GODDARD
GRANT
IN-43-CR
166487
38P.

Annual Report

Under
Contract # NAG - 5 - 1010
National Aeronautics and Space Administration

Improvement and extension of a radar forest backscattering model

Prepared by:
David S. Simonett
Yong Wang

(NASA-CR-183259) IMPROVEMENT AND EXTENSION OF A RADAR FOREST BACKSCATTERING MODEL Annual Report (California Univ.) 38 p N89-11292
CSCL 02F Unclas
G3/43 0166487

DEPARTMENT OF GEOGRAPHY
UNIVERSITY OF CALIFORNIA SANTA BARBARA

September 20, 1988

ANNUAL REPORT

1. Introduction:

Research to-date has focused on modeling development and programming based on model components we proposed during the past several months and research progress made by Simonett's team. In this report, we summarize the model components and programs (in C language under UNIX) finished to-date. These model components may help explain the contributions of various vegetation structural components to the attenuation and backscattering of vegetated surfaces to extract useful data concerning forest stands and their underlying surfaces for both the seawater-on and seawater-off cases.

2. Model components*:

There are six model components as shown in Figure 1, which are described as follows:

1 Direct backscattering from ground surface (water-off case only) with attenuation from upper and lower canopies (short form *dbs*).

2 Direct volume scattering from upper tree canopy (short form *vsu*).

3 Direct volume scattering from lower tree canopy with attenuation from the upper tree canopy (two way) (*vsl*).

4 Interaction of ground / trunk forward reflection with attenuation from upper and lower tree canopies (two way) (*tg*).

5 Interaction of ground / lower tree canopy forward scattering with attenuation from upper tree canopy (two way) (*glc*).

6 Interaction of Ground / upper tree canopy forward scattering with attenuation from lower tree canopy (two way) (*guc*).

3. Programs:

head.h: a header file in which all the constants and external variables are defined. These constants and variables will be used in the whole program of modeling radar backscattering from forest (Sundri and Gewa) stands for the Bangladesh research project.

*note: Upper canopy is the canopy of trees whose dbhs are >= 5.0 cm. Lower canopy is the canopy of trees whose dbhs are < 5.0 cm, which are also called as regeneration trees.

main.c: the main program, detailed as follows:

NAME

SIGMA - radar returns of forest stands.

SYNOPSIS

SIGMA intab indata inpxl indbh sts_o t_o dbs_o vsu_o vsl_o tg_o glc_o guc_o

DESCRIPTION

Random tree dbhs based on numbers of each dbh data segments in one hectare (data from Dr. Imhoff), and tree positions are generated with uniform distribution in a stand, which is defined as an area of 10 hectares or 160 pixels (each pixel with size 25 * 25 m). Radar returns are simulated for each model component in each pixel. The returns are further averaged to get returns of each pixel and of a stand.

SIGMA is executable.

intab file is a look-up table of radar cross sections of dielectric cylinders derived from exact solutions for Sundri and Gewa. This table covers one angle and tree dbh from 5.0 to 39.9 cm with 0.1 cm as a dbh step.

indata file contains parameters needed for simulating a forest stand with optional ground conditions and optional solutions of model components.

inpxl file is the input data, which has 14 columns of dbh segments (5 -- 9.9, 10 -- 14.9, 15 -- 19.9, 20 -- 24.9, 25 -- 29.9, 30 -- 34.9, and 35 -- 39.9) for Sundri and Gewa in one pixel corresponding to tree numbers of each dbh segment in each pixel. The tree positions or (x, y) coordinates are also randomly generated with uniform distribution in a stand.

indbh file contains all the data (dbh >= 5cm) with one data number per line for Sundri and Gewa in the whole stand.

sts_o is a output file of statistical data (means and standard derivations) for each model component, and the sum of these components in a stand, respectively.

t_o file summarizes radar returns of those components in each pixel.

*The rest *_o files* are outputs corresponding to six radar backscattering components (1 to 6) mentioned above, respectively.

NOTE

*FILE pointers, *fptab, *fpin, *fppxl, *fpdbh, *fps, *fpt, *fp1, *fp2, *fp3, *fp4, *fp5, and *fp6,* point to intab, indata, inpxl, indbh, sts_o, t_o, dbs_o, vsu_o, vsl_o, tg_o, glc_o, and guc_o.

The following are the subroutines called by the main.c program.

head.c: defines all the external variables employed in the head.h file.

indata.c: input parameters to choose model options and different surface conditions. Also, prints out these input parameters to check that the parameters are input correctly.

openfl.c: open files to read and write. Totally, there are 12 files opened, in which four files are input and eight files are output.

phs_x.c: random phases and/or x coordinates (distances in range direction) for Sundri and Gewa are generated by using random() functions in the UNIX system, which are employed to count phase differences when summing each trunk / surface interaction terms.

equi.c: calculate an equivalent value for two values x and y, weighted by their numbers n1 and n2.

ref.c: Fresnel reflection coefficient (both magnitude and phase) of H or V polarization incidence from eqns (9.1 - 49) and (9.1 -46) (Ruck, et al., 1970).

gama.c:* incoherent component of forward scattering coefficient from a very rough surface using equations 9.7-118 - 132, in Ruck et al. v. 2, pp. 720-722.

beta.c:* β parameters for a very rough surface scattering, which is defined in equations 9.1 - 124 to 9.1 - 132 (p. 722, Ruck, et al., 1970).

biomas.c: tree biomass in one pixel (25 * 25 m), which is calculated based on data from Dr. Imhoff. The biomass data is employed to compute attenuation coefficient of trees (Sundri and Gewa) for each pixel. Furthermore, the biomass data varies with pixels as a result of

*note: these programs are not used in this recent Bangladesh research project, but were written for the sake of completeness and for extension to different roughnesses of ground surfaces, in the future.

ORIGINAL PAGE IS
OF POOR QUALITY

tree number change for each pixel.

tau.c: the attenuation coefficient (τ) of a canopy is calculated by using equations in Battan (1970, p. 67) eqns 6.5 and 6.6. The attenuation coefficient is equal to a scattering coefficient plus an absorption coefficient.

ep.c: calculate dielectric constants (ε' and ε'') (note: $\varepsilon = \varepsilon' - j\,\varepsilon''$) of mixture layers (the air as a host material, and materials, such as leaves, branches, trunks, etc. as inclusions) based on the formula in Ulaby, et al., (1986, E.62). The reflective model is chosen.

vsw.c: canopy volume scattering model (Attema and Ulaby, 1978). Further effort will be made in 1989 to improve the simulation of the canopy component by extending this model component to include the disk model for broad-leaf evergreen forest (Eom and Fung, 1984; Lang and Saleh, 1985; Le Vine, et al., 1982, 1985).

tge.c: radar forward scattering coefficient per unit area is derived from exact solutions of trunks (Sundri and Gewa), which are treated as dielectric cylinders with finite lengths and with smooth trunk surfaces.

tgc.c: is the corner reflector model for tree trunks of Sundri and Gewa, which is the same as that used by Richards, et al. (1987).

surI.c: backscattering coefficients per unit area from a slightly rough surface using a small perturbation model (Dobson and Ulaby, 1986), when the ground surface is in the water-off condition.

surII.c: backscattering coefficients per unit area from a very rough surface using eqs. 9.1-139 and 141, in Ruck et al., vol. 2., p. 721 - 725. This program is again not used for the Bangladesh research project, but has been written for completeness in the whole modeling process.

ee.c: interaction of canopy - surface scattering per unit area using Engheta and Elachi's model (1982) by means of finding σ^o in terms of the water cloud model (Attema and Ulaby, 1978). This formula is similar to that used in Sun and Simonett (1988).

4. Accessory programs:

The following are accessory programs, which are used to provide necessary input data in the correct format required by the above programs, and to maintain them handily.

dbh.c: produces all the DBHs of total tree numbers (Sundri and Gewa) in a stand area, which are generated by the random() function in the UNIX system with random uniform distribution. The input number is the total number of the dbhs*.

po_tree.c: random (x, y) positions of total Sundri and Gewa trees in a stand are generated by cutting the whole output data of the random() function in the UNIX system into two parts. The first half of the data set is used to yield x coordinates, and the second half y coordinates. Twice the number of the positions are input. The (x, y) positions are designated in centimeter units. The stand has an area of *N* hectares.

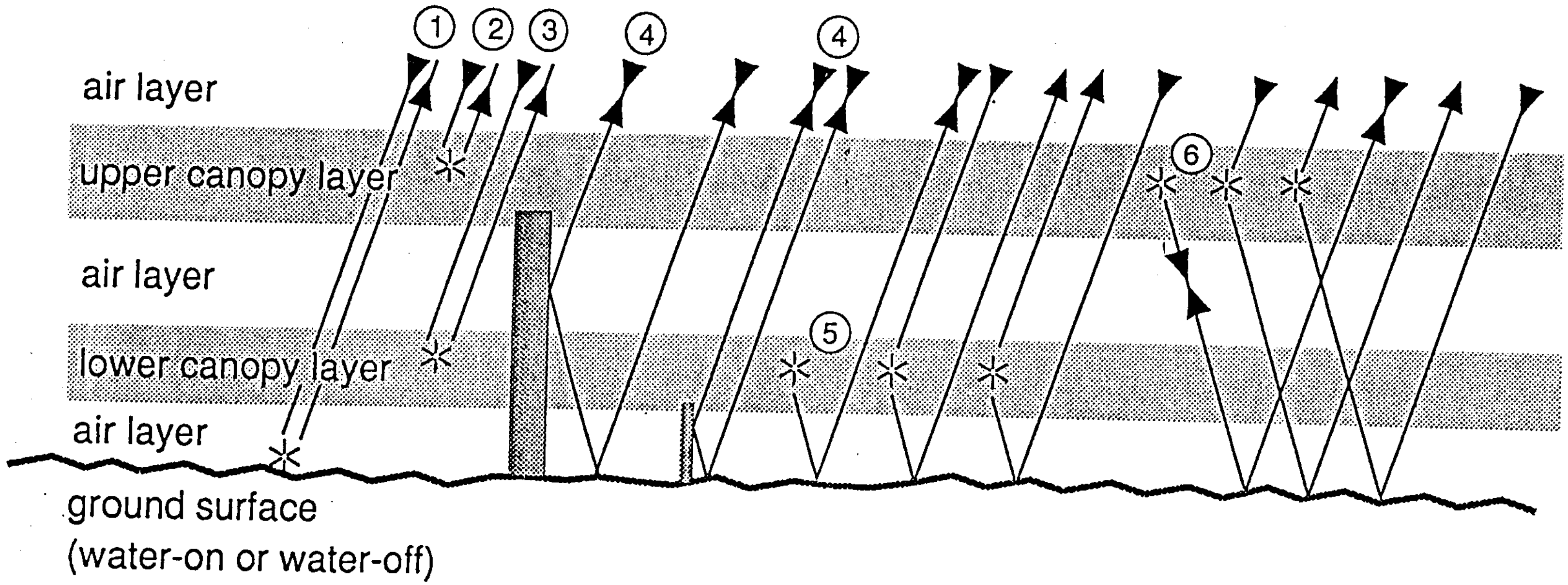
po_num.c: uses the output of above (x, y) coordinate data as its input to produce the tree numbers of each dbh segment respectively for each simulated pixel in the whole stand. To run the program, tree number (TOTAL) for each dbh segment in a hectare and the position data should be in one input data file. The first line of input data is TOTAL and the rest of the data lines (N x TOTAL) are positions or (x, y) data, where N is a constant and can be changed for different stand sizes. In our case, the N is 10.

makefile: maintains the above programs, which is based on the make command on the UNIX system.

RUN: is an executable macro. Instead of typing the "SIGMA intab indata inpxl indbh sts_o t_o dbs_o vsu_o vsl_o tg_o glc_o guc_o" for each model being run, just type RUN, which does the same job as that inside the double quotation.

ORIGINAL PAGE IS
OF POOR QUALITY

*note: all the dbhs range from 0.0 to 4.9 cm. Constants 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, and 35.0 are added to the dbh data set to produce dbhs in the following ranges (5.0 -- 9.9, 10.0 -- 14.9, 15.0 -- 19.9, 20.0 -- 24.9, 25.0 -- 29.9, 30.0 -- 34.9, and 35.0 -- 39.9 cm) in the main.c program, respectively.



ORIGINAL PAGE IS
OF POOR QUALITY

Fig. 1 Model components.

```c
/*      a header file in which all the constants and external variables
**      are defined.  These constants and variables are used in the
**      whole program of modeling radar backscattering from forests.
*/


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

#define PI      3.141593

#define MAX     30000   /* # of trees in a stand        */

#define MAX1    1000    /* # of trees in a pixel        */

#define TAB     350     /* size of look-up tables       */

#define M_CM    100.0   /* 1 m == 100.0 cm              */

/*      FILE pointers of input and output files
*/
FILE    *fpdbh, *fptab, *fpin, *fppxl;
FILE    *fps, *fpt, *fp1, *fp2, *fp3, *fp4, *fp5, *fp6;

/*      model and output options:
        te      1: calculate tau using Battan equations (p. 67, 1970)
                2: use inputed tau

        vs      1: water cloud model

        tg      1: exact solutions of trunk forward scattering coefficients
                2: corner reflector

        ph      1: random phase
                2: phase due to range differences

        kf      1: seawater surface
                2: slightly rough surface
                3: Gaussian type distribution for very rough surface
                4: Exponential type distribution for very rough surface

        lo      1: 100% Sundri as lower canopy layer
                2: 70% Sundri and 30% Gewa as lower canopy layer
                3: 30% Sundri and 70% Gewa as lower canopy layer
                4: 100% Gewa as lower canopy layer

        ot      1: print detail information for analysis
                2: print plotting data only
*/
extern  int     te, vs, tg, ph, kf, lo, ot;

/*      model option definitions
*/
extern  char    *cte[];
extern  char    *can[];
extern  char    *trk[];
extern  char    *phs[];
extern  char    *sur[];
extern  char    *low[];
extern  char    *pol[];

/*      tree dbhs (>= 5 cm || <= 39.9 cm) of Sundri and Gewa in one pixel
*/
extern  double  DBH_s[], DBH_g[];

/*      look-up tables of exact solutions of forward scattering
        coefficients for Sundri and Gewa trunks
*/
extern  double  TAB_s[], TAB_g[];

/*      random phases of Sundri and Gewa in one stand
*/
extern  double  PHS[];

/*      x coordinates of Sundri and Gewa in one stand
*/
extern  double  X[];

/*      np:     # of pixels to be simulated in a forest stand
        rg, rz: range and azimuth resolutions
                area: pixel size (rg * rz)
*/
extern  int     np;
extern  double  rg, rz, area;

/*      radar parameters: wvl: wavelength
                          pl: polarization(1-HH, 2-HV, 3-VH, 4-VV)
                          Theta: incidence angle (in degree)
                                theta:  incidence angle (in radian)
                                k:      radar wavenumber
*/
extern  int     pl;
extern  double  Theta, theta;
extern  double  wvl, k;

/*      dielectric constants of trunks and leaves
        Sundri: eptr_s, epti_s, eplr_s, epli_s
        Gewa: eptr_g, epti_g, eplr_g, epli_g
*/
extern  double  eptr_s, epti_s, eplr_s, epli_s;
extern  double  eptr_g, epti_g, eplr_g, epli_g;

/*      ground surface and seawater dielectric constants:
        ground surface: epsr, epsi
        seawater: epwr, epwi
*/
extern  double  epsr, epsi, epwr, epwi;

/*      coefficients of regression models: y = A + B * x,
        A1, B1: tree height (m) on dbh (cm) for Sundri
        A2, B2: canopy depth (m) on dbh (cm) for Sundri
        A3, B3: tree height (m) on dbh (cm) for Gewa
        A4, b4: canopy depth (m) on dbh (cm) for Gewa
*/
extern  double  A1, B1, A2, B2, A3, B3, A4, B4;

/*      tau_u: extinction coefficient of the upper canopy
        tau_l: extinction coefficient of the lower canopy
*/
extern  double  tau_u, tau_l;

/*      f_u: a ratio of eta_u to tau_u of the upper canopy
        f_l: a ratio of eta_l to tau_l of the lower canopy
*/
extern  double  f_u, f_l;

/*      rat_l: a ratio of the material to the air of the lower canopy
        rat_u: a ratio of the material to the air of the upper canopy
```

```
*/
extern  double  rat_l, rat_u;

/*      h: mean-square root roughness height of ground surface
        l: correlation length of ground surface
*/
extern  double  h, l;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```c
/*      NAME
**              SIGMA - radar return of a forest stand
**
**      SYNOPSIS
**              SIGMA intab indata inpxl indbh sts_o t_o dbs_o vsu_o \
**              vsl_o tg_o glc_o guc_o
**
**      DESCRIPTION
**
**                      Random tree dbhs and positions, based on tree
**              numbers of each dbh segment in one hectare, are
**              generated with uniform distribution in a stand, which
**              has an area of 10 hectares or 160 pixels (each pixel
**              with size 25 * 25 m).  Radar returns are simulated for
**              each model component in each pixel.  The returns are
**              further averaged to get returns of each pixel and of a
**              stand.
**
**                      SIGMA is executable.
**
**                      intab file is a look-up table of forward
**              scattering coefficient of dielectric cylinders derived
**              from exact solutions for Sundri and Gewa trunks.  This
**              table covers one angle and tree dbh from 5.0 to 39.9 cm
**              (with 0.1 cm as a dbh step).
**
**                      indata file contains parameters needed for
**              simulating a forest stand with optional ground
**              conditions and optional solutions of model components.
**
**                      inpxl file is the input data, which has 14
**              columns of dbh segments (5 -- 9.9, 10 -- 14.9,
**              15 -- 19.9, 20 -- 24.9, 25 -- 29.9, 30 -- 34.9, and
**              35 -- 39.9) for Sundri and Gewa in one pixel,
**              corresponding to tree numbers of each dbh segment in
**              that pixel.  The tree positions or (x, y) coordinates
**              are also randomly generated with uniform distribution
**              in a stand.
**
**                      indbh file contains all the dbh data with one
**              data number per line for Sundri and Gewa in the whole
**              stand.
**
**                      sts_o is a output file of statistical data
**              (means and standard derivations) for each model
**              component, and the sum of these components in a stand.
**
**                      t_o file summarizes radar returns of those
**              components in each pixel.
**
**                      The rest *_o files are outputs corresponding to
**              six radar backscattering components, respectively.
**
**      NOTE
**                      FILE pointers, *fptab, *fpin, *fppxl, *fpdbh,
**              *fps, *fpt, *fp1, *fp2, *fp3, *fp4, *fp5, and *fp6,
**              point to intab, indata, inpxl, indbh, sts_o, t_o,
**              dbs_o, vsu_o, vsl_o, tg_o, glc_o, and guc_o.
**
**      AUTHORS
**                      This program was written by  Y. Wang and G. Sun.
*/

#include        "head.h"

/*      regeneration biomass (g / m / m). (data from Dr. Imhoff)
*/
#define CON     11415.1

/*      regeneration tree number (# / pixel). (data from Dr. Imhoff)
*/
#define NUM     2744

/*      each dbh segment constant (cm)
*/
#define SEG1    5.0
#define SEG2    10.0
#define SEG3    15.0
#define SEG4    20.0
#define SEG5    25.0
#define SEG6    30.0
#define SEG7    35.0

/*      mean of 0.0 and 4.9 (cm) dbh segment
*/
#define M_L     2.45

main(argc, argv)

        int     argc;
        char    *argv[];

{

/*      reflection coefficients of Sundri and Gewa trunk surfaces
*/
        double  rmt_s, rpt_s, rmt_g, rpt_g;

/*      reflection coefficients of the smooth, the rough, and the very
        rough surfaces (no phase term for very rough surface)
*/
        double  rms, rms_r, rms_v;
        double  rps, rps_r;

/*      as_s: trunk shadow area of Sundri
        as_g: trunk shadow area of Gewa
        as: total shadow areas (as_s + as_g)
*/
        double  as_s, as_g, as;

/*      weighted leaf dielectric constant of Sundri and Gewa
*/
        double  eplr, epli;

/*      me_can_s: mean depth of upper Sundri canopy
        me_can_g: mean depth of upper Gewa canopy
        me_can_u: weighted mean canopy depth of the upper canopy
*/
        double  me_can_s, me_can_g, me_can_u;

/*      weighted mean depth of lower canopy
*/
        double  me_can_l;

/*      eta_u: volume scattering coefficient of the upper canopy
        eta_l: volume scattering coefficient of the lower canopy
*/
```

```c
	double	eta_u, eta_l;

	double	TAU_U, TAU_L;

/*	M: biomass (g / m / m)
*/
	double	M;

/*	# of trees for each dbh segment of Sundri and Gewa
*/
	int	s_5, s_10, s_15, s_20, s_25, s_30, s_35;
	int	g_5, g_10, g_15, g_20, g_25, g_30, g_35;

/*	numbers of Sundri and/or Gewa (dbh >= 5 cm) in one pixel
*/
	int	n_sun, n_gewa, num;

/*	backscattering coefficients of a component
*/
	double	sigm;

/*	total backscattering coefficient in a pixel
*/
	double	sigma;

/*	means and stds of sum of all the components, and each component
*/
	double	me_tot = 0.0, std_tot = 0.0;

	double	me_dbs = 0.0, std_dbs = 0.0;
	double	me_vsu = 0.0, std_vsu = 0.0;
	double	me_vsl = 0.0, std_vsl = 0.0;
	double	 me_tg = 0.0,  std_tg = 0.0;
	double	me_glc = 0.0, std_glc = 0.0;
	double	me_guc = 0.0, std_guc = 0.0;

	int	i, j;

	double	tmp;
	double	thet;

	double	surI(), surII(), vsw(), tge(), tgc(), ee();

/* open files to input and output	*/
	openfl(argc, argv);

/* input data for modeling	*/
	indata();

/* random phases and random x coordinates for Sundri and Gewa	*/
	phs_x();

/* ratios of volume scattering coefficient to attenuation
   coefficients of lower and upper canopies		*/
	fscanf(fpin, "%lf %lf", &f_l, &f_u);

	if (ot == 1)
		printf("f_l = %lf, f_u = %lf\n\n", f_l, f_u);

/* reflection coefficients of Sundri and Gewa trunks
*/
/* thet: an angle from the normal direction of the trunk surface */
	thet = 0.5 * PI - theta;

	if (pl == 1)	{
		rfhh(thet, eptr_s, epti_s, &rmt_s, &rpt_s);
		rfhh(thet, eptr_g, epti_g, &rmt_g, &rpt_g);
	}
	else if (pl == 4)	{
		rfvv(thet, eptr_s, epti_s, &rmt_s, &rpt_s);
		rfvv(thet, eptr_g, epti_g, &rmt_g, &rpt_g);
	}
	else if (pl == 2 || pl == 3)	{
		rmt_s = 0.0;
		rpt_s = 0.0;
		rmt_g = 0.0;
		rpt_g = 0.0;
	}
	else	{
		printf("pola can't be %d", pl);
		exit(1);
	}

	if (ot == 1)	{
		printf("rmt_s = %lf, rpt_s = %lf\n", rmt_s, rpt_s);
		printf("rmt_g = %lf, rpt_g = %lf\n\n", rmt_g, rpt_g);
	}

/* reflection coefficients from seawater surface
*/
	if (kf == 1)	{
		if (pl == 1)
			rfhh(theta, epwr, epwi, &rms, &rps);
		else if (pl == 4)
			rfvv(theta, epwr, epwi, &rms, &rps);
		else	{
			rms = 0.0;
			rps = 0.0;
		}

		if (ot == 1)	{
			printf("ref. coef. from seawater surface:\n");
			printf("rms = %lf, rps = %lf\n\n", rms, rps);
		}
	}

/* reflection coefficients from slightly rough surface
*/
	else if (kf == 2) {
		if (pl == 1)
			rfhh(theta, epsr, epsi, &rms_r, &rps_r);
		else if (pl == 4)
			rfvv(theta, epsr, epsi, &rms_r, &rps_r);
		else	{
			rms_r = 0.0;
			rps_r = 0.0;
		}

/* reflection coefficients (Ruck, et al., 1970, p. 700)	*/
		tmp = cos(theta) * cos(theta);
		rms_r *= exp(-2.0 * k * k * h * h * tmp);

		if (ot == 1)	{
			printf("ref. coef. from slightly rough surface:\n");
			printf("rms_r = %lf, rps_r = %lf\n\n", rms_r, rps_r);
		}
	}
```

```c
/* reflection coefficients from very rough surface
*/
        else if (kf == 3 || kf == 4) {
                gama(epsr, epsi, &rms_v);
                rms_v /= (4.0 * PI);
                rms_v = sqrt(rms_v);

                if (ot == 1)    {
                        printf("ref. coef. from very rough surface:\n");
                        printf("rms_v = %lf\n\n", rms_v);
                }
        }

        else    {
                printf("kf can't be %d\n", kf);
                exit(1);
        }

/* attenuation coefficients of lower and upper canopies
*/
        if (te == 1)    {

/* the attenuation coefficient of the lower canopy      */
        if (lo == 1)    {
                me_can_l = A2 + B2 * M_L;
                M = CON;
                num = NUM;
                tau(M, me_can_l, eplr_s, epli_s, num, rat_l, &tau_l);
        }
        else if (lo == 2)       {
                me_can_l = (A2 + B2 * M_L) * 0.7;
                me_can_l += (A4 + B4 * M_L) * 0.3;
                eplr = eplr_s * 0.7 + eplr_g * 0.3;
                epli = epli_s * 0.7 + eplr_g * 0.3;
                M = CON;
                num = NUM;
                tau(M, me_can_l, eplr, epli, num, rat_l, &tau_l);
        }
        else if (lo == 3)       {
                me_can_l = (A2 + B2 * M_L) * 0.3;
                me_can_l += (A4 + B4 * M_L) * 0.7;
                eplr = eplr_s * 0.3 + eplr_g * 0.7;
                epli = epli_s * 0.3 + eplr_g * 0.7;
                M = CON;
                num = NUM;
                tau(M, me_can_l, eplr, epli, num, rat_l, &tau_l);
        }
        else if (lo == 4)       {
                me_can_l = A4 + B4 * M_L;
                M = CON;
                num = NUM;
                tau(M, me_can_l, eplr_g, epli_g, num, rat_l, &tau_l);
        }

        else    {
                printf("lo can't be %d\n", lo);
                exit(1);
        }

        }

/* input an attenuation coefficient of the lower canopy
*/
        else if (te == 2)
                fscanf(fpin, "%lf", &tau_l);

        else    {
                printf("te can't be %d\n", te);
                exit(1);
        }

        if (ot == 1)
                printf("tau_l = %lf\n\n", tau_l);

/* an volume scattering coefficient of the lower canopy and a constant
*/
        eta_l = f_l * tau_l;
        TAU_L = exp(-2.0 * tau_l * me_can_l / cos(theta));

        /** each loop in the 'for' simulates one pixel  **/

        for (j = 0; j < np; j++)        {

/* input tree #s of following Sundri and Gewa dbh segments:
        5.0 -- 9.9, 10.0 -- 14.9, 15.0 -- 19.9, 20.0 -- 24.9,
        25.0 -- 29.9, 30.0 -- 34.9, and 35.0 -- 39.9 cm in a pixel
*/
        fscanf(fppxl, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d",
                &s_5, &s_10, &s_15, &s_20, &s_25, &s_30, &s_35,
                &g_5, &g_10, &g_15, &g_20, &g_25, &g_30, &g_35);

        if (ot == 1) | {
                printf("trees (each dbh segment) in pixel %d:\n", j + 1);
                printf("Sundri: %d %d %d %d %d %d %d\n",
                        s_5, s_10, s_15, s_20, s_25, s_30, s_35);
                printf("Gewa: %d %d %d %d %d %d %d\n\n",
                        g_5, g_10, g_15, g_20, g_25, g_30, g_35);
        }

/* Sundri dbh data in a pixel   */
        n_sun = s_5;
        for (i = 0; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG1;
        }

        n_sun += s_10;
        for (; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG2;
        }

        n_sun += s_15;
        for (; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG3;
        }

        n_sun += s_20;
        for (; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG4;
        }

        n_sun += s_25;
        for (; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG5;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        }

        n_sun += s_30;
        for (; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG6;
        }

        n_sun += s_35;
        for (; i < n_sun; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_s[i] = tmp + SEG7;
        }

        if (ot == 1)
                printf("# of Sundri in the pixel: %d\n\n", n_sun);

/* the mean depth of Sundri upper canopy in a pixel derived from
   a canopy regression model
*/
        if (n_sun == 0) {
                me_can_s = 0.0;
                as_s = 0.0;
        }
        else    {
                for (me_can_s = 0.0, i = 0; i < n_sun; i++)
                        me_can_s += DBH_s[i];

                me_can_s /= (double) n_sun;
                me_can_s = A2 + B2 * me_can_s;  /* the reg. model  */

/* trunk shadow area (in square meters) of Sundri in one pixel
   derived from a regression height model
*/
                for (as_s = 0.0, i =0; i < n_sun; i++)  {
                        tmp = A1 + B1 * DBH_s[i];  /* the reg. model  */
                        tmp -= (me_can_s / 3.0);   /* eff. height     */
                        as_s += (tmp * DBH_s[i] * tan(theta) / M_CM);
                }
        }

        if (ot == 1)
                printf("trunk shadow area of Sundri: %lf\n\n", as_s);

/* Gewa dbh data in a pixel     */
        n_gewa = g_5;
        for (i = 0; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG1;
        }

        n_gewa += g_10;
        for (; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG2;
        }

        n_gewa += g_15;
        for (; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG3;
        }

        n_gewa += g_20;
        for (; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG4;
        }

        n_gewa += g_25;
        for (; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG5;
        }

        n_gewa += g_30;
        for (; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG6;
        }

        n_gewa += g_35;
        for (; i < n_gewa; i++) {
                fscanf(fpdbh, "%lf", &tmp);
                DBH_g[i] = tmp + SEG7;
        }

        if (ot == 1)
                printf("# of Gewa in the pixel: %d\n\n", n_gewa);

/* the mean depth of Gewa upper canopy in a pixel derived from
   a canopy regression model
*/
        if (n_gewa == 0)        {
                me_can_g = 0.0;
                as_g = 0.0;
        }

        else    {
                for (me_can_g = 0.0, i = 0; i < n_gewa; i++)
                        me_can_g += DBH_g[i];

                me_can_g /= (double) n_gewa;
                me_can_g = A4 + B4 * me_can_g;  /* the reg. model  */

/* trunk shadow area (in square meters) of Gewa in one pixel
   derived from a height regression model
*/
                for (as_g = 0.0, i = 0; i < n_gewa; i++)        {
                        tmp = A3 + B3 * DBH_g[i];  /* the reg. model  */
                        tmp -= (me_can_g / 3.0);   /* eff. height     */
                        as_g += tmp * DBH_g[i] * tan(theta) / M_CM;
                }
        }

        if (ot == 1)
                printf("trunk shadow area of Gewa: %lf\n\n", as_g);

/* Sundri and Gewa shadow areas */
        as = as_s + as_g;

/* initialization       */
        sigma = 0.0;

/** if no trees in a pixel, calculate the dbs component directly with
    the attenuation of the lower canopy only     **/

        if (n_sun == 0 && n_gewa == 0)  {
```

ORIGINAL PAGE IS OF POOR QUALITY

```
        if (kf == 1)
                sigm = 0.0;
        else if (kf == 2)
                sigm = surI(rms_r);
        else
                sigm = surII(epsr, epsl) ;

        sigm *= TAU_L;

        if (sigm > 1.0e-10)     /* some sigma may be too small  */
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel   */
        fprintf(fp1, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_dbs += tmp;
        std_dbs += (tmp * tmp);

/* sigma of this pixel  */
        sigma += sigm;

        }

/** there is at least one tree in this pixel    **/
        else    {

/* an extinction coefficient of upper canopy
*/
        if (te == 1)    {

/* equivalent dielectric constant of leaves and mean canopy depth
of the upper canopy             */
                equi(eplr_s, n_sun, eplr_g, n_gewa, &eplr);
                equi(epli_s, n_sun, epli_g, n_gewa, &epli);
                equi(me_can_s, n_sun, me_can_g, n_gewa, &me_can_u);

                biomas(s_5, s_10, s_15, s_20, s_25, s_30, s_35,
                        g_5, g_10, g_15, g_20, g_25, g_30, g_35, &M);

                num = n_sun + n_gewa;   /* tree # in a pixel */
                tau(M, me_can_u, eplr, epli, num, rat_u, &tau_u);
        }

        else if (te == 2)
                fscanf(fpin, "%lf\n", &tau_u);

        if (ot == 1)
                printf("tau_u = %lf\n", tau_u);

/* volume scattering of the upper canopy and a constant
*/
        eta_u = f_u * tau_u;
        TAU_U = exp(-2.0 * tau_u * me_can_u / cos(theta));

        /** directly backscattering from unshadowed area in a pixel
        with attenuation of the upper and the lower canopies    **/

        if (kf == 1)
                sigm = 0.0;
        else if (kf == 2)
                sigm = surI(rms_r);
        else
                sigm = surII(epsr, epsl) ;

        sigm *= (TAU_U * TAU_L * (area - as) / area);

        if (sigm > 1.0e-10)
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel   */
                fprintf(fp1, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_dbs += tmp;
        std_dbs += (tmp * tmp);

/* sigma of this pixel  */
        sigma += sigm;

        /** volume scattering of the upper canopy **/

        if (vs == 1)
                sigm = vsw(tau_u, eta_u, TAU_U);
        else    {
                printf("vs can't be %d\n", vs);
                exit(1);
        }

        if (sigm > 1.0e-10)
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel   */
        fprintf(fp2, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_vsu += tmp;
        std_vsu += (tmp * tmp);

/* sigma of this pixel  */
        sigma += sigm;

        /** volume scattering of the lower canopy with attenuation
        of the upper canopy             **/

        if (vs == 1)
                sigm = vsw(tau_l, eta_l, TAU_L) * TAU_U;

        if (sigm > 1.0e-10)
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel   */
        fprintf(fp3, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_vsl += tmp;
        std_vsl += (tmp * tmp);

/* sigma of this pixel  */
```

```
        sigma += sigm;

        /** the interaction of trunk / ground surface   **/

/* exact solution       */
        if (tg == 1)
                if (kf == 1)
                        sigm = tge(n_sun, n_gewa, rms);
                else if (kf == 2)
                        sigm = tge(n_sun, n_gewa, rms_r);
                else
                        sigm = tge(n_sun, n_gewa, rms_v);

/* corner reflector     */
        else if (tg == 2)
                if (kf == 1)
                        sigm = tgc(n_sun, n_gewa, me_can_u,
                                rmt_s, rmt_g, rms);
                else if (kf == 2)
                        sigm = tgc(n_sun, n_gewa, me_can_u,
                                rmt_s, rmt_g, rms_r);
                else
                        sigm = tgc(n_sun, n_gewa, me_can_u,
                                rmt_s, rmt_g, rms_v);

        else {
                printf("tg can't be %d\n", tg);
                exit(1);
        }

/* with the attenuation of the upper and lower canopies */
        sigm *= (TAU_U * TAU_L);

        if (sigm > 1.0e-10)
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel   */
        fprintf(fp4, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_tg += tmp;
        std_tg += (tmp * tmp);

/* sigma of this pixel  */
        sigma += sigm;

        /** the interaction of ground surface / lower canopy with the
        attenuation of the upper canopy         **/

        if (kf == 1)
                sigm = ee(tau_l, me_can_l, eta_l, rms) * TAU_U;

        else if (kf == 2)
                sigm = ee(tau_l, me_can_l, eta_l, rms_r) * TAU_U;

        else
                sigm = ee(tau_l, me_can_l, eta_l, rms_v) * TAU_U;

        if (sigm > 1.0e-10)
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel    */
        fprintf(fp5, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_glc += tmp;
        std_glc += (tmp * tmp);

/* sigma of this pixel  */
        sigma += sigm;

        /* the interaction of ground surface / upper canopy
        with attenuation of the lower canopy    */

        if (kf == 1)
                sigm = ee(tau_u, me_can_u, eta_u, rms) * TAU_L;

        else if (kf == 2)
                sigm = ee(tau_u, me_can_u, eta_u, rms_r) * TAU_L;

        else
                sigm = ee(tau_u, me_can_u, eta_u, rms_v) * TAU_L;

        if (sigm > 1.0e-10)
                tmp = 10.0 * log10(sigm);
        else
                tmp = -100.0;

/* output this component in dB for each pixel    */
        fprintf(fp6, "%16.6e\n", tmp);

/* mean and std of this component in a stand    */
        me_guc += tmp;
        std_guc += (tmp * tmp);

/* sigma of one pixel after summing all the components  */
        sigma += sigm;

        }

/* sigma of this pixel (in dB)  */

        if (sigma > 1.0e-10)
                sigma = 10.0 * log10(sigma);
        else
                sigma = -100.0;

/* output the sum in dB for each pixel   */
        fprintf(fpt, "%16.6e\n", sigma);

/* mean and std in a stand      */
        me_tot += sigma;
        std_tot += sigma * sigma;

        }

        /**     one stand simulation is finished  **/

/* means and std of the sum of all the components in one stand
*/
        me_tot = me_tot / np;
        std_tot = sqrt((std_tot - np * me_tot * me_tot) / (np - 1));

/* directly backscattering from the ground surface (dbs)
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
*/
        me_dbs = me_dbs / np;
        std_dbs = sqrt((std_dbs - np * me_dbs * me_dbs) / (np -1));

/* volume scattering from the upper canopy (vsu)
*/
        me_vsu = me_vsu / np;
        std_vsu = sqrt((std_vsu - np * me_vsu * me_vsu) / (np - 1));

/* volume scattering from the lower canopy (vsl)
*/
        me_vsl = me_vsl / np;
        std_vsl = sqrt((std_vsl - np * me_vsl * me_vsl) / (np - 1));

/* the interaction of the trunk / surface (tg)
*/
        me_tg = me_tg / np;
        std_tg = sqrt((std_tg - np * me_tg * me_tg) / (np -1));

/* interaction of the ground / lower canopy (glc)
*/
        me_glc = me_glc / np;
        std_glc = sqrt((std_glc - np * me_glc * me_glc) / (np - 1));

/* the interaction of the ground / upper canopy (guc)
*/
        me_guc = me_guc / np;
        std_guc = sqrt((std_guc - np * me_guc * me_guc) / (np - 1));


        /***    final output of a forest stand data     ***/

        fprintf(fps, "me_tot = %16.6f, std_tot = %16.6f\n", me_tot, std_tot);
        fprintf(fps, "me_dbs = %16.6f, std_dbs = %16.6f\n", me_dbs, std_dbs);
        fprintf(fps, "me_vsu = %16.6f, std_vsu = %16.6f\n", me_vsu, std_vsu);
        fprintf(fps, "me_vsl = %16.6f, std_vsl = %16.6f\n", me_vsl, std_vsl);
        fprintf(fps, " me_tg = %16.6f,  std_tg = %16.6f\n", me_tg, std_tg);
        fprintf(fps, "me_glc = %16.6f, std_glc = %16.6f\n", me_glc, std_glc);
        fprintf(fps, "me_guc = %16.6f, std_guc = %16.6f\n", me_guc, std_guc);
}
```

ORIGINAL PAGE IS
OF POOR QUALITY

```c
/*      all the external variables employed in the head.h file  */

#include        "head.h"

/*      model and output optional parameters:
*/
int     te, vs, tg, ph, kf, lo, ot;

/*      model option definitions
*/

char    *cte[2] = {
                        "using Battan equations",
                        "using inputed tau",
                };

char    *can[1] = {
                        "water cloud model",
                };

char    *trk[2] = {
                        "exact solution of trunk forward scatter coef.",
                        "corner reflection model",
                };

char    *phs[2] = {
                        "random phase",
                        "phase due to range differences",
                };

char    *sur[4] = {
                        "seawater surface",
                        "slightly rough surface",
                        "Gaussian type distribution for very rough surface",
                        "Exponential type distribution for very rough surface",
                };

char    *low[4] = {
                        "100% Sundri as lower canopy layer",
                        "70% Sundri and 30% Gewa as lower canopy layer",
                        "30% Sundri and 70% Gewa as lower canopy layer",
                        "100% Gewa as lower canopy layer",
                };

char    *pol[4] = {
                        "HH polarization",
                        "HV polarization",
                        "VH polarization",
                        "VV polarization",
                };

/*      tree dbhs (>= 5 cm || <= 39.9 cm) of Sundri and Gewa in one pixel
*/
double  DBH_s[MAX1], DBH_g[MAX1];

/*      look-up tables of exact trunk solutions of forward scattering
        coefficients
*/
double  TAB_s[TAB], TAB_g[TAB];

/*      random phases of Sundri and Gewa in one stand
*/
double  PHS[MAX];

/*      x coordinates of Sundri and Gewa in one stand
*/
double  X[MAX];

/*      np:     # of pixels in a forest stand
        rg, rz: range and azimuth resolutions
                area: pixel size (rg * rz)
*/
int     np;
double  rg, rz, area;

/*      radar parameters
*/
int     pl;
double  k, wvl, Theta, theta;

/*      dielectric constants of trunks and leaves
*/
double  eptr_s, epti_s, eplr_s, epli_s;
double  eptr_g, epti_g, eplr_g, epli_g;

/*      dielectric constants of ground surface and seawater:
*/
double  epsr, epsi, epwr, epwi;

/*      regression coefficients
*/
double  A1, A2, A3, A4, B1, B2, B3, B4;

/*      tau_u: extinction coefficient of the upper canopy
        tau_l: extinction coefficient of the lower canopy
*/
double  tau_u, tau_l;

/*      f_u: ratios of eta_u to tau_u of the upper canopy
        f_l: ratios of eta_l to tau_l of the lower canopy
*/
double f_u, f_l;

/*      rat_l: ratios of the material to the air of the lower canopy
        rat_u: ratios of the material to the air of the upper canopy
*/
double  rat_l, rat_u;

/*      h: mean-square root roughness height of ground surface
        l: correlation length of ground surface
*/
double  h, l;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
/*      open files to read and write.  Totally, there are 12 files
**      opened, in which 5 files are input and 7 files are output.
*/

#include        "head.h"

openfl(argc, argv)

        int     argc;
        char    *argv[];

{

        FILE    *fopen();

/* check correct numbers of input and output files including SIGMA  */
        if (argc != 13) {
                printf("incorrect numbers of input and output files\n");
                exit(1);
        }

/* input look-up tables of trunk exact solutions         */
        if ((fptab = fopen(*++argv, "r")) == NULL)      {
                printf("can't open %s to read\n", *argv);
                exit(1);
        }

/* input data for modeling      */
        if ((fpin = fopen(*++argv, "r")) == NULL)       {
                printf("can't open %s to read\n", *argv);
                exit(1);
        }

/* input # of trees for each dbh segment                 */
        if ((fppxl = fopen(*++argv, "r")) == NULL)      {
                printf("can't open %s to read\n", *argv);
                exit(1);
        }

/* input dbhs of Sundri and Gewa        */
        if ((fpdbh = fopen(*++argv, "r")) == NULL)      {
                printf("can't open %s to read\n", *argv);
                exit(1);
        }

/* output statistical data in one forest stand          */
        if ((fps = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }

/* output the sum of six component sigmas for each pixel */
        if ((fpt = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }

/* output dbs, vsu, vsl, tg, glc, and guc for each pixel */
        if ((fp1 = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }
        if ((fp2 = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }
        if ((fp3 = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }
        if ((fp4 = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }
        if ((fp5 = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }
        if ((fp6 = fopen(*++argv, "w")) == NULL)        {
                printf("can't open %s to write\n", *argv);
                exit(1);
        }

}
```

ORIGINAL PAGE IS
OF POOR QUALITY

```c
/*      Tree biomass in one pixel (25 * 25 m), which is calculated based
**      on data from Dr. Imhoff.  The biomass data is employed to
**      compute attenuation coefficient of trees (Sundri and Gewa) for
**      each pixel.  Furthermore, the biomass data varies with pixels
**      as a result of tree number change in each pixel.
**
**      note: n1, ..., and n7, and m1, ..., and m7 are tree numbers
**            in a pixel corresponding to seven dbh segments for
**            Sundri and Gewa, respectively.
**
**      mass with unit: g / (m * m).
*/

#include        "head.h"

#define S1      13844.8         /* ratios of tree biomass (Sundri */
#define S2      52434.3         /* and Gewa) to tree numbers for  */
#define S3      146604.7        /* each dbh segment with unit:    */
#define S4      251068.2        /* g / (# of tree).               */
#define S5      293730.0        /* data from Dr. Imhoff           */
#define S6      302709.8
#define S7      311666.7

#define G1      7024.9
#define G2      20699.4
#define G3      56674.3
#define G4      122545.2
#define G5      225944.9
#define G6      396306.0
#define G7      528300.0

biomas(n1, n2, n3, n4, n5, n6, n7, m1, m2, m3, m4, m5, m6, m7, M)

        int     n1, n2, n3, n4, n5, n6, n7;
        int     m1, m2, m3, m4, m5, m6, m7;

        double  *M;

{

        double  M1, M2;

/* tree biomass of Sundri       */
        M1 = S1 * n1 + S2 * n2 + S3 * n3 + S4 * n4;
        M1 += S5 * n5 + S6 * n6 + S7 * n7;

/* biomass per square meters    */
        M1 /= area;

/* tree biomass of Gewa         */
        M2 = G1 * m1 + G2 * m2 + G3 * m3 + G4 * m4;
        M2 += G5 * m5 + G6 * m6 + G7 * m7;

/* biomass per square meters    */
        M2 /= area;

        if (ot == 1)    {
                printf("Sundri biomass in a pixel: M1 = %lf\n", M1);
                printf("Gewa biomass in a pixel: M2 = %lf\n\n", M2);
        }

/* total tree biomass   */
        *M = M1 + M2;
```

```c
/*      Fresnel reflection coefficients (magnitude and phase) of H or V
**      polarization incidence from eqns (9.1 - 49) and (9.1 -46) (Ruck,
**      et al., 1970).
**
**      note: relative complex dielectric constant, ep = ep' - jep".
**            relative permeability equals 1.
*/

#include        <math.h>

rfhh(thet, epr, epi, rfm, rfp)  /* HH polarization */

        double  thet, epr, epi, *rfm, *rfp;

{

        double  a, b, numm, nump, denm, denp;
        double  ci;

        ci = cos(thet);

a_b(thet, epr, epi, &a, &b);

/* magnitude    */
        numm = sqrt((ci - a) * (ci - a) + b * b);
        denm = sqrt((ci + a) * (ci + a) + b * b);

        *rfm = numm / denm;

/* phase        */
        nump = atan2(-b, (ci - a));
        denp = atan2(b, (ci + a));

        *rfp = nump - denp;

}

rfvv(thet, epr, epi, rfm, rfp)  /* VV polarization */

        double  thet, epr, epi, *rfm, *rfp;

{

        double  a, b, numm, nump, denm, denp;
        double  ci;

        ci = cos(thet);

a_b(thet, epr, epi, &a, &b);

/* magnitude    */
        numm = (epr * ci - a) * (epr * ci - a);
        numm += (epi * ci + b) * (epi * ci + b);

        denm = (epr * ci + a) * (epr * ci + a);
        denm += (epi * ci - b) * (epi * ci - b);

        *rfm = sqrt(numm) / sqrt(denm);

/* phase        */
        nump = atan2(-(epi * ci + b), (epr * ci - a));
        denp = atan2(-(epi * ci - b), (epr * ci + a));

        *rfp = nump - denp;

}

a_b(thet, epr, epi, a, b)       /* sqrt (a + jb)        */

        double thet, epr, epi, *a, *b;

{

        double  r, phss;
        double  si;

        si = sin(thet);

/* magnitude and phase  */
        r = sqrt((epr - si * si) * (epr - si * si) + epi * epi);
        r = sqrt(r);

        phss = atan2(-epi, (epr - si * si));
        phss /= 2.0;

        *a = r * cos(phss);
        *b = r * sin(phss);

}
```

```c
/*      Incoherent component of forward scattering coefficient from a
**      very rough surface using equations 9.7-118 - 132, in Ruck,
**      et al.,  vol. 2, pp. 720-722.
**
**      note: this program is not used in Bangladesh project, which
**            is written for the completion of the whole program.
*/


#include        "head.h"

gama(epr, epi, rmsv)

double  epr;    /* real part of dielectric constant of surface */
double  epi;    /* imagery part of dielectric constant of surface */
double  *rmsv;  /* incoherent forward scattering coefficient    */

{

        double  phi;    /* azimuth angle from inc. to sca. direction */
        double  thetas; /* scattering angle             */

        double  bthh(), bthv(), btvh(), btvv();
        double  kx, ky, kz;
        double  ji, btph2, si, ss, ci, cs, sp, cp, s2;

/* for forward scattering       */
        thetas = -theta;
        phi = 0.0;

        si = sin(theta);
        ss = sin(thetas);
        ci = cos(theta);
        cs = cos(thetas);
        sp = sin(phi);
        cp = cos(phi);

        kx = si - ss * cp;      /* 9.1 - 53 (Ruck et al., 1970) */
        ky = ss * sp;           /* 9.1 - 54     */
        kz = - ci - cs;         /* 9.1 - 55     */
        s2 = 4.0 * h * h / (l * l);

        if (kf == 3)
                ji = 4.0 * exp(-(kx*kx + ky*ky)/(s2*kz*kz))/(s2*kz*kz);
        else    {
                ji = exp(-sqrt(6.0) * sqrt((kx*kx + ky*ky) / (s2*kz*kz)));
                ji *= (12.0 / (s2 * kz * kz));
        }

        if (pl == 1 )
                btph2 = bthh(theta, thetas, phi, epr, epi);
        else if (pl == 2 )
                btph2 = bthv(theta, thetas, phi, epr, epi);
        else if (pl == 3)
                btph2 = btvh(theta, thetas, phi, epr, epi);
        else
                btph2 = btvv(theta, thetas, phi, epr, epi);

        *rmsv = btph2 * ji;

}
```

```
/*      beta parameters for a very rough surface scattering, which is
**      defined in equations 9.1 - 124 to 9.1 - 132 (p. 722, Ruck, et
**      al., 1970)
*/

#include        <math.h>

double  si, ss, ci, cs, sp, cp;
double  a1, a2, a3, a4, in;

double  btvv(thei, thes, phi, epr, epi)

        double  thei, thes, phi, epr, epi;

{

        double  mh, hh, mv, vv, m0, r1, i1;

inita(thei, thes, phi);

rfhh(in, epr, epi, &mh, &hh);
rfvv(in, epr, epi, &mv, &vv);

        r1 = a2 * a3 * mv * cos(vv) + si * ss * sp * sp * mh * cos(hh);
        i1 = a2 * a3 * mv * sin(vv) + si * ss * sp * sp * mh * sin(hh);
        m0 = sqrt(r1 * r1 + i1 * i1);
        m0 /= a1 * a4;
        m0 *= m0;
        return(m0);
}

double  btvh(thei, thes, phi, epr, epi)

        double  thei, thes, phi, epr, epi;

{

        double  mh, hh, mv, vv, m0, r1, i1;

inita(thei, thes, phi);

rfhh(in, epr, epi, &mh, &hh);
rfvv(in, epr, epi, &mv, &vv);

        r1 = ss * a2 * mv * cos(vv) - si * a3 * mh * cos(hh);
        i1 = ss * a2 * mv * sin(vv) - si * a3 * mh * sin(hh);
        m0 = sp * sqrt(r1 * r1 + i1 * i1) / (a1 * a4);
        m0 *= m0;
        return(m0);

}

double  bthv(thei, thes, phi, epr, epi)

        double  thei, thes, phi, epr, epi;

{

        double  mh, hh, mv, vv, m0, r1, i1;

inita(thei, thes, phi);

rfhh(in, epr, epi, &mh, &hh);
rfvv(in, epr, epi, &mv, &vv);

        r1 = ss * a2 * mh * cos(hh) - si * a3 * mv * cos(vv);
        i1 = ss * a2 * mh * sin(hh) - si * a3 * mv * sin(vv);
        m0 = sp * sqrt(r1 * r1 + i1 * i1) / (a1 * a4);
        m0 *= m0;
        return(m0);

}

double  bthh(thei, thes, phi, epr, epi)

        double  thei, thes, phi, epr, epi;

{

        double  mh, hh, mv, vv, m0, r1, i1;

inita(thei, thes, phi);

rfhh(in, epr, epi, &mh, &hh);
rfvv(in, epr, epi, &mv, &vv);

        r1 = -si * ss * sp * sp * mv * cos(vv) - a2 * a3 * mh * cos(hh);
        i1 = -si * ss * sp * sp * mv * sin(vv) - a2 * a3 * mh * sin(hh);
        m0 = sqrt(r1 * r1 + i1 * i1) / (a1 * a4);
        m0 *= m0;
        return(m0);
}

inita(thei, thes, phi)

        double  thei, thes, phi;

{

        si = sin(thei);
        ss = sin(thes);
        ci = cos(thei);
        cs = cos(thes);
        sp = sin(phi);
        cp = cos(phi);
        a1 = 1.0 + si * ss * cp - ci * cs;
        a2 = ci * ss + si * cs * cp;
        a3 = si * cs + ci * ss * cp;
        a4 = ci + cs;
        in = 0.7071068 * sqrt(1.0 - si * ss * cp + ci * cs);
        in = acos(in);

}
```

```
/*      calculate dielectric constants (epsilon' and epsilon")
**      (note: epsilon = epsilon' - j epsilon") of mixtures layers
**      (the air as a host material, and materials, such as leaves,
**      branches, trunks, etc. as inclusions) based on the formula in
**      Ulaby, et al., (1986, E.62).  The reflective model is chosen.
*/

#include        "head.h"

#define ep11    1.0     /* dielectric constant of air   */
#define ep12    0.0

ep(v, ep21, ep22, epr, epi)

        double  v, ep21, ep22, *epr, *epi;

{

        double  tmp1, tmp2, mag, phs;

        tmp1 = ep11 * ep21 - ep12 * ep22;
        tmp2 = -ep22 * ep11 - ep12 * ep21;

        mag = sqrt(tmp1 * tmp1 + tmp2 * tmp2);
        mag = sqrt(mag);

        phs = atan2 (tmp2, tmp1);
        phs /= 2.0;

        tmp1 = mag * cos(phs);  /* real part            */
        tmp2 = mag * sin(phs);  /* imaginary part       */

/* equivalent dielectric constant       */
        *epr = (1.0 - v) * (1.0 - v) * ep11 + v * v * ep21;
        *epr += (2.0 * v * (1.0 - v) * tmp1);

        *epi = (1.0 - v) * (1.0 - v) * ep12 + v * v * ep22;
        *epi -= (2.0 * v * (1.0 - v) * tmp2);

        if (ot == 1)    {
                printf("dielectric constant:\n");
                printf("epr = %lf, epi = -%lf\n\n", *epr, *epi);
        }

}
```

```c
/*      backscattering coefficient per unit area from a slightly rough
**      surface using a small perturbation model (Dobson and Ulaby,
**      1986), when the ground surface is in the water-off condition.
**
**      note: without the canopy attenuation, which is calculated in
**            main.c.
*/

#include        "head.h"

double surI(refs)

        double  refs;

{

        double  sigma, ci2, si;

        ci2 = cos(theta) * cos(theta);
        si = sin(theta);

        sigma = 4.0 * k * k * h * h * k * k * l * l;
        sigma *= (ci2 * ci2 * exp(-(k * k * l * l * si * si)));
        sigma *= (refs * refs);

        return(sigma);
}
```

```c
/*	backscattering coefficients per unit area from a very rough
**	surface using eqs. 9.1-139 and 141, in Ruck et al., vol. 2.,
**	p. 721 - 725.
**
**	note: backscattering coefficients of HH and VV are identical.
**	      backscattering coefficients of VH and HV are zero.
*/

#include	"head.h"

double	surII(epr, epi)

	double	epr, epi;

{

	double	co4, s2, mgama;
	double	bm, bp;

	s2 = 4.0 * h * h / (l * l);	/*  s * s, p. 721	*/

	co4 = cos(theta) * cos(theta) * cos(theta) * cos(theta);

	if (pl == 1 || pl == 4)
		rfvv(0.0, epr, epi, &bm, &bp);
	else	{
		bm = 0.0;
		bp = 0.0;
	}

	if (kf == 3) {
		mgama = exp(-tan(theta) * tan(theta) / s2);
		mgama *= bm * bm / (s2 * co4);
	}
	else	{
		mgama = exp(-sqrt(6.0 / s2) * tan(theta));
		mgama *= 3.0 * bm * bm / (s2 * co4);
	}

	return(mgama);
}
```

```
/*      canopy volume scattering (per unit area) (Attema and Ulaby,
**      1978).
*/

#include        "head.h"

double  vsw(tau, eta, TAU)

        double  tau, eta, TAU;

{

        double  sigma;

        sigma = eta * cos(theta) * (1.0 - TAU) / 2.0 / tau;

        return(sigma);

}
```

```c
/*      radar backscattering coefficient per unit area is derived from
**      exact solutions of trunks (Sundri and Gewa), which are treated
**      as dielectric cyliners with finite lengths and with smooth
**      trunk surface.
**
**      note: the effective trunk length is equal to the tree height
**      (derived from regression models based on data provided from Dr.
**      Imhoff) minus the one third of the mean canopy depth in a pixel.
*/

#include "head.h"

double tge(N_SUN, N_GEWA, refs)

        int     N_SUN, N_GEWA;

        double  refs;
{

        static  int     po = 0;

        int     i, tmp;

        double  sum11 = 0.0, sum12 = 0.0, sigma1, sigmap1;
        double  sum21 = 0.0, sum22 = 0.0, sigma2, sigmap2;
        double  alph_s, alph_g;
        double  sigma;

        for(i = 0; i < N_SUN; ++po, i++)        {
                tmp = (int) (10.0 * (DBH_s[i] - 5.0) + 0.1);
                sigmap1 = 2.0 * TAB_s[tmp];     /* double path */

                if (ph == 1)
                        alph_s = PHS[po];
                else if (ph == 2)
                        alph_s = 2.0 * k * X[po] / sin(theta);
                else    {
                        printf("ph can't be %d\n", ph);
                        exit(1);
                }

                sigmap1 = sqrt(sigmap1);
                sum11 += sigmap1 * cos(alph_s);
                sum12 += sigmap1 * sin(alph_s);
        }

        for(i = 0; i < N_GEWA; ++po, i++)       {
                tmp = (int) (10.0 * (DBH_g[i] - 5.0) + 0.1);
                sigmap2 = 2.0 * TAB_g[tmp];     /* double path */

                if (ph == 1)
                        alph_g = PHS[po];
                else
                        alph_g = 2.0 * k * X[po] / sin(theta);

                sigmap2 = sqrt(sigmap2);
                sum21 += sigmap2 * cos(alph_g);
                sum22 += sigmap2 * sin(alph_g);
        }

        sigma1 = sum11 * sum11 + sum12 * sum12;
        sigma2 = sum21 * sum21 + sum22 * sum22;

        sigma = (sigma1 + sigma2) * refs *refs;

        sigma /= area;          /* per unit area        */

        return(sigma);

}
```

ORIGINAL PAGE IS
OF POOR QUALITY

```c
/*      corner reflector model for tree trunks of Sundri and Gewa,
**      which is the same as that used by Richards, et al., (1987).
**
**      note: the effective trunk length is equal to tree height
**            (derived from regression models based on data provided
**            by Dr. Imhoff) minus one third the mean canopy depth in
**            a pixel.
*/

#include "head.h"

double  tgc(N_SUN, N_GEWA, me_can, ref_s, ref_g, refs)

        int     N_SUN, N_GEWA;

        double  me_can, ref_s, ref_g, refs;

{

        static  int     PO = 0;

        int     i;

        double  tmp, ht;
        double  sum11 = 0.0, sum12 = 0.0, sigma1, sigmap1;
        double  sum21 = 0.0, sum22 = 0.0, sigma2, sigmap2;
        double  alph_s, alph_g;
        double  sigma;

        for(i = 0; i < N_SUN; ++PO, i++)        {

        /* effective height of Sundri trunk */
                ht = A1 + B1 * DBH_s[i];
                ht -= me_can / 3.0;

        /* effective planar width of Sundri trunk */
                tmp = sqrt (0.25 * wvl * (DBH_s[i] / M_CM));

        /* effective area of Sundri trunk       */
                tmp *= (2.0 * ht * sin(theta));

        /* radar cross section of Sundri trunk  */
                sigmap1 = 4.0 * PI * tmp * tmp / wvl / wvl;

                if (ph == 1)
                        alph_s = PHS[PO];
                else if (ph == 2)
                        alph_s = 2.0 * k * X[PO] / sin(theta);
                else    {
                        printf("ph can't be %d\n", ph);
                        exit(1);
                }

                sigmap1 = sqrt(sigmap1);
                sum11 += sigmap1 * cos(alph_s);
                sum12 += sigmap1 * sin(alph_s);

        }

        for(i = 0; i < N_GEWA; ++PO, i++)       {

        /* effective height of Gewa trunk */
                ht = A3 + B3 * DBH_g[i];
                ht -= me_can / 3.0;

        /* effective planar width of Gewa trunk */
                tmp = sqrt (0.25 * wvl * (DBH_g[i] / M_CM));

        /* effective area of Gewa trunk */
                tmp *= 2.0 * ht * sin(theta);

        /* radar cross section of Gewa trunk    */
                sigmap2 = 4.0 * PI * tmp * tmp / wvl / wvl;

                if (ph == 1)
                        alph_g = PHS[PO];
                else
                        alph_g = 2.0 * k * X[PO] / sin(theta);

                sigmap2 = sqrt(sigmap2);
                sum21 += sigmap2 * cos(alph_g);
                sum22 += sigmap2 * sin(alph_g);

        }

        sigma1 = (sum11 * sum11 + sum12 * sum12) * ref_s * ref_s;
        sigma2 = (sum21 * sum21 + sum22 * sum22) * ref_g * ref_g;
        sigma = (sigma1 + sigma2) * refs * refs;

        sigma /= area;  /* per unit area        */

        return(sigma);

}
```

ORIGINAL PAGE IS OF POOR QUALITY

```
/*      interaction of canopy - surface scattering per unit area
**      using Engheta and Elachi's model (1982) by means of finding
**      sigma0 in terms of water cloud model (Attema and Ulaby, 1978).
**      This formula is the same as that in Sun and Simonett (1988).
**
**      note: sigma0 = eta * me_can
*/

#include        "head.h"

double  ee(tau, me_can, eta, refs)

        double  tau, me_can, eta, refs;
{

        double  sigma, alph, sigma0;
        double  tmp;

        sigma0 = eta * me_can;

        alph = tau * me_can / cos(theta);
        tmp = exp(-2.0 * k * k * h * h * cos(theta) * cos(theta));

        sigma = exp(-alph) * 0.5 * (exp(alph) - exp(-alph) / alph);
        sigma *= refs * refs * tmp;
        sigma += 2.0;
        sigma *= (refs * refs * tmp * exp(-2.0 * alph) * sigma0);

        return(sigma);
}
```

```
/*      produces all the DBHs of total tree numbers (Sundri and Gewa)
**      in a stand area, which are generated by the random() function
**      in the UNIX system with random uniform distribution.  The input
**      number is the total number of the dbhs*.
**
**      note:   all the dbhs range from 0.0 to 4.9 cm.  Constants 5.0,
**              10.0, 15.0, 20.0, 25.0, 30.0, and 35.0 are added to the
**              dbh data set to produce dbhs in the following ranges:
**              5.0 -- 9.9, 10.0 -- 14.9, 15.0 -- 19.9, 20.0 -- 24.9,
**              25.0 -- 29.9, 30.0 -- 34.9, and 35.0 -- 39.9 cm in the
**              main.c program,  respectively.
*/

#define SIZE    50      /* dbh (mm)     */

main ()

{

        int i, N;
        long    random();

/* input total dbh numbers in one stand         */
        scanf("%d\n", &N);

/* dbh in cm with 0.1 cm as its accuracy        */
        for (i = 0; i < N; i++)
                printf("%f\n", ((double)(random()%SIZE)) / 10.0);

}
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
/*      random (x, y) positions of total Sundri and Gewa trees in a
**      stand are generated by cutting the whole output data of the
**      random() function in the UNIX system into two parts.  The first
**      half of the data set is used to yield x coordinates, and the
**      second half y coordinates.  Twice the number of the positions
**      are input.  The (x, y) positions are designated in centimeter
**      units.  The stand has an area of N * hectares.
*/

#define SIZE    10000   /* centimeters of 100 meters    */

main ()

{
        int i, N;
        long    random();

/* input the twice number of the positions      */
        scanf("%d", &N);

/* (x, y) position in cm        */
        for (i = 0; i < N; i++)
                printf("%d\n", random()%SIZE);

}
```

```
/*      uses the output of above (x, y) coordinate data as its input to
**      produce the tree numbers of each dbh segment respectively for
**      each simulated pixel in the whole stand.  To run the program,
**      tree number (TOTAL) for each dbh segment in a hectare and the
**      position data should be in one input data file.  The first line
**      of input data is TOTAL and the rest of the data lines
**      (N x TOTAL) are positions or (x, y) data, where N is a constant
**      and can be changed for different stand sizes.  In our case, the
**      N is 10.
*/

#define         N       10      /* 10 hectares in a stand       */

main()

{

        int i, w, cnt[16], x[3000], y[3000], temp_x, temp_y, TOTAL;

/* tree number for one dbh segment in a hectare area */
        scanf("%d", &TOTAL);

        for (w = 0; w < N; w++ ) {

        for (i = 0; i < 16; i++)            /* initialization for each ha  */
                cnt[i] = 0;

        /* input positions one ha by one */

        for ( i = 0; i < TOTAL; i++ ) {
                scanf("%d %d", &temp_x, &temp_y);
                x[i] = temp_x;
                y[i] = temp_y;
                if ( (i + 1) % TOTAL == 0)      /* after one ha input data */
                break;
                }

        /* locate each tree in 16 pixels */
        for (i = 0; i < TOTAL; i++) {

           if (0 <= x[i] && x[i] <= 2500 && 0 <= y[i] && y[i] <= 2500)
                        cnt[0]++;
           else if (0 <= x[i] && x[i] <= 2500 && 2500 < y[i] && y[i] <= 5000)
                        cnt[1]++;
           else if (0 <= x[i] && x[i] <= 2500 && 5000 < y[i] && y[i] <= 7500)
                        cnt[2]++;
           else if (0 <= x[i] && x[i] <= 2500 && 7500 < y[i] && y[i] <= 10000)
                        cnt[3]++;
           else if (2500 < x[i] && x[i] <= 5000 && 0 <= y[i] && y[i] <= 2500)
                        cnt[4]++;
           else if (2500 < x[i] && x[i] <= 5000 && 2500 < y[i] && y[i] <= 5000)
                        cnt[5]++;
           else if (2500 < x[i] && x[i] <= 5000 && 5000 < y[i] && y[i] <= 7500)
                        cnt[6]++;
           else if (2500 < x[i] && x[i] <= 5000 && 7500 < y[i] && y[i] <= 10000)
                        cnt[7]++;
           else if (5000 < x[i] && x[i] <= 7500 && 0 <= y[i] && y[i] <= 2500)
                        cnt[8]++;
           else if (5000 < x[i] && x[i] <= 7500 && 2500 < y[i] && y[i] <= 5000)
                        cnt[9]++;
           else if (5000 < x[i] && x[i] <= 7500 && 5000 < y[i] && y[i] <= 7500)
                        cnt[10]++;
           else if (5000 < x[i] && x[i] <= 7500 && 7500 < y[i] && y[i] <= 10000)
                        cnt[11]++;
           else if (7500 < x[i] && x[i] <= 10000 && 0 <= y[i] && y[i] <= 2500)
                        cnt[12]++;
           else if (7500 < x[i] && x[i] <= 10000 && 2500 < y[i] && y[i] <= 5000)
                        cnt[13]++;
           else if (7500 < x[i] && x[i] <= 10000 && 5000 < y[i] && y[i] <= 7500)
                        cnt[14]++;
           else
                        cnt[15]++;

        }

                /* tree number in each pixel for one ha*/

                for(i = 0; i < 16; i++)
                        printf("%4d\n", cnt[i]);

        }

}
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
PROGRAM = SIGMA

RCSMC = main.c

RCSMO = main.o

RCSSUBC = head.c \
          openfl.c \
          indata.c \
          phs_x.c \
          tau.c \
          equi.c \
          biomas.c \
          ref.c \
          gama.c \
          beta.c \
          ep.c \
          surI.c \
          surII.c \
          vsw.c \
          tge.c \
          tgc.c \
          ee.c

RCSSUBO = head.o \
          openfl.o \
          indata.o \
          phs_x.o \
          tau.o \
          equi.o \
          biomas.o \
          ref.o \
          gama.o \
          beta.o \
          ep.o \
          surI.o \
          surII.o \
          vsw.o \
          tge.o \
          tgc.o \
          ee.o

plint:
	lint $(RCSMC) $(RCSSUBC) > /tmp/er.lint &

pcmain:
	cc -O -c $(RCSMC) -lm

pcc:
	cc -O -o $(PROGRAM) $(RCSMC) $(RCSSUBC) -lm

pco:
	cc -O -o $(PROGRAM) $(RCSMO) $(RCSSUBO) -lm
```

RUN Sat Sep 24 11:36:09 1988 1

```
SIGMA intab indata inpxl indbh sts_o t_o dbs_o vsu_o vsl_o tg_o glc_o guc_o > check_out
```