THEORETICAL STUDY OF AN ULTRASONIC SPARSE RANDOM PHASED ARRAY SURGICAL APPLICATOR

BY

JEFFREY MIN-JER YANG

B.S., University of Illinois at Urbana-Champaign, 1995

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1996

Urbana, Illinois

ACKNOWLEDGMENTS

The author expresses much gratitude to his advisor, Professor Leon Frizzell, for his guidance, support, and especially his patience throughout the course of this research. The author would also like to extend thanks to Dr. Stephen A. Goss and Mr. Jeff Kouzmanoff of Labthermics Technologies, Inc. for providing helpful suggestions and insights to this project.

The author wants to express his deepest appreciation to his friend, Mr. Joseph Mark Barich, who convinced the author to continue his graduate school education. Joe is both the best friend and best advisor anyone can ever hope for.

A million thanks go to the author's parents, Steve and Christina, and sister, Judy, for always believing. Without their support the author would never have accomplished nearly as much. Finally, yet most importantly, the author wishes to thank his wife, Sally, for standing by him through all the hard times. The author owes the greatest gratitude to her.

TABLE OF CONTENTS

CHAPTER PAGE

CHAPTER 1

INTRODUCTION

The field of medical ultrasound encompasses a wide spectrum of clinical applications that range from diagnosis to surgery. In diagnostic applications, the intent is not to affect the tissues, whereas in therapy, the tissue may be heated or destroyed. The amount of energy delivered to the tissue and the bioeffect induced are determined by the application. At the high end of the energy scale is focused ultrasound surgery, where destruction or necrosis of the target tissue is desired. The specific goal of this study is to examine theoretically the feasibility of employing sparse random element array configurations for focal surgery.

Focused ultrasound surgery falls into the category of minimally invasive surgery. Potential applications of this technique have been explored in a number of clinical fields, including ophthalmology [1], urology [2], and oncology [3], [4]. Such surgical applications require precise control of the size and location of the focal therapy beam, as well as consideration of the effects of the entrance and exit beams associated with the use of focal fields. Early equipment was heavy and cumbersome, and imaging techniques were inadequate to ensure accurate placement of lesions at the target site in a living patient. Improvements in instrumentation have meant that these problems have been largely overcome, and there is a resurgence of interest in the clinical applications of this technique.

Ultrasonic focal surgery has clear advantages over conventional forms of surgery. It can be used to target tissues underneath the skin surface without the

complications of conventional surgery. By focusing high power ultrasound beams at a distance from the source, total necrosis of tissues lying within the focal volume may be achieved without damage to any structures lying elsewhere in the path of the beam. Nevertheless, there are some limitations in using ultrasound as a surgical tool. For instance, because the sound must be focused, a large acoustic window at the skin surface must be available to ablate tissue volumes at depth. Also, because of the propagation properties of sound, it is essential that neither bone nor gas lies in the path of the beam. It is important for the safe and effective use of focused ultrasound surgery that the intensity at the skin surface be kept low enough that it will not cause a burn [5].

Surgical ultrasound beams have historically been produced by single element transducers focused geometrically [6]. The acoustic beams produced by such devices are fixed in shape, and in position, with respect to the face of the source transducer. Changing the physical properties of the source transducer such as radius of curvature or diameter is unavoidable if one has to modify the focal size of the beam. The scanning of the focus in these cases can be accomplished only by using a precisely controlled, rapidly translatable mechanical system.

Phased arrays offer an advantage over the single element transducer in that electronic steering of a focal ultrasound beam is possible, which means that precise target positions can be obtained without physically moving the transducer. In addition, the nature of phased arrays provides a means for modifying the characteristics of the ultrasonic surgical beam [7]. This electronic beam synthesis can be applied to steer the beam to specific target locations, as well as to vary the focal volume produced by the array, thus catering to specific clinical needs [8]. It has been shown that an array inter-

element spacing of a half wavelength can be employed to avoid the production of grating lobes which can degrade system performance [9]. At megahertz frequencies, this spacing requirement limits the array to small apertures for reasonable numbers of array elements. However, a large aperture is generally necessary to develop sufficient intensity gain for tissue ablation. This requires adding more array elements whose individual size is inversely proportional to the driving frequency. The need to provide separate RF drive and control for each additional element further complicates the construction procedure for surgical ultrasound phased arrays.

The solution proposed in this thesis is to use a sparsely filled spherical ultrasonic phased array to reduce the number of array elements and RF drive channels required for practical implementation. Sparse, random arrays have long been used in the design of communication antenna array systems [10]. For this particular application, the design concept involves a spherical acoustic aperture, with the focus of the aperture geometrically centered on the tumor when all elements of the array are excited at a common phase. The position of the focal region is controlled by modifying the phase of the signals applied to individual elements to ablate the entire tumor volume using a succession of exposures at various sites within the tumor. Even though the element spacing is greater than half the wavelength in a sparse array, the use of spherical geometry and random location of elements will minimize grating lobes.

Goss et al. [11] have conducted studies of the spherical sparse array. They constructed a spherical segment array with 108 elements arranged in a hexagonal pattern as an experimental protocol. However, only 64 elements, randomly chosen, were excited at one time. The spherical shell had a diameter and a radius of curvature

that were both 10 cm. For the theoretical analyses of the hexagonal array, and for all simulations in this thesis, each individual transducer was driven with a surface velocity of 0.1 m/s , which corresponds to an output intensity of 0.77 W/cm^2. Both the simulation and experimental results showed that six grating lobes with significant amplitude with respect to the focus were produced. Electronic steering of focus will cause the focal intensity to decrease while the trailing grating lobes increased to unacceptably large amplitudes.

This thesis provides an in-depth study of ways to improve the performance of the hexagonal phased array system proposed by Goss et al. [11]. The details of the computational model used to generate the theoretical results are outlined in Chapter 2. Chapter 3, 4, and 5 contain details of the various attempts to improve system performance. In Chapter 3, randomization processes are investigated in an effort to minimize the grating lobe level. Chapter 4 investigates the relationship between the input and output parameters of this Multi-Input-Multi-Output (MIMO) array system to produce a mathematical model, which can be utilized for system performance evaluation. Chapter 5 examines the possibility of optimizing the array by modifying the transducer characteristics. The goal of this work is to design a phased array system with desired focal intensity and steering width.

CHAPTER 2

COMPUTATIONAL MODEL DESCRIPTION AND OPTIMIZATION

This chapter will begin with a short discussion of the theoretical basis of the random sparse array. Details of the derivation, upon which the computational model is based, have been provided by Barich [12]. The only section of derivation that will be included in this thesis contains a correction to the original work. Then, a brief outline of the simulation will be provided to explain the operation of the Random Sparse Array Simulator (RSAS) used to compute the acoustic field produced by the array. The algorithms that were improved to optimize the performance of the simulator will also be presented in this chapter.

2.1 Theoretical Basis of the Computational Model

The Random Sparse Array (RSA) is one form of an aperiodic array. Since a regular pattern of spacing does not exist, the large grating lobes associated with a periodic array are not present. The spacing between the elements, and the elements themselves, are larger than half a wavelength. This reduces the number of elements that are placed on the transducer aperture. As a result, the RSA can be built more economically than the conventional fully sampled phased array.

The field produced by a random array in general can be determined as discussed below. The behavior of the random array can be investigated using the mathematics of random processes. Since the array can be described statistically, one can derive its properties in a probabilistic manner. This can then lead to specifying

the statistical averages of the important properties of the radiation patterns such as average grating lobe level. However, the statistical study of the random array is beyond the scope of this thesis and will not be discussed here.

The Random Sparse Array in this study consists of circular piston sources (elements) mounted on a spherical shell. Thus, with no phasing of signals applied to the sources, the focus is located at the geometric focus, but the focal region can be electronically steered to ablate an entire treatment volume by varying the phases of signals applied to each individual element. The directivity pattern of the elements will determine the steering range of the array. The relation between the two is given in a later chapter. The pressure field of the entire array is calculated by superposition of individual piston source field patterns, which are calculated using the Point Radiator Method. The theoretical development for the pressure field of a piston source was provided by Barich [12]. However, the equations used by Barich to obtain the field pattern of the individual element contained an error. The pertinent portion of the derivation with the correction is shown below. The derivation is based on the schematic in Figure 2.1.

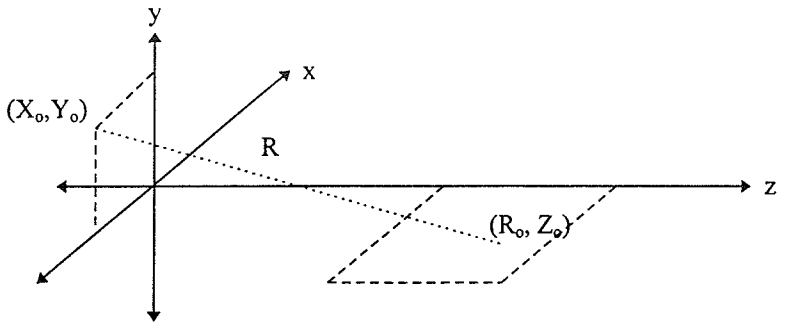

Figure 2.1 Schematic of the Point Radiator Method.

The pressure P_o produced at the point (R_o, Z_o) by a point source positioned at (X_o, Y_o) is

$$P_o = j\frac{\rho_o ckU_o \Delta A}{2\pi} \sum_{surface} \frac{\exp -(\alpha + jk)R}{R} \qquad (2.1.1)$$

where ρ_o is the density of the propagating medium, c is the speed of sound in the medium, U_o is the surface velocity, ΔA is the incremental area of the source, k is the wave number, R is the distance between the points (X_o, Y_o) and (Ro, Z_o), and α is the attenuation constant of the propagating medium. In this part of the code, in which the pressure field of a single piston source is calculated, the attenuation constant α is set to zero. The effect of attenuation is included when calculating the field of the entire array. With $\alpha = 0$, and $k = \frac{2\pi f}{c}$, the pressure equation becomes

$$P_o = j\rho_o f U_o \Delta A \sum_{surface} \frac{\exp -(jkR)}{R} \qquad (2.1.2)$$

From Euler's formula where $\exp(j\theta) = \cos\theta + j\sin\theta$,

$$P_o = \rho_o f U_o \Delta A \sum_{surface} \frac{1}{R}\left[\sin(kR) + j\cos(kR)\right] \qquad (2.1.3)$$

The above equation from Barich has been corrected where a j was missing from Equation (2.1.2). However, even though the Barich result was off in phase by 90 degrees, this did not affect the results of the simulation because the relative phases between the elements was the same.

2.2 Computational Model Overview

The flowchart in Figure 2.2 summarizes the main operations of the RSAS, which is used to calculate the acoustic field of the random sparse array. The actual simulation code is listed in Appendix A.

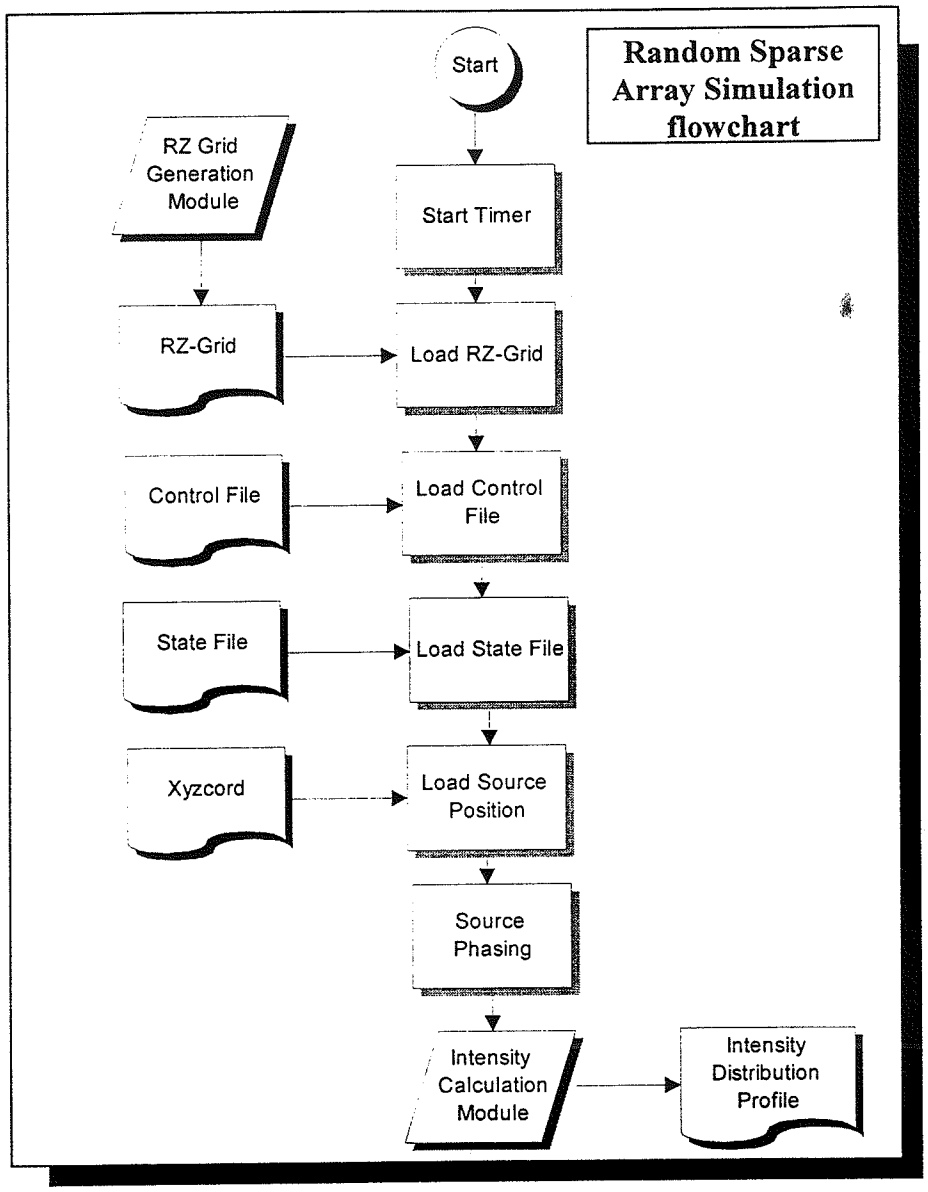

Figure 2.2 General flowchart of the Random Sparse Array Simulator.

The pressure field for a single piston source is calculated using the *RZ-Grid Generator Module* and is stored in a look-up table called the *RZ-Grid* as magnitude

and phase of the pressure versus r and z. This table is an input to the RSAS algorithm. Next, the RSAS reads in the control file, which contains information such as the focal coordinates, output file name, name of the state file, and name of the coordinate file. The coordinate file contains the location of the source elements while the state file indicates those elements that are turned on during the simulation. The simulator then uses the data from the input files to calculate the phase of each of the elements required to generate a focus at the desired location in the *Source Phasing* subroutine. The phasing information is then passed onto the subsequent subroutine called the *Intensity Calculation Module*. This is the heart of the simulation for it calculates the pressure at various points in a pre-specified area or volume through superposition of the appropriate pressure values from the *RZ-Grid* with proper phase adjustment. The output file from this module contains the intensity distribution profile, which can be plotted or analyzed through data manipulation programs such as Matlab. The detailed operation of the two main modules, the *RZ-grid Generation Module* and *Intensity Calculation Module*, is described extensively by Barich [12] and will not be discussed here.

2.3 Improvement to Previously Developed Modules

To improve the performance of the simulator so that the result gives a correct representation of the actual array system and so that the process of analyzing the data can be accomplished with ease, several modifications were made to the simulation written by Barich and are discussed in this section.

2.3.1 Output file format of RSAS.

To better visualize the pressure field produced by the random sparse array, the output scheme of the RSAS was revised. In the original simulator, the code had the ability to output only one-dimensional (1D) plots. Specifically, the RSAS could output only files that contained information pertaining to only one of the axes (see Figure 2.3(a)). This was not the most ideal format because researchers were not able to identify all of the grating lobes and their positions. Thus, an algorithm was developed to allow the simulator to output the intensity profile in matrix form which could then be plotted using Matlab. An example of the resulting plot, showing the intensity profile in two dimensions (2D), is shown in Figure 2.3(b)). The grating lobes that were unobservable in the 1D output format can be seen with clarity in the 2D plot.

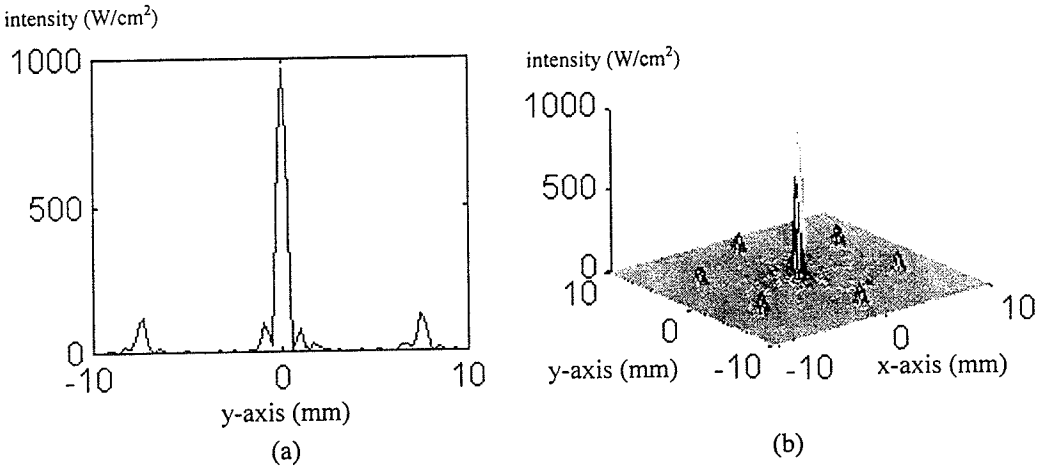

Figure 2.3 The 1D and 2D plots of the intensity profile at the focal plane of the RSA is show in (a) and (b), respectively.

2.3.2 Phase quantization

In the original code written by Barich [12], the phase of the signal applied to each element was assumed to be exact. However, this is not true for the actual hardware employed in the experimental system. The transducer driver built by Labthermics Technologies utilizes a 16-level phase quantization scheme in which the phase is stepped at an interval of 22.5 °. This scheme was incorporated into the RSAS code to better simulate the experimental conditions. The incorporation of the quantization did not affect the result noticeably. Figures 2.4(a) and 2.4(b) show the intensity profile through the focus along the y direction for the hexagonal array prior to and after the implementation of the phase quantization scheme, respectively. The intensity and position of the grating lobes are the same for each plot, and no other obvious differences were observed. Thus, the effect of quantization appears to be negligible.

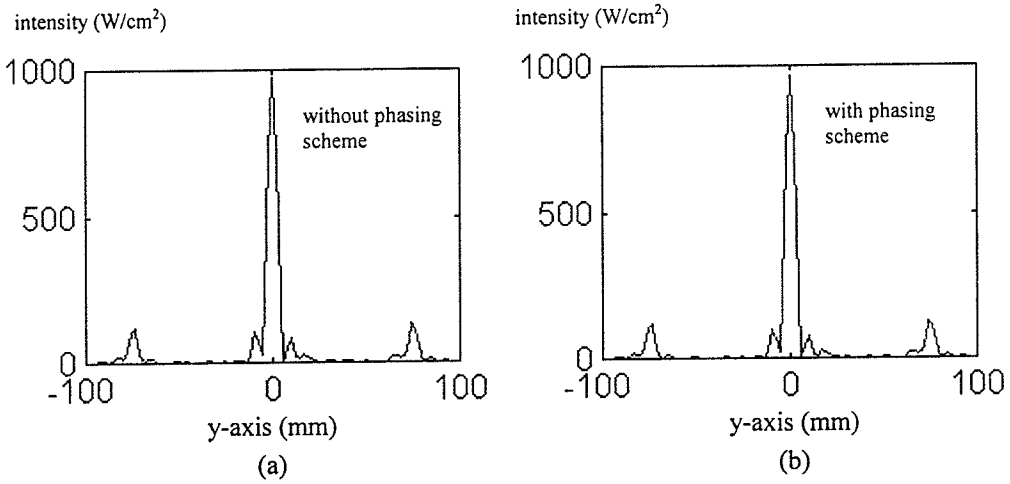

Figure 2.4 The 1D intensity profile plots for the hexagonal array. The profile before and after quantization is show in (a) and (b), respectively.

2.3.3 Optimizing simulation time

In the original simulation developed by Barich [12], many output modules were installed to monitor the operation of the RSAS and to ensure the proper operation of the code. However, these diagnostic functions are the most time-consuming processes in the simulation. The code is at a stage now that the basic functions have been checked several times for faults. Thus, many of these modules have been eliminated to shorten the simulation time. In addition, since the output of the *RZ-Grid Generator module* is in ASCII form, a converter program was written to transform the output file format from ASCII to binary (see Appendix B). This alteration speeded up the read-in time from six minutes to 1.5 minutes. To better monitor simulation time, a time module was installed in the RSAS to indicate the total elapsed time of the program. Currently, the total simulation time for the RSAS is about 10 minutes for a 100 by 100 grid output.

The most CPU intense module currently is the *RZ-Grid Generator Module*. To produce a 600 by 1000 reference grid for a circular piston source can take from 2 days to a week depending on the frequency and the radius of the element. Since we are only interested in the intensity profile of the array near its geometric focus, this module has been modified to reduce the region for field calculation resulting in only a 150 by 100 reference grid. This change limits the longest running case to approximately one day. An expanded grid can be generated if it is desired to examine the field of the array over a larger area.

2.3.4 Data extrapolation scheme

In the original RSAS written by Barich [12], the contribution to the pressure at a point (r_p,z_p) from a particular element was the average of the pressures at the four closest points in the *RZ-Grid* surrounding the point (r_p,z_p). However, this means that any point falling within the region bordered by these points had the same value. To better estimate the value of the pressure at a given point, a first-order data extrapolation scheme has been devised. A graphical representation is shown in Figure 2.5 to facilitate the understanding of this scheme.

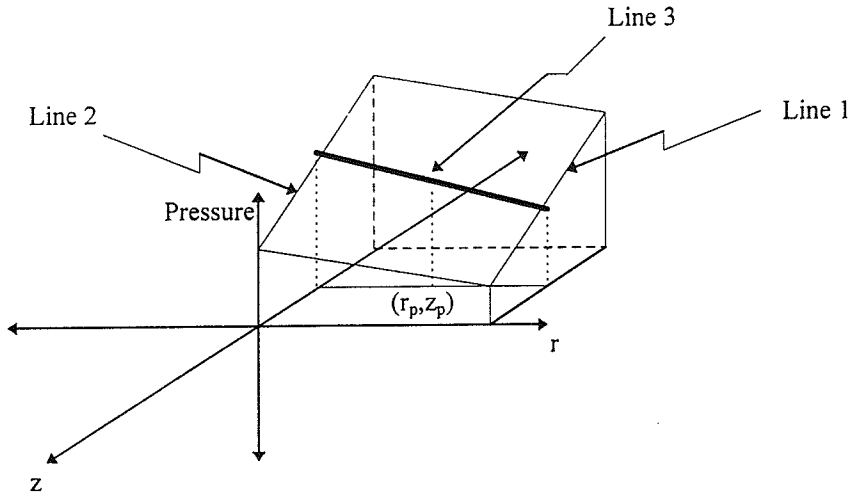

Figure 2.5 Conceptual drawing of the data extrapolation scheme.

Since the reference values of pressure are given at the four corners of the region as determined from the known coordinate positions in the *R-Z Grid*, one can find the equation of lines labeled 1 and 2 in Figure 2.5. Then, by using the point on each line which has the same z coordinates as the point where the field is desired, the equation for line 3 can be obtained. This approximation to the pressure at (r_p,z_p)

is given by the equation for line 3. A Matlab function for this extrapolation scheme is included in Appendix D.1.

The result of this implementation is shown in Figure 2.6. Figure 2.6(a) shows the envelope of steering with simple averaging as implemented by Barich [12]. Here, the envelope of steering is defined as the change in intensity of the focus as it is steered in the y direction from the geometrical focus. The effect of the new extrapolation scheme is demonstrated in Figure 2.6(b). Note that the envelope in Figure 2.6(b) is smooth as expected instead of having a sharp transition in the center as seen in Figure 2.6(a).

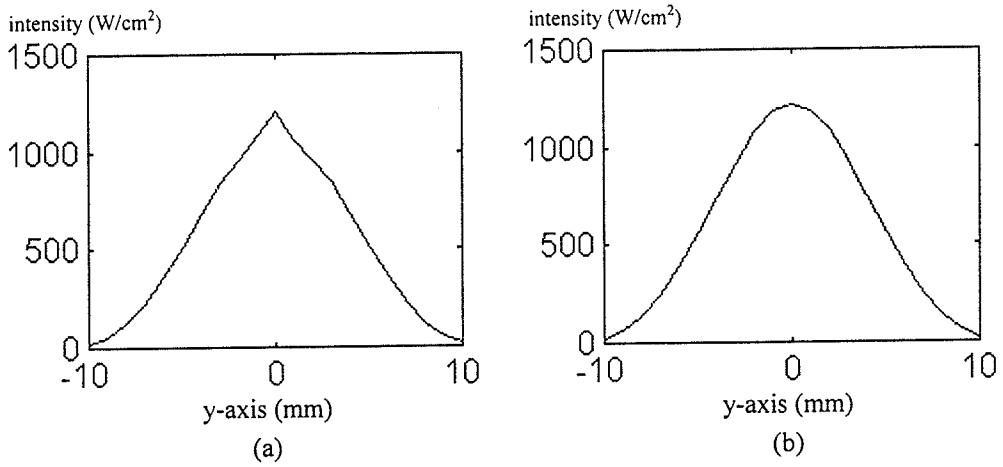

Figure 2.6 Plots of the steering envelope for the RSA. Plot (a) shows the envelope before extrapolation scheme was implemented, while plot (b) shows the result after the implementation.

CHAPTER 3

EFFECT OF ELEMENT POSITION RANDOMIZATION

The result obtained for the hexagonal array studied by Goss et al. [11] indicates that by merely turning regularly spaced elements on and off in a random fashion still results in significant grating lobes. When the focus was steered, these grating lobes could have intensities higher than the focus. This is unacceptable clinically because unwanted tissue damage will occur due to the abnormally high grating lobe level. This chapter examines other approaches to reduce the grating lobe level of the sparse array.

3.1 Altering Position Utilizing a Damping Cap

The intensity profile of the hexagonal array in Figure 2.3(b) shows that a regular pattern still exists even though elements were randomly activated. To further increase the randomness of the system, one can apply absorbing damping caps with circular openings smaller than the diameter of the element on the face of the transducer element, where the positions of the circular openings in the caps are placed in a random fashion. The consequence of adding the damping caps is that the radii of the elements are reduced and their positions modified. To obtain new coordinates of element positions, a Matlab subroutine was written (see Appendix D.2), which could randomize the position of the cap opening through either R or θ (see Figure 3.1).

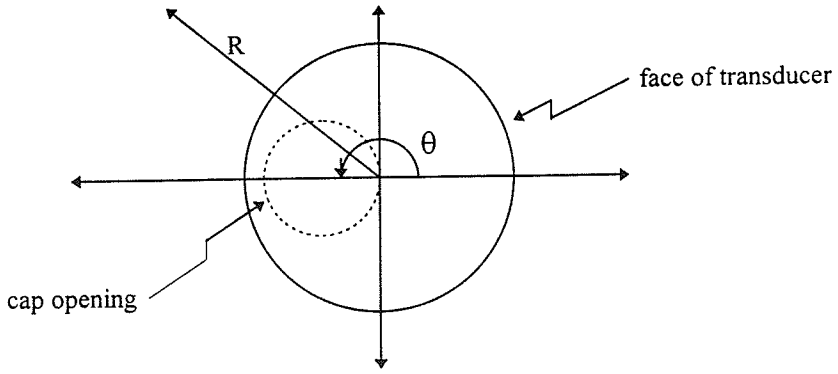

Figure 3.1 Conceptual drawing of the cap with opening on the face of a transducer element.

The center coordinates of the opening then were used in the RSAS as the coordinates of the source elements with the reduced radius. The effect of this modification is shown in Figure 3.2. The output intensity of the array was reduced greatly due to its fourth-order dependence on the radius, as will be shown in Section 4.5. When R is randomized (see Figure 3.2(a)), the opening in the cap can still be very close to the center of the element. Thus, the regular hexagonal pattern still largely persists and the six grating lobes are still present. A better result was obtained when θ was randomized while R was maintained at half the element radius (see Figure 3.2(b)). The next highest peak was 2 dB down instead of 1 dB down from the peak at the focus as in the case where only R was randomized. Nevertheless, the six grating lobes are still prominent in this case. The result obtained from the simulation was a clear indication that another avenue had to be found to minimize the grating lobes if this array system was to be used clinically.

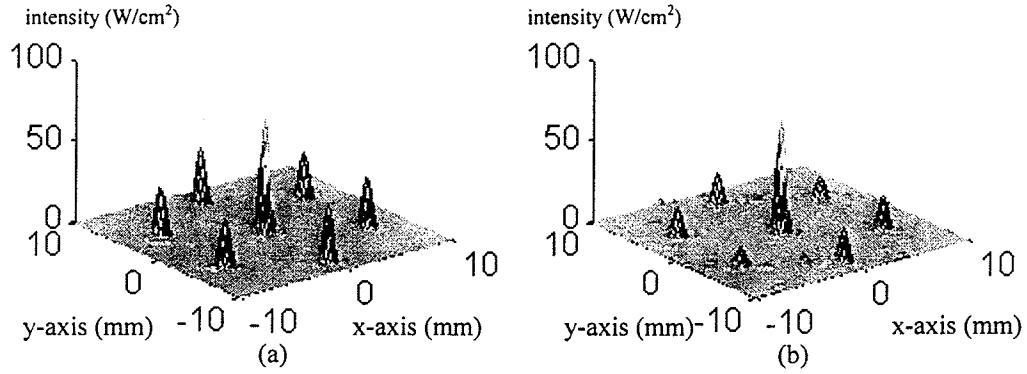

Figure 3.2 The intensity profile of the array with damping cap placed on the face of the transducer element. Plot (a) shows the profile with random radii, and plot (b) shows the profile with the constant radii.

3.2 Random Element Position

To eliminate the grating lobes associated with the hexagonal array, it is imperative to re-examine the design of the array. The only solution that will eliminate the six grating lobes is to go to a "true" random array. To accomplish this, the actual position of these elements was determined pseudo-randomly by the computer. Since 108 elements in the hexagonal array were tightly packed onto the spherical shell, the number of elements had to be reduced to allow random placement of elements. Sixty-four elements were chosen for compatibility with the number of channels in the experimental drive system. In addition, a constraint was made in the computer algorithm to prevent overlapping of elements. The result of this modification is shown in Figure 3.3(a), (b), and (c) where the focus was steered at 0, 2.5 , 5 mm, respectively. The six grating lobes that were prominent in the hexagonal array no longer existed. Elements randomly spaced on the array surface produced a field that was free of the undesired grating lobes, and more energy was present at the

focal point. Even when steered 5 mm from the geometric focus, the intensity of the highest grating lobe compared to the focal intensity increased from -14 dB to -8 dB. This is much better than for the hexagonal array where the relative intensity of the grating lobe was about -9 dB without steering and +1.5 dB when the focus was steered 5 mm.

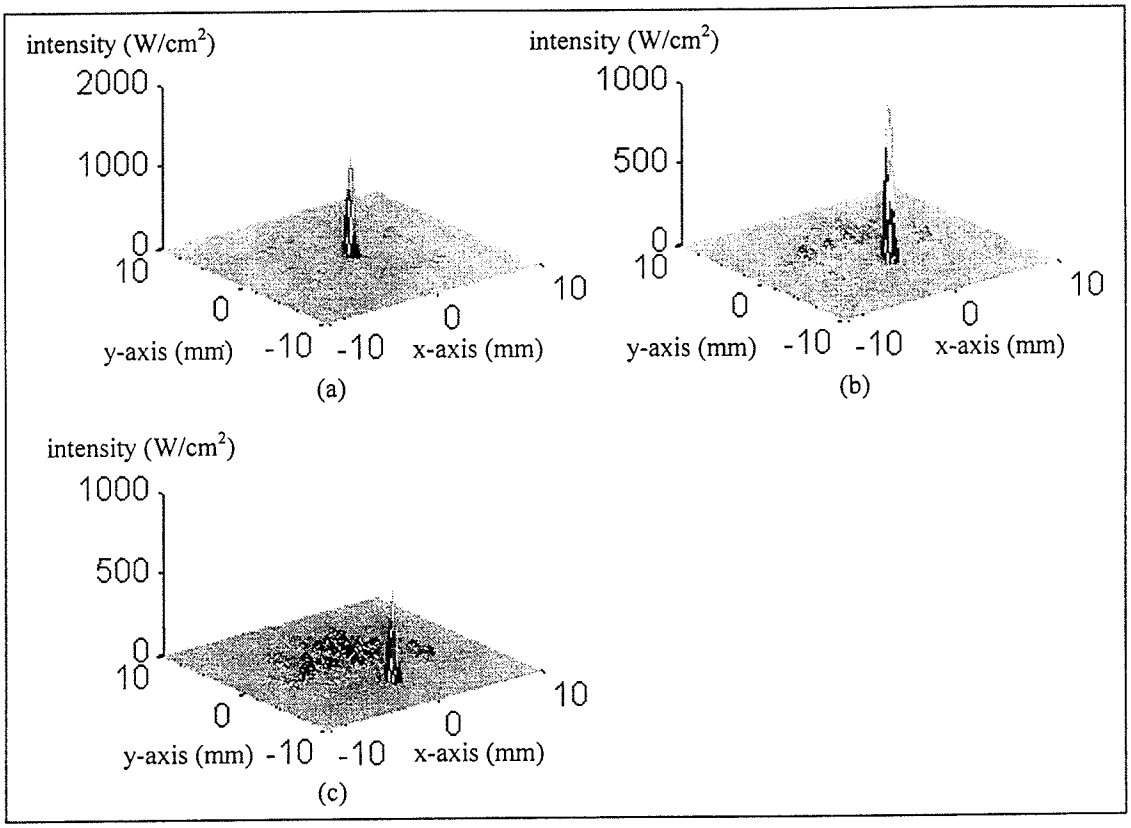

Figure 3.3 Above illustrations show the intensity in a transverse plane through the focus for the random array when focus is located at (a) the geometric focus (0,0,0), (b) 2.5 mm of the geometric focus, and (c) 5 mm off the geometric focus.

3.3 Effect of Different Random Element Coordinate Sets

The question that arises with the choice of many possible sets of random element positions is whether there is a set that can produce the lowest grating lobe level. Many other research groups have proposed complicated algorithms in an attempt to find an answer to this question [13-15]. The optimization method used in this study was much simpler than the methods used by other laboratories. First, many sets of element positions were generated by the computer in a pseudo-random fashion. Then, every one of these coordinate files was fed into the RSAS and simulated for various focal locations. Specifically, eight positions, all of which were 5 mm from the geometric focus, were used (see Figure 3.4).

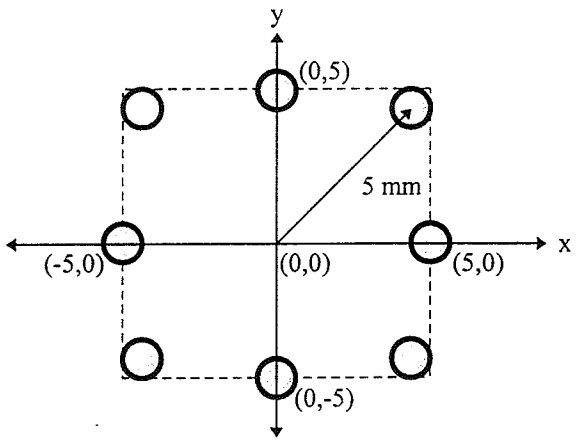

Figure 3.4 Focal positions for the grating lobe study.

The number of grating lobes and the intensity of the highest grating lobe on the focal plane within a 2 x 2 cm^2 area around the geometric focus were recorded using a computer algorithm. Note here that the number of grating lobes was defined as the number of points which were at least 10 W/cm^2 above its nearest neighbor, and the highest grating lobe level was the intensity level of the next highest peak from the

focus. The set of random coordinates that produced the lowest average number of grating lobes was then used for all of the system parameter studies. Table 3.1 shows the average number of grating lobes and average highest grating lobe level for five sets of random coordinate files. Currently, element positions in set III are utilized in the RSAS.

Table 3.1 Summary of the number of side lobes and side-lobe level for various random coordinate files.

Random Coordinate Set	AVG Number of Side Lobes	AVG Side Lobe Level
I	51	-6.6
II	44	-6.5
III	40	-6.3
IV	55	-6.9
V	45	-5.0

CHAPTER 4

EFFECT OF DESIGN PARAMETERS

This chapter will analyze how the input and output parameters of the array system are related to each other. Based on the information obtained from this analysis, a mathematical model of the array was constructed which correlates the input and the output parameters. Then, two optimization strategies are proposed from the mathematical model to improve system performance.

4.1 System Overview

In order to improve the current design, a more in-depth look into the relationship between the various input and output parameters is essential. Figure 4.1 shows a general structural diagram of the Multi Input-Multi Output (MIMO) array system.

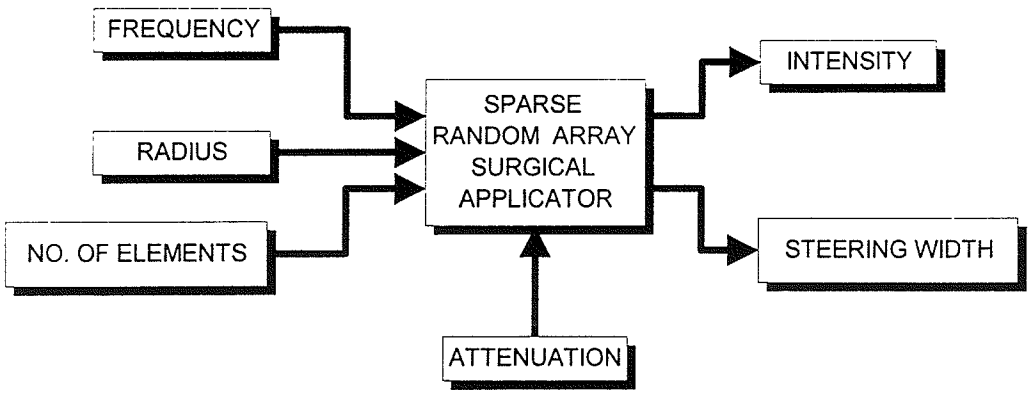

Figure 4.1 MIMO model of the random sparse array system.

The figure shows that by controlling the driving frequency, radius of the transducer, number of transducer elements, and attenuation along the path, the output parameters

such as peak focal intensity and steering width can be modified. In the following discussions, predictions will be made about how the inputs can affect the behavior of the outputs based on theory. Then, the data from the simulation will be compared to the theoretically derived results to verify the accuracy of our prediction.

4.2 Output vs. Tissue Thickness

In this section, the effect of tissue thickness (T) on intensity is investigated at different frequencies and radii. Tissue thickness is defined as the distance from the plane interface between the coupling medium and the tissue to the focal plane (see Figure 4.2).

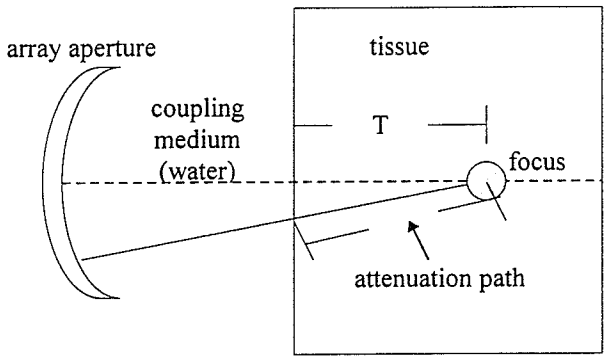

Figure 4.2 Beam path from source to the point of focus.

As can be seen, by increasing the tissue thickness, one effectively increases the portion of the path with attenuating tissue. The attenuation coefficient, α, is given as a function of frequency, f, in MHz by

$$\alpha = 0.005(f)^{1.1} \text{ (Np/mm)} \qquad (4.2.1)$$

Consequently, one would anticipate that the peak focal intensity of the array would be inversely related to the tissue thickness. Figure 4.3 confirms this prediction.

Both Figure 4.3(a) and 4.3(b) show that as we increase the tissue thickness, the intensity decreases as a result.

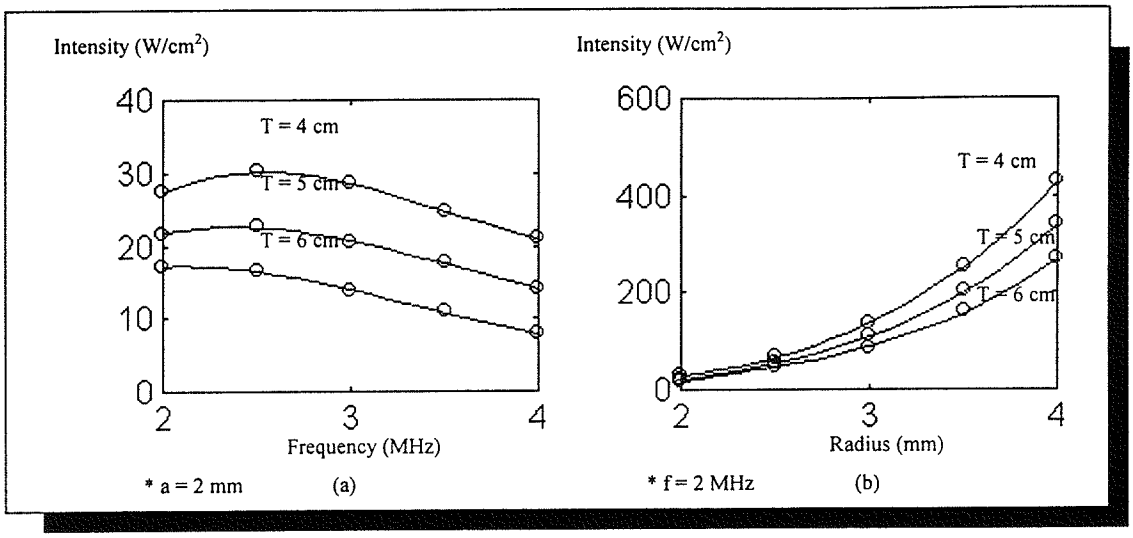

Figure 4.3 Effects of tissue thickness on acoustic intensity output. Plot (a) shows the effect of varying frequency, while plot (b) shows the effect of varying radius of the element.

4.3 Intensity Vs. Number of Elements

The resultant pressure field of the array is the sum of the contributions from all the individual elements. Thus, one would expect the intensity of the array to increase linearly as a function of number of elements. Simulations have been run to verify the validity of this assumption. The results are shown in Figure 4.4.

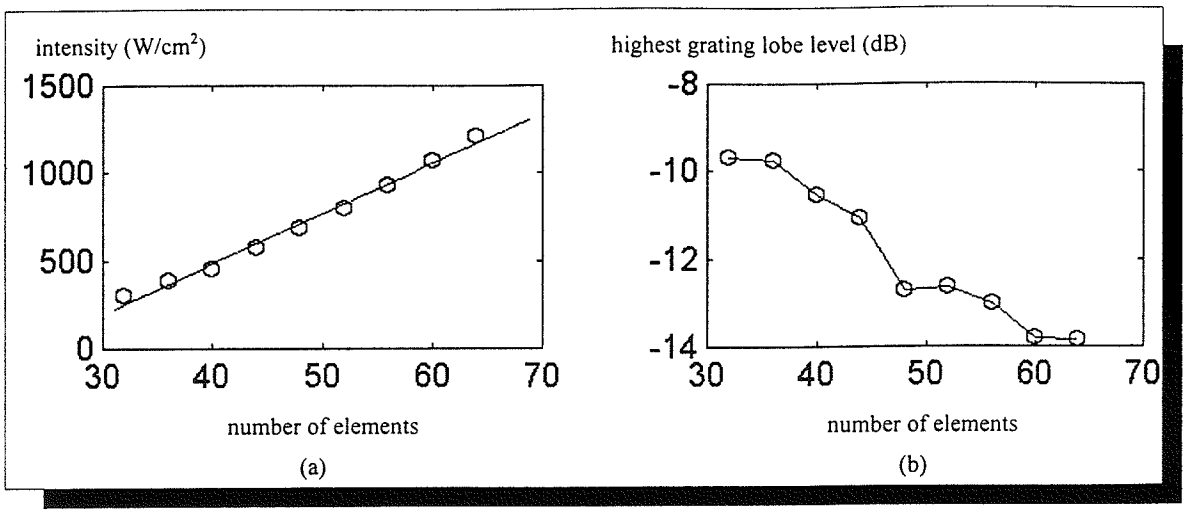

Figure 4.4 Effect of changing the number of elements in RSA. Plot (a) shows the relationship between intensity and the number of elements. Plot (b) shows the grating lobe level as number of elements increase.

Figure 4.4(a) shows that the intensity of the array is a positive linear function of the number of elements as predicted. Also, as the number of elements is decreased, the intensity of the highest grating lobe compared to the focal intensity increases drastically, as presented in Figure 4.4(b). Since the purpose of using the sparse array configuration is to minimize the size of the grating lobes while using the least number of elements, results of this study suggest that there is a lower limit as to how "sparse" an array can be.

4.4 Intensity vs. Frequency

The formula below is the equation of pressure amplitude on axis for a single piston source.

$$P_{ax}(r) = \frac{1}{2}\rho_o c U_o \frac{a}{r} ka = \left(\frac{\rho_o U_o \pi}{z}\right) a^2 f \qquad (4.4.1)$$

Since $I \propto P^2$, one can see that the intensity is proportional to the square of the frequency for a single piston element. By the fact that the composite pressure of the array is just the superposition of pressure produced by each individual transducer, the effect of frequency on intensity of a single piston source should be a relatively good model for the array system. A simulation study has been conducted to record intensities of the array with frequencies between 2 to 4 MHz at 0.5 MHz intervals, for a lossless case (no tissue) and for a tissue thickness of 5 cm. The results are shown in Figure 4.5 for three different transducer radii. The extrapolation lines fit to the data points are calculated using a third-order polynomial.

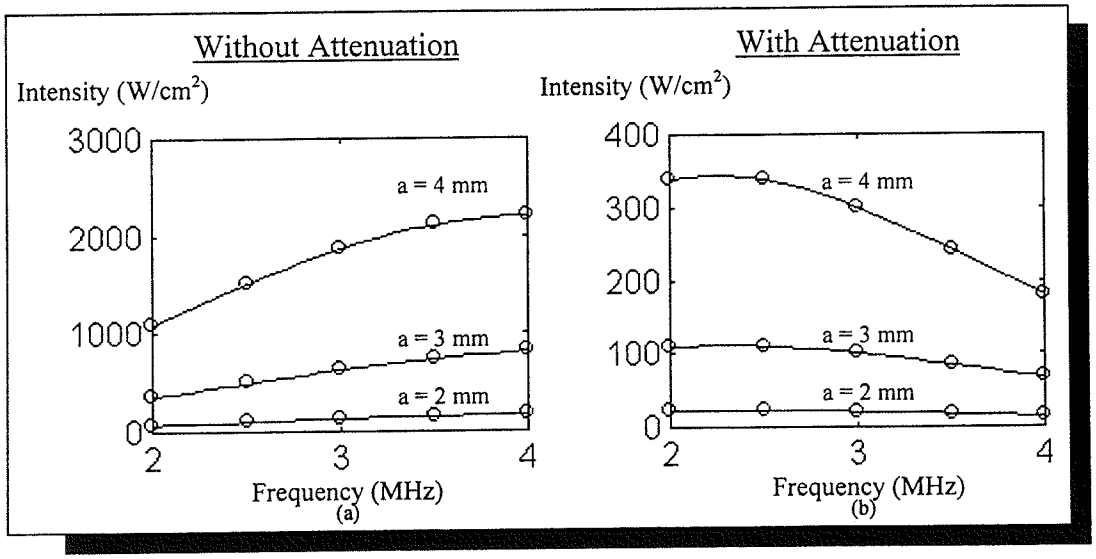

Figure 4.5 Intensity plots showing the effect of varying driving frequency. Plot (a) presents the relationship without attenuation, while plot (b) presents the relationship with attenuation.

Where waves travel in a lossless medium, as shown in Figure 4.5(a), the results indicate that the square law does apply for all three different radii. However, for the 5 cm thickness of tissue, Figure 4.5(b) shows that the intensity decreases with

frequency over most of the range examined since attenuation is strongly frequency dependent. In fact, there is a maximum in intensity versus frequency indicating that there is an optimal frequency at which maximum intensity can be realized.

4.5 Intensity vs. Radius of Element

From Equation (4.4.1), it is obvious that intensity is directly related to the radius of the element to the fourth power. Simulation results are shown below for the radius of elements ranging from 2 mm to 4 mm for the lossless case and for a tissue thickness of 5 cm.

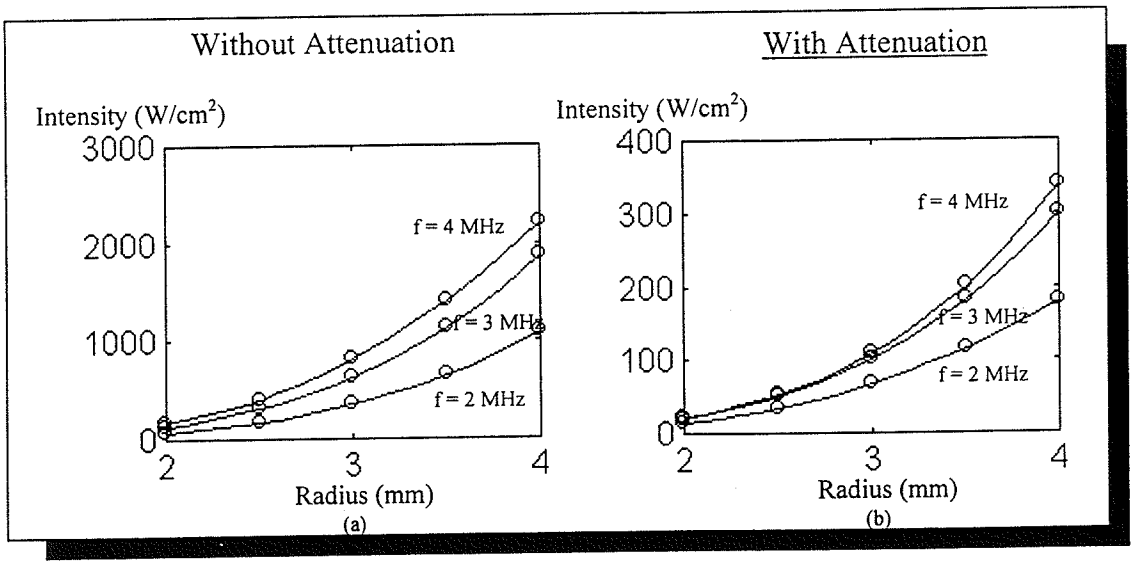

Figure 4.6 Plots showing intensity at the focus versus radius of the element at different frequencies. Plot (a) presents the relationship without attenuation, while plot (b) presents the relationship with attenuation.

Where attenuation is negligible, as in Figure 4.6(a), the theory correctly predicts the relationship between the peak focal intensity of the array and the radius of the element. Figure 4.6(b) shows the effect of the attenuation for 5 cm thick tissue.

The magnitude of intensity falls almost by an order of magnitude in comparison to the case for a lossless medium.

4.6 Steering Width vs. Input Parameters

Next, the relationship between frequency and steering width was investigated. The steering width here was defined as the distance between the half-power points on the steering envelope. Theory suggests that the steering envelope of the array can be described by the equation of the directivity pattern for the single piston source (see Equation (4.6.1)).

$$H(\theta) = \left[\frac{2J_1(ka \sin\theta)}{ka \sin\theta} \right] \qquad (4.6.1)$$

where k is the wave number, a is the radius of the element, and θ is the angular departure from the beam axis. To verify this, the half-power width of the directivity pattern for the single piston source and the steering width of the random sparse array were calculated, and are presented in Figures 4.7(a) and 4.7(b), respectively. The data from Figure 4.7 confirms the theoretical prediction because the plots look exactly alike.

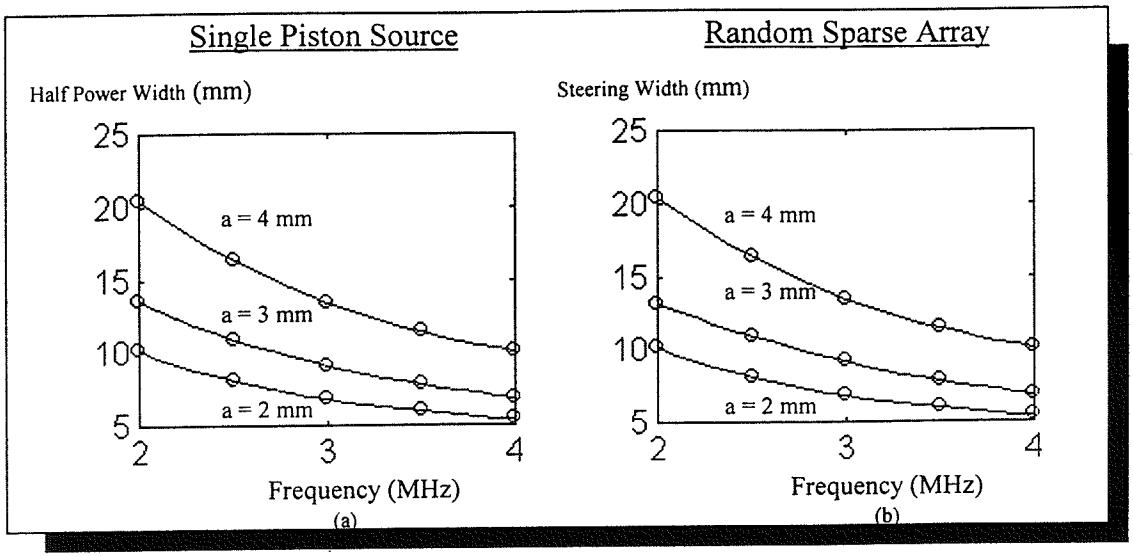

Figure 4.7 Plots (a) and (b) show the half-power width of the directivity pattern for a single transducer and the steering width of the array, respectively.

Several observations can be made for $H(\theta)$. First, it does not depend on the attenuation constant α. Thus, one would expect the steering width of the array to remain the same in either a lossy or a lossless medium for the same combination of frequency and element radius. Second, a closer look at $H(\theta)$ suggests that the only independent variables are frequency f and radius a, which are contained in the (ka) term. The value of (ka) will be a constant for cases in which the products of frequency and radius are the same. In other words, the steering width for these particular cases will be identical. In the parameter space investigated, if the radius of the element is varied instead of frequency , the result closely resembled the result of modifying the frequency as shown in Figure 4.7(b).

Figure 4.8 shows the simulation result of steering width versus (ka). Note that the steering width is indeed unaffected by attenuation and that the relationship between steering width and (ka) agrees with that given by $H(\theta)$.

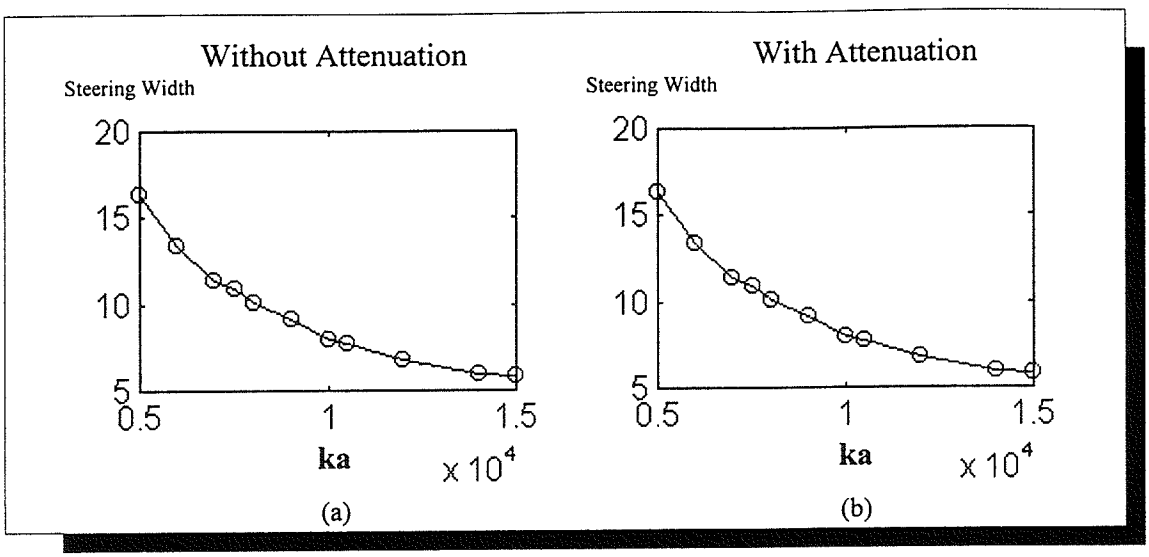

Figure 4.8 Plot of steering width as a function of (ka). Plot (a) presents the relationship without attenuation, while plot (b) presents the relationship with attenuation.

4.7 Optimizing Output Using Design Parameters

In order to optimize the output of the system, an equation is needed to describe the quantity to be optimized for the system. Since the intensity output of the array is basically the superposition of all the element outputs, the equation for the array must contain all the dependencies which appear in the single element case. From Equation (4.2.1), the intensity is known to be proportional to the square of the frequency and to the fourth power of radius. Assuming homogeneous media, the attenuation effect can be accounted for by multiplying by an exponential term which contains the attenuation coefficient. The resulting equation is

$$I(a,f)= K_{array} f^2 a^4 e^{-2\alpha T} \qquad (4.7.1)$$

where K_{array} is a constant which contains all the superposition effects of the array,

f is frequency in megahertz and T is the tissue thickness in millimeters at which the attenuation occurs. However, due the placement of the sources, every element "sees" a different length of path through the tissue. Thus, an effective thickness must be found which pertains to what the entire array "sees" as the tissue thickness. The Equation (4.7.1) can be manipulated to yield an effective tissue thickness in the following form.

$$T_{eff} = 10\ln\left(\frac{K_{array} f^2 a^4}{I}\right)\left(\frac{f}{1x10^6}\right)^{-1.1} \quad (cm) \qquad (4.7.2)$$

Effective tissue thickness then can be arrived at iteratively by using the different combinations of frequency, radius, and intensity found in Chapter 4. The result is summarized in Table 4.1.

Table 4.1 Effective tissue thickness at various actual tissue thickness.

T	K	T(effective)
4.0 mm	1.75	6.65 mm
5.0 mm	1.75	7.50 mm
6.0 mm	1.75	8.80 mm

To test whether or not Equation (4.7.1) describes the system precisely, a plot of intensity versus frequency was generated by substituting the effective tissue distance and K_{array} into Equation (4.7.1) (see Figure 4.9). The data from the simulation are also included as circles for comparison.

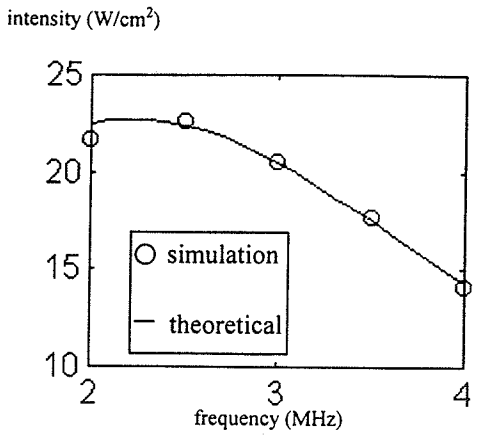

Figure 4.9 Comparison of the analytical form using T_{eff} and results from simulation, where the tissue distance is 5 cm and the radius of elements is 2 mm.

As can be seen, Equation (4.7.1) is a very good model that describes the relationship between intensity and frequency. To find the frequency that will produce the highest intensity at the focal point, one can utilize the maximum value property of a continuous function and simply take the partial derivative of Equation (4.7.1) with respect to a and f. Note here that since the effective attenuation path changes for different tissue thicknesses, the optimal frequency also changes with a varying tissue thicknesses. As an example, at a=4mm and T=5cm, the optimal frequency is calculated to be about 2.24 MHz.

4.8 Optimizing Steering by Moving Focal Plane along the Beam Axis

An alternative method that can increase the steering width of the array is to move the focal plane up or down the beam axis from the natural focal plane, where the natural focal plane is defined as a transverse plane perpendicular to the beam axis which contains the geometric focus of the array. However, this increase in steering

width is achieved at the expense of decreased output intensity. Figure 4.10 shows the plots of intensity and steering width of the array when the focal plane is varied over the range of -2 to 2 cm from the geometrical focal plane in 0.5 cm steps.

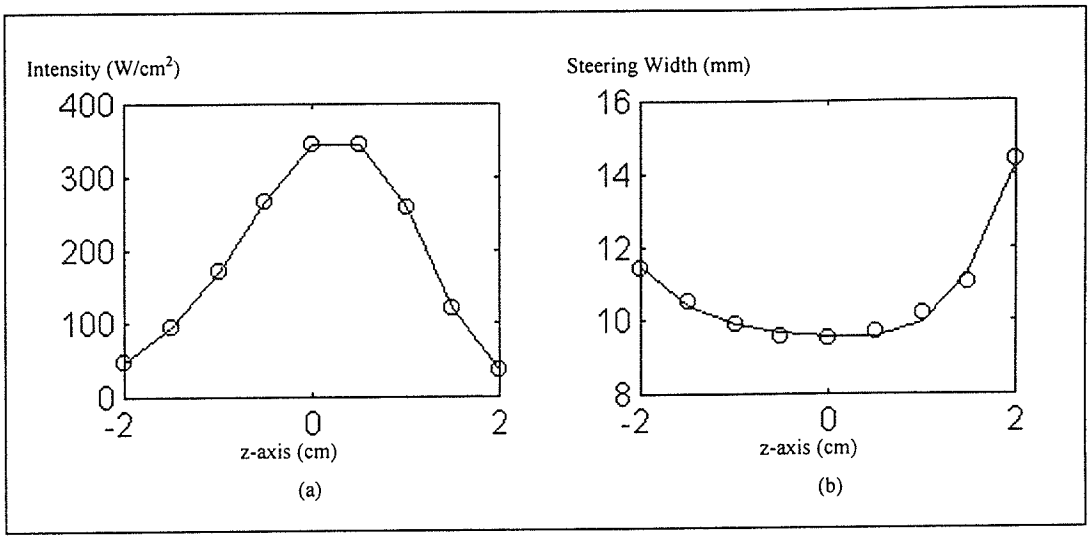

Figure 4.10 Intensity and steering width along the beam axis. Plot (a) shows the varying in intensity as the focus moving along the z-axis. Plot (b) shows the varying in steering width as the focus moving along the z-axis.

The data from Figure 4.10(a) indicate that by moving the focal plane from the plane containing the geometric focus, the intensity of the array drops very sharply. At the $\pm$ 2 cm mark, the intensity drops by an order of magnitude while the steering width increased by a maximum of only 4 mm (see Figure 4.10(b)). The results of this investigation suggest that this method is not an optimal way to expand the steering width of the array.

CHAPTER 5

EFFECT OF MODIFYING TRANSDUCER CHARACTERISTICS

In this chapter, changes in the characteristics of the transducer elements will be examined to determine if they enhance system output, i.e., increase steering width with minimal loss of focal intensity. Two approaches are evaluated, one using transducers that have a Gaussian beam profile and the other utilizes focused transducers. The results are compared to those using circular unfocused sources.

5.1 Using Gaussian Transducer

Breazeale et al. [16] have devised a way to fabricate a transducer that can produce a Gaussian beam profile. The transducer cross section is illustrated in Figure 5.1.

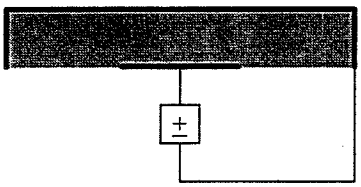

Figure 5.1 Simplified view of a Gaussian transducer in cross section, shaded portion is transducer material.

The face of the transducer element, the radiating face, is covered entirely with a ground electrode while the back electrode is connected to the positive lead of the power amplifier. Then, the positive electrode is apodized to decrease the surface area of the back electrode. The resultant beam has two very useful features. First, as the

wave propagates, the sound pressure on the axis reduces gradually with distance. In the radiated beam, none of the maxima and minima typical of the Fresnel zone of a piston transducer appears. Second, the beam does not develop the side lobes characteristic of the far-field directivity pattern of a piston transducer. This Gaussian beam study was conducted with the expectation of extending the steering width of the random sparse array.

5.1.1 Gaussian diffraction theory

Du and Breazeale [17] obtained a relatively accurate description of the Gaussian beam from the basic linear wave equation. The details of the derivation are included in Appendix E. Since the RZ grid needs a phase term for the pressure, Equation (E.9) in Appendix E must be separated into real and imaginary parts. Again, attenuation will not be taken into account as in the case of the non-Gaussian piston source. By using all the related equations in Appendix E, the pressure equation is derived to be the following

$$P(\xi,\sigma)= K_G e^{i(-kz+\gamma)} = K_G e^{i(\theta)} \Rightarrow P(r,z)= K_G(\sin\theta + j\cos\theta) \qquad (5.1.1)$$

where

$$K_G = (\rho_o c_o U_o)\left(\frac{1}{\sqrt{1+(B\sigma)^2}}\right)\left[\exp\left(-\frac{B}{1+(B\sigma)^2}\right)\xi^2\right] \qquad (5.1.2)$$

The above form looks exactly like the form of the equation for the non-apodized source in Barich [12]. The difference is in how the constant and the angle are defined.

5.1.2 Analysis of the Gaussian beam study

Simulation cases have been run to compare the Gaussian beam and the non-Gaussian beam. *RZ-Grid Generator Module* was modified to output the pressure field for a transducer with Gaussian characteristics (see Appendix C). A value for B of 0.486 was used in the Gaussian beam simulation as suggested by Du and Breazeale [16]. The results are shown in Figure 5.2. Note that the pressure of the Gaussian transducer falls off smoothly unlike the non-Gaussian transducer where there is a rapidly varying near-field region. Both transducers produced pressure that is relatively close in magnitude at 10 cm, which is the focal plane of the array system.

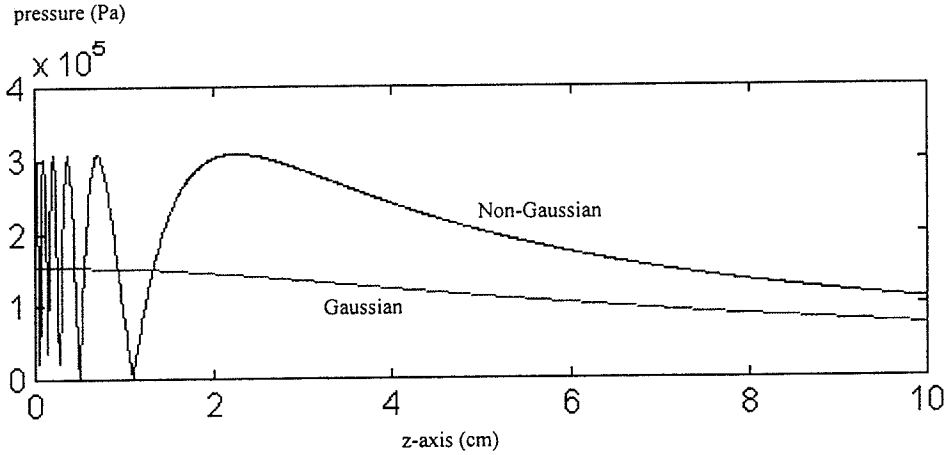

Figure 5.2 Comparison of the pressure field along the z-axis for both the non-Gaussian and Gaussian transducer.

The simulation also shows that the steering width of the Gaussian transducer was not much better than for the non-Gaussian transducer at the same driving frequency and radius (see Table 5.1). To verify this, the steering width for both the Gaussian and non-Gaussian transducers was found experimentally use the equipment at Labthermics Technologies, Inc (Champaign, IL). The steering widths found for

these two types of transducer, both by simulation and experiment, are tabulated in Table 5.1.

Table 5.1 Steering width of the Gaussian and non-Gaussian transducer.

Setting \ Transducer	Non-Gaussian	Gaussian
Simulation	10.01 mm	11.50 mm
Experiment	9.6 mm	12.0 mm

The results of this study suggest that no gain can be realized by altering the elements to produce a Gaussian beam profile. The present setup with non-Gaussian transducers can generate intensities and steering widths, which are comparable to those of a Gaussian transducer.

5.2 Using Focused Transducer

Next, the possibility of using a focused transducer to improve the steering width of the array is investigated. The geometric configuration upon which the derivation was based is shown in Figure 5.3. The points on the curved surface can be described by the equation of a circle where the center of the circle is at the point ($-R_c$, 0), and the radius is equal to the radius of curvature.

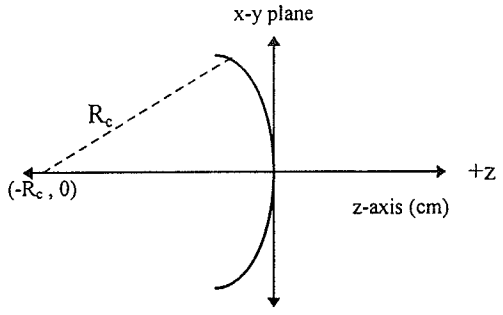

Figure 5.3 Graphical description of the convex transducer surface using radius of curvature.

Equation (5.2.1) shows the equation of the circle as described.

$$\left(z-(-R_c)\right)^2+\left((x^2+y^2)-0\right)=R_c^{\,2} \qquad (5.2.1)$$

With a little mathematical manipulation, the above equation can be simplified to the standard quadratic form.

$$z^2+(2R_c)z+(x^2+y^2)=0 \qquad (5.2.2)$$

Thus, two values of z can be obtained by using the quadratic formula. However, only the z value that is closest to the x-y plane is desired. The transducer can be changed from an unfocused to a focused state in the simulation by adding the phase delays to points along the flat surface of the unfocused transducer to emulate a curved surface. The phase delay is defined as

$$\text{phase}=2\pi\left(\frac{z}{\lambda}\right) \qquad (5.2.3)$$

A Matlab function was constructed to verify the above equations. The function allows the focus to be positioned in either the positive region, toward the tissue, or the negative z region. For the focus placed in the positive z region, at a value less than 10 cm, the field from each source will converge at its focus, but then diverge as it approaches the geometric focus of the array. For the focus in the negative z region, the field will diverge as it approaches the geometric focus of the array. A copy of the function is included in Section D.3.

Figure 5.4 shows the results of the simulation for a single source. The plots were compiled using the pressure profile at 10 cm, the geometric focal plane of the array, for distance off axis , r, ranging form 0 to 1.5 cm, for various radii of

curvature. The difference between Figure 5.4(a) and 5.4(b) is that the former used negative radii of curvature while the latter used positive radii of curvature.

As the absolute value of the radius of curvature decreases, the magnitude of the pressure at the center of the beam drops about an order of magnitude for a 3 cm change in radius of curvature. This pressure reduction is due to the fact that the energy from the main lobe is transferred to the side lobes. However, the calculated beam width and, therefore, the steering width of the array did not change much with varying radius of curvature. In fact, the beam width was still largely controlled by the radius of these small elements. In conclusion, the focused transducer did not prove to be a suitable method for optimizing the array.

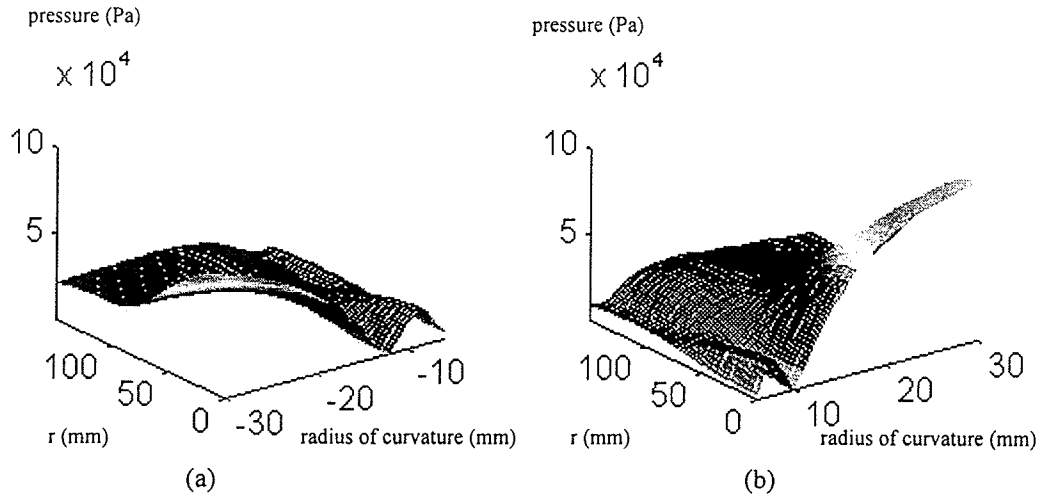

Figure 5.4 Pressure magnitude as a function of radius of curvature and r position. Plot (a) shows the varying in pressure with negative radius of curvature. Plot (b) shows the varying in pressure with positive radius of curvature.

CHAPTER 6

CONCLUSIONS AND CLOSING REMARKS

This study has characterized various system parameter relationships for the random sparse array. Several methods were examined in an attempt to optimize intensity output and steering width of the array for a constant number of elements, 64. However, even though some of the methods examined could enhance one output parameter, it was at the expense of the other. No one method was able to improve significantly both of these output quantities simultaneously. The results of this theoretical study suggest that the best system configuration is a random placement of the 64 elements, with a radius of 2 mm, which is driven at a frequency about 2.23 MHz, for a 5 cm tissue thickness. This array system will produce a steering width of 2 cm and the maximum intensity for a 5 cm thickness of homogeneous tissue, as described in Section 4.7. However, in order for the 2 mm radius element to produce the same intensity as the 4 mm radius element used in the experimental hexagonal array, the transducer has to be driven at a considerably larger amplitude. This modification may shorten the transducer lifetime. A more appropriate method is to increase the number of elements in the array since the reduced element radius leaves more surface area on the aperture for the random placement of more transducer elements.

Future work should address the development of a more realistic computational model. For instance, under actual clinical conditions, there will rarely be a layer of tissue that is uniform in composition. In other words, scattering and refraction

effects must be taken into account to better simulate actual physical conditions. Also, the effect of perfusion should be studied using the bio-heat equation to estimate the temperature changes of the tissue due to the ultrasound. This result should be accompanied with actual experimental verification. In addition, the study of a random sparse array by Erstad and Holm [15] indicates that it is possible to mathematically produce a random array configuration that can produce an average grating lobe level four times lower than for a simple pseudo-random positioning of the array elements. Thus, it may be worthwhile to investigate a method to optimize the positioning of the array elements to reduce the grating lobe levels.

In conclusion, the theoretical work shows that the random sparse array has enormous potential as a noninvasive surgical device. The electronic beam synthesis offers great flexibility for treating tissues in various locations with widely varying volumes.

APPENDIX A.

RANDOM SPARSE ARRAY SIMULATOR

This appendix contains the simulation code for calculating the pressure profile produced by the spherical shell surgical array. This code has been modified from its previous form in many ways. The actual optimizations are described in Chapter 2. The array system modeled by this code contains 64 randomly positioned ultrasound transducer elements. Independent variables such as tissue distance and driving frequency can be changed to study the system under various conditions. The total acoustic pressure generated by the array at any point of the field is computed by superposition of all of the pressure field of the individual array elements. The actual simulation code is listed below.

```
//*************************************************************************
//
//        Load Library of Functions
//
//*************************************************************************
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <fstream.h>
#include <time.h>
#include <string.h>
#include <iomanip.h>
#include <stdio.h>

//*************************************************************************
//
//        This first section defines the globals and subroutine that are to
//                be used in the simulation
//
//*************************************************************************
// Structure Definitions
  struct complex { // define structure for complex pressure
   double rp,ip; // real and imaginary parts
   };
   struct source {  // define structure to be used for each source in array
   float xcord,ycord,zcord; // location of source
```

```
  float phase,amp;          // phase of source
  float tissue_distance;
  int element_number;
  };

// Cylindrical source pressure amplitude calculation using the
// point source approximation. Using cylindrical symmetry
// Constructs the rz array for a single transducer
void get_rz_array(void);

// Load the sources activated instead of random select
void get_source_from_file(source [],const char [],const char []);

// determine phasing of the 64 sources to achieve desired focus
// Inputs are ramdom positions array and the focus location
void get_source_phasing(source [],const source,const char []);

// Complete main grid calculation
void calculate_grid(const source [],const source,char[]);

// Declare globals

  int number_of_sources=64;// Reads in from state file-needed in grid
                                                    // calculation (64=default)
  const int r_length=600;
  const int z_length=1000;
  complex rz_array[r_length][z_length]; // complex pressure array for
                                                                    // single transducer 5cm by 10 cm
                                                                    // spacing of .1 mm
  float intensity_final;
  float freq=2.133e6;

//*************************************************************************
//
//        Program Main
//
//*************************************************************************
main()
{
  source source_array[64]; // max possible size for source array

  char out_file_id[10];// name of output files
  char randcord[10];// name of file contains random source position
  char external_state_file[10];
  char control_option,state_file_option,source_location_option[1];
  ifstream from_control_file;      // link to control file
  source focus;

  //set up clock
  clock_t start, finish;
  double  duration;
  start = clock();

  cout << endl;
  cout << "Welcome to the Ramdomized Sparse Array Simulator" << endl;
```

```
cout << "Reading From RZ-Grid" << endl;

//Load the rz array
get_rz_array();
//Read from control file
 cout << "Reading From Control File" << endl;
 from_control_file.open("control",ios::in);
 from_control_file >> control_option;

 while (control_option == 'Y') {
  cout << "Run another? " << control_option << endl;

  //Enter output identifier (5 char)?
  from_control_file >> setw(6) >> out_file_id;
  cout << out_file_id <<endl;

  //Enter focus x,y,z in mm
  from_control_file >> focus.xcord >> focus.ycord >> focus.zcord;
  cout << "focus at " << endl;
  cout << "(" << focus.xcord << "," << focus.ycord << ","
     << focus.zcord << ")" << endl;

  //Enter name of random source location file(TOTALLY RANDOM!!!!)if call for
   from_control_file >> source_location_option;
   cout<<"random(r) or hex(h)?"<<source_location_option<<endl;
   from_control_file >> randcord;
   cout << "source location file = "<< randcord << endl;

  //Establish source positions on shell
  from_control_file >> state_file_option;
  cout << "Use external state file? " << state_file_option << endl;
    from_control_file >> external_state_file;
    get_source_from_file(source_array,external_state_file,randcord);
          cout << "source state file = "<< external_state_file << endl;

  //Establish source phasing
  get_source_phasing(source_array,focus,out_file_id);
  cout << "Source Phasing Completed" << endl << endl;

  // Complete main calculations
  calculate_grid(source_array,focus,out_file_id);
  cout << "Main Grid Calculated "  << out_file_id << endl;

  //load new control option
  from_control_file >> control_option;
}; // close control while loop

// close stream to control file
  from_control_file.close();

// print out total process time
finish = clock();
duration = (double)(finish - start) / CLOCKS_PER_SEC/60;
cout << endl <<endl;
printf( "Total process time = %2.1f min\n", duration );
```

```
  return 0;

}// end main
//*************************************************************************
//
//          Subroutine get_rz_array
//
//*************************************************************************
void get_rz_array(void)

 {
  //write binary output file
  FILE *infile;
  infile=fopen("rz2mm2hz","rb");
  for(int r=0;r<r_length;r++){
     for (int z=0;z<z_length;z++){
  fread (&(rz_array[r][z]),sizeof(complex),1,infile);
     }
  cout << r <<endl;
   }
  fclose (infile);
  cout << "RZ-Grid loaded"<<endl<<endl;
  }

//*************************************************************************
//
//          Subroutine get_source_from_file
//
//*************************************************************************

void get_source_from_file (source sarray[],const char state_file_name[],
                    const char rand_file_name[])

{

  ifstream from_xyz_file;  // establish input stream
  ifstream from_state_file;
  source element_positions[108];
  int discard,state;
  int counter;
  char buffer[100];


  from_xyz_file.open(rand_file_name,ios::in);

  //get rid of comment lines at top of file
  from_xyz_file.getline(buffer,100);
  from_xyz_file.getline(buffer,100);


  //Read xyz cords (in mm) from file
  for(int i=0;i<108;i++){
    from_xyz_file >> element_positions[i].element_number;
    from_xyz_file >> element_positions[i].xcord;
    from_xyz_file >> element_positions[i].ycord;
```

```
    from_xyz_file >> element_positions[i].zcord;
  }
  from_xyz_file.close();

  // Read the states of the sources from file and assign
  from_state_file.open(state_file_name,ios::in);
  from_state_file.getline(buffer,100);

  counter=0;
  for(int j=0;j<108;j++){
    from_state_file >> discard;
    from_state_file >> state;
    if (state == 1){
      sarray[counter].element_number=element_positions[j].element_number;
      sarray[counter].xcord=element_positions[j].xcord;
      sarray[counter].ycord=element_positions[j].ycord;
      sarray[counter].zcord=element_positions[j].zcord;
      counter++;
    }
  }
  number_of_sources=counter;
  from_state_file.close();
}// end get_source_from_file

//*************************************************************************
//
//          Subroutine get_source_phasing
//
//*************************************************************************

void get_source_phasing(source sarray[],const source focus,const char id_string[])
{
  ofstream to_phase_file;
  int phase_int;
  float distance_change,phase_change,distance_to_source;
  char temp_id_string[10];

  strcpy(temp_id_string,id_string);
  //Open stream to output phase output file
  to_phase_file.open(strcat(temp_id_string,"pha"),ios::out);

  // produce phase front at desired focus
  for(int n=0;n<number_of_sources;n++){
    distance_to_source=float(sqrt(pow(sarray[n].xcord - focus.xcord,2)
                  +pow(sarray[n].ycord - focus.ycord,2)
                  +pow(sarray[n].zcord - focus.zcord,2)));//(mm)
    distance_change=float( -102.26 + distance_to_source);//
    phase_change=float(distance_change * 360/(1500/freq*1000)); //*360 degrees/wavelength(mm)
    sarray[n].phase=float(phase_change * 3.14159/180);// convert to radians
    sarray[n].amp=1 ;//- 100 * distance_change/102.26;

    // 16 level quantization for phase
    sarray[n].phase=float(sarray[n].phase/(2*3.14159)*16);
    phase_int=int (floor(sarray[n].phase+0.5));
    sarray[n].phase=float(phase_int*0.392699);  //0.392699rad = 22.5 degree
```

```
    //end quantization

    to_phase_file << n <<"  "<< sarray[n].phase << endl;

    // Find distance traveled through tissue
    // Interface at -50mm
    sarray[n].tissue_distance=(50/sarray[n].zcord)*distance_to_source;

  }
   to_phase_file.close();

}// end get_source_phasing

//*************************************************************************
//
//          Subroutine calculate_grid
//
//*************************************************************************

void calculate_grid(const source source_array[],const source focus,
                char id_string[])
{

  complex total_pressure,new_pressure;
  // physical constants
  const float alpha=0.0;
  const float tissue_attenuation=float(0.005*pow(freq/1e6,1.1)*alpha); //Np/mm

  //main grid variables
  float c_scale,r_projection,z_projection,pressure_amp,intensity;
  //float total_power,old_intensity;
  float xpos,ypos,zpos;
  float amplitude,angle;
  int r_low,r_high,z_low,z_high; // nearest points in rz grid
  int counter;
  ofstream to_grid_file;
  float d_between_p1p0,d_between_p2p0,d_between_p3p0,d_between_p4p0;
  float d_between_p1p4;
  float alpha1,alpha2,alpha3,alpha4;


  //open stream to output file
  to_grid_file.open(strcat(id_string,"grd"),ios::out);
  cout << "Beginning Calculations" << endl;

  //output attenuation status
  if (alpha == 0.0){
  cout << "Attenuation OFF" << endl;}
  else {
  cout << "Attenuation ON" << endl;}

//for each point in main grid, calculate complex contribution
//from each of the 64 sources
// x, y, z from -3 cm to +3 cm = -300 to +300 (by .1 mm)
  intensity_final=0;
```

```
counter=0;

for(int z=0;z<1;z+=1){

  for(int y=100;y>=-100;y-=2){

    for(int x=-100;x<=100;x+=2){

      xpos=float(x/10.0);ypos=float(y/10.0);zpos=float(z/10.0);  //convert to mm
      total_pressure.rp=0.0;
      total_pressure.ip=0.0;
      //Sum contributions from all sources for each grid point
      for(int i=0;i<number_of_sources;i++){
        new_pressure.rp=0.0;
        new_pressure.ip=0.0;
        c_scale=(source_array[i].xcord * xpos
             +source_array[i].ycord * ypos
             +source_array[i].zcord * zpos)/
             (source_array[i].xcord*source_array[i].xcord
             +source_array[i].ycord*source_array[i].ycord
             +source_array[i].zcord*source_array[i].zcord);
       r_projection=float (10.0*sqrt(pow(xpos-c_scale*source_array[i].xcord,2)
                  +pow(ypos-c_scale*source_array[i].ycord,2)
                  +pow(zpos-c_scale*source_array[i].zcord,2)));

       //Exact radius measurement = 102.26 mm
       z_projection=float (10.0*(102.26 - c_scale*
                    sqrt(pow(source_array[i].xcord,2)
                      +pow(source_array[i].ycord,2)
                      +pow(source_array[i].zcord,2))));

       //If r,z not in array range then error
       if (((r_projection) > r_length)||
          ((z_projection) > z_length+500 )||
          ((z_projection) < 500 ))
          cout << "source " << i << "out of range at "
          << xpos << " " << ypos << " " << zpos << endl;
       //Convert to rz grid (.1 mm spacing, 5 cm z offset)
       //and average for smoother values
       r_low=int (floor(r_projection));
       r_high=int (ceil(r_projection));
       z_low=int (floor(z_projection)+500);
       z_high=int (ceil(z_projection)+500);

       //putting in the weighting factor(alpha)
       d_between_p1p0=float(sqrt((pow(r_low-r_projection,2))
                    +(pow(z_high-(z_projection+500),2))));
       d_between_p2p0=float(sqrt((pow(r_high-r_projection,2))
                    +(pow(z_high-(z_projection+500),2))));
       d_between_p3p0=float(sqrt((pow(r_low-r_projection,2))
                    +(pow(z_low-(z_projection+500),2))));
       d_between_p4p0=float(sqrt((pow(r_high-r_projection,2))
                    +(pow(z_low-(z_projection+500),2))));
       d_between_p1p4=float(sqrt((pow(r_low-r_high,2))
                    +(pow(z_high-z_low,2))));
```

```
   alpha1=float(1.0/d_between_p1p0);
   alpha2=float(1.0/d_between_p2p0);
   alpha3=float(1.0/d_between_p3p0);
   alpha4=float(1.0/d_between_p4p0);

   //calculate new pressure
   new_pressure.rp=(rz_array[r_low][z_low].rp*alpha3
              +rz_array[r_low][z_high].rp*alpha1
              +rz_array[r_high][z_low].rp*alpha4
              +rz_array[r_high][z_high].rp*alpha2)
              /(alpha1+alpha2+alpha3+alpha4);
   new_pressure.ip=(rz_array[r_low][z_low].ip*alpha3
              +rz_array[r_low][z_high].ip*alpha1
              +rz_array[r_high][z_low].ip*alpha4
              +rz_array[r_high][z_high].ip*alpha2)
              /(alpha1+alpha2+alpha3+alpha4);
   //convert complex quantity to mag, angle form
   amplitude=float(sqrt(pow(new_pressure.rp,2)
            +pow(new_pressure.ip,2)));
   if (new_pressure.rp > 0.0){
     angle=float(atan(new_pressure.ip/new_pressure.rp)
          + source_array[i].phase);}
   else{
     angle=float(atan(new_pressure.ip/new_pressure.rp)
          + source_array[i].phase + 3.14159);}
   //convert back to rp,ip
   new_pressure.rp=amplitude*cos(angle);
   new_pressure.ip=amplitude*sin(angle);
   //Add amplitude scaling
   new_pressure.rp=new_pressure.rp *source_array[i].amp;
   new_pressure.ip=new_pressure.ip *source_array[i].amp;
   // Attenuation due to distance in tissue (water attenuation=0)
   new_pressure.rp=new_pressure.rp
              * exp(-1 * source_array[i].tissue_distance
                   * tissue_attenuation);
   new_pressure.ip=new_pressure.ip
              * exp(-1 * source_array[i].tissue_distance
                   * tissue_attenuation);
   //sum total pressure
   total_pressure.rp= total_pressure.rp + new_pressure.rp;
   total_pressure.ip= total_pressure.ip + new_pressure.ip;
   }
//pressure amplitude is magnitude of complex pressure
pressure_amp =float( sqrt(pow(total_pressure.rp ,2)
              +pow(total_pressure.ip ,2)));
              //*.000114; //correction to get Pa(in mm)

// add the following to get max in a array of data
intensity=float(pow(pressure_amp,2)/(2.0*1500.0*1026.0*10000));// W/cm2

if (intensity>=intensity_final){
intensity_final=intensity;}

// send position and pressure to output file
```

```
        //to_grid_file << xpos << "  " << ypos << "  " << zpos << "  "
        //          << pressure_amp * .000001<< "  " //in MPa
        //          << intensity << endl;// in W/cm2
       // total_power=((intensity+old_intensity)/2.0)*.0001*10000*10;// W
       // old_intensity=intensity;

       //below move the pointer to the next row for the 3d-data //

       to_grid_file << intensity << "  ";
       if (counter==100   ){
       counter=-1;
       to_grid_file << endl;}
       counter=1+counter;
       //cout << x << endl;
      }//end x-for loop

    cout << y << endl;
    } //end y-for loop
  //cout << z << endl;
  } //end z-for loop
  //cout << "total power (W) " << total_power << endl;
  cout << "Pressure grid calculated" << endl;
  to_grid_file.close();
}// end calculate grid

//*************************************************************************
//
//          END SIMULATION
//
//*************************************************************************
```

APPENDIX B.

ASCII-BINARY CONVERTER CODE

This code converts the format of the RZ-Grid from ASCII to Binary as described in Section 2.3.3.

```
//*************************************************************************
//
//          Load Library of Functions
//
//*************************************************************************
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include <string.h>
#include <stdio.h>


  struct complex{ // define structure for complex pressure
   double rp,ip; // real and imaginary parts
   };
  const int r_length=600;//large is 600     small is 300
  const int z_length=1000;//large is 1000   small is 400

  complex rz_array[r_length][z_length];       // complex pressure array for
                                              // single transducer 6cm by 10 cm
                                             // spacing of .1 mm

void get_rz_array(void);                      //declare subroutine.

//*************************************************************************
//
//          Program Main
//
//*************************************************************************
void main(void)
 {
  cout << sizeof (complex) <<endl;
  cout << sizeof (rz_array[0][0]) <<endl;
  //Load the rz array(ascii)
  get_rz_array();
  cout << "RZ array loaded"<<endl<<endl;

  //write binary output file
  FILE *outfile;
  outfile=fopen("rz6mm2hz.21h","wb");//sml = smaller rzgrid file
  for(int r=0;r<r_length;r++){
     for (int z=0;z<z_length;z++){
  fwrite (&(rz_array[r][z]),sizeof(complex),1,outfile);
```

```
    }
  cout << r <<endl;
   }
  fclose (outfile);
  cout << "rzdata written"<<endl<<endl;
  }


  void get_rz_array(void)
{
  int zcord,rcord;
  ifstream from_rz_file;  // establish input stream
  from_rz_file.open("smallrz",ios::in);

   //Read rz array from file
    for(int r=0;r<r_length;r++){
      for(int z=0;z<z_length;z++){
        from_rz_file >> zcord ;
        from_rz_file >> rcord ;
        from_rz_file >> rz_array[r][z].rp;
        from_rz_file >> rz_array[r][z].ip;
      }
      cout << r << endl;
  }
  from_rz_file.close();
}//end get_rz_array

//*****************************************************************************
//
//          END PROGRAM
//
//*****************************************************************************
```

APPENDIX C.

GAUSSIAN PISTON SOURCE RZ GRID GENERATOR

This appendix contains the simulation code for computing the pressure field of a single Gaussian piston source as discussed in Chapter 5. This computational model uses an RZ coordinate system with the z-coordinate along the beam axis and r-coordinate perpendicular to the beam axis. The complex acoustic pressure was derived in Equation (5.1.2) and is implemented by the point radiator method in the simulation. The actual simulation code is listed below.

```
//*************************************************************************
//
//        Load Library of Functions
//
//*************************************************************************

#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <fstream.h>
#include <stdio.h>
#include <time.h>

//*************************************************************************
//
//        This first section defines the globals and subroutine that are to
//                be used in the simulation
//
//*************************************************************************

        // Structure for complex quantities
        struct complex {
                double rp,ip; // real and imaginary parts
        };

        // Structure for vector quantities
        struct vector {
                float xcord,ycord,zcord; // e.g. location of source
        };

//              // Construct the rz array for a single transducer using
//              // pnt source methods
//      void construct_rz_array(void);
```

```
// Declare rz grid of complex pressure
// 6 cm in r, 10 cm in z, .1 mm point spacing
        const int r_length=600;
        const int z_length=1000;
        complex rz_array;

//*************************************************************************
//
//      Program Main
//
//*************************************************************************

main(){

        //complex pressure_at_point; // Summed pressure at rz grid
        //char rz_output_file[20];// name of output file
        ofstream to_rz_file;  // establish output stream
  clock_t start, finish;
  double  duration;
  struct tm *newtime;
        time_t aclock;
        start = clock();

// New constants and variables!!

        const float pi=3.141592654;
        const float speed_of_sound=1500;// in m/s
        const float surface_velocity=0.1;// m/s
        const float density= 1026; // Kg/m3
        // Physical and spatial variables
        // f=2 MHz, r=4 mm
        float delta_z,delta_r;
        float freq  = 2000000.0,radius=.002;
        float omega=float(2.0*pi*freq); //radiant freq
  float A, B=0.486; //gaussian factor for 2 < D/T < 4
  float sigma,ro,chi; //non-dimensional factors
  float pressure_magnitude;
  float gamma,theta;

        cout << "Welcome to the Single Source G_Beam Construstor" << endl;
        // set up output stream
        to_rz_file.open("gauss.dat",ios::out);

//      //Construct the single rz array

  ro=float(pow(radius,2)*omega/2.0/speed_of_sound);

        // start filling pressure grid
        for(int r=0;r<600;r++){// r from axis out to 6 cm

                for(int z=500;z<1500;z++){

      delta_r=0.0001;
      delta_z=0.0001;
```

```
                sigma=z*delta_z/ro;
                A=float(B/(1.0+pow(B*sigma,2)));
                chi=r*delta_r/radius;
                pressure_magnitude=float(density*speed_of_sound*surface_velocity
                                        /sqrt(1.0+pow(B*sigma,2))*exp(-A*pow(chi,2)));

                                gamma=float((pow(B,2)*sigma/(1+pow(B*sigma,2)))*pow(chi,2)
                                        -atan(B*sigma)+pi/2);
        theta=-(omega/speed_of_sound*z*delta_z)+gamma;
                                rz_array.rp=pressure_magnitude*
                                                cos(theta);
                                rz_array.ip=pressure_magnitude*
                                                sin(theta);


                                to_rz_file  << z << "  " << r << "  "
                                << rz_array.rp << "   "
                                << rz_array.ip << endl;

                } // end for grid z

// this index indicates the current r position!!

        cout << r << endl;

        } // end for grid r


        to_rz_file.close();//close streams to output files

   //output process time
   finish = clock();
   duration = (double)(finish - start) / CLOCKS_PER_SEC/60;
   cout << endl <<endl;
   printf( "Total process time = %2.1f min\n", duration  );
   time( &aclock );              /* Get time in seconds */
   newtime = localtime( &aclock );  /* Convert time to struct tm form  */
   /* Print local time as a string */
   printf( "Process end at: %s", asctime( newtime ) );

        return 0;

}// end main

//*************************************************************************
//
//      END SIMULATION
//
//*************************************************************************
```

APPENDIX D.

MATLAB FUNCTIONS

D.1 Extrapolation test function

This function tests the extrapolation scheme used in the RSAS. The only input the function needs is the coordinate of the point where the magnitude of pressure is desired. The result of this implementation is discussed in Section 2.3.4.

```
function extrapol (r,z)

%*******************************
% Set intensity grid %
%*******************************
I1=-1.0;
I2=1.0;
I3=3.0;
I4=5.0;

%*******************************
% Set position grid
%*******************************
D=1;
z1=1;
r1=1;

%*******************************
% extrapolation for sides of the cell
%*******************************
m1=(I1-I2)/D;
m2=(I3-I4)/D;
Ia=(I1-m1*z1)+m1*z;
Ib=(I3-m2*z1)+m2*z;

%*******************************
% extrapolation of intensity
%*******************************
m3=(Ia-Ib)/(-D);
I=(Ia-m3*r1)+m3*r
```

D.2 Damping cap position randomizer

The function below will randomize the (r,θ) position for the opening in the damping cap on the circular piston source surface in an attempt to disrupt the regular hexagonal pattern. The output file contains the new element positions to be used by the RSAS.

```
function cap(Rl,Rs)
format short

%*********************************************
% enter R(large) and R(small)  to generate random position
% plots using tempcord which is xyzcord w/o
% heading.
%*********************************************
load tempcord

%*********************************************
% calculate randomized xyz coord.
%*********************************************
theta=2*pi*rand(108);
Rs=Rs*rand(108);

for i=1:108,
   x1(i)=cos(theta(i))*Rs(i);
   y1(i)=sin(theta(i))*Rs(i);
end

x0=tempcord(:,2);
y0=tempcord(:,3);

for i=1:108,
   new_x(i)=(x0(i)+x1(i));
   new_y(i)=(y0(i)+y1(i));
   new_z(i)=(sqrt((102.26)^2-((new_x(i))^2+(new_y(i))^2)));
   index(i)=(i);
end
```

```
%*********************************************
% write to file
%*********************************************
newcord(:,1)=(index');
newcord(:,2)=(new_x');
newcord(:,3)=(new_y');
newcord(:,4)=(new_z');
save newcord2. newcord -ascii

%*********************************************
% output visual to screen
%*********************************************
angle=0.0001:.05:2*pi;
[a,b]=size(angle);

for j=1:16,
   for i=1:b,
      xlc(i)=cos(angle(i))*(Rl)+x0(j);
      ylc(i)=sin(angle(i))*(Rl)+y0(j);
      xsc(i)=cos(angle(i))*(2)+new_x(j);
      ysc(i)=sin(angle(i))*(2)+new_y(j);
   end
   subplot(4,4,j),plot(xlc,ylc)
   hold on
   plot(xsc,ysc)
   hold off
   title(int2str(j))
   axis equal
   axis off
end
```

D.3 Radius of curvature caculation

This Matlab function calculates the points on the curve surface of a focused transducer given radius of curvature and element radius. A surface plot is generated as output to illustrate the transducer surface.

```
function curve(R,f,a)

%************************************
%       R = Radius of Curvature(mm)
%       f = Frequency
%       a = radius of the element(mm)
%       R^2 must be > (a^2 + a^2)
%************************************

%*****  Known Variable  *****
c=1500000     %mm/s
b=(2*R)
lambda=c/f;

%*****  calculate quadratic equation  *****
for y=a:-1:-(a);
        for x=-(a):1:a;
                c=x^2+y^2;
                y1=abs(y-(a+1));
                x1=abs(x+(a+1));
                z(y1,x1)=(-b+sqrt(b^2-4*c))/2;
        end
end

%*****  Surface plot of the curved surface  *****
x=-a:1:a
y=a:-1:-a
mesh(x,y,z)
```

APPENDIX E.

GAUSSIAN BEAM DERIVATION

This appendix contains the derivation of the Gaussian beam profile for a single transducer element. A simple definition of variables in the derivation is included in the Table E.1. If More detailed definition is desired, they can be found in the paper by Du and Breazeale [17].

Table E.1 Derivation parameter listing.

Parameters	Definition
a	radius of the back electrode
f	driving frequency
ρ_o	static density of the medium
ω	radiant frequency
ρ	radial coordinate
σ	normalized axial distance form the source
ζ	normalized transverse distance from axis
B	Gaussian coefficient of source
A	Gaussian coefficient of sound field
τ	retarded time
P_o	sound pressure amplitude at center of transducer
$\overline{P}$	normalized pressure
r_o	Rayleigh distance
α	attenuation constant
c_o	speed of sound in medium
U_o	Surface Velocity

Du and Breazeale [17] obtained a relatively accurate description of the Gaussian beam from the basic linear wave equation obtained by Ivar and Barkve [18] (see below)

$$\left(4\frac{\partial^2}{\partial\tau\partial\sigma} - \overline{\nabla}_\perp^2 - 4\alpha r_o \frac{\partial^3}{\partial\tau^3}\right)\overline{p} = 0 \qquad \text{(E.1)}$$

The linearized solution for an axially symmetric source which oscillates sinusoidally in time is

$$\bar{p}(\xi,\sigma,\tau)= \mathrm{Re}\left[iq_1(\xi,\sigma)\exp(-i\tau-\alpha r_o\sigma)\right] \tag{E.2}$$

where

$$q(\xi,\sigma)=\frac{2}{i\sigma}\int_0^\infty \exp\left(i\frac{\xi^2+\xi'^2}{\sigma}\right)J_o\left(\frac{2\xi\xi'}{\sigma}\right)q_1(\xi')\xi' d\xi' \tag{E.3}$$

The boundary condition at the source (σ=0) is

$$\bar{p}(\xi,0,\tau)= \bar{q}_1(\xi)\exp(-i\tau) \tag{E.4}$$

The Gaussian amplitude distribution at the source (σ=0) is

$$\bar{q}(\xi')= \exp(-B\xi'^2) \tag{E.5}$$

where B is the Gaussian coefficient which is related to the thickness and the radius of the back electrode, and can be determined experimentally. Next, Equation (E.5) is substituted into Equation (E.3) to yield the following equation.

$$q(\xi,\sigma)=\frac{2}{i\sigma}\int_0^\infty J_o\left(\frac{2\xi\xi'}{\sigma}\right)\exp\left(\left(\frac{i}{\sigma}-B\right)\xi'^2+i\frac{\xi^2}{\sigma}\right)\xi' d\xi' \tag{E.6}$$

which can be integrated to give

$$q(\xi,\sigma)= \frac{1}{\sqrt{1+(B\sigma)^2}}\exp\left(-\frac{B}{1+(B\sigma)^2}\xi^2\right)\exp(i\gamma) \tag{E.7}$$

where the phase shift is

$$\gamma=\left\{\left(\frac{B^2\sigma}{1+(B\sigma)^2}\right)\xi^2-\tan^{-1}(B\sigma)+\frac{\pi}{2}\right\} \tag{E.8}$$

Then, by inserting Equation (E.7) into Equation (E.2), the amplitude of the sound field produced by a transducer with a Gaussian velocity distribution is described by

$$P_A(\xi,\sigma)= P_o \frac{\exp(-\alpha r_o \sigma)}{\sqrt{1+(B\sigma)^2}} \exp(-A\xi^2) \qquad \text{(E.9)}$$

where $A = \frac{B}{1+(B\sigma)^2}$ is the Gaussian coefficient of the sound field and $P_o = \rho_o c_o U_o$ is the sound-pressure amplitude in the fluid in the center of the transducer.

REFERENCES

[1] D. J. Coleman, F.L. Lizzi, J. Driller, A. L. Rosado, S. Chang, T. Iwamoto, and D. Rosenthal, "Therapeutic ultrasound in the treatment of glaucoma," *Ophthalmology*, vol. 92, 339-346 (1985).

[2] R. S. Foster, R. Bihrle, N. T. Sanghvi, F. J. Fry, and J. P. Donohue, "High intensity focused ultrasound in the treatment of prostitic disease," *Eur. Urol.*, vol. 23, 29-33 (1993).

[3] G. R. ter Harr, I. Riven, L. Chen, and S. Riddler, "High intensity focused ultrasound for treatment of rat tumors," *Phys. Med. Biol.* vol. 36, 495-501 (1991).

[4] Goss, S. A. and Fry, F. J., "High Intensity Ultrasonic Treatment of Tumors," *IEEE Trans. Sonic and Ultrasound.*, vol. 31,491-496 (1984).

[5] G. R. ter Harr, "Ultrasound Focal Beam Surgery," Ultrasound Med. Biol., vol 21, no. 9, 1089-1100 (1995).

[6] F. J. Fry, "Intense focused ultrasound: Its production, effects, and utilization," *Ultrasound: Its Application in Medicine and Biology,* Part II, F. J. Fry, Ed., New York, Elsevier, 689-736 (1978).

[7] E. S. Ebbini and C. A.Cain, "Experiment evaluation of a prototype cylindrical section ultrasound hyperthermia phased-array applicator," *IEEE Trans. Ultrasound., Ferroelectr., and Freq. Control*, vol. 38, 510-520 (1991).

[8] S. A. Goss, "Stationary and phased arrays for ultrasound hyperthermia applicators," *Hyperthermic Oncology 1988*, vol. 2, T. Sugahara, and M. Saito, Eds., New York, Taylor and Francis, 670-673 (1988).

[9] L. E. Kinsler, A. R. Frey, C. Sanders, and J. V. Sanders, *Fundamentals of Acoustics*, New York, John Wiley and Sons (1982).

[10] B. D. Steinberg, *Principles of Aperture and Array System Design*, New York, John Wiley and Sons (1976).

[11] S. A. Goss, L. A. Frizzell, J. T. Kouzmanoff, J. M. Barich, and J. M. Yang, "Sparse random ultrasound phased array for focal surgery," *IEEE Trans. on Ultrasound, Ferroelectrics and Frequency Control*, special issue, in press (1996).

[12] J. M. Barich, "Theoretical analysis of an ultrasonic sparse random phased array surgical applicator," M.S. thesis, Univ. of Illinois at Urbana-Champaign, 1996.

[13] G. R. Lockwood and F. S. Foster, "Optimizing sparse two-dimensional transducer arrays using an effective aperture approach," *1994 Ultrasonic Symposium*, 1497-1501 (1994).

[14] P. K. Weber, R. M. Schmitt, B. D. Tylkowski, and J. Steck, "Optimization of random sparse 2-D transducer arrays for 3-D electronic beam steering and focusing," *1994 Ultrasonic Symposium,* 1503-1505 (1994).

[15] J. O. Erstad and S. Holm, "An approach to the design of sparse array systems." *1994 Ultrasonic Symposium*, 1507-1510 (1994).

[16] M. A. Breazeale, F. D. Martin, and B. Blackburn, "Reply to radiation pattern of partially electroded piezoelectric transducers," *J. Acoust. Soc. Am.* vol. 70(6), 1791-1793 (1981).

[17] Gonghuan Du and M. A. Breazeale, "The ultrasonic field of a gaussian transducer," *J. Acoust. Soc. Am.* vol. 78, 2083-2086 (1993).

[18] S. Ivar and T. Barkve, "Distortion and harmonic generation in the nearfield of a finite amplitude sound beam," *J. Acoust. Soc. Am.* vol. 75(3), 749-753 (1984).