ARMY RESEARCH LABORATORY

Ultra Wide Bandwidth Synthetic Aperture Radar Focusing of Dispersive Targets

by John McCorkle and Lam Nguyen

HDL Report
R-HD-ST-R-92-004 (Reprinted in March 2010)

March 1992

Army Research Laboratory
Adelphi, MD 20783-1197

HDL Report **March 1992**
R-HD-ST-R-92-004 (Reprinted in March 2010)

Ultra Wide Bandwidth Synthetic Aperture Radar Focusing of Dispersive Targets

by John McCorkle and Lam Nguyen

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* March 1992	2. REPORT TYPE	3. DATES COVERED (From - To)

4. TITLE AND SUBTITLE Ultra Wide Bandwidth Synthetic Aperture Radar Focusing of Dispersive Targets	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) John McCorkle and Lam Nguyen	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory Attn: AMSRL-SS-SG 2800 Powder Mill Road Adelphi, MD 20783-1197	8. PERFORMING ORGANIZATION REPORT NUMBER HDL Report R-HD-ST-R-02-004 (Reprinted in March 2010)
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory 2800 Powder Mill Road Adelphi, MD 20783-1197	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report addresses focusing of synthetic aperture radar (SAR) data. More specifically, it addresses the SAR focusing problem with special attention to focusing an area in the near-field of the synthetic aperture over a decade or more of bandwidth in a manner that preserves target resonance characteristics. A method for solving this image formation problem is described, along with a computationally efficient algorithm that is applicable to real-time processing with motion compensation. Both simplified program examples are given, as well as a complete program listing that executes on a several single-chip array processors simultaneously. An error analysis shows quantitatively when the depth of focus is adequate to preserve long-duration target resonance ringing effects for a given target Q, geometry, and bandwidth.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	86	Lam Nguyen
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER *(Include area code)* (301) 394-0847

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

CONTENTS

FIGURES

TABLES

1. Introduction

The process of obtaining images of the reflectivity or density of target areas that are rotating and translating with respect to a sensor such as a mono-static or bistatic radar, a sonar, or an x-ray CAT scanner, has been studied for the past 40 years[1,2,3,4]. Ausherman etal.[5] have written an excellent review of the work done in this area. Target areas that rotate and translate relative to a radar include, for example, planet surfaces observed from a satellite, ground terrain observed from air-borne platforms, sub-surface objects and voids observed from moving vehicles, and people scanned by a rotating x-ray system. Although the processing described is applicable to other systems, this article shall treat the topic from the point of view of a SAR (Synthetic Aperture Radar). As the aspect angle between the sensor and a target changes with time, the sensor collects a sequence of signal records. After collecting data for T_s seconds, the aspect angle has changed by θ_s degrees. These received signals are then coherently processed to obtain the reflectivity profile of the target area. Down-range resolution into range-bins is obtained primarily by the bandwidth of the sensor. Cross-range resolution is obtained primarily by coherently processing the received signals such that a very wide aperture is simulated; an aperture that is θ_s degrees wide.

Implementation and study of this image formation processing topic has been limited in two ways. First, the image formation processing has assumed that targets are isotropic point scatterers. Many targets, however, are anisotropic resonant scatterers. Treatment of this resonant case becomes important when the sensor spectrum covers the Rayleigh, resonant, and optical regions of a family of targets. For example, discrimination between scatterers can be based on the unique signature of each target. But, in order for the discrimination to work in the context of microwave reflectivity imaging, the information in the signature must be preserved during the image formation process.

---

[1] *William M. Brown and Ronald J. Fredricks, "Range-Doppler Imaging with Motion through Resolution Cells", IEEE Trans. Aerosp. Electron. Syst., AES-5 No.1, (Jan 1969), 98-102*

[2] *J.L. Bauck and W.K. Jenkins, "Tomographic Processing Of Spotlight-Mode Synthetic Aperture Radar Signals With Compensation For Wavefront Curvature", IEEE Interna. Conf. on Acoust. Speech and Sig. Proc. (Apr.11-14 1988)*

[3] *Edwin D. Banta, "Limitations on SAR Image Area Due to Motion Through Resolution Cells", Correspondence, IEEE Trans. Aerosp. Electron. Syst., AES-22, No. 6 (Nov 1986), 799-803*

[4] *D. Mensa, and G. Heidbreder, "Bistatic Synthetic-Aperture Radar Imaging of Rotating Objects", IEEE Trans. Aerosp. Electron. Syst., AES-18, No. 4 (July 1982), 423-431*

[5] *Dale A. Ausherman, Adam Kozma, Jack L. Walker, Harrison M. Jones, and Enrico C. Poggio, "Developments in Radar Imaging" IEEE Trans. Aerosp. Electron. Syst. AES-20, No.5 (July 1984), 363-400*

The second limitation in previous work is that relatively narrow bandwidth systems have been assumed. For example, the compressed pulse width of the sensor is assumed to be at least several, and usually many cycles of an RF carrier frequency. So a single range-bin is derived from several cycles of the carrier. Newer UWB (Ultra Wide Bandwidth) sensors, however, have made it possible to make the image range-bin size roughly 1/2 the wavelength of the highest frequency in the sensors' spectrum. The bin size is much smaller (1/10 or less) than the wavelength of the lowest frequency in the spectrum. This wide bandwidth exacerbates range walk and wavefront curvature errors to the point where conventional FFT based processing must be restricted to very small patches within an image area.

These two limitations are not independent. It is bandwidth that allows target discrimination based on signature analysis and it is bandwidth that makes image formation more difficult. It is the purpose of this paper to examine image formation processing that preserves resonant target signatures and to present an efficient method of solving the microwave reflectivity imaging problem for UWB signals and resonant targets.

2. Background

2.1. Terminology

SAR systems depend upon collecting data coherently along a path. This path is referred to as the "synthetic aperture." Figure 1 shows a sketch of the scenario projected onto a plane. An aircraft flies at some elevation h above the ground, over a distance L_s to form the synthetic aperture. The image area grid is referenced to the center of the aperture. Range is marked off by the parameter i. The parameter k marks off azimuthal (bearing) lines. The sample points in the aperture are marked off by the parameter j. The distance between the j^{th} point in the aperture and the $(i,k)^{th}$ position in the image area is denoted by $d_{i,j,k}$. Although nearly all operational SAR systems use a rectangular grid, there are advantages to the polar grid that will be discussed later in this report. Both the azimuthal lines and the range lines are referenced to the center of the aperture.[6] Table 1 summarizes the nomenclature to be used in the rest of the report.

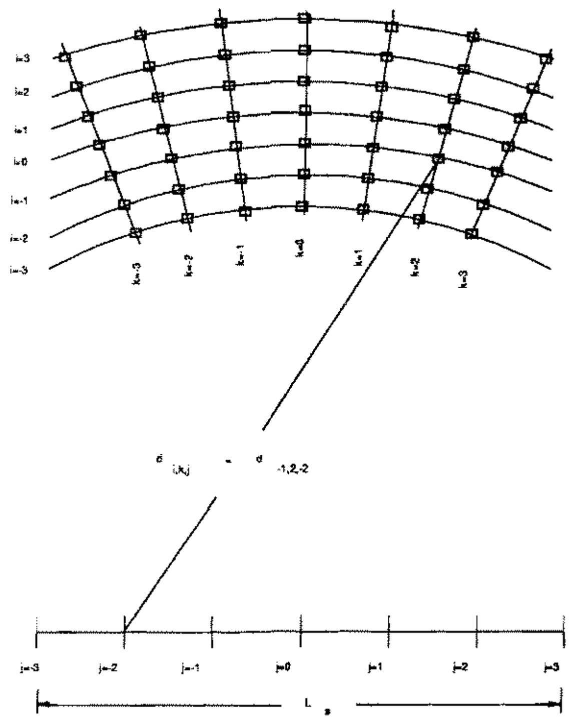


Figure 1. Basic Flight Path and Image Area Geometry

[6]*This grid is convenient for post processing. An X-Y grid can also be computed using the techniques described.*

Table 1. Analysis Nomenclature

L_s	Length (meters) of the synthetic aperture.
$s_j(t)$	Denotes a received signal amplitude (volts) as a function of time (seconds from the leading edge of the transmitted pulse), at the jth position in the aperture. The A/D outputs represent this signal.
Over type hat e.g. $\hat{s}(t)$	Denotes that the function is an estimate. The example denotes an estimate for a received signal.
Subscript of i,k,j	Particularizes the function to be at particular geometry's, like the jth position in the synthetic aperture, or the kth bearing line, or the (i,k)th position in the image area,
$d_{i,j,k}$	Denotes the distance (meters) between the jth position of the radar, and the (i,k)th position in the area to be imaged.
$t_{i,k,j} = \frac{2d_{i,k,j}}{c}$	Denotes the round trip time (seconds) for the radio energy to travel from the jth position in the synthetic aperture, to the (i,k)th position in the image area, and back.
c	The speed of light.
$\alpha = t_{i+1,k,j} - t_{i,k,j}$	Time shift between range bin i and range bin i+1 at the center of the aperture (where j=0).
PRF	Pulse-Repetition-Frequency of the radar in Hertz.
μ	Relative Bandwidth $\mu = (F_{hi} - F_{lo}) / F_0$ and $F_0 = (F_{hi} + F_{lo}) / 2$
UWB	Ultra-Wide Bandwidth, where $\mu \approx 1$
λ	Wavelength in meters.

2.2. Approaches to Focusing

SAR processing, or SAR focusing, is sometimes referred to as a Doppler based process. Why is this "Doppler" paradigm used? Referring to Figure 1, assume that the radar platform is moving at velocity v as it collects data along the aperture. Therefore, the data collection points are spaced equally in time, occurring at a rate governed v and the PRF. (This sampling across the aperture is sometimes referred to as "slow-time".) If the radar is operating at a frequency f_0, then the echo from a target in the image area will have a Doppler shift profile as the platform moves through the aperture. It will have an upward shift while the platform approaches, the shift will drop to zero as the platform moves to a position broadside to the target, and the shift will be downward as the platform recedes. Every target position will have a unique Doppler profile. Since every position in the image has a distinct Doppler profile, an image can be formed by simply assigning to each pixel a filter matched to the Doppler profile expected for a target at the location represented by that pixel.

The classic "Polar Formatting"[7] approach to computing a SAR image simply adjusts (time shifts) the data at each aperture point so that the new data-set appears as if the platform had moved in a short circular path - with the circle centered at the center of the image area. Once this formatting is done, a target at the center point has the unique Doppler profile of zero over the entire aperture. Other points have other unique Doppler profiles. A Fourier transform is typically used to recover the image from the Doppler profiles. The Fourier approach works well as long as the circular path is sufficiently short, the image area sufficiently small, the signal bandwidth sufficiently small, and the distance from the radar to the image center sufficiently long. This paper addresses the case where none of these restrictions apply.

Although the Doppler paradigm has helped countless people to visualize SAR focusing, it is not necessary, nor always helpful, in solving the SAR focusing problem. For example, Doppler shift, which is defined as $2v/\lambda$, works fine when λ varies by a few percent, but it becomes a stumbling block in the UWB case where λ can vary by 10 or 100 to 1. Doppler has proven to be a very convenient narrow band concept. But its convenience breaks down at wide relative bandwidths. The focusing problem can also, however, be looked at as a stationary array of N antennas whose outputs are digitally stored and combined in a computer to form beams. It is the author's view that the focusing problem is easier to visualize and solve using this stationary array paradigm for the case where neither the geometry nor the bandwidth are restricted.

Using the stationary array paradigm one can see that equation 1 coherently focuses the SAR data by summing across the array. It may be helpful to point out that both approaches can be seen to be identical at the center point of the "Polar Format" patch. Notice that regardless of carrier frequency, the center point of the polar formatted data is always analyzed by the DC term of the Fourier transform. And the DC term of a Fourier transform is simply a summation of the data points. The summation shown in equation 1 is identical to finding the DC term of the polar formatted SAR data. But instead of formatting once and then finding many "Doppler" profiles via an FFT, equation 1 formats and "sums" the data many times; each formatting makes a different pixel the center and then the DC term summation is calculated to get the value for that pixel. The result is that optimally focused beams are formed - beams taking full advantage of both the entire aperture and the entire signal bandwidth. The focusing is truly frequency-independent.

[7] *J. L. Walker, "Range-Doppler imaging of rotating objects," IEEE Trans. Aerosp. Electron. Syst., AES-16 (Jan. 1980), 23-52*

How does the beam width compare with "conventional" (far-field narrow band) antenna theory? The rule-of-thumb half-power beamwidth of a line array is approximately λ/L where L is the length of the array. The beams formed by the processing described above follow this rule and have a width proportional to λ. Low frequencies have wide beams and high frequencies have narrow beams. An interesting aspect of this fact is how it manifests itself in the time-domain as a source moves through a beam. The impulse response at the center of the beam is a nice narrow pulse. But as one moves away from the center of the beam, the impulse response gets broader and broader. This time-domain broadening occurs because more and more high frequency energy is lost as the source moves out of the narrowing high-frequency beam.

As soon as one switches from the Doppler to the stationary array approach, it may be tempting to also switch to thinking in "phased array" terms. To do so is a mistake when bandwidth and/or geometry are not restricted. Whenever the geometry is not restricted to the far-field of an aperture, plane-wave simplifications break down. This break down invalidates simple phase steering. Since a Fourier transform forms beams by simple phase steering, Fourier techniques becomes less and less useful as targets move into the near-field. In an ultra wide bandwidth system, phase becomes meaningless with regard to defining element positions or the beam forming network. To speak of shifting one element 180 degrees with respect to another element implies a fixed λ. If, for example, λ changed by 2 to 1, then a delay-line that provided 180 degrees at one frequency would provide 360 degrees at the other. Yet delay-lines are precisely the element needed to build a wide bandwidth "phased array" antenna. When λ varies several octaves, the best parameters to use in the equations defining the antenna is the time and distance - time-shift of the delay lines that form the beam-forming network, and distance between antenna elements. So it is best if one switches to thinking in "timed array" terms. This time-based framework results in derivations that are frequency and geometry independent.

2.3. Null Steering and Pattern Forming

Generally, an antenna designer would say that the most critically important aspect of making desirable antenna patterns is forming nulls. A simple classic case is spacing 2 elements at 90^o and phasing them by 90^o to form an endfire beam in one direction and a null in the opposite direction. In the simplest case (using element weighting of +1 and -1), to form a UWB null one must time-steer the array to where the null should be, and then invert half the elements prior to summing. Of course, using other weighting factors allows more freedom. An impulse signal $\delta(t)$ coming from a direction other than the null would produce in the receiver some array-induced waveform. Ignoring bandwidth effects from each element, that waveform would be a function of the + and - weighting and element spacing. And by definition, that waveform is the antenna-array impulse response for that beam angle. Matched filtering to that waveform forms a mainlobe in that direction.

Grating lobes are customarily defined as lobes whose gain is equal to the mainbeam. It is interesting to note that this definition leads to confusion in the UWB case. Are there grating lobes? Well, that depends. If one looks at the response to the UWB signal, like an impulse, then the answer is no. The peak response on the mainlobe is higher than the response at any other angle. So the definition fails. On the other hand, however, the antenna is a linear system. It behaves just like an identical array operating at a single frequency. So, if one looks at the response to a CW signal, the answer can be yes. If the elements are physically spaced greater than $\lambda/2$ apart at the highest frequency component in the UWB waveform, then yes, there certainly will be a grating lobe at that frequency component. All the insight gained from CW antenna analysis holds and remains useful. One must, however, be careful when applying terms like grating-lobes that presuppose a narrow band signal. In the case of SAR, the element spacing (Velocity/PRF) needed will be a function of what sidelobes are permissible at the various frequency components in the UWB waveform.

2.4. Target Resonance Effects

Historically, the relative bandwidth of radars has been sufficiently small that a target's echo is adequately modeled by a single number, [sigma], the Radar Cross Section (RCS); usually given in square meters. When $\mu \geq .5$, however, a single number may no longer adequate -- σ is a function of frequency. For example, Figure 2 is a plot of the RCS of a sphere. In addition to the magnitude characteristic plotted, there is also a phase characteristic. These two frequency-domain characteristics can also be represented in the time domain by a ringing or resonant response. In either case - time domain or frequency domain - the plots can be referred to as the impulse response of the target.

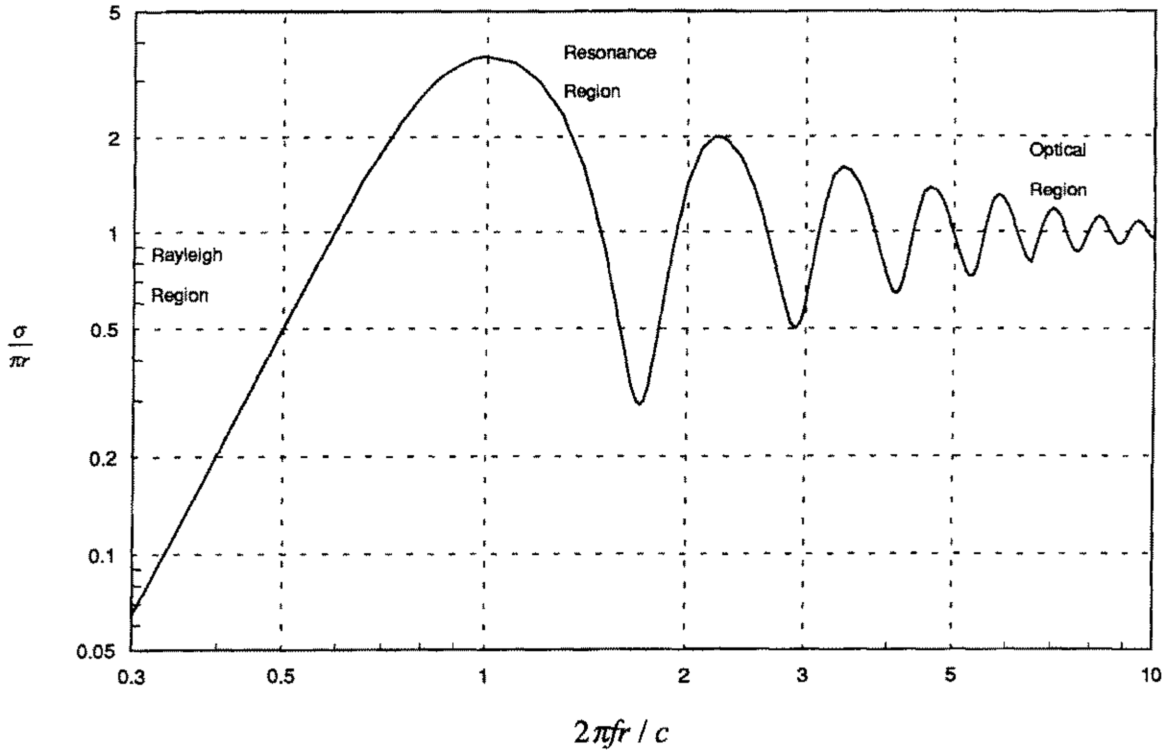


Figure 2. Radar Cross Section of a Sphere

Since historical SAR systems have not been interested in target ringing, no attention was paid on how to preserve the ringing information. This paper describes and analyzes a procedure to focus a large array, over ultra-wide bandwidths, in such a way as to preserve the resonant response of targets; even when they are in the near-field.

3. Fundamental Time Based, Physical Array Approach

Consider an aperture looking at completely empty space except for an isotropic scatterer at position (i,k). Also assume that an ideal impulse $\delta(t)$ is broadcast. It is desirable to take advantage of all the echo energy across the aperture. To do that, the energy from all collection points along the synthetic aperture is summed. However, this summation must be done such that the energy adds coherently at all frequencies. Frequency independent adding is accomplished as,

$$f_{i,k}(t) = \sum_j s_j(T_{i,k,j} + t) \text{ for } t \le 0 \tag{1}$$

Here, $f_{i,k}(t)$ would be the impulse response of the target located at position (i,k). Note that the $T_{i,k,j}$ term time shifts the received signals s_j such that the target impulse response starts at t=0 -- at all points in the aperture.

The above discussion assumes that a target's impulse response is sufficiently similar at all points along the aperture to consider equation 1 true. The response of a vertical dipole, for example, does not change depending on where the radar is positioned in the aperture. It is, therefore, an isotropic target.

Suppose we relax the isotropic requirement, and suppose that a complex scatterer is at position (i,k) in the image area. A horizontal dipole, for example, is an anisotropic target. Advantage could be taken of the anisotropic behavior to extend the detection and target recognition performance of the radar by rewriting equation 1 as

$$f_{i,k}(t) = \sum_j [x_j \otimes s_j](T_{i,k,j} + t) \text{ for } t \le 0, \tag{2}$$

where the convolution step represents filtering. In this case, a set of filters $X_j(\omega)$ would be needed, each one matched to the target response at the bearing of the j^{th} position in the aperture. In order to simplify the rest of this report, equation 1 will be used as the fundamental equation. Nonetheless, one must recognize that real targets are complex anisotropic polarimetric scatterers. This fact should not be ignored for large arrays.

4. Short Impulse Response Approximation

A typical 2-D SAR image is simply the echo magnitude mapped to intensity. Equation 1 expands the typical 2-D image into a 3-D image with the target ringing along the third dimension. Given the heavy computation load of typical SAR processing, if it is required that an $f(t)$ be computed for every pixel instead of a single value, then a massive computer would be needed. The impact of this computational load is even worse when one considers that the mass of resulting data must be analyzed in the target detection/identification phase. This section defines an approximation which reduces the problem back to a 2-D case and presents an error analysis of the approximation.

4.1. Theory

The problem is that a third dimension has been added to measure target ringing. Note, however, that if there was only one aperture position, then ringing of the target would just appear in pixels behind the target. Even with an aperture of many positions, ringing will appear behind the target. But since the geometry is not constrained, and near-field operation is presumed, the antenna beam defocuses behind the target. Thus (1) will perfectly focus ringing of unbounded duration. But for practical purposes, the target only rings for a finite duration, say M range-bins. The question is, given that the ringing is finite in duration, can the ring information be retained in a 2-D image. In other words, suppose that instead of calculating a time series $f_{i,k}(t)$ for each pixel in the image, only one value is calculated, for example: $f_{i,k}(\tau)$ where τ is fixed for the image. If the target identification problem can be solved with this 2D image, then a great reduction in computational load is obtained. The aim of this section is to describe and quantify the error bounds when a 2-D image $f_{i,k}(\tau)$ is used.

First, define α to be the round-trip time it takes the radar pulse to traverse one range bin on the grid; that is,

$$\alpha = t_{i+1,k,j} - t_{i,k,j} : \forall k, j = 0 \qquad (3)$$

Since the grid for the image has been defined to be referenced to the center of the aperture (the j=0 position), the i parameter can be thought of as a quantized time t parameter. While t is in units of seconds, i is in units of range bins, and they are related by α -- one range bin equals α seconds. Now we can write

$$s_j(T_{i,k,j} + m\alpha) \equiv s_j(T_{i+m,k,j}) \text{ for } \begin{Bmatrix} \text{case 1: } j = 0, \forall m, \forall i, \forall k \\ \text{case 2: } m = 0, \forall j, \forall i, \forall k \end{Bmatrix} \qquad (4)$$

An approximation is made to equation 4, generalizing it to include all aperture points over a limited range of m to arrive at

$$s_j(T_{i,k,j}+m\alpha)\approx s_j(T_{i+m,k,j}) \text{ for } \forall i,\forall k,\forall j, m=-M\cdots M \tag{5}$$

The following may be said of the approximation in Equation 5.

1. It is perfect at the center of the aperture regardless of m.
2. It is perfect at m=0 regardless of j;
3. gets worse as m deviates further from zero;
4. and is worst at the end-points of the aperture.

A solution is desired to (1) in the form of $f_{i,k}(\tau)$ where τ is a constant. In a practical system τ will be mapped to discrete range bins, so let

$$\tau = n\alpha. \tag{6}$$

Substituting equations 5 and 6 into 1 we define a 2-D "image" $f(i,k)$ as

$$\begin{aligned} f_{i,k}(0) &= \sum_j s_j(T_{i,k,j}) \\ &\approx f_{i-n,k}(n\alpha) = \sum_j s_j(T_{i-n,k,j}+n\alpha) \equiv f(i,k) \end{aligned} \tag{7}$$

If one thinks of a ringing target, then equation 7 can be described as allowing the point of perfect focus to be adjusted to any depth n in the ring. For example, if n=0, then the perfect focus point would be at the leading edge (the first sample) of the ringing response. If $n \neq 0$, say n=3, then the third sample of the ringing response would be perfectly focused.

Next consider the indexing. The indexing is performed such that $f(i,k)$ always represents the leading edge of the response from a target located at (i,k) regardless of whether it is perfectly focused or not. Once $f(i,k)$ is found, we now wish to find the discrete samples of the target ringing. The samples will be counted as m=0 for the first sample, m=1 for the second and so on. These samples are obtained by simply incrementing the i index. The m^{th} value is just $f(i+m,k)$; which follows from equations 5 and 7 as

$$\begin{gathered} f_{i,k}(m\alpha) \approx f_{i-n+m,k}(n\alpha) = f(i+m,k) \text{ or,} \\ \sum_j s_j(T_{i,k,j}+m\alpha) \approx f_{i-n+m,k}(n\alpha) = f(i+m,k) \end{gathered} \tag{8}$$

Clearly equation 8 is identical to equation 1 (i.e. perfect) when m=n. So equation 8 gives perfect focus at m=n.. The name "short impulse response approximation" is used because the approximation only needs to remain accurate over the duration of a target's impulse response.

4.2. Examples

To illustrate the use of equation 8, we follow these steps:

1. Fix *n*;
2. Place a target (for the purposes of the illustration) at say *i*=237 in range on the k^{th} bearing.
3. Use equation 7 to calculate *f*(*i*,*k*) for all (*i*,*k*);

The cases of interest are where *n*=0 and where $n \neq 0$. We will consider each separately.

4.2.1. *Case where n=0*

Use equation 8 to find the ringing response of the target. The response for the first three points of the focused impulse response is:

$$f_{237,k}(0) = f_{237-n+m,k}(0) = f_{237,k}(0) = f(237,k) \ \{\text{case for } m=0\}$$
$$f_{237,k}(\alpha) \approx f_{237-n+m,k}(0) = f_{238,k}(0) = f(238,k) \ \{\text{case for } m=1\}$$
$$f_{237,k}(2\alpha) \approx f_{237-n+m,k}(0) = f_{239,k}(0) = f(239,k) \ \{\text{case for } m=2\}$$

Since *n*=0, the amplitude of the leading edge (*m*=0) of the impulse response of that target is perfectly focused. As *m* increases, the approximation gets worse. So the approximation is useful as long as the target resonance dies before the approximation gets too bad.

4.2.2. *Case where* $n \neq 0$

In this case, perfect focus is $n\alpha$ seconds past the leading edge of the target impulse response. Suppose, for this example, *n*=2. The first 4 data points for the impulse response is:

$$f_{237,k}(0) \approx f_{237-n+m,k}(n\alpha) = f_{235,k}(2\alpha) = f(237,k) \ \{\text{case for } m=0\}$$
$$f_{237,k}(\alpha) \approx f_{237-n+m,k}(n\alpha) = f_{236,k}(2\alpha) = f(238,k) \ \{\text{case for } m=1\}$$
$$f_{237,k}(2\alpha) = f_{237-n+m,k}(n\alpha) = f_{237,k}(2\alpha) = f(239,k) \ \{\text{case for } m=2\}$$
$$f_{237,k}(3\alpha) \approx f_{237-n+m,k}(n\alpha) = f_{238,k}(2\alpha) = f(240,k) \ \{\text{case for } m=3\}$$

Note that the leading edge is not perfectly focused, as it was when *n=0*. Incrementing. The important point here is that one can choose $n \neq 0$ to allow the leading edge to defocus slightly for the sake of keeping later points in better focus.

5. Error of Short Impulse Response Approximation

Landt, Miller, and Van Blaricum[8] show the transient echo response of a thin (hi-Q) dipole. Its ringing is damped after 5 cycles. This response is a good example to gain an intuitive idea of what kind of range is needed in the approximation. A 1/2 foot dipole should ring for 5 cycles at 1 GHz. If the A/D sampler collects data at 2 Gs/s, then at M=10, all of the 5 cycles will have been collected. If the dipole were 5 feet long, then it would ring for 5 cycles at 100 MHz. So M=100 to collect the 5 cycles. Generally, high frequencies damp quickly and low frequencies damp slowly.

Figure 3 illustrates the geometry of the approximation. The discussion will assume this geometry. If a target is located at *(i,k)=(237,0)*, then *R* is the distance from the center of the array to the target. *a1* is the distance from the end of the array to the target. $f_{237,0}(0)$ is the perfectly-focused data point for the impulse response of the target. The next point (m=1) in the impulse response is approximated by saying that $a2 \approx a1 + \alpha_d$, or in general $a2 \approx a1 + m\alpha_d$. If ε is the difference (error) between the approximation and the actual, then

$$\begin{aligned}\varepsilon &= \text{approximation - actual}\\ &= (\mathrm{a1} + \mathrm{m}\alpha_{\mathrm{d}}) - a2 \qquad (9)\\ &= \sqrt{R^2 + \frac{L_s}{2} - L_s R\cos(\theta)} + m\alpha_d - \sqrt{(R + m\alpha_d)^2 + \frac{L_s}{2} - L_s(R + m\alpha_d)\cos(\theta)}\end{aligned}$$

[8] *J.A. Landt, E.K. Miller, and M. Van Blaricum "WT-MBA/LLL1B: A COMPUTER PROGRAM FOR THE TIME-DOMAIN ELECTROMAGNETIC RESPONSE OF THIN-WIRE STRUCTURES" Lawrence Livermore Laboratory; May 6, 1974*

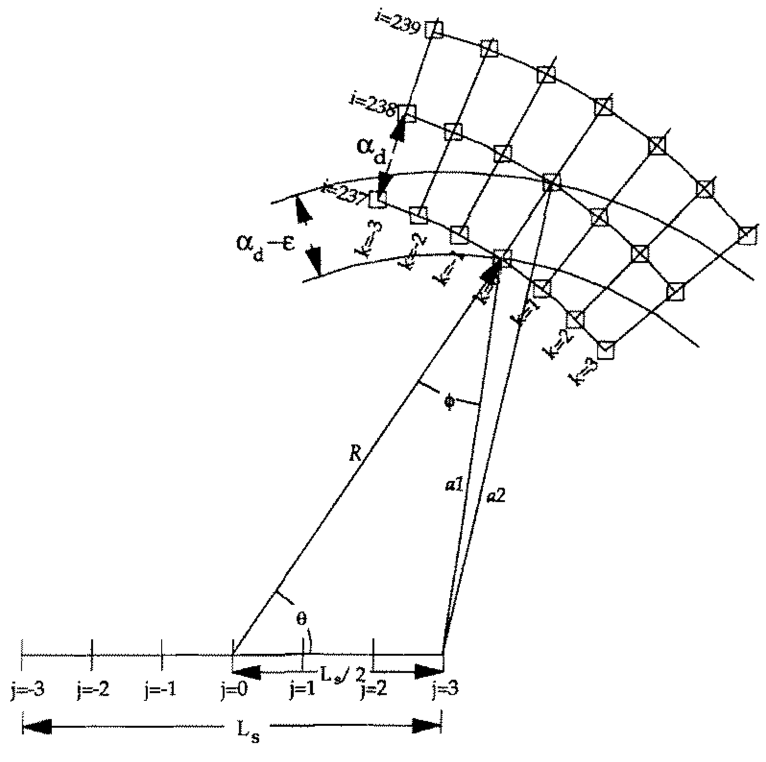
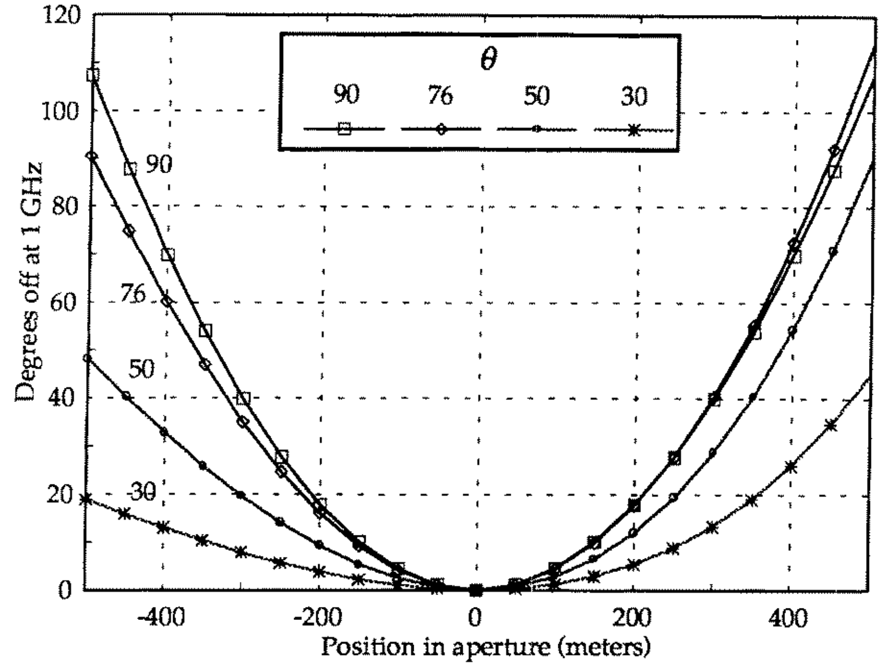


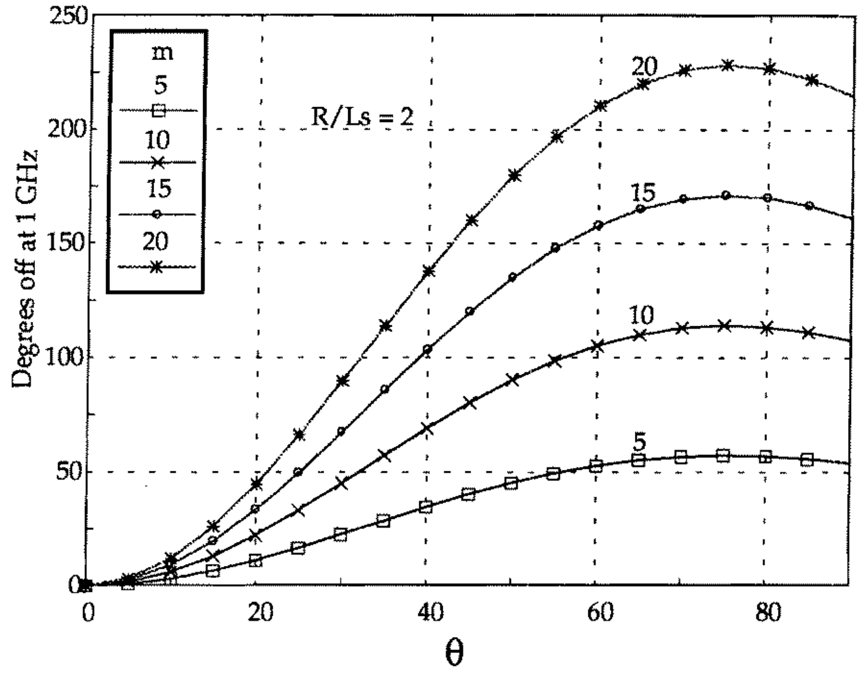
Figure 3. Geometry for calculation of approximation error

As an example, suppose R=2 Km, $\theta = 90°$, L_s=1 Km, and m=10. How many degrees off (round trip) would the end points be at 1 GHz? Plugging into equation 9, we find ε=.0447 meters. So the round trip phase error at 1 GHz is 107°. Figure 4 is a plot of the error as a function of the position in the aperture for θ=90,76,50, and 30 degrees. A peculiar characteristic is that the error peaks at 76°. This peaking is a result of working at a range of only twice the aperture length. Figures 5 and 6 show the error as a function of the beam angle θ, for m = 5, 10, 15, and 20 at two ranges.



Slant Range =2000
m=10
Sample Rate=2GHz
τ=0

Figure 4. Error as a function of position in aperture



Slant Range =2000
Sample Rate=2GHz
τ=0

Figure 5. Error as a function of θ with R/Ls=2

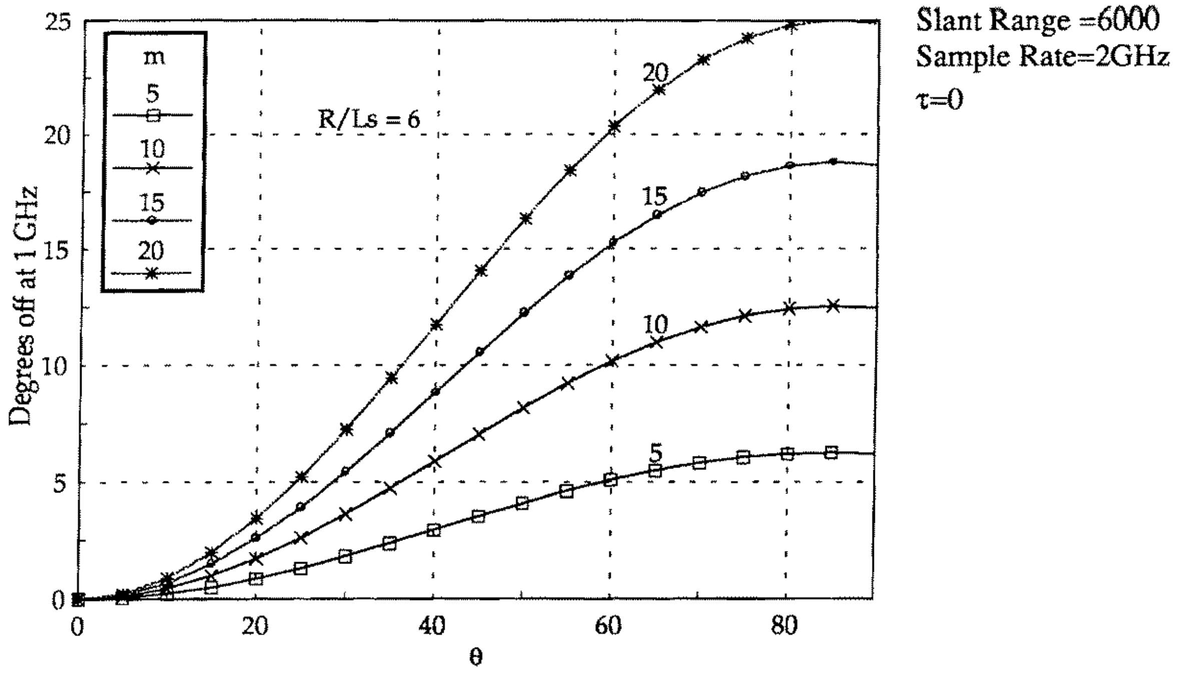


Figure 6. Error as a function of θ with R/Ls=6

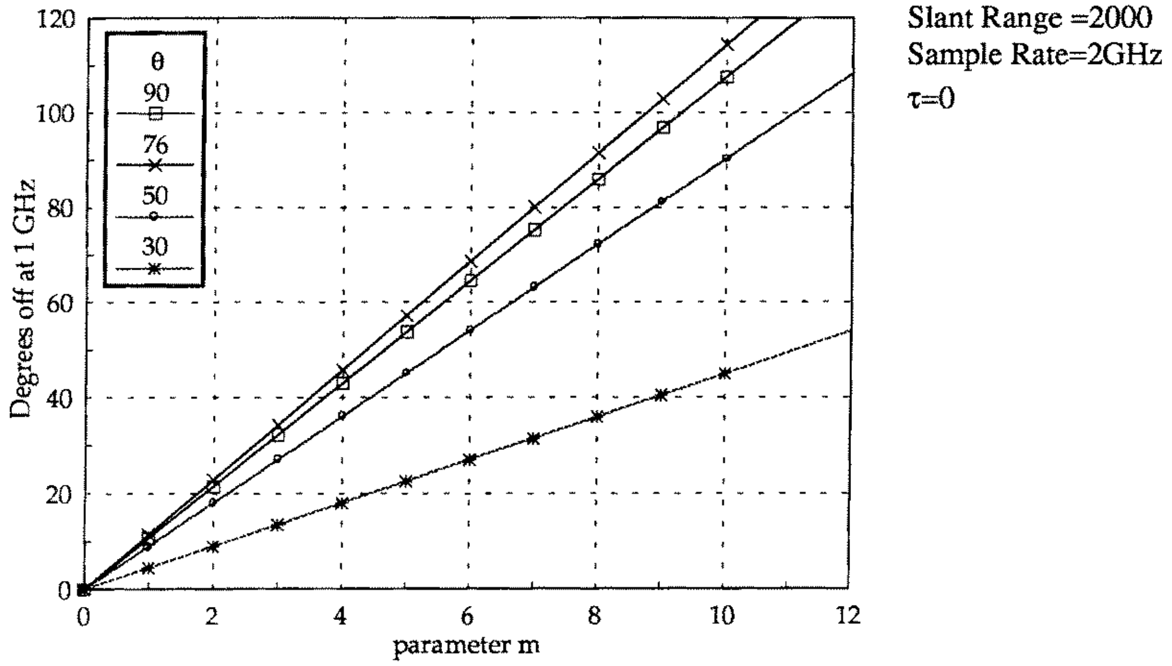
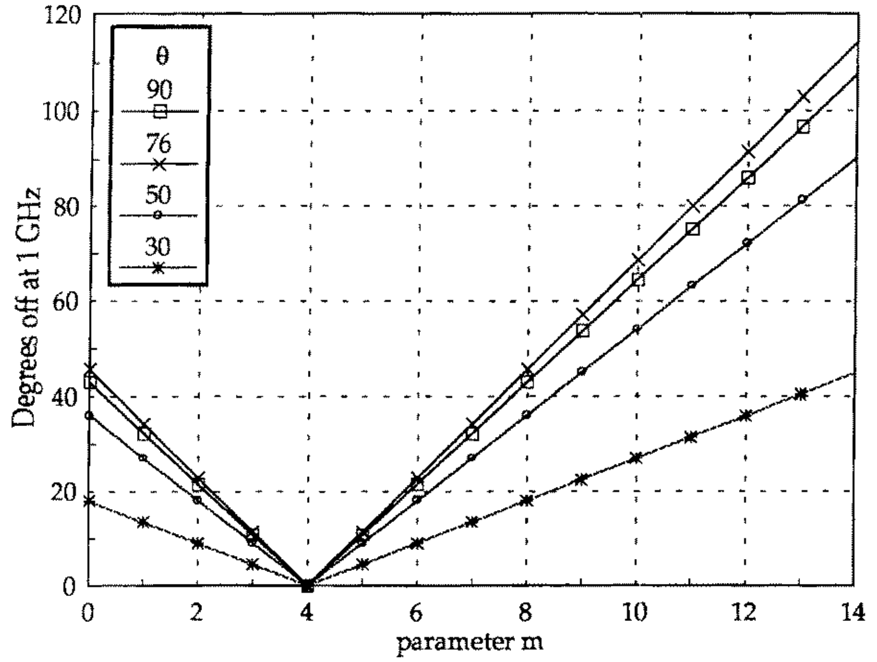


Figure 7. Error at right aperture end as a function of m

Figure 7 is a plot of the error at the right end point of the aperture as a function of m, with τ=0. The error at the right end is greater than the error at the left end for θ angles below 90° - so this is the worst case. Figure 8 is the same plot but with $\tau = n\alpha$ and n=4. In this case, the leading edge of the impulse response (at m=0) is out of focus by about 45° at 1 GHz. Perfect focus occurs when m=n=4.



Slant Range =2000
Sample Rate=2GHz
τ=4α

Figure 8. Error at right aperture end as a function of m

To conclude, Figures 4 through 8 show give an indication of the bounds over which a 5-cycle ring of a hi-Q scatterer is adequately captured by the approximation used in equation 4. These figures also show that making $\tau \neq 0$ can significantly increase the depth of focus on resonant targets.

6. Efficient Calculation

Fast focusing can be broken into 3 stages: first, pre-processing, where interpolation of the raw data is performed; second, computing a set of polynomial coefficients that will be used for fast index calculation; and third, performing the focusing summation of equation 7 using the coefficients found in second stage to find the index corresponding to the proper time shift.

6.1. Pre-Processing

The first aspect requiring a solution is a method to efficiently implement time shifting. The SAR data is collected by an A/D converter that outputs a vector of N numbers (voltages) at each aperture position. This vector will be called $\tilde{s}_j(\tilde{i})$. A time shift is, therefore, simply a shift in the index $\tilde{i}$. Typically, the time shifting obtained by indexing on the original N-point vector is not fine enough. Finer resolution is gained by a two-stage interpolator. First, a high quality interpolator is used to produce a new K point vector $s_j(i)$. It is usually implemented by inserting M-1 zeros between each data point in the original vector and then passing the new sequence through a low-pass FIR (Finite Impulse Response) filter. The process results in a new vector with nearly the desired time-shift resolution. The length, in this case, is $K = MN$. Extra fine resolution is gained by using a floating-point index with simple linear interpolation to find a value between any two data points in the interpolated data vector. For example, if the index i were 6.3, then the interpolated value would just be $.7s_j(6)+.3s_j(7)$.

The focusing algorithm that follows uses these techniques in the following sequence.

1. Read in one N-point vector $\tilde{s}_j(\tilde{i})$ of raw data.
2. Do an M-point interpolation, to form a new K-point vector $s_j(i)$.
3. Iterate for all pixels:
    a. Find the floating point index needed for a pixel.
    b. Find the data-value for that pixel by either rounding the index and grabbing the value, or by linearly interpolating between adjacent values.
    c. Sum into that pixel the data-value obtained.

6.2. Fast Index Calculation

6.2.1. *Approach*

Typically, the greatest computational load in backprojection focusing is in finding the index. Let *P(i,k,j)* be the exact index needed. Then equation 7, the 2-D focusing equation, becomes:

$$f_{i,k} = \sum_{j} s_j(P(i,k,j)) \tag{10}$$

Calculating $P(i,k,j)$ exactly involves 3-D trigonometric solutions for every pixel in the image at every position in the aperture. Such calculation is prohibitive! So for practical implementation, a secondary approximation is utilized to speed the computations. Extremely efficient computational methods exist for finding evenly spaced solutions to polynomials. Therefore, the approach will be to define a polynomial in three variables *G(i,k,j)*, where $G(i,k,j) \approx P(i,k,j)$, and use *G* to compute the index.

The first issue that arises is choosing the order of the polynomial needed for each variable. For now (to explain the method) suppose that a second degree polynomial is adequate for each of the 3 variables (*i,k,j*). Now the problem can be re-stated as, "for a given image pixel (*i,k*), and a given aperture position (*j*), find the coefficients a_m for m=0..26 such that

$$\begin{aligned}
G(i,k,j) &\approx P(i,k,j) \\
&= q_{0,k,j} + q_{1,k,j}i + q_{2,k,j}i^2 \\
&= [c_{0,j}+c_{1,j}k + c_{2,j}k^2] + [c_{3,j} + c_{4,j}k + c_{5,j}k^2]i + [c_{6,j} + c_{7,j}k + c_{8,j}k^2]i^2 \\
&= [(a_0 + a_1 j + a_2 j^2) + (a_3 + a_4 j + a_5 j^2)k + (a_6 + a_7 j + a_8 j^2)k^2] \\
&+ [(a_9 + a_{10} j + a_{11} j^2) + (a_{12} + a_{13} j + a_{14} j^2)k + (a_{15} + a_{16} j + a_{17} j^2)k^2]i \\
&+ [(a_{18} + a_{19} j + a_{20} j^2) + (a_{21} + a_{22} j + a_{23} j^2)k + (a_{24} + a_{25} j + a_{26} j^2)k^2]i^2
\end{aligned} \tag{11}$$

Written in this format, it is easy to see that one can code the index calculation as loops nested 3 deep; with *j* as the outer loop, *k* as the middle loop, and *i* as the inner loop.

6.2.2. *Solving For The Coefficients*

The coefficients a_m can be found numerically. The approach is to calculate the exact index required for *M* points in the image at each of *H* positions in the aperture, and do a least-squares fit to find the coefficients

a_m for m=0..26. A generic solution to this problem, for arbitrary image size, is found by using a pseudo inverse to perform the least-squares fit. When matrix **A** is not square but rectangular, then $\mathbf{A}^{-1}$ is known as the Pseudo Inverse and is defined as:

$$\mathbf{A}^{-1} = \left(\mathbf{A}^{\mathrm{T}} \bullet \mathbf{A}\right)^{-1} \bullet \mathbf{A}^{\mathrm{T}}. \tag{12}$$

An example will explain the method. To simplify the example, we will find a solution for only a single position in the aperture at *j*=0. So we will find the c_0 through c_8 needed to calculate the index needed for s_0(G(*i*,*k*,0). Take M=25 points on a patch to be focused: 5 ranges (close-in to far-out: *i*= 0, *b*, 2*b*, 3*b*, 4*b*) on each of 5 azimuths (left side to right side: *k*=0, *a*, 2*a*, 3*a*, 4*a*). The vector $\vec{\mathbf{B}}$ will be the exact (floating-point) calculated index needed for those 25 points in the image. The matrix A will be set up so that the width of the patch is 4*a* and the depth of the patch is 4*b*. So we have

$$G(i,k,j)\Big|_{j=0} = c_{0,0} + c_{1,0}k + c_{2,0}k^2 + c_{3,0}i + c_{4,0}ik + c_{5,0}ik^2 + c_{6,0}i^2 + c_{7,0}i^2k + c_{8,0}i^2k^2 \tag{13}$$

or

$$G(i,k,j) = \begin{bmatrix} 1 & k & k^2 & i & ik & ik^2 & i^2 & i^2k & i^2k^2 \end{bmatrix} \bullet \vec{\mathbf{C}}_j$$

with

$$\vec{\mathbf{A}} \bullet \vec{\mathbf{C}} = \vec{\mathbf{B}} \tag{14}$$

which is,

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & a & a^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2a & 4a^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 3a & 9a^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 4a & 16a^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & b & 0 & 0 & b^2 & 0 & 0 \\
1 & a & a^2 & b & ab & a^2b & b^2 & ab^2 & a^2b^2 \\
1 & 2a & 4a^2 & b & 2ab & 4a^2b & b^2 & 2ab^2 & 4a^2b^2 \\
1 & 3a & 9a^2 & b & 3ab & 9a^2b & b^2 & 3ab^2 & 9a^2b^2 \\
1 & 4a & 16a^2 & b & 4ab & 16a^2b & b^2 & 4ab^2 & 16a^2b^2 \\
1 & 0 & 0 & 2b & 0 & 0 & 4b^2 & 0 & 0 \\
1 & a & a^2 & 2b & 2ab & 2a^2b & 4b^2 & 4ab^2 & 4a^2b^2 \\
1 & 2a & 4a^2 & 2b & 4ab & 8a^2b & 4b^2 & 8ab^2 & 16a^2b^2 \\
1 & 3a & 9a^2 & 2b & 6ab & 18a^2b & 4b^2 & 12ab^2 & 36a^2b^2 \\
1 & 4a & 16a^2 & 2b & 8ab & 32a^2b & 4b^2 & 16ab^2 & 64a^2b^2 \\
1 & 0 & 0 & 3b & 0 & 0 & 9b^2 & 0 & 0 \\
1 & a & a^2 & 3b & 3ab & 3a^2b & 9b^2 & 9ab^2 & 9a^2b^2 \\
1 & 2a & 4a^2 & 3b & 6ab & 12a^2b & 9b^2 & 18ab^2 & 36a^2b^2 \\
1 & 3a & 9a^2 & 3b & 9ab & 27a^2b & 9b^2 & 27ab^2 & 81a^2b^2 \\
1 & 4a & 16a^2 & 3b & 12ab & 48a^2b & 9b^2 & 36ab^2 & 144a^2b^2 \\
1 & 0 & 0 & 4b & 0 & 0 & 16b^2 & 0 & 0 \\
1 & a & a^2 & 4b & 4ab & 4a^2b & 16b^2 & 16ab^2 & 16a^2b^2 \\
1 & 2a & 4a^2 & 4b & 8ab & 16a^2b & 16b^2 & 32ab^2 & 64a^2b^2 \\
1 & 3a & 9a^2 & 4b & 12ab & 36a^2b & 16b^2 & 48ab^2 & 144a^2b^2 \\
1 & 4a & 16a^2 & 4b & 16ab & 64a^2b & 16b^2 & 64ab^2 & 256a^2b^2
\end{bmatrix}
\begin{bmatrix}
c_{0,0} \\ c_{1,0} \\ c_{2,0} \\ c_{3,0} \\ c_{4,0} \\ c_{5,0} \\ c_{6,0} \\ c_{7,0} \\ c_{8,0}
\end{bmatrix}
=
\begin{bmatrix}
P(0,0,0) \\ P(0,a,0) \\ P(0,2a,0) \\ P(0,3a,0) \\ P(0,4a,0) \\
P(b,0,0) \\ P(b,a,0) \\ P(b,2a,0) \\ P(b,3a,0) \\ P(b,4a,0) \\
P(2b,0,0) \\ P(2b,a,0) \\ P(2b,2a,0) \\ P(2b,3a,0) \\ P(2b,4a,0) \\
P(3b,0,0) \\ P(3b,a,0) \\ P(3b,2a,0) \\ P(3b,3a,0) \\ P(3b,4a,0) \\
P(4b,0,0) \\ P(4b,a,0) \\ P(4b,2a,0) \\ P(4b,3a,0) \\ P(4b,4a,0)
\end{bmatrix}$$

The inverse of A is found to be:

$$(A^{-1})^T=\begin{bmatrix}
\frac{961}{1225} & \frac{-837}{1225a} & \frac{31}{245a^2} & \frac{-837}{1225b} & \frac{729}{1225ab} & \frac{-27}{245a^2b} & \frac{31}{245b^2} & \frac{-27}{245ab^2} & \frac{1}{49a^2b^2} \\
\frac{279}{1225} & \frac{403}{2450a} & \frac{-31}{490a^2} & \frac{-243}{1225b} & \frac{-351}{2450ab} & \frac{27}{490a^2b} & \frac{9}{245b^2} & \frac{13}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{-93}{1225} & \frac{124}{245a} & \frac{-31}{245a^2} & \frac{81}{1225b} & \frac{-108}{245ab} & \frac{27}{245a^2b} & \frac{-3}{245b^2} & \frac{4}{49ab^2} & \frac{-1}{49a^2b^2} \\
\frac{-31}{245} & \frac{837}{2450a} & \frac{-31}{490a^2} & \frac{27}{245b} & \frac{-729}{2450ab} & \frac{27}{490a^2b} & \frac{-1}{49b^2} & \frac{27}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{93}{1225} & \frac{-403}{1225a} & \frac{31}{245a^2} & \frac{-81}{1225b} & \frac{351}{1225ab} & \frac{-27}{245a^2b} & \frac{3}{245b^2} & \frac{-13}{245ab^2} & \frac{1}{49a^2b^2} \\
\frac{279}{1225} & \frac{-243}{1225a} & \frac{9}{245a^2} & \frac{403}{2450b} & \frac{-351}{2450ab} & \frac{13}{490a^2b} & \frac{-31}{490b^2} & \frac{27}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{81}{1225} & \frac{117}{2450a} & \frac{-9}{490a^2} & \frac{117}{2450b} & \frac{169}{4900ab} & \frac{-13}{980a^2b} & \frac{-9}{490b^2} & \frac{-13}{980ab^2} & \frac{1}{196a^2b^2} \\
\frac{-27}{1225} & \frac{36}{245a} & \frac{-9}{245a^2} & \frac{-39}{2450b} & \frac{26}{245ab} & \frac{-13}{490a^2b} & \frac{3}{490b^2} & \frac{-2}{49ab^2} & \frac{1}{98a^2b^2} \\
\frac{-9}{245} & \frac{243}{2450a} & \frac{-9}{490a^2} & \frac{-13}{490b} & \frac{351}{4900ab} & \frac{-13}{980a^2b} & \frac{1}{98b^2} & \frac{-27}{980ab^2} & \frac{1}{196a^2b^2} \\
\frac{27}{1225} & \frac{-117}{1225a} & \frac{9}{245a^2} & \frac{39}{2450b} & \frac{-169}{2450ab} & \frac{13}{490a^2b} & \frac{-3}{490b^2} & \frac{13}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{-93}{1225} & \frac{81}{1225a} & \frac{-3}{245a^2} & \frac{124}{245b} & \frac{-108}{245ab} & \frac{4}{49a^2b} & \frac{-31}{245b^2} & \frac{27}{245ab^2} & \frac{-1}{49a^2b^2} \\
\frac{-27}{1225} & \frac{-39}{2450a} & \frac{3}{490a^2} & \frac{36}{245b} & \frac{26}{245ab} & \frac{-2}{49a^2b} & \frac{-9}{245b^2} & \frac{-13}{490ab^2} & \frac{1}{98a^2b^2} \\
\frac{9}{1225} & \frac{-12}{245a} & \frac{3}{245a^2} & \frac{-12}{245b} & \frac{16}{49ab} & \frac{-4}{49a^2b} & \frac{3}{245b^2} & \frac{-4}{49ab^2} & \frac{1}{49a^2b^2} \\
\frac{3}{245} & \frac{-81}{2450a} & \frac{3}{490a^2} & \frac{-4}{49b} & \frac{54}{245ab} & \frac{-2}{49a^2b} & \frac{1}{49b^2} & \frac{-27}{490ab^2} & \frac{1}{98a^2b^2} \\
\frac{-9}{1225} & \frac{39}{1225a} & \frac{-3}{245a^2} & \frac{12}{245b} & \frac{-52}{245ab} & \frac{4}{49a^2b} & \frac{-3}{245b^2} & \frac{13}{245ab^2} & \frac{-1}{49a^2b^2} \\
\frac{-31}{245} & \frac{27}{245a} & \frac{-1}{49a^2} & \frac{837}{2450b} & \frac{-729}{2450ab} & \frac{27}{490a^2b} & \frac{-31}{490b^2} & \frac{27}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{-9}{245} & \frac{-13}{490a} & \frac{1}{98a^2} & \frac{243}{2450b} & \frac{351}{4900ab} & \frac{-27}{980a^2b} & \frac{-9}{490b^2} & \frac{-13}{980ab^2} & \frac{1}{196a^2b^2} \\
\frac{3}{245} & \frac{-4}{49a} & \frac{1}{49a^2} & \frac{-81}{2450b} & \frac{54}{245ab} & \frac{-27}{490a^2b} & \frac{3}{490b^2} & \frac{-2}{49ab^2} & \frac{1}{98a^2b^2} \\
\frac{1}{49} & \frac{-27}{490a} & \frac{1}{98a^2} & \frac{-27}{490b} & \frac{729}{4900ab} & \frac{-27}{980a^2b} & \frac{1}{98b^2} & \frac{-27}{980ab^2} & \frac{1}{196a^2b^2} \\
\frac{-3}{245} & \frac{13}{245a} & \frac{-1}{49a^2} & \frac{81}{2450b} & \frac{-351}{2450ab} & \frac{27}{490a^2b} & \frac{-3}{490b^2} & \frac{13}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{93}{1225} & \frac{-81}{1225a} & \frac{3}{245a^2} & \frac{-403}{1225b} & \frac{351}{1225ab} & \frac{-13}{245a^2b} & \frac{31}{245b^2} & \frac{-27}{245ab^2} & \frac{1}{49a^2b^2} \\
\frac{27}{1225} & \frac{39}{2450a} & \frac{-3}{490a^2} & \frac{-117}{1225b} & \frac{-169}{2450ab} & \frac{13}{490a^2b} & \frac{9}{245b^2} & \frac{13}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{-9}{1225} & \frac{12}{245a} & \frac{-3}{245a^2} & \frac{39}{1225b} & \frac{-52}{245ab} & \frac{13}{245a^2b} & \frac{-3}{245b^2} & \frac{4}{49ab^2} & \frac{-1}{49a^2b^2} \\
\frac{-3}{245} & \frac{81}{2450a} & \frac{-3}{490a^2} & \frac{13}{245b} & \frac{-351}{2450ab} & \frac{13}{490a^2b} & \frac{-1}{49b^2} & \frac{27}{490ab^2} & \frac{-1}{98a^2b^2} \\
\frac{9}{1225} & \frac{-39}{1225a} & \frac{3}{245a^2} & \frac{-39}{1225b} & \frac{169}{1225ab} & \frac{-13}{245a^2b} & \frac{3}{245b^2} & \frac{-13}{245ab^2} & \frac{1}{49a^2b^2}
\end{bmatrix}$$

Now $\vec{\mathbf{C}}$ is calculated as $\vec{\mathbf{C}} = \vec{\mathbf{A}}^{-1}\vec{\mathbf{B}}$, and is used to find a floating-point index pointing into the data array. This floating-point index is either rounded to pick the nearest point or is used to do a linear interpolation between the two closest data points in order to find a data value.

Two options are available to find the coefficients $a_0 \ldots a_{26}$. The first method is to go through the same process as shown but for the full problem. That process would involve: enlarging $\vec{\mathbf{C}}$ to hold the 27 $a_{0..26}$ coefficients; enlarging $\vec{\mathbf{B}}$ to hold the ideal values for not just one position in the aperture, but more, say 5 positions; and enlarging $\vec{\mathbf{A}}$ to match.

The second method is to solve for vector $\vec{\mathbf{C}}$ (as shown) at a number of positions in the aperture. Then construct a polynomial in j for each coefficient. This solution does not involve inverting the large A matrix but will result in a less than optimum solution.

One beauty of this second method, is that it lends itself to correcting for airplane motion. Since **A** is independent of the airplane position, it is inverted once. The matrix multiplication to find the index polynomial coefficients can be applied at every aperture position or short sub apertures if desired. So the algorithm allows real-time UWB motion compensation.

6.2.3. *Fast Polynomial Calculation*

Now that we have a polynomial to find the index, we need a fast way to compute the polynomial. Directly calculating an Nth degree polynomial usually requires N adds and N multiplies. If, however, a sequence of solutions is desired with the variable changed in fixed increments - the case we have here - then more efficient means are available. An algorithm by Nuttall[9] describes a method which can be applied here, to calculate the index recursively without the multiplications. The procedure is as follows:

1. Let $X(i) = q_0 + q_1 i + q_2 i^2$ be the polynomial to be solved, where i=0,1,2,3....

2. Let $X_1(i) = X(i) - X(i-1) = q_1 - q_2 + 2q_2 i$

3. Observe that $X_1(i) - X_1(i-1) = 2q_2$

4. Therefore a recursion can be set up where:

[9]Albert H. Nuttall, "Efficient Evaluation of Polynomials and Exponentials of Polynomials for Equispaced Arguments" IEEE ASSP-35 No.10 (Oct. 1987) pp1486-1487

$$X_1(i) = X_1(i-1) + 2q_2, \text{ and } X(i) = X_1(i-1) + X_1(i).$$

The starting values for the recursion are:

$$X(0) = q_0, \text{ and } X_1(0) = q_1 - q_2 + 2q_2 i = q_1 - q_2$$

The same derivation can be applied to an arbitrary N^th degree polynomial to produce a recursive formula that requires N adds per step. The technique can also be expanded to the multiple dimensions by nesting. Appendix B, lists all routines used in the fast focusing algorithm. Subroutine poly2 in section B.5 of appendix B uses this technique directly.

6.2.4. *Coefficient Generator Program*

Figure 9 is a diagram showing how the image is broken down along with the names used in the subroutines.

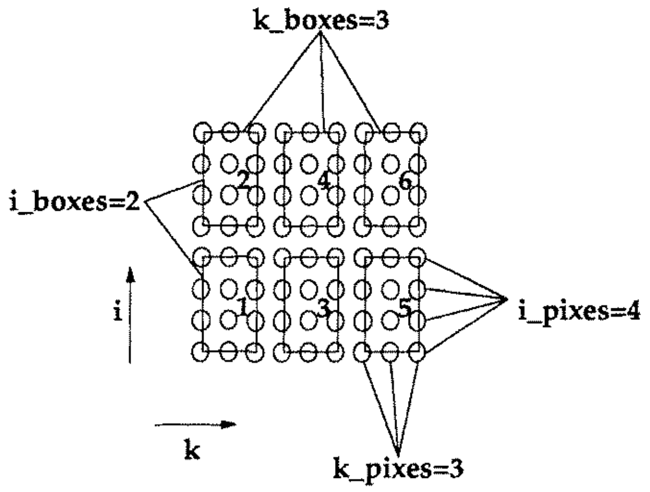


Figure 9. Image Partitioning and Notation

The basic flow chart is shown in figure 10. Each time the inner loop goes through all the n's, The Z_p's are the a0 to a26 coefficients for the box (defined by h and m) and the sub-aperture (defined by l). The flow-chart shows that the polynomial degree for the C_n is not always the same. The degree was optimized based on the errors caused by going to a lower degree. Appendix A provides a complete listing of the program that generates the focusing coefficients from an input file with the geometry.

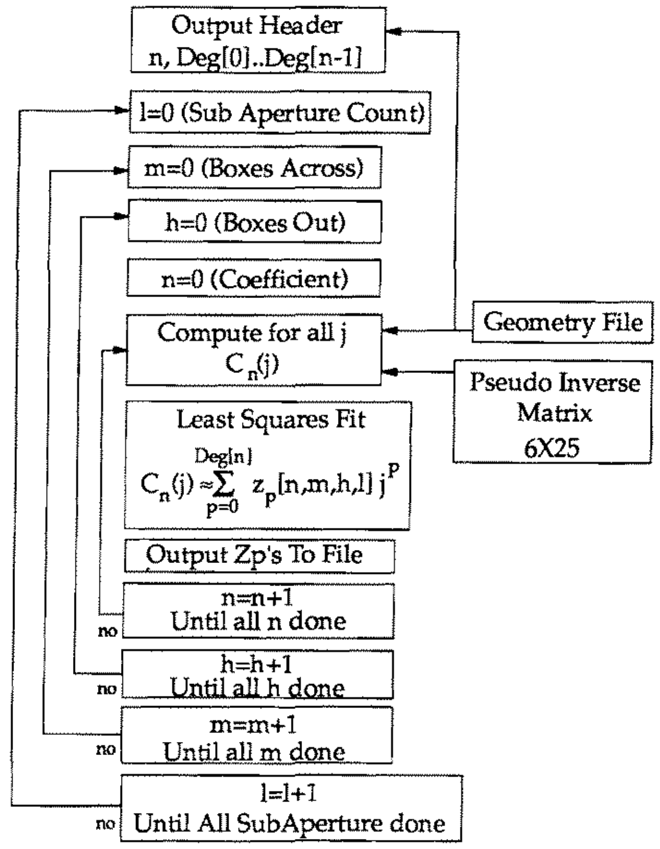


Figure 10. Coefficient Generator Flow Chart

6.3. Fast Focusing Algorithm

The fast focusing algorithm simply applies the fast polynomial solution technique to find the indexing and does the signal summation into all the pixels in the image. As outlined above, the index calculation is done in three nested loops. To aid in explanation, subroutines Inner_Loop, Middle_Loop, and Outer_Loop are pseudo-coded[10] examples of how the focusing routine is written using the recursive technique with nesting. A second degree polynomial is used throughout for illustration purposes. These subroutines assume that the image is broken into a number of boxes where each box has its own set of coefficients. Inner_Loop is the simplest subroutine and follows the derivation of the recursive formula directly. It essentially increments i to focus a single line in a box.

Inner_Loop (f, s, s_max, q, i_pixes)	Inner Loop Subroutine
float *f;	pointer to first pixel in a line; f(i_start,k)
float *s;	pointer to signal-data vector from jth aperture position

float *q;	pointer to coefficients vector for index calculation
int s_max;	s range is s[0]..s[s_max] s_max;
int i_pixes;	size of the box (pixels) along i axis is i_pixes;

f_stop_i = f + i_pixes-1;
T=q[2]+q[2];
R1=q[1]-q[2];
R0=q[0];
	Set up initial conditions

If R0 < -0.5 then repeat
    {f++;
    R1=R1+T;
    R0=R0+R1};
    until R0 > -0.5};
If (f > f_stop_i) then return;
	Perform Clipping when index<0
	(prior to beginning of actual data)

index=Round(R0);
*f = *f + *(s+index);
	Perform 1st summation

repeat	begin loop for a line in box
{f++;	increment pointer to next pixel
R1=R1+T;	increment index R0=R0+R1;

index=Round(R0);
    if index>s_max return;
	(clip when index is beyond actual data record length)

*f = *f +;*(s+index)	sum into pixel the new data;

until (f = f_stop);
return;
	(END Inner_Loop)

Nesting of the recursive approach means that the coefficients q0, q1, and q2 are each formed by polynomials in k. Each polynomial is incremented recursively in the subroutine Middle_Loop. So Middle_Loop sums into a single box the data from a single aperture position.

Middle_Loop(f, s, c, s_max, i_pixes, k_inc, box_offset);	Subroutine
float *f;	pointer to first pixel of box
float *s;	pointer to jth data vector
float *c;	pointer to coefficients vector for index calculation
int s_max;	s range is s[0]..s[s_max]
int k_inc;	F[i,k] = *(f + i + k * k_inc) so k_inc is the total range line length
int i_pixes;	size of the box (pixels) along i axis
int box_offset;	Number to add to f, to move pointer to last line in box
f_last = f + box_offset;	f_last = address of the first pixel in the last line of box

q0_T=c[2]+c[2];
q0_R1=c[1]-c[2];
q[0]=c[0];
	Set up initial conditions for q0

q1_T=c[5]+c[5];
q1_R1=c[4]-c[5];
q[1]=c[3];
	Set up initial conditions for q1

q2_T=c[8]+c[8];
q2_R1=c[7]-c[8];
q[2]=c[6];
	Set up initial conditions for q2

Call Inner_Loop(f,s,q..);	Perform summation on first line
repeat	start loop for rest of lines
f = f + k_inc;	increment pointer to next line in box

q0_R1=q0_R1+q0_T;
    q[0]=q[0]+q0_R1;
	iterate q0

q1_R1=q1_R1+q1_T;
    q[1]=q[1]+q1_R1
	iterate q1

q2_R1=q2_R1+q2_T;
    q[2]=q[2]+q2_R1;
	iterate q2

Call Inner_Loop(..);	Perform summation on current line in box

Until (f = f_last);
Return;
	Loop until all lines done

Again, nesting of the recursive approach means that the coefficients c0..c8 are each formed by polynomials in j. Each polynomial is incremented recursively in the subroutine Outer_Loop. So Outer_Loop sums into a each box the data from a multiple aperture positions. Since it is desirable to only read the signal-data once, Outer_Loop will both call Middle_Loop for each box and increment the coefficients for each box separately as it increments j across the aperture.

Outer_Loop(f,a,s_max,k_boxes,i_boxes,k_pixes,i_pixes,j_beg, j_end)	
float *fo;	pointer to first pixel in entire image; all pixels set to zero.
float *a;	pointer to array of coefficient vectors for index calculation; one vector per box.
int s_max;	s has range of s[0] to s[s_max]
int	number of boxes in k axis k_boxes;
int	number of boxes in i axis i_boxes;
int k_pixes;	size of the box (pixels) in k axis
int	size of the box (pixels) in i axis i_pixes;
int j_beg;	start of data vectors to be processed
int j_end;	stop of data vectors to be processed
k_inc = i_pixes * i_boxes;	(F(i,k))= *(f+i+k*k_inc)
box_offset = k_inc * (k_pixes-1);	Used by Middle_Loop
box_inc_k = i_pixes + box_offset;	box_inc_i = i_pixes;
j=j_start; read s[0..s_max] for j th position;	read signal data for first aperture position

```
f = fo - (box_inc_k+box_inc_i);                  set up to loop through all boxes
box=-2;
for k_box = 0 to k_boxes-1;
   box = box + 1;
   f = f + box_inc_k;
   for i_box = 0 to i_boxes-1;
      box = box + 1;
      f = f + box_inc_i;
        c0_T[box]=a[2,box]+a[2,box];                initial cond. for c0 for current box
        c0_R1[box]=a[1,box]-a[2,box];
        c[0,box]=a[0,box];
        c1_T[box]=a[5,box]+a[5,box];                initial cond. for current box
        c1_R1[box]=a[4,box]-a[5,box];
        c1 c[1,box]=a[3,box];
        ..                                          initial cond. for .c2..c7 for current
        ..                                          box
        ..
        c8_T[box]=a[27,box]+a[27,box];              initial cond. for c8 for current box
        c8_R1[box]=a[26,box]-a[27,box];
        c[8,box]=a[25,box];
        Call Middle_Loop(f,s,c[box],...);           first summation for current box
   END (for i_box);                              loop through all boxes
END (for k_box);
repeat                                           loop through all aperture positions
   j=j+1;                                          increment aperture
   read s[0..s_max];                               position read data for that aperture
                                                   position
   f = fo - (box_inc_k+box_inc_i);
   box=-2;                                         set up to loop through all boxes
   for k_box=0 to k_boxes-1;
      box = box + 1;
      f = f + box_inc_k;
      for i_box=0 to i_boxes-1;
         box = box + 1;
         f = f + box_inc_i;
           c0_R1[box]=c0_R1[box]+c0_T[box];           iterate c0 for current box
           c[0,box]=c[0,box]+c0_R1[box];
           c1_R1[box]=c1_R1[box]+c1_T[box];           iterate c1 for current box
           c[1,box]=c[1,box]+c1_R1[box];
           ..                                         iterate c2..c7 for current box
           c8_R1[box]=c8_R1[box]+c8_T[box];           iterate c8 for current box
           c[8,box]=c[8,box]+c8_R1[box];
           Call Middle_Loop(f,s,c[box],..);           Sum into current box
      END (for i_box);                             Loop until all boxes done
   END (for k_box);
```

Until j=j_stop; Return (END Middle_Loop)	Loop until all aperture positions are done.

6.3.1. *Polynomial Order Selection And Computational Load*

A study was conducted to determine the effect of polynomial order on the image size and aperture length. Referring to Figure 3, the scenario geometry was for an aperture of L_S=384 feet, a squint angle of $\theta = 85^\circ$, a slant range of 550 feet to the closest range line i=0, an elevation of 60 feet, an image area 54° wide, for the worst-case position and box j=3 and (i,k)=(0,3), and for a 4 Gs/s data rate

with record lengths (s_max) of 4096 points. If the error in the floating point index is restricted to $\pm.5$, then this geometry produces the following results:

Poly Nom Degree	max i_pixes	max k_pixes
1	228	27
2	964	105
3	2191	315
4		657

By using the nested recursive technique as shown in the example subroutines, the computational load is computed as shown below.

i'	polynomial order for i index;
k'	polynomial order for k index;
j'	polynomial order for j index;
J	Number of points in the aperture;
I	Number of range bins (pixels) in box (i_pixes);
K	Number of azimuth bins (pixels) in box (k_pixes);
B	Number of boxes
L0=I*(i'+1)	Adds per Inner Loop call
L1=K*(Lo+(i'+1)k')	Adds per Middle Loop call
L=J*(L1+(i'+1)(k'+1)j')	Adds per Outer Loop (adds per image)

Grouping terms we get:

L=(inner loop adds) + (middle loop adds) + (outer loop adds)

=IKJB(i'+1) + KJB(i'+1)k' + JB(i'+1)(k'+1)j'

From this equation it is clear that a low order polynomial is crucial in the inner loop. Conversely, the order of the polynomial in the outer loop can be high with little impact on the overall speed. The table below shows the computational load for various configurations with j'=3 and J=2304. The upper number is the total number of adds (L)in Giga-adds. The lower number is the total number of boxes in the image (B). The center number is the bytes of storage needed for the coefficient tables which is (i'+1)(k'+1)(j'+1)*B*(4 bytes/word).

Table 2. Computational Load Versus Partitioning

			i'	1	2	3	4
			K	27	103	256	512
			boxes across	19	5	2	1
			total pixels across	513	515	512	512
q	I	boxes down	total pixels down				
1	227	18	4086	L=9.711 S=21888 B=342	L=9.786 S=8640 B=90	L=9.769 S=4608 B=36	L=9.810 S=2880 B=18
2	817	5	4085	L=14.51 S=9120 B=95	L=14.58 S=3600 B=25	L=14.51 S=1920 B=10	L=14.53 S=1200 B=5
3	2043	2	4086	L=19.33 S=4864 B=38	L=19.41 S=1920 B=10	L=19.31 S=1024 B=4	L=19.32 S=640 B=2

6.4. Implementation Notes

The algorithm developed here requires no multiplication except in the pre-processing (interpolation) stage. There already exist DSP chips whose architecture is ideal for the FIR interpolation task. The bulk of the computations, however, occur in the summing and index calculation stage. Two facts make the summing algorithm attractive. First, adders take considerably less area to implement in a VLSI chip than multipliers. And second, parallel operation is extremely simple; the image can simply be broken into boxes with a separate processor working independently on each box. Except for passing the signal-data to each processor, the processors could run independently; only the coefficient table would be different. Therefore, design of a custom LSI chip appears practical and could result in real-time speeds.

If an off-the-shelf DSP chip with a parallel adder and multiplier is used to perform this algorithm, then other options become available. For example, the multiplier can be used in the summing algorithm to do interpolation on the indexing rather than rounding. The multiplication required can be done during other add cycles such that it takes zero time.

Another possibility is to calculate the index polynomial directly instead of recursively. Finally, all the operations can be fixed point. So the address-generator ALU (Arithmetic Logic Unit) can sometimes be used in parallel with the floating point units.

7. Beam Patterns and Side Lobe Structure

To develop an intuitive understanding for the beam shape, 16 figures were assembled to display, in the time domain, the main-lobe and side-lobe beam patterns. Simulation of a resonant target was done by generating a data record for every position in the aperture. The simulated scene was a single point target. The target bearing, refering to figure 3, was squinted 15^{o} off broadside ($\theta=105^{o}$). Range to the target R was 750 feet. The aperture length L_S was 385 feet. The aperture height was 60 feet. The point target had an impulse response of

$$s(t)=\begin{cases} \sin(2\pi ft)[.5+.5\cos(4\pi ft)] & \text{for } 0<t<\dfrac{1}{4f} \\ \sin(2\pi ft)e^{(\frac{1}{4f}-t)0.23f} & \text{for } t\geq\dfrac{1}{4f} \end{cases} \qquad (15)$$

The data simulated a 2 GHz sample rate. A record length of 2048 samples was made for each aperture position. Each record was pre-processed with a times-8 interpolator to produce 16K records. Interpolation was done by the standard FIR (Finite Impulse Response) filter method. A 255 tap Parks-McClellan low-pass filter with a 950 MHz cutoff was used to do the interpolation. Equation 7 (with n=0) was used to produce the focused beams. A Hilbert Transform was used to obtain the magnitude that is plotted in the figures. Plots were made on a dB scale. Sixteen plots were made to fill the matrix shown in table 3.

Table 3. Point Response Function Plots

Frequency	On-Axis	On-Axis	Above-Axis	Above-Axis
Equal	Equal	Hamming	Equal	Hamming
50MHz	Figure 11	Figure 12	Figure 13	Figure 14
200MHz	Figure 15	Figure 16	Figure 17	Figure 18
400MHz	Figure 19	Figure 20	Figure 21	Figure 22
900MHz	Figure 23	Figure 24	Figure 25	Figure 26

The on axis plots were made so that amplitude values could be easily read off of the plots. The above axis plots were made in order to see the sidelobe structure, and the ring-down time of the target. The axis labels were shifted so that 0 range was at the point target and so that 0 degrees was the bearing centered on the target. Notice that all plots are 3-D, even the on-axis plots. The various horizontal lines in the on-axis plots are the beam pattern on the i^{th} range bin. These horizontal lines are identical to those shown in the above axis plots; they are just being viewed "end on."

These figures are unique in that they show how the sidelobes spread in time as one moves off of the main beam. Due to the sharp rise-time of the target echo, the contribution from the near and far ends of the aperture can be seen as the early and late peaks in the sidelobe structure. Hamming weighting was applied to the array to enable one to see the effect of weighting on the sidelobe structure. It is interesting to note that although the peak sidelobe levels only drop marginally, the average, or integrated sidelobe levels are significantly lower when tapered aperture weighting is used. The weighting also reduces the near and far peaks in the sidelobes since the weighting reduces the contribution of the array ends.

Since the antenna is a linear system, it is no surprise that the beam pattern behaves basically as classic antenna theory predicts. The aperture beam pattern is a function of 1) how long the aperture is relative to wavelength, and 2) what kind of weighting is used to sum the points in the aperture. If uniform aperture weighting is used (a straight summation), then the typical $\sin(\beta\psi)/(\beta\psi)$ pattern occurs, where: ψ is the angle off the main beam; and $\beta \propto L_S/\lambda$. β establishes the width of the main beam. As the wavelength gets small, or as the aperture gets long, the beam gets narrow.

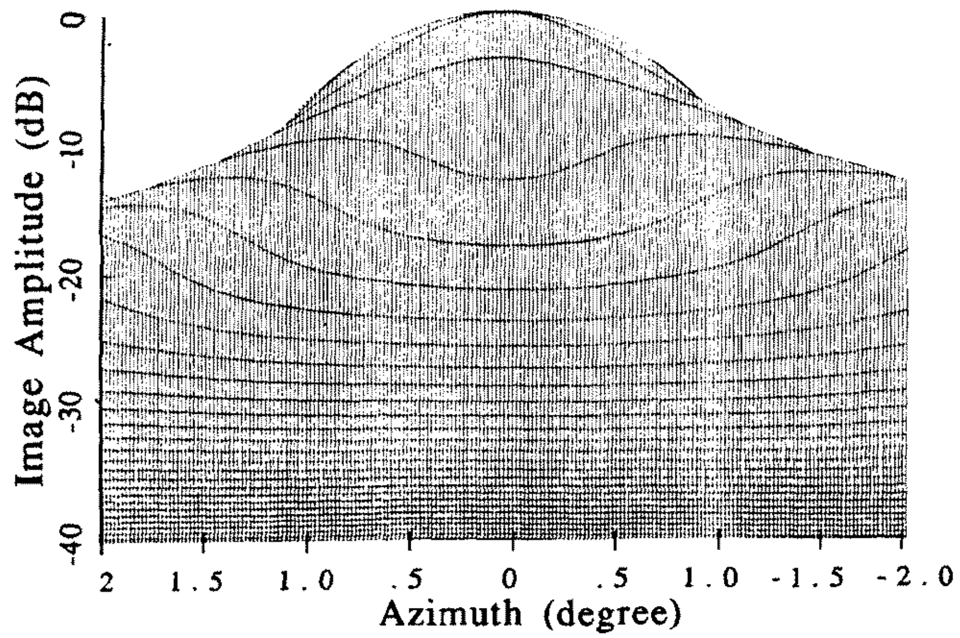


Figure 11.
50MHz Point reflector
with Equal Weighting
End View

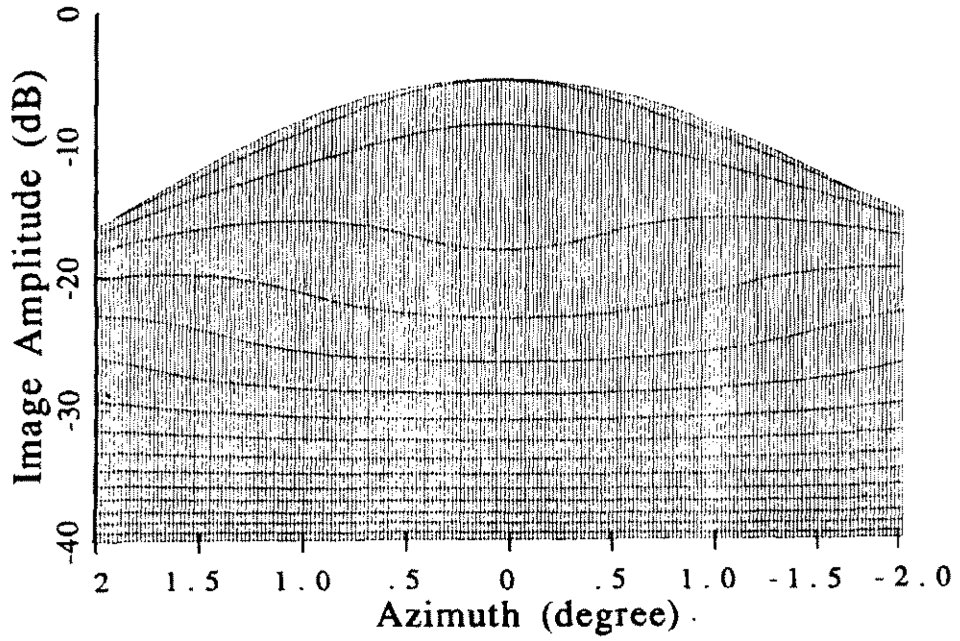


Figure 12.
50MHz Point reflector,
Hamming Weighting,
End View

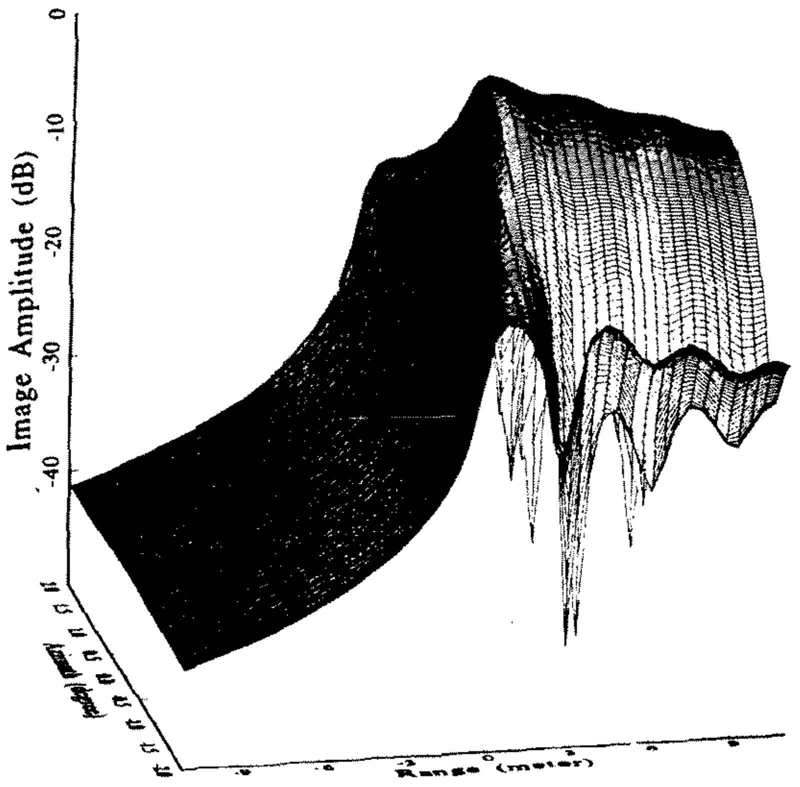


Figure 13.
50MHz Point reflector,
Equal Weighting,
Above Axis

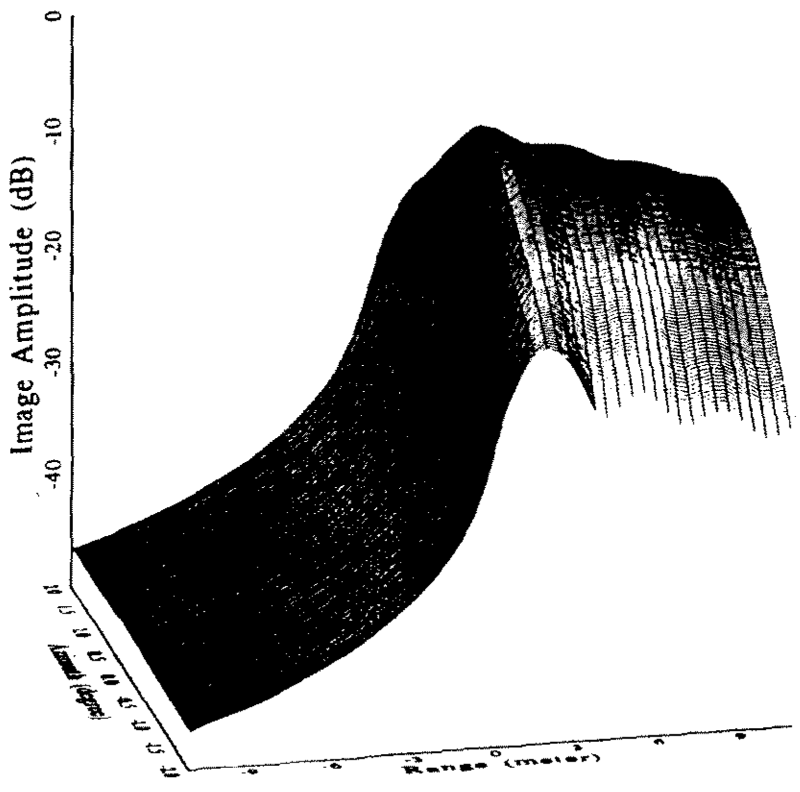


Figure 14.
50MHz Point reflector
Hamming Weighting
Above Axis

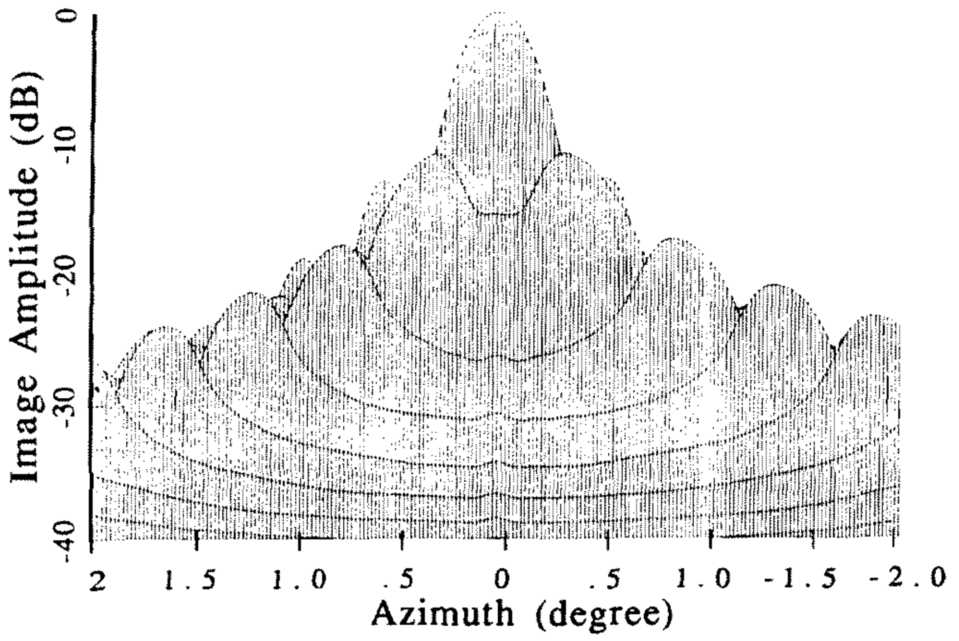


Figure 15.
200MHz Point reflector
Equal Weighting
End View

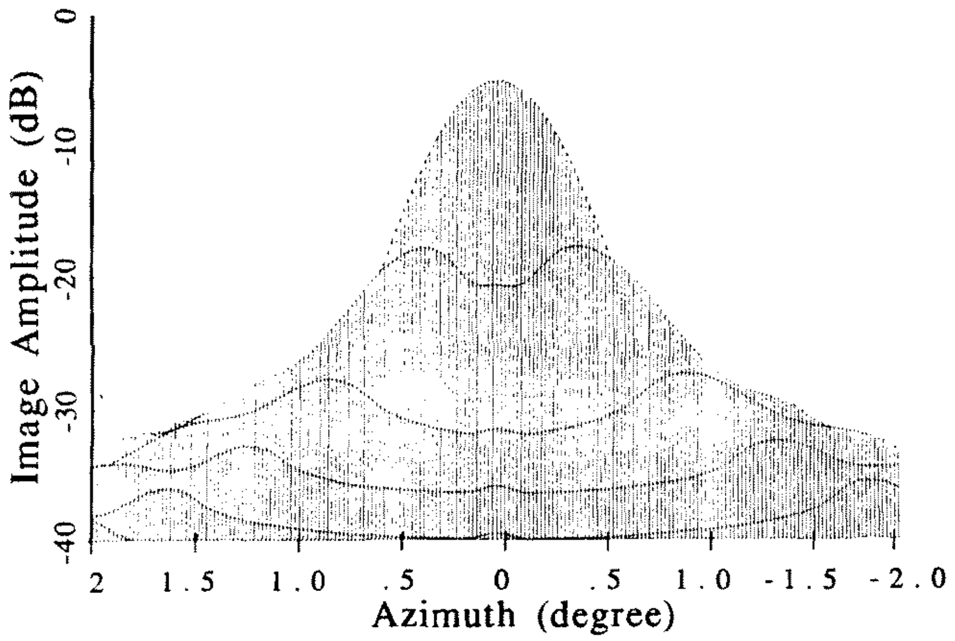


Figure 16.
200MHz Point reflector
Hamming Weighting
End View

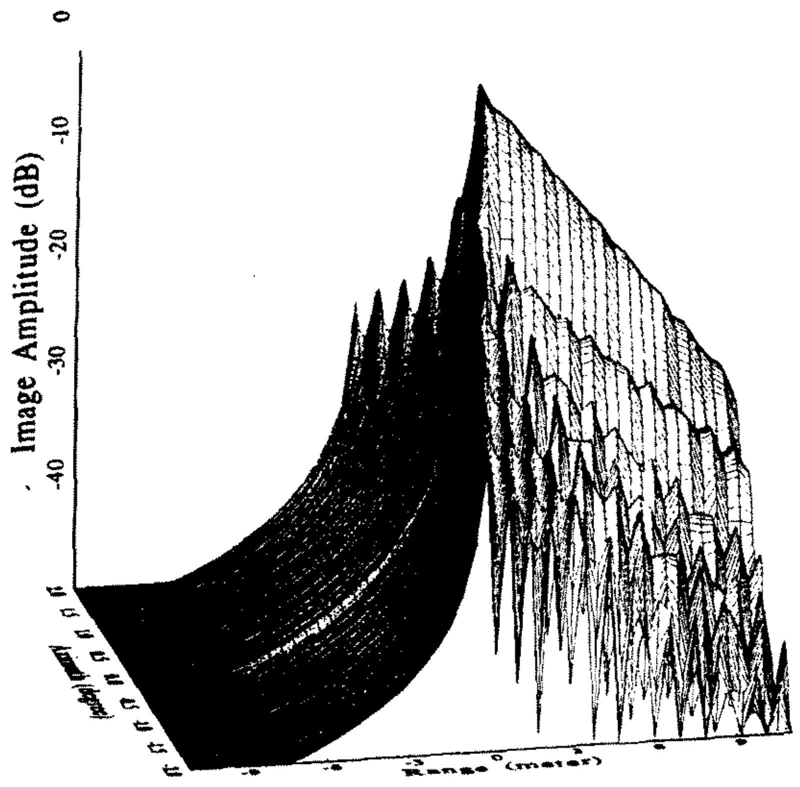


Figure 17.
200MHz Point reflector
Equal Weighting
Above Axis

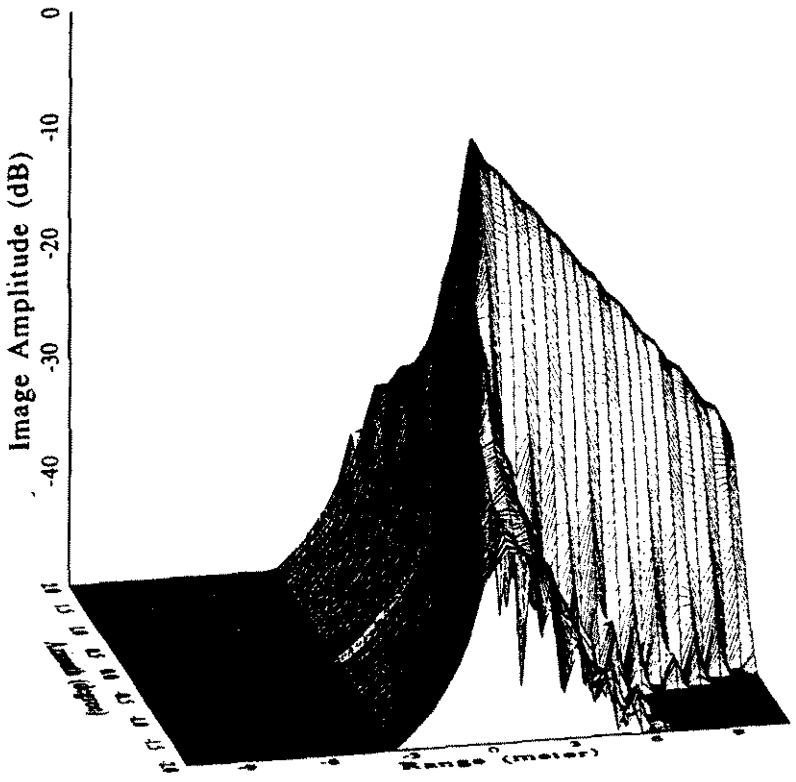


Figure 18.
200MHz Point reflector
Hamming Weighting
Above Axis

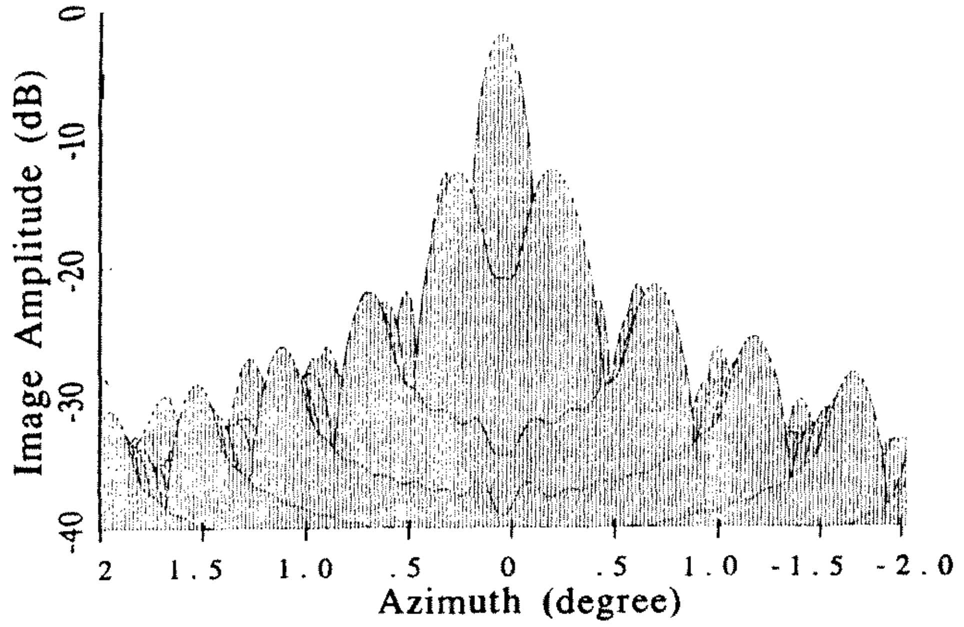


Figure 19.
400MHz Point reflector
Hamming Weighting
End View

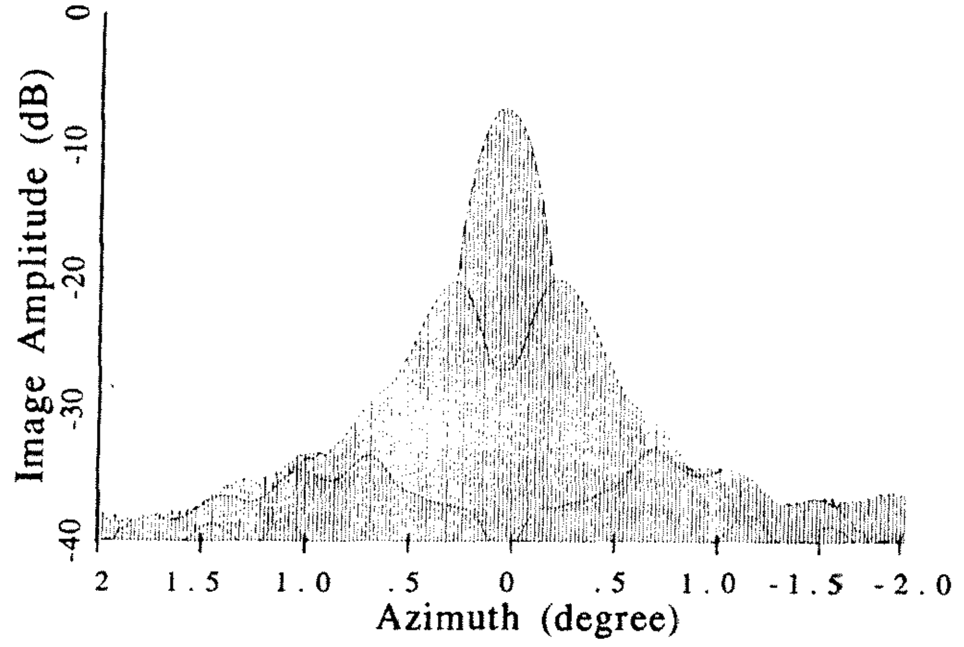


Figure 20.
400MHz Point reflector
Hamming Weighting
End View

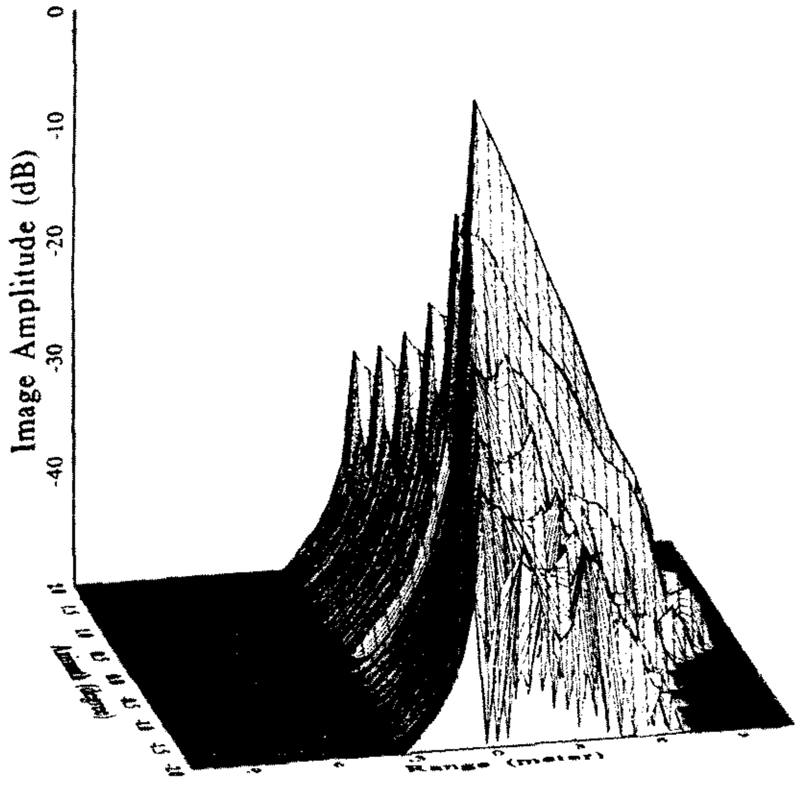


Figure 21.
400MHz Point reflector
Equal Weighting
Above Axis

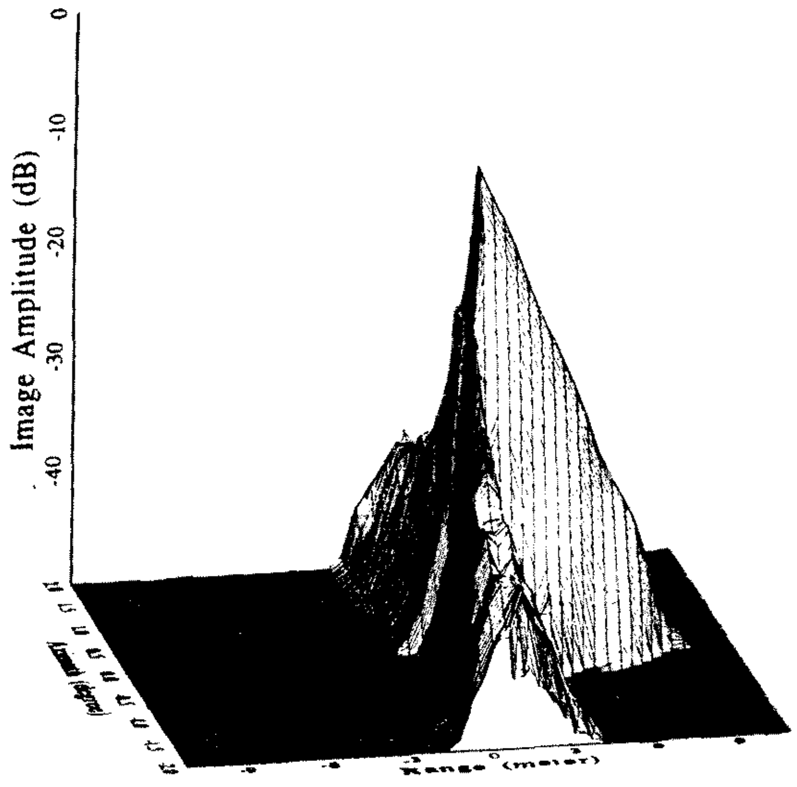


Figure 22.
400MHz Point reflector
Hamming Weighting
Above Axis

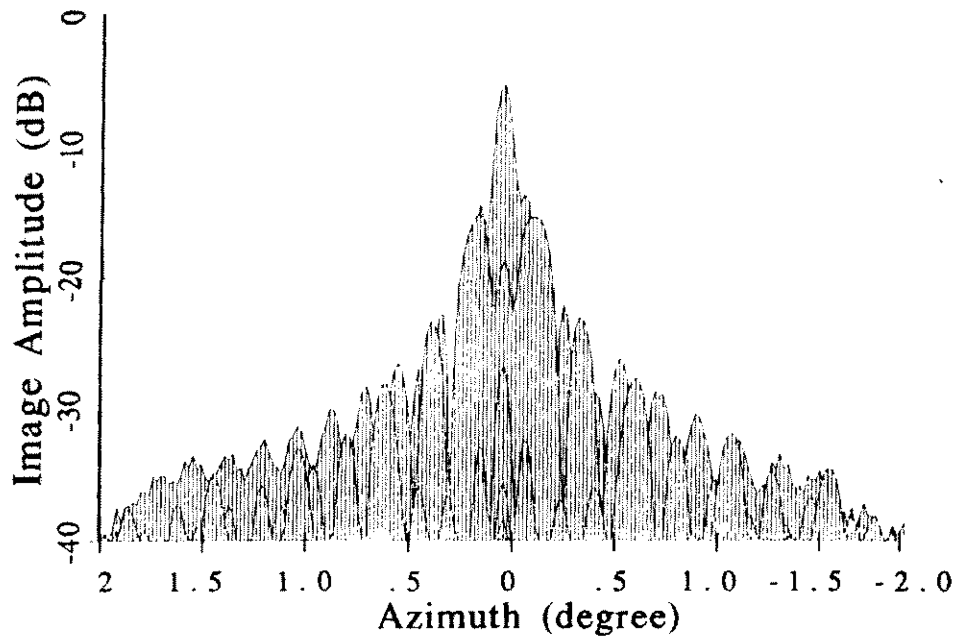


Figure 23.
900MHz Point reflector
Equal Weighting
End View

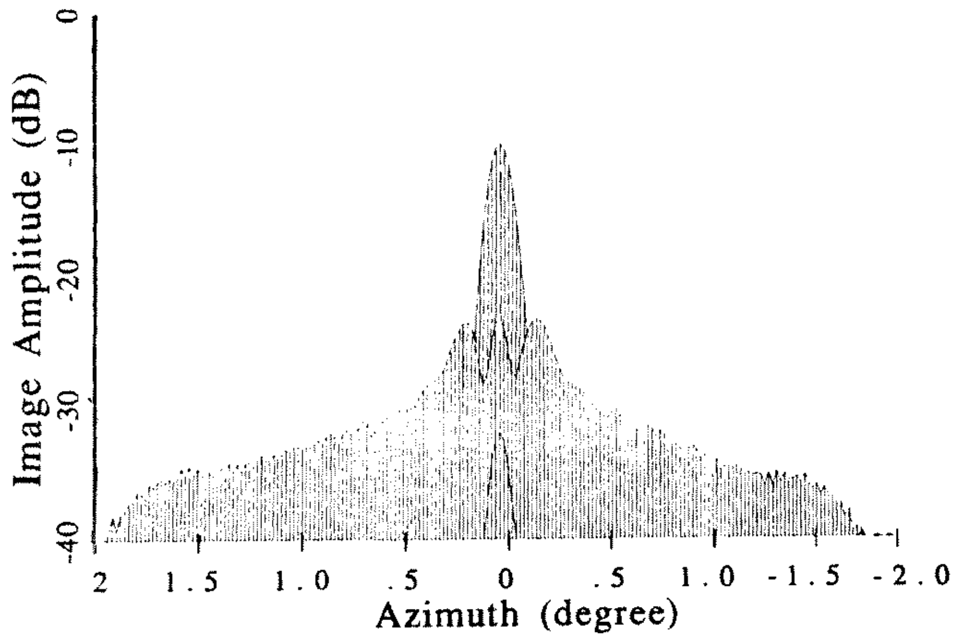


Figure 24.
900MHz Point reflector
Hamming Weighting
End View

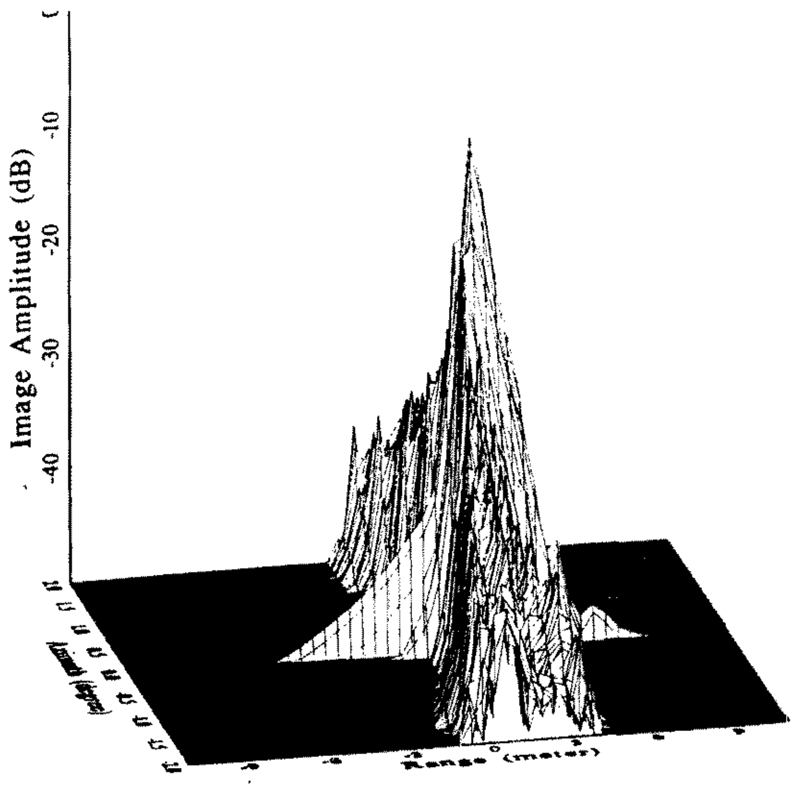


Figure 25.
900MHz Point reflector
Equal Weighting
Above Axis

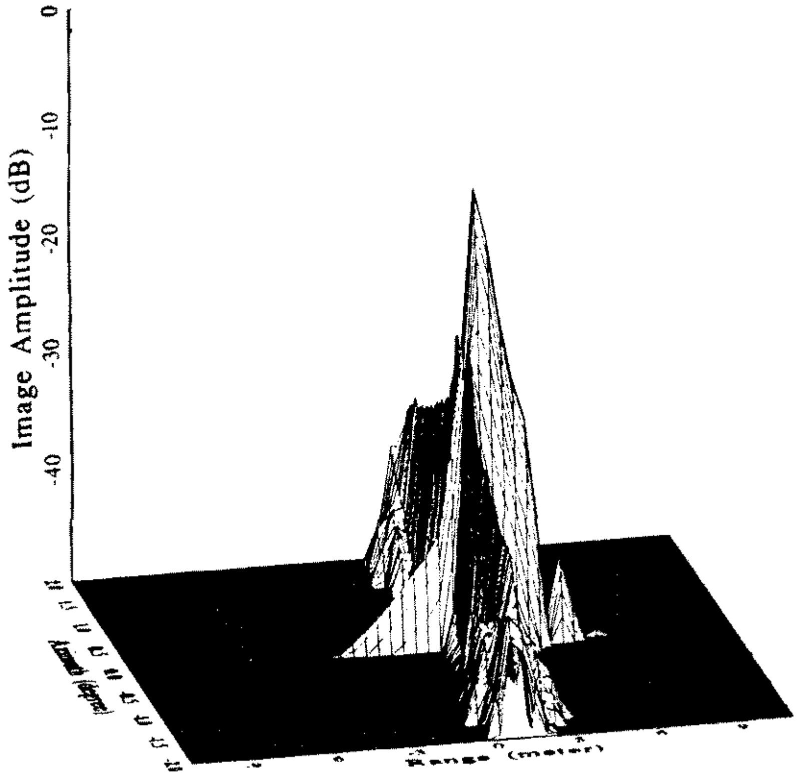


Figure 26.
900MHz Point reflector
Hamming Weighting
Above Axis

8. Conclusion

This report has identified a frequency independent processing algorithm to get a perfectly focused impulse response from any object at any position from an ultra wide bandwidth synthetic aperture radar. The report also presented an approximation that reduces the computational complexity of the algorithm. An error analysis of this approximation demonstrated its applicability to focus even hi-Q objects in the near field of an aperture. An implementation that is both computationally efficient and applicable to real-time motion compensation was also presented. Finally, plots were made demonstrating the capability of the algorithm to focus ringing targets over an 18-to-1 bandwidth.

APPENDIX A

Coefficient Genorator Program

A.1. Main Program to Calculate Coefficients (main.c)

```
/**************************************************************************

  Program : coefgen

  Description : This is the main  program to generate coefficients
            for fast focusing algorithm .


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

#include <malloc.h>
#include <stdio.h>
#include <math.h>
#include "coefgen.const"
#include "coefgen.var"

main()
{
 long cacheset;
 FILE *fp,*tstart_fp;
 char fn[80];
 long i;
 ia=(BOX_SIZE_AZIMUTH-4)/4;
 ib=(BOX_SIZE_RADIAL-3)/4;
/* initialize variables */
 coefgen_varinit();
/* initialize pseudo inver matrix */
 initmat();

 printf("Enter coefficient a  output file name ===> ");
 scanf("%s",fn);
 fp=fopen(fn,"w");
 if(fp==NULL)
  {
   printf("Error open output file \n");
   exit(1);
  }
/*
 printf("Enter data acquisition timing file name ===> ");
 scanf("%s",fn);
*/
/*
 tstart_fp=fopen("delaytime.dat","r");
 if(tstart_fp==NULL)
   {
    printf("Error open input file \n");
    exit(0);
   }
 read_delaytime(tstart_fp);
*/
 coefgen(fp);
}
```

A.2. Main Coefficient Generator Routine (coefgen.c)

```
#include "coefgen.h"                /*geometry file*/
#include "coefgen.var"              /*list of variables*/
#include <stdio.h>
#include <math.h>
extern  index();
extern  mxmul_();
extern  poly_fit();
/* Note that Reference point is now taken at coordinate kpixel= n_azimuth/2-1  */
/*                                        ipixel= n_radial/2 -1  */
/* At every position data were taken so that the reference point is at center  */
/* data buffer                                            */

void coefgen(fp)
FILE *fp;
{
 float input[NPOINT_APER/4];
 long i,j;
 long section,kbox,ibox,position,position_start,position_stop;
  long ipoint,kpoint;
 for(i=0;i<NPOINT_APER/n_section;i++)
   {
   input[i]=(float)i;
   }
 for(section=0;section<n_section;section++)
   {
    position_start=section*n_aper/n_section;
    position_stop=position_start+n_aper/n_section;
    for(kbox=0;kbox<NBOX_AZIMUTH;kbox++)
      {
      for(ibox=0;ibox<NBOX_RADIAL;ibox++)
        {
         printf("processing section %d ibox %d kbox %d\n",section,ibox,kbox);
        for(position=position_start;position<position_stop;position++)
         {
/******Calculating Actual index for all samples points in a box**********/
          i=0;
          for(ipoint=0;ipoint<NPOINT_BOXSAMPLE_RADIAL;ipoint++)
           {
            for(kpoint=0;kpoint<NPOINT_BOXSAMPLE_AZIMUTH;kpoint++)
             {
              ipixel=ibox*BOX_SIZE_RADIAL+ipoint*ib;
              jpixel=kbox*BOX_SIZE_AZIMUTH+kpoint*ia;

  d=((float)position-((float)n_aper/2.-1.))*(aper_length/((float)n_aper-1.)); /* distance from
radar to center of aperture */
  x=sqrt(rcenter_ref*rcenter_ref-radar_height2);
  r_ref=sqrt(radar_height2+x*x*c_theta_center_ref*c_theta_center_ref+(d-
x*s_theta_center_ref)*(d-x*s_theta_center_ref));
  rmin=r_ref-d_rcenter*((float)n_radial/2.-1.);
              index(&ipixel,&jpixel,&findex[i]);
               i++;
             }
           }
/* generate coef for this box at this position */

mxmuls(mat,&mat_c_stride,&stride,findex,&findex_c_stride,&stride,coef,&coef_c_stride,&stri
de,&n_coef,&i_one,&n_boxsample);

/* save in coefc[i][position]  */
          for(i=0;i<COEF_SIZE;i++)
           {
           coefc[i][position-position_start]=coef[i];
           }

/* for debug only */
/*   printf("pos= %d tstart= %.3e\n",position,2.0*rmin/c);  */
/* end debug */

        } /* position */
      for(i=0;i<COEF_SIZE;i++)
       {
        printf("Doing polyfit for coef %d\n",i);
poly_fit(input,&coefc[i][0],n_aper/n_section,deg[i],&coefa[i][0]);
        for(j=0;j<(deg[i]+1);j++)
         {
          fprintf(fp,"%e\n",coefa[i][j]);  /* save a[i] to file */
         }
       }
      } /* ibox */
     }  /* kbox */
  } /* section */

} /* subroutine */
```

A.3. Subroutine Find Ideal Index Vector (index.c)

```
/***********************************************************************/

 Subroutine:     index.c
 Input:
       ipixel        pixel radial coordinate
       jpixel        pixel angle coordinate
       a_span        angle spanned by the patch
       a_ofset       offset angle of the center line
       d             distance from middle position
       d_rcenter     sampling range
       dr8           (sampling range)/8
       rmincenter    min range rom center position
       hgt           height of building
       hgt2          square the height
       length        length of the building
 Output:
       findex        floating point index to ivalue of that pixel

***********************************************************************/
#include "coefgen.const"
#include "coefgen.var"
#include <math.h>

index(ipixel,kpixel,findex)
int *ipixel,*kpixel;
float *findex;
{

  theta=a_ofset-a_span/2.+(float)*kpixel*d_theta;
  c_theta=cos(theta);
  s_theta=sin(theta);
  rcenter=rmincenter+(float)*ipixel*d_rcenter;

  x=sqrt(rcenter*rcenter-radar_height2);
  r=sqrt(radar_height2+x*x*c_theta*c_theta+(d-x*s_theta)*(d-x*s_theta));
  *findex=(r-rmin)/dr8;
}
```

A.4. Geometry File - Declare All Global Constants (coefgen.h)

```
#define NPOINT_AZIMUTH  600        /* number of bearing lines  */
#define NPOINT_RADIAL   4095       /* number of radial lines   */
#define NPOINT_APER     2304       /* # of positions in aperture */
#define NPOINT_DATA     2048       /* number of original data points */
#define NPOINT_DATA_INTER  NPOINT_DATA*8 /* number of data points after
                                    interpolated */
#define PI 3.141592654
#define RADAR_HEIGHT  60.          /* radar height in feet    */
#define A_OFSET       -15.0        /* offset angle from center line */
#define A_SPAN        54.0         /* spanning angle of the patch */
#define RMINCENTER    550.         /* range from center position to
                         nearest point on the patch */
#define RCENTER_REF  802.0         /* range (ft) of ref. point from center pos. */
#define THETA_CENTER_REF  -15.0 /* angle (deg)of ref. point from center pos. */
#define SAMPLING_RATE 2.0e09       /* sampling frequency of signal  */
#define SAMPLING_PERIOD 5.0e-10    /* ts=1/fs                 */


#define BOX_SIZE_RADIAL     195
#define BOX_SIZE_AZIMUTH    100
#define NBOX_RADIAL     NPOINT_RADIAL/BOX_SIZE_RADIAL
#define NBOX_AZIMUTH  NPOINT_AZIMUTH/BOX_SIZE_AZIMUTH

#define SECTION_SIZE 4             /* # of sections for one aperture */
#define COEF_SIZE   6              /* # of coeffs for curve fit   */
#define MAXDEG      3              /* maximum degree for poly. fit */
#define NPOINT_BOXSAMPLE_RADIAL 5   /* # of samples in a box in radial
                         direction            */
#define NPOINT_BOXSAMPLE_AZIMUTH 5  /* # of samples in azimuth direction
                         for every box           */
#define NPOINT_BOXSAMPLE
NPOINT_BOXSAMPLE_RADIAL*NPOINT_BOXSAMPLE_AZIMUTH
                      /* # of samples in a box for curve fit */
```

A.5. Declare All Global Variables (coefgen.var)

```
struct complex { float r,i;};              /* define a complex type     */

/* Data Variables */
float pixel[NPOINT_AZIMUTH][NPOINT_RADIAL]; /* array of image            */
float data[NPOINT_DATA];                 /* original data bufer      */
float data_inter[NPOINT_DATA_INTER];       /* interpolated data buffer   */
struct complex data_inter_fft[NPOINT_DATA_INTER/2]; /* Real->Complex Forward FFT
```

```
                              of data_inter          */
 float filter_coef[NPOINT_DATA_INTER];      /* filter coefficients       */
 struct complex filter_fft[NPOINT_DATA_INTER/2];   /* Real->Complex Forward FFT
                              of filter_coef       */
 float nyquist_filter;        /* value of FFT of filter at Nyquist point */
 float nyquist_data;          /* value of FFT of data at Nyquist Point   */

 /* Geometry Variables */
 float a_ofset;              /* offset angle from the center line      */
 float a_span;               /* spanning angle of the patch            */
 float d_theta;              /* delta angle                   */
 float x,theta;              /* coordinate of a pixel              */
 float c_theta,s_theta;        /* cosine and sine of theha            */
 float d;                   /* distance from radar to center position
                            positive to the right, neg. to the left */
 float rcenter,rmincenter,rmaxcenter; /* range from center position        */
 float rcenter_ref;           /* range from center pos. to ref point     */
 float theta_center_ref;        /* angle of ref. point from center position */
 float r_ref;                /* range from any pos. to ref. point       */
 float d_rcenter;             /* sampling distance from center position  */
 float dr8;                 /* (sampling distance)/8              */
 float r,rmin,rmax;            /* range from any position             */
 float aper_length;            /* length of aperture               */
 float radar_height;           /* height of radar                 */
 float radar_height2;          /* square the height of radar           */

 /* Radar Variables */
 float c;                   /* speed of wave                 */
 float ts;                  /* sampling period of signal          */
 float fs;                  /* sampling frequency of signal         */

 long n_azimuth;              /* # of points in azimuth direction       */
 long n_radial;              /* # of points in radial direction       */
 long n_aper;                /* # of positions in aperture          */
 long pix_size;              /* # of points in pixel array          */
 long n_data;                /* number of original data points        */
 long n_data_inter;            /* number of interpolated data points     */
 long n_data_inter_half;         /* 1/2 # of interpolated data points      */
 long n_boxsample;             /* # of samples in a box for curve fit     */
 long n_coef;                /* # of coefficients curve fit          */
 long n_section;              /* # of sections for one aperture        */

 /* SSL VAriables */
 float f_zero;               /* floating point zero             */
 float f_one;                /* floating point 1               */
 long  i_one;                /* integer 1                  */
 long stride;                /* stride for supercard ssl            */

 /* working variables */
 long n_position;
 float e_index,a_index,error,maxerror,minerror;
 long ierr_max,jerr_max,ierr_min,jerr_min;
 long column_max,row_max,column_min,row_min;
 long ia,ib;
 long ipixel,jpixel;
 float c_theta_center_ref,s_theta_center_ref;
 float
mat[COEF_SIZE][NPOINT_BOXSAMPLE],findex[NPOINT_BOXSAMPLE],coef[COEF_SIZ
E];
 float coefc[COEF_SIZE][NPOINT_APER/4]; /* c coef. for each section      */
 float coefa[COEF_SIZE][MAXDEG+1];      /* a coef. for each c          */
 long deg[COEF_SIZE];                /* degree for each coefc        */
 long mat_c_stride,findex_c_stride,coef_c_stride;
 float tstart[NPOINT_APER];    /* time from trigger to start data acq. at */
                        /*     each position                   */
```

A.6. Matrix Pseudo Inverse Coefficients (initmat.c)

```
#include <stdio.h>
#include <math.h>
#include "coefgen.const"
#include "coefgen.var"

initmat()
{
int i,j;
float a,b;

a=((float)BOX_SIZE_AZIMUTH-4.)/4.;
b=((float)BOX_SIZE_RADIAL-3.)/4.;

mat[0][0]=93./175.;
mat[0][1]=27./175.;
mat[0][2]=(-9.)/175.;
mat[0][3]=(-3.)/35.;
mat[0][4]=9./175.;
mat[0][5]=62./175.;
mat[0][6]=18./175.;
mat[0][7]=(-6.)/175.;
mat[0][8]=(-2.)/35.;
mat[0][9]=6./175.;
mat[0][10]=31./175.;
mat[0][11]=9./175.;
```

```
mat[0][12]=(-3.)/175.;
mat[0][13]=(-1.)/35.;
mat[0][14]=3./175.;
mat[0][15]=0.;
mat[0][16]=0.;
mat[0][17]=0.;
mat[0][18]=0.;
mat[0][19]=0.;
mat[0][20]=(-31.)/175.;
mat[0][21]=(-9.)/175.;
mat[0][22]=3./175.;
mat[0][23]=1./35.;
mat[0][24]=(-3.)/175.;

mat[1][0]=(-81.)/(175.*a);
mat[1][1]=39./(350.*a);
mat[1][2]=12./(35.*a);
mat[1][3]=81./(350.*a);
mat[1][4]=(-39.)/(175.*a);
mat[1][5]=(-54.)/(175.*a);
mat[1][6]=13./(175.*a);
mat[1][7]=8./(35.*a);
mat[1][8]=27./(175.*a);
mat[1][9]=(-26.)/(175.*a);
mat[1][10]=(-27.)/(175.*a);
mat[1][11]=13./(350.*a);
mat[1][12]=4./(35.*a);
mat[1][13]=27./(350.*a);
mat[1][14]=(-13.)/(175.*a);
mat[1][15]=0.;
mat[1][16]=0.;
mat[1][17]=0.;
mat[1][18]=0.;
mat[1][19]=0.;
mat[1][20]=27./(175.*a);
mat[1][21]=(-13.)/(350.*a);
mat[1][22]=(-4.)/(35.*a);
mat[1][23]=(-27.)/(350.*a);
mat[1][24]=13./(175.*a);

mat[2][0]=3./(35.*pow(a,2.));
mat[2][1]=(-3.)/(70.*pow(a,2.));
mat[2][2]=(-3.)/(35.*pow(a,2.));
mat[2][3]=(-3.)/(70.*pow(a,2.));
mat[2][4]=3./(35.*pow(a,2.));
mat[2][5]=2./(35.*pow(a,2.));
```

```
mat[2][6]=(-1.)/(35.*pow(a,2.));
mat[2][7]=(-2.)/(35.*pow(a,2.));
mat[2][8]=(-1.)/(35.*pow(a,2.));
mat[2][9]=2./(35.*pow(a,2.));
mat[2][10]=1./(35.*pow(a,2.));
mat[2][11]=(-1.)/(70.*pow(a,2.));
mat[2][12]=(-1.)/(35.*pow(a,2.));
mat[2][13]=(-1.)/(70.*pow(a,2.));
mat[2][14]=1./(35.*pow(a,2.));
mat[2][15]=0.;
mat[2][16]=0.;
mat[2][17]=0.;
mat[2][18]=0.;
mat[2][19]=0.;
mat[2][20]=(-1.)/(35.*pow(a,2.));
mat[2][21]=1./(70.*pow(a,2.));
mat[2][22]=1./(35.*pow(a,2.));
mat[2][23]=1./(70.*pow(a,2.));
mat[2][24]=(-1.)/(35.*pow(a,2.));

mat[3][0]=(-31.)/(175.*b);
mat[3][1]=(-9.)/(175.*b);
mat[3][2]=3./(175.*b);
mat[3][3]=1./(35.*b);
mat[3][4]=(-3.)/(175.*b);
mat[3][5]=(-31.)/(350.*b);
mat[3][6]=(-9.)/(350.*b);
mat[3][7]=3./(350.*b);
mat[3][8]=1./(70.*b);
mat[3][9]=(-3.)/(350.*b);
mat[3][10]=0.;
mat[3][11]=0.;
mat[3][12]=0.;
mat[3][13]=0.;
mat[3][14]=0.;
mat[3][15]=31./(350.*b);
mat[3][16]=9./(350.*b);
mat[3][17]=(-3.)/(350.*b);
mat[3][18]=(-1.)/(70.*b);
mat[3][19]=3./(350.*b);
mat[3][20]=31./(175.*b);
mat[3][21]=9./(175.*b);
mat[3][22]=(-3.)/(175.*b);
mat[3][23]=(-1.)/(35.*b);
mat[3][24]=3./(175.*b);
```

```
mat[4][0]=27./(175.*a*b);
mat[4][1]=(-13.)/(350.*a*b);
mat[4][2]=(-4.)/(35.*a*b);
mat[4][3]=(-27.)/(350.*a*b);
mat[4][4]=13./(175.*a*b);
mat[4][5]=27./(350.*a*b);
mat[4][6]=(-13.)/(700.*a*b);
mat[4][7]=(-2.)/(35.*a*b);
mat[4][8]=(-27.)/(700.*a*b);
mat[4][9]=13./(350.*a*b);
mat[4][10]=0.;
mat[4][11]=0.;
mat[4][12]=0.;
mat[4][13]=0.;
mat[4][14]=0.;
mat[4][15]=(-27.)/(350.*a*b);
mat[4][16]=13./(700.*a*b);
mat[4][17]=2./(35.*a*b);
mat[4][18]=27./(700.*a*b);
mat[4][19]=(-13.)/(350.*a*b);
mat[4][20]=(-27.)/(175.*a*b);
mat[4][21]=13./(350.*a*b);
mat[4][22]=4./(35.*a*b);
mat[4][23]=27./(350.*a*b);
mat[4][24]=(-13.)/(175.*a*b);

mat[5][0]=(-1.)/(35.*pow(a,2.)*b);
mat[5][1]=1./(70.*pow(a,2.)*b);
mat[5][2]=1./(35.*pow(a,2.)*b);
mat[5][3]=1./(70.*pow(a,2.)*b);
mat[5][4]=(-1.)/(35.*pow(a,2.)*b);
mat[5][5]=(-1.)/(70.*pow(a,2.)*b);
mat[5][6]=1./(140.*pow(a,2.)*b);
mat[5][7]=1./(70.*pow(a,2.)*b);
mat[5][8]=1./(140.*pow(a,2.)*b);
mat[5][9]=(-1.)/(70.*pow(a,2.)*b);
mat[5][10]=0.;
mat[5][11]=0.;
mat[5][12]=0.;
mat[5][13]=0.;
mat[5][14]=0.;
mat[5][15]=1./(70.*pow(a,2.)*b);
mat[5][16]=(-1.)/(140.*pow(a,2.)*b);
mat[5][17]=(-1.)/(70.*pow(a,2.)*b);
mat[5][18]=(-1.)/(140.*pow(a,2.)*b);
mat[5][19]=1./(70.*pow(a,2.)*b);
```

```
mat[5][20]=1./(35.*pow(a,2.)*b);
mat[5][21]=(-1.)/(70.*pow(a,2.)*b);
mat[5][22]=(-1.)/(35.*pow(a,2.)*b);
mat[5][23]=(-1.)/(70.*pow(a,2.)*b);
mat[5][24]=1./(35.*pow(a,2.)*b);

}
```

A.7. Initialize All variables (varinit.c)

```
/***************************************************************************

 Subroutine: coefgen_varinit

 Description : initialize all neccessary variables

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

#include <stdio.h>
#include <math.h>
#include "coefgen.const"
#include "coefgen.var"
coefgen_varinit()
{
 n_azimuth=NPOINT_AZIMUTH;           /* # of points in azi. direction */
 n_radial=NPOINT_RADIAL;             /* # of points in rad. direction */
 pix_size=n_azimuth*n_radial; /* Pixel array size          */
 n_data=NPOINT_DATA;                 /* # of orig. data points      */
 n_data_inter=NPOINT_DATA_INTER;      /* # of interpolated. data points*/
 n_data_inter_half=NPOINT_DATA_INTER/2; /* 1/2 # of inter. data points  */
 n_aper=NPOINT_APER;                 /* # of points in the aperture  */
 n_boxsample=NPOINT_BOXSAMPLE;        /* # of samples in box for curve fit */
 n_coef=COEF_SIZE;                  /* # of coeffs for curve fit    */
 n_section=SECTION_SIZE;             /* # of sections for one aperture*/

/* Radar Variables */
 fs=SAMPLING_RATE;                   /* Sampling Frequency        */
 ts=SAMPLING_PERIOD;                  /* Sampling period          */
 c=3.e8;                       /* Speed of wave           */

/* Geometry variables  */
 a_ofset=A_OFSET*PI/180.;            /* Offset angle from center line */
 a_span=A_SPAN*PI/180.;             /* Spanning angle of the patch  */
 d_theta=a_span/((float)n_azimuth-1.); /* delta theta         */
 aper_length=(n_aper-1)*2.0*.0254;    /* length of aperture          */
 radar_height=RADAR_HEIGHT*12.*0.0254;  /* height of radar              */
 radar_height2=radar_height*radar_height; /* square the height */
 rmincenter=RMINCENTER*12.*0.0254;     /* min range from center position
                                  to reference point          */
 d_rcenter=ts*c/(2.0*2.0);         /* sampling range              */
                                   /* project back to 2*n_data samples */
 dr8=ts*c/(2.0*8.0);       /* interpolated 16K buffer */

/*
 theta_center=a_ofset-a_span/2.+((float)n_azimuth/2.-1.)*d_theta;
 c_theta_center=cos(theta_center);
 s_theta_center=sin(theta_center);
 rcenter_ref=rmincenter+d_rcenter*((float)n_radial/2.-1.);
*/
 rcenter_ref=RCENTER_REF*12.*.0254; /* range of ref. point from center pos. */
 theta_center_ref=THETA_CENTER_REF*PI/180.; /* angle of ref. point from center
                                position              */
 c_theta_center_ref=cos(theta_center_ref);
 s_theta_center_ref=sin(theta_center_ref);

/* SSL Variables */
 f_zero=0.;                 /* floating point zero         */
 stride=1;                  /* stride for ssl           */
 f_one=1.0;                 /* floating point 1           */
 i_one=1;                   /* integer 1               */
 mat_c_stride=n_boxsample;          /* column stride for mat        */
 findex_c_stride=1;              /* column stride for findex      */
 coef_c_stride=1;               /* column stride for coeff      */
 deg[0]=3;                  /* degree for first coefc        */
 deg[1]=3;
 deg[2]=3;
 deg[3]=3;
 deg[4]=2;
 deg[5]=2;

/* xgints_(pixel,&f_zero,&pix_size,&stride); */ /* clear pixel[][] */

}
```

A.8. Find Least Squares Polynomial Fit (poly_fit.c)

```
/*************************************************************************
  Subroutine: poly_fit.c
  Description:
          Perform data fit to polynomial
  Input:   x[n]        input array
          y[n]        input array
          n          number of elements
          deg         degree of poly.
```

```
   output    coeff[deg+1]     coefficients of poly
*************************************************************************/
#include <stdio.h>
#include <malloc.h>
#include <math.h>
void poly_fit(x,y,n,deg,coeff)
float x[],y[],coeff[];
long n,deg;
{
 double *dx,*dy,*dcoeff;
 double *dX,*dXtrans,*dA,*dB,*dC;
 long m,i,j,one;
 one=1;
 m=deg+1;
 dx=malloc(n*sizeof(double));
 if(dx==NULL)
  {
   printf("Memory allocation Error !!!\n");
   exit(1);
  }
 dy=malloc(n*sizeof(double));
 if(dy==NULL)
  {
   printf("Memory allocation Error !!!\n");
   exit(1);
  }
 dcoeff=malloc(m*sizeof(double));
 if(dcoeff==NULL)
  {
   printf("Memory allocation Error !!!\n");
   exit(1);
  }
 dX=malloc(n*m*sizeof(double));
 if(dX==NULL)
  {
   printf("Memory allocation Error !!!\n");
   exit(1);
  }
 dXtrans=malloc(n*m*sizeof(double));
 if(dX==NULL)
  {
   printf("Memory allocation Error !!!\n");
   exit(1);
  }
 dA=malloc(m*m*sizeof(double));
 if(dA==NULL)
```

```
   {
    printf("Memory allocation Error !!!\n");
   exit(1);
   }
  dB=malloc(m*m*sizeof(double));
  if(dB==NULL)
   {
   printf("Memory allocation Error !!!\n");
   exit(1);
   }
  dC=malloc(m*sizeof(double));
  if(dx==NULL)
   {
   printf("Memory allocation Error !!!\n");
   exit(1);
   }

  for(i=0;i<n;i++)
   {
   dx[i]=(double)x[i];
   dy[i]=(double)y[i];
   }
 for(i=0;i<n;i++)
  {
   dX[i*m]=1.0;
  }
 for(j=1;j<m;j++)
  {
  for(i=0;i<n;i++)
   {
    dX[j+i*m]=pow(dx[i],(double)j);
   }
  }
 mxtrans(dX,n,m,dXtrans);
 mxmuld(dXtrans,&n,&one,dX,&m,&one,dA,&m,&one,&m,&m,&n);
 mat_inverse(dA,dB,m);
 mxmuld(dXtrans,&n,&one,dy,&one,&one,dC,&one,&one,&m,&one,&n);
 mxmuld(dB,&m,&one,dC,&one,&one,dcoeff,&one,&one,&m,&one,&m);
 for(i=0;i<m;i++)
  {
  coeff[i]=(float)dcoeff[i];
  }
 free(dx);
 free(dy);
 free(dA);
 free(dB);
 free(dC);
 free(dX);
 free(dXtrans);
 free(dcoeff);
}
```

A.9. Matrix Inverter (inverse.c)

```
#include <stdio.h>

mat_inverse(source_mat,dest_mat,size_mat)
double source_mat[];
double dest_mat[];
long size_mat;
{
 long n,i,j,k,ki,count;
 double b,b1;
 double err=0.0001;
 double a[100][100];
 n=size_mat;
 count=0;

 for(i=0;i<size_mat;i++)
   {
   for(j=0;j<size_mat;j++)
     {
      a[i+1][j+1]=source_mat[count++];
     }
   }

 for(i=1;i<=n;i++)
   {
     for(j=n+1;j<=2*n;j++)
        {
         if((j-n-i)==0)
           { a[i][j]=1.0; }
         else
           { a[i][j]=0.; }
        }
   }

 for(k=1;k<=n-1;k++)

   {
    b=a[k][k];
```

```
     ki=k;
     for(i=k+1;i<=n;i++)
       {
        if((abs(b)-abs(a[i][k]))<0)
          {
           b=a[i][k];
           ki=i;
          }
       }

     if((abs(b)-err)<0)
       {
        printf("Error matrix inverse , matrix is singular\n");
        exit(1);
       }

     if((ki-k)!=0)
       {
        for(j=k;j<=2*n;j++)
          {
           b1=a[k][j];
           a[k][j]=a[ki][j];
           a[ki][j]=b1;
          }
       }

    for(j=k+1;j<=2*n;j++)
      {
       a[k][j]=a[k][j]/b;
      }

    for(i=k+1;i<=n;i++)
      {
       for(j=k+1;j<=2*n;j++)
         {
          a[i][j]=a[i][j]-a[i][k]*a[k][j];
         }
      }
    }
  for(j=n+1;j<=2*n;j++)
   {
   a[n][j]=a[n][j]/a[n][n];
   }

  for(k=n-1;k>=1;k=k-1)
   {
      for(j=n+1;j<=2*n;j++)
        {
         for(i=k+1;i<=n;i++)
           {
            a[k][j]=a[k][j]-a[k][i]*a[i][j];
           }
        }
      }

   count=0;
   for(i=0;i<size_mat;i++)
    {
     for(j=0;j<size_mat;j++)
      {
       dest_mat[count++]=a[i+1][size_mat+j+1];
      }
    }
  }
```

APPENDIX B

B.1. Main Program For SUN Computer(focus.c)

```
/*****************************************************************

 program: focus.c
 description:
       this programs reads radar data from file, interpolates
       data using convolution in freq. domain, and projects
       data back to the polar grid on the ground. The indeces
       used in back projection are computed from the file
       which contains coefficients for a particular geometry

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

#include <stdio.h>
#include "focus.h"
#include "focus.var"

 float data[NPOINT_DATA]; /* data buffer for host */
 struct cstype *cs[NO_SUPERCARD];

main ()
 {
 float tstart_err[NPOINT_APER]; /* tstart-tstart_trunc for each position */
 FILE *inputfile;   /* input file contains radar data         */
 FILE *outputfile;  /* output file contains focused image      */
 FILE *filterfile;  /* input file contains filter coefficients  */
 FILE *coeffile;    /* input file contains back projection      */
              /*                coefficients        */
 FILE *tstart_err_file; /* input file contains tstart-tstart_trunc  */
 FILE *junkfile;
 char inputname[80];
 char outputname[80];
 char filtername[80];
 char coefname[80];
 char tstart_err_name[80];
 char temp_string[20];
 long section,position,box,coefc_order,coefa_order;
 long i,j;          /* just working variable      */
 float f_position;
 long data_ready=1;  /* maibox to indicate data ready  host ---> SC */
 long data_consumed=2; /* mailbox to indicate data consumed  host <--- SC */
 long msg;          /* message read from mailbox  */
```

```
  long one=1;        /* nonzero message to put in mailbox */
  long cacheset=1;   /* cacheset=0 turn off cache  */
  long numpar=1;
  float pixel_zero[BOX_SIZE_AZIMUTH][NPOINT_RADIAL];


/*** open supercards *********************************************/
 for (i=0;i<NO_SUPERCARD;i++)
  {
   cs[i]=(struct cstype *)xlubgn_(0,&cacheset,"sc.lo");
  }

/*** initialize supercard variables *****************************/
 focus_varinit();

/*** initialize supercards' id **********************************/
 for(i=0;i<NO_SUPERCARD;i++)
  {
   cs[i]->supercard_id=i;
  }

/*** Enter Input ************************************************/
 printf("Enter input file name ( .raw )===> ");
 scanf("%s",inputname);
 printf("Enter output file name ( .focus)===> ");
 scanf("%s",outputname);
/*
 printf("Enter back projection coefficient file name (coefa.dat)===> ");
 scanf("%s",coefname);
 printf("Enter start time difference file name (delaytime_diff.dat)===> ");
 scanf("%s",tstart_err_name);
*/
 strcpy(coefname,"coefa.dat");
 strcpy(tstart_err_name,"delaytime_diff.dat");

 printf("Do you want hamming weight accross the aperture (y or n) ===> ");
 scanf("%s",temp_string);

 for(j=0;j<NO_SUPERCARD;j++)
  {
   cs[j]->ham_flag=0;
   if(temp_string[0]=='y')
    {
     cs[j]->ham_flag=1;
    }
  }
 strcpy(filtername,"filter.dat");
 if((inputfile=fopen(inputname,"rb"))==NULL)
   {
    printf("Error open input file\n");
    exit(1);
   }
 if((outputfile=fopen(outputname,"wb"))==NULL)
   {
    printf("Error open output file\n");
    exit(1);
   }
 if((filterfile=fopen(filtername,"r"))==NULL)
   {
    printf("Error open filter file\n");
    exit(1);
   }
 if((coeffile=fopen(coefname,"r"))==NULL)
   {
    printf("Error open back projection coefficient file\n");
    exit(1);
   }
 if((tstart_err_file=fopen(tstart_err_name,"r"))==NULL)
   {
    printf("Error open tstart difference file\n");
    exit(1);
   }

/*** Read filter coefficients ********************************/
/*** into 1st supercard and copy to others supercards **********/
 printf(">>> Reading filter coefficients\n");
 for(i=0;i<FILTER_LENGTH;i++)
  {
   fscanf(filterfile,"%f",&cs[0]->filter_coef[i]);
  }

 for(j=1;j<NO_SUPERCARD;j++)
  {
   for(i=0;i<FILTER_LENGTH;i++)
    {
    cs[j]->filter_coef[i]=cs[0]->filter_coef[i];
    }
  }
```

```
/*** Read in tstart_err for all positions to first supercard *****/
/*** and then copy to other supercards ***************************/
 printf(">>> Reading tstart_err data \n");
 for(i=0;i<NPOINT_APER;i++)
  {
   fscanf(tstart_err_file,"%d %f",&j,&(cs[0]->tstart_err[i]));
  }
 for(j=1;j<NO_SUPERCARD;j++)
  {
   for(i=0;i<NPOINT_APER;i++)
    {
    cs[j]->tstart_err[i]=cs[0]->tstart_err[i];
    }
  }


/*** Read in coefficients a for back projection ******************/
/*** to the first supercard and then copy to other supercards ****/
 printf(">>> Reading backprojection coefficients \n");
 for(section=0;section<SECTION_SIZE;section++)
  {
   for(box=0;box<NBOX_RADIAL*NBOX_AZIMUTH;box++)
    {
     for(coefc_order=0;coefc_order<COEF_SIZE;coefc_order++)
      {
       for(coefa_order=0;coefa_order<(cs[0]->deg[coefc_order]+1);coefa_order++)
        {
fscanf(coeffile,"%f",&(cs[0]->coefa[section][box][coefc_order][coefa_order]));
        for(j=1;j<NO_SUPERCARD;j++)
          {
          cs[j]->coefa[section][box][coefc_order][coefa_order]=
             cs[0]->coefa[section][box][coefc_order][coefa_order];
          }
        } /* coefa_order */
      }  /* coefc_order */
    }    /* box       */
  }      /* section    */

/*** Start supercards *********************************************/
 printf(">>> Starting supercard programs \n");
 for(j=0;j<NO_SUPERCARD;j++)
  {
   xrcall_("sc",&numpar,&cs[j]->dummy);
  }

/*** For every position, the host computer reads data for that ***/
```

```
 /*** position and copy data into each supercard memory. **********/
 /*** There are two mail boxes used betwwen host and each *********/
 /*** supercard to provide synchronization ***********************/

  for(section=0;section<SECTION_SIZE;section++)
   {
    for(position=0;position<NPOINT_APER/SECTION_SIZE;position++)
     {
     printf(">>>>>processing section %d pos %d <<<<<<\n",section,position);

 /*** Read in data for a position ********************************/
     fread(data,sizeof(data),1,inputfile);

     for(j=0;j<NO_SUPERCARD;j++)
       {
 /*** Copy data in each supercard memory *************************/
       for(i=0;i<NPOINT_DATA;i++)
         {
         cs[j]->data_temp[i]=data[i];
         }
 /*** Send mail to supercard to inform data is ready **************/
       xlnwxmt_(cs[j],&data_ready,&one);
       }
 /*** Wait until data is consumed by supercards ******************/
    for(j=0;j<NO_SUPERCARD;j++)
      {
      xlwtrec_(cs[j],&data_consumed,&msg);
      }
    }   /* position */
  }     /* section  */


 /*** Check for supercards to finish all processing *****************/
 for(j=0;j<NO_SUPERCARD;j++)
   {
   xldone_(cs[j]);
   }

 /*** Save resulting radar image ************************************/
 for(j=0;j<BOX_BASE_START;j++)
  {
  printf(">>>Prepad zero to output file \n");
  fwrite(pixel_zero,sizeof(pixel_zero),1,outputfile);
  }
 printf(">>>Save image to output file \n");
 for(j=0;j<NO_SUPERCARD;j++)
```

```
   {
    fwrite(cs[j]->pixel,

 sizeof(float)*BOX_SIZE_AZIMUTH*NPOINT_RADIAL*NO_KBOX_PROCESS/NO_SUPE
 RCARD,
     1,outputfile);
   }
  for(j=BOX_BASE_START+NO_KBOX_PROCESS;j<NBOX_AZIMUTH;j++)
   {
    printf(">>>Postpad zero to output file \n");
    fwrite(pixel_zero,sizeof(pixel_zero),1,outputfile);
   }

  fclose(inputfile);
  fclose(outputfile);

/*** Closing all supercards ****************************************/
  for(j=NO_SUPERCARD-1;j>=0;j--)
   {
    xlclos_(cs[j]);
   }

} /* end main */
```

B.2. Main Program For Multiple Array Processors (sc.c)

```
/*****************************************************************

  Program: sc.c

  Description: This is the main program to be executed by each of
    the CSPI i860 array processor.

*****************************************************************/
#include "focus.h"
#include "focus.var"

struct cstype *cs;


void sc(dummy)
long *dummy;
 {
  void focus_varinit();
  void filter_init();
```

```
   void haminit();
   void hamwt();
   void erase();
   void inter();
   void fix_data_pointer();
   void poly();
   void bp();
   void xwtrec_();
   void xnwxmt_();
   void xvmov_();
   void xvclr_();

   float *data_pointer;     /* pointer to actual data                */
   long section,position,box,coefc_order,kbox,ibox;
   long data_ready=1;  /* maibox to indicate data ready  host ---> SC */
   long data_consumed=2; /* mailbox to indicate data consumed  host <--- SC */
   long msg;           /* message read from mailbox  */
   long one=1;         /* nonzero message to put in mailbox */
   float f_position;
   long boxbase;

 /*** initialize supercard pointer ***********************************/
   cs=0;

 /*** initialize image with zeros ************************************/
   xvclr_(cs->pixel,&cs->pix_size,&cs->i_one);

 /*** initialize filter **********************************************/
   filter_init();

 /*** initialize hamming weight coefficients ************************/
   haminit(cs->ham_coef,NPOINT_APER);

 /*** Start forming image ********************************************/

  boxbase=BOX_BASE_START+cs->supercard_id*NO_KBOX_PROCESS/NO_SUPERCARD;

  for(section=0;section<SECTION_SIZE;section++)
   {
   for(position=0;position<NPOINT_APER/SECTION_SIZE;position++)
    {
/*** Wait for new data from host computer *************************/
    xwtrec_(&data_ready,&msg);

/*** Copy data to working buffer ***********************************/
    xvmov_(cs->data,cs->data_temp,&cs->n_data,&cs->i_one,&cs->i_one);
/*** Signal host computer that data are consumed ******************/
     xnwxmt_(&data_consumed,&one);

     f_position=(float)position;

/*** Hamming weight data if needed ********************************/
     if(cs->ham_flag==1)
      {
      hamwt(cs->data,cs->n_data,section*NPOINT_APER/SECTION_SIZE+position,
          cs->ham_coef);
      }

/*** zeros last portion of data for circular convolution ***********/
    erase(cs->data,cs->n_data,32);

/*** Interpolate data *********************************************/
    inter(cs->data,cs->n_data,cs->data_inter,
         cs->n_data_inter,cs->filter_fft);

    fix_data_pointer(&data_pointer,&cs->data_inter[0],
        cs->tstart_err[NPOINT_APER/SECTION_SIZE*section+position],
         cs->ts,8);

    for(kbox=boxbase;kbox<(boxbase+NO_KBOX_PROCESS/NO_SUPERCARD);kbox++)
     {
     for(ibox=IBOX_START;ibox<(IBOX_START+NO_IBOX_PROCESS);ibox++)
      {
      box=kbox*NBOX_RADIAL+ibox;

/*** Generate coefficients for back projection *******************/
       for(coefc_order=0;coefc_order<COEF_SIZE;coefc_order++)
        {
         poly(&f_position,&cs->coefc[coefc_order],1,
            &cs->coefa[section][box][coefc_order][0],
            cs->deg[coefc_order]);
        } /* coefc_order */

/*** Perform back projection **************************************/
       bp(cs->pixel,data_pointer,cs->n_data_inter,cs->coefc,
         boxbase,kbox,ibox,
         cs->k_pix_size,cs->i_pix_size,cs->i_box_size);

      } /* i_box */
     }  /* k_box */
   }    /* position */
```

```
 }      /* section  */

} /* end supercard program */
```

B.3. Routines Used to Interpolate (inter.c)

```
#include <math.h>
#include "focus.h"
#include "focus.var"
/***************************************************************************
 Subroutine: haminit.c
 Description:
        this subroutine initialize array of hamming weighting coeffs
        accross the aperture.
 Input:    float ham_coef[NPOINT_APER]
        long  npoint     Number of points for hamming coeffs
 Output:   float ham_coef[NPOINT_APER] is initialized
***************************************************************************/
haminit(ham_coef,npoint)
float *ham_coef;
long npoint;
{
 long i;
 for(i=0;i<npoint;i++)
   {
    ham_coef[i]=.54-.46*cos(2.*3.1415927*(float)i/(float)npoint);
   }
}


/***************************************************************************
 Subroutine: hamwt.c

 Description:
        this subroutine takes the radar signal at a position and
        multiplies the entire signal array with the hamming coef at
        that position.
 Input:
        float data[NPOINT_DATA]
        long  npoint_data
        long  position
        float ham_coef[];

 Output:
       float data[NPOINT_DATA]
```

```
**************************************************************************/

hamwt(data,npoint_data,position,ham_coef)
float *data,*ham_coef;
long npoint_data,position;
{
 long i;
 for(i=0;i<npoint_data;i++)
   {
    data[i]=data[i]*ham_coef[position];
   }
}



/**************************************************************************
 Subroutine: inter.c
 Description:
         This routine performs FIR interpolation by using convolution
         in the frequency domain. The input buffer contains 2K of data
         and is interpolated into 16K of data. The FIR filter has
         breakpoints at .95 Ghz and 1.05 Ghz
                 # of taps = 255
                 name of filter coeff. file : filter.dat
 Input:   data[NPOINT_DATA]
         complex filter_fft[NPOINT_DATA_INTER/2]
         float  nyquist_filter
 Output:  data_inter[NPOINT_DATA_INTER]
**************************************************************************/

inter()
{
 void xvclr_();
 void xfrf_();
 void xcvmls_();
 void xfri_();

 extern struct cstype *cs;
 int i;

/* clear data_inter buffer */
 xvclr_(cs->data_inter,&cs->n_data_inter,&cs->i_one);

/* interleave data into data-inter buffer */
 for(i=0;i<NPOINT_DATA;i++)
  {
   cs->data_inter[i*8]=cs->data[i];
   }

 /* FFT of interleaved data */
  xfrf_(cs->data_inter_fft,&cs->nyquist_data,cs->data_inter,
      &cs->n_data_inter_half);

 /* FFT of interpolated data (multiply with filter in freq. domain)*/
  xcvmls_(cs->data_inter_fft,&cs->f_one,cs->data_inter_fft,
       cs->filter_fft,&cs->n_data_inter_half);
  cs->nyquist_data=cs->nyquist_data*cs->nyquist_filter;

 /* inverse FFT to get interpolated data */
  xfri_(cs->data_inter,&cs->nyquist_data,cs->data_inter_fft,
      cs->data_inter_fft,&cs->n_data_inter_half);

 }


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

  Subroutine: fix_data_pointer

  Description: This subroutine fixes the pointer to the start of
          data_inter buffer. The pointer needed to be adjusted
          due to the following reasons:

          (a) The linear FIR interpolation filter has phase
             shift and the actual data point starts at bin 127
             of the data_inter buffer ( FIR has 255 coeffs )
          (b) The actual time to start data acquisition has to
             be rounded of to 1 nsec resolution for the scope
             DSA602. However the backprojection algorithm uses
             the exact time since the indeces have to be smooth
             for curve fit.

 Input:
        float  *data_inter    ( address of data_inter buffer )
        float  tstart_err    ( tstart-tstart_trunc          )
        float  ts            ( radar sampling period        )
        long   inter_factor  (interpolation factor          )
 Output:
        float  *data_pointer ( actual start pointer for data )

***********************************************************************/
```

```
fix_data_pointer(pointer,data_inter,tstart_err,ts,inter_factor)
float **pointer;
float *data_inter;
float tstart_err;
float ts;
long inter_factor;
 {
  long i;
  double float1,float2;
  unsigned long index_ofset;
  float1=modf(tstart_err*inter_factor/ts,&float2);
  if(float1>=.5) float2=float2+1.;
  index_ofset=(unsigned long)127+(unsigned long)float2;
  *pointer=data_inter+index_ofset;

/* zeros fill first 128 points of data_inter since phase shift from linear filter  */
  for(i=0;i<128;i++)
   {
    data_inter[i]=0.;
   }

/* zeros fill the last 128 points of data_inter plus 8 points for tstart_err     */
  for(i=0;i<136;i++)
   {
    data_inter[NPOINT_DATA_INTER+i]=0.;
   }

/* first point and last point of actual data array are zeros for backprojection */
  **pointer=0.;
  *(*pointer+NPOINT_DATA_INTER-1)=0.;
 }


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

  Subroutine: erase

  Description: Zero out the last nzero points of input buffer

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

erase(buffer,ndata,nzero)
float *buffer;
long ndata,nzero;
{
  long i;
  for(i=ndata-nzero;i<ndata;i++)
   {
    buffer[i]=0.;
   }
 }
```

B.4. Initialize Interpolation Filter (filtinit.c)

```
/***************************************************************************

  This subroutine performs the following :
       1- Read filter coefficients from file to filter buffer
       2- Zerro pad the filter buffer to the size of interpolated data
          buffer size
       3- Perform Real Forward FFT of filter buffer to prepare for
          interpolation

/***************************************************************************/
#include <stdio.h>
#include "focus.h"
#include "focus.var"

extern struct cstype *cs;

void filter_init()
{
 void xfrf_();
 long i;

/* zero pad filter coeff                    */
 for(i=FILTER_LENGTH;i<NPOINT_DATA_INTER;i++)
   {
   cs->filter_coef[i]=0.;
   }

/* take fft of filter coef for each supercard */
 for(i=0;i<NO_SUPERCARD;i++)
   {
   xfrf_(cs->filter_fft,&cs->nyquist_filter,
        cs->filter_coef,&cs->n_data_inter_half);
   }

}
```

B.5. Main Back Projection Focusing Routing(bp.c)

```
#include <math.h>
```

```
/***************************************************************************
   Subroutine: poly.c
   DEscription:
            Calculates values of polynomials
   Input:    x[]       input array
            n          number of elements
            coeff[]    coefficients of poly
            deg        degree of poly
  Output:    y[]       output array
***************************************************************************/

poly(x,y,n,coeff,deg)
 float *x,*y,*coeff;
 long n,deg;
 {
  long i,j;
  for(i=0;i<n;i++)
   {
    y[i]=coeff[0];
    for(j=1;j<(deg+1);j++)
     {
      y[i]=y[i]+coeff[j]*pow(x[i],(float)j);
     }
   }
 }


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

    Subroutine: poly2.c
    Description: calculate y=c0+c1*x+c2*x^2
             where x=[0,1,2,...,n-1]
    Input:
        float c[3]     coefficients
        long n (max : 1000)  number of points
    Output:
       float y[]     output vector
***************************************************************************/

#define MAX_ELEMENT  1000

void poly2(y,n,c)
float y[],c[];
long n;
{
 void xvrmp_();
```

```
  void xdintg_();
  float temp;
  float y1[MAX_ELEMENT];
  float y1_init,y1_inc;
  y1_init=c[1]-c[2];
  y1_inc=2.*c[2];
  xvrmp_(y1,&temp,&y1_init,&y1_inc,&n);
  y1[0]=0.;
  xdintg_(y,&c[0],y1,&temp,&n);
 }



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

 Two-dimensional  back projection  subroutine
 using fast algorithm by Nuttal to calculates the index to data
 array for each pixel on the ground in i_box,k_box positioned by i,k
 k: azimuth ( 2nd order)
 i: range   ( ist order)
 The equation for index calculation is
      index= c0+c1*k+c2*k^2 + (c3+c4*k+c5*k^2)*i
          =      d0       +       d1*i
      where c(i) (i=0,5) is a set of coefficients for each box and
           each position in the aperture
      Input:
           float *pixel   pointer to first point of the image
                              2-dimensional area
           float *data    pointer to data array
                     Important: data[0]=0.0
                              data[dat_size-1]=0.0
           long  dat_size size of data buffer
           float *c       pointer to  six  coefficients for
                                 index computation
           long  k_pix_size   size of the patch (pixels) in k axis
           long  i_pix_size   size of the patch (pixels) in i axis
           long  k_box    patch number in k axis
           long  i_box    patch number in i axis
           long  i_box_size   number of patches in i axis
      Output:
           index to data array is calculated and pixel is updated

*************************************************************************/
#define MAX_K_PIX_SIZE   1000
#define MAX_I_PIX_SIZE   1000
```

```
void bp(pixel,data,dat_size,c,k_box_base,k_box,i_box,k_pix_size,
      i_pix_size,i_box_size)

float *pixel;    /* array of pixels of the whole 2-dimentional area      */
float *data;     /*  data array                                     */
long  dat_size;  /* size of data array                                */
float *c;        /* pointer to  six  coefficients for index computation   */
long  k_box_base; /* 0 for 1st SC, i*NBOX_AZIMUTH/NO_SUPERCARD for ith SC  */
long  k_box;     /* patch number in k axis                            */
long  i_box;     /* patch number in i axis                            */
long  k_pix_size; /* size of the patch (pixels) in k axis               */
long  i_pix_size; /* size of the patch (pixels) in i axis               */
long  i_box_size; /* number of patches in i axis                        */

{
 void xvclip_();
 void vclip_();
 void xvfx4_();
 void fix4_();
 void xvrmp_();
 void vindex_();
 void vramp_();
 void vgathr_();
 void vadd_();
 void vtabi_();

float findex[MAX_I_PIX_SIZE];   /* data index value                      */
long  lindex[MAX_I_PIX_SIZE];   /* data index value (round to integer)   */
float pix_temp[MAX_I_PIX_SIZE]; /* temp buffer for back projection       */
float d0[MAX_K_PIX_SIZE];       /* d0= c0+c1*k+c2*k^2                     */
float d1[MAX_K_PIX_SIZE];       /* d1= c3+c4*k+c5*k^2                     */
float *pix_index;  /* pointer to current pixel                          */
long  pix_index_inc; /* each time k is incremented , pixel pointer is jumped */
long  k;          /* for k and i loops control                          */
float maxindex;       /* maximum value for data index                    */
float zero=0.0;
float one=1.0;
long i_one=1;
float temp;

maxindex=(float)(dat_size-1);

pix_index=pixel+(k_box-k_box_base)*k_pix_size*i_box_size*
      i_pix_size+i_box*i_pix_size;
                  /* index of first point of the patch   */
```

```
                    /* with respect to pixel            */
  pix_index_inc=i_pix_size*i_box_size; /* index increment along k axis */

  poly2(d0,k_pix_size,c);      /* generate array of d0                 */
  poly2(d1,k_pix_size,c+3);    /* generate array of d1                 */

  for(k=0;k<k_pix_size;k++)
    {

 /* generate floating point index vector */
   vramp_(&d0[k],&d1[k],findex,&i_one,&i_pix_size);

 /* table look up and interpolate        */
 /* this function replaces xvclip(),xvfx4(),vgathr()   */
 /*
 vtabi_(findex,&i_one,&one,&zero,data,pix_temp,&i_one,&dat_size,&i_pix_size);
 */

 /* clip index */
   vclip_(findex,&i_one,&zero,&maxindex,findex,&i_one,&i_pix_size);

 /* convert to integer index */
   fix4_(findex,&i_one,lindex,&i_one,&i_pix_size);

 /* gather data into temporary buffer */
   vgathr_(data,lindex,&i_one,pix_temp,&i_one,&i_pix_size);

 /* update pixel array with data from temporary buffer */
   vadd_(pix_index,&i_one,pix_temp,&i_one,pix_index,&i_one,&i_pix_size);

 /* pix_index jumps along k axis  */
   pix_index=pix_index+pix_index_inc;

   } /* k loop */


 } /* subroutine */
```

B.6. Declare All Global Constants (Focus.h)

```
#define NO_SUPERCARD    3          /* number of supercards      */
#define NPOINT_AZIMUTH  600          /* number of bearing lines */
#define NPOINT_RADIAL   4095        /* number of radial lines   */
#define NPOINT_APER     2304       /* # of positions in aperture */
#define NPOINT_DATA     2048       /* number of original data points */
```

```
#define NPOINT_DATA_INTER  NPOINT_DATA*8 /* number of data points after
                                    interpolated */
#define FILTER_LENGTH   255        /* length of FIR filter    */
#define PI 3.141592654
#define RADAR_HEIGHT  60.          /* radar height in feet    */
#define A_OFSET      -15.0       /* offset angle from center line */
#define A_SPAN       54.0       /* spanning angle of the patch */
#define RMINCENTER    550.         /* range from center position to
                        reference point on tha patch */
#define RCENTER_REF   802.0       /* range (ft) of ref. point from center pos. */
#define THETA_CENTER_REF  -15.0 /* angle (deg)of ref. point from center pos. */
#define SAMPLING_RATE 2.0e09       /* sampling frequency of signal  */
#define SAMPLING_PERIOD 5.0e-10     /* ts=1/fs                 */


#define BOX_SIZE_RADIAL     195
#define BOX_SIZE_AZIMUTH    100
#define NBOX_RADIAL    NPOINT_RADIAL/BOX_SIZE_RADIAL
#define NBOX_AZIMUTH  NPOINT_AZIMUTH/BOX_SIZE_AZIMUTH
#define BOX_BASE_START  2
#define NO_KBOX_PROCESS  3
#define IBOX_START      10
#define NO_IBOX_PROCESS  1

#define SECTION_SIZE 4            /* # of sections for one aperture */
#define COEF_SIZE    6           /* # of coeffs for curve fit   */
#define MAXDEG      3           /* maximum degree for poly. fit */
#define NPOINT_BOXSAMPLE_RADIAL 5    /* # of samples in a box in radial
                        direction          */
#define NPOINT_BOXSAMPLE_AZIMUTH 5  /* # of samples in azimuth direction
                        for every box          */
#define NPOINT_BOXSAMPLE
NPOINT_BOXSAMPLE_RADIAL*NPOINT_BOXSAMPLE_AZIMUTH
                     /* # of samples in a box for curve fit */
```

B.7. Declare All Global Variables (Focus.var)

```
struct complex { float r,i;};            /* define a complex type     */

struct cstype {

/* Data Variables */
float pixel[NPOINT_AZIMUTH/NO_SUPERCARD][NPOINT_RADIAL]; /* array of
image*/
float data_temp[NPOINT_DATA];  /* temp buffer for data              */
float data[NPOINT_DATA];             /* original data bufer      */
float filter_coef[NPOINT_DATA_INTER];       /* filter coefficients       */
float data_inter[NPOINT_DATA_INTER+136];    /* interpolated data buffer  */
float tstart_err[NPOINT_APER]; /* tstart-tstart_trunc for each position */
struct complex data_inter_fft[NPOINT_DATA_INTER/2];  /* Real->Complex Forward FFT
                                   of data_inter       */
struct complex filter_fft[NPOINT_DATA_INTER/2];    /* Real->Complex Forward FFT
                                   of filter_coef       */
float nyquist_filter;       /* value of FFT of filter at Nyquist point  */
float nyquist_data;         /* value of FFT of data at Nyquist Point    */
long ham_flag;              /* to indicate wheter to use hamming or not */
float ham_coef[NPOINT_APER];  /* hamming weight coefficients            */


/* Geometry Variables */
float a_ofset;              /* offset angle from the center line       */
float a_span;               /* spanning angle of the patch             */
float d_theta;              /* delta angle                    */
float x,theta;              /* coordinate of a pixel                */
float c_theta,s_theta;      /* cosine and sine of theha               */
float d;                    /* distance from radar to center position
                            positive to the right, neg. to the left  */
float rcenter,rmincenter,rmaxcenter; /* range from center position          */
float rcenter_ref;          /* range from center pos. to ref point     */
float theta_center_ref;     /* angle of ref. point from center position */
float r_ref;                /* range from any pos. to ref. point       */
float d_rcenter;            /* sampling distance from center position  */
float dr8;                  /* (sampling distance)/8                */
float r,rmin,rmax;          /* range from any position                 */
float aper_length;          /* length of aperture                   */
float radar_height;         /* height of radar                      */
float radar_height2;        /* square the height of radar              */

/* Radar Variables */
float c;                    /* speed of wave                  */
float ts;                   /* sampling period of signal            */
float fs;                   /* sampling frequency of signal          */

long n_azimuth;             /* # of points in azimuth direction        */
long n_radial;              /* # of points in radial direction        */
long n_aper;                /* # of positions in aperture           */
long pix_size;              /* # of points in pixel array           */
long n_data;                /* number of original data points          */
long n_data_inter;          /* number of interpolated data points      */
long n_data_inter_half;     /* 1/2 # of interpolated data points       */
long n_boxsample;           /* # of samples in a box for curve fit     */
long n_coef;                /* # of coefficients curve fit          */
```

Distribution

ADMINISTRATOR
COMMANDER DEFENSE TECHNICAL INFORMATION CENTER
AF ROME AIR DEVELOPMENT CENTER
ATTN DTIC-DDA (2 COPIES RADC/RBC CAMERON STATION, BUILDING 5
ATTN M. WICKS ALEXANDRIA, VA 22304-6145 ATTN P. VAN ETTEN

OFFICE OF THE SECRETARY OF DEFENSE
COMMANDER DDR&E/RESEARCH & ADV. TECHNOLOGY
PENTAGON, WASHINGTON D.C. 20301
ATTN COL. BARRY CRANE

US ARMY MISSILE COMMAND ELECTRONIC SYSTEMS
ATTN AMSMI- - - , J. LOOMIS
ATTN AMSMI- - - , C. KROLINGER