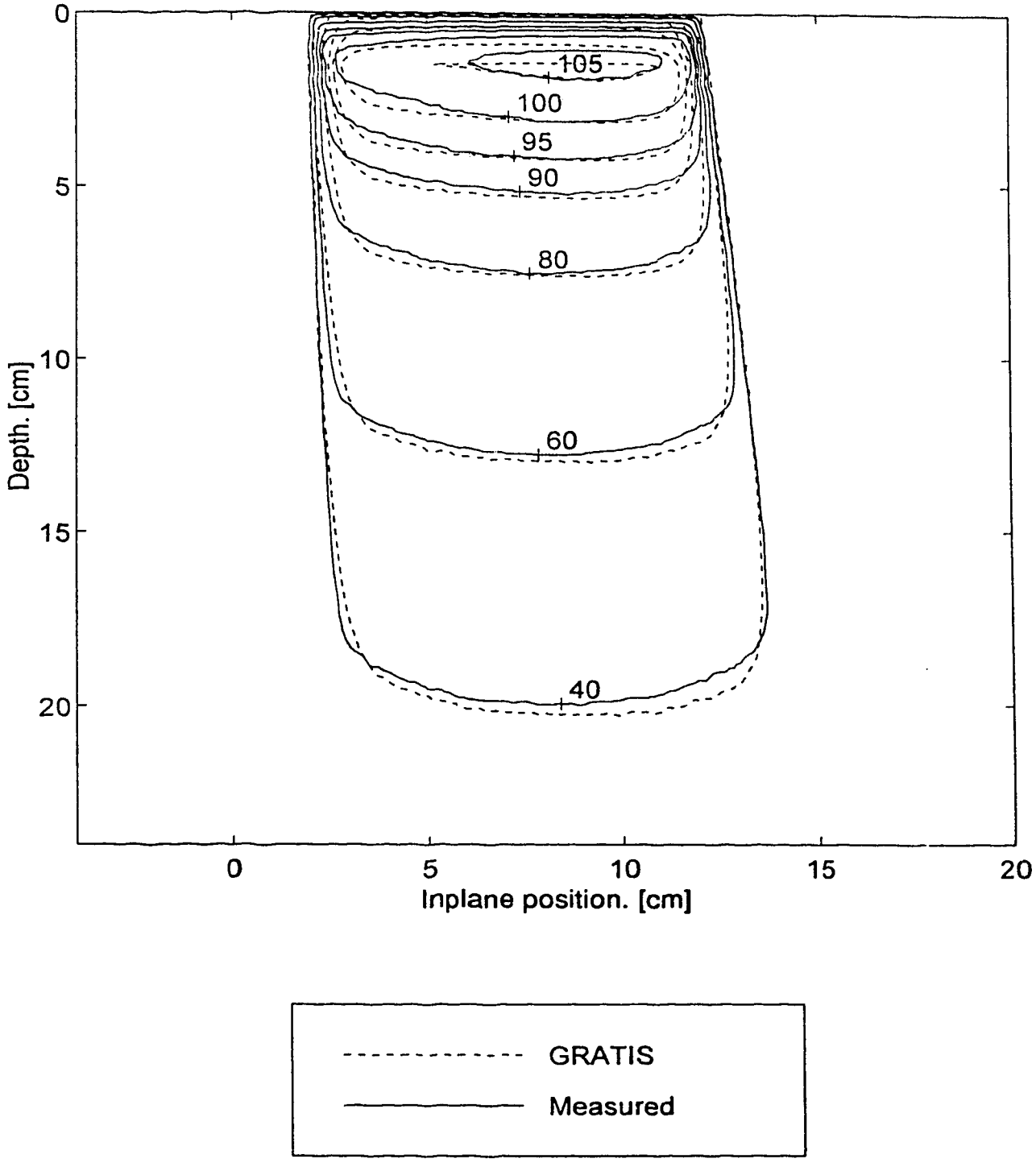
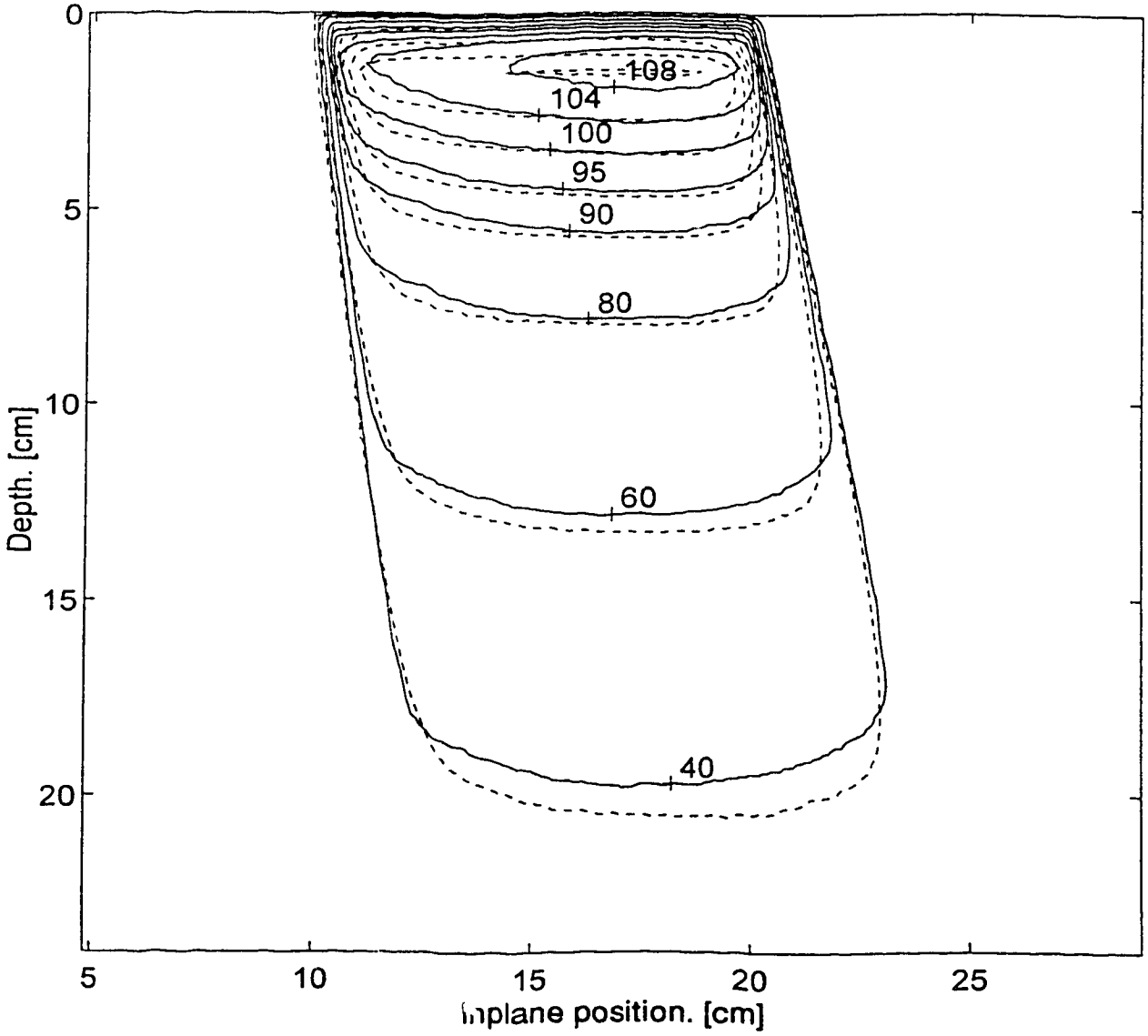


Figure 4.13e. 10x10, X=7 Y=15, YZ-iso X=7, tilted db & map fix.



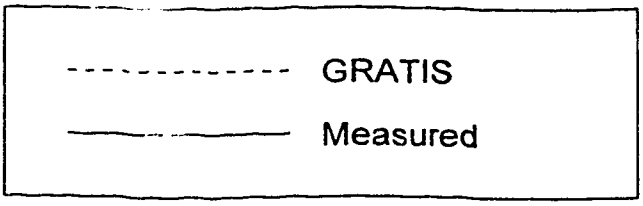


Figure 4.13f. 10x10, X=7 Y=15, XZ-iso Y=15, tilted db & map fix.

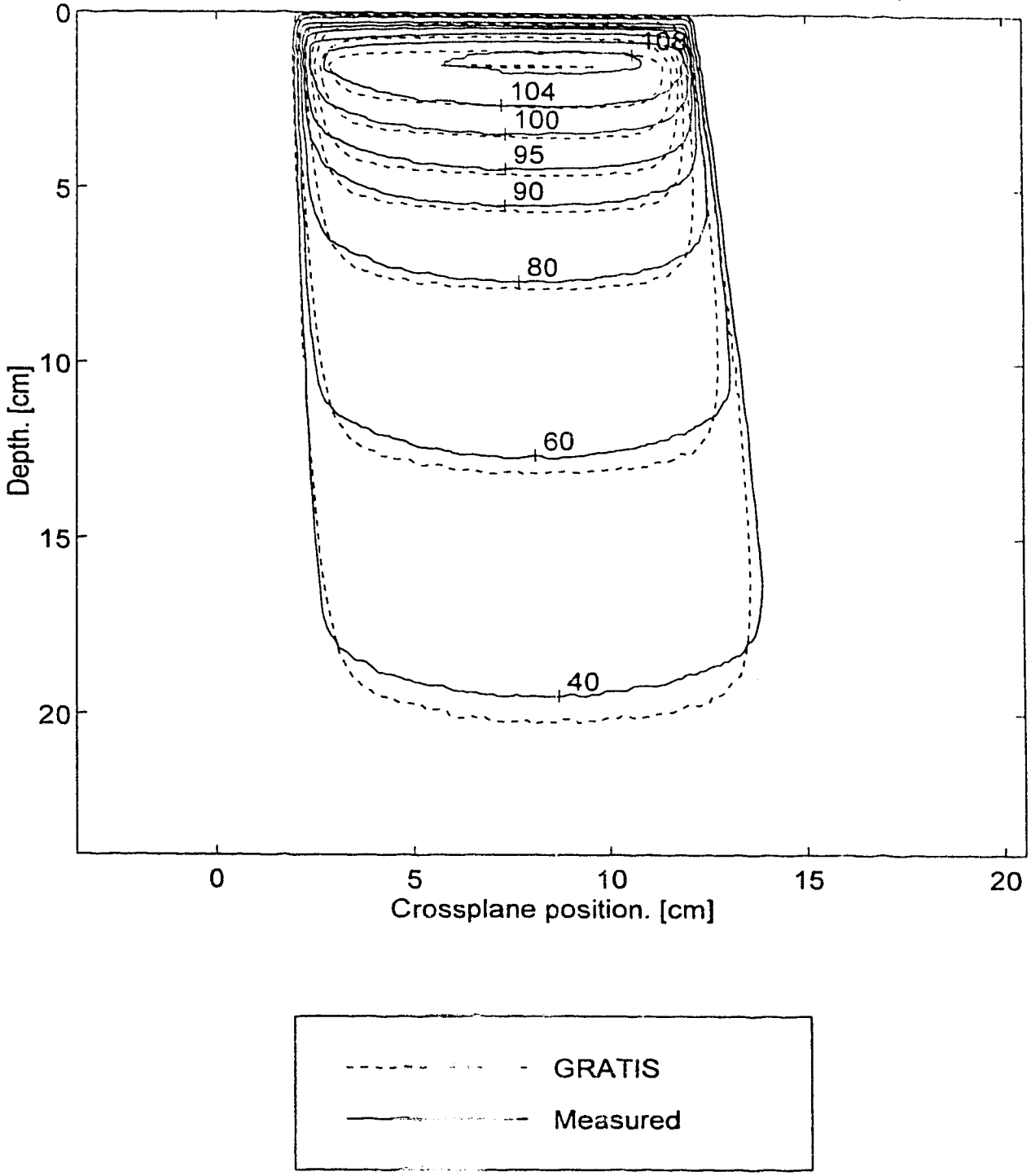


Figure 4.13g. 10x10, X=3 Y=3, YZ-iso X=3, tilted db & map fix.

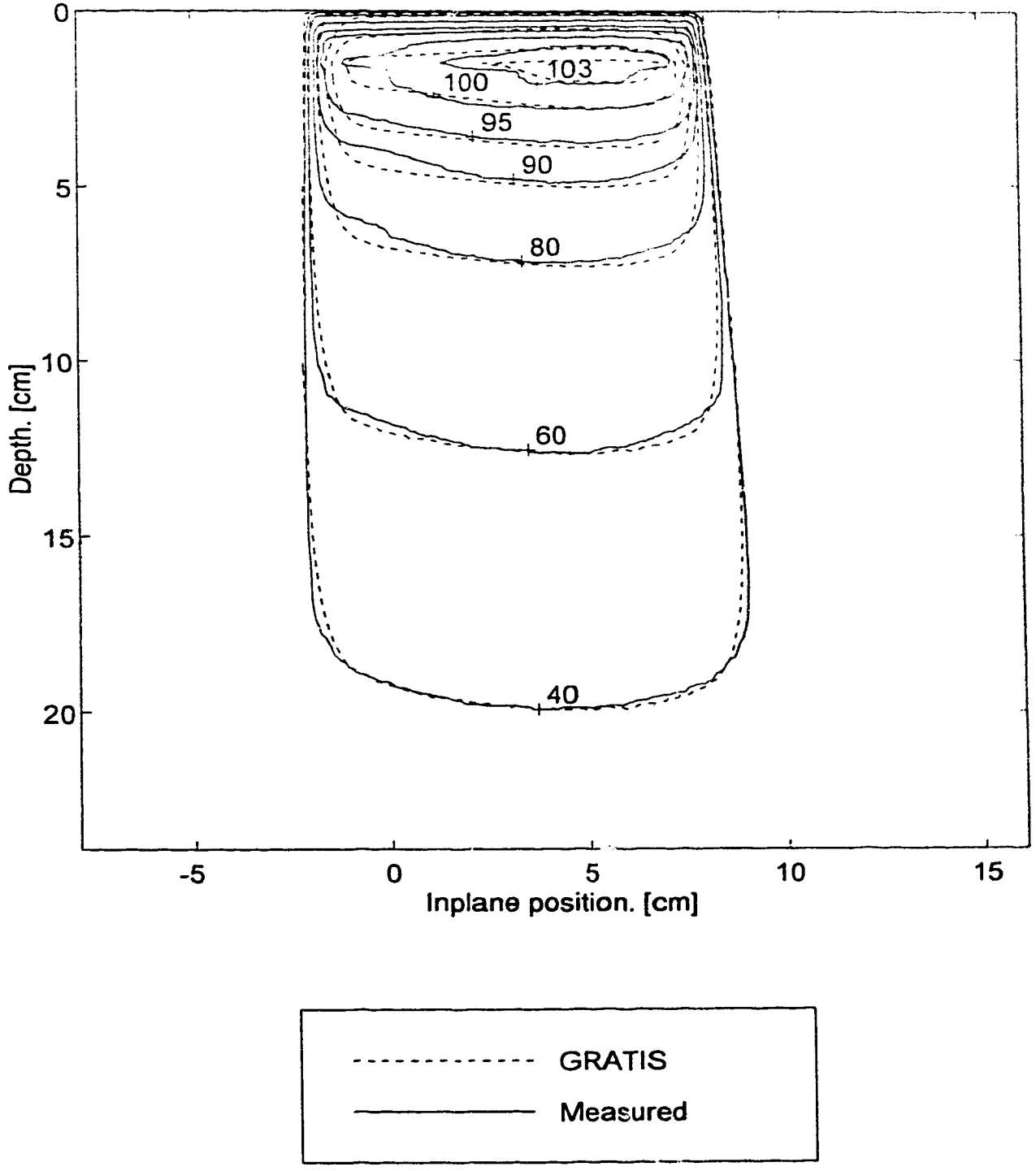


Figure 4.14a. 10x10, X=7 Y=15, YZ-iso X=7, 4% discrepancy level.

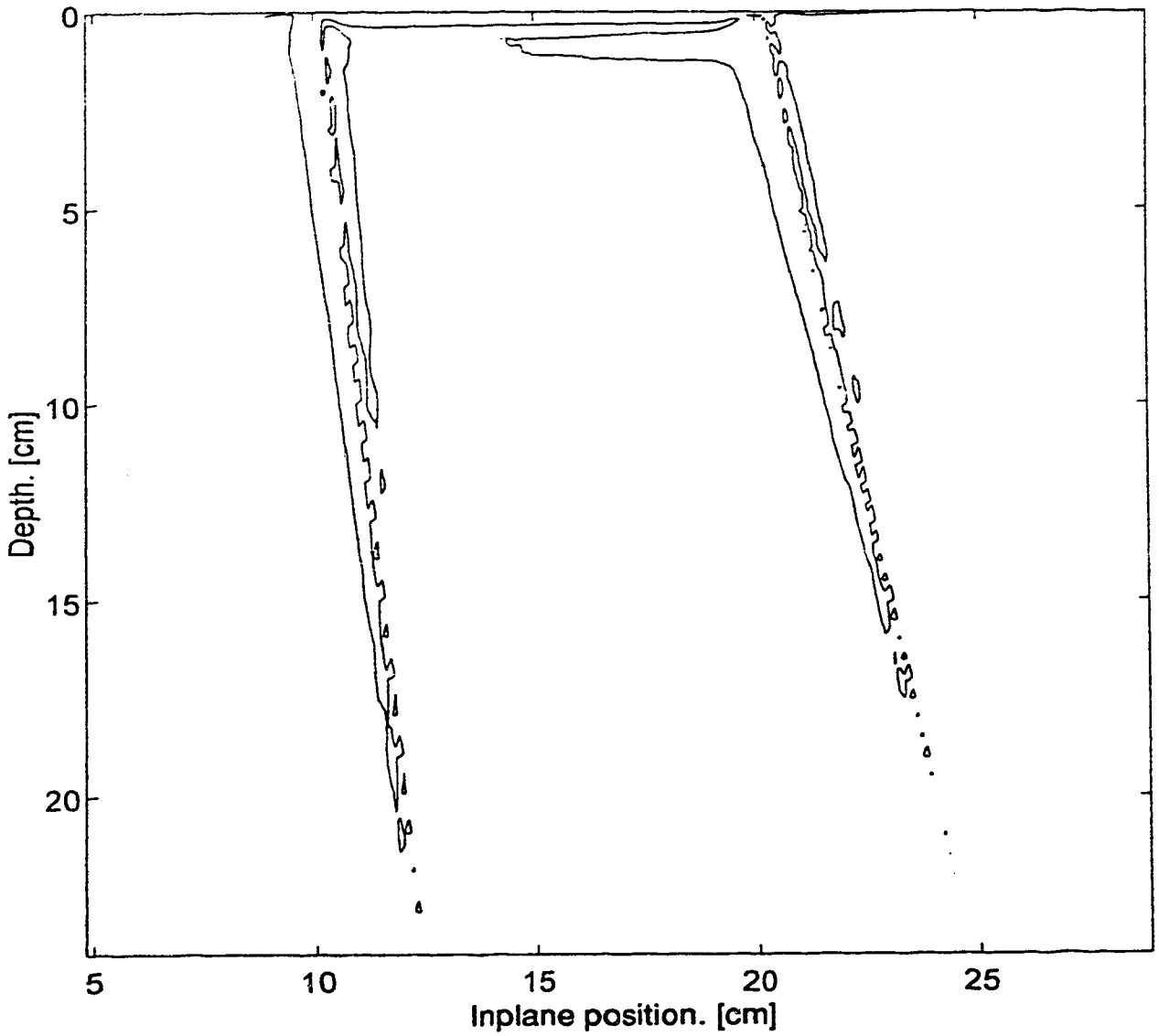


Figure 4.14b. 10x10, X=7 Y=15, YZ-iso X=7, 3% discrepancy level.

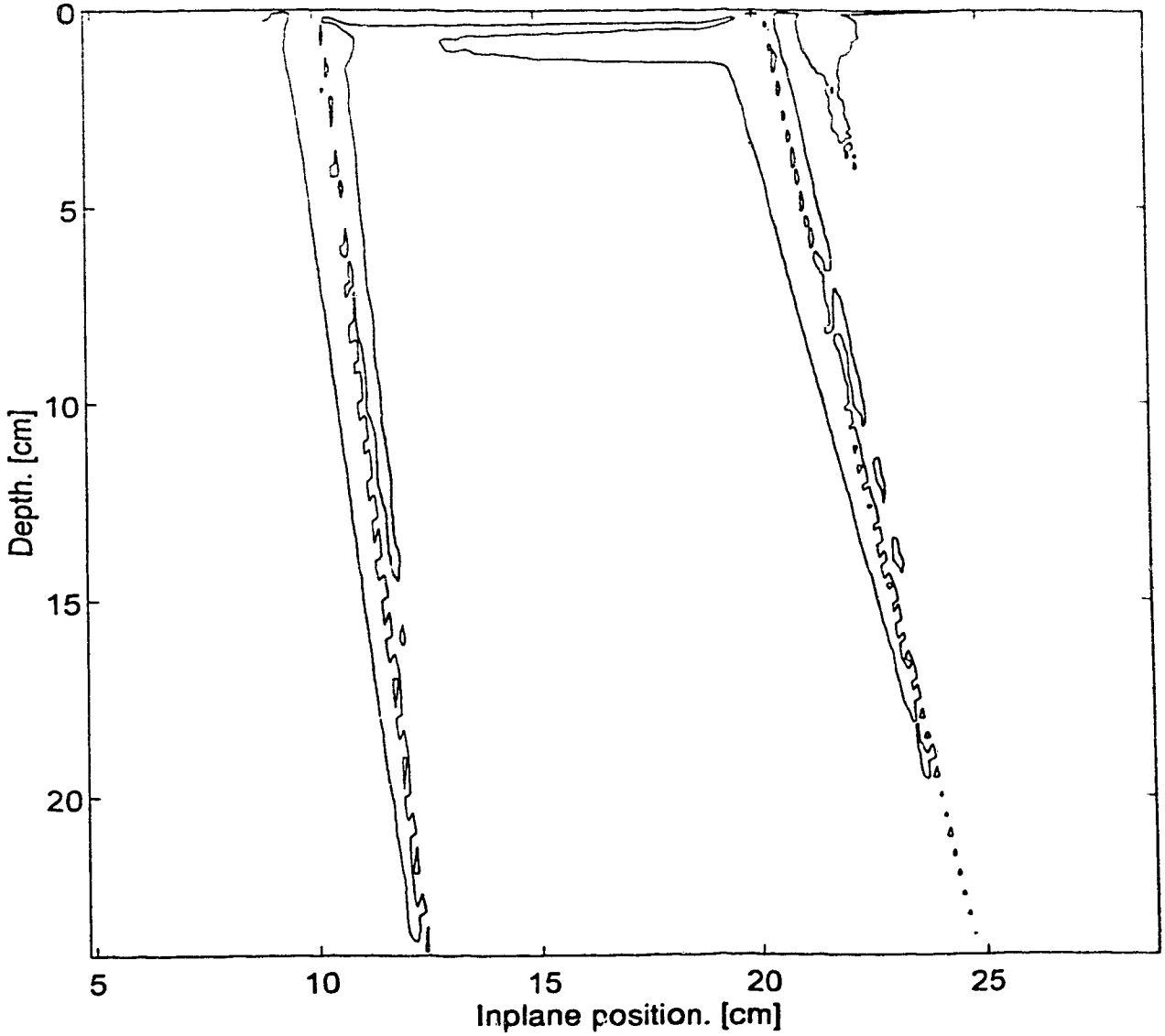


Figure 4.14c. 10x10, X=7 Y=15, YZ-iso X=7, 2% discrepancy level.

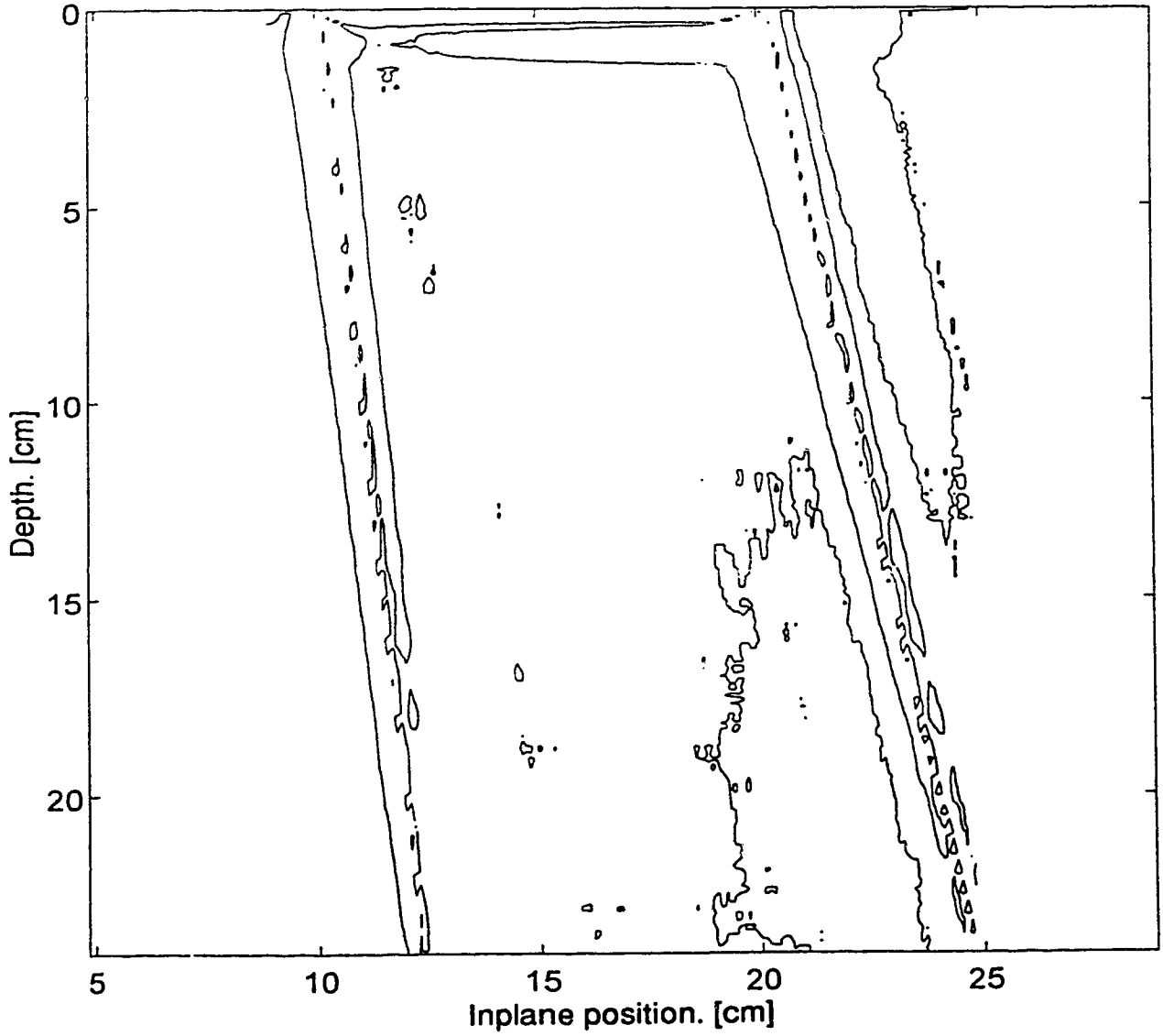


Figure 4.14d. 10x10, X=7 Y=15, YZ-iso X=7, 1% discrepancy level.

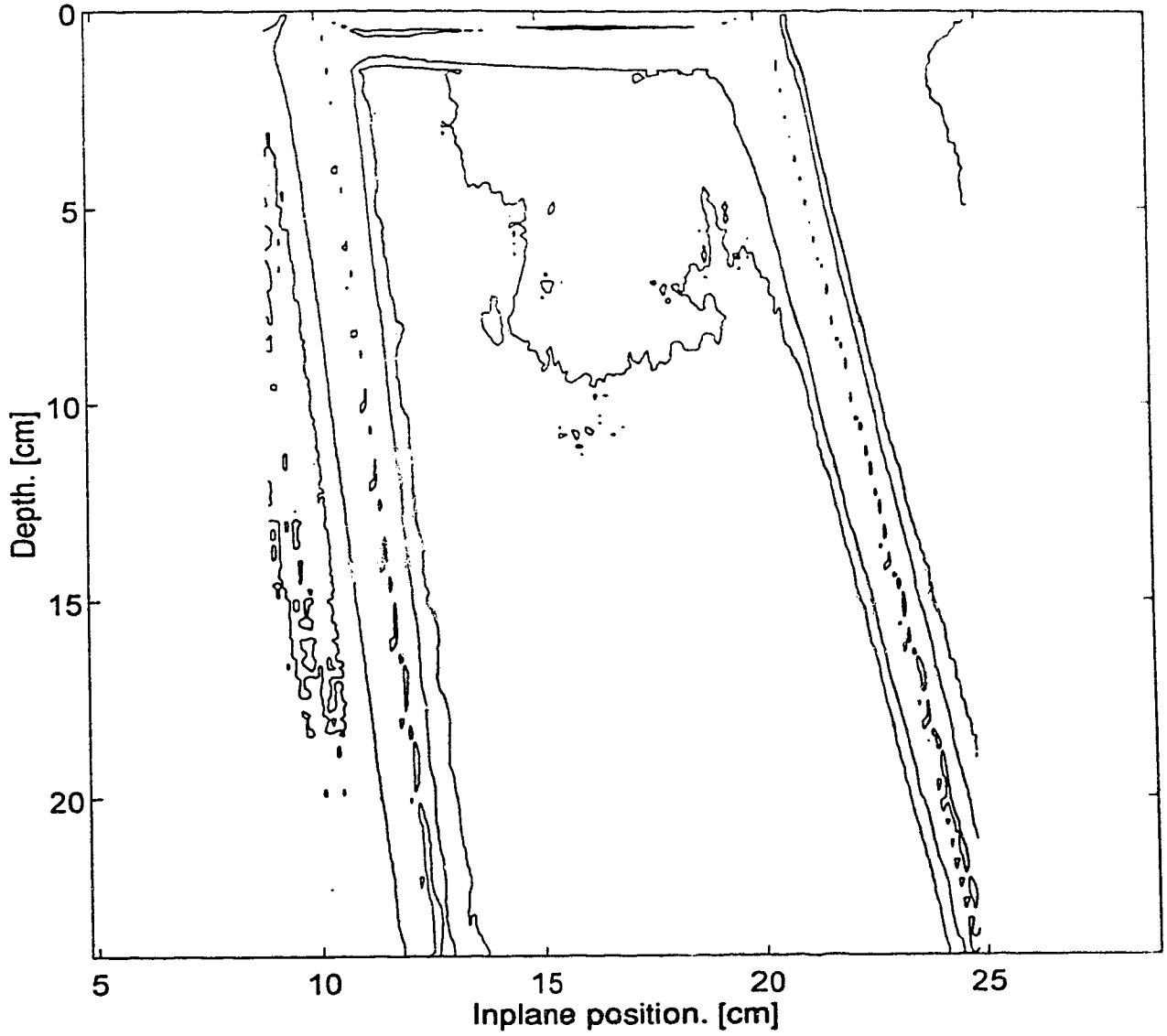


Figure 4.15a. 10x10, X=7 Y=15, XZ-iso Y=15, 4% discrepancy level.

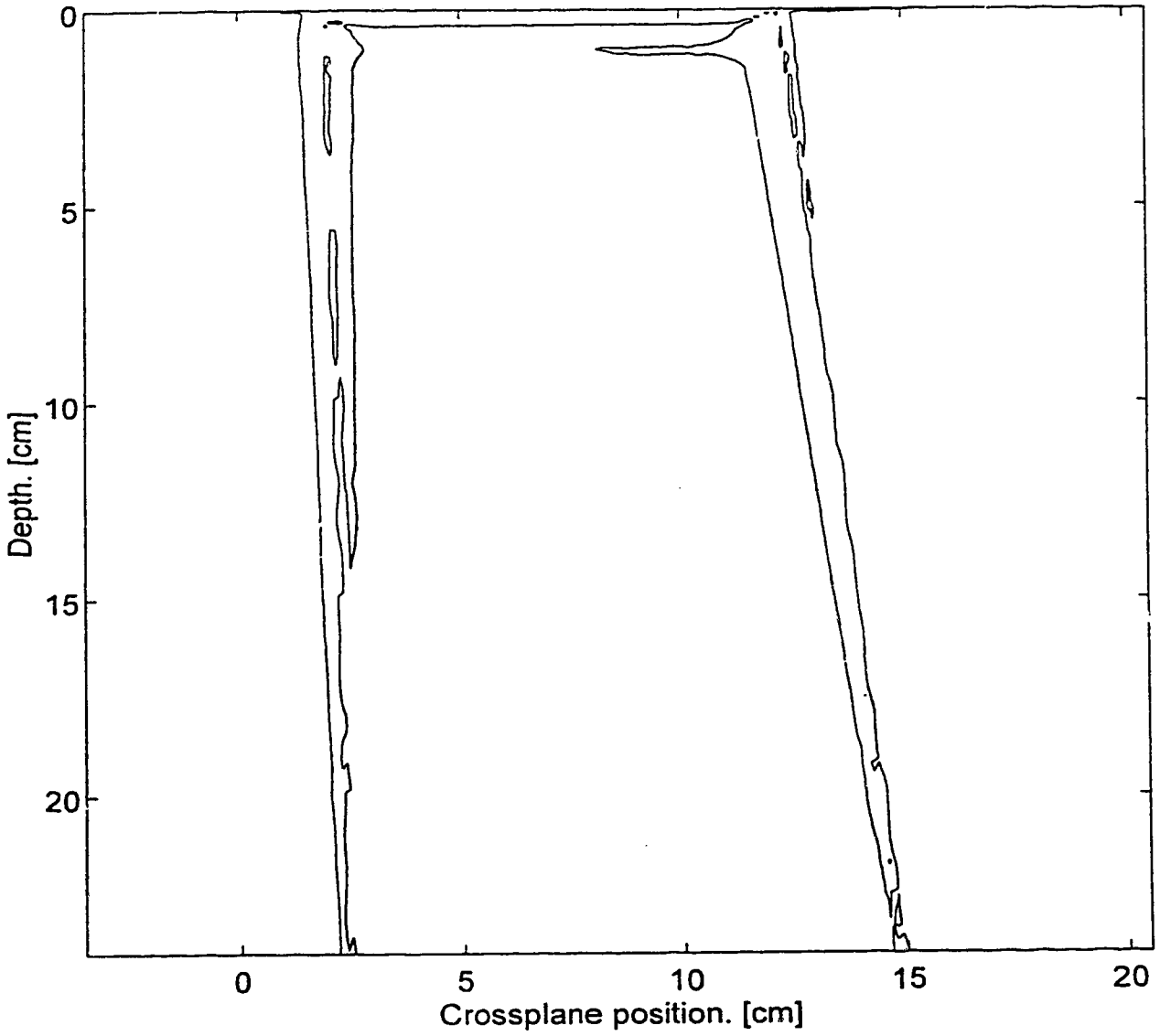


Figure 4.15b. 10x10, X=7 Y=15, XZ-iso Y=15, 3% discrepancy level.

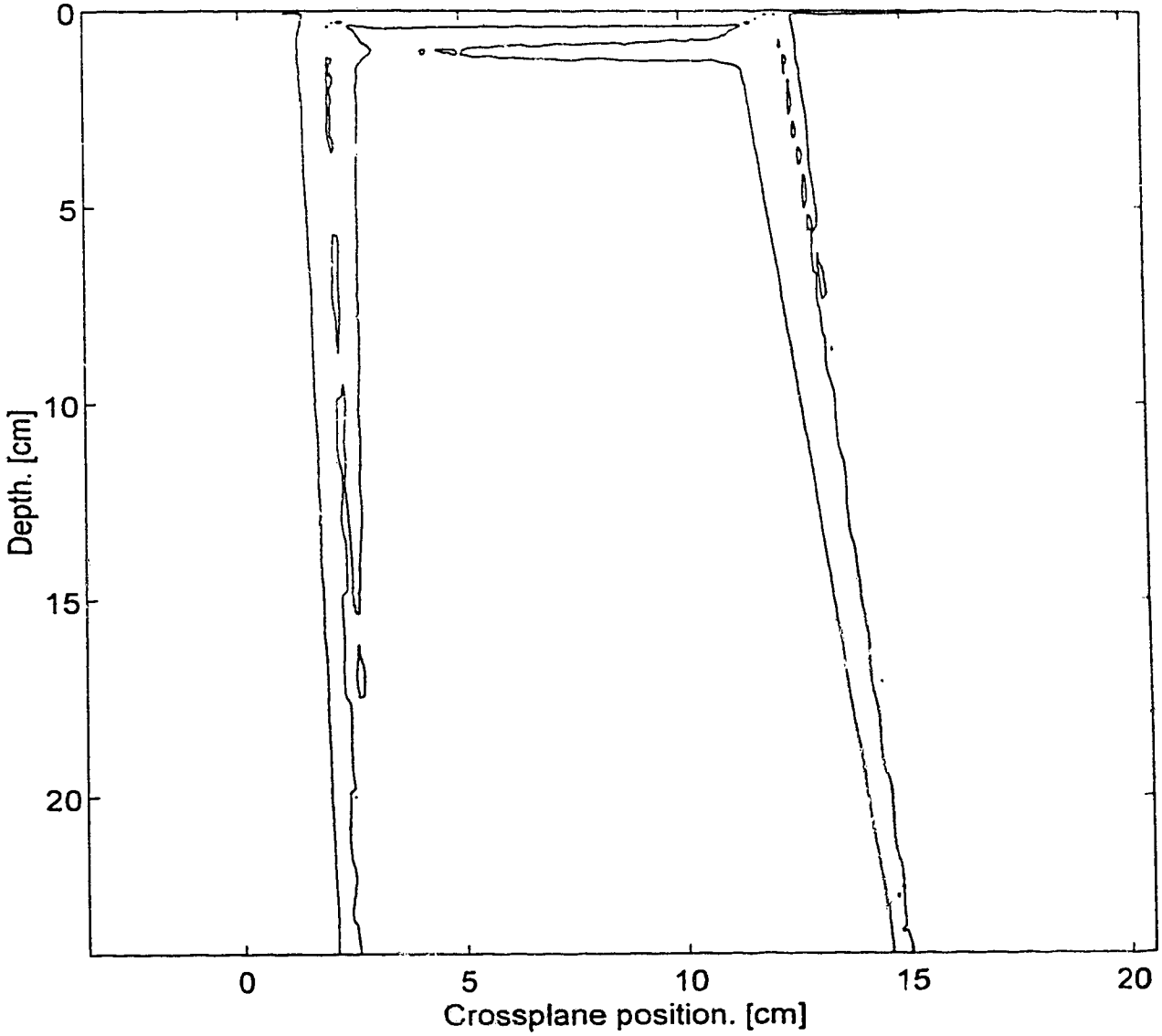


Figure 4.15c. 10x10, X=7 Y=15, XZ-iso Y=15, 2% discrepancy level.

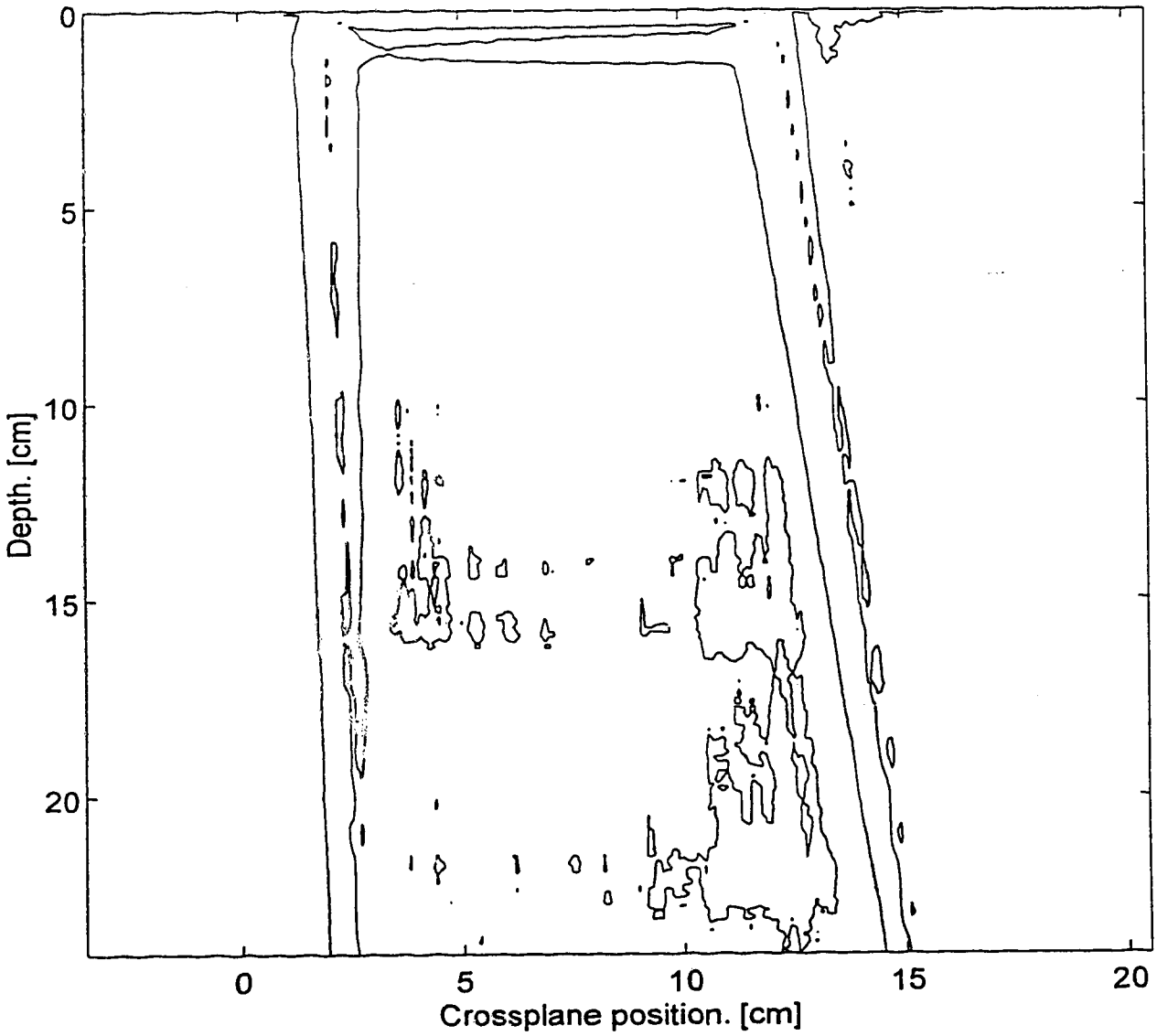
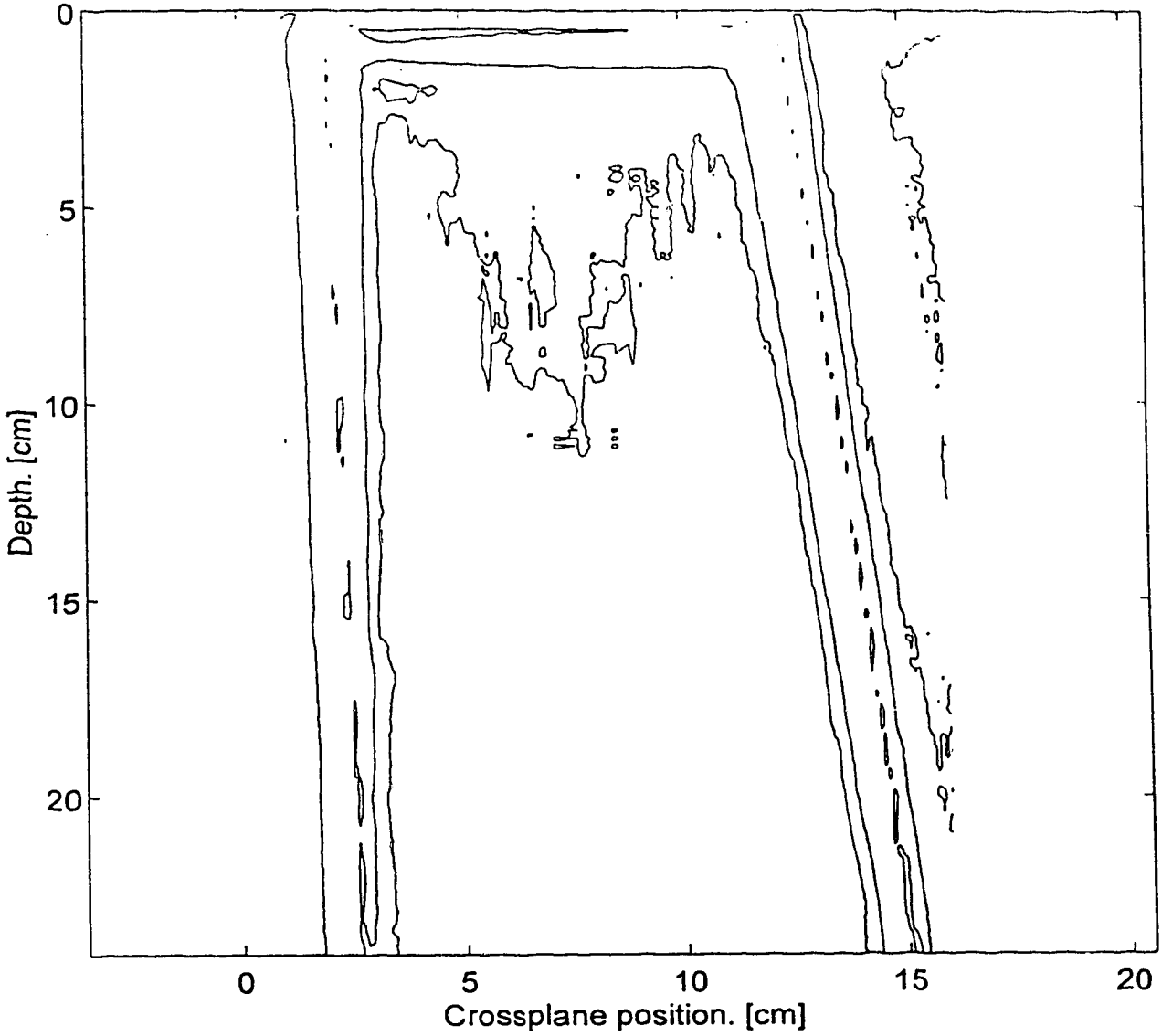


Figure 4.15d. 10x10, X=7 Y=15, XZ-iso Y=15, 1% discrepancy level.



There still appears to be increasing disagreement in the isodoses as the fields become more asymmetric. The extreme cases in figures 4.13e and 4.13f show the worst discrepancies, and plots of these discrepancies are presented in figures 4.14 and 4.15. These plots resemble the extreme single asymmetric cases in figures 4.12, with discrepancies developing in the shallow region and in the lower right area of the plot (figures 4.14c and 4.15c). As was suggested in the single asymmetric analysis, these discrepancies may be artifacts of beam-softening, and thus in figure 4.16 an experiment to test this hypothesis is presented.

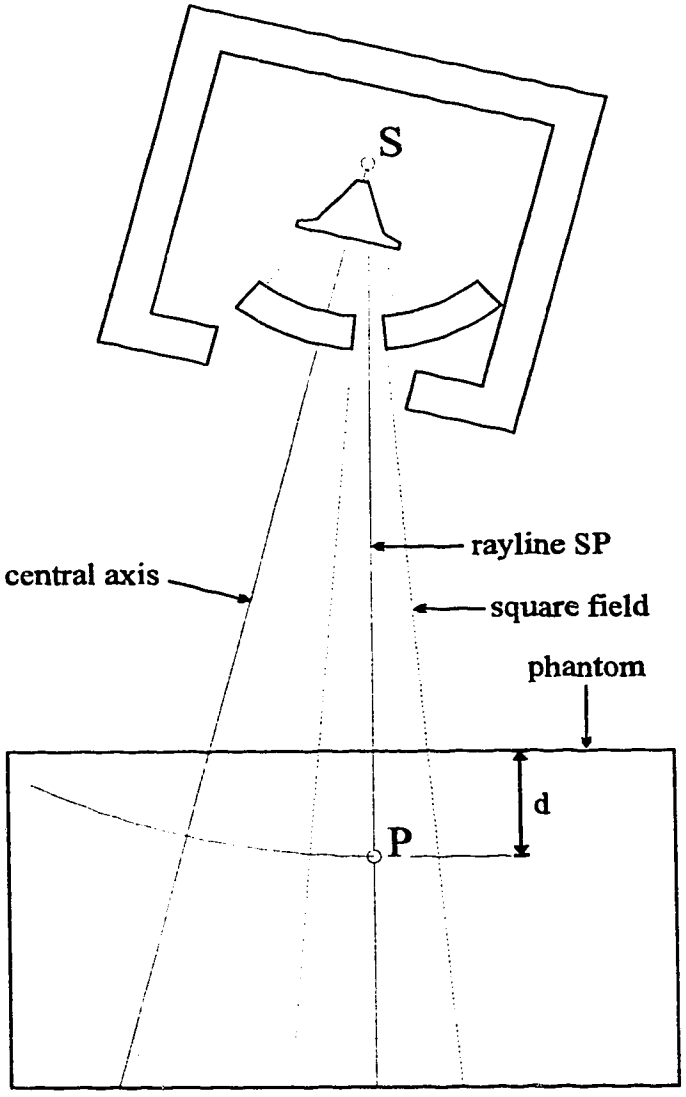


Figure 4.16. Off-axis measurement of TAR's.

The setup in figure 4.16 is designed to produce a square field on the surface of the watertank (SSD = 100 cm) with a gantry tilt of 6.5° and with the beam axis remaining vertical (i.e. radiation along the beam axis is normal to the water surface). The gantry angle was chosen to be the maximum angle that still allowed for field sizes between 3x3 cm^2 and 10x10 cm^2. That is, the greater the gantry rotation the further the collimators must be moved asymmetrically. The maximum asymmetric placement of the collimators thus defines the maximum gantry angle that will still support the desired field sizes. In the case of the linac being used for the measurements in this thesis the Y collimators can move to a maximum position of 10 cm **over** the central axis and 20 cm **away** from the central axis. Thus the maximum asymmetric position for a 10x10 field is 15 cm off the central axis, and is 11.5 cm off the central axis for the 3x3 field. It was thus the maximally asymmetric 3x3 cm^2 field that determined the gantry angle used for the measurements in figure 4.16. The setup procedure for these measurements is given below.

1) Set up and level the water tank.
2) Set SSD = 98.5 cm.
3) Position the diode on the central axis using the field light crosshairs.
4) Lower diode so that rounded top of buildup cap is just touching water surface. (i.e. The 1.5 cm lucite buildup cap is left on.)
5) Enter the diode's position as isocenter into the Wellhöfer scanning software.
6) Using the software, reposition the diode to $d = 100 \cdot (1 - \cos\theta) = 0.64$ cm and $x = 100 \cdot \sin\theta = 11.32$ cm. (Figure 4.17.)
7) Set the collimator positions for the field "3x3 X=0 Y=11.5".
8) Rotate the collimators 90° to make the Y axis the crossplane axis.
9) Rotate the gantry to $\theta = \arctan(11.5/100) \approx 6.5°$.
10) Lower the water until the top of the buildup cap is just touching the surface.
11) Recheck that the diode is in the center of the 3x3 field and compensate with the diode-positioning hand console.

Step 11 resulted in the diode being repositioned to a crossplane position of 11.4 cm, thus explaining why 11.4 cm was used here and in the extreme single asymmetric case of section 4.1.

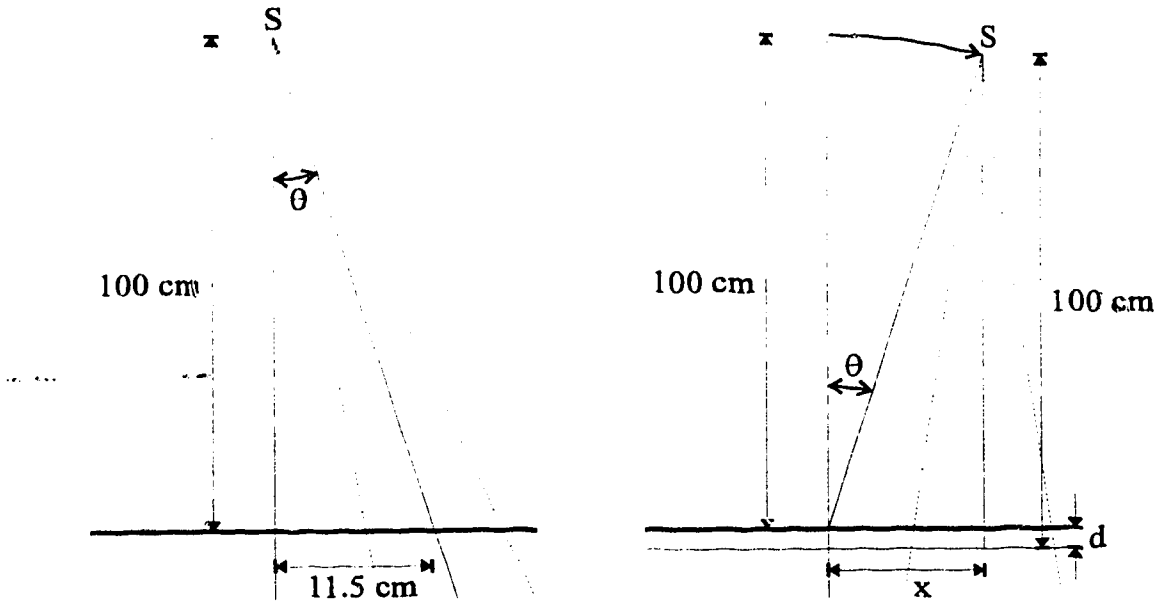


Figure 4.17. Parameters for the rotated gantry arrangement.

Using the configuration just described, percent depth doses were measured along the beam axis (i.e. vertically down the center of the field), for field sizes of 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9 and 10x10 cm^2. Peak scatter factors were also measured for these fields, and the procedure for constructing TAR's and SAR's (section 3.2) was performed on this off-axis data. The off-axis TAR's for three field sizes is shown in figure 4.18, and SAR's are shown in figure 4.19.

Figure 4.18c shows that at large depths the TAR differs by just over 2% of the maximum value. Thus there is the possibility of a significant discrepancy in calculated dose for asymmetric fields if central axis TAR data is used. This will explain the discrepancies seen, for instance, in the large depth regions of the extreme asymmetric fields in figure 4.13e and also in figure 4.12a of section 4.2. The central axis TAR data used by GRATIS overestimates the actual TAR's for these far asymmetric fields, and as a result the GRATIS isodoses are pushed down relative to the measured isodoses (i.e. GRATIS shows higher dose than measured). An easy test to verify the suggested effect on the isodoses is to recalculate the field in figure 4.12a (10x10, X=0 Y=11.4, YZ-iso X=0) using the new off-axis TAR data. This is shown in figures 4.20.

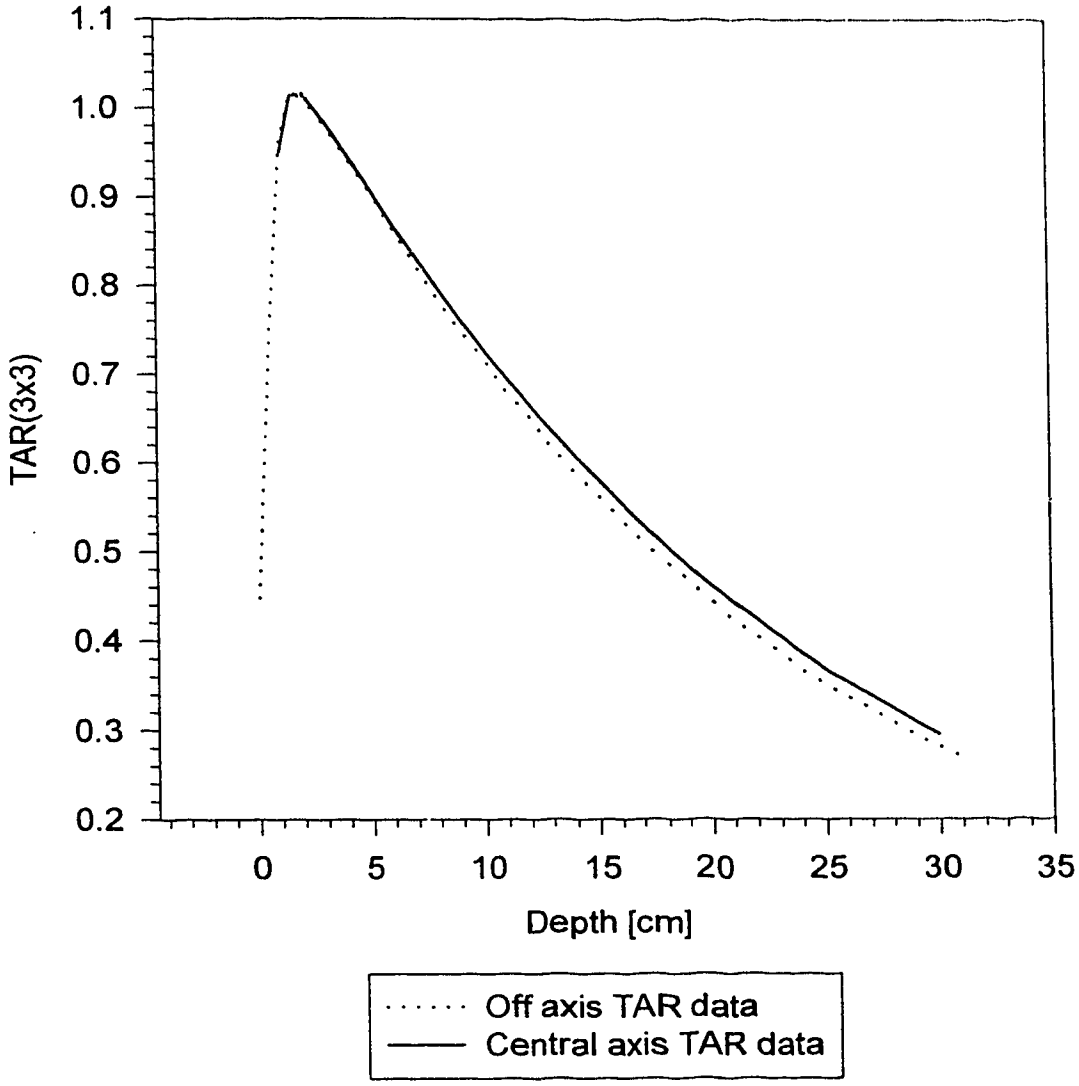


Figure 4.18a. Comparison of off-axis and central axis TAR's for a 3x3 cm^2 field.

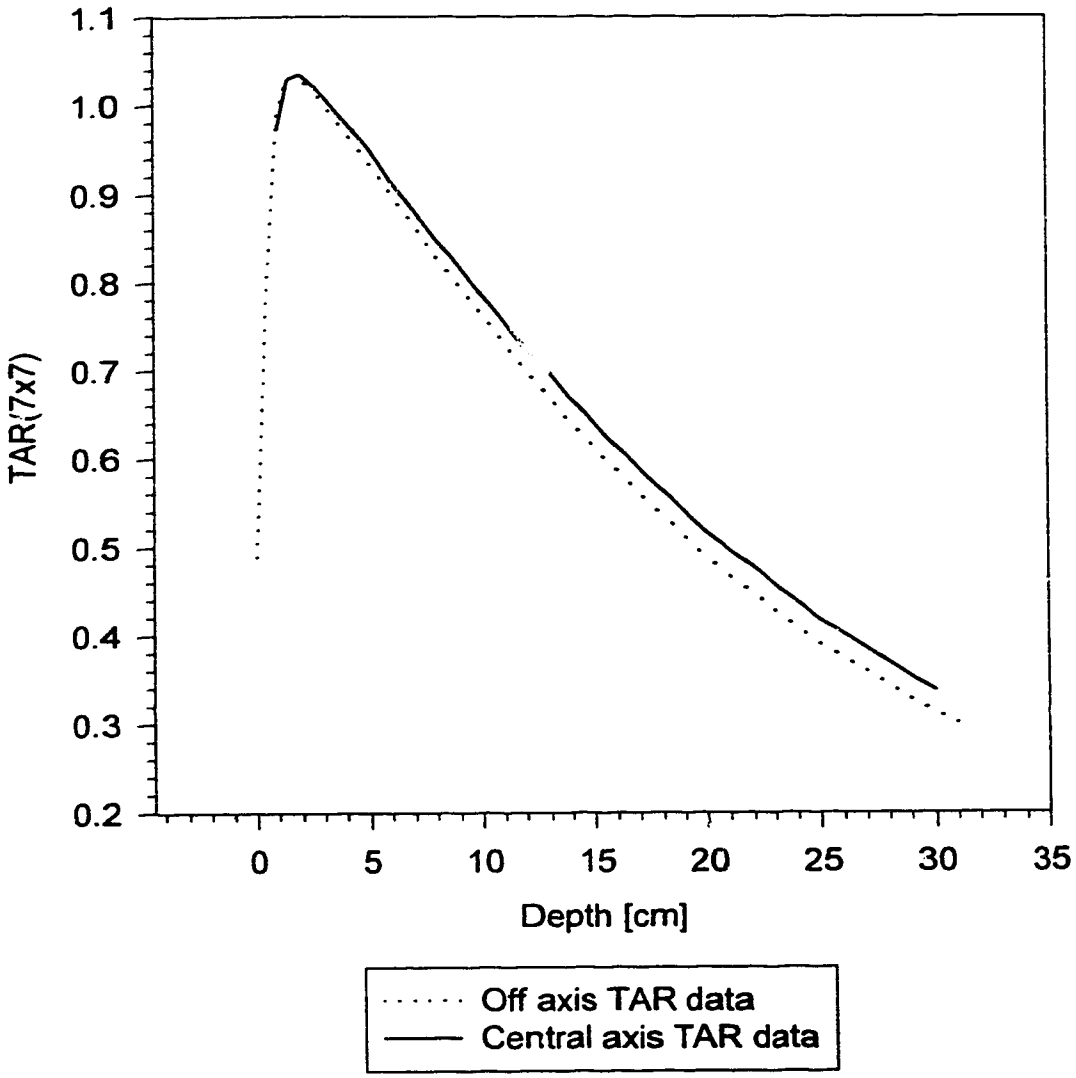


Figure 4.18b. Comparison of off-axis and central axis TAR's for a 7x7 cm^2 field.

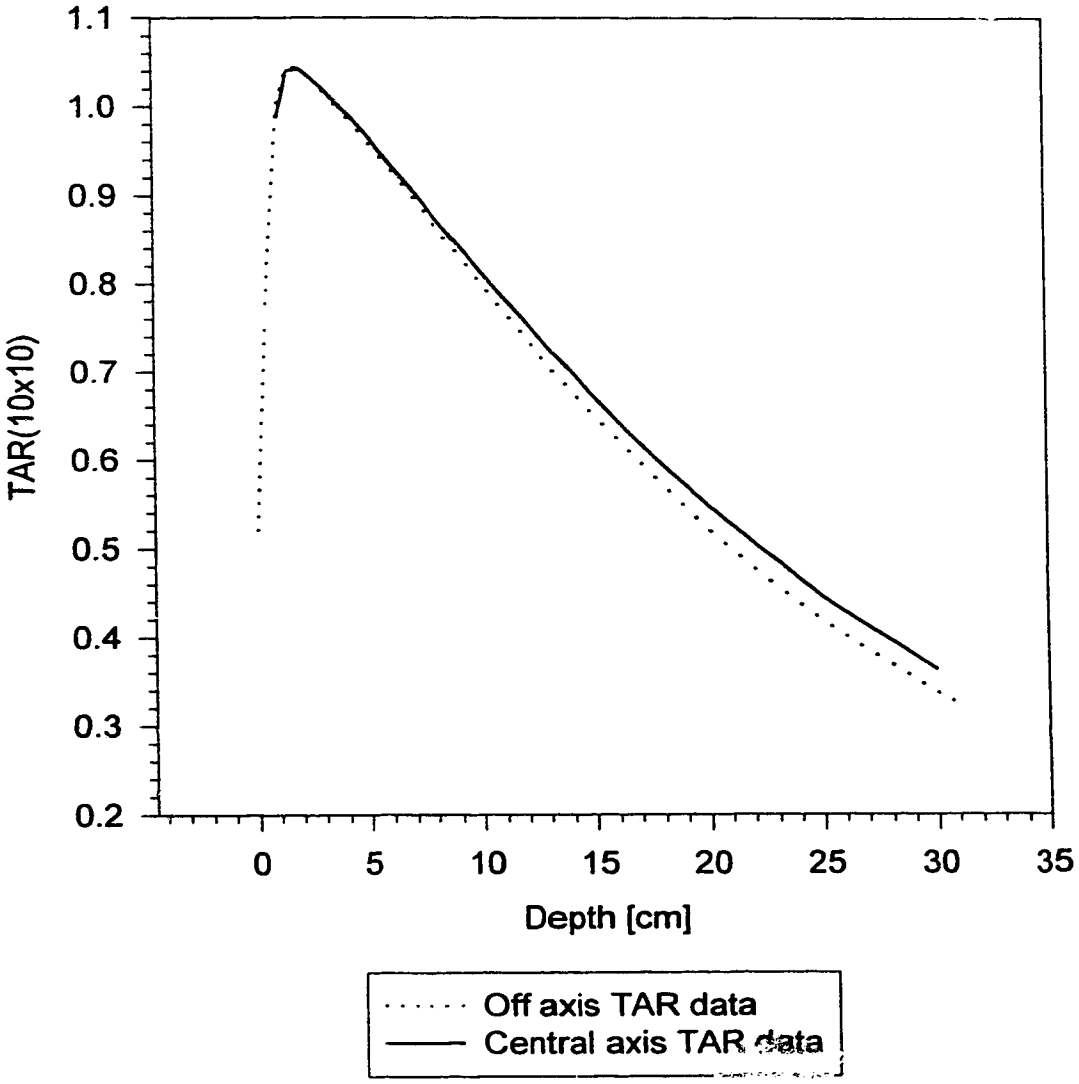


Figure 4.18c. Comparison of off-axis and central axis TAR's for a 10x10 cm^2 field.

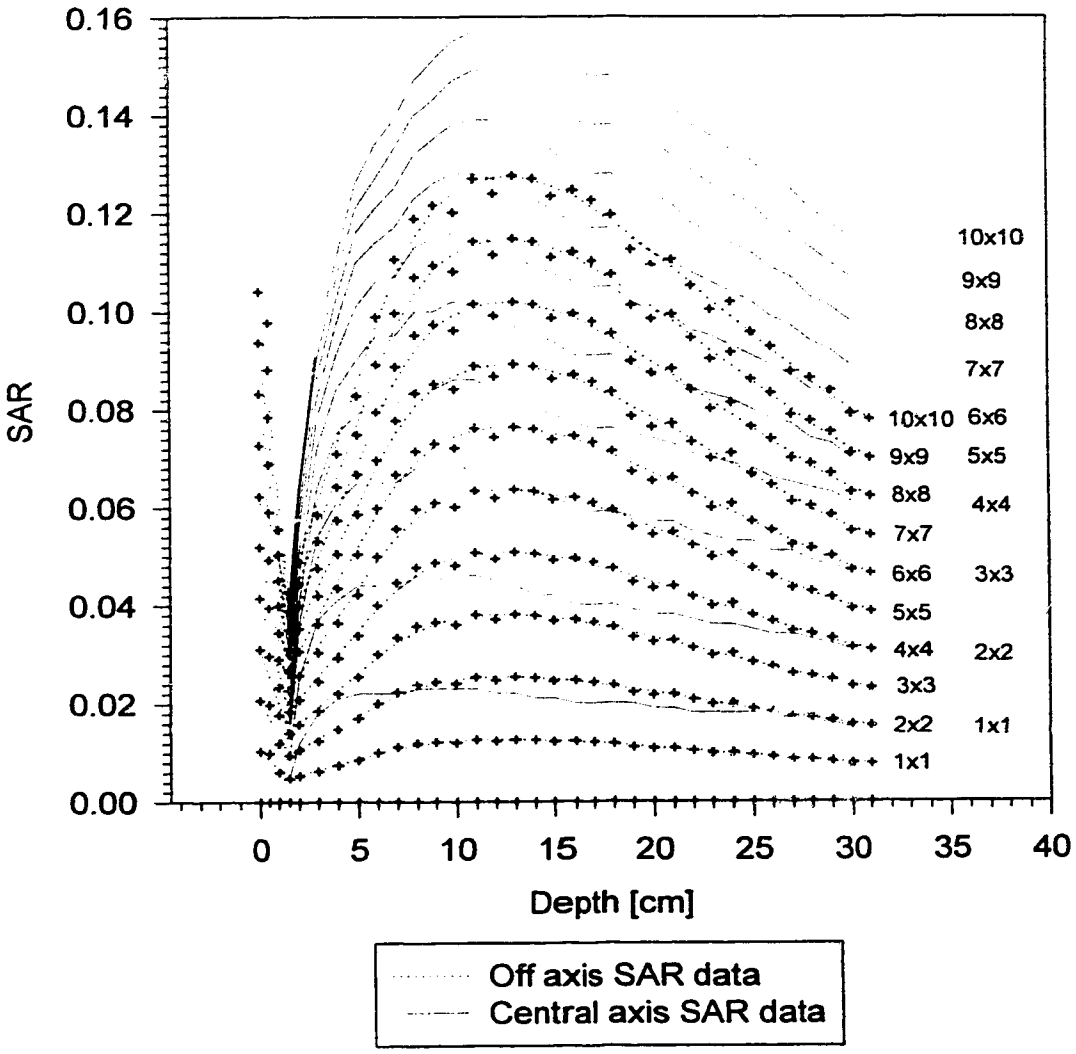


Figure 4.19. Comparison of off-axis and central axis SAR's.

Figure 4.20a. 10x10, X=0 Y=11.4, YZ-iso X=0, off-axis SAR data.

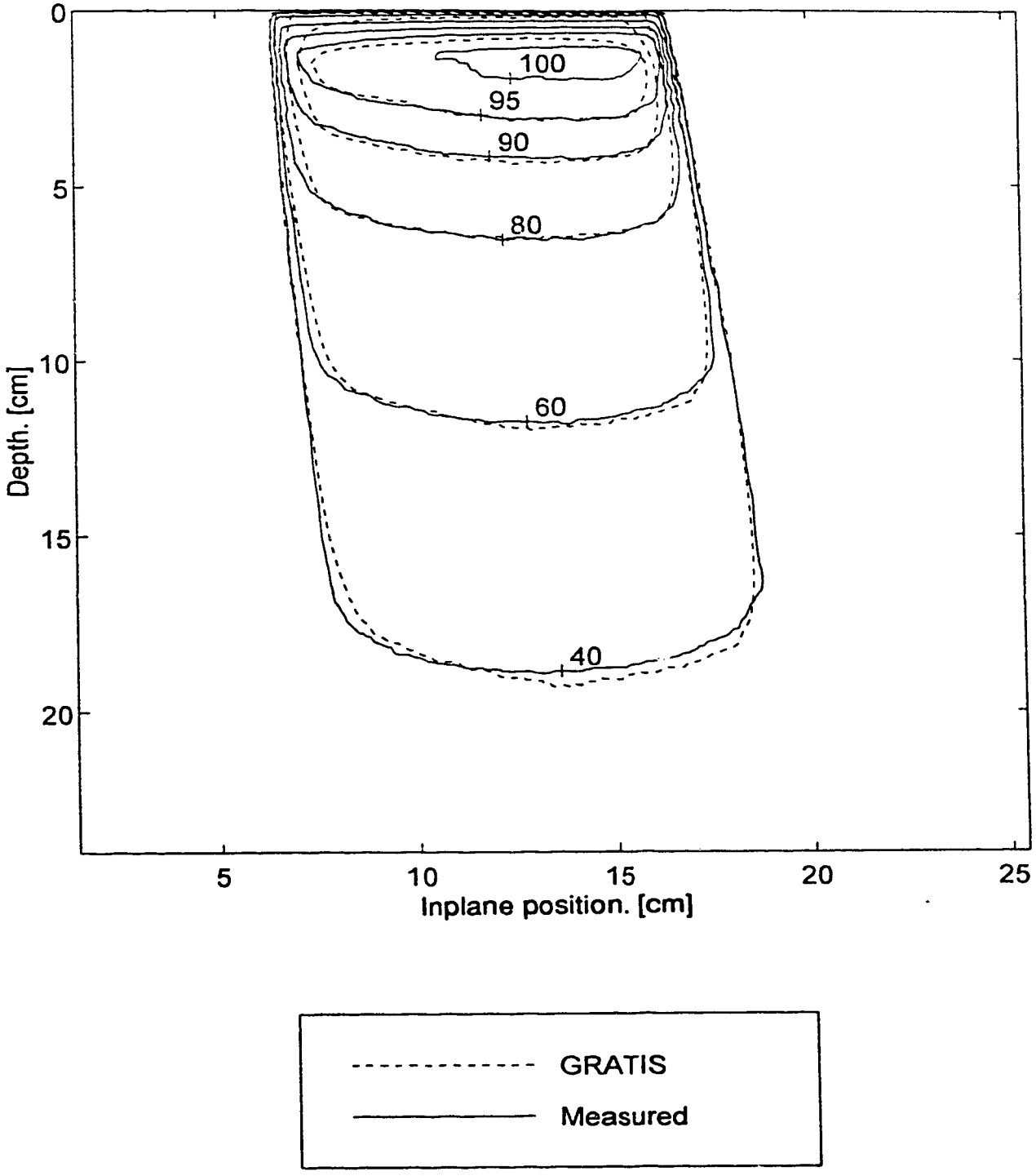


Figure 4.20b. 10x10, X=0 Y=11.4, YZ-iso X=0, 4% discrepancy level.

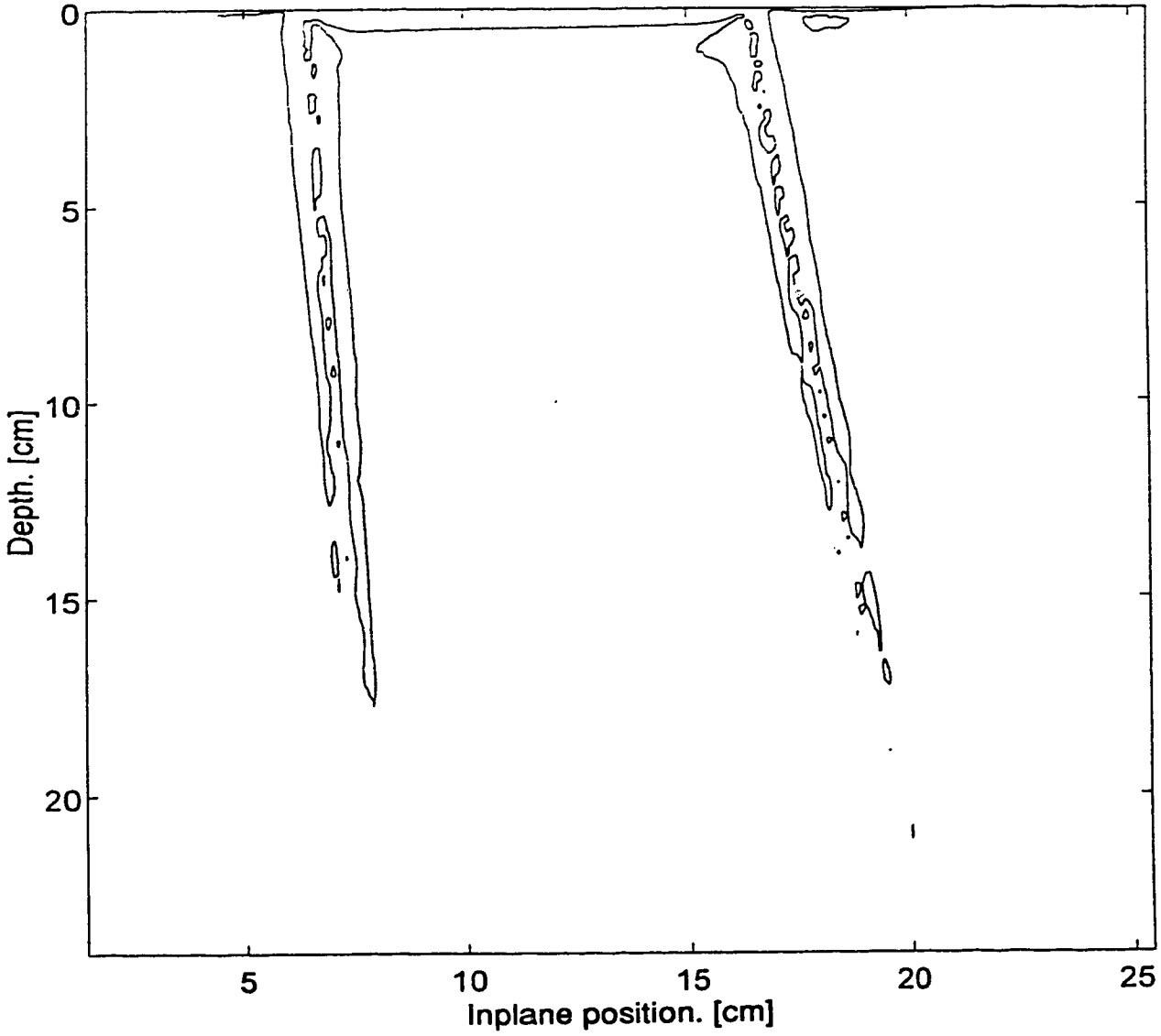


Figure 4.20c. 10x10, X=0 Y=11.4, YZ-iso X=0, 2% discrepancy level.

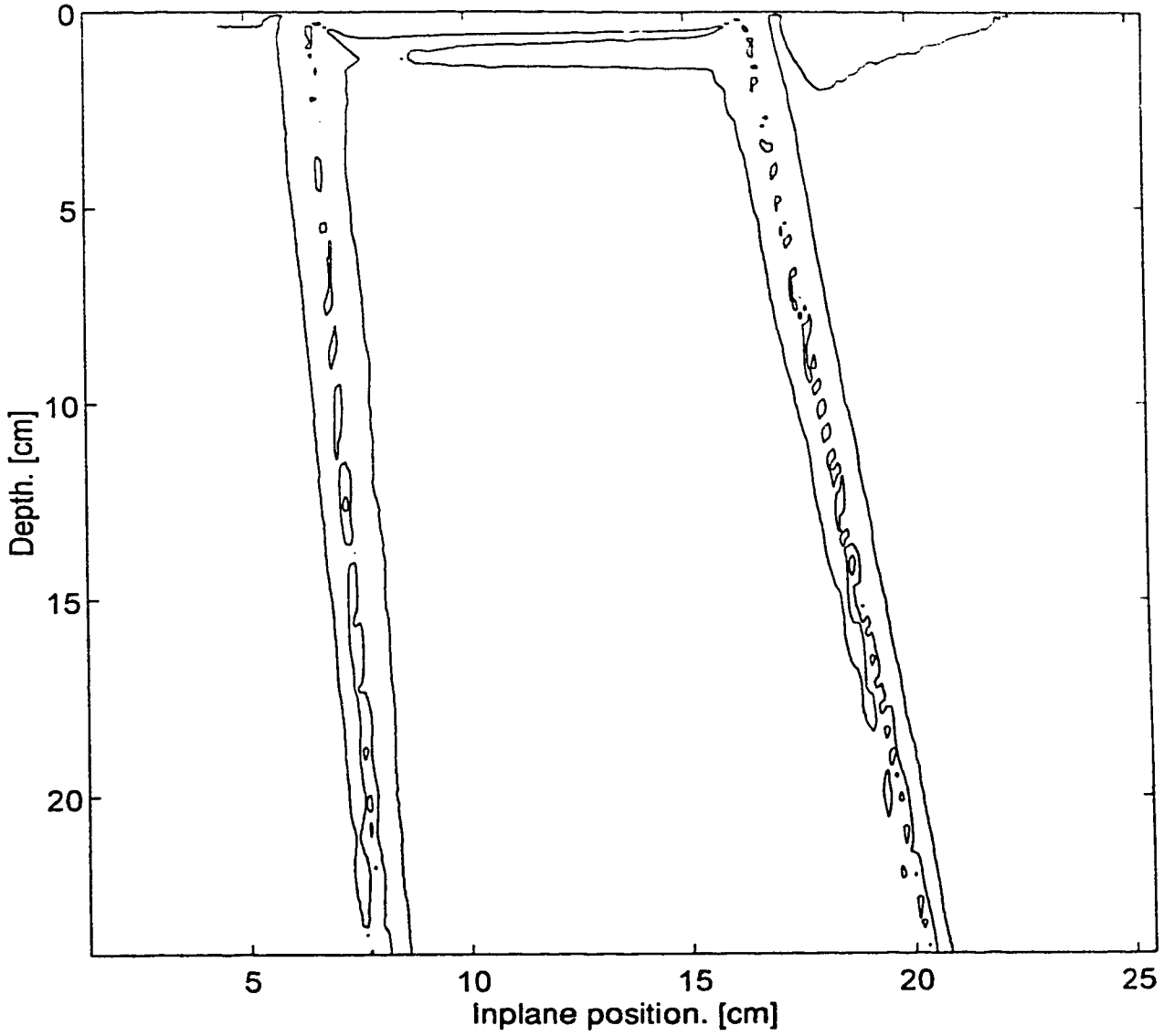
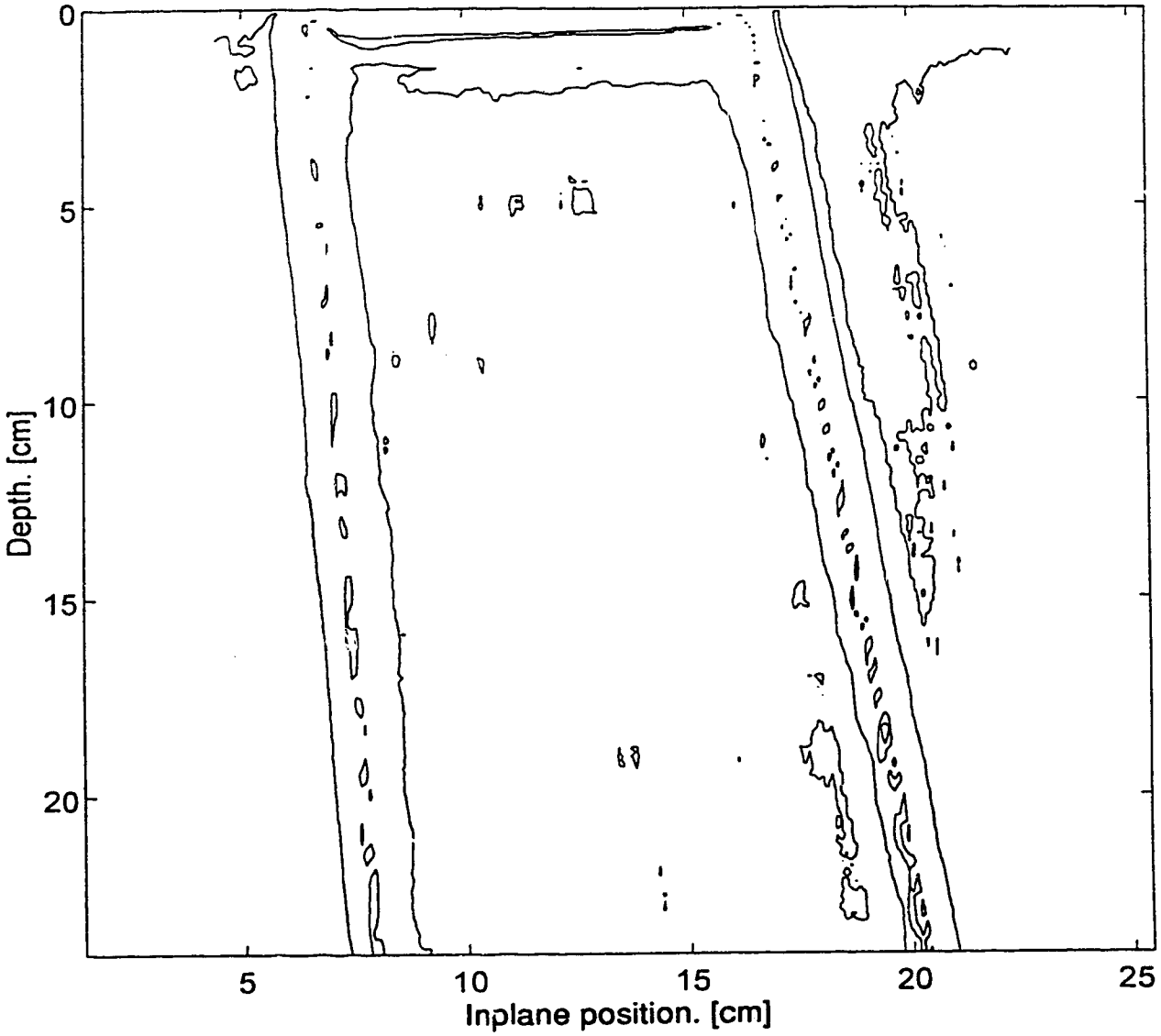


Figure 4.20d. 10x10, X=0 Y=11.4, YZ-iso X=0, 1% discrepancy level.



The measured isodoses in figure 4.20a have been normalized to 100% at a point with a depth of 1.5 cm and a crossplane position of 11.4 cm. However, the calculated dose values still seem to be in disagreement in the shallow depths, and so normalizing to a point at a depth of 1.5 cm would pull all the isodoses out of agreement. The calculated isodoses were thus normalized to a point at 3 cm depth and 11.4 cm crossplane position, to the same dose value as the corresponding point in the measured data. This normalization effectively forces the 95% isodoses to agree, rather than the 100% isodoses. The 95% isodose also agreed very well in the 10x10 X=0 Y=11.4 plot that used central axis SAR data in figure 4.12a, and thus the two figures (4.12a and 4.20a) are easily comparable. The off-axis SAR data has made a noticeable improvement in the isodose agreement for all isodoses except the 100% isodose in the shallow depth region. Also, the 1% discrepancy plot in figure 4.12d has been improved in figure 4.20d since the large region of over 1% discrepancy has been eliminated. It thus seems that beam softening with distance from the central axis has a noticeable effect on the isodoses.

5. Summary and Conclusions

In this thesis the limits of accuracy of the scatter integration method of dose calculation have been investigated and, where possible, extended to a wider range of beam geometries. Test cases for symmetric, single asymmetric and dual asymmetric fields were studied and resulted in significant modifications to the calculation algorithm as implemented in GRATIS. As well, the analysis necessitated an in depth look at the theory behind the scatter integration method (or dSAR method), resulting in a clarification of the empirical parameters used in the algorithm and more rigorous definitions.

Discrepancies between measured and calculated isodoses for symmetric fields, and the subsequent analysis of the code in GRATIS, indicated a problem with how measured SAR data was used in the reconstruction of dose. The depth and dartboard radius parameters used to look up the differential SAR data during the scatter integration were found to be inconsistent with the manner in which the SAR's were originally acquired, and this led to a redefinition of the scatter pencils. The result was a reinterpretation of the "dartboard" from one that was shifted to one that is tilted, and the changes made to the GRATIS code to reflect this new interpretation produced significant improvement in isodose agreement for large symmetric fields. It was then suggested that the remaining discrepancies in the large symmetric fields was due to beam softening with distance from the central axis, and further analysis of beam spectrum effects was deferred until more data could be collected from the asymmetric field investigations.

The tests of single asymmetric fields uncovered an assumption of symmetric fields that was originally made and documented in the GRATIS code. This assumption made the construction of the fluence and contour maps dependent upon the position of only two of the four collimators, resulting in zero scatter contribution for fields in which these two collimators crossed the central axis. The code changes to remove this assumption were relatively simple, and the single asymmetric test cases were repeated with the new version of GRATIS. The analysis that followed again suggested that beam softening effects might be responsible for the small discrepancies observed.

Dual asymmetric fields also suggested that the change in beam quality off axis might have a significant effect, so an experiment to measure off-axis SAR data and use this data in GRATIS was performed. The off-axis data resulted in a noticeable improvement in the isodose agreement for extreme asymmetric fields, indicating that the change in beam quality has a small but significant effect on the isodoses.

The experiment discussed in section 4.3 to examine the effect of beam softening on the calculation suggests that the scatter integration method could be further generalized by tabulating SAR data for beam axes other than the central axis. That is, the SAR data could be made dependent upon the distance of the calculation point from the central axis, in addition to the dependence upon field size and calculation point depth. However, the dSAR's are constructed by partitioning the circular field SAR's into sectors of equal size, and it is assumed that the scatter contribution from each sector is equal. This is true on the central axis where the fluence has radial symmetry, but around another beam axis the radial symmetry is lost and the assumption made in the dSAR construction breaks down. Thus, although the experiment in section 4.3 suffices to demonstrate the significance of beam softening, it cannot easily be used to generalize the scatter integration algorithm. A generalization of the algorithm to account for beam softening could still be made if another technique was used to construct dSAR's with dependence upon the calculation point to central axis distance,. Monte Carlo, for instance, might be used to generate dSAR's rather than constructing dSAR's from measured data.

The penumbra model was not investigated in this thesis, since the discrepancies discovered between measured and calculated isodoses were explainable by limitations in the scatter integration. Testing of the calculation routines for still more general configurations, such as beam modifying devices and arbitrary shaped fields, may uncover limitations in the penumbra model being used, in which case an improvement in the source model should be investigated.

More general situations that still remain to be investigated include contoured surfaces, tissue inhomogeneities, beam modifiers, shaped fields, extended SSDs, and different beam energies. However, the scatter integration method of dose calculation was still found to be fairly accurate for the asymmetric cases examined in this project, provided that the quantities involved are well established and controlled.

The main advantage of this method over the two other methods discussed in chapter two is its speed of calculation, and since it can be made to be reasonably accurate for asymmetric fields, the scatter integration method remains a good choice for the dose calculator in such configurations.

6. Bibliography

Anders Ahnesjö, Mikael Saxner, Avo Trepp,
A Pencil Beam Model for Photon Dose Calculation,
Medical Physics, 1992, Vol. 19, No. 2, pp. 263-273.

Anders Ahnesjö, Tommy Knöös, Anders Montelius,
Application of the Convolution Method for Calculation of Output Factors for Therapy Photon Beams,
Medical Physics, 1992, Vol. 19, No. 2, pp. 295-301.

P. Andreo,
Monte-Carlo Techniques in Medical Radiation Physics,
Physics in Medicine & Biology, 1991, Vol. 36, pp. 861-920.

Frank H. Attix,
Introduction to Radiological Physics and Radiation Dosimetry,
Copyright © 1986 by John Wiley & Sons, Inc.

K. Ayyangar, J. R. Palta, J. W. Sweet, N. Suntharalingam,
Experimental Verification of a Three-Dimensional Dose Calculation Algorithm Using a Specially Designed Heterogeneous Phantom,
Medical Physics, 1993, Vol. 20, No. 2, pp. 325-329.

J. D. Bourland, E. L. Chaney,
A Finite-Size Pencil Beam Model for Photon Dose Calculations in Three Dimensions,
Medical Physics, 1992, Vol. 19, No. 6, pp. 1401-1412.

Arthur L. Boyer, Edward C. Mok,
Calculation of Photon Dose Distributions in an Inhomogeneous Medium Using Convolutions,
Medical Physics, 1986, Vol. 13, No. 4, pp. 503-509.

Arthur L. Boyer, Edward C. Mok,
A Photon Dose Distribution Model Employing Convolution Calculations,
Medical Physics, 1985, Vol. 12, No. 2, pp. 169-177.

J. E. Burns,
Conversion of Percentage Depth Doses for Photon Beams from One SSD to Another and Calculation of TAR, TMR and TPR,
British Journal of Radiology, 1983, Supplement 17, pp. 115-119.

E. L. Chaney, T. J. Cullip,
A Monte Carlo Study of Accelerator Head Scatter,
Medical Physics, 1994, Vol. 21, No. 9, pp. 1383-1390.

Chen-Shou Chui, Radhe Mohan,
Extraction of Pencil Beam Kernels by the Deconvolution Method,
Medical Physics, 1988, Vol. 15, No. 2, pp. 138-144.

J. R. Cunningham,
Scatter-Air Ratios,
Physics in Medicine & Biology, 1972, Vol. 17, No. 1, pp. 42-51.

G. E. Desobry, A. L. Boyer,
An Analytic Calculation of the Energy Fluence Spectrum of a Linear Accelerator,
Medical Physics, 1994, Vol. 21, No. 12, pp. 1943-1952.

Ellen El-Khatib, J. J. Battista,
Improved Lung Dose Calculation Using Tissue-Maximum Ratios in the Batho Correction,
Medical Physics, 1984, Vol. 11, No. 3, pp. 279-286.

C. Field, J. J. Battista,
Photon Dose Calculations Using Convolution in Real and Fourier Space: Assumptions and Time Estimates,
The Use of Computers in Radiation Therapy, 1987, p. 103.

Tawfiq K. Haider, Ellen E. El-Khatib,
Differential Scatter Integration in Regions of Electronic Non-Equilibrium,
Physics in Medicine & Biology, 1995, Vol. 40, pp. 31-43.

Akira Iwasaki,
A Convolution Method for Calculating 10-MV X-Ray Primary and Scatter Dose Including Electron Contamination Dose,
Medical Physics, 1992, Vol. 19, No. 4, pp. 907-915.

Akira Iwasaki,
10-MV X-Ray Primary and Scatter Dose Calculation Using Convolutions,
Medical Physics, 1990, Vol. 17, No. 2, pp. 203-211.

H. E. Johns, J. R. Cunningham,
The Physics of Radiology, 4th Ed.,
Copyright © 1983 by Charles C. Thomas, Publisher.

C. J. Karzmark,
Advances in Linear Accelerator Design for Radiotherapy,
Medical Physics, 1984, Vol. 11, No. 2, pp. 105-127.

Faiz M. Khan, Bruce J. Gerbi, Firmin C. Deibel,
Dosimetry of Asymmetric X-Ray Collimators,
Medical Physics, 1986, Vol. 13, No. 6, pp. 936-941.

Faiz M. Khan, Wilfred Sewchand, Joseph Lee, Jeffrey F. Williamson,
Revision of Tissue-Maximum Ratio and Scatter-Maximum Ratio Concepts for Cobalt 60 and Higher Energy X-Ray Beams,
Medical Physics, 1980, Vol. 7, No. 3, pp. 230-237.

T. R. Mackie, J. W. Scrimger, J. J. Battista,
A Convolution Method of Calculating Dose for 15-MV X Rays,
Medical Physics, 1985, Vol. 12, No. 2, pp. 188-196.

T. R. Mackie, J. W. Scrimger,
Computing Radiation Dose for High Energy X-Rays Using a Convolution Method,
IEEE Computer Society Reprint, 1984, pp. 36-40,
Reprinted from Proceedings of the Eighth International Conference on the Use of Computers in Radiation Therapy, July 9-12, 1984.

G. Marinello, A. Dutreix,
A General Method to Perform Dose Calculations Along the Axis of Symmetrical and Asymmetrical Photon Beams,
Medical Physics, 1992, Vol. 19, No. 2, pp. 275-281.

R. Mohan, et. al.,
Three-Dimensional Dose Calculations for Radiation Treatment Planning,
International Journal of Radiation Oncology Biology Physics, 1991, Vol. 21, pp. 25-36.

Walter R. Nelson, Hideo Hirayama, David W. O. Rogers,
The EGS4 Code System
SLAC-Report-265, December 1985
Stanford Linear Accelerator Center, Stanford University, Stanford, California.

Paul S. Nizin,
Geometrical Aspects of Scatter-to-Primary Ratio and Primary Dose,
Medical Physics, 1991, Vol. 18, No. 2, pp. 153-160.

Simon J. Thomas,
A Modified Power-Law Formula for Inhomogeneity Corrections in Beams of High-Energy X Rays,
Medical Physics, 1991, Vol. 18, No. 4, pp. 719-723.

S. Webb, R. P. Parker,
A Monte Carlo Study of the Interaction of External Beam X-Radiation with Inhomogeneous Media,
Physics in Medicine & Biology, 1978, Vol. 23, No. 6, pp. 1043-1059.

John W. Wong, James A. Purdy,
On Methods of Inhomogeneity Corrections for Photon Transport,
Medical Physics, 1990, Vol. 17, No. 5, pp. 807-814.

John W. Wong, R. Mark Henkelman,
A New Approach to CT Pixel-Based Photon Dose Calculations in Heterogeneous Media,
Medical Physics, 1983, Vol. 10, No. 2, pp. 199-208.

M. K. Woo, A. Fung, P. O'Brien,
Treatment Planning for Asymmetric Jaws on a Commercial TP System,
Medical Physics, 1992, Vol. 19, No. 5, pp. 1273-1275.

7. Appendices

The following sections include information that is relevant to this thesis, but also too lengthy or technical to be presented in the main text.

Appendix A contains a mathematical analysis of the scatter integration that was originally performed to determine exactly what factors are implemented in the GRATIS code, and what factors are neglected. It also served to rigorously define the quantities used in GRATIS so that modifications to the code (and the consequences) were well understood.

Appendix B contains a listing of the C-language code for the GRATIS routine (photon_point_dose) that implements the scatter integration. Both the original version and the version with all the final modifications are presented for comparison.

Appendix C contains a listing of the C-language code for the GRATIS routine (mk_contour_map) that constructs the table of SSD's. Both the original version and the version with all the final modifications are presented for comparison.

Appendix D contains a listing of the C-language code for the GRATIS routine (mk_dSAR_table) that constructs the table of dSAR's from the SAR data. Both the original version and the version with all the final modifications are presented for comparison.

7.1. **Appendix A.** Examination of the dose calculation algorithm in GRATIS.

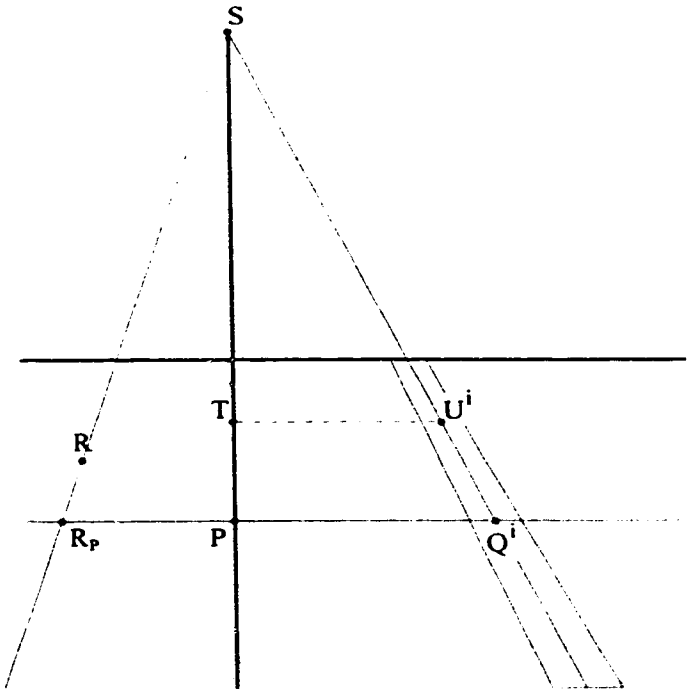


Figure 7.1. Geometry for central axis measurements.

Similar triangles.

$$\frac{SQ^i}{SP} = \frac{SU^i}{ST}$$

Definition of collimator scatter factor.

$$S_{coll}(P) = \frac{D^P_{air}}{D^P_{air,no-coll}}$$

Inverse-square relation along a rayline.

$$(ST)^2 \cdot D^T_{air,no-coll} = (SP)^2 \cdot D^P_{air,no-coll}$$

Inverse-square relation along a rayline.

$$\frac{D_{air}^{P}}{D_{air}^{T}} = \frac{(ST)^2}{(SP)^2} \cdot \frac{S_{coll}(P)}{S_{coll}(T)}$$

Definition of beam intensity (relative to P).

$$\text{rel\_intensity}_P(R) = \frac{D_{air}^{R}}{D_{air}^{P}}$$

Definition of beam flatness (relative to P).

$$\text{flatness}_P(R) = \text{rel\_intensity}_P(R_P) = \frac{D_{air}^{R_P}}{D_{air}^{P}}$$

Expression of the differential-scatter-air-ratio as a ratio of doses.
(For use in the derivation of pt_SAR.)

$$\Delta SAR(P,Q^i) = \frac{\left(D_{scat_i}^{P} \Big/ \text{rel\_intensity}_P(Q^i) \right)}{D_{air}^{P}}$$

$$= \frac{D_{scat_i}^{P}}{D_{air}^{P}} \cdot \frac{1}{\text{rel\_intensity}_P(Q^i)}$$

$$= \frac{D_{scat_i}^{P}}{D_{air}^{P}} \cdot \frac{1}{\left(D_{air}^{Q^i} \Big/ D_{air}^{P} \right)}$$

$$= \frac{D_{scat_i}^{P}}{D_{air}^{Q^i}}$$

Note that $Q^i = Q_P^i$ when ΔSAR is generated in GRATIS, so that in the above we have

$$\text{rel\_intensity}_P(Q^i) = \text{flatness}_P(Q^i)$$

$$\begin{aligned}
\text{rel_intensity}_P(R) &= \frac{D_{air}^{R}}{D_{air}^{P}} \\
&= \frac{D_{air}^{R}}{D_{air}^{R_P}} \cdot \frac{D_{air}^{R_P}}{D_{air}^{P}} \\
&= \frac{D_{air}^{R}}{D_{air}^{R_P}} \cdot \text{rel_intensity}_P(R_P) \\
&= \frac{D_{air}^{R}}{D_{air}^{R_P}} \cdot \text{flatness}_P(R) \\
&= \frac{(SR_P)^2}{(SR)^2} \cdot \frac{S_{coll}(R)}{S_{coll}(R_P)} \cdot \text{flatness}_P(R) \\
&= \frac{(SP)^2}{(SR_z)^2} \cdot \frac{S_{coll}(R)}{S_{coll}(R_P)} \cdot \text{flatness}_P(R)
\end{aligned}$$

Which gives us the following substitutions for the pt_TAR_o and pt_SAR calculations.

$$\text{rel\_intensity}_C(A) = \frac{(SC)^2}{(SA_z)^2} \cdot \frac{S_{coll}(A)}{S_{coll}(A_c)} \cdot \text{flatness}_C(A)$$

$$\text{rel\_intensity}_C(B^i) = \frac{(SC)^2}{(SB_z^i)^2} \cdot \frac{S_{coll}(B^i)}{S_{coll}(B_c^i)} \cdot \text{flatness}_C(B^i)$$

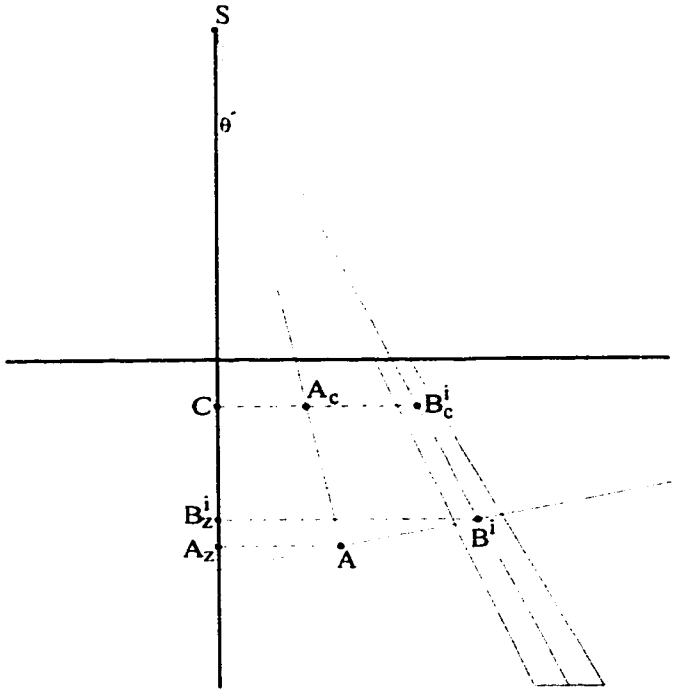


Figure 7.2. Geometry for off-axis calculations.

Dose calculation algorithm used by GRATIS.

$$
\begin{aligned}
D_{total}^{A} &= D_{prim}^{A} + D_{scat}^{A} \\
&= D_{air}^{C} \cdot \left\{ \frac{D_{prim}^{A}}{D_{air}^{C}} + \frac{D_{scat}^{A}}{D_{air}^{C}} \right\} \\
&= \left\{ D_{air}^{C} \cdot (SC)^2 \right\} \cdot \left\{ \frac{1}{(SC)^2} \cdot \frac{D_{prim}^{A}}{D_{air}^{C}} + \frac{1}{(SC)^2} \cdot \frac{D_{scat}^{A}}{D_{air}^{C}} \right\} \\
&= \{global\_factor\} \cdot \{pt\_TAR_0 + pt\_SAR\} \\
&= \{global\_factor\} \cdot \{photon\_point\_dose\}
\end{aligned}
$$

$$
D_{air}^{C} = S_{coll}(C) \cdot D_{air,no-coll}^{C} = \{lookup\_Sc\} \cdot \{cal\_dose\_rate\}
$$

$$
\begin{aligned}
\mathrm{pt\_TAR}_0 &= \frac{1}{(\mathrm{SC})^2}\cdot\frac{\mathrm{D}^{\mathrm{A}}_{\mathrm{prim}}}{\mathrm{D}^{\mathrm{C}}_{\mathrm{air}}} \\
&= \frac{1}{(\mathrm{SC})^2}\cdot\frac{\mathrm{D}^{\mathrm{A}}_{\mathrm{air}}}{\mathrm{D}^{\mathrm{C}}_{\mathrm{air}}}\cdot\frac{\mathrm{D}^{\mathrm{A}}_{\mathrm{prim}}}{\mathrm{D}^{\mathrm{A}}_{\mathrm{air}}} \\
&= \frac{1}{(\mathrm{SC})^2}\cdot \mathrm{rel\_intensity}_{\mathrm{C}}(\mathrm{A})\cdot \mathrm{TAR}_0(\mathrm{A}) \\
&= \frac{1}{(\mathrm{SC})^2}\cdot\left[\frac{(\mathrm{SC})^2}{(\mathrm{SA}_z)^2}\cdot\frac{\mathrm{S}_{\mathrm{coll}}(\mathrm{A})}{\mathrm{S}_{\mathrm{coll}}(\mathrm{A}_c)}\cdot \mathrm{flatness}_{\mathrm{C}}(\mathrm{A})\right]\cdot \mathrm{TAR}_0(\mathrm{A}) \\
&= \frac{1}{(\mathrm{SA}_z)^2}\cdot\frac{\mathrm{S}_{\mathrm{coll}}(\mathrm{A})}{\mathrm{S}_{\mathrm{coll}}(\mathrm{A}_c)}\cdot \mathrm{flatness}_{\mathrm{C}}(\mathrm{A})\cdot \mathrm{TAR}_0(\mathrm{A})
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{pt\_SAR} &= \frac{1}{(\mathrm{SC})^2}\cdot\frac{\mathrm{D}^{\mathrm{A}}_{\mathrm{scat}}}{\mathrm{D}^{\mathrm{C}}_{\mathrm{air}}} \\
&= \frac{1}{(\mathrm{SC})^2}\cdot\frac{1}{\mathrm{D}^{\mathrm{C}}_{\mathrm{air}}}\cdot\sum_i \mathrm{D}^{\mathrm{A}}_{\mathrm{scat}_i} \\
&= \frac{1}{(\mathrm{SC})^2}\cdot\sum_i\left\{\frac{\mathrm{D}^{\mathrm{B}^i}_{\mathrm{air}}}{\mathrm{D}^{\mathrm{C}}_{\mathrm{air}}}\cdot\frac{\mathrm{D}^{\mathrm{A}}_{\mathrm{scat}_i}}{\mathrm{D}^{\mathrm{B}^i}_{\mathrm{air}}}\right\} \\
&= \frac{1}{(\mathrm{SC})^2}\cdot\sum_i\left\{\mathrm{rel\_intensity}_{\mathrm{C}}(\mathrm{B}^i)\cdot\Delta\mathrm{SAR}(\mathrm{A},\mathrm{B}^i)\right\} \\
&= \frac{1}{(\mathrm{SC})^2}\cdot\sum_i\left\{\left[\frac{(\mathrm{SC})^2}{(\mathrm{SB}^i_z)^2}\cdot\frac{\mathrm{S}_{\mathrm{coll}}(\mathrm{B}^i)}{\mathrm{S}_{\mathrm{coll}}(\mathrm{B}^i_c)}\cdot\mathrm{flatness}_{\mathrm{C}}(\mathrm{B}^i)\right]\cdot\Delta\mathrm{SAR}(\mathrm{A},\mathrm{B}^i)\right\} \\
&= \sum_i\left\{\frac{1}{(\mathrm{SB}^i_z)^2}\cdot\frac{\mathrm{S}_{\mathrm{coll}}(\mathrm{B}^i)}{\mathrm{S}_{\mathrm{coll}}(\mathrm{B}^i_c)}\cdot\mathrm{flatness}_{\mathrm{C}}(\mathrm{B}^i)\cdot\Delta\mathrm{SAR}(\mathrm{A},\mathrm{B}^i)\right\}
\end{aligned}
$$

GRATIS currently makes the implicit assumption that S_{coll} does not vary significantly along a rayline.

i.e. $$\frac{S_{coll}(A)}{S_{coll}(A_c)} \cong 1 \qquad \text{and} \qquad \frac{S_{coll}(B^i)}{S_{coll}(B^i_c)} \cong 1$$

The calculations currently performed by GRATIS are then

$$pt\_TAR_0 = \frac{1}{(SA_z)^2} \cdot flatness_C(A) \cdot TAR_0(A)$$

$$pt\_SAR = \sum_i \left\{ \frac{1}{(SB^i_z)^2} \cdot flatness_C(B^i) \cdot \Delta SAR(A, B^i) \right\}$$

$$D^C_{air} = S_{coll}(C) \cdot D^C_{air,no-coll} = \{lookup\_Sc\} \cdot \{cal\_dose\_rate\}$$

$$D^A_{total} = \{global\_factor\} \cdot \{pt\_TAR_0 + pt\_SAR\}$$

Variation of beam flatness profile with distance from source.

$$\frac{D_{air}^{Q^i}}{D_{air}^{P}} = \left[\frac{D_{air}^{Q^i}\cdot(SQ^i)^2}{S_{coll}(Q^i)}\right]\cdot\left[\frac{S_{coll}(P)}{D_{air}^{P}\cdot(SP)^2}\right]\cdot\frac{(SP)^2}{(SQ^i)^2}\cdot\frac{S_{coll}(Q^i)}{S_{coll}(P)}$$

$$= \left[\frac{D_{air}^{U^i}\cdot(SU^i)^2}{S_{coll}(U^i)}\right]\cdot\left[\frac{S_{coll}(T)}{D_{air}^{T}\cdot(ST)^2}\right]\cdot\frac{(SP)^2}{(SQ^i)^2}\cdot\frac{S_{coll}(Q^i)}{S_{coll}(P)}$$

$$= \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\left(\frac{SU^i}{ST}\frac{SP}{SQ^i}\right)^2\cdot\frac{S_{coll}(Q^i)}{S_{coll}(U^i)}\cdot\frac{S_{coll}(T)}{S_{coll}(P)}$$

$$= \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\left(\frac{SQ^i}{SP}\frac{SP}{SQ^i}\right)^2\cdot\frac{S_{coll}(Q^i)}{S_{coll}(U^i)}\cdot\frac{S_{coll}(T)}{S_{coll}(P)}$$

$$= \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\frac{S_{coll}(Q^i)}{S_{coll}(U^i)}\cdot\frac{S_{coll}(T)}{S_{coll}(P)}$$

or $$\frac{D_{air}^{Q^i}}{D_{air}^{P}}\cdot\frac{S_{coll}(P)}{S_{coll}(Q^i)} = \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\frac{S_{coll}(T)}{S_{coll}(U^i)}$$

If $S_{coll}(Q^i) = S_{coll}(U^i)$ (i.e. if collimator scatter is not dependent upon distance from source) then

$$\frac{D_{air}^{Q^i}}{D_{air}^{P}} = \frac{D_{air}^{U^i}}{D_{air}^{T}}$$

7.2. **Appendix B.** Code listing of photon_point_dose.c

This appendix contains a listing of the code for the GRATIS routine photon_point_dose, which contains the scatter integration. Both the original version and the version with all the final modifications are listed side by side to facilitate comparisons. Also, the code changes that are relevant to the tilting of the dartboard are shown in bold to set them apart from changes that involve debugging statements, documentation and reformatting of C code.

photon_point_dose.c.v0

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/photon_point_dose.c.v 3.3
        1990/10/14 16:25:58 sherouse Exp $";

/*
 * $Log: photon_point_dose.c.v $
 * Revision 3.3  1990/10/14  16:25:58  sherouse
 * - use library version of exact_fluence()
 *
 * Revision 3.2  89/01/26  16:26:13  sherouse
 * - more fiddling around with points on upstream side of patient
 *
 * Revision 3.1  89/01/18  14:45:04  sherouse
 * - support new beam description
 * - add filters
 * - make it OK to calculate dose at 0 depth
 *
 * Revision 2.2  88/05/31  17:23:01  sherouse
 * - spruced up documentation (just a tad)
```

photon_point_dose.c.v3

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/photon_point_dose.c.v 3.3
        1990/10/14 16:25:58 sherouse Exp $";

/*
 * $Log: photon_point_dose.c.v $
 *
 * Revision 3.6  1994/07/27  11:30:00  jkollar
 * - Document the debugging switches and clean them up a little.
 *
 * Revision 3.5  1994/07/06  15:58:00  jkollar
 * - Change the inverse square scaling to use the z-coordinate of the
 *   calculation point, rather than the rayline distance from the source.
 *   We feel that the z-distance is the correct one to use here.
 *   This inverse square scaling is required to adjust the final dose
 *   from being at the reference distance (which is where the tables and
 *   maps are defined) to being at the correct calculation point distance.
 * - Add an "inverse square" correction to the sector fluence that makes
 *   the fluence be relative to the calculation point, rather than relative
 *   to the calibration point (as the original fluence table is defined).
 *   We believe that this is the proper way to weight the scatter pencils,
 *   giving us the total SAR at the calculation point.
 *
 * Revision 3.4  1994/02/12  17:27:00  jkollar
 * - Tip the dartboard to improve the algorithm's accuracy.
 * - Dartboard is now normal to the source-calc_pt rayline,
 * - rather than normal to the central axis.
 *
 * Revision 3.3  1990/10/14  16:25:58  sherouse
 * - use library version of exact_fluence()
 *
 * Revision 3.2  89/01/26  16:26:13  sherouse
 * - more fiddling around with points on upstream side of patient
 *
 * Revision 3.1  89/01/18  14:45:04  sherouse
 * - support new beam description
 * - add filters
 * - make it OK to calculate dose at 0 depth
 *
 * Revision 2.2  88/05/31  17:23:01  sherouse
 * - spruced up documentation (just a tad)
```

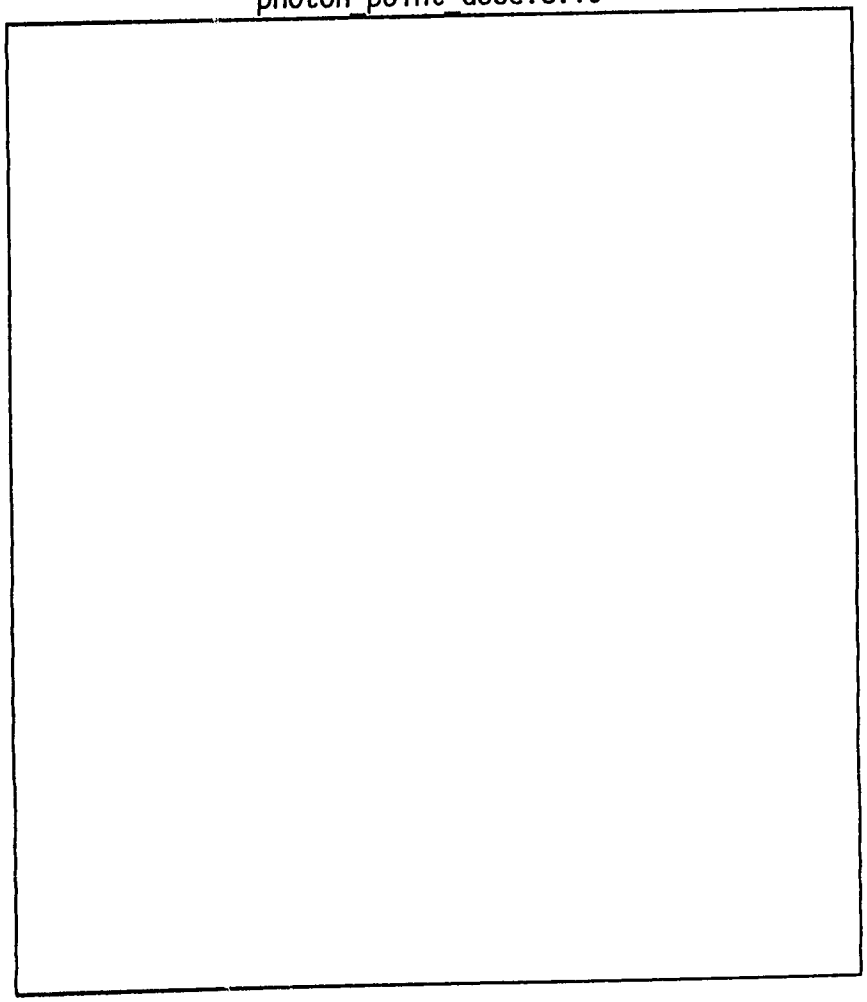
photon_point_dose.c.v0

```
 * Revision 2.1  88/05/31  09:43:20  sherouse
 * - miraculous transformation into a near-production version
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define SQRT(x) ((float) sqrt((double)(x)))
#define exact_TAR0(x) v_interp(0, request.SAR_table.depth_count,\
                                  request.SAR_table.depth_vec, (x),\
                                  request.SAR_table.TAR0, &index, &fx)
```

photon_point_dose.c.v3

```
 * Revision 2.1  88/05/31  09:43:20  sherouse
 * - miraculous transformation into a near-production version
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define SQRT(x) ((float) sqrt((double)(x)))
#define exact_TAR0(x) v_interp(0, request.SAR_table.depth_count,\
                                  request.SAR_table.depth_vec, (x),\
                                  request.SAR_table.TAR0, &index, &fx)

/*****************************************************************************/
/* Debugging switches.
*/

/* George Sherouse */
/* Print a message informing when a grid point is outside the patient.       */
/* #define EBUG1 */

/* George Sherouse */
/* Print a message of the form "T?R0 for depth ?.?? is ??.??" where the '?'s */
/* are determined just after the primary has been calculated, but before the */
/* scatter integration has begun.                                            */
/* #define EBUG2 */

/* John R. Kollar */
/* Print out the BOOLEAN variable "inside" and the float variables           */
/* "nearest", "point_depth" and "pathlength", for each calculation point.    */
/* These variables are the result of a call to:                              */
/*    prism (&request.patient_anastruct, patient_source, patient_pt,         */
/*           &inside, &nearest, &point_depth, &pathlength);                  */
/* #define J_DEBUG1 */
```

photon_point_dose.c.v0



photon_point_dose.c.v3

```
/* John R. Kollar */
/* Print out the int variables "rloop" & "ang_loop", and the float          */
/* variables "tables.dSAR_radii[rloop]" & "sec_fluence", for each scatter   */
/* pencil.                                                                  */
/* John R. Kollar W9407.27 10:00 */
/* Print out the float variables "sec_depth_along_pt" and "sec_z_dr..n" for  */
/* each scatter pencil.                                                     */
/* Print out the POINT variable "sc" and the float variable "sc_dist" for   */
/* each scatter pencil.                                                     */
/* #define J_DEBUG2 */

/* John R. Kollar */
/* Print out the int variables "tables.dSAR_sector_count" and               */
/* "tables.dSAR_radius_count" for each calculation point.                   */
/* #define J_DEBUG3 */

/* John R. Kollar */
/* Print the float values of "pt_TAR0", "pt_SAR" & "pt_TAR0 + pt_SAR" just  */
/* before the inverse-square term is multiplied in.                         */
/* These values are printed out for each calculation point.                 */
/* #define J_DEBUG4 */

/* John R. Kollar */
/* Print out the float variable "pt_TAR0" and the POINT variable "beam_pt"  */
/* (i.e. the calculation point) just before the inverse-square term is      */
/* multiplied in. These values are printed out for each calculation point.  */
/* #define J_DEBUG5 */

/* John R. Kollar T9407.26 15:00 */
/* Use a sector fluence of 1.0 in the scatter integration. (But not in the  */
/* primary calculation.) This results in pt_SAR = sum(dSAR*1.0).            */
/* #define J_DEBUG6 */
```

photon_point_dose.c.v0

```
float photon_point_dose (patient_source, patient_pt, pt_TARO, pt_SAR)
POINT patient_source;
POINT patient_pt;
float *pt_TARO;
float *pt_SAR;
```

photon_point_dose.c.v3

```
/* John R. Kollar */
/* Print out the int variable "request.SAR_table.depth_count", the float     */
/* variables "point_dist" & "sec_SSD", the float value returned by the       */
/* function call                                                             */
/*    v_interp (0, request.SAR_table.depth_count,                            */
/*                 request.SAR_table.depth_vec, sec_depth_along_pt,          */
/*                 tables.dSAR[rloop], &index, &fx).                         */
/* and the resulting values of the int variable "index" and the float        */
/* variable "fx" immediately following the above function call.              */
/* These values are printed for each scatter pencil.                         */
/* #define J_DEBUG7 */

/* John R. Kollar W9407.27 10:23 */
/* Print out the float variable "request.unit.SAD" and the POINT variable    */
/* "beam_pt" for each calculation point.                                     */
/* #define J_DEBUG8 */

/* John R. Kollar T9408.02 14:37 */
/* Print out the int variables "contour_map_x" and "contour_map_y" for each  */
/* scatter pencil.                                                           */
/* #define J_DEBUG9 */

/* John R. Kollar F9408.05 14:01 */
/* Print out the "exact_fluence" used in the primary calculation, for each   */
/* calculation point.                                                        */
/* #define J_DEBUG10 */

/*****************************************************************************/

float photon_point_dose (patient_source, patient_pt, pt_TARO, pt_SAR)
POINT patient_source;
POINT patient_pt;
float *pt_TARO;
float *pt_SAR;
```

photon_point_dose.c.v0

```
/*
-----------------------------------------------------------------------

  NAME
        photon_point_dose - here's where the work gets done

  SYNOPSIS

  DESCRIPTION
        This routine actually looks up the zero-area T*R, does the
        sector integration of S*R, and scales everything by the
        appropriate inverse square.

  RETURN VALUE
                                                  2
        (T*R0 + S*R) / (point distance)   Note that this is not really
        dose as the name implies but is well on the way.

  BUGS

  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        26 May 1988


        (c) Copyright 1988 - George W. Sherouse
        All rights reserved.

-----------------------------------------------------------------------
*/
```

photon_point_dose.c.v3

```
/*
-----------------------------------------------------------------------

  NAME
        photon_point_dose - here's where the work gets done

  SYNOPSIS

  DESCRIPTION
        This routine actually looks up the zero-area T*R, does the
        sector integration of S*R, and scales everything by the
        appropriate inverse square.

  RETURN VALUE
                                                  2
        (T*R0 + S*R) / (point distance)   Note that this is not really
        dose as the name implies but is well on the way.

  BUGS

  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        26 May 1988


        (c) Copyright 1988 - George W. Sherouse
        All rights reserved.

  SUBSEQUENT DEVELOPERS
        John R. Kollar
        Department of Medical Physics
        Cross Cancer Institute
        11560 University Avenue
        Edmonton, Alberta T6G-1Z2
        Canada
        Phone: 403-492-6776
        Email: jkollar@phys.ualberta.ca
-----------------------------------------------------------------------
*/
```

photon_point_dose.c.v0

```
{
    extern REQUESTrequest.
    extern TABLES tables;

    static float  map_max = ((float) MAP_SIZE) / 4.0;

    POINTbeam_pt;
    floatpoint_dist_sq.
                dist_scale.
                point_dist.
                point_depth.
                nearest.
                pathlength.
                x. y.
                sec_SSD.
                sec_fluence.
                fx;









    BOOLEAN       inside;
    int           ang_loop.
                  rloop.
                  fluence_map_x.
                  fluence_map_y.
                  contour_map_x.
                  contour_map_y.
                  index;

    float     v_interp (). exact_fluence ();
```

photon_point_dose.c.v3

```
{
    extern REQUESTrequest;
    extern TABLES tables.

    static float  map_max = ((float) MAP_SIZE) / 4.0.

    POINTbeam_pt.
    floatpoint_dist_sq.
                dist_scale.
                point_dist.
                point_depth.
                nearest.
                pathlength.
                x. y.
                sec_SSD.
                sec_fluence.
                fx;

    /* John R Kollar Sa9402.12 16:45 */
    double      ang1, ang2.
                C1, S1, C2, S2, C3.
                rot[3][3];
    float       sec_depth_along_pt.
                sec_z_depth.
                sc_dist.
                dist_scale2;
    POINT       sc;

    /* John R. Kollar F9408.05 14:01 */
    float       pt_fluence;

    BOOLEAN       inside;
    int           ang_loop.
                  rloop.
                  fluence_map_x.
                  fluence_map_y.
                  contour_map_x.
                  contour_map_y.
                  index;

    float     v_interp (). exact_fluence ();
```

photon_point_dose.c.v0

```
/*
  Transform the patient coordinates of the grid point into beam
  coordinates and intersect the associated ray with the patient.

  The inside argument is wasted here - we know the source point is not
  inside the patient.
*/
    transform_point (patient_pt, request.beam.T_pat_to_beam, &beam_pt);
    prism (&request.patient_anastruct, patient_source, patient_pt,
            &inside, &nearest, &point_depth, &pathlength);









/*
  Nearest should be set to a very large number if the grid point is
  outside the patient, either upstream or beside.  We currently do not
  flag points outside the patient on the downstream side for fear of
  generating artifactual build-down.  This is a matter of ongoing
  evaluation.
*/
    if (nearest > 1000.0)
    {
#ifdef EBUG
        printf ("grid point outside patient\n");
#endif EBUG
        *pt_TAR0 = 0.0;
        *pt_SAR = 0.0;
        return (0.0);
    }
```

photon_point_dose.c.v3

```
/*
  Transform the patient coordinates of the grid point into beam
  coordinates and intersect the associated ray with the patient.

  The inside argument is wasted here - we know the source point is not
  inside the patient.
*/
    transform_point (patient_pt, request.beam.T_pat_to_beam, &beam_pt);
    prism (&request.patient_anastruct, patient_source, patient_pt,
            &inside, &nearest, &point_depth, &pathlength);

#ifdef J_DEBUG1
    printf("\n");
    printf("inside = %u\n",inside);
    printf("nearest = %f\n",nearest);
    printf("point_depth = %f\n",point_depth);
    printf("pathlength = %f\n",pathlength);
#endif J_DEBUG1

/*
  Nearest should be set to a very large number if the grid point is
  outside the patient, either upstream or beside.  We currently do not
  flag points outside the patient on the downstream side for fear of
  generating artifactual build-down.  This is a matter of ongoing
  evaluation.
*/
    if (nearest > 1000.0)
    {
#ifdef EBUG1
        printf ("grid point outside patient\n");
#endif EBUG1
        *pt_TAR0 = 0.0;
        *pt_SAR = 0.0;
        return (0.0);
    }

#ifdef J_DEBUG8
    printf("SAD = %f\n",request.unit.SAD);
    printf("beam_pt = (%f,%f,%f)\n",beam_pt.x,beam_pt.y,beam_pt.z);
#endif J_DEBUG8
```

photon_point_dose.c.v0

```
   /* factor to scale beam x and y coordinates to unit isocenter */
    * which is where all the tables are defined.                  */
   dist_scale = request.unit.SAD / beam_pt.z;

   point_dist_sq = beam_pt.x * beam_pt.x +
                   beam_pt.y * beam_pt.y +
                   beam_pt.z * beam_pt.z;
   point_dist = SQRT (point_dist_sq);

   *pt_TAR0 = exact_TAR0 (point_depth) *
        exact_fluence (beam_pt, point_depth, dist_scale,
                       (request.shaped_field) ?
                       request.tray.block_dist :
                       request.unit.SDD,
                       &request.beam, &request.unit,
                       &request.SAR_table, &request.filter);


#ifdef EBUG
    printf ("%s0 for depth %f is %f\n",
            T_THING (request.SAR_table.type),
            point_depth,
            exact_TAR0 (point_depth));
#endif EBUG
```

photon_point_dose.c.v3

```
   /* factor to scale beam x and y coordinates to unit isocenter */
   /* which is where all the tables are defined.                  */
   dist_scale = request.unit.SAD / beam_pt.z;

   point_dist_sq = beam_pt.x * beam_pt.x +
                   beam_pt.y * beam_pt.y +
                   beam_pt.z * beam_pt.z;
   point_dist = SQRT(point_dist_sq);

   pt_fluence = exact_fluence (beam_pt, point_depth, dist_scale,
                       (request.shaped_field) ?
                       request.tray.block_dist :
                       request.unit.SDD,
                       &request.beam, &request.unit,
                       &request.SAR_table, &request.filter);
   *pt_TAR0 = exact_TAR0 (point_depth) * pt_fluence;

#ifdef EBUG2
    printf ("%s0 for depth %f is %f\n",
            T_THING (request.SAR_table.type),
            point_depth,
            exact_TAR0 (point_depth));
#endif EBUG2

#ifdef J_DEBUG10
    printf("exact_fluence(%f,%f,%f) = %f\n",beam_pt.x,beam_pt.y,beam_pt.z,
            pt_fluence);
#endif J_DEBUG10
```

photon_point_dose.c.v0

```
*pt_SAR = 0.0;

for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
{
     /*
      * The sectors at rloop=0 are not interesting as dSAR==0.0.  Hence we
      * start rloop at 1. See mk_dSAR_table.
      */
     for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
     {
         x = (beam_pt.x + tables.sec_center_x[ang_loop][rloop]) *
              dist_scale;
         y = (beam_pt.y + tables.sec_center_y[ang_loop][rloop]) *
              dist_scale;
```

photon_point_dose.c.v3

```
   /* John R Kollar Sa9402.12 14:31 */
   /* Create rotation matrix for tipping the dartboard. */
   ang1 = atan2(beam_pt.y,beam_pt.x);
   ang2 = atan2(SQRT(beam_pt.x*beam_pt.x + beam_pt.y*beam_pt.y),beam_pt.z);
   C1 = cos(ang1); S1 = sin(ang1);
   C2 = cos(ang2); S2 = sin(ang2);
   C3 = C2 - 1;
   rot[0][0] = C1*C1*C3 + 1;
   rot[1][0] = C1*S1*C3;
   rot[2][0] = -C1*S2;
   rot[0][1] = rot[1][0];
   rot[1][1] = S1*S1*C3 + 1;
   rot[2][1] = -S1*S2;
   /* rot[0][2] = -rot[2][0]; */
   /* rot[1][2] = -rot[2][1]; */
   /* rot[2][2] = C3 + 1;     */

   *pt_SAR = 0.0;

#ifdef J_DEBUG3
    printf("\n");
    printf("sector count = %d\n",tables.dSAR_sector_count);
    printf("radius count = %d\n",tables.dSAR_radius_count);
#endif J_DEBUG3

    for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
    {
        /*
         * The sectors at rloop=0 are not interesting as dSAR==0.0.  Hence we
         * start rloop at 1. See mk_dSAR_table.
         */
        for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        {
          /* John R Kollar Sa9402.12 15:02 */
          /* sc = sector center for the tipped dartboard. */
          sc.x = beam_pt.x + rot[0][0]*tables.sec_center_x[ang_loop][rloop]
                           + rot[0][1]*tables.sec_center_y[ang_loop][rloop];
          sc.y = beam_pt.y + rot[1][0]*tables.sec_center_x[ang_loop][rloop]
                           + rot[1][1]*tables.sec_center_y[ang_loop][rloop];
          sc.z = beam_pt.z + rot[2][0]*tables.sec_center_x[ang_loop][rloop]
                           + rot[2][1]*tables.sec_center_y[ang_loop][rloop];
```

photon_point_dose.c.v0

```
/*
  This needs some proving but it sure speeds things up...
*/
            if (rloop > 2)
            {
                 if ((x > request.beam.jaw[X_JAW].pos_2) ||
                     (x < request.beam.jaw[X_JAW].pos_1))
                     continue;
                 if ((y > request.beam.jaw[Y_JAW].pos_2) ||
                     (y < request.beam.jaw[Y_JAW].pos_1))
                     continue;
            }

/*
  Compute index into contour map (recall it is only filled at the
  center) and into the fluence map (recall it is fleshed out).

  In both cases, if the current point is beyond the edge of the map,
  extrapolate by using the map edge value.
*/
            contour_map_y = fluence_map_y =
                 (int) ((map_max - y) * 2.0);




            if (fluence_map_y < 0)
                 fluence_map_y = 0;
            else
            if (fluence_map_y >= MAP_SIZE)
                 fluence_map_y = MAP_SIZE - 1;
```

photon_point_dose.c.v3

```
            sc_dist = SQRT(sc.x*sc.x + sc.y*sc.y + sc.z*sc.z);
            /* Project sc back to isocenter for table lookup of */
            /* patient contour, fluence and filter values. */
            dist_scale2 = request.unit.SAD/sc.z;
            x = dist_scale2*sc.x;
            y = dist_scale2*sc.y;

/*
  This needs some proving but it sure speeds things up...
*/
            if (rloop > 2)
            {
                 if ((x > request.beam.jaw[X_JAW].pos_2) ||
                     (x < request.beam.jaw[X_JAW].pos_1))
                     continue;
                 if ((y > request.beam.jaw[Y_JAW].pos_2) ||
                     (y < request.beam.jaw[Y_JAW].pos_1))
                     continue;
            }

/*
  Compute index into contour map (recall it is only filled at the
  center) and into the fluence map (recall it is fleshed out).

  In both cases, if the current point is beyond the edge of the map,
  extrapolate by using the map edge value.
*/
            contour_map_y = fluence_map_y =
                 (int) ((map_max - y) * 2.0);

#ifdef J_DEBUG9
            printf("contour_map_y = %d    ",contour_map_y);
#endif J_DEBUG9

            if (fluence_map_y < 0)
                 fluence_map_y = 0;
            else
            if (fluence_map_y >= MAP_SIZE)
                 fluence_map_y = MAP_SIZE - 1;
```

photon_point_dose.c.v0

```
            if (contour_map_y > tables.con_map_max_y_index)
                 contour_map_y = tables.con_map_max_y_index;
            else
            if (contour_map_y < tables.con_map_min_y_index)
                 contour_map_y = tables.con_map_min_y_index;

            contour_map_x = fluence_map_x =
                 (int) ((x + map_max) * 2.0);




            if (fluence_map_x < 0)
                 fluence_map_x = 0;
            else
            if (fluence_map_x >= MAP_SIZE)
                 fluence_map_x = MAP_SIZE - 1;

            if (contour_map_x > tables.con_map_max_x_index)
                 contour_map_x = tables.con_map_max_x_index;
            else
            if (contour_map_x < tables.con_map_min_x_index)
                 contour_map_x = tables.con_map_min_x_index;


/*
  See if there is any patient under this sector.
*/
            sec_SSD = tables.contour_map[contour_map_y][contour_map_x];

            if (sec_SSD == 0.0)
                 continue;
```

photon_point_dose.c.v3

```
            if (contour_map_y > tables.con_map_max_y_index)
                 contour_map_y = tables.con_map_max_y_index;
            else
            if (contour_map_y < tables.con_map_min_y_index)
                 contour_map_y = tables.con_map_min_y_index;

            contour_map_x = fluence_map_x =
                 (int) ((x + map_max) * 2.0);


#ifdef J_DEBUG9
            printf("contour_map_x = %d\n",contour_map_x);
#endif J_DEBUG9

            if (fluence_map_x < 0)
                 fluence_map_x = 0;
            else
            if (fluence_map_x >= MAP_SIZE)
                 fluence_map_x = MAP_SIZE - 1;

            if (contour_map_x > tables.con_map_max_x_index)
                 contour_map_x = tables.con_map_max_x_index;
            else
            if (contour_map_x < tables.con_map_min_x_index)
                 contour_map_x = tables.con_map_min_x_index;


/*
  See if there is any patient under this sector.
*/
            sec_SSD = tables.contour_map[contour_map_y][contour_map_x];

            if (sec_SSD == 0.0)
                 continue;

           /* John R Kollar Sa9402.12 15:27                */
           /* Depth values required for table lookup of */
           /* dSAR and fluence values.                     */
           sec_depth_along_pt = point_dist - sec_SSD*point_dist/sc_dist;
           sec_z_depth = sc.z - sec_SSD*sc.z/sc_dist;
```

photon_point_dose.c.v0

```
            if (sec_SSD >= point_dist)
                continue;

/*
  Can't avoid it - look up nearest neighbor fluence and dSAR for depth
  corresponding to nearest neighbor SSD.
*/
            sec_fluence =
                (request.SAR_table.flatness_depth_count == 1) ?
                    tables.fluence_map[fluence_map_y][fluence_map_x][0] :
                    v_interp (0, request.SAR_table.flatness_depth_count,
                        request.SAR_table.flatness_depth_vec,
                        point_dist - sec_SSD,
                        tables.fluence_map[fluence_map_y][fluence_map_x],
                        &index, &fx);
```

photon_point_dose.c.v3

```
            /* John R Kollar M9411.14 12:00                                  */
            /* This is a fudge to try and account for a slowly varying       */
            /* patient contour. The idea is that the differential scatter    */
            /* not only depends upon the "depth" of the scatter pencil,      */
            /* but also on the amount of tissue above the calculation point. */
            /* That is, the dSAR table was constructed from measured data    */
            /* that includes the contribution from multiple scatter, and     */
            /* hence the dose at the calculation point depends on scattering */
            /* from surrounding phantom material other than the pencil.      */
            /* (The pencil can be thought of as the "source" of 1st          */
            /* scattered radiation).                                         */
            /* sec_depth_along_pt = (sec_depth_along_pt + point_depth)/2.0;  */

            /* John R Kollar M9411.14 14:10                                  */
            /* This will eliminate the cases where sec_depth_along_pt < 0    */
            if (sec_depth_along_pt < 0) continue;

/*
  Can't avoid it - look up nearest neighbor fluence and dSAR for depth
  corresponding to nearest neighbor SSD.
*/
            sec_fluence =
                (request.SAR_table.flatness_depth_count == 1) ?
                    tables.fluence_map[fluence_map_y][fluence_map_x][0] :
                    v_interp (0, request.SAR_table.flatness_depth_count,
                        request.SAR_table.flatness_depth_vec,
                        sec_z_depth, /* John R Kollar Sa9402.12 15:33 */
                        tables.fluence_map[fluence_map_y][fluence_map_x],
                        &index, &fx);

            /* John R Kollar W9407.06 16:10                                  */
            /* This establishes a fluence profile along the TIPPED           */
            /* dartboard, with sec_fluence=1.0 at the calculation point.     */
            /* That is, for weighting the scatter pencils we need the        */
            /* fluence along the dartboard "relative to the calculation      */
            /* point", rather than "relative to the calibration point"       */
            /* (which is how the actual fluence MAP is defined).             */
            sec_fluence *= (beam_pt.z*beam_pt.z)/(sc.z*sc.z);
```

photon_point_dose.c.v0

```
/* If there is a filter, look up the filter factor using the fluence
   map indices.
*/
            if (request.beam.filter_count)
                sec_fluence *=
                    request.filter.map[fluence_map_y][fluence_map_x];



            *pt_SAR += sec_fluence *
                v_interp (0, request.SAR_table.depth_count,
                            request.SAR_table.depth_vec,
                            point_dist - sec_SSD,
                            tables.dSAR[rloop], &index, &fx);
        }
    }
```

photon_point_dose.c.v3

```
#ifdef J_DEBUG2
            printf("rloop = %d   r = %f   ang_loop = %d   ",rloop,
                                                tables.dSAR_radii[rloop],ang_loop);
            printf("sec_fluence = %f\n",sec_fluence);
            printf("sc = |(%f,%f,%f)| = %f\n",sc.x,sc.y,sc.z,sc_dist);
            printf("sec_depth_along_pt = %f    ",sec_depth_along_pt);
            printf("sec_z_depth = %f\n\n",sec_z_depth);
#endif J_DEBUG2

/* If there is a filter, look up the filter factor using the fluence
   map indices.
*/
            if (request.beam.filter_count)
                sec_fluence *=
                    request.filter.map[fluence_map_y][fluence_map_x];

#ifdef J_DEBUG6
            sec_fluence = 1.0;
#endif J_DEBUG6

            *pt_SAR += sec_fluence *
                v_interp (0, request.SAR_table.depth_count,
                            request.SAR_table.depth_vec,
                            sec_depth_along_pt /* John R Kollar Sa9402.12 16:50 */,
                            tables.dSAR[rloop], &index, &fx);

#ifdef J_DEBUG7
            printf("depth_count = %d   ",request.SAR_table.depth_count);
            printf("point_dist = %f\n",point_dist);
            printf("sec_SSD = %f   ",sec_SSD);
            printf("index = %d   ",index);
            printf("fx = %f   ",fx);
            printf("dSAR = %f\n\n",
                        v_interp (0, request.SAR_table.depth_count,
                        request.SAR_table.depth_vec,
                        sec_depth_along_pt,
                        tables.dSAR[rloop], &index, &fx));
#endif J_DEBUG7

        }
    }
```

photon_point_dose.c.v0

```
    *pt_SAR /= tables.dSAR_sector_count;









    *pt_SAR /= point_dist_sq;
    *pt_TAR0 /= point_dist_sq;

    return (*pt_TAR0 + *pt_SAR);
}
```

photon_point_dose.c.v3

```
    *pt_SAR /= tables.dSAR_sector_count;

#ifdef J_DEBUG4
    printf("%f = TAR = TAR0 + SAR = %f + %f\n",*pt_TAR0 + *pt_SAR,
                                                    *pt_TAR0,*pt_SAR);
#endif J_DEBUG4

#ifdef J_DEBUG5
    printf("TAR0(%f,%f,%f) = %f\n\n",beam_pt.x,beam_pt.y,beam_pt.z,*pt_TAR0);
#endif J_DEBUG5

    /* John R Kollar W9407.06 15:58 */
    *pt_SAR /= (beam_pt.z * beam_pt.z);
    *pt_TAR0 /= (beam_pt.z * beam_pt.z);

    return (*pt_TAR0 + *pt_SAR);
}
```

7.3. **Appendix C.** Code listing of mk_contour_map.c

This appendix contains a listing of the code for the GRATIS routine mk_contour_map, which constructs the table of SSD's used by the scatter integration. Both the original version and the version with all the final modifications are listed side by side to facilitate comparisons. Also, the code changes that are relevant to the tilting of the dartboard are shown in bold to set them apart from changes that involve debugging statements, documentation and reformatting of C code.

mk_contour_map.c.v0

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_contour_map.c.v 3.2
        1990/10/14 16:24:27 sherouse Exp $";

/*
 * $Log: mk_contour_map.c.v $









 * Revision 3.2  1990/10/14  16:24:27  sherouse
 * - turn of verbose
 *
 * Revision 3.1  89/01/18  14:42:08  sherouse
 * - support new beam description
 * - keep up with changes in UNIT structure
 *
 * Revision 2.2  88/05/31  17:22:46  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:03  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.3  88/05/18  13:55:38  sherouse
 * - working version, saved before major assault
 *
 * Revision 1.2  87/05/22  10:57:52  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:17:41  sherouse
 * Initial revision
 *
 */
```

mk_contour_map.c.v2

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_contour_map.c.v 3.2
        1990/10/14 16:24:27 sherouse Exp $";

/*
 * $Log: mk_contour_map.c.v $
 * Revision 3.4  1994/08/04  21:33:00  jkollar
 * - A bug with revision 3.3 was fixed. The contour was flipped in the
 *   Y-direction, and this showed up in the tests of asymmetric fields.
 *
 * Revision 3.3  1993/11/11  14:27:00  jkollar
 * - The assumption of symmetric jaws in determining the map limits has
 * - been removed. The map now supports asymmetric collimators.
 *
 * Revision 3.2  1990/10/14  16:24:27  sherouse
 * - turn of verbose
 *
 * Revision 3.1  89/01/18  14:42:08  sherouse
 * - support new beam description
 * - keep up with changes in UNIT structure
 *
 * Revision 2.2  88/05/31  17:22:46  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:03  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.3  88/05/18  13:55:38  sherouse
 * - working version, saved before major assault
 *
 * Revision 1.2  87/05/22  10:57:52  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:17:41  sherouse
 * Initial revision
 *
 */
```

mk_contour_map.c.v0

```
#include <stdio.h>
#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define verbose (0)









mk_contour_map()
/*
 ----------------------------------------------------------------------

   NAME
        mk_contour_map - make a map of SSDs for quick reference

   SYNOPSIS


   DESCRIPTION
        This routine builds a table of SSDs on a 5 mm grid as
        described in detail below.

   BUGS
```

mk_contour_map.c.v2

```
#include <stdio.h>
#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define verbose (0)

/* #define EBUG */

/* Print out the jaw x-y positions and the corresponding contour map */
/* indicies. */
/* #define J_DEBUG1 */

/* Print the contour map out to a text file. */
/* #define J_DEBUG2 */

mk_contour_map()
/*
 ----------------------------------------------------------------------

   NAME
        mk_contour_map - make a map of SSDs for quick reference

   SYNOPSIS


   DESCRIPTION
        This routine builds a table of SSDs on a 5 mm grid as
        described in detail below.

   BUGS
```

mk_contour_map.c.v0

```
  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        3 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

 ------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int             xloop,
                    yloop,
                    x_size_in_cells,
                    y_size_in_cells;
    BOOLEAN         inside;

    float           x_inc,
                    y_inc,
                    scale,
                    SSD,
                    thickness,
                    pathlength;


    POINT           bm_source,
                    bm_target,
                    pat_source,
                    pat_target;
```

mk_contour_map.c.v2

```
  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        3 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

 ------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int             xloop,
                    yloop,
                    x_size_in_cells,
                    y_size_in_cells;
    BOOLEAN         inside;

    float           x_inc,
                    y_inc,
                    scale,
                    SSD,
                    thickness,
                    pathlength;


    POINT           bm_source,
                    bm_target,
                    pat_source,
                    pat_target;

    POINT           bt;

#ifdef J_DEBUG2
    FILE *fpr;
#endif J_DEBUG2
```

mk_contour_map.c.v0

```
/*
  Set up a vector from the source to a point on a plane which is
  perpendicular to the c. axis and 200 cm from the source for each
  cell in the contour map.  (There is nothing magic about 200 cm and
  in fact this should probably be handled a little more
  intelligently.)  Recall - you *have* read defines.h - that the
  contour map is defined as a 5 mm grid at the beams definition
  distance layed out so that the central axis intersects the corner of
  four cells - there are an even number of cells across a rectangular
  beam.

  Having set up the ray in beam coordinates, transform to patient
  coordinates and intersect with the patient.   Store the SSD in the
  grid matrix - a value >1000.0 means no intersection.

  The map extends only to the geometric edge (+ epsilon) of the beam.
  Beyond that one should do bidirectional nearest-neighbor.
*/

    bm_source.x = bm_source.y = bm_source.z = 0.0;
    transform_point (bm_source, request.beam.T_beam_to_pat, &pat_source);

#ifdef EBUG
    printf("pat_source.x,y,z = %f %f %f\n",
            pat_source.x, pat_source.y, pat_source.z);
#endif EBUG

    bm_target.z = 200.0;   /* should be a little smarter about this */
    scale = bm_target.z / request.unit.SAD;
```

mk_contour_map.c.v2

```
/*
  Set up a vector from the source to a point on a plane which is
  perpendicular to the c. axis and 200 cm from the source for each
  cell in the contour map.  (There is nothing magic about 200 cm and
  in fact this should probably be handled a little more
  intelligently.)  Recall - you *have* read defines.h - that the
  contour map is defined as a 5 mm grid at the beams definition
  distance layed out so that the central axis intersects the corner of
  four cells - there are an even number of cells across a rectangular
  beam.

  Having set up the ray in beam coordinates, transform to patient
  coordinates and intersect with the patient.   Store the SSD in the
  grid matrix - a value >1000.0 means no intersection.

  The map extends only to the geometric edge (+ epsilon) of the beam
  Beyond that one should do bidirectional nearest-neighbor
*/

    bm_source.x = bm_source.y = bm_source.z = 0.0;
    transform_point (bm_source, request.beam.T_beam_to_pat, &pat_source);

#ifdef EBUG
    printf("pat_source.x,y,z = %f %f %f\n",
            pat_source.x, pat_source.y, pat_source.z);
#endif EBUG

    bm_target.z = 200.0;   /* should be a little smarter about this */
    scale = bm_target.z / request.unit.SAD;
```

mk_contour_map.c.v0

```
/*
  Figure out how much of the map we need to compute for this beam.

  * def is the half width in cm.  Multiply by two to get the number of
  5mm cells, round up and truncate.

  The following assumes symmetric jaws.  Watch out.
*/
    x_size_in_cells = (int) (request.beam.jaw[X_JAW].pos_2 * 2.0 + 0.5);
    if (x_size_in_cells > (MAP_SIZE / 2))
    {
         fprintf (stderr, "ARRRGGH.  Field too big in x (%d cells)\n",
                  x_size_in_cells);
         exit (1);
    }
    y_size_in_cells = (int) (request.beam.jaw[Y_JAW].pos_2 * 2.0 + 0.5);
    if (y_size_in_cells > (MAP_SIZE / 2))
    {
         fprintf (stderr, "ARRRGGH.  Field too big in y (%d cells)\n",
                  y_size_in_cells);
         exit (1);
    }
    tables.con_map_min_x_index = (MAP_SIZE / 2) - x_size_in_cells;
    tables.con_map_max_x_index = (MAP_SIZE / 2) + x_size_in_cells - 1;
    tables.con_map_min_y_index = (MAP_SIZE / 2) - y_size_in_cells;
    tables.con_map_max_y_index = (MAP_SIZE / 2) + y_size_in_cells - 1;
```

mk_contour_map.c.v2

```
/*
  Figure out how much of the map we need to compute for this beam

  * def is the half width in cm.  Multiply by two to get the number of
  5mm cells, round up and truncate.

*/

    /* John R. Kollar 9311.11 */
    tables.con_map_min_x_index = (int)(MAP_SIZE/2.0 - 0.5
                                       + request.beam.jaw[X_JAW].pos_1 * 2.0);
    tables.con_map_max_x_index = (int)(MAP_SIZE/2.0 - 0.5
                                       + request.beam.jaw[X_JAW].pos_2 * 2.0);
    tables.con_map_min_y_index = (int)(MAP_SIZE/2.0 - 0.5
                                       - request.beam.jaw[Y_JAW].pos_2 * 2.0);
    tables.con_map_max_y_index = (int)(MAP_SIZE/2.0 - 0.5
                                       - request.beam.jaw[Y_JAW].pos_1 * 2.0);



#ifdef J_DEBUG1
    printf("request.beam.jaw[X_JAW].pos_1 = %f",request.beam.jaw[X_JAW].pos_1);
    printf("    tables.con_map_min_x_index = %d\n",tables.con_map_min_x_index);
    printf("request.beam.jaw[X_JAW].pos_2 = %f",request.beam.jaw[X_JAW].pos_2);
    printf("    tables.con_map_max_x_index = %d\n",tables.con_map_max_x_index);
    printf("request.beam.jaw[Y_JAW].pos_1 = %f",request.beam.jaw[Y_JAW].pos_1);
    printf("    tables.con_map_min_y_index = %d\n",tables.con_map_min_y_index);
    printf("request.beam.jaw[Y_JAW].pos_2 = %f",request.beam.jaw[Y_JAW].pos_2);
    printf("    tables.con_map_max_y_index = %d\n",tables.con_map_max_y_index);
#endif J_DEBUG1
```

mk_contour_map.c.v0

```
/*
  Step through the cells in raster order.  Note that we are filling in
  the center of the map, not the edge.  Central axis is always between
  elements MAP_SIZE/2 and MAP_SIZE/2 - 1.
*/
    y_inc = -0.5 * scale;
    x_inc = 0.5 * scale;
```

mk_contour_map.c.v2

```
    /* Some array bounds checking. */
    if ((tables.con_map_min_x_index < 0)
                        || (tables.con_map_min_x_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: X_JAW_1 = %f",
                                        request.beam.jaw[X_JAW].pos_1);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }
    if ((tables.con_map_max_x_index < 0)
                        || (tables.con_map_max_x_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: X_JAW_2 = %f",
                                        request.beam.jaw[X_JAW].pos_2);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }
    if ((tables.con_map_min_y_index < 0)
                        || (tables.con_map_min_y_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: Y_JAW_1 = %f",
                                        request.beam.jaw[Y_JAW].pos_1);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }
    if ((tables.con_map_max_y_index < 0)
                        || (tables.con_map_max_y_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: Y_JAW_2 = %f",
                                        request.beam.jaw[Y_JAW].pos_2);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }

/*
  Step through the cells in raster order.  Note that we are filling in
  the center of the map, not the edge.  Central axis is always between
  elements MAP_SIZE/2 and MAP_SIZE/2 - 1.
*/
    y_inc = -0.5 * scale;
    x_inc = 0.5 * scale;
```

mk_contour_map.c.v0

```
bm_target.y = ((y_size_in_cells * 0.5) - 0.25) * scale;

if (verbose)
     printf ("\n");
for (yloop = tables.con_map_min_y_index;
      yloop <= tables.con_map_max_y_index;
      yloop++)
{
     bm_target.x = ((x_size_in_cells * 0.5) - .25) * -scale;
     for (xloop = tables.con_map_min_x_index;
          xloop <= tables.con_map_max_x_index;
          xloop++)
     {
          transform_point(bm_target, request.beam.T_beam_to_pat,
                          &pat_target);

#ifdef EBUG
          printf("pat_target.x,y,z = %f %f %f\n",
                 pat_target.x, pat_target.y, pat_target.z);
#endif EBUG

/*
  This costs like the dickens.  Isn't there some more efficient way to
  do this?
*/
          prism(&request.patient_anastruct, pat_source, pat_target,
                &inside, &SSD,
                &thickness, &pathlength);
          tables.contour_map[yloop][xloop] = SSD;
```

mk_contour_map.c.v2

```
/* John R. Kollar R9408.04 */
bt.y = (MAP_SIZE/2 - tables.con_map_min_y_index - 0.5)*0.5*scale;
bt.x = (tables.con_map_min_x_index - MAP_SIZE/2 + 0.5)*0.5*scale;
bm_target.y = bt.y;

if (verbose)
     printf ("\n");
for (yloop = tables.con_map_min_y_index;
      yloop <= tables.con_map_max_y_index;
      yloop++)
{
     bm_target.x = bt.x;
     for (xloop = tables.con_map_min_x_index;
          xloop <= tables.con_map_max_x_index;
          xloop++)
     {
          transform_point(bm_target, request.beam.T_beam_to_pat,
                          &pat_target);

#ifdef EBUG
          printf("pat_target.x,y,z = %f %f %f\n",
                 pat_target.x, pat_target.y, pat_target.z);
#endif EBUG

/*
  This costs like the dickens.  Isn't there some more efficient way to
  do this?
*/
          prism(&request.patient_anastruct, pat_source, pat_target,
                &inside, &SSD,
                &thickness, &pathlength);
          tables.contour_map[yloop][xloop] = SSD;
```

mk_contour_map.c.v0

```
#ifdef EBUG
            if (thickness != 0.0)
                printf("SSD[%d][%d] = %f\n", yloop, xloop, SSD);
#else
            if (verbose)
            {
                printf(".");
                fflush (stdout);
            }
#endif EBUG

            bm_target.x += x_inc;
        }
#ifndef EBUG
        if (verbose)
            printf ("\n");
#endif
        bm_target.y += y_inc;
    }









    return;
}
```

mk_contour_map.c.v2

```
#ifdef EBUG
            if (thickness != 0.0)
                printf("SSD[%d][%d] = %f\n", yloop, xloop, SSD);
#else
            if (verbose)
            {
                printf(".");
                fflush (stdout);
            }
#endif EBUG

            bm_target.x += x_inc;
        }
#ifndef EBUG
        if (verbose)
            printf ("\n");
#endif
        bm_target.y += y_inc;
    }

#ifdef J_DEBUG2
    fpr = fopen("contour_map.txt","w");
    for (yloop = 0;yloop < MAP_SIZE;yloop++) {
        for (xloop = 0;xloop < MAP_SIZE-1;xloop++) {
            fprintf(fpr,"%f ",tables.contour_map[yloop][xloop]);
        }
        fprintf(fpr,"%f\n",tables.contour_map[yloop][xloop]);
    }
    fclose(fpr);
#endif J_DEBUG2

   return;
}
```

7.4. **Appendix D.** Code listing of mk_dSAR_table.c

This appendix contains a listing of the code for the GRATIS routine mk_dSAR_table, which constructs the table of dSAR's used by the scatter integration. Both the original version and the version with all the final modifications are listed side by side to facilitate comparisons. Also, the code changes that are relevant to the tilting of the dartboard are shown in bold to set them apart from changes that involve debugging statements, documentation and reformatting of C code.

mk_dSAR_table.c.v0

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_dSAR_table.c,v 3.3
        1992/05/28 18:29:03 sherouse Exp $";

/*
 * $Log: mk_dSAR_table.c,v $




 * Revision 3.3  1992/05/28  18:29:03  sherouse
 * - ifdef out the wild idea
 *
 * Revision 3.2  1990/10/14  16:24:58  sherouse
 * - the PSF has moved to the sar data file - see make_sar
 *
 * Revision 3.1  89/01/18  14:43:24  sherouse
 * - support new beam description
 * - add PSF table calculation
 *
 * Revision 2.2  88/05/31  17:22:51  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:09  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.2  87/05/22  11:09:28  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:18:29  sherouse
 * Initial revision
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"
```

mk_dSAR_table.c.v3

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_dSAR_table.c,v 3.3
        1992/05/28 18:29:03 sherouse Exp $";

/*
 * $Log: mk_dSAR_table.c,v $
 *
 * Revision 3.4  1993/11/19  12:02:00  jkollar
 * - The assumption of symmetric jaws in calculating the diagonal has been
 * - removed.
 *
 * Revision 3.3  1992/05/28  18:29:03  sherouse
 * - ifdef out the wild idea
 *
 * Revision 3.2  1990/10/14  16:24:58  sherouse
 * - the PSF has moved to the sar data file - see make_sar
 *
 * Revision 3.1  89/01/18  14:43:24  sherouse
 * - support new beam description
 * - add PSF table calculation
 *
 * Revision 2.2  88/05/31  17:22:51  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:09  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.2  87/05/22  11:09:28  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:18:29  sherouse
 * Initial revision
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"
```

mk_dSAR_table.c.v0

```
#include "defines.h"

#define SQRT(x) ((float) sqrt((double) (x)))
#define SIN(x) ((float) sin((double) (x)))
#define COS(x) ((float) cos((double) (x)))









mk_dSAR_table()
/*
 ------------------------------------------------------------------------

   NAME
        mk_dSAR_table - make a table of differential S*Rs

   SYNOPSIS


   DESCRIPTION
        This routine constructs a table of differential S*Rs.  It
        starts by choosing a set of sector radii based on the
        collimator setting and some heuristics (there is a wealth of
        godd projects in this).  It then builds the table for the
        centers of those sectors an the depths of the original S*R
        table.  Some other tables for quick reference are built.
        Finally, the dS*R table is modified to "remove" some flatness
        dependence.

   BUGS
        See comments regarding flatness removal below.
```

mk_dSAR_table.c.v3

```
#include "defines.h"

#define SQRT(x) ((float) sqrt((double) (x)))
#define SIN(x) ((float) sin((double) (x)))
#define COS(x) ((float) cos((double) (x)))

/* John R Kollar R9401.20 */
/* Since our beam flatness table does not roll off for large fields */
/* (ie. It is extrapolated beyond the largest measured field size) */
/* we believe that the WILD_IDEA is correct. */
#define WILD_IDEA

/* Print the dSAR table out to the file SARtest.txt */
#define J_DEBUG

/* #define OLD_dRADIUS */

mk_dSAR_table()
/*
 ------------------------------------------------------------------------

   NAME
        mk_dSAR_table - make a table of differential S*Rs

   SYNOPSIS


   DESCRIPTION
        This routine constructs a table of differential S*Rs.  It
        starts by choosing a set of sector radii based on the
        collimator setting and some heuristics (there is a wealth of
        godd projects in this).  It then builds the table for the
        centers of those sectors an the depths of the original S*R
        table.  Some other tables for quick reference are built.
        Finally, the dS*R table is modified to "remove" some flatness
        dependence.

   BUGS
        See comments regarding flatness removal below.
```

mk_dSAR_table.c.v0

```
AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        2 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int         rloop,
                dloop,
                ang_loop,
                index,
                loop;

    float       diagonal,
                radius,
                SAR_so_far,
                SAR,
                fx,
                angle_inc,
                angle,
                sine,
                cosine,
                previous_r,
                sec_center,
                last_one,
                x,
                xn,
                inc,
                depth_scale,
                flatness[MAX_FLATNESS_RADII];
```

mk_dSAR_table.c.v3

```
AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        2 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int         rloop,
                dloop,
                ang_loop,
                index,
                loop;

    float       diagonal,
                radius,
                SAR_so_far,
                SAR,
                fx,
                angle_inc,
                angle,
                sine,
                cosine,
                previous_r,
                sec_center,
                last_one,
                x,
                xn,
                inc,
                depth_scale,
                flatness[MAX_FLATNESS_RADII];
```

mk_dSAR_table.c.v0

```
    float       v_interp();




/*
  Set up the radius vector for the dSAR table.  These radii are the
  *outer* bounds of the scatter sectors.  Sometime it would be nice
  to devise a heuristic for optimizing these sector sizes but for now
  I'll just use equal partitioning.

  The following assumes symmetric jaws.  Watch out.
*/
    diagonal = 2.5 * SQRT(request.beam.jaw[X_JAW].pos_2 *
                          request.beam.jaw[X_JAW].pos_2 +
                          request.beam.jaw[Y_JAW].pos_2 *
                          request.beam.jaw[Y_JAW].pos_2);

#ifdef OLD_dRADIUS

/* Here's the old way */
    tables.dSAR_radius_count = dSAR_MAX_RADII;
    radius_inc = diagonal / (tables.dSAR_radius_count - 1);
    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        tables.dSAR_radii[rloop] = (radius += radius_inc);
```

mk_dSAR_table.c.v3

```
#ifdef OLD_dRADIUS
    float radius_inc;
#endif OLD_dRADIUS

    float       v_interp();

    float       W,L; /* John R Kollar M9311.15 */

#ifdef J_DEBUG
    FILE *fpr;
#endif J_DEBUG

/*
  Set up the radius vector for the dSAR table.  These radii are the
  *outer* bounds of the scatter sectors.  Sometime it would be nice
  to devise a heuristic for optimizing these sector sizes but for now
  I'll just use equal partitioning.
*/


    /* John R Kollar M9311.15 */
    W = request.beam.jaw[X_JAW].pos_2 - request.beam.jaw[X_JAW].pos_1;
    L = request.beam.jaw[Y_JAW].pos_2 - request.beam.jaw[Y_JAW].pos_1;
    diagonal = 1.25*SQRT(W*W + L*L);

#ifdef OLD_dRADIUS

/* Here's the old way */
    tables.dSAR_radius_count = dSAR_MAX_RADII;
    radius_inc = diagonal / (tables.dSAR_radius_count - 1);
    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        tables.dSAR_radii[rloop] = (radius += radius_inc);
```

mk_dSAR_table.c.v0

```
#else
/*
  And here's an experimental heuristic.  We'll use a recursive
  geometric series.  We can calculate the basic increment and
  recurse from there.  If the basic increment is too small, we
  can make the table smaller and take the time saving to the bank.

                    n
  r  = r    + inc * x
   n    n-1

  r0 = 0.0
*/
    last_one = x = xn = 1.14;
    for (loop = 0; loop < tables.dSAR_radius_count - 2; loop++)
         last_one = (last_one + 1.0) * x;
    inc = diagonal / last_one;

    printf("increment, last_one = %f, %f\n", inc, last_one);

    if (inc < diagonal / 100.0) inc = diagonal / 100.0;

    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
    {
         tables.dSAR_radii[rloop] = tables.dSAR_radii[rloop - 1] + inc * xn;
         if (tables.dSAR_radii[rloop] > diagonal)
             break;
         xn *= x;
    }
    tables.dSAR_radius_count = (rloop >= tables.dSAR_radius_count) ?
         tables.dSAR_radius_count : rloop + 1;

    printf("modified increment = %f, radius count = %d\n",
            inc, tables.dSAR_radius_count);

    for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         printf (" %f", tables.dSAR_radii[rloop]);
    printf("\n");

#endif
```

mk_dSAR_table.c.v3

```
#else
/*
  And here's an experimental heuristic.  We'll use a recursive
  geometric series.  We can calculate the basic increment and
  recurse from there.  If the basic increment is too small, we
  can make the table smaller and take the time saving to the bank.

                    n
  r  = r    + inc * x
   n    n-1

  r0 = 0.0
*/
    last_one = x = xn = 1.14;
    for (loop = 0; loop < tables.dSAR_radius_count - 2; loop++)
         last_one = (last_one + 1.0) * x;
    inc = diagonal / last_one;

    printf("increment, last_one = %f, %f\n", inc, last_one);

    if (inc < diagonal / 100.0) inc = diagonal / 100.0;

    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
    {
         tables.dSAR_radii[rloop] = tables.dSAR_radii[rloop - 1] + inc * xn;
         if (tables.dSAR_radii[rloop] > diagonal)
             break;
         xn *= x;
    }
    tables.dSAR_radius_count = (rloop >= tables.dSAR_radius_count) ?
         tables.dSAR_radius_count : rloop + 1;

    printf("modified increment = %f, radius count = %d\n",
            inc, tables.dSAR_radius_count);




#endif
```

mk_dSAR_table.c.v0

```
tables.dSAR_depth_count = request.SAR_table.depth_count;




for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
{
     tables.dSAR[0][dloop] = 0.0;
     SAR_so_far = request.SAR_table.SAR[dloop][0];




     for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
     {
         radius = tables.dSAR_radii[rloop];
         SAR = v_interp(0, request.SAR_table.radius_count,
                        request.SAR_table.radius_vec, radius,
                        request.SAR_table.SAR[dloop], &index, &fx);
         tables.dSAR[rloop][dloop] = SAR - SAR_so_far;
         SAR_so_far = SAR;




     }



}
```

mk_dSAR_table.c.v3

```
    printf("\n");
    for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         printf ("radius[%d] = %f\n",rloop, tables.dSAR_radii[rloop]);
    printf("\n");

    tables.dSAR_depth_count = request.SAR_table.depth_count;

#ifdef J_DEBUG
    fpr = fopen("SARtest.txt","w");
#endif J_DEBUG

    for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
    {
         tables.dSAR[0][dloop] = 0.0;
         SAR_so_far = request.SAR_table.SAR[dloop][0];

#ifdef J_DEBUG
         fprintf(fpr,"[%f]  ",request.SAR_table.depth_vec[dloop]);
#endif J_DEBUG

         for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
         {
             radius = tables.dSAR_radii[rloop];
             SAR = v_interp(0, request.SAR_table.radius_count,
                            request.SAR_table.radius_vec, radius,
                            request.SAR_table.SAR[dloop], &index, &fx);
             tables.dSAR[rloop][dloop] = SAR - SAR_so_far;
             SAR_so_far = SAR;

#ifdef J_DEBUG
             fprintf(fpr,"%f  ",tables.dSAR[rloop][dloop]);
#endif J_DEBUG

         }

#ifdef J_DEBUG
        fprintf(fpr,"\n");
#endif J_DEBUG

    }
```

mk_dSAR_table.c.v0

```
/*
  Calculate x, y offsets for centers of scatter sectors
*/
    angle_inc = 2.0 * PI / tables.dSAR_sector_count;
    angle = 0.0;
    sine = 0.0;
    cosine = 1.0;
    for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
    {
         previous_r = 0.0;
         for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         {
             sec_center = (tables.dSAR_radii[rloop] + previous_r) / 2.0;
             tables.sec_center_x[ang_loop][rloop] = sec_center * cosine;
             tables.sec_center_y[ang_loop][rloop] = sec_center * sine;
             previous_r = tables.dSAR_radii[rloop];
         }
         angle += angle_inc;
         sine = SIN(angle);
         cosine = COS(angle);
    }

#ifdef WILD_IDEA
/*
  Here's a wild idea...  Divide the differential SARs by the relative
  fluence at the sector centers.  The reasoning is something like this.
  The measured SAR table is for the central axis and includes the effects
  of beam non-flatness.  That's fine if we are caculating on the central
  axis, but off axis this causes the sector weightings to get screwed.
  If we divide out beam non-flatness now and multiply it back in later
  for the proper geometry we may get marginally better control over the
  shape of the beam profile.
*/
```

mk_dSAR_table.c.v3

```
#ifdef J_DEBUG
    fclose(fpr);
#endif J_DEBUG

/*
  Calculate x, y offsets for centers of scatter sectors
*/
    angle_inc = 2.0 * PI / tables.dSAR_sector_count;
    angle = 0.0;
    sine = 0.0;
    cosine = 1.0;
    for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
    {
         previous_r = 0.0;
         for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         {
             sec_center = (tables.dSAR_radii[rloop] + previous_r) / 2.0;
             tables.sec_center_x[ang_loop][rloop] = sec_center * cosine;
             tables.sec_center_y[ang_loop][rloop] = sec_center * sine;
             previous_r = tables.dSAR_radii[rloop];
         }
         angle += angle_inc;
         sine = SIN(angle);
         cosine = COS(angle);
    }

#ifdef WILD_IDEA
/*
  Here's a wild idea...  Divide the differential SARs by the relative
  fluence at the sector centers.  The reasoning is something like this.
  The measured SAR table is for the central axis and includes the effects
  of beam non-flatness.  That's fine if we are caculating on the central
  axis, but off axis this causes the sector weightings to get screwed
  If we divide out beam non-flatness now and multiply it back in later
  for the proper geometry we may get marginally better control over the
  shape of the beam profile.
*/
```

mk_dSAR_table.c.v0

```
/*
  The problem with this is that if the profile rolls off dramatically
  at the edge, the dSARs for large radii get multiplied by large
  numbers and so become too heavily weighted in the scatter
  integration.  This is very bad, particularly for large fields.
  Back to the drawing board.  28 May 1992 GWS
*/

    for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
    {
        /*
          Need to scale radius from depth back to def_dist before
          doing flatness lookup.  The following is not quite right.  I am
          tacitly assuming here that the SARs were measured with
          the phantom surface at def_dist.  This is (almost) the case
          if the S*R data were extracted from PDD measurements but is
          badly wrong if the T*Rs were measured directly with each
          depth at def_dist.  The S*R table really should include a
          measurement distance for each depth - or better yet the S*R
          table should already have flatness taken out of it.
          */
        depth_scale = request.unit.SAD /
            (request.unit.SAD + request.SAR_table.depth_vec[dloop]);

        if (request.SAR_table.flatness_depth_count == 1)
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] = request.SAR_table.flatness[loop][0];
        else
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] =
                     v_interp (0, request.SAR_table.flatness_depth_count,
                                  request.SAR_table.flatness_depth_vec,
                                  request.SAR_table.depth_vec[dloop],
                                  request.SAR_table.flatness[loop], &index,&fx);
```

mk_dSAR_table.c.v3

```
/*
  The problem with this is that if the profile rolls off dramatically
  at the edge, the dSARs for large radii get multiplied by large
  numbers and so become too heavily weighted in the scatter
  integration.  This is very bad, particularly for large fields.
  Back to the drawing board.  28 May 1992 GWS
*/

    for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
    {
        /*
          Need to scale radius from depth back to def_dist before
          doing flatness lookup.  The following is not quite right.  I am
          tacitly assuming here that the SARs were measured with
          the phantom surface at def_dist.  This is (almost) the case
          if the S*R data were extracted from PDD measurements but is
          badly wrong if the T*Rs were measured directly with each
          depth at def_dist.  The S*R table really should include a
          measurement distance for each depth - or better yet the S*R
          table should already have flatness taken out of it.
        */
        depth_scale = request.unit.SAD /
            (request.unit.SAD + request.SAR_table.depth_vec[dloop]);

        if (request.SAR_table.flatness_depth_count == 1)
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] = request.SAR_table.flatness[loop][0];
        else
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] =
                     v_interp (0, request.SAR_table.flatness_depth_count,
                                  request.SAR_table.flatness_depth_vec,
                                  request.SAR_table.depth_vec[dloop],
                                  request.SAR_table.flatness[loop],&index,&fx);
```

mk_dSAR_table.c.v0

```
        for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        {
            radius = (tables.dSAR_radii[rloop] +
                      tables.dSAR_radii[rloop - 1]) * 0.5;
            radius *= depth_scale;
            tables.dSAR[rloop][dloop] /=
                 v_interp(0, request.SAR_table.flatness_radius_count,
                          request.SAR_table.flatness_radius_vec, radius,
                          flatness, &index, &fx);
        }
    }
#endif WILD_IDEA
}
```

mk_dSAR_table.c.v3

```
        for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        {
            radius = (tables.dSAR_radii[rloop] +
                      tables.dSAR_radii[rloop - 1]) * 0.5;
            radius *= depth_scale;
            tables.dSAR[rloop][dloop] /=
                 v_interp(0, request.SAR_table.flatness_radius_count,
                          request.SAR_table.flatness_radius_vec, radius,
                          flatness, &index, &fx);
        }
    }
#endif WILD_IDEA
}
```

Figure 4.13g. 10x10, X=3 Y=3, YZ-iso X=3, tilted db & map fix.

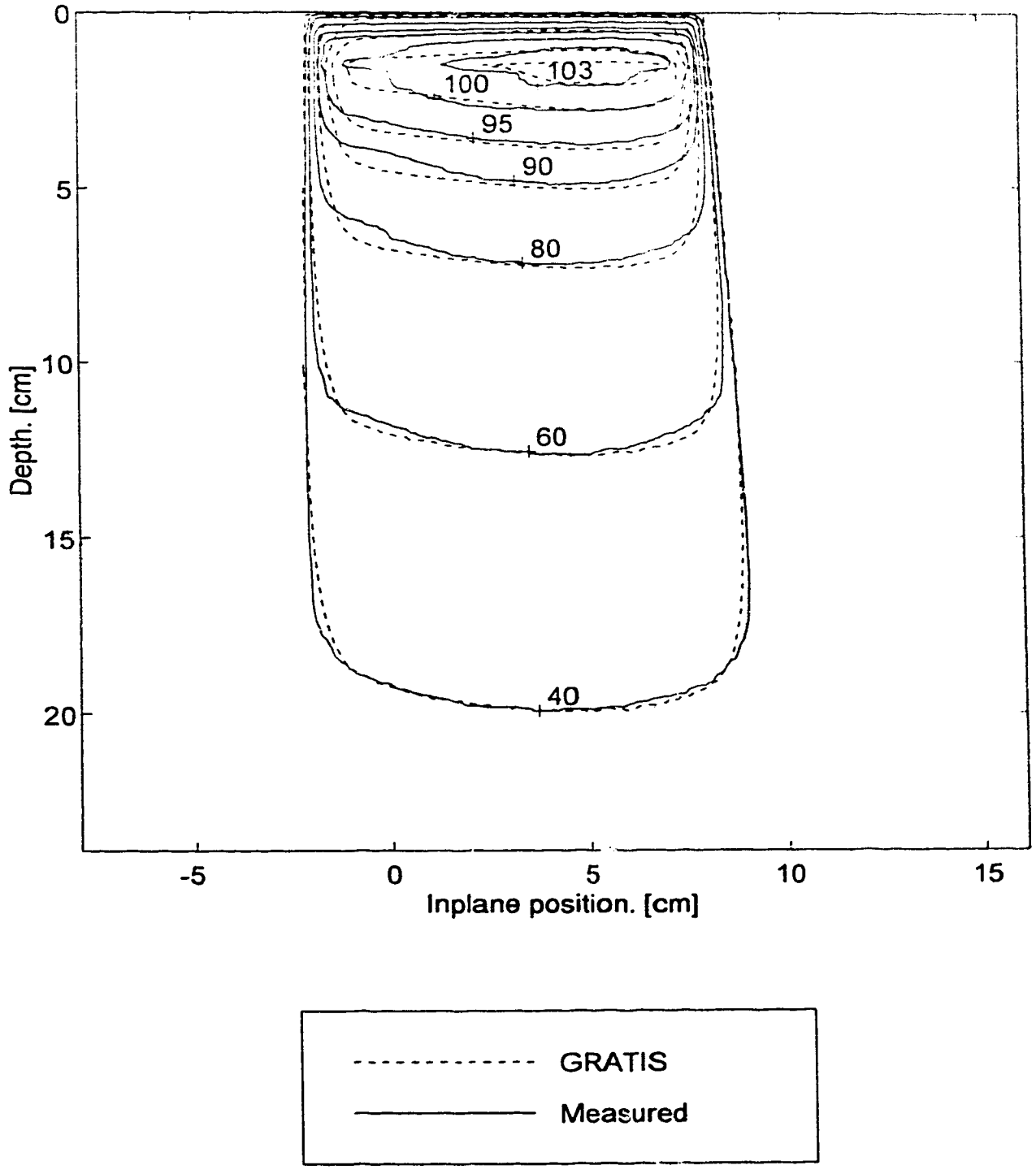


Figure 4.14a. 10x10, X=7 Y=15, YZ-iso X=7, 4% discrepancy level.

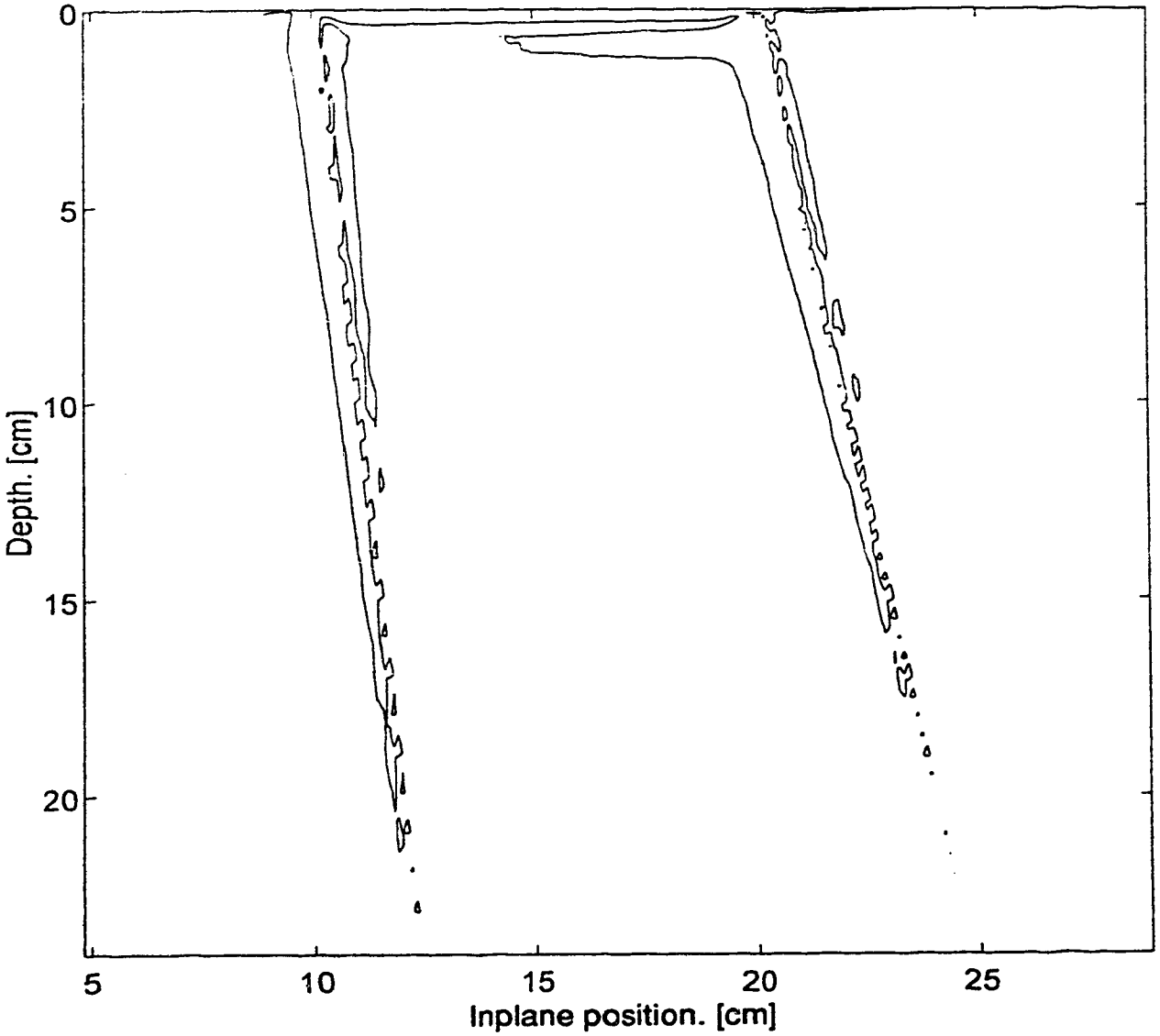


Figure 4.14b. 10x10, X=7 Y=15, YZ-iso X=7, 3% discrepancy level.

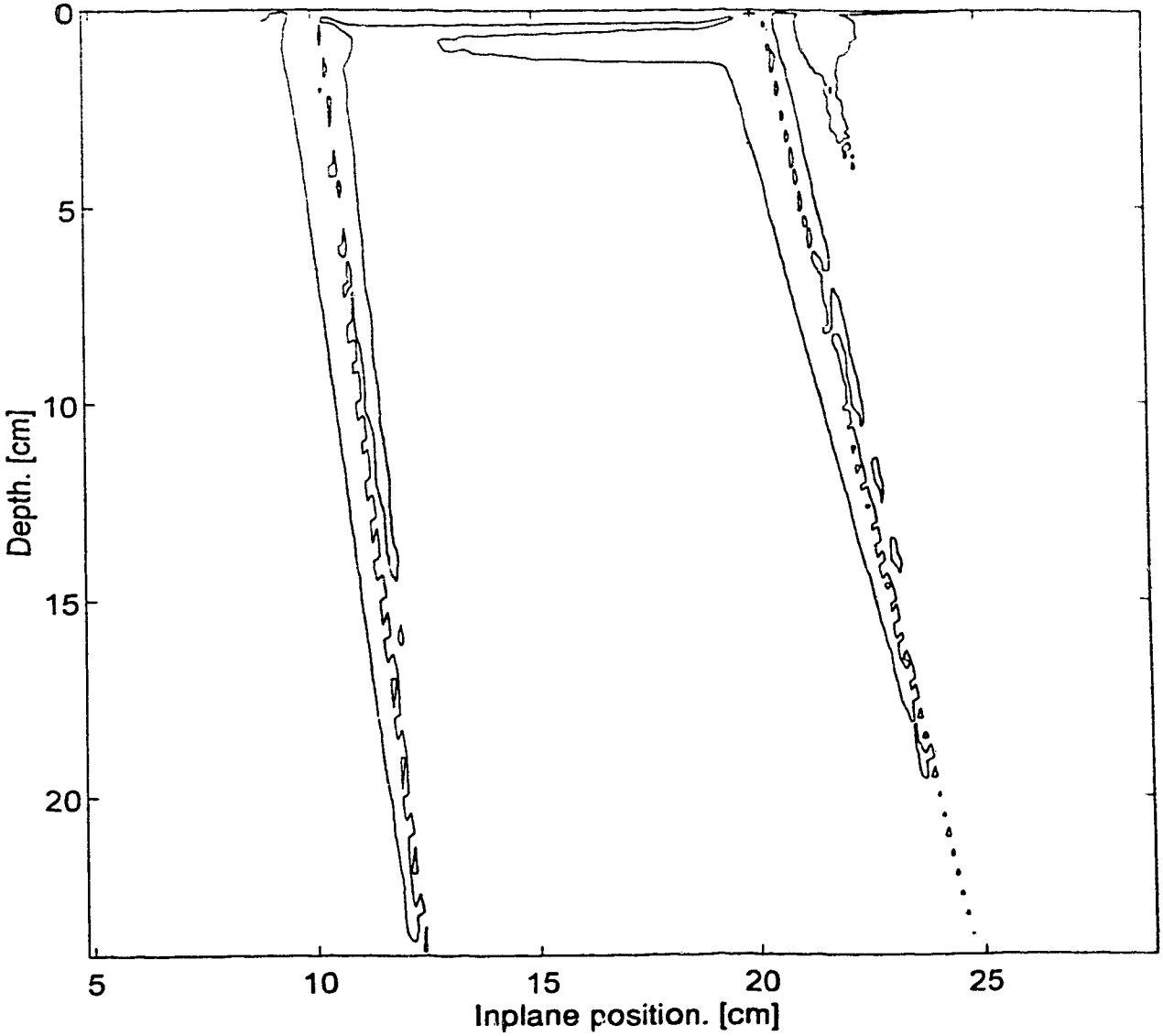


Figure 4.14c. 10x10, X=7 Y=15, YZ-iso X=7, 2% discrepancy level.

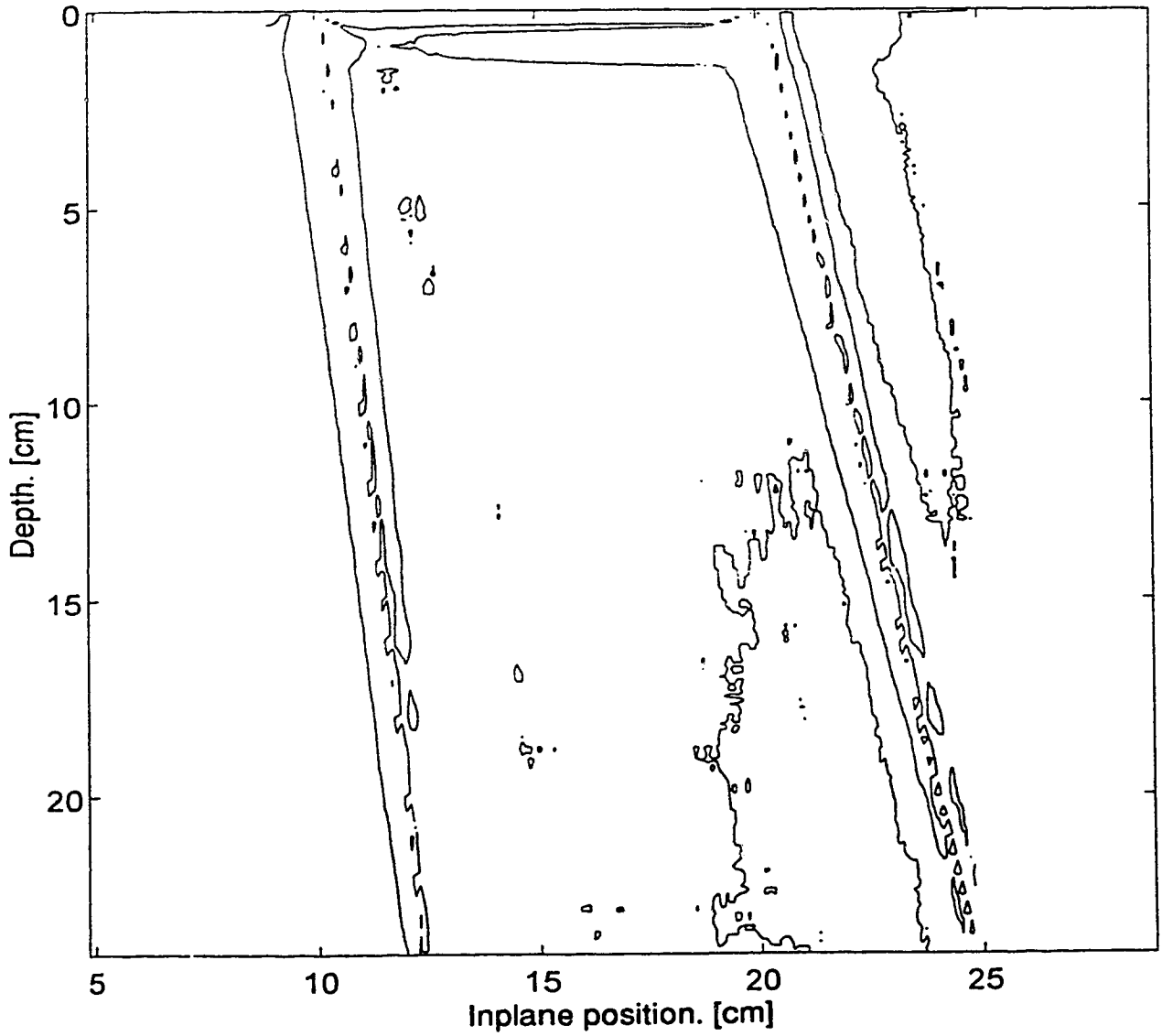


Figure 4.14d. 10x10, X=7 Y=15, YZ-iso X=7, 1% discrepancy level.

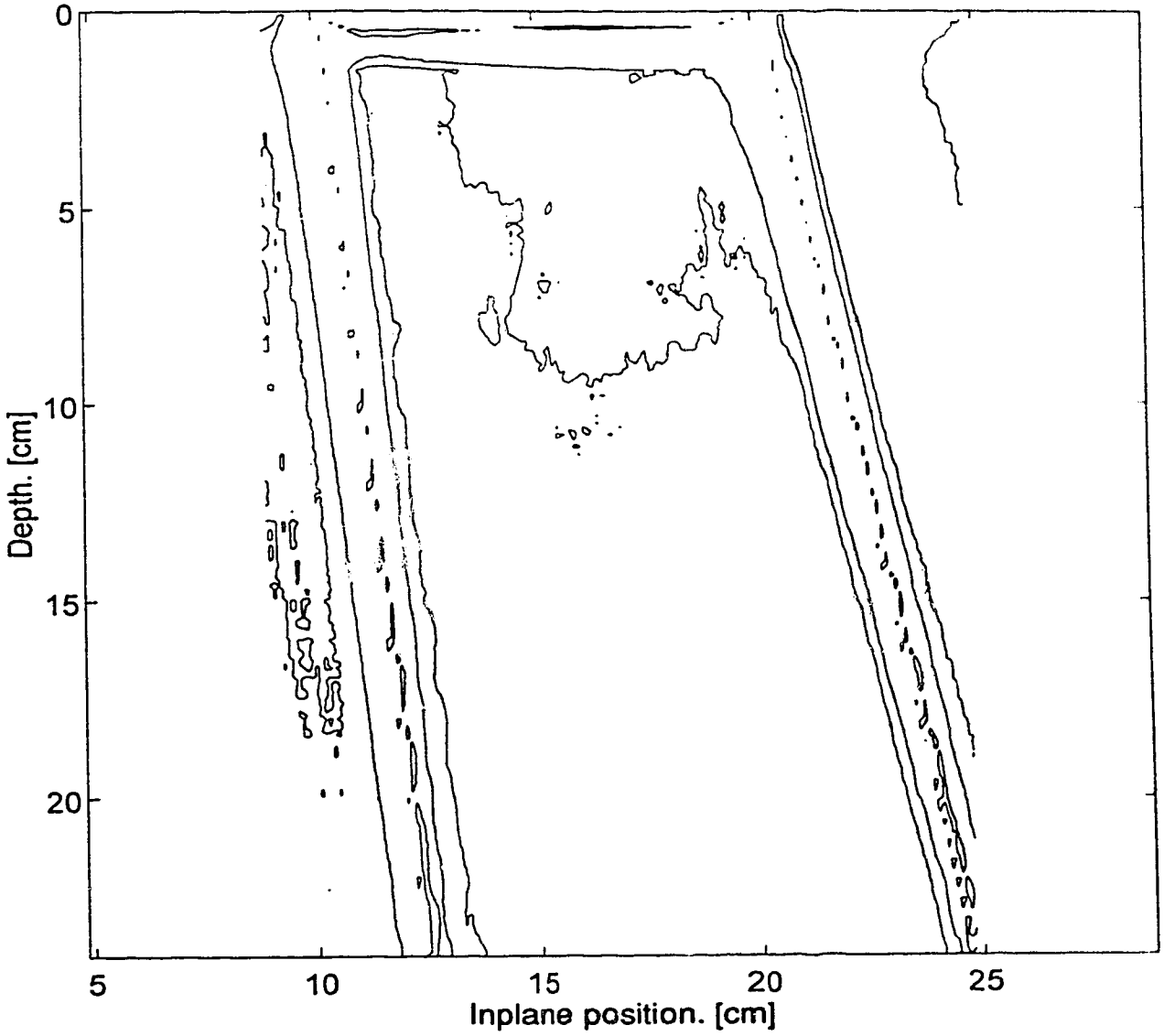


Figure 4.15a. 10x10, X=7 Y=15, XZ-iso Y=15, 4% discrepancy level.

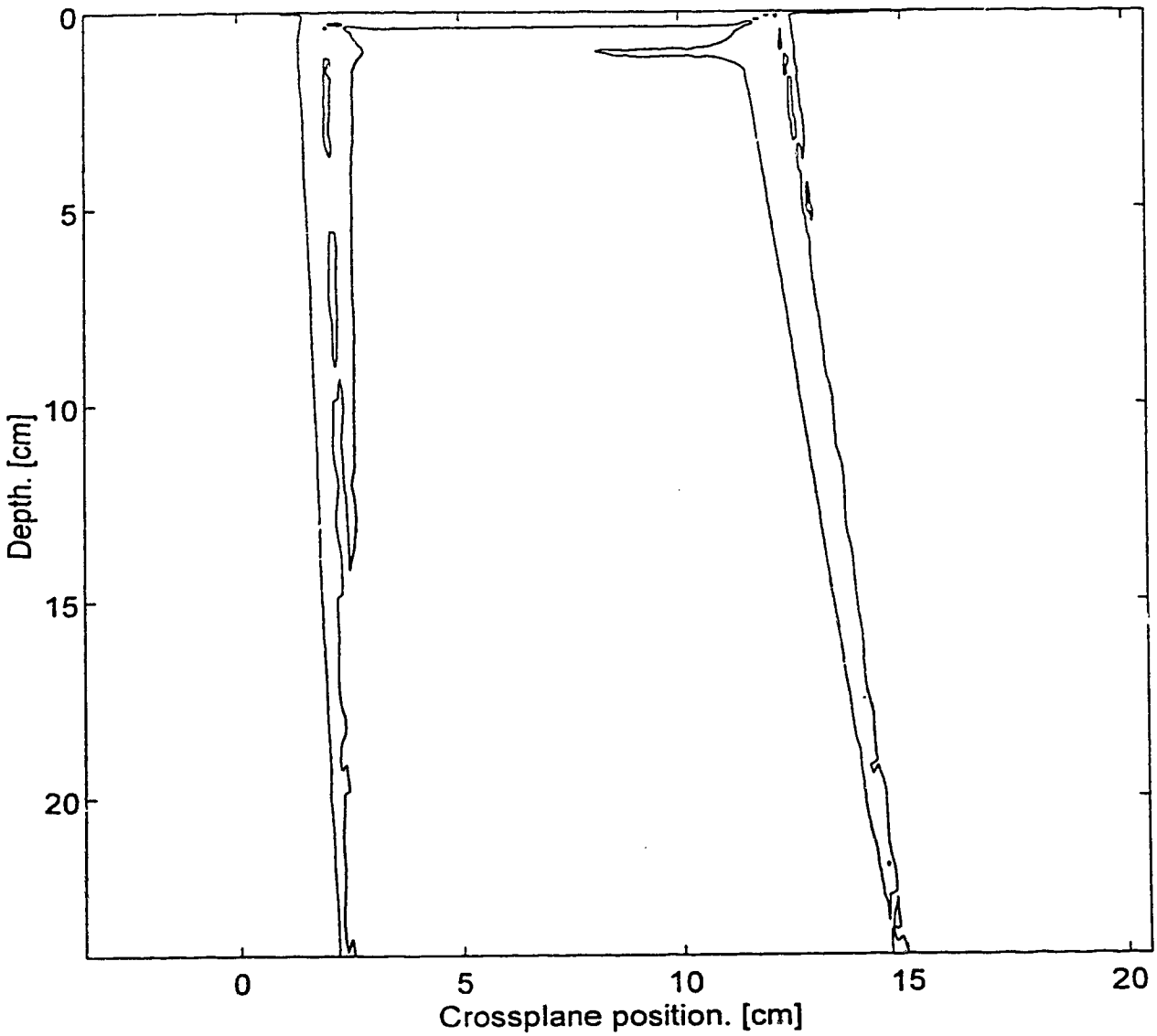


Figure 4.15b. 10x10, X=7 Y=15, XZ-iso Y=15, 3% discrepancy level.

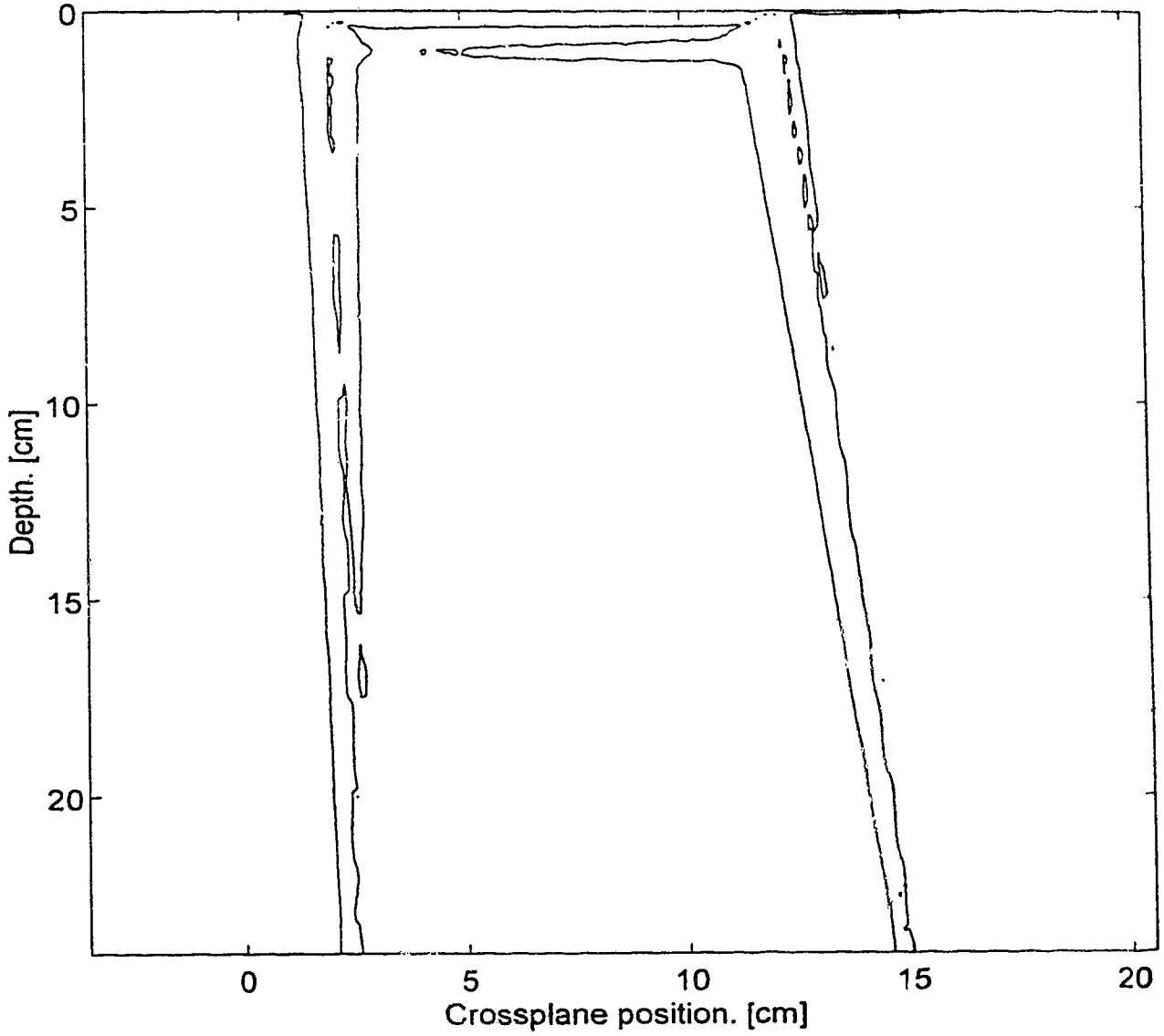


Figure 4.15c. 10x10, X=7 Y=15, XZ-iso Y=15, 2% discrepancy level.

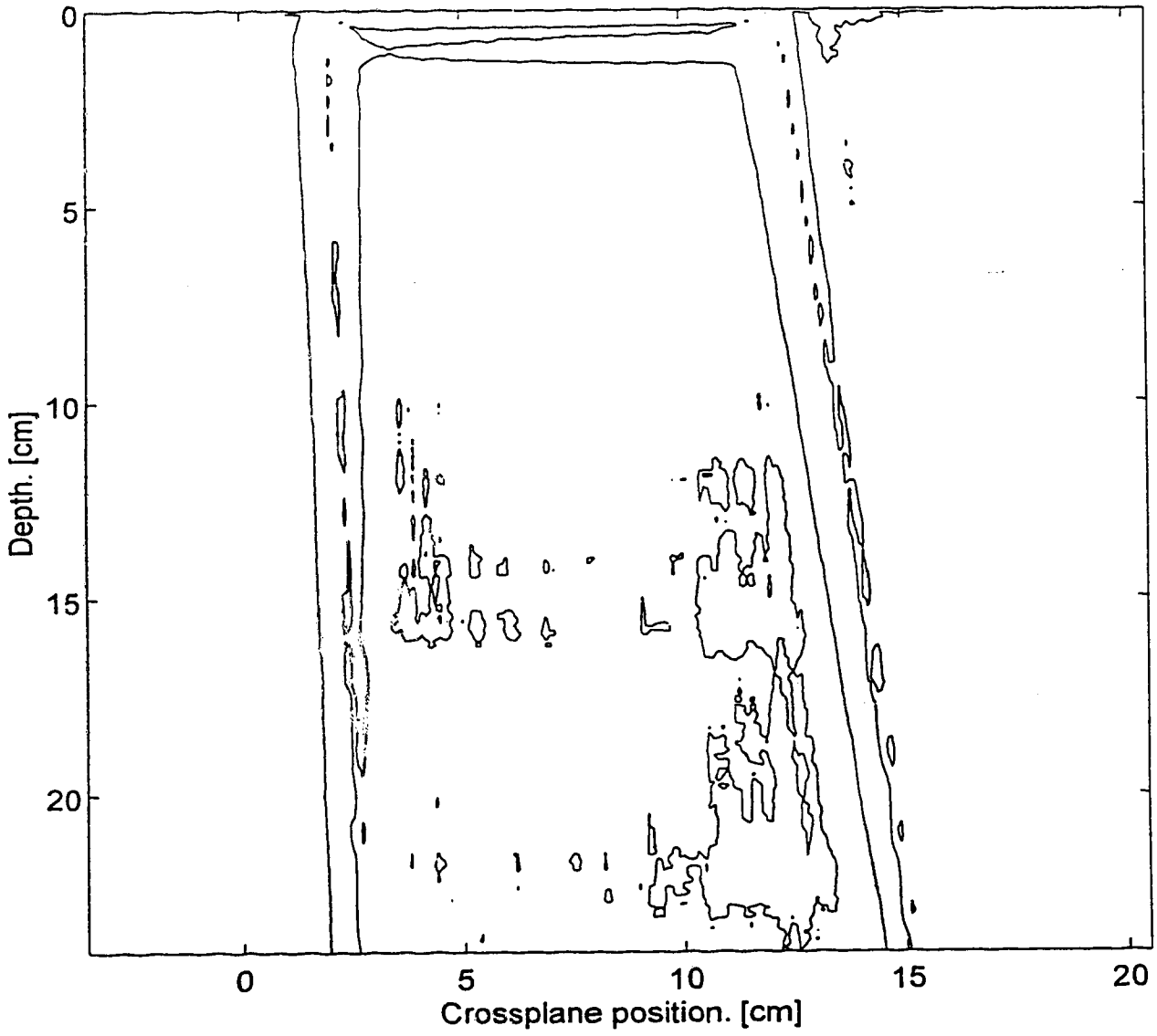
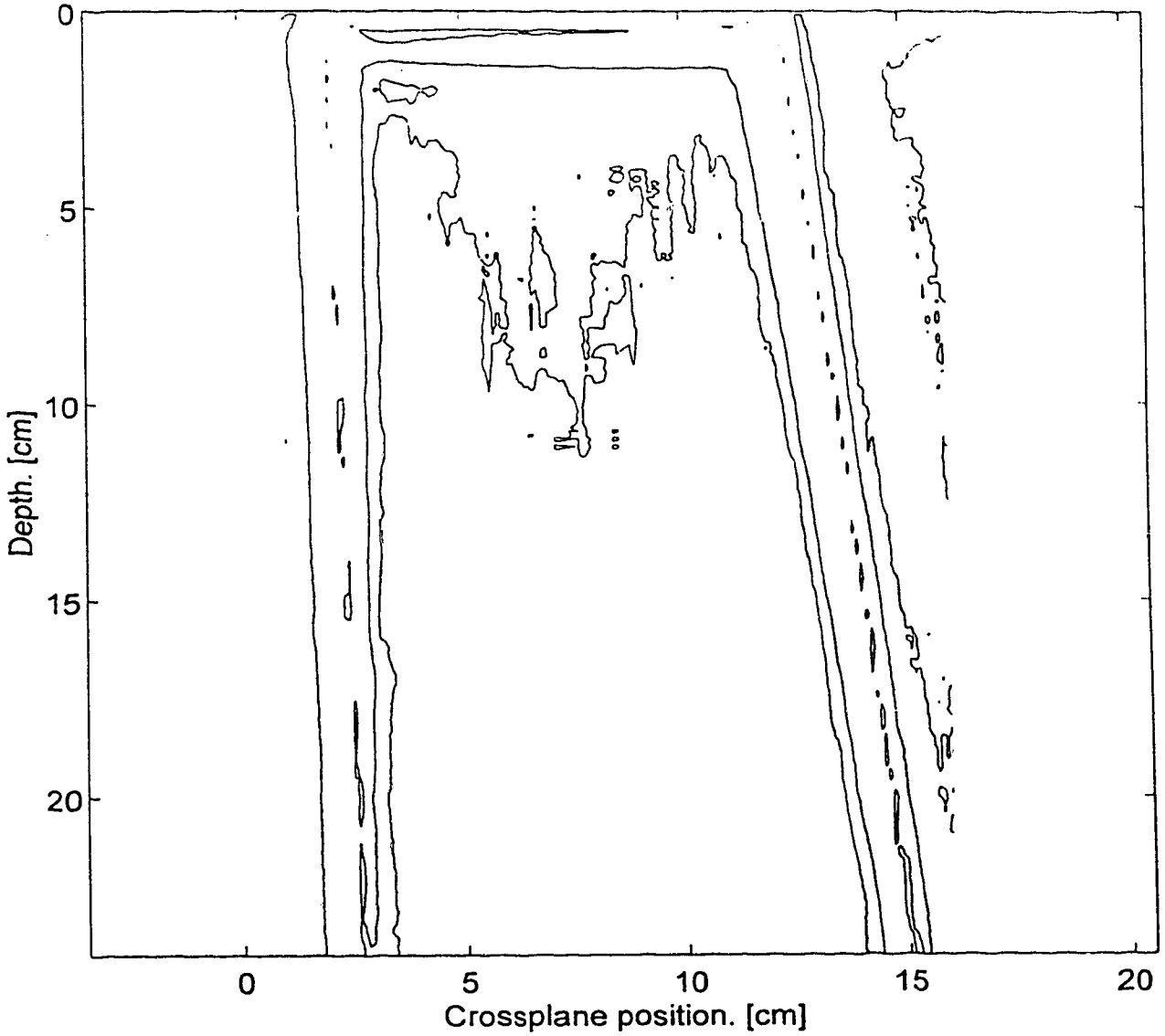


Figure 4.15d. 10x10, X=7 Y=15, XZ-iso Y=15, 1% discrepancy level.



There still appears to be increasing disagreement in the isodoses as the fields become more asymmetric. The extreme cases in figures 4.13e and 4.13f show the worst discrepancies, and plots of these discrepancies are presented in figures 4.14 and 4.15. These plots resemble the extreme single asymmetric cases in figures 4.12, with discrepancies developing in the shallow region and in the lower right area of the plot (figures 4.14c and 4.15c). As was suggested in the single asymmetric analysis, these discrepancies may be artifacts of beam-softening, and thus in figure 4.16 an experiment to test this hypothesis is presented.

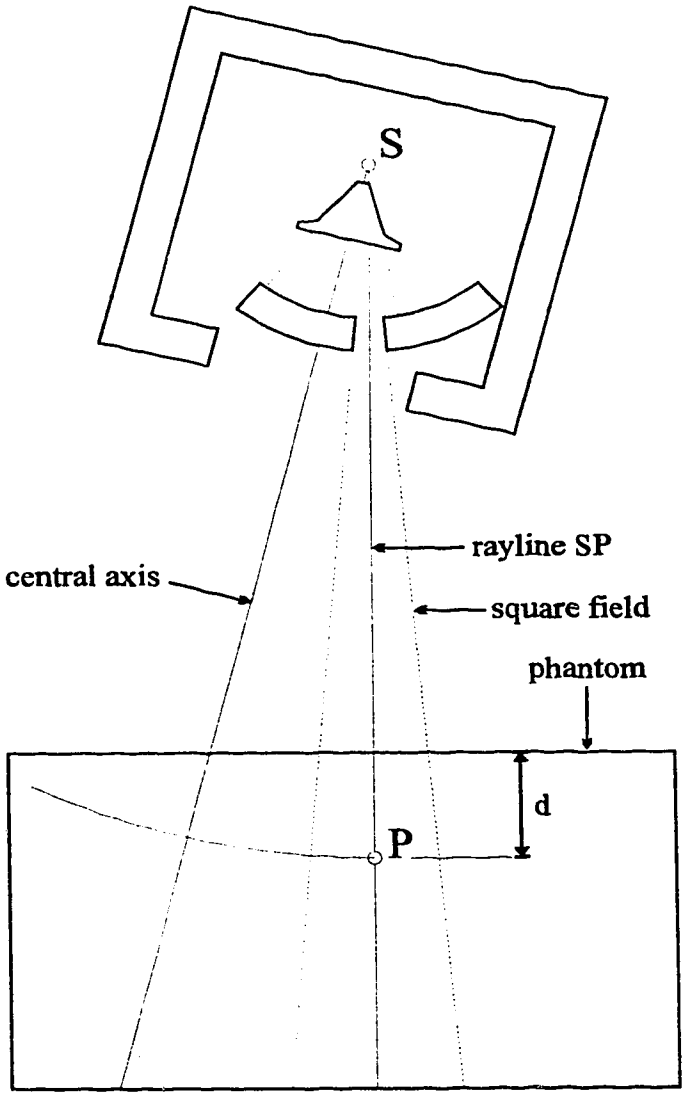


Figure 4.16. Off-axis measurement of TAR's.

The setup in figure 4.16 is designed to produce a square field on the surface of the watertank (SSD = 100 cm) with a gantry tilt of 6.5° and with the beam axis remaining vertical (i.e. radiation along the beam axis is normal to the water surface). The gantry angle was chosen to be the maximum angle that still allowed for field sizes between 3x3 cm^2 and 10x10 cm^2. That is, the greater the gantry rotation the further the collimators must be moved asymmetrically. The maximum asymmetric placement of the collimators thus defines the maximum gantry angle that will still support the desired field sizes. In the case of the linac being used for the measurements in this thesis the Y collimators can move to a maximum position of 10 cm **over** the central axis and 20 cm **away** from the central axis. Thus the maximum asymmetric position for a 10x10 field is 15 cm off the central axis, and is 11.5 cm off the central axis for the 3x3 field. It was thus the maximally asymmetric 3x3 cm^2 field that determined the gantry angle used for the measurements in figure 4.16. The setup procedure for these measurements is given below.

1) Set up and level the water tank.
2) Set SSD = 98.5 cm.
3) Position the diode on the central axis using the field light crosshairs.
4) Lower diode so that rounded top of buildup cap is just touching water surface. (i.e. The 1.5 cm lucite buildup cap is left on.)
5) Enter the diode's position as isocenter into the Wellhöfer scanning software.
6) Using the software, reposition the diode to $d = 100 \cdot (1 - \cos\theta) = 0.64$ cm and $x = 100 \cdot \sin\theta = 11.32$ cm. (Figure 4.17.)
7) Set the collimator positions for the field "3x3 X=0 Y=11.5".
8) Rotate the collimators 90° to make the Y axis the crossplane axis.
9) Rotate the gantry to $\theta = \arctan(11.5/100) \approx 6.5°$.
10) Lower the water until the top of the buildup cap is just touching the surface.
11) Recheck that the diode is in the center of the 3x3 field and compensate with the diode-positioning hand console.

Step 11 resulted in the diode being repositioned to a crossplane position of 11.4 cm, thus explaining why 11.4 cm was used here and in the extreme single asymmetric case of section 4.1.

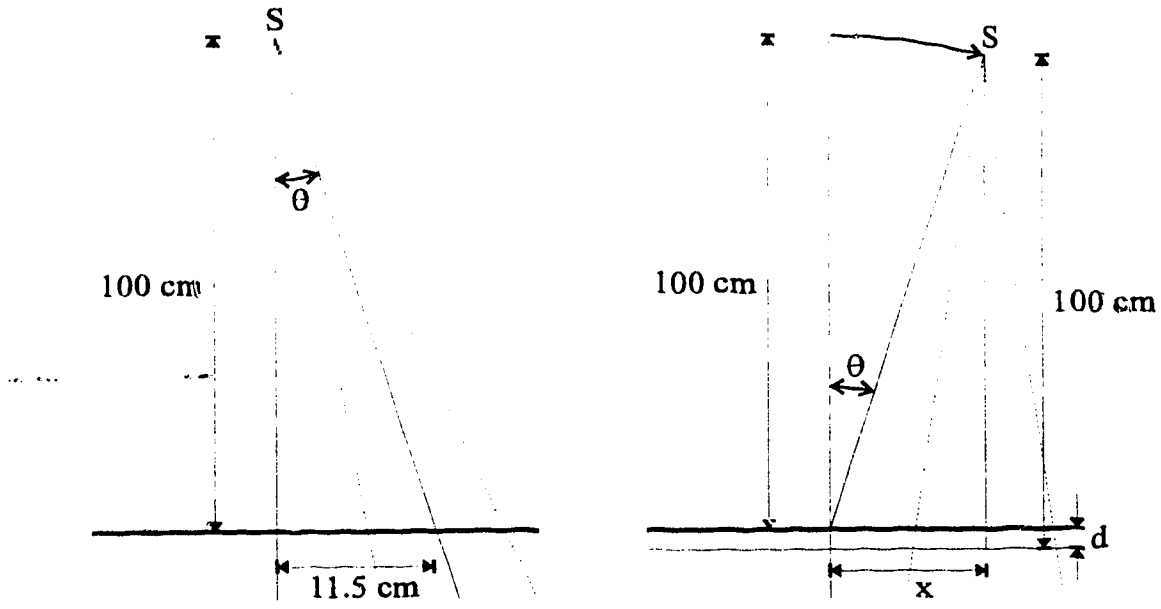


Figure 4.17. Parameters for the rotated gantry arrangement.

Using the configuration just described, percent depth doses were measured along the beam axis (i.e. vertically down the center of the field), for field sizes of 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9 and 10x10 cm^2. Peak scatter factors were also measured for these fields, and the procedure for constructing TAR's and SAR's (section 3.2) was performed on this off-axis data. The off-axis TAR's for three field sizes is shown in figure 4.18, and SAR's are shown in figure 4.19.

Figure 4.18c shows that at large depths the TAR differs by just over 2% of the maximum value. Thus there is the possibility of a significant discrepancy in calculated dose for asymmetric fields if central axis TAR data is used. This will explain the discrepancies seen, for instance, in the large depth regions of the extreme asymmetric fields in figure 4.13e and also in figure 4.12a of section 4.2. The central axis TAR data used by GRATIS overestimates the actual TAR's for these far asymmetric fields, and as a result the GRATIS isodoses are pushed down relative to the measured isodoses (i.e. GRATIS shows higher dose than measured). An easy test to verify the suggested effect on the isodoses is to recalculate the field in figure 4.12a (10x10, X=0 Y=11.4, YZ-iso X=0) using the new off-axis TAR data. This is shown in figures 4.20.

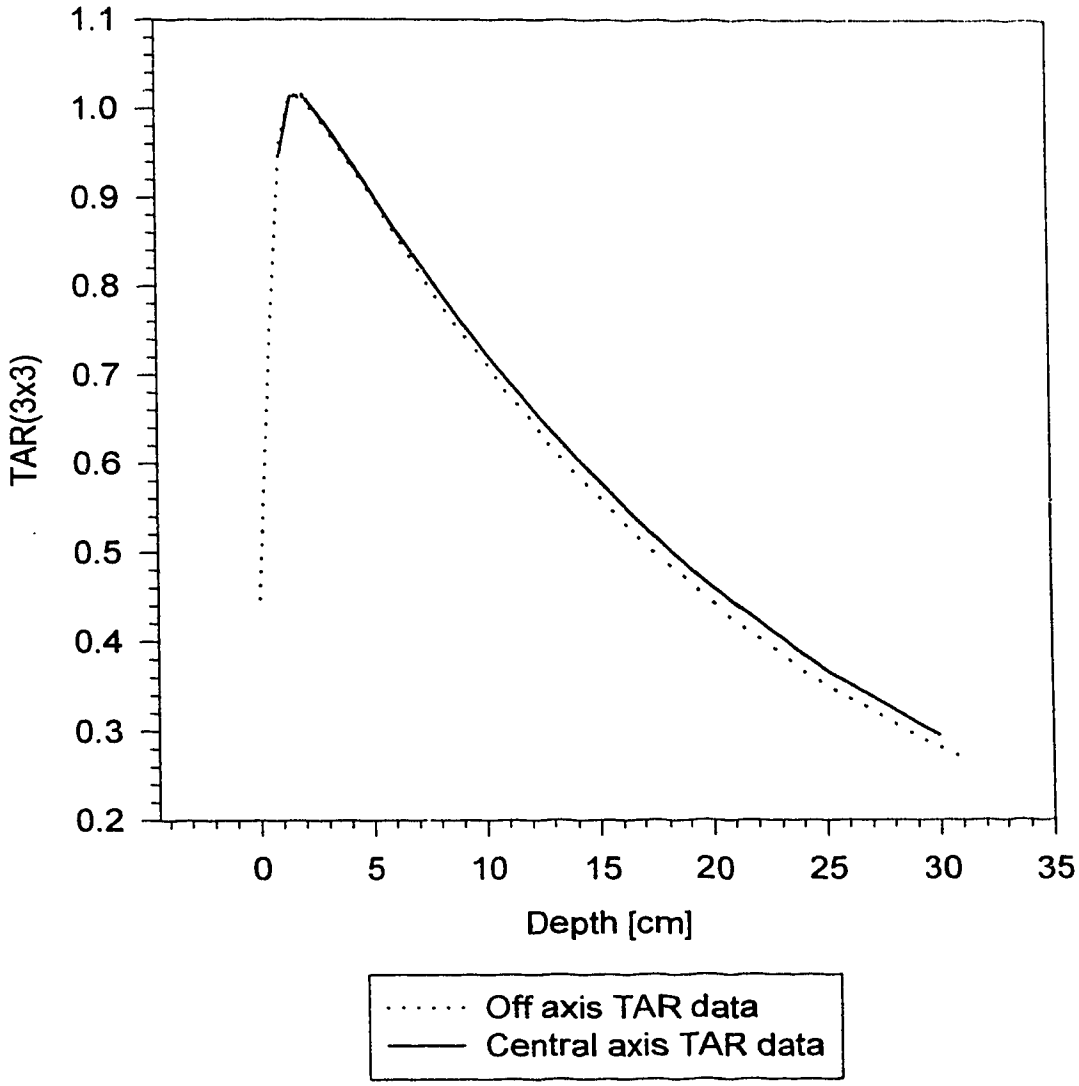


Figure 4.18a. Comparison of off-axis and central axis TAR's for a 3x3 cm^2 field.

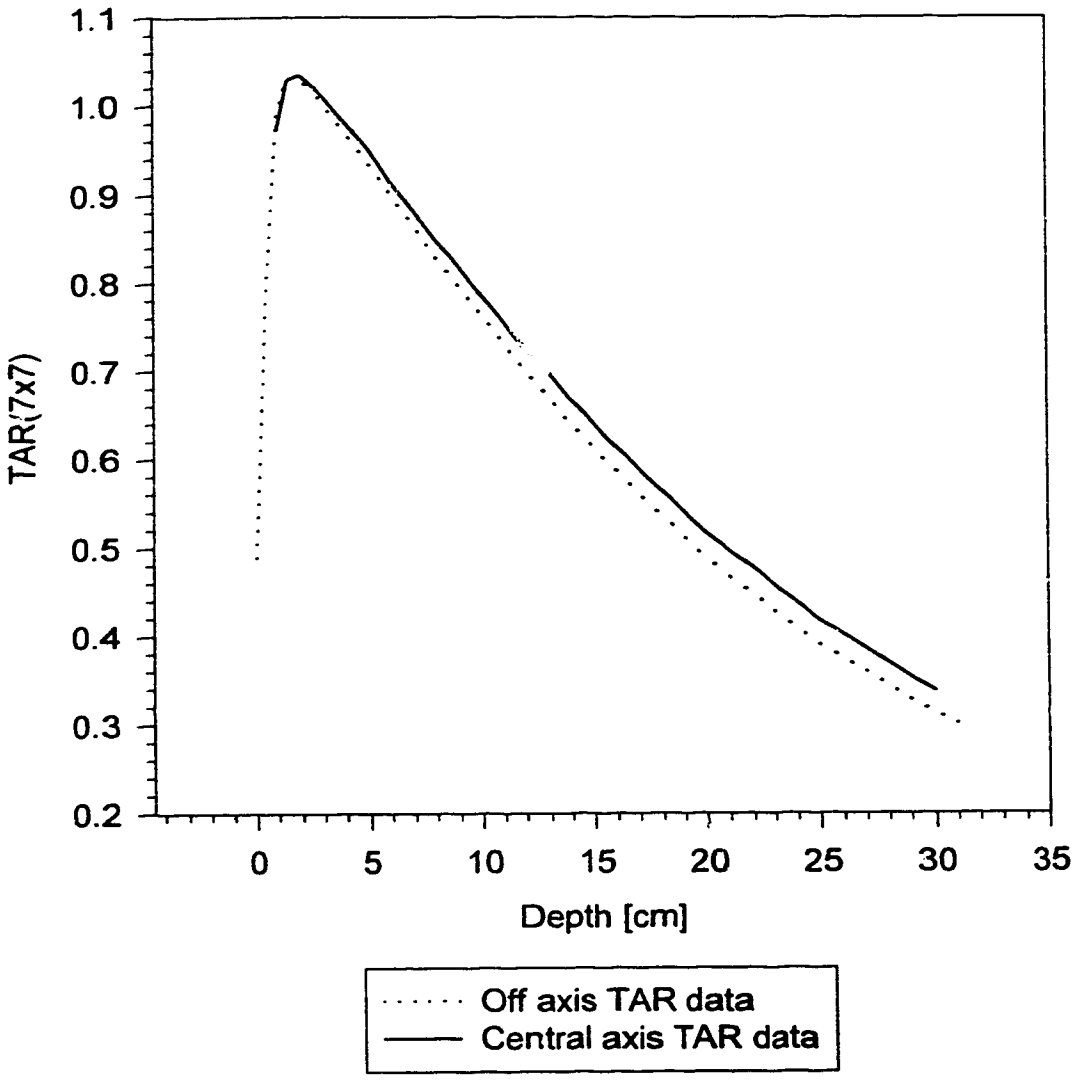


Figure 4.18b. Comparison of off-axis and central axis TAR's for a 7x7 cm^2 field.

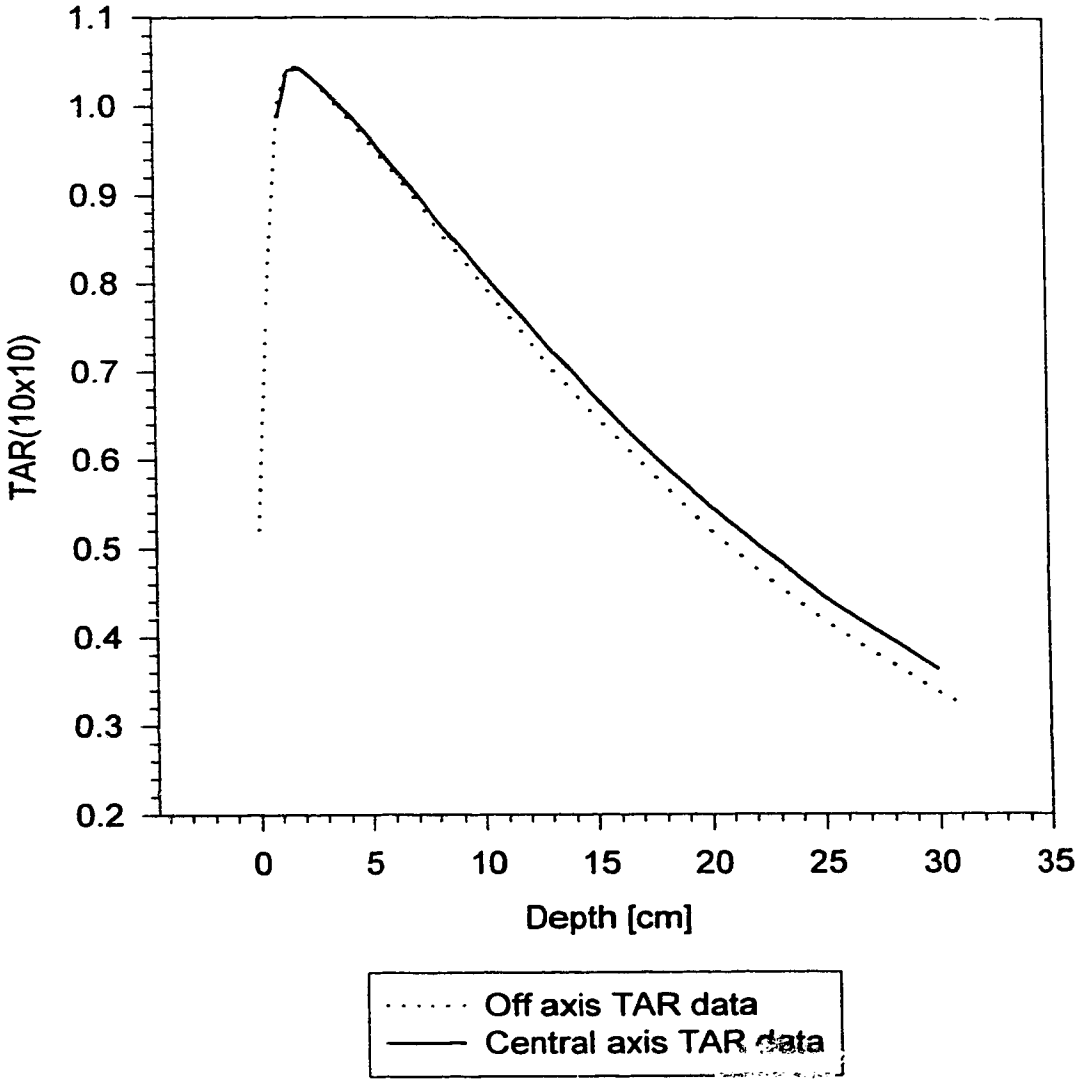


Figure 4.18c. Comparison of off-axis and central axis TAR's for a 10x10 cm^2 field.

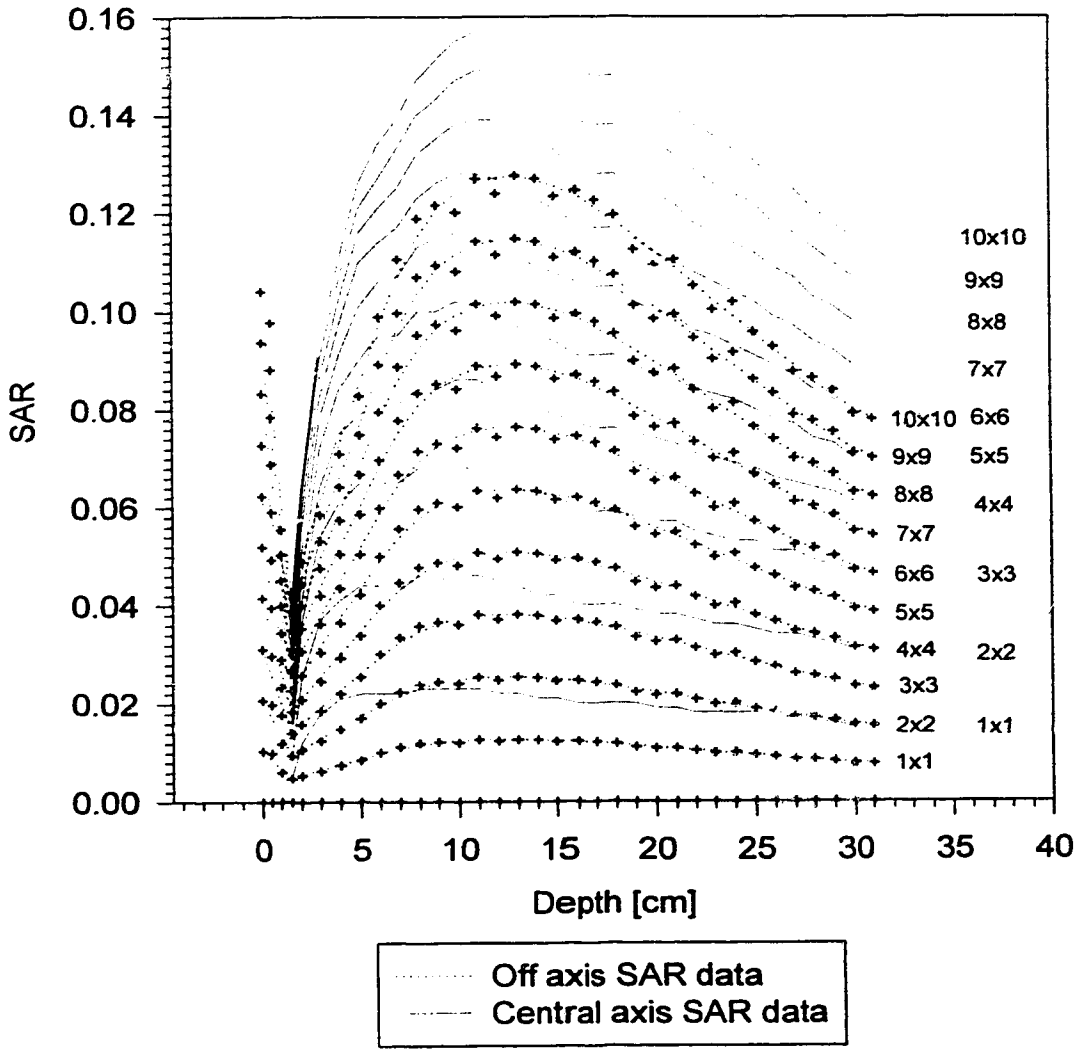


Figure 4.19. Comparison of off-axis and central axis SAR's.

Figure 4.20a. 10x10, X=0 Y=11.4, YZ-iso X=0, off-axis SAR data.

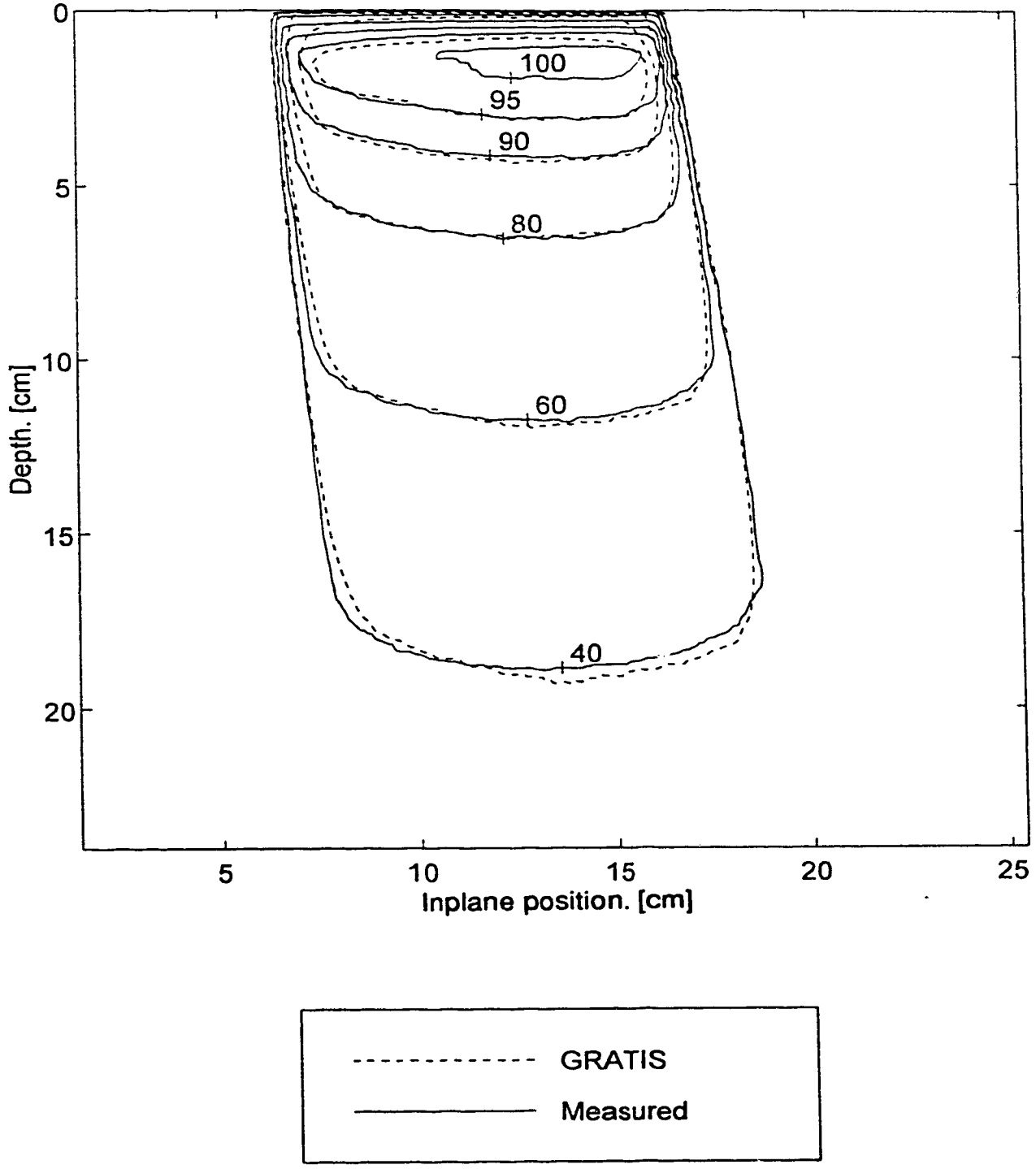


Figure 4.20b. 10x10, X=0 Y=11.4, YZ-iso X=0, 4% discrepancy level.

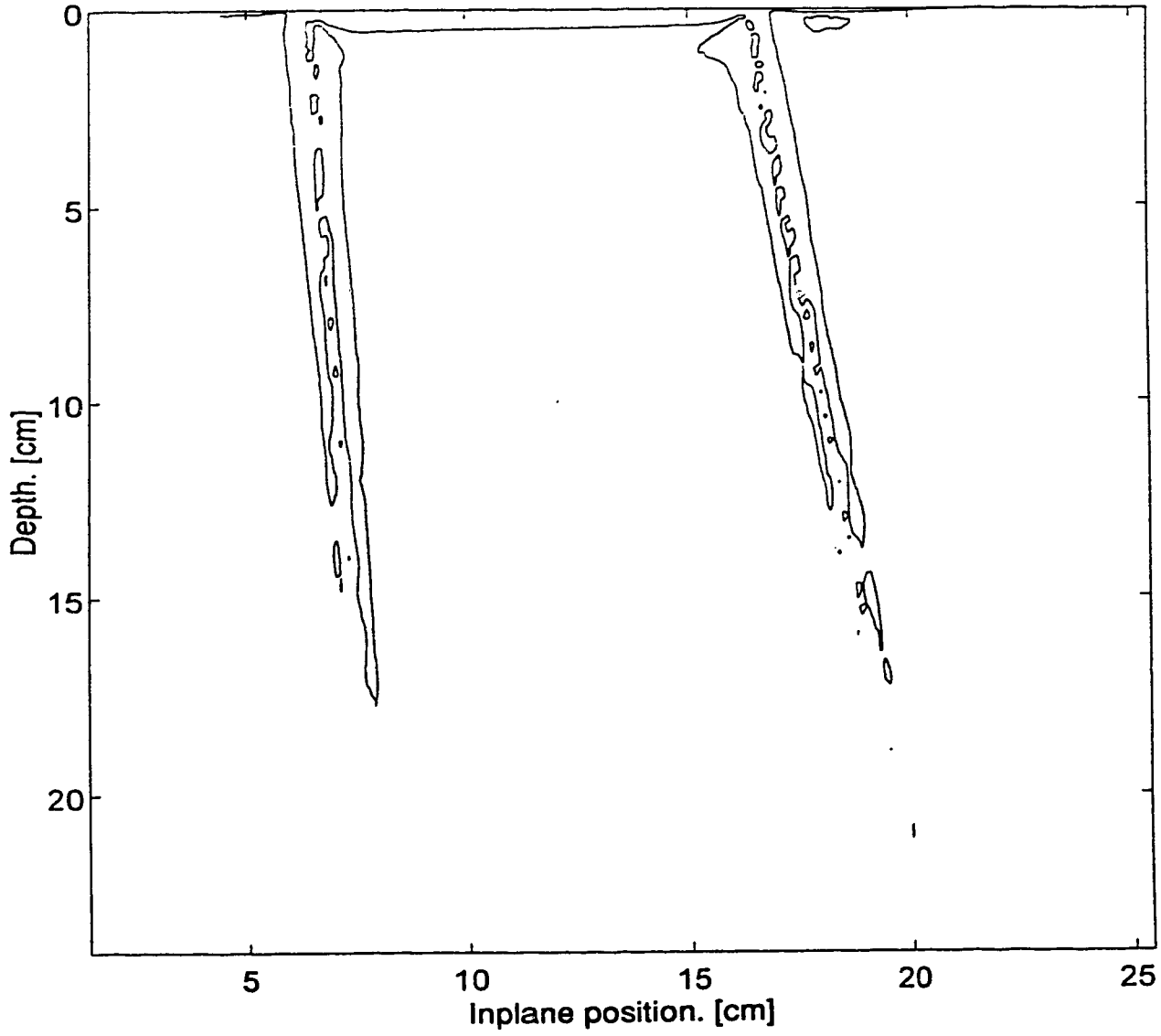


Figure 4.20c. 10x10, X=0 Y=11.4, YZ-iso X=0, 2% discrepancy level.

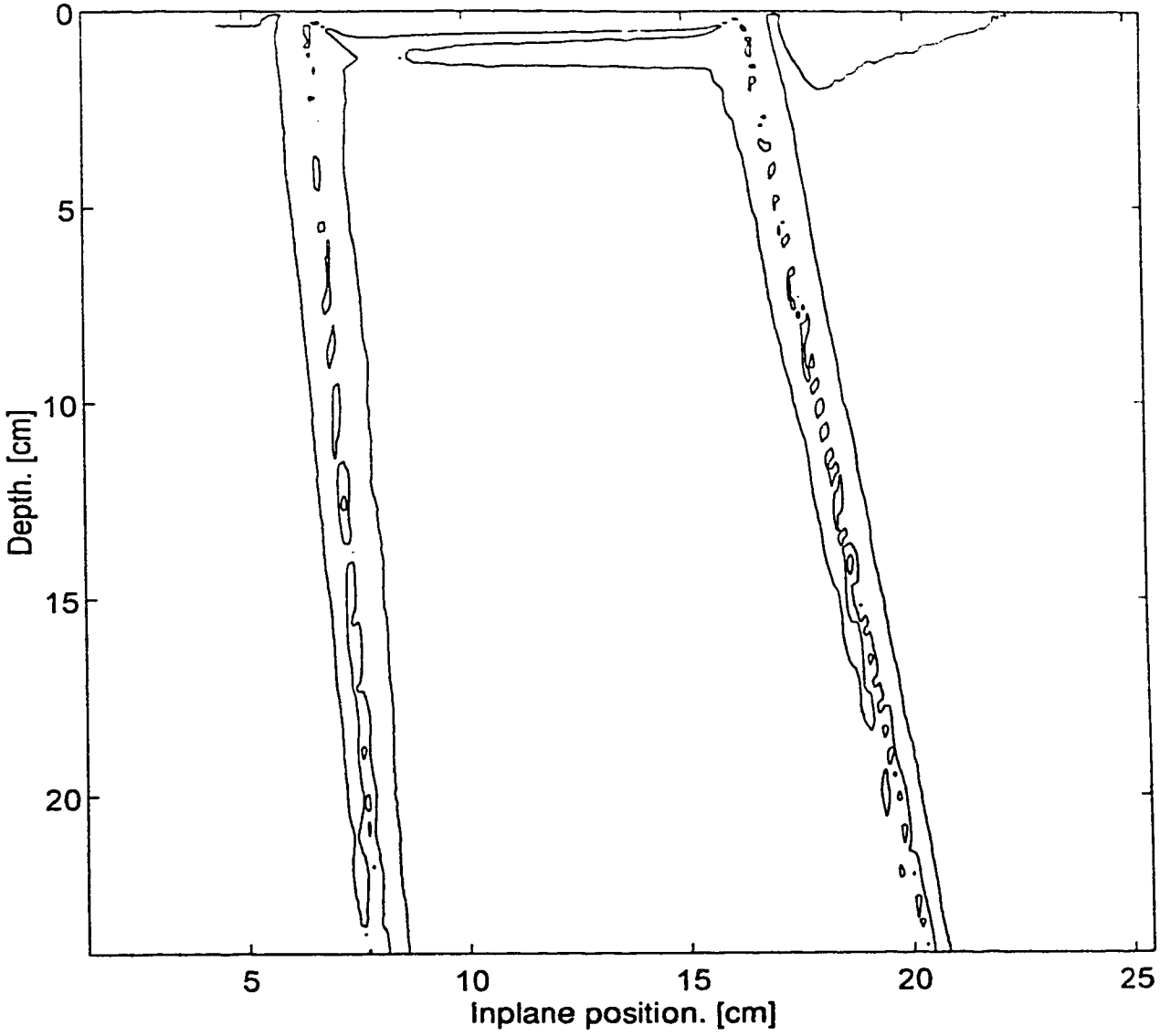
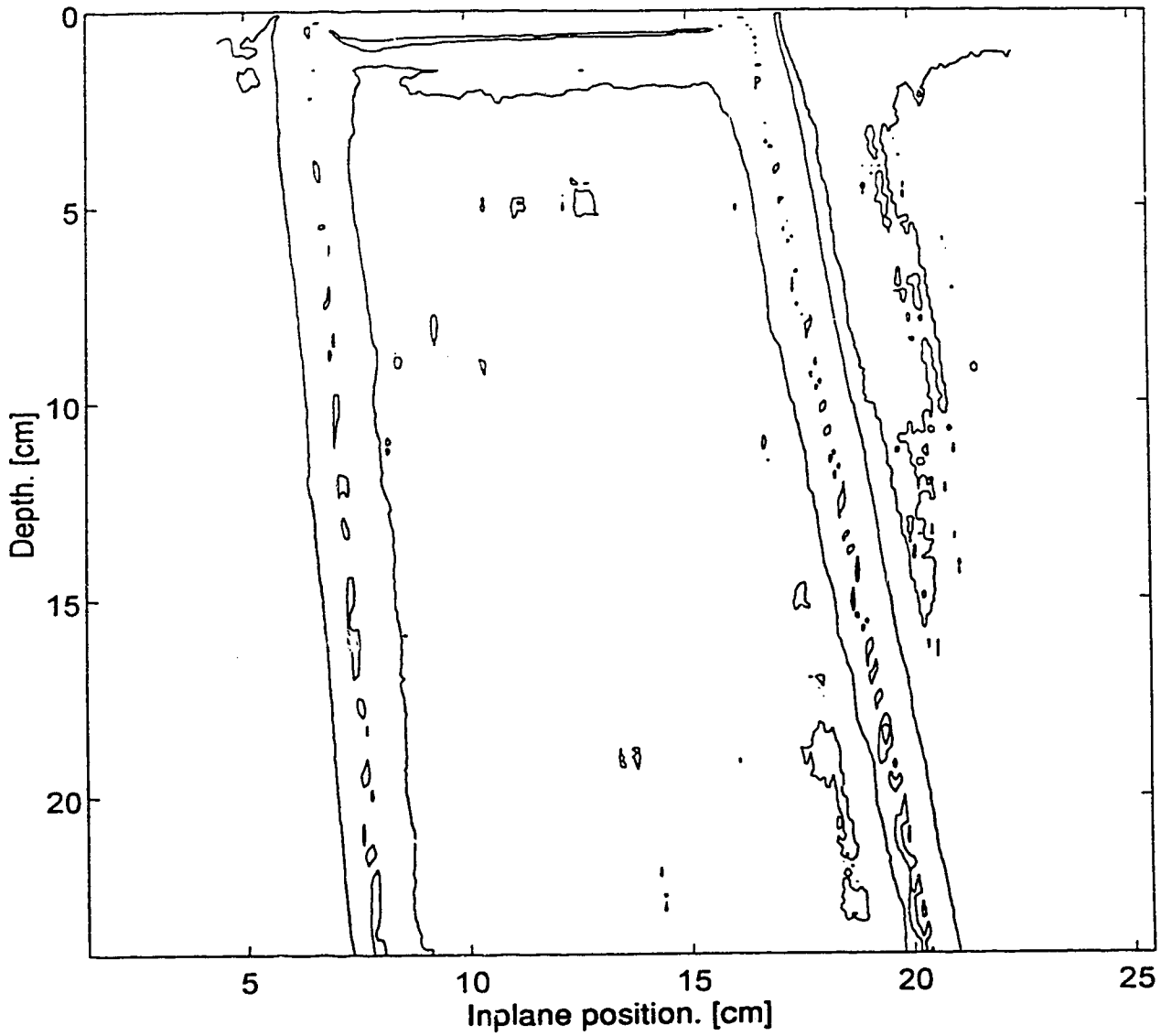


Figure 4.20d. 10x10, X=0 Y=11.4, YZ-iso X=0, 1% discrepancy level.



The measured isodoses in figure 4.20a have been normalized to 100% at a point with a depth of 1.5 cm and a crossplane position of 11.4 cm. However, the calculated dose values still seem to be in disagreement in the shallow depths, and so normalizing to a point at a depth of 1.5 cm would pull all the isodoses out of agreement. The calculated isodoses were thus normalized to a point at 3 cm depth and 11.4 cm crossplane position, to the same dose value as the corresponding point in the measured data. This normalization effectively forces the 95% isodoses to agree, rather than the 100% isodoses. The 95% isodose also agreed very well in the 10x10 X=0 Y=11.4 plot that used central axis SAR data in figure 4.12a, and thus the two figures (4.12a and 4.20a) are easily comparable. The off-axis SAR data has made a noticeable improvement in the isodose agreement for all isodoses except the 100% isodose in the shallow depth region. Also, the 1% discrepancy plot in figure 4.12d has been improved in figure 4.20d since the large region of over 1% discrepancy has been eliminated. It thus seems that beam softening with distance from the central axis has a noticeable effect on the isodoses.

5. Summary and Conclusions

In this thesis the limits of accuracy of the scatter integration method of dose calculation have been investigated and, where possible, extended to a wider range of beam geometries. Test cases for symmetric, single asymmetric and dual asymmetric fields were studied and resulted in significant modifications to the calculation algorithm as implemented in GRATIS. As well, the analysis necessitated an in depth look at the theory behind the scatter integration method (or dSAR method), resulting in a clarification of the empirical parameters used in the algorithm and more rigorous definitions.

Discrepancies between measured and calculated isodoses for symmetric fields, and the subsequent analysis of the code in GRATIS, indicated a problem with how measured SAR data was used in the reconstruction of dose. The depth and dartboard radius parameters used to look up the differential SAR data during the scatter integration were found to be inconsistent with the manner in which the SAR's were originally acquired, and this led to a redefinition of the scatter pencils. The result was a reinterpretation of the "dartboard" from one that was shifted to one that is tilted, and the changes made to the GRATIS code to reflect this new interpretation produced significant improvement in isodose agreement for large symmetric fields. It was then suggested that the remaining discrepancies in the large symmetric fields was due to beam softening with distance from the central axis, and further analysis of beam spectrum effects was deferred until more data could be collected from the asymmetric field investigations.

The tests of single asymmetric fields uncovered an assumption of symmetric fields that was originally made and documented in the GRATIS code. This assumption made the construction of the fluence and contour maps dependent upon the position of only two of the four collimators, resulting in zero scatter contribution for fields in which these two collimators crossed the central axis. The code changes to remove this assumption were relatively simple, and the single asymmetric test cases were repeated with the new version of GRATIS. The analysis that followed again suggested that beam softening effects might be responsible for the small discrepancies observed.

Dual asymmetric fields also suggested that the change in beam quality off axis might have a significant effect, so an experiment to measure off-axis SAR data and use this data in GRATIS was performed. The off-axis data resulted in a noticeable improvement in the isodose agreement for extreme asymmetric fields, indicating that the change in beam quality has a small but significant effect on the isodoses.

The experiment discussed in section 4.3 to examine the effect of beam softening on the calculation suggests that the scatter integration method could be further generalized by tabulating SAR data for beam axes other than the central axis. That is, the SAR data could be made dependent upon the distance of the calculation point from the central axis, in addition to the dependence upon field size and calculation point depth. However, the dSAR's are constructed by partitioning the circular field SAR's into sectors of equal size, and it is assumed that the scatter contribution from each sector is equal. This is true on the central axis where the fluence has radial symmetry, but around another beam axis the radial symmetry is lost and the assumption made in the dSAR construction breaks down. Thus, although the experiment in section 4.3 suffices to demonstrate the significance of beam softening, it cannot easily be used to generalize the scatter integration algorithm. A generalization of the algorithm to account for beam softening could still be made if another technique was used to construct dSAR's with dependence upon the calculation point to central axis distance,. Monte Carlo, for instance, might be used to generate dSAR's rather than constructing dSAR's from measured data.

The penumbra model was not investigated in this thesis, since the discrepancies discovered between measured and calculated isodoses were explainable by limitations in the scatter integration. Testing of the calculation routines for still more general configurations, such as beam modifying devices and arbitrary shaped fields, may uncover limitations in the penumbra model being used, in which case an improvement in the source model should be investigated.

More general situations that still remain to be investigated include contoured surfaces, tissue inhomogeneities, beam modifiers, shaped fields, extended SSDs, and different beam energies. However, the scatter integration method of dose calculation was still found to be fairly accurate for the asymmetric cases examined in this project, provided that the quantities involved are well established and controlled.

The main advantage of this method over the two other methods discussed in chapter two is its speed of calculation, and since it can be made to be reasonably accurate for asymmetric fields, the scatter integration method remains a good choice for the dose calculator in such configurations.

6. Bibliography

Anders Ahnesjö, Mikael Saxner, Avo Trepp,
A Pencil Beam Model for Photon Dose Calculation,
Medical Physics, 1992, Vol. 19, No. 2, pp. 263-273.

Anders Ahnesjö, Tommy Knöös, Anders Montelius,
Application of the Convolution Method for Calculation of Output Factors for Therapy Photon Beams,
Medical Physics, 1992, Vol. 19, No. 2, pp. 295-301.

P. Andreo,
Monte-Carlo Techniques in Medical Radiation Physics,
Physics in Medicine & Biology, 1991, Vol. 36, pp. 861-920.

Frank H. Attix,
Introduction to Radiological Physics and Radiation Dosimetry,
Copyright © 1986 by John Wiley & Sons, Inc.

K. Ayyangar, J. R. Palta, J. W. Sweet, N. Suntharalingam,
Experimental Verification of a Three-Dimensional Dose Calculation Algorithm Using a Specially Designed Heterogeneous Phantom,
Medical Physics, 1993, Vol. 20, No. 2, pp. 325-329.

J. D. Bourland, E. L. Chaney,
A Finite-Size Pencil Beam Model for Photon Dose Calculations in Three Dimensions,
Medical Physics, 1992, Vol. 19, No. 6, pp. 1401-1412.

Arthur L. Boyer, Edward C. Mok,
Calculation of Photon Dose Distributions in an Inhomogeneous Medium Using Convolutions,
Medical Physics, 1986, Vol. 13, No. 4, pp. 503-509.

Arthur L. Boyer, Edward C. Mok,
A Photon Dose Distribution Model Employing Convolution Calculations,
Medical Physics, 1985, Vol. 12, No. 2, pp. 169-177.

J. E. Burns,
Conversion of Percentage Depth Doses for Photon Beams from One SSD to Another and Calculation of TAR, TMR and TPR,
British Journal of Radiology, 1983, Supplement 17, pp. 115-119.

E. L. Chaney, T. J. Cullip,
A Monte Carlo Study of Accelerator Head Scatter,
Medical Physics, 1994, Vol. 21, No. 9, pp. 1383-1390.

Chen-Shou Chui, Radhe Mohan,
Extraction of Pencil Beam Kernels by the Deconvolution Method,
Medical Physics, 1988, Vol. 15, No. 2, pp. 138-144.

J. R. Cunningham,
Scatter-Air Ratios,
Physics in Medicine & Biology, 1972, Vol. 17, No. 1, pp. 42-51.

G. E. Desobry, A. L. Boyer,
An Analytic Calculation of the Energy Fluence Spectrum of a Linear Accelerator,
Medical Physics, 1994, Vol. 21, No. 12, pp. 1943-1952.

Ellen El-Khatib, J. J. Battista,
Improved Lung Dose Calculation Using Tissue-Maximum Ratios in the Batho Correction,
Medical Physics, 1984, Vol. 11, No. 3, pp. 279-286.

C. Field, J. J. Battista,
Photon Dose Calculations Using Convolution in Real and Fourier Space: Assumptions and Time Estimates,
The Use of Computers in Radiation Therapy, 1987, p. 103.

Tawfiq K. Haider, Ellen E. El-Khatib,
Differential Scatter Integration in Regions of Electronic Non-Equilibrium,
Physics in Medicine & Biology, 1995, Vol. 40, pp. 31-43.

Akira Iwasaki,
A Convolution Method for Calculating 10-MV X-Ray Primary and Scatter Dose Including Electron Contamination Dose,
Medical Physics, 1992, Vol. 19, No. 4, pp. 907-915.

Akira Iwasaki,
10-MV X-Ray Primary and Scatter Dose Calculation Using Convolutions,
Medical Physics, 1990, Vol. 17, No. 2, pp. 203-211.

H. E. Johns, J. R. Cunningham,
The Physics of Radiology, 4th Ed.,
Copyright © 1983 by Charles C. Thomas, Publisher.

C. J. Karzmark,
Advances in Linear Accelerator Design for Radiotherapy,
Medical Physics, 1984, Vol. 11, No. 2, pp. 105-127.

Faiz M. Khan, Bruce J. Gerbi, Firmin C. Deibel,
Dosimetry of Asymmetric X-Ray Collimators,
Medical Physics, 1986, Vol. 13, No. 6, pp. 936-941.

Faiz M. Khan, Wilfred Sewchand, Joseph Lee, Jeffrey F. Williamson,
Revision of Tissue-Maximum Ratio and Scatter-Maximum Ratio Concepts for Cobalt 60 and Higher Energy X-Ray Beams,
Medical Physics, 1980, Vol. 7, No. 3, pp. 230-237.

T. R. Mackie, J. W. Scrimger, J. J. Battista,
A Convolution Method of Calculating Dose for 15-MV X Rays,
Medical Physics, 1985, Vol. 12, No. 2, pp. 188-196.

T. R. Mackie, J. W. Scrimger,
Computing Radiation Dose for High Energy X-Rays Using a Convolution Method,
IEEE Computer Society Reprint, 1984, pp. 36-40,
Reprinted from Proceedings of the Eighth International Conference on the Use of Computers in Radiation Therapy, July 9-12, 1984.

G. Marinello, A. Dutreix,
A General Method to Perform Dose Calculations Along the Axis of Symmetrical and Asymmetrical Photon Beams,
Medical Physics, 1992, Vol. 19, No. 2, pp. 275-281.

R. Mohan, et. al.,
Three-Dimensional Dose Calculations for Radiation Treatment Planning,
International Journal of Radiation Oncology Biology Physics, 1991, Vol. 21, pp. 25-36.

Walter R. Nelson, Hideo Hirayama, David W. O. Rogers,
The EGS4 Code System
SLAC-Report-265, December 1985
Stanford Linear Accelerator Center, Stanford University, Stanford, California.

Paul S. Nizin,
Geometrical Aspects of Scatter-to-Primary Ratio and Primary Dose,
Medical Physics, 1991, Vol. 18, No. 2, pp. 153-160.

Simon J. Thomas,
A Modified Power-Law Formula for Inhomogeneity Corrections in Beams of High-Energy X Rays,
Medical Physics, 1991, Vol. 18, No. 4, pp. 719-723.

S. Webb, R. P. Parker,
A Monte Carlo Study of the Interaction of External Beam X-Radiation with Inhomogeneous Media,
Physics in Medicine & Biology, 1978, Vol. 23, No. 6, pp. 1043-1059.

John W. Wong, James A. Purdy,
On Methods of Inhomogeneity Corrections for Photon Transport,
Medical Physics, 1990, Vol. 17, No. 5, pp. 807-814.

John W. Wong, R. Mark Henkelman,
A New Approach to CT Pixel-Based Photon Dose Calculations in Heterogeneous Media,
Medical Physics, 1983, Vol. 10, No. 2, pp. 199-208.

M. K. Woo, A. Fung, P. O'Brien,
Treatment Planning for Asymmetric Jaws on a Commercial TP System,
Medical Physics, 1992, Vol. 19, No. 5, pp. 1273-1275.

7. Appendices

The following sections include information that is relevant to this thesis, but also too lengthy or technical to be presented in the main text.

Appendix A contains a mathematical analysis of the scatter integration that was originally performed to determine exactly what factors are implemented in the GRATIS code, and what factors are neglected. It also served to rigorously define the quantities used in GRATIS so that modifications to the code (and the consequences) were well understood.

Appendix B contains a listing of the C-language code for the GRATIS routine (photon_point_dose) that implements the scatter integration. Both the original version and the version with all the final modifications are presented for comparison.

Appendix C contains a listing of the C-language code for the GRATIS routine (mk_contour_map) that constructs the table of SSD's. Both the original version and the version with all the final modifications are presented for comparison.

Appendix D contains a listing of the C-language code for the GRATIS routine (mk_dSAR_table) that constructs the table of dSAR's from the SAR data. Both the original version and the version with all the final modifications are presented for comparison.

7.1. Appendix A. Examination of the dose calculation algorithm in GRATIS.

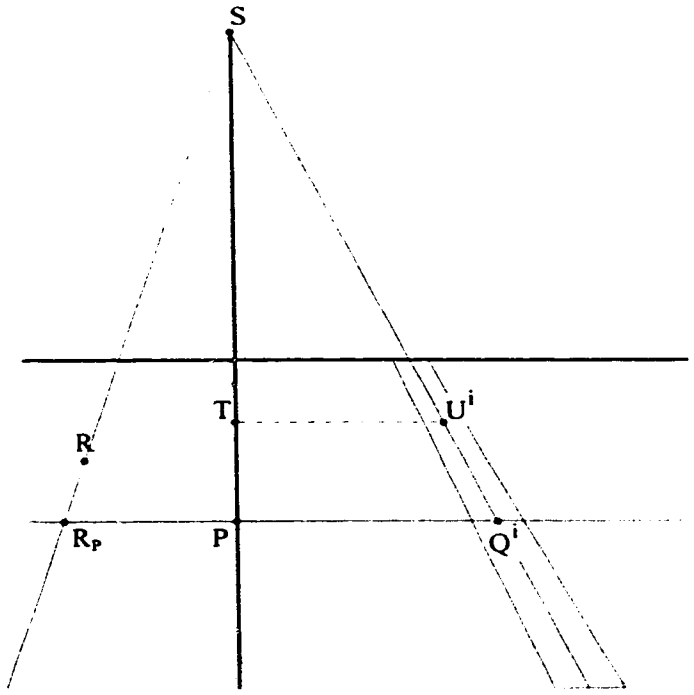


Figure 7.1. Geometry for central axis measurements.

Similar triangles.

$$\frac{SQ^i}{SP} = \frac{SU^i}{ST}$$

Definition of collimator scatter factor.

$$S_{coll}(P) = \frac{D^P_{air}}{D^P_{air,no-coll}}$$

Inverse-square relation along a rayline.

$$(ST)^2 \cdot D^T_{air,no-coll} = (SP)^2 \cdot D^P_{air,no-coll}$$

Inverse-square relation along a rayline.

$$\frac{D_{air}^{P}}{D_{air}^{T}} = \frac{(ST)^2}{(SP)^2} \cdot \frac{S_{coll}(P)}{S_{coll}(T)}$$

Definition of beam intensity (relative to P).

$$\text{rel\_intensity}_P(R) = \frac{D_{air}^{R}}{D_{air}^{P}}$$

Definition of beam flatness (relative to P).

$$\text{flatness}_P(R) = \text{rel\_intensity}_P(R_P) = \frac{D_{air}^{R_P}}{D_{air}^{P}}$$

Expression of the differential-scatter-air-ratio as a ratio of doses.
(For use in the derivation of pt_SAR.)

$$\Delta SAR(P,Q^i) = \frac{\left(D_{scat_i}^{P} \Big/ \text{rel\_intensity}_P(Q^i) \right)}{D_{air}^{P}}$$

$$= \frac{D_{scat_i}^{P}}{D_{air}^{P}} \cdot \frac{1}{\text{rel\_intensity}_P(Q^i)}$$

$$= \frac{D_{scat_i}^{P}}{D_{air}^{P}} \cdot \frac{1}{\left(D_{air}^{Q^i} \Big/ D_{air}^{P} \right)}$$

$$= \frac{D_{scat_i}^{P}}{D_{air}^{Q^i}}$$

Note that $Q^i = Q_P^i$ when ΔSAR is generated in GRATIS, so that in the above we have

$$\text{rel\_intensity}_P(Q^i) = \text{flatness}_P(Q^i)$$

$$\text{rel\_intensity}_P(R) = \frac{D_{air}^R}{D_{air}^P}$$

$$= \frac{D_{air}^R}{D_{air}^{R_P}} \cdot \frac{D_{air}^{R_P}}{D_{air}^P}$$

$$= \frac{D_{air}^R}{D_{air}^{R_P}} \cdot \text{rel\_intensity}_P(R_P)$$

$$= \frac{D_{air}^R}{D_{air}^{R_P}} \cdot \text{flatness}_P(R)$$

$$= \frac{(SR_P)^2}{(SR)^2} \cdot \frac{S_{coll}(R)}{S_{coll}(R_P)} \cdot \text{flatness}_P(R)$$

$$= \frac{(SP)^2}{(SR_z)^2} \cdot \frac{S_{coll}(R)}{S_{coll}(R_P)} \cdot \text{flatness}_P(R)$$

Which gives us the following substitutions for the pt_TAR$_o$ and pt_SAR calculations.

$$\text{rel\_intensity}_C(A) = \frac{(SC)^2}{(SA_z)^2} \cdot \frac{S_{coll}(A)}{S_{coll}(A_c)} \cdot \text{flatness}_C(A)$$

$$\text{rel\_intensity}_C(B^i) = \frac{(SC)^2}{(SB_z^i)^2} \cdot \frac{S_{coll}(B^i)}{S_{coll}(B_c^i)} \cdot \text{flatness}_C(B^i)$$

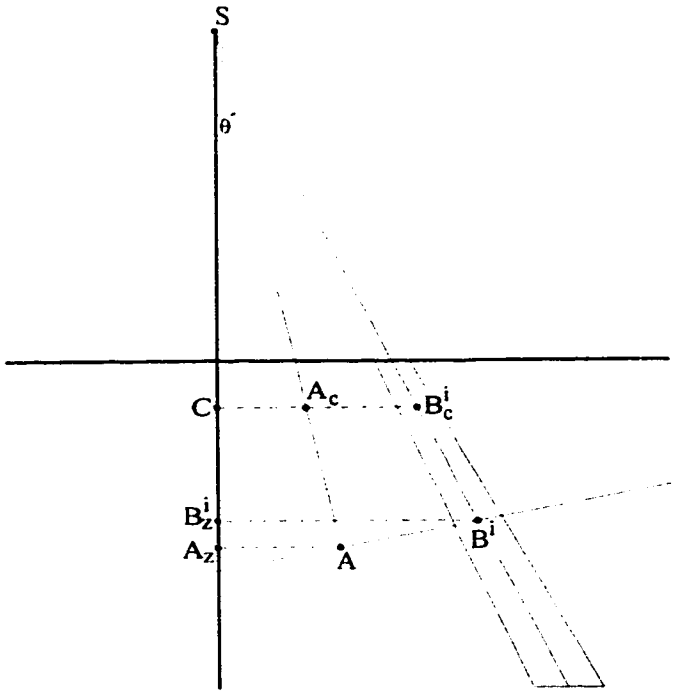


Figure 7.2. Geometry for off-axis calculations.

Dose calculation algorithm used by GRATIS.

$$
\begin{aligned}
D_{total}^{A} &= D_{prim}^{A} + D_{scat}^{A} \\
&= D_{air}^{C} \cdot \left\{ \frac{D_{prim}^{A}}{D_{air}^{C}} + \frac{D_{scat}^{A}}{D_{air}^{C}} \right\} \\
&= \left\{ D_{air}^{C} \cdot (SC)^2 \right\} \cdot \left\{ \frac{1}{(SC)^2} \cdot \frac{D_{prim}^{A}}{D_{air}^{C}} + \frac{1}{(SC)^2} \cdot \frac{D_{scat}^{A}}{D_{air}^{C}} \right\} \\
&= \{global\_factor\} \cdot \{pt\_TAR_0 + pt\_SAR\} \\
&= \{global\_factor\} \cdot \{photon\_point\_dose\}
\end{aligned}
$$

$$
D_{air}^{C} = S_{coll}(C) \cdot D_{air,no-coll}^{C} = \{lookup\_Sc\} \cdot \{cal\_dose\_rate\}
$$

$$
\begin{aligned}
pt\_TAR_0 &= \frac{1}{(SC)^2}\cdot\frac{D^A_{prim}}{D^C_{air}} \\
&= \frac{1}{(SC)^2}\cdot\frac{D^A_{air}}{D^C_{air}}\cdot\frac{D^A_{prim}}{D^A_{air}} \\
&= \frac{1}{(SC)^2}\cdot rel\_intensity_C(A)\cdot TAR_0(A) \\
&= \frac{1}{(SC)^2}\cdot\left[\frac{(SC)^2}{(SA_z)^2}\cdot\frac{S_{coll}(A)}{S_{coll}(A_c)}\cdot flatness_C(A)\right]\cdot TAR_0(A) \\
&= \frac{1}{(SA_z)^2}\cdot\frac{S_{coll}(A)}{S_{coll}(A_c)}\cdot flatness_C(A)\cdot TAR_0(A)
\end{aligned}
$$

$$
\begin{aligned}
pt\_SAR &= \frac{1}{(SC)^2}\cdot\frac{D^A_{scat}}{D^C_{air}} \\
&= \frac{1}{(SC)^2}\cdot\frac{1}{D^C_{air}}\cdot\sum_i D^A_{scat_i} \\
&= \frac{1}{(SC)^2}\cdot\sum_i\left\{\frac{D^{B^i}_{air}}{D^C_{air}}\cdot\frac{D^A_{scat_i}}{D^{B^i}_{air}}\right\} \\
&= \frac{1}{(SC)^2}\cdot\sum_i\left\{rel\_intensity_C(B^i)\cdot\Delta SAR(A,B^i)\right\} \\
&= \frac{1}{(SC)^2}\cdot\sum_i\left\{\left[\frac{(SC)^2}{(SB^i_z)^2}\cdot\frac{S_{coll}(B^i)}{S_{coll}(B^i_c)}\cdot flatness_C(B^i)\right]\cdot\Delta SAR(A,B^i)\right\} \\
&= \sum_i\left\{\frac{1}{(SB^i_z)^2}\cdot\frac{S_{coll}(B^i)}{S_{coll}(B^i_c)}\cdot flatness_C(B^i)\cdot\Delta SAR(A,B^i)\right\}
\end{aligned}
$$

GRATIS currently makes the implicit assumption that S_{coll} does not vary significantly along a rayline.

i.e. $$\frac{S_{coll}(A)}{S_{coll}(A_c)} \cong 1 \qquad \text{and} \qquad \frac{S_{coll}(B^i)}{S_{coll}(B^i_c)} \cong 1$$

The calculations currently performed by GRATIS are then

$$pt\_TAR_0 = \frac{1}{(SA_z)^2} \cdot flatness_C(A) \cdot TAR_0(A)$$

$$pt\_SAR = \sum_i \left\{ \frac{1}{(SB^i_z)^2} \cdot flatness_C(B^i) \cdot \Delta SAR(A, B^i) \right\}$$

$$D^C_{air} = S_{coll}(C) \cdot D^C_{air,no-coll} = \{lookup\_Sc\} \cdot \{cal\_dose\_rate\}$$

$$D^A_{total} = \{global\_factor\} \cdot \{pt\_TAR_0 + pt\_SAR\}$$

Variation of beam flatness profile with distance from source.

$$\frac{D_{air}^{Q^i}}{D_{air}^{P}} = \left[\frac{D_{air}^{Q^i}\cdot(SQ^i)^2}{S_{coll}(Q^i)}\right]\cdot\left[\frac{S_{coll}(P)}{D_{air}^{P}\cdot(SP)^2}\right]\cdot\frac{(SP)^2}{(SQ^i)^2}\cdot\frac{S_{coll}(Q^i)}{S_{coll}(P)}$$

$$= \left[\frac{D_{air}^{U^i}\cdot(SU^i)^2}{S_{coll}(U^i)}\right]\cdot\left[\frac{S_{coll}(T)}{D_{air}^{T}\cdot(ST)^2}\right]\cdot\frac{(SP)^2}{(SQ^i)^2}\cdot\frac{S_{coll}(Q^i)}{S_{coll}(P)}$$

$$= \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\left(\frac{SU^i}{ST}\frac{SP}{SQ^i}\right)^2\cdot\frac{S_{coll}(Q^i)}{S_{coll}(U^i)}\cdot\frac{S_{coll}(T)}{S_{coll}(P)}$$

$$= \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\left(\frac{SQ^i}{SP}\frac{SP}{SQ^i}\right)^2\cdot\frac{S_{coll}(Q^i)}{S_{coll}(U^i)}\cdot\frac{S_{coll}(T)}{S_{coll}(P)}$$

$$= \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\frac{S_{coll}(Q^i)}{S_{coll}(U^i)}\cdot\frac{S_{coll}(T)}{S_{coll}(P)}$$

or $$\frac{D_{air}^{Q^i}}{D_{air}^{P}}\cdot\frac{S_{coll}(P)}{S_{coll}(Q^i)} = \frac{D_{air}^{U^i}}{D_{air}^{T}}\cdot\frac{S_{coll}(T)}{S_{coll}(U^i)}$$

If $S_{coll}(Q^i) = S_{coll}(U^i)$ (i.e. if collimator scatter is not dependent upon distance from source) then

$$\frac{D_{air}^{Q^i}}{D_{air}^{P}} = \frac{D_{air}^{U^i}}{D_{air}^{T}}$$

7.2. **Appendix B.** Code listing of photon_point_dose.c

This appendix contains a listing of the code for the GRATIS routine photon_point_dose, which contains the scatter integration. Both the original version and the version with all the final modifications are listed side by side to facilitate comparisons. Also, the code changes that are relevant to the tilting of the dartboard are shown in bold to set them apart from changes that involve debugging statements, documentation and reformatting of C code.

photon_point_dose.c.v0

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/photon_point_dose.c.v 3.3
        1990/10/14 16:25:58 sherouse Exp $";

/*
 * $Log: photon_point_dose.c.v $
 * Revision 3.3  1990/10/14  16:25:58  sherouse
 * - use library version of exact_fluence()
 *
 * Revision 3.2  89/01/26  16:26:13  sherouse
 * - more fiddling around with points on upstream side of patient
 *
 * Revision 3.1  89/01/18  14:45:04  sherouse
 * - support new beam description
 * - add filters
 * - make it OK to calculate dose at 0 depth
 *
 * Revision 2.2  88/05/31  17:23:01  sherouse
 * - spruced up documentation (just a tad)
```

photon_point_dose.c.v3

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/photon_point_dose.c.v 3.3
        1990/10/14 16:25:58 sherouse Exp $";

/*
 * $Log: photon_point_dose.c.v $
 *
 * Revision 3.6  1994/07/27  11:30:00  jkollar
 * - Document the debugging switches and clean them up a little.
 *
 * Revision 3.5  1994/07/06  15:58:00  jkollar
 * - Change the inverse square scaling to use the z-coordinate of the
 *   calculation point, rather than the rayline distance from the source.
 *   We feel that the z-distance is the correct one to use here.
 *   This inverse square scaling is required to adjust the final dose
 *   from being at the reference distance (which is where the tables and
 *   maps are defined) to being at the correct calculation point distance.
 * - Add an "inverse square" correction to the sector fluence that makes
 *   the fluence be relative to the calculation point, rather than relative
 *   to the calibration point (as the original fluence table is defined).
 *   We believe that this is the proper way to weight the scatter pencils,
 *   giving us the total SAR at the calculation point.
 *
 * Revision 3.4  1994/02/12  17:27:00  jkollar
 * - Tip the dartboard to improve the algorithm's accuracy.
 * - Dartboard is now normal to the source-calc_pt rayline,
 * - rather than normal to the central axis.
 *
 * Revision 3.3  1990/10/14  16:25:58  sherouse
 * - use library version of exact_fluence()
 *
 * Revision 3.2  89/01/26  16:26:13  sherouse
 * - more fiddling around with points on upstream side of patient
 *
 * Revision 3.1  89/01/18  14:45:04  sherouse
 * - support new beam description
 * - add filters
 * - make it OK to calculate dose at 0 depth
 *
 * Revision 2.2  88/05/31  17:23:01  sherouse
 * - spruced up documentation (just a tad)
```

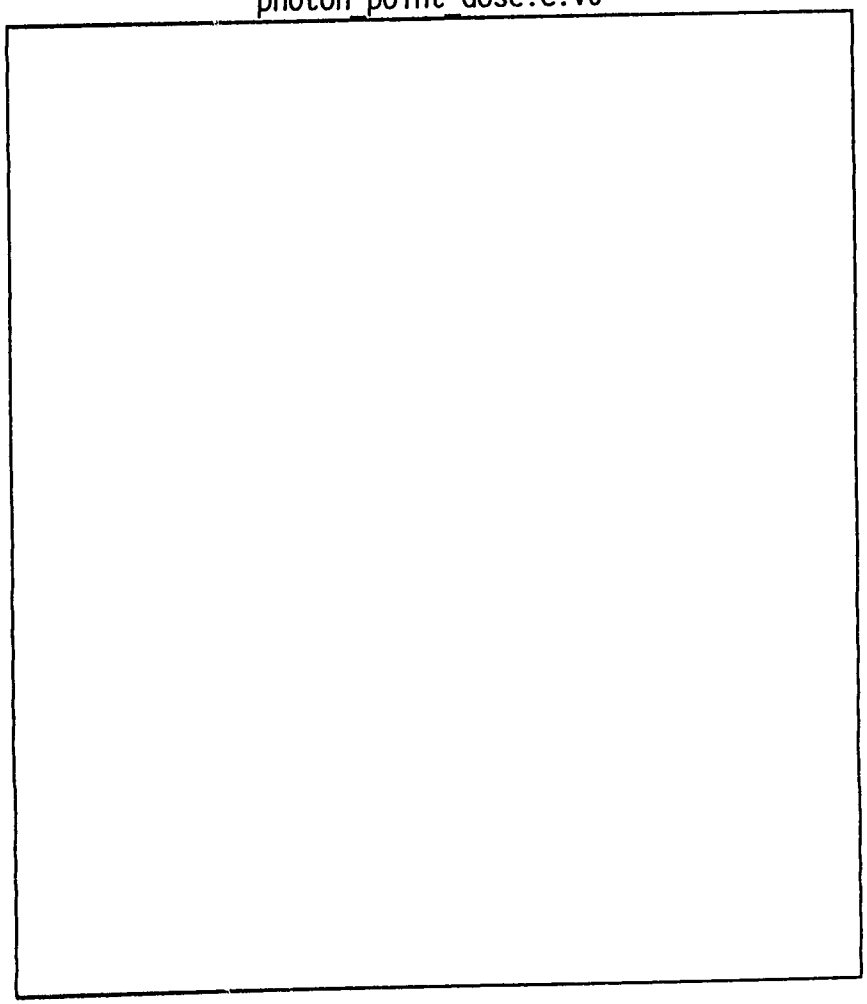
photon_point_dose.c.v0

```
 * Revision 2.1  88/05/31  09:43:20  sherouse
 * - miraculous transformation into a near-production version
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define SQRT(x) ((float) sqrt((double)(x)))
#define exact_TAR0(x) v_interp(0, request.SAR_table.depth_count,\
                                  request.SAR_table.depth_vec, (x),\
                                  request.SAR_table.TAR0, &index, &fx)
```

photon_point_dose.c.v3

```
 * Revision 2.1  88/05/31  09:43:20  sherouse
 * - miraculous transformation into a near-production version
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define SQRT(x) ((float) sqrt((double)(x)))
#define exact_TAR0(x) v_interp(0, request.SAR_table.depth_count,\
                                  request.SAR_table.depth_vec, (x),\
                                  request.SAR_table.TAR0, &index, &fx)

/*****************************************************************************/
/* Debugging switches.
*/

/* George Sherouse */
/* Print a message informing when a grid point is outside the patient.       */
/* #define EBUG1 */

/* George Sherouse */
/* Print a message of the form "T?R0 for depth ?.?? is ??.??" where the '?'s */
/* are determined just after the primary has been calculated, but before the */
/* scatter integration has begun.                                            */
/* #define EBUG2 */

/* John R. Kollar */
/* Print out the BOOLEAN variable "inside" and the float variables           */
/* "nearest", "point_depth" and "pathlength", for each calculation point.    */
/* These variables are the result of a call to:                              */
/*    prism (&request.patient_anastruct, patient_source, patient_pt,         */
/*           &inside, &nearest, &point_depth, &pathlength);                  */
/* #define J_DEBUG1 */
```

photon_point_dose.c.v0



photon_point_dose.c.v3

```
/* John R. Kollar */
/* Print out the int variables "rloop" & "ang_loop", and the float          */
/* variables "tables.dSAR_radii[rloop]" & "sec_fluence", for each scatter   */
/* pencil.                                                                  */
/* John R. Kollar W9407.27 10:00 */
/* Print out the float variables "sec_depth_along_pt" and "sec_z_dr..n" for  */
/* each scatter pencil.                                                     */
/* Print out the POINT variable "sc" and the float variable "sc_dist" for   */
/* each scatter pencil.                                                     */
/* #define J_DEBUG2 */

/* John R. Kollar */
/* Print out the int variables "tables.dSAR_sector_count" and               */
/* "tables.dSAR_radius_count" for each calculation point.                   */
/* #define J_DEBUG3 */

/* John R. Kollar */
/* Print the float values of "pt_TAR0", "pt_SAR" & "pt_TAR0 + pt_SAR" just  */
/* before the inverse-square term is multiplied in.                         */
/* These values ar  printed out for each calculation point.                 */
/* #define J_DEBUG4 */

/* John R. Kollar */
/* Print out the float variable "pt_TAR0" and the POINT variable "beam_pt"  */
/* (i.e. the calculation point) just before the inverse-square term is      */
/* multiplied in. These values are printed out for each calculation point.  */
/* #define J_DEBUG5 */

/* John R. Kollar T9407.26 15:00 */
/* Use a sector fluence of 1.0 in the scatter integration. (But not in the  */
/* primary calculation.) This results in pt_SAR = sum(dSAR*1.0).            */
/* #define J_DEBUG6 */
```

photon_point_dose.c.v0

```
float photon_point_dose (patient_source, patient_pt, pt_TAR0, pt_SAR)
POINT patient_source;
POINT patient_pt;
float *pt_TAR0;
float *pt_SAR;
```

photon_point_dose.c.v3

```
/* John R. Kollar */
/* Print out the int variable "request.SAR_table.depth_count", the float     */
/* variables "point_dist" & "sec_SSD", the float value returned by the       */
/* function call                                                             */
/*    v_interp (0, request.SAR_table.depth_count,                            */
/*                 request.SAR_table.depth_vec, sec_depth_along_pt,          */
/*                 tables.dSAR[rloop], &index, &fx).                         */
/* and the resulting values of the int variable "index" and the float        */
/* variable "fx" immediately following the above function call.              */
/* These values are printed for each scatter pencil.                         */
/* #define J_DEBUG7 */

/* John R. Kollar W9407.27 10:23 */
/* Print out the float variable "request.unit.SAD" and the POINT variable   */
/* "beam_pt" for each calculation point.                                     */
/* #define J_DEBUG8 */

/* John R. Kollar T9408.02 14:37 */
/* Print out the int variables "contour_map_x" and "contour_map_y" for each  */
/* scatter pencil.                                                           */
/* #define J_DEBUG9 */

/* John R. Kollar F9408.05 14:01 */
/* Print out the "exact_fluence" used in the primary calculation, for each   */
/* calculation point.                                                        */
/* #define J_DEBUG10 */

/*****************************************************************************/

float photon_point_dose (patient_source, patient_pt, pt_TAR0, pt_SAR)
POINT patient_source;
POINT patient_pt;
float *pt_TAR0;
float *pt_SAR;
```

photon_point_dose.c.v0

```
/*
-------------------------------------------------------------------------

NAME
      photon_point_dose - here's where the work gets done

SYNOPSIS

DESCRIPTION
      This routine actually looks up the zero-area T*R, does the
      sector integration of S*R, and scales everything by the
      appropriate inverse square.

RETURN VALUE
                                                2
      (T*R0 + S*R) / (point distance)   Note that this is not really
      dose as the name implies but is well on the way.

BUGS

AUTHOR
      George W. Sherouse
      Radiation Oncology
      North Carolina Memorial Hospital
      University of North Carolina
      26 May 1988


      (c) Copyright 1988 - George W. Sherouse
      All rights reserved.

-------------------------------------------------------------------------
*/
```

photon_point_dose.c.v3

```
/*
-------------------------------------------------------------------------

NAME
      photon_point_dose - here's where the work gets done

SYNOPSIS

DESCRIPTION
      This routine actually looks up the zero-area T*R, does the
      sector integration of S*R, and scales everything by the
      appropriate inverse square.

RETURN VALUE
                                                2
      (T*R0 + S*R) / (point distance)   Note that this is not really
      dose as the name implies but is well on the way.

BUGS

AUTHOR
      George W. Sherouse
      Radiation Oncology
      North Carolina Memorial Hospital
      University of North Carolina
      26 May 1988


      (c) Copyright 1988 - George W. Sherouse
      All rights reserved.

SUBSEQUENT DEVELOPERS
      John R. Kollar
      Department of Medical Physics
      Cross Cancer Institute
      11560 University Avenue
      Edmonton, Alberta T6G-1Z2
      Canada
      Phone: 403-492-6776
      Email: jkollar@phys.ualberta.ca
-------------------------------------------------------------------------
*/
```

photon_point_dose.c.v0

```
{
    extern REQUESTrequest.
    extern TABLES tables;

    static float  map_max = ((float) MAP_SIZE) / 4.0;

    POINTbeam_pt;
    floatpoint_dist_sq.
                dist_scale.
                point_dist.
                point_depth.
                nearest.
                pathlength.
                x. y.
                sec_SSD.
                sec_fluence.
                fx;









    BOOLEAN         inside;
    int             ang_loop.
                    rloop.
                    fluence_map_x.
                    fluence_map_y.
                    contour_map_x.
                    contour_map_y.
                    index;

    float     v_interp (). exact_fluence ();
```

photon_point_dose.c.v3

```
{
    extern REQUESTrequest;
    extern TABLES tables.

    static float  map_max = ((float) MAP_SIZE) / 4.0.

    POINTbeam_pt.
    floatpoint_dist_sq.
                dist_scale.
                point_dist.
                point_depth.
                nearest.
                pathlength.
                x. y.
                sec_SSD.
                sec_fluence.
                fx;

    /* John R Kollar Sa9402.12 16:45 */
    double      ang1, ang2.
                C1, S1, C2, S2, C3.
                rot[3][3];
    float       sec_depth_along_pt.
                sec_z_depth.
                sc_dist.
                dist_scale2;
    POINT       sc;

    /* John R. Kollar F9408.05 14:01 */
    float       pt_fluence;

    BOOLEAN         inside;
    int             ang_loop.
                    rloop.
                    fluence_map_x.
                    fluence_map_y.
                    contour_map_x.
                    contour_map_y.
                    index;

    float     v_interp (). exact_fluence ();
```

photon_point_dose.c.v0

```
/*
  Transform the patient coordinates of the grid point into beam
  coordinates and intersect the associated ray with the patient.

  The inside argument is wasted here - we know the source point is not
  inside the patient.
*/
    transform_point (patient_pt, request.beam.T_pat_to_beam, &beam_pt);
    prism (&request.patient_anastruct, patient_source, patient_pt,
            &inside, &nearest, &point_depth, &pathlength);









/*
  Nearest should be set to a very large number if the grid point is
  outside the patient, either upstream or beside.  We currently do not
  flag points outside the patient on the downstream side for fear of
  generating artifactual build-down.  This is a matter of ongoing
  evaluation.
*/
    if (nearest > 1000.0)
    {
#ifdef EBUG
        printf ("grid point outside patient\n");
#endif EBUG
        *pt_TAR0 = 0.0;
        *pt_SAR = 0.0;
        return (0.0);
    }
```

photon_point_dose.c.v3

```
/*
  Transform the patient coordinates of the grid point into beam
  coordinates and intersect the associated ray with the patient.

  The inside argument is wasted here - we know the source point is not
  inside the patient.
*/
    transform_point (patient_pt, request.beam.T_pat_to_beam, &beam_pt);
    prism (&request.patient_anastruct, patient_source, patient_pt,
            &inside, &nearest, &point_depth, &pathlength);

#ifdef J_DEBUG1
    printf("\n");
    printf("inside = %u\n",inside);
    printf("nearest = %f\n",nearest);
    printf("point_depth = %f\n",point_depth);
    printf("pathlength = %f\n",pathlength);
#endif J_DEBUG1

/*
  Nearest should be set to a very large number if the grid point is
  outside the patient, either upstream or beside.  We currently do not
  flag points outside the patient on the downstream side for fear of
  generating artifactual build-down.  This is a matter of ongoing
  evaluation.
*/
    if (nearest > 1000.0)
    {
#ifdef EBUG1
        printf ("grid point outside patient\n");
#endif EBUG1
        *pt_TAR0 = 0.0;
        *pt_SAR = 0.0;
        return (0.0);
    }

#ifdef J_DEBUG8
    printf("SAD = %f\n",request.unit.SAD);
    printf("beam_pt = (%f,%f,%f)\n",beam_pt.x,beam_pt.y,beam_pt.z);
#endif J_DEBUG8
```

photon_point_dose.c.v0

```
   /* factor to scale beam x and y coordinates to unit isocenter */
    * which is where all the tables are defined.                  */
   dist_scale = request.unit.SAD / beam_pt.z;

   point_dist_sq = beam_pt.x * beam_pt.x +
                   beam_pt.y * beam_pt.y +
                   beam_pt.z * beam_pt.z;
   point_dist = SQRT (point_dist_sq);

   *pt_TAR0 = exact_TAR0 (point_depth) *
        exact_fluence (beam_pt, point_depth, dist_scale,
                       (request.shaped_field) ?
                       request.tray.block_dist :
                       request.unit.SDD,
                       &request.beam, &request.unit,
                       &request.SAR_table, &request.filter);


#ifdef EBUG
    printf ("%s0 for depth %f is %f\n",
            T_THING (request.SAR_table.type),
            point_depth,
            exact_TAR0 (point_depth));
#endif EBUG
```

photon_point_dose.c.v3

```
   /* factor to scale beam x and y coordinates to unit isocenter */
   /* which is where all the tables are defined.                  */
   dist_scale = request.unit.SAD / beam_pt.z;

   point_dist_sq = beam_pt.x * beam_pt.x +
                   beam_pt.y * beam_pt.y +
                   beam_pt.z * beam_pt.z;
   point_dist = SQRT(point_dist_sq);

   pt_fluence = exact_fluence (beam_pt, point_depth, dist_scale,
                       (request.shaped_field) ?
                       request.tray.block_dist :
                       request.unit.SDD,
                       &request.beam, &request.unit,
                       &request.SAR_table, &request.filter);
   *pt_TAR0 = exact_TAR0 (point_depth) * pt_fluence;

#ifdef EBUG2
    printf ("%s0 for depth %f is %f\n",
            T_THING (request.SAR_table.type),
            point_depth,
            exact_TAR0 (point_depth));
#endif EBUG2

#ifdef J_DEBUG10
    printf("exact_fluence(%f,%f,%f) = %f\n",beam_pt.x,beam_pt.y,beam_pt.z,
            pt_fluence);
#endif J_DEBUG10
```

photon_point_dose.c.v0

```
*pt_SAR = 0.0;

for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
{
     /*
      * The sectors at rloop=0 are not interesting as dSAR==0.0.  Hence we
      * start rloop at 1. See mk_dSAR_table.
      */
     for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
     {
         x = (beam_pt.x + tables.sec_center_x[ang_loop][rloop]) *
              dist_scale;
         y = (beam_pt.y + tables.sec_center_y[ang_loop][rloop]) *
              dist_scale;
```

photon_point_dose.c.v3

```
   /* John R Kollar Sa9402.12 14:31 */
   /* Create rotation matrix for tipping the dartboard. */
   ang1 = atan2(beam_pt.y,beam_pt.x);
   ang2 = atan2(SQRT(beam_pt.x*beam_pt.x + beam_pt.y*beam_pt.y),beam_pt.z);
   C1 = cos(ang1); S1 = sin(ang1);
   C2 = cos(ang2); S2 = sin(ang2);
   C3 = C2 - 1;
   rot[0][0] = C1*C1*C3 + 1;
   rot[1][0] = C1*S1*C3;
   rot[2][0] = -C1*S2;
   rot[0][1] = rot[1][0];
   rot[1][1] = S1*S1*C3 + 1;
   rot[2][1] = -S1*S2;
   /* rot[0][2] = -rot[2][0]; */
   /* rot[1][2] = -rot[2][1]; */
   /* rot[2][2] = C3 + 1;     */

   *pt_SAR = 0.0;

#ifdef J_DEBUG3
   printf("\n");
   printf("sector count = %d\n",tables.dSAR_sector_count);
   printf("radius count = %d\n",tables.dSAR_radius_count);
#endif J_DEBUG3

   for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
   {
       /*
        * The sectors at rloop=0 are not interesting as dSAR==0.0.  Hence we
        * start rloop at 1. See mk_dSAR_table.
        */
       for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
       {
          /* John R Kollar Sa9402.12 15:02 */
          /* sc = sector center for the tipped dartboard. */
          sc.x = beam_pt.x + rot[0][0]*tables.sec_center_x[ang_loop][rloop]
                           + rot[0][1]*tables.sec_center_y[ang_loop][rloop];
          sc.y = beam_pt.y + rot[1][0]*tables.sec_center_x[ang_loop][rloop]
                           + rot[1][1]*tables.sec_center_y[ang_loop][rloop];
          sc.z = beam_pt.z + rot[2][0]*tables.sec_center_x[ang_loop][rloop]
                           + rot[2][1]*tables.sec_center_y[ang_loop][rloop];
```

photon_point_dose.c.v0

```
/*
  This needs some proving but it sure speeds things up...
*/
            if (rloop > 2)
            {
                 if ((x > request.beam.jaw[X_JAW].pos_2) ||
                     (x < request.beam.jaw[X_JAW].pos_1))
                     continue;
                 if ((y > request.beam.jaw[Y_JAW].pos_2) ||
                     (y < request.beam.jaw[Y_JAW].pos_1))
                     continue;
            }

/*
  Compute index into contour map (recall it is only filled at the
  center) and into the fluence map (recall it is fleshed out).

  In both cases, if the current point is beyond the edge of the map,
  extrapolate by using the map edge value.
*/
            contour_map_y = fluence_map_y =
                 (int) ((map_max - y) * 2.0);




            if (fluence_map_y < 0)
                 fluence_map_y = 0;
            else
            if (fluence_map_y >= MAP_SIZE)
                 fluence_map_y = MAP_SIZE - 1;
```

photon_point_dose.c.v3

```
            sc_dist = SQRT(sc.x*sc.x + sc.y*sc.y + sc.z*sc.z);
            /* Project sc back to isocenter for table lookup of */
            /* patient contour, fluence and filter values. */
            dist_scale2 = request.unit.SAD/sc.z;
            x = dist_scale2*sc.x;
            y = dist_scale2*sc.y;


/*
  This needs some proving but it sure speeds things up...
*/
            if (rloop > 2)
            {
                 if ((x > request.beam.jaw[X_JAW].pos_2) ||
                     (x < request.beam.jaw[X_JAW].pos_1))
                     continue;
                 if ((y > request.beam.jaw[Y_JAW].pos_2) ||
                     (y < request.beam.jaw[Y_JAW].pos_1))
                     continue;
            }

/*
  Compute index into contour map (recall it is only filled at the
  center) and into the fluence map (recall it is fleshed out).

  In both cases, if the current point is beyond the edge of the map,
  extrapolate by using the map edge value.
*/
            contour_map_y = fluence_map_y =
                 (int) ((map_max - y) * 2.0);

#ifdef J_DEBUG9
            printf("contour_map_y = %d    ",contour_map_y);
#endif J_DEBUG9

            if (fluence_map_y < 0)
                 fluence_map_y = 0;
            else
            if (fluence_map_y >= MAP_SIZE)
                 fluence_map_y = MAP_SIZE - 1;
```

photon_point_dose.c.v0

```
        if (contour_map_y > tables.con_map_max_y_index)
             contour_map_y = tables.con_map_max_y_index;
        else
        if (contour_map_y < tables.con_map_min_y_index)
             contour_map_y = tables.con_map_min_y_index;

        contour_map_x = fluence_map_x =
             (int) ((x + map_max) * 2.0);




        if (fluence_map_x < 0)
             fluence_map_x = 0;
        else
        if (fluence_map_x >= MAP_SIZE)
             fluence_map_x = MAP_SIZE - 1;

        if (contour_map_x > tables.con_map_max_x_index)
             contour_map_x = tables.con_map_max_x_index;
        else
        if (contour_map_x < tables.con_map_min_x_index)
             contour_map_x = tables.con_map_min_x_index;


/*
  See if there is any patient under this sector.
*/
        sec_SSD = tables.contour_map[contour_map_y][contour_map_x];

        if (sec_SSD == 0.0)
             continue;
```

photon_point_dose.c.v3

```
        if (contour_map_y > tables.con_map_max_y_index)
             contour_map_y = tables.con_map_max_y_index;
        else
        if (contour_map_y < tables.con_map_min_y_index)
             contour_map_y = tables.con_map_min_y_index;

        contour_map_x = fluence_map_x =
             (int) ((x + map_max) * 2.0);


#ifdef J_DEBUG9
          printf("ccntour_map_x = %d\n".contour_map_x);
#endif J_DEBUG9

        if (fluence_map_x < 0)
             fluence_map_x = 0;
        else
        if (fluence_map_x >= MAP_SIZE)
             fluence_map_x = MAP_SIZE - 1;

        if (contour_map_x > tables.con_map_max_x_index)
             contour_map_x = tables.con_map_max_x_index;
        else
        if (contour_map_x < tables.con_map_min_x_index)
             contour_map_x = tables.con_map_min_x_index;


/*
  See if there is any patient under this sector.
*/
        sec_SSD = tables.contour_map[contour_map_y][contour_map_x];

        if (sec_SSD == 0.0)
             continue;

        /* John R Kollar Sa9402.12 15:27                */
        /* Depth values required for table lookup of */
        /* dSAR and fluence values.                     */
        sec_depth_along_pt = point_dist - sec_SSD*point_dist/sc_dist;
        sec_z_depth = sc.z - sec_SSD*sc.z/sc_dist;
```

photon_point_dose.c.v0

```
            if (sec_SSD >= point_dist)
                continue;

/*
  Can't avoid it - look up nearest neighbor fluence and dSAR for depth
  corresponding to nearest neighbor SSD.
*/
            sec_fluence =
                (request.SAR_table.flatness_depth_count == 1) ?
                    tables.fluence_map[fluence_map_y][fluence_map_x][0] :
                    v_interp (0, request.SAR_table.flatness_depth_count,
                        request.SAR_table.flatness_depth_vec,
                        point_dist - sec_SSD,
                        tables.fluence_map[fluence_map_y][fluence_map_x],
                        &index, &fx);
```

photon_point_dose.c.v3

```
            /* John R Kollar M9411.14 12:00                                  */
            /* This is a fudge to try and account for a slowly varying       */
            /* patient contour. The idea is that the differential scatter    */
            /* not only depends upon the "depth" of the scatter pencil,      */
            /* but also on the amount of tissue above the calculation point. */
            /* That is, the dSAR table was constructed from measured data    */
            /* that includes the contribution from multiple scatter, and     */
            /* hence the dose at the calculation point depends on scattering */
            /* from surrounding phantom material other than the pencil.      */
            /* (The pencil can be thought of as the "source" of 1st          */
            /* scattered radiation).                                         */
            /* sec_depth_along_pt = (sec_depth_along_pt + point_depth)/2.0;  */

            /* John R Kollar M9411.14 14:10                                  */
            /* This will eliminate the cases where sec_depth_along_pt < 0    */
            if (sec_depth_along_pt < 0) continue;

/*
  Can't avoid it - look up nearest neighbor fluence and dSAR for depth
  corresponding to nearest neighbor SSD.
*/
            sec_fluence =
                (request.SAR_table.flatness_depth_count == 1) ?
                    tables.fluence_map[fluence_map_y][fluence_map_x][0] :
                    v_interp (0, request.SAR_table.flatness_depth_count,
                        request.SAR_table.flatness_depth_vec,
                        sec_z_depth, /* John R Kollar Sa9402.12 15:33 */
                        tables.fluence_map[fluence_map_y][fluence_map_x],
                        &index, &fx);

            /* John R Kollar W9407.06 16:10                                  */
            /* This establishes a fluence profile along the TIPPED           */
            /* dartboard, with sec_fluence=1.0 at the calculation point.     */
            /* That is, for weighting the scatter pencils we need the        */
            /* fluence along the dartboard "relative to the calculation      */
            /* point", rather than "relative to the calibration point"       */
            /* (which is how the actual fluence MAP is defined).             */
            sec_fluence *= (beam_pt.z*beam_pt.z)/(sc.z*sc.z);
```

photon_point_dose.c.v0

```
/* If there is a filter, look up the filter factor using the fluence
   map indices.
*/
            if (request.beam.filter_count)
                sec_fluence *=
                    request.filter.map[fluence_map_y][fluence_map_x];




            *pt_SAR += sec_fluence *
                v_interp (0, request.SAR_table.depth_count,
                            request.SAR_table.depth_vec,
                            point_dist - sec_SSD,
                            tables.dSAR[rloop], &index, &fx);
```

photon_point_dose.c.v3

```
#ifdef J_DEBUG2
            printf("rloop = %d   r = %f   ang_loop = %d   ",rloop,
                                                tables.dSAR_radii[rloop],ang_loop);
            printf("sec_fluence = %f\n",sec_fluence);
            printf("sc = |(%f,%f,%f)| = %f\n",sc.x,sc.y,sc.z,sc_dist);
            printf("sec_depth_along_pt = %f    ",sec_depth_along_pt);
            printf("sec_z_depth = %f\n\n",sec_z_depth);
#endif J_DEBUG2

/* If there is a filter, look up the filter factor using the fluence
   map indices.
*/
            if (request.beam.filter_count)
                sec_fluence *=
                    request.filter.map[fluence_map_y][fluence_map_x];


#ifdef J_DEBUG6
            sec_fluence = 1.0;
#endif J_DEBUG6

            *pt_SAR += sec_fluence *
                v_interp (0, request.SAR_table.depth_count,
                            request.SAR_table.depth_vec,
                            sec_depth_along_pt /* John R Kollar Sa9402.12 16:50 */,
                            tables.dSAR[rloop], &index, &fx);


#ifdef J_DEBUG7
            printf("depth_count = %d   ",request.SAR_table.depth_count);
            printf("point_dist = %f\n",point_dist);
            printf("sec_SSD = %f   ",sec_SSD);
            printf("index = %d   ",index);
            printf("fx = %f   ",fx);
            printf("dSAR = %f\n\n",
                        v_interp (0, request.SAR_table.depth_count,
                        request.SAR_table.depth_vec,
                        sec_depth_along_pt,
                        tables.dSAR[rloop], &index, &fx));
#endif J_DEBUG7

        }
    }
```

photon_point_dose.c.v0

```
    *pt_SAR /= tables.dSAR_sector_count;









    *pt_SAR /= point_dist_sq;
    *pt_TAR0 /= point_dist_sq;

    return (*pt_TAR0 + *pt_SAR);
}
```

photon_point_dose.c.v3

```
    *pt_SAR /= tables.dSAR_sector_count;

#ifdef J_DEBUG4
    printf("%f = TAR = TAR0 + SAR = %f + %f\n",*pt_TAR0 + *pt_SAR,
                                                    *pt_TAR0,*pt_SAR);
#endif J_DEBUG4

#ifdef J_DEBUG5
    printf("TAR0(%f,%f,%f) = %f\n\n",beam_pt.x,beam_pt.y,beam_pt.z,*pt_TAR0);
#endif J_DEBUG5

    /* John R Kollar W9407.06 15:58 */
    *pt_SAR /= (beam_pt.z * beam_pt.z);
    *pt_TAR0 /= (beam_pt.z * beam_pt.z);

    return (*pt_TAR0 + *pt_SAR);
}
```

7.3. **Appendix C.** Code listing of mk_contour_map.c

This appendix contains a listing of the code for the GRATIS routine mk_contour_map, which constructs the table of SSD's used by the scatter integration. Both the original version and the version with all the final modifications are listed side by side to facilitate comparisons. Also, the code changes that are relevant to the tilting of the dartboard are shown in bold to set them apart from changes that involve debugging statements, documentation and reformatting of C code.

mk_contour_map.c.v0

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_contour_map.c.v 3.2
        1990/10/14 16:24:27 sherouse Exp $";

/*
 * $Log: mk_contour_map.c.v $









 * Revision 3.2  1990/10/14  16:24:27  sherouse
 * - turn of verbose
 *
 * Revision 3.1  89/01/18  14:42:08  sherouse
 * - support new beam description
 * - keep up with changes in UNIT structure
 *
 * Revision 2.2  88/05/31  17:22:46  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:03  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.3  88/05/18  13:55:38  sherouse
 * - working version, saved before major assault
 *
 * Revision 1.2  87/05/22  10:57:52  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:17:41  sherouse
 * Initial revision
 *
 */
```

mk_contour_map.c.v2

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_contour_map.c.v 3.2
        1990/10/14 16:24:27 sherouse Exp $";

/*
 * $Log: mk_contour_map.c.v $
 * Revision 3.4  1994/08/04  21:33:00  jkollar
 * - A bug with revision 3.3 was fixed. The contour was flipped in the
 *   Y-direction, and this showed up in the tests of asymmetric fields.
 *
 * Revision 3.3  1993/11/11  14:27:00  jkollar
 * - The assumption of symmetric jaws in determining the map limits has
 * - been removed. The map now supports asymmetric collimators.
 *
 * Revision 3.2  1990/10/14  16:24:27  sherouse
 * - turn of verbose
 *
 * Revision 3.1  89/01/18  14:42:08  sherouse
 * - support new beam description
 * - keep up with changes in UNIT structure
 *
 * Revision 2.2  88/05/31  17:22:46  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:03  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.3  88/05/18  13:55:38  sherouse
 * - working version, saved before major assault
 *
 * Revision 1.2  87/05/22  10:57:52  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:17:41  sherouse
 * Initial revision
 *
 */
```

mk_contour_map.c.v0

```
#include <stdio.h>
#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define verbose (0)









mk_contour_map()
/*
 ----------------------------------------------------------------------

   NAME
        mk_contour_map - make a map of SSDs for quick reference

   SYNOPSIS


   DESCRIPTION
        This routine builds a table of SSDs on a 5 mm grid as
        described in detail below.

   BUGS
```

mk_contour_map.c.v2

```
#include <stdio.h>
#include "gen.h"
#include "extbm.h"

#include "defines.h"

#define verbose (0)

/* #define EBUG */

/* Print out the jaw x-y positions and the corresponding contour map */
/* indicies. */
/* #define J_DEBUG1 */

/* Print the contour map out to a text file. */
/* #define J_DEBUG2 */

mk_contour_map()
/*
 ----------------------------------------------------------------------

   NAME
        mk_contour_map - make a map of SSDs for quick reference

   SYNOPSIS


   DESCRIPTION
        This routine builds a table of SSDs on a 5 mm grid as
        described in detail below.

   BUGS
```

mk_contour_map.c.v0

```
  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        3 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

 ------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int             xloop,
                    yloop,
                    x_size_in_cells,
                    y_size_in_cells;
    BOOLEAN         inside;

    float           x_inc,
                    y_inc,
                    scale,
                    SSD,
                    thickness,
                    pathlength;


    POINT           bm_source,
                    bm_target,
                    pat_source,
                    pat_target;
```

mk_contour_map.c.v2

```
  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        3 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

 ------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int             xloop,
                    yloop,
                    x_size_in_cells,
                    y_size_in_cells;
    BOOLEAN         inside;

    float           x_inc,
                    y_inc,
                    scale,
                    SSD,
                    thickness,
                    pathlength;


    POINT           bm_source,
                    bm_target,
                    pat_source,
                    pat_target;

    POINT           bt;

#ifdef J_DEBUG2
    FILE *fpr;
#endif J_DEBUG2
```

mk_contour_map.c.v0

```
/*
  Set up a vector from the source to a point on a plane which is
  perpendicular to the c. axis and 200 cm from the source for each
  cell in the contour map.  (There is nothing magic about 200 cm and
  in fact this should probably be handled a little more
  intelligently.)  Recall - you *have* read defines.h - that the
  contour map is defined as a 5 mm grid at the beams definition
  distance layed out so that the central axis intersects the corner of
  four cells - there are an even number of cells across a rectangular
  beam.

  Having set up the ray in beam coordinates, transform to patient
  coordinates and intersect with the patient.   Store the SSD in the
  grid matrix - a value >1000.0 means no intersection.

  The map extends only to the geometric edge (+ epsilon) of the beam.
  Beyond that one should do bidirectional nearest-neighbor.
*/

    bm_source.x = bm_source.y = bm_source.z = 0.0;
    transform_point (bm_source, request.beam.T_beam_to_pat, &pat_source);

#ifdef EBUG
    printf("pat_source.x,y,z = %f %f %f\n",
            pat_source.x, pat_source.y, pat_source.z);
#endif EBUG

    bm_target.z = 200.0;   /* should be a little smarter about this */
    scale = bm_target.z / request.unit.SAD;
```

mk_contour_map.c.v2

```
/*
  Set up a vector from the source to a point on a plane which is
  perpendicular to the c. axis and 200 cm from the source for each
  cell in the contour map.  (There is nothing magic about 200 cm and
  in fact this should probably be handled a little more
  intelligently.)  Recall - you *have* read defines.h - that the
  contour map is defined as a 5 mm grid at the beams definition
  distance layed out so that the central axis intersects the corner of
  four cells - there are an even number of cells across a rectangular
  beam.

  Having set up the ray in beam coordinates, transform to patient
  coordinates and intersect with the patient.   Store the SSD in the
  grid matrix - a value >1000.0 means no intersection.

  The map extends only to the geometric edge (+ epsilon) of the beam
  Beyond that one should do bidirectional nearest-neighbor
*/

    bm_source.x = bm_source.y = bm_source.z = 0.0;
    transform_point (bm_source, request.beam.T_beam_to_pat, &pat_source);

#ifdef EBUG
    printf("pat_source.x,y,z = %f %f %f\n",
            pat_source.x, pat_source.y, pat_source.z);
#endif EBUG

    bm_target.z = 200.0;   /* should be a little smarter about this */
    scale = bm_target.z / request.unit.SAD;
```

mk_contour_map.c.v0

```
/*
  Figure out how much of the map we need to compute for this beam.

  * def is the half width in cm.  Multiply by two to get the number of
  5mm cells, round up and truncate.

  The following assumes symmetric jaws.  Watch out.
*/
    x_size_in_cells = (int) (request.beam.jaw[X_JAW].pos_2 * 2.0 + 0.5);
    if (x_size_in_cells > (MAP_SIZE / 2))
    {
         fprintf (stderr, "ARRRGGH.  Field too big in x (%d cells)\n",
                  x_size_in_cells);
         exit (1);
    }
    y_size_in_cells = (int) (request.beam.jaw[Y_JAW].pos_2 * 2.0 + 0.5);
    if (y_size_in_cells > (MAP_SIZE / 2))
    {
         fprintf (stderr, "ARRRGGH.  Field too big in y (%d cells)\n",
                  y_size_in_cells);
         exit (1);
    }
    tables.con_map_min_x_index = (MAP_SIZE / 2) - x_size_in_cells;
    tables.con_map_max_x_index = (MAP_SIZE / 2) + x_size_in_cells - 1;
    tables.con_map_min_y_index = (MAP_SIZE / 2) - y_size_in_cells;
    tables.con_map_max_y_index = (MAP_SIZE / 2) + y_size_in_cells - 1;
```

mk_contour_map.c.v2

```
/*
  Figure out how much of the map we need to compute for this beam

  * def is the half width in cm.  Multiply by two to get the number of
  5mm cells, round up and truncate.

*/

    /* John R. Kollar 9311.11 */
    tables.con_map_min_x_index = (int)(MAP_SIZE/2.0 - 0.5
                                       + request.beam.jaw[X_JAW].pos_1 * 2.0);
    tables.con_map_max_x_index = (int)(MAP_SIZE/2.0 - 0.5
                                       + request.beam.jaw[X_JAW].pos_2 * 2.0);
    tables.con_map_min_y_index = (int)(MAP_SIZE/2.0 - 0.5
                                       - request.beam.jaw[Y_JAW].pos_2 * 2.0);
    tables.con_map_max_y_index = (int)(MAP_SIZE/2.0 - 0.5
                                       - request.beam.jaw[Y_JAW].pos_1 * 2.0);

#ifdef J_DEBUG1
    printf("request.beam.jaw[X_JAW].pos_1 = %f",request.beam.jaw[X_JAW].pos_1);
    printf("    tables.con_map_min_x_index = %d\n",tables.con_map_min_x_index);
    printf("request.beam.jaw[X_JAW].pos_2 = %f",request.beam.jaw[X_JAW].pos_2);
    printf("    tables.con_map_max_x_index = %d\n",tables.con_map_max_x_index);
    printf("request.beam.jaw[Y_JAW].pos_1 = %f",request.beam.jaw[Y_JAW].pos_1);
    printf("    tables.con_map_min_y_index = %d\n",tables.con_map_min_y_index);
    printf("request.beam.jaw[Y_JAW].pos_2 = %f",request.beam.jaw[Y_JAW].pos_2);
    printf("    tables.con_map_max_y_index = %d\n",tables.con_map_max_y_index);
#endif J_DEBUG1
```

mk_contour_map.c.v0

```
/*
  Step through the cells in raster order.  Note that we are filling in
  the center of the map, not the edge.  Central axis is always between
  elements MAP_SIZE/2 and MAP_SIZE/2 - 1.
*/
    y_inc = -0.5 * scale;
    x_inc = 0.5 * scale;
```

mk_contour_map.c.v2

```
    /* Some array bounds checking. */
    if ((tables.con_map_min_x_index < 0)
                            || (tables.con_map_min_x_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: X_JAW_1 = %f",
                                            request.beam.jaw[X_JAW].pos_1);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }
    if ((tables.con_map_max_x_index < 0)
                            || (tables.con_map_max_x_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: X_JAW_2 = %f",
                                            request.beam.jaw[X_JAW].pos_2);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }
    if ((tables.con_map_min_y_index < 0)
                            || (tables.con_map_min_y_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: Y_JAW_1 = %f",
                                            request.beam.jaw[Y_JAW].pos_1);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }
    if ((tables.con_map_max_y_index < 0)
                            || (tables.con_map_max_y_index >= MAP_SIZE)) {
        fprintf(stderr,"mk_contour_map: Y_JAW_2 = %f",
                                            request.beam.jaw[Y_JAW].pos_2);
        fprintf(stderr," is beyond the contour-map edge.\n");
        exit(1);
    }

/*
  Step through the cells in raster order.  Note that we are filling in
  the center of the map, not the edge.  Central axis is always between
  elements MAP_SIZE/2 and MAP_SIZE/2 - 1.
*/
    y_inc = -0.5 * scale;
    x_inc = 0.5 * scale;
```

mk_contour_map.c.v0

```
bm_target.y = ((y_size_in_cells * 0.5) - 0.25) * scale;

if (verbose)
     printf ("\n");
for (yloop = tables.con_map_min_y_index;
     yloop <= tables.con_map_max_y_index;
     yloop++)
{
     bm_target.x = ((x_size_in_cells * 0.5) - .25) * -scale;
     for (xloop = tables.con_map_min_x_index;
          xloop <= tables.con_map_max_x_index;
          xloop++)
     {
          transform_point(bm_target, request.beam.T_beam_to_pat,
                          &pat_target);

#ifdef EBUG
          printf("pat_target.x,y,z = %f %f %f\n",
                 pat_target.x, pat_target.y, pat_target.z);
#endif EBUG

/*
  This costs like the dickens.  Isn't there some more efficient way to
  do this?
*/
          prism(&request.patient_anastruct, pat_source, pat_target,
                &inside, &SSD,
                &thickness, &pathlength);
          tables.contour_map[yloop][xloop] = SSD;
```

mk_contour_map.c.v2

```
/* John R. Kollar R9408.04 */
bt.y = (MAP_SIZE/2 - tables.con_map_min_y_index - 0.5)*0.5*scale;
bt.x = (tables.con_map_min_x_index - MAP_SIZE/2 + 0.5)*0.5*scale;
bm_target.y = bt.y;

if (verbose)
     printf ("\n");
for (yloop = tables.con_map_min_y_index;
     yloop <= tables.con_map_max_y_index;
     yloop++)
{
     bm_target.x = bt.x;
     for (xloop = tables.con_map_min_x_index;
          xloop <= tables.con_map_max_x_index;
          xloop++)
     {
          transform_point(bm_target, request.beam.T_beam_to_pat,
                          &pat_target);

#ifdef EBUG
          printf("pat_target.x,y,z = %f %f %f\n",
                 pat_target.x, pat_target.y, pat_target.z);
#endif EBUG

/*
  This costs like the dickens.  Isn't there some more efficient way to
  do this?
*/
          prism(&request.patient_anastruct, pat_source, pat_target,
                &inside, &SSD,
                &thickness, &pathlength);
          tables.contour_map[yloop][xloop] = SSD;
```

mk_contour_map.c.v0

```
#ifdef EBUG
            if (thickness != 0.0)
                printf("SSD[%d][%d] = %f\n", yloop, xloop, SSD);
#else
            if (verbose)
            {
                printf(".");
                fflush (stdout);
            }
#endif EBUG

            bm_target.x += x_inc;
        }
#ifndef EBUG
        if (verbose)
            printf ("\n");
#endif
        bm_target.y += y_inc;
    }









    return;
}
```

mk_contour_map.c.v2

```
#ifdef EBUG
            if (thickness != 0.0)
                printf("SSD[%d][%d] = %f\n", yloop, xloop, SSD);
#else
            if (verbose)
            {
                printf(".");
                fflush (stdout);
            }
#endif EBUG

            bm_target.x += x_inc;
        }
#ifndef EBUG
        if (verbose)
            printf ("\n");
#endif
        bm_target.y += y_inc;
    }

#ifdef J_DEBUG2
    fpr = fopen("contour_map.txt","w");
    for (yloop = 0;yloop < MAP_SIZE;yloop++) {
        for (xloop = 0;xloop < MAP_SIZE-1;xloop++) {
            fprintf(fpr,"%f ",tables.contour_map[yloop][xloop]);
        }
        fprintf(fpr,"%f\n",tables.contour_map[yloop][xloop]);
    }
    fclose(fpr);
#endif J_DEBUG2

   return;
}
```

7.4. **Appendix D.** Code listing of mk_dSAR_table.c

This appendix contains a listing of the code for the GRATIS routine mk_dSAR_table, which constructs the table of dSAR's used by the scatter integration. Both the original version and the version with all the final modifications are listed side by side to facilitate comparisons. Also, the code changes that are relevant to the tilting of the dartboard are shown in bold to set them apart from changes that involve debugging statements, documentation and reformatting of C code.

mk_dSAR_table.c.v0

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_dSAR_table.c,v 3.3
        1992/05/28 18:29:03 sherouse Exp $";

/*
 * $Log: mk_dSAR_table.c,v $




 * Revision 3.3  1992/05/28  18:29:03  sherouse
 * - ifdef out the wild idea
 *
 * Revision 3.2  1990/10/14  16:24:58  sherouse
 * - the PSF has moved to the sar data file - see make_sar
 *
 * Revision 3.1  89/01/18  14:43:24  sherouse
 * - support new beam description
 * - add PSF table calculation
 *
 * Revision 2.2  88/05/31  17:22:51  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:09  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.2  87/05/22  11:09:28  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:18:29  sherouse
 * Initial revision
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"
```

mk_dSAR_table.c.v3

```
static char *RCSid =
        "$Header: /GRATIS/MASTER/src/photon/c/RCS/mk_dSAR_table.c,v 3.3
        1992/05/28 18:29:03 sherouse Exp $";

/*
 * $Log: mk_dSAR_table.c,v $
 *
 * Revision 3.4  1993/11/19  12:02:00  jkollar
 * - The assumption of symmetric jaws in calculating the diagonal has been
 * - removed.
 *
 * Revision 3.3  1992/05/28  18:29:03  sherouse
 * - ifdef out the wild idea
 *
 * Revision 3.2  1990/10/14  16:24:58  sherouse
 * - the PSF has moved to the sar data file - see make_sar
 *
 * Revision 3.1  89/01/18  14:43:24  sherouse
 * - support new beam description
 * - add PSF table calculation
 *
 * Revision 2.2  88/05/31  17:22:51  sherouse
 * - spruced up documentation (just a tad)
 *
 * Revision 2.1  88/05/31  09:43:09  sherouse
 * - miraculous transformation into a near-production version
 *
 * Revision 1.2  87/05/22  11:09:28  sherouse
 * - minor mods to accommodate new data structures
 *
 * Revision 1.1  87/05/18  15:18:29  sherouse
 * Initial revision
 *
 */

#include <stdio.h>
#include <math.h>

#include "gen.h"
#include "extbm.h"
```

mk_dSAR_table.c.v0

```
#include "defines.h"

#define SQRT(x) ((float) sqrt((double) (x)))
#define SIN(x) ((float) sin((double) (x)))
#define COS(x) ((float) cos((double) (x)))









mk_dSAR_table()
/*
 ------------------------------------------------------------------------

   NAME
        mk_dSAR_table - make a table of differential S*Rs

   SYNOPSIS


   DESCRIPTION
        This routine constructs a table of differential S*Rs.  It
        starts by choosing a set of sector radii based on the
        collimator setting and some heuristics (there is a wealth of
        godd projects in this).  It then builds the table for the
        centers of those sectors an the depths of the original S*R
        table.  Some other tables for quick reference are built.
        Finally, the dS*R table is modified to "remove" some flatness
        dependence.

   BUGS
        See comments regarding flatness removal below.
```

mk_dSAR_table.c.v3

```
#include "defines.h"

#define SQRT(x) ((float) sqrt((double) (x)))
#define SIN(x) ((float) sin((double) (x)))
#define COS(x) ((float) cos((double) (x)))

/* John R Kollar R9401.20 */
/* Since our beam flatness table does not roll off for large fields */
/* (ie. It is extrapolated beyond the largest measured field size) */
/* we believe that the WILD_IDEA is correct. */
#define WILD_IDEA

/* Print the dSAR table out to the file SARtest.txt */
#define J_DEBUG

/* #define OLD_dRADIUS */

mk_dSAR_table()
/*
 ------------------------------------------------------------------------

   NAME
        mk_dSAR_table - make a table of differential S*Rs

   SYNOPSIS


   DESCRIPTION
        This routine constructs a table of differential S*Rs.  It
        starts by choosing a set of sector radii based on the
        collimator setting and some heuristics (there is a wealth of
        godd projects in this).  It then builds the table for the
        centers of those sectors an the depths of the original S*R
        table.  Some other tables for quick reference are built.
        Finally, the dS*R table is modified to "remove" some flatness
        dependence.

   BUGS
        See comments regarding flatness removal below.
```

mk_dSAR_table.c.v0

```
  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        2 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

 ------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int         rloop,
                dloop,
                ang_loop,
                index,
                loop;

    float       diagonal,
                radius,
                SAR_so_far,
                SAR,
                fx,
                angle_inc,
                angle,
                sine,
                cosine,
                previous_r,
                sec_center,
                last_one,
                x,
                xn,
                inc,
                depth_scale,
                flatness[MAX_FLATNESS_RADII];
```

mk_dSAR_table.c.v3

```
  AUTHOR
        George W. Sherouse
        Radiation Oncology
        North Carolina Memorial Hospital
        University of North Carolina
        2 January 1986

        (c) Copyright 1986, 1987, 1988 - George W. Sherouse
        All rights reserved.

 ------------------------------------------------------------------------
*/

{
    extern REQUEST request;
    extern TABLES tables;

    int         rloop,
                dloop,
                ang_loop,
                index,
                loop;

    float       diagonal,
                radius,
                SAR_so_far,
                SAR,
                fx,
                angle_inc,
                angle,
                sine,
                cosine,
                previous_r,
                sec_center,
                last_one,
                x,
                xn,
                inc,
                depth_scale,
                flatness[MAX_FLATNESS_RADII];
```

mk_dSAR_table.c.v0

```
    float       v_interp();




/*
  Set up the radius vector for the dSAR table.  These radii are the
  *outer* bounds of the scatter sectors.  Sometime it would be nice
  to devise a heuristic for optimizing these sector sizes but for now
  I'll just use equal partitioning.

  The following assumes symmetric jaws.  Watch out.
*/
    diagonal = 2.5 * SQRT(request.beam.jaw[X_JAW].pos_2 *
                          request.beam.jaw[X_JAW].pos_2 +
                          request.beam.jaw[Y_JAW].pos_2 *
                          request.beam.jaw[Y_JAW].pos_2);

#ifdef OLD_dRADIUS

/* Here's the old way */
    tables.dSAR_radius_count = dSAR_MAX_RADII;
    radius_inc = diagonal / (tables.dSAR_radius_count - 1);
    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        tables.dSAR_radii[rloop] = (radius += radius_inc);
```

mk_dSAR_table.c.v3

```
#ifdef OLD_dRADIUS
    float radius_inc;
#endif OLD_dRADIUS

    float       v_interp();

    float       W,L; /* John R Kollar M9311.15 */

#ifdef J_DEBUG
    FILE *fpr;
#endif J_DEBUG

/*
  Set up the radius vector for the dSAR table.  These radii are the
  *outer* bounds of the scatter sectors.  Sometime it would be nice
  to devise a heuristic for optimizing these sector sizes but for now
  I'll just use equal partitioning.
*/


    /* John R Kollar M9311.15 */
    W = request.beam.jaw[X_JAW].pos_2 - request.beam.jaw[X_JAW].pos_1;
    L = request.beam.jaw[Y_JAW].pos_2 - request.beam.jaw[Y_JAW].pos_1;
    diagonal = 1.25*SQRT(W*W + L*L);

#ifdef OLD_dRADIUS

/* Here's the old way */
    tables.dSAR_radius_count = dSAR_MAX_RADII;
    radius_inc = diagonal / (tables.dSAR_radius_count - 1);
    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        tables.dSAR_radii[rloop] = (radius += radius_inc);
```

mk_dSAR_table.c.v0

```
#else
/*
  And here's an experimental heuristic.  We'll use a recursive
  geometric series.  We can calculate the basic increment and
  recurse from there.  If the basic increment is too small, we
  can make the table smaller and take the time saving to the bank.

                    n
  r  = r    + inc * x
   n    n-1

  r0 = 0.0
*/
    last_one = x = xn = 1.14;
    for (loop = 0; loop < tables.dSAR_radius_count - 2; loop++)
         last_one = (last_one + 1.0) * x;
    inc = diagonal / last_one;

    printf("increment, last_one = %f, %f\n", inc, last_one);

    if (inc < diagonal / 100.0) inc = diagonal / 100.0;

    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
    {
         tables.dSAR_radii[rloop] = tables.dSAR_radii[rloop - 1] + inc * xn;
         if (tables.dSAR_radii[rloop] > diagonal)
             break;
         xn *= x;
    }
    tables.dSAR_radius_count = (rloop >= tables.dSAR_radius_count) ?
         tables.dSAR_radius_count : rloop + 1;

    printf("modified increment = %f, radius count = %d\n",
            inc, tables.dSAR_radius_count);

    for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         printf (" %f", tables.dSAR_radii[rloop]);
    printf("\n");

#endif
```

mk_dSAR_table.c.v3

```
#else
/*
  And here's an experimental heuristic.  We'll use a recursive
  geometric series.  We can calculate the basic increment and
  recurse from there.  If the basic increment is too small, we
  can make the table smaller and take the time saving to the bank.

                    n
  r  = r    + inc * x
   n    n-1

  r0 = 0.0
*/
    last_one = x = xn = 1.14;
    for (loop = 0; loop < tables.dSAR_radius_count - 2; loop++)
         last_one = (last_one + 1.0) * x;
    inc = diagonal / last_one;

    printf("increment, last_one = %f, %f\n", inc, last_one);

    if (inc < diagonal / 100.0) inc = diagonal / 100.0;

    tables.dSAR_radii[0] = radius = 0.0;
    for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
    {
         tables.dSAR_radii[rloop] = tables.dSAR_radii[rloop - 1] + inc * xn;
         if (tables.dSAR_radii[rloop] > diagonal)
             break;
         xn *= x;
    }
    tables.dSAR_radius_count = (rloop >= tables.dSAR_radius_count) ?
         tables.dSAR_radius_count : rloop + 1;

    printf("modified increment = %f, radius count = %d\n",
            inc, tables.dSAR_radius_count);

#endif
```

mk_dSAR_table.c.v0

```
tables.dSAR_depth_count = request.SAR_table.depth_count;



for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
{
     tables.dSAR[0][dloop] = 0.0;
     SAR_so_far = request.SAR_table.SAR[dloop][0];



     for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
     {
         radius = tables.dSAR_radii[rloop];
         SAR = v_interp(0, request.SAR_table.radius_count,
                        request.SAR_table.radius_vec, radius,
                        request.SAR_table.SAR[dloop], &index, &fx);
         tables.dSAR[rloop][dloop] = SAR - SAR_so_far;
         SAR_so_far = SAR;



     }



}
```

mk_dSAR_table.c.v3

```
    printf("\n");
    for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
        printf ("radius[%d] = %f\n",rloop, tables.dSAR_radii[rloop]);
    printf("\n");

    tables.dSAR_depth_count = request.SAR_table.depth_count;

#ifdef J_DEBUG
    fpr = fopen("SARtest.txt","w");
#endif J_DEBUG

    for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
    {
        tables.dSAR[0][dloop] = 0.0;
        SAR_so_far = request.SAR_table.SAR[dloop][0];

#ifdef J_DEBUG
        fprintf(fpr,"[%f]  ",request.SAR_table.depth_vec[dloop]);
#endif J_DEBUG

        for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        {
            radius = tables.dSAR_radii[rloop];
            SAR = v_interp(0, request.SAR_table.radius_count,
                           request.SAR_table.radius_vec, radius,
                           request.SAR_table.SAR[dloop], &index, &fx);
            tables.dSAR[rloop][dloop] = SAR - SAR_so_far;
            SAR_so_far = SAR;

#ifdef J_DEBUG
           fprintf(fpr,"%f  ",tables.dSAR[rloop][dloop]);
#endif J_DEBUG

        }

#ifdef J_DEBUG
       fprintf(fpr,"\n");
#endif J_DEBUG

    }
```

mk_dSAR_table.c.v0

```
/*
  Calculate x, y offsets for centers of scatter sectors
*/
    angle_inc = 2.0 * PI / tables.dSAR_sector_count;
    angle = 0.0;
    sine = 0.0;
    cosine = 1.0;
    for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
    {
         previous_r = 0.0;
         for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         {
             sec_center = (tables.dSAR_radii[rloop] + previous_r) / 2.0;
             tables.sec_center_x[ang_loop][rloop] = sec_center * cosine;
             tables.sec_center_y[ang_loop][rloop] = sec_center * sine;
             previous_r = tables.dSAR_radii[rloop];
         }
         angle += angle_inc;
         sine = SIN(angle);
         cosine = COS(angle);
    }

#ifdef WILD_IDEA
/*
  Here's a wild idea...  Divide the differential SARs by the relative
  fluence at the sector centers.  The reasoning is something like this.
  The measured SAR table is for the central axis and includes the effects
  of beam non-flatness.  That's fine if we are caculating on the central
  axis, but off axis this causes the sector weightings to get screwed.
  If we divide out beam non-flatness now and multiply it back in later
  for the proper geometry we may get marginally better control over the
  shape of the beam profile.
*/
```

mk_dSAR_table.c.v3

```
#ifdef J_DEBUG
    fclose(fpr);
#endif J_DEBUG

/*
  Calculate x, y offsets for centers of scatter sectors
*/
    angle_inc = 2.0 * PI / tables.dSAR_sector_count;
    angle = 0.0;
    sine = 0.0;
    cosine = 1.0;
    for (ang_loop = 0; ang_loop < tables.dSAR_sector_count; ang_loop++)
    {
         previous_r = 0.0;
         for (rloop = 0; rloop < tables.dSAR_radius_count; rloop++)
         {
             sec_center = (tables.dSAR_radii[rloop] + previous_r) / 2.0;
             tables.sec_center_x[ang_loop][rloop] = sec_center * cosine;
             tables.sec_center_y[ang_loop][rloop] = sec_center * sine;
             previous_r = tables.dSAR_radii[rloop];
         }
         angle += angle_inc;
         sine = SIN(angle);
         cosine = COS(angle);
    }

#ifdef WILD_IDEA
/*
  Here's a wild idea...  Divide the differential SARs by the relative
  fluence at the sector centers.  The reasoning is something like this.
  The measured SAR table is for the central axis and includes the effects
  of beam non-flatness.  That's fine if we are caculating on the central
  axis, but off axis this causes the sector weightings to get screwed
  If we divide out beam non-flatness now and multiply it back in later
  for the proper geometry we may get marginally better control over the
  shape of the beam profile.
*/
```

mk_dSAR_table.c.v0

```
/*
  The problem with this is that if the profile rolls off dramatically
  at the edge, the dSARs for large radii get multiplied by large
  numbers and so become too heavily weighted in the scatter
  integration.  This is very bad, particularly for large fields.
  Back to the drawing board.  28 May 1992 GWS
*/

    for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
    {
        /*
          Need to scale radius from depth back to def_dist before
          doing flatness lookup.  The following is not quite right.  I am
          tacitly assuming here that the SARs were measured with
          the phantom surface at def_dist.  This is (almost) the case
          if the S*R data were extracted from PDD measurements but is
          badly wrong if the T*Rs were measured directly with each
          depth at def_dist.  The S*R table really should include a
          measurement distance for each depth - or better yet the S*R
          table should already have flatness taken out of it.
          */
        depth_scale = request.unit.SAD /
            (request.unit.SAD + request.SAR_table.depth_vec[dloop]);

        if (request.SAR_table.flatness_depth_count == 1)
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] = request.SAR_table.flatness[loop][0];
        else
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] =
                     v_interp (0, request.SAR_table.flatness_depth_count,
                                  request.SAR_table.flatness_depth_vec,
                                  request.SAR_table.depth_vec[dloop],
                                  request.SAR_table.flatness[loop], &index,&fx);
```

mk_dSAR_table.c.v3

```
/*
  The problem with this is that if the profile rolls off dramatically
  at the edge, the dSARs for large radii get multiplied by large
  numbers and so become too heavily weighted in the scatter
  integration.  This is very bad, particularly for large fields.
  Back to the drawing board.  28 May 1992 GWS
*/

    for (dloop = 0; dloop < tables.dSAR_depth_count; dloop++)
    {
        /*
          Need to scale radius from depth back to def_dist before
          doing flatness lookup.  The following is not quite right.  I am
          tacitly assuming here that the SARs were measured with
          the phantom surface at def_dist.  This is (almost) the case
          if the S*R data were extracted from PDD measurements but is
          badly wrong if the T*Rs were measured directly with each
          depth at def_dist.  The S*R table really should include a
          measurement distance for each depth - or better yet the S*R
          table should already have flatness taken out of it.
        */
        depth_scale = request.unit.SAD /
            (request.unit.SAD + request.SAR_table.depth_vec[dloop]);

        if (request.SAR_table.flatness_depth_count == 1)
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] = request.SAR_table.flatness[loop][0];
        else
            for (loop = 0;
                  loop < request.SAR_table.flatness_radius_count;
                  loop++)
                 flatness[loop] =
                     v_interp (0, request.SAR_table.flatness_depth_count,
                                  request.SAR_table.flatness_depth_vec,
                                  request.SAR_table.depth_vec[dloop],
                                  request.SAR_table.flatness[loop],&index,&fx);
```

mk_dSAR_table.c.v0

```
        for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        {
            radius = (tables.dSAR_radii[rloop] +
                      tables.dSAR_radii[rloop - 1]) * 0.5;
            radius *= depth_scale;
            tables.dSAR[rloop][dloop] /=
                 v_interp(0, request.SAR_table.flatness_radius_count,
                          request.SAR_table.flatness_radius_vec, radius,
                          flatness, &index, &fx);
        }
    }
#endif WILD_IDEA
}
```

mk_dSAR_table.c.v3

```
        for (rloop = 1; rloop < tables.dSAR_radius_count; rloop++)
        {
            radius = (tables.dSAR_radii[rloop] +
                      tables.dSAR_radii[rloop - 1]) * 0.5;
            radius *= depth_scale;
            tables.dSAR[rloop][dloop] /=
                 v_interp(0, request.SAR_table.flatness_radius_count,
                          request.SAR_table.flatness_radius_vec, radius,
                          flatness, &index, &fx);
        }
    }
#endif WILD_IDEA
}
```