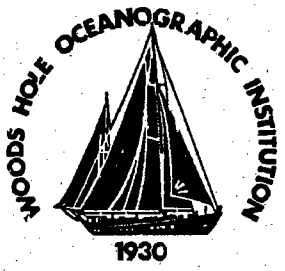
WHOI-93-10

Woods Hole Oceanographic Institution



Ocean Acoustical Ray-Tracing Software RAY

by

James B. Bowlin

John L. Spiesberger

Timothy F. Duda

Lee F. Freitag

October 1992

Technical Report

Funding was provided by the Office of Naval Research under contract N00014-86-C-0358 and the Office of Naval Technology under contract N00014-90-C-0096.

WHOI-93-10

Ocean Acoustical Ray-Tracing Software RAY

by

James B. Bowlin
John L. Spiesberger
Timothy F. Duda
Lee F. Freitag

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

October 1992

Technical Report

Funding was provided by the Office of Naval Research under contract N00014-86-C-0358 and the Office of Naval Technology under contract N00014-90-C-0098.

Approved for Distribution:

George V. Frisk

George V. Frisk, Chair
Department of Applied Ocean Physics & Engineering

Abstract:

A new computer program for accurate calculation of acoustic ray paths through a range-varying ocean sound channel has been written. It is based on creating a model of the speed of sound in the ocean, consistent with input data, that produces the smoothest possible wavefronts. This scheme eliminates "false caustics" from the wavefront. It may be useful in calculating an approximate solution to the full wave equation at megameter ranges.

1. Introduction

The Ray program is a part of an ongoing effort by John Spiesberger's research group to create a fully automated system for performing basin scale ocean acoustic tomography.

One key part of such a system is forward modeling of multipath, which, given a model ocean sound–speed field and bathymetry, predicts when sound emitted from a source will arrive at a receiver and where the sound will travel. There are currently several ways to solve the forward problem including normal modes, the parabolic equation and geometric ray tracing. Some theoretical investigations into forward modeling have led us to the concept of eigentubes which are the full wave generalization of eigenrays [*Bowlin*, 1991]. These considerations lead to the conclusion that the wavefront generated by a geometrical ray trace is useful in solving the forward problem at frequencies much lower than would be expected by a simple analysis of the high frequency assumptions implicit in the geometrical acoustics approximation.

The idea we use is that for a given physical situation (environment and source placement) the geometrical wavefront will be independent of the frequency and bandwidth of the source. Consider a collection of geometric rays all starting from a single source location with different starting angles and all ending at the receiver range. The depths, arrival times, and angles of these rays (at the receiver range) as a function of launch angle are what we call a wavefront. The actual acoustic field for a given source frequency and bandwidth can then be calculated from the geometrical wavefront [*Buchal and Keller*, 1960]. We envision the geometrical rays as a (frequency independent) skeleton to which a flesh is added with (frequency dependent) diffraction effects.

In addition to offering a framework for calculating the complete acoustic field, the geometrical optics or ray model offers accurate and numerically efficient determination of distinctly separated multipath signals. The multipath calculation is equivalent to finding the spatial structure of a wavefront formed from an impulse source, a structure considered since the early stages of acoustic tomography [*Brown et al.*, 1980]. This structure was recently analyzed with an experiment [*Duda et al.*, 1992].

None of the existing ray trace codes of which we are aware are suitable for the full-wave numerical extension at the megameter ranges of basin scale tomography. Our primary considerations for evaluating ray trace codes have been accuracy and speed. To add a diffracted flesh to the geometric bones, we must also include "smoothness of the wavefront" as a primary consideration.

Most fast ray tracing codes for ocean acoustics approximate the sound speed of the ocean as piecewise linear. The discontinuities in the first derivative of these piecewise linear approximations cause "false caustics" or more precisely "non-turning point caustics" (NTP caustics). One way to see this is to look at the group velocity as a function of launch angle in a range independent

environment. Denote

$$S \equiv \frac{dv_g}{d\psi}$$

where ψ is the launch angle and v_g is the "one loop group velocity" defined as the range of a single loop of a ray divided by the time it takes the ray to traverse the loop. A little bit of calculus shows that S depends upon integrals along the ray of the form

$$\int \left(\alpha_i + \beta_i \frac{c''c}{(c')^2}\right) \frac{dz}{\sin\theta}$$

where α_i and β_i are bounded, smoothly varying functions and θ is the angle of the ray with respect to the horizontal. For a piecewise linear sound speed, $c''c/(c')^2$ becomes a delta function while $1/\sin\theta$ has an integrable infinity (with respect to dz) at the turning points of the rays. When the turning point of a ray approaches a knot in a piecewise linear ocean then S can become arbitrarily large. This causes discontinuous jumps in the wavefront which would not occur if the second derivative of the sound speed were bounded. Caustics appear in a wavefront at extrema of the function of depth at the receiver range vs. launch angle, where

$$\frac{dz_{\text{rec}}}{d\psi} = 0.$$

This condition is satisfied whenever the wavefront contains a ray at a turning point. These we call turning point caustics. Wherever there is a jump in the wavefront due to a very large value of S then a caustic can also arise. These caustics are not restricted to lie on a turning point of a ray so we call them NTP caustics.

Measurements of acoustic pulses at a few hundred Hz and at distances of one to three thousand kilometers show that the wavefront is simpler than predicted by ray traces which model the sound speed as piecewise linear [*Duda et al.*, 1992; *Sparrock*, 1990; *Spiesberger and Metzger*, 1991]. A reasonable requirement of a ray calculation (or ray-tracing) code is the reproduction of a stable averaged wavefront structure when a smooth averaged ocean is considered. The program Ray is designed to produce continuous wavefronts at megameter ranges. The overall philosophy has been to find a simple model of the environment, consistent with the input environmental data (sound speeds and depths), that produces the smoothest wavefronts. This strategy eliminates many but not all of the NTP caustics. The NTP caustics that remain are due the the sound–speed structure of the environment. These irreducible NTP caustics extend smoothly over a finite fraction of the wavefront. This means that the acoustic field in the shadow zones associated with these caustics can be found analytically from the geometrical wavefront.

Although the idea of smoothing sound–speed profiles in order to produce smooth wavefronts has been around for a long time [*Pederson*, 1961], we have implemented an automated version of

the smoothing that is integrated with efficient numerical techniques that enable us to trace rays quickly through a realistic model of the ocean.

Mathematical algorithms of Ray are described in section 2. Sections 3, 4 and 5, respectively, describe the input, the operation, and the output of the program. Section 6 briefly describes the performance of Ray. Section 7 is a summary.

2. Computational Algorithms

Equations of Motion and the Spherical Earth Correction

The equations of motion for a ray travelling through the ocean can be cast in Cartesian coordinates as follows,

$$\frac{d\theta}{dr} = \frac{\partial_r c}{c}\tan\theta - \frac{\partial_z c}{c}$$
$$\frac{dz}{dr} = \tan\theta$$
$$\frac{dt}{dr} = \frac{\sec\theta}{c}$$

where θ is the angle of the ray with respect to the horizontal r axis, and z is the vertical coordinate. These equations are derived from Fermat's principle of least time in appendix A.

For long range ocean acoustics, the curvature of the Earth's surface makes non-Cartesian coordinates more suitable for ray tracing. Let new $\tilde{z}$ axes lie along radii passing through the center of the Earth with $\tilde{z} = 0$ at sea level and $\tilde{z} = R_e$ at the earth's center, where R_e is the radius of the earth. and let the new $\tilde{r}$ be the range measured along a circular arc at sea level. Three things happen when the equations of motion are translated into this new coordinate system. The first is a trivial change in the sign of z and θ. The second effect is from the conversion from dr to $d\tilde{r}$. Imagine a fish that stays at a depth $\tilde{z}$, directly below a moving boat. The fish will travel a shorter distance than the boat by a factor of $f_e = dr/d\tilde{r} = (R_e - \tilde{z})/R_e$. The third effect is a rotation of the coordinate system by $d\tilde{r}/R_e$ radians as we take a step of size $d\tilde{r}$.

The new equations of motion which include geometrical effects due to a spherical earth are

$$\frac{d\theta}{d\tilde{r}} = f_e\frac{\partial_{\tilde{z}} c}{c} - \frac{\partial_{\tilde{r}} c}{c}\tan\theta - \frac{1}{R_e}$$
$$\frac{d\tilde{z}}{d\tilde{r}} = f_e\tan\theta$$
$$\frac{dt}{d\tilde{r}} = \frac{f_e\sec\theta}{c}.$$

These are the equations that Ray integrates subject to the modifications and approximations discussed below.

Smoothing the sound speed

The interpolation of sound speed as a function of depth is the most important algorithm in Ray. For accurate calculation of ray paths and ray travel-times, sound speed is required at arbitrary locations. A procedure is required which takes as input a set of sound speeds defined on a discrete grid of depths, the most prevalent form of input, and produces a function describing the sound

speed at any depth. The sound-speed field should be continuous with a bounded second derivative, and should be a realistic approximation of the ocean.

Begin with a range independent sound–speed profile. Let the input depth grid and sound speeds be labeled z_i and c_i for $1 \leq i \leq N$ such that

$$c(z_i) = c_i.$$

Each depth z_i is called a knot for reasons that become apparent below. The continuous piecewise linear approximation to these points is denoted $c^{(1)}(z)$. It will be

$$c^{(1)}(z) = c_i + \beta_i(z - z_i), \qquad z_i \leq z \leq z_{i+1}$$

with $\beta_i \equiv (c_{i+1} - c_i)/(z_{i+1} - z_i)$. This is a continuous function. The first derivative is piecewise constant of the form

$$\frac{dc^{(1)}}{dz} = \beta_i, \qquad z_i < z < z_{i+1}.$$

The second derivative consists of a sum of delta functions,

$$\frac{d^2c^{(1)}}{dz^2} = \sum_{i=2}^{N-1} \alpha_i \delta(z - z_i)$$

where $\alpha_i \equiv \beta_i - \beta_{i-1}$. To smooth out the the sharp corners at the knots, and to reduce the magnitude of the second derivative, one can convolve this continuous piecewise linear model with a normalized symmetric tophat function defined by

$$g(w; z) = \begin{cases} (2w)^{-1}: & |z| \leq w \\ 0: & |z| > w \end{cases}.$$

The result of this operation,

$$\begin{aligned} c^{(2)}(w; z) &\equiv \int c^{(1)}(z - z')g(w; z')dz' \\ &= \frac{1}{2w}\int_{-w}^{w} c^{(1)}(z - z')dz' \end{aligned}$$

is a continuous piecewise parabolic function with a continuous piecewise first derivative and a finite piecewise constant (but discontinuous) second derivative. If the width, w, of $g()$ is less than the spacing between knots, then $c^{(2)}$ will be parabolic for z within w of a knot and it will be linear and equal to $c^{(1)}$ for all z that are not within w of a knot. In the parabolic regions the second derivative of $c^{(2)}$ is constant, and in the linear regions it is zero.

There is a tradeoff in determining w, the width of $g(w; z)$. We are forced to use an interpolation scheme because of our ignorance of the actual $c(z)$ away from the points z_i where c is measured.

A w is sought which gives the smoothest wavefront possible, and still remains consistent with the original set of measured values, $\{c_i\}$. The wider the w, then the smaller the second derivative of $c^{(2)}$ and the smoother the wavefront. For equally spaced z_i one choice is to set $w = (z_{i+1} - z_i)/2$, which will eliminate all of the linear sections of $c^{(2)}$ where the second derivative is zero and tend to minimize the second derivative in the parabolic sections.

A complete set of data that is available for determining global ocean sound sound is the Levitus data base of temperature and salinity [*Levitus*, 1982], which does not have equally spaced z_i. The above method of smoothing can be generalized to let the parameter w vary with depth. Define a set of widths w_i, such that the width of $g(z)$ is w_i when $z = z_i$. If the constraints

$$z_i - w_i \geq z_{i-1}$$

$$z_i + w_i \leq z_{i+1}$$

are imposed on the set of widths, then it is always possible to use these widths to construct a continuous piecewise parabolic function (with continuous first derivative) analogous to $c^{(2)}$, which will be equal to $c^{(2)}(w_i; z)$ for z such that $\max(z_{i-1} + w_{i-1}, z_i - w_i) < z < \min(z_{i+1} - w_{i+1}, z_i + w_i)$. This is not proven here, instead expressions are given for the resulting smoothed function, denoted by $c^{(3)}$.

The functional form of $c^{(3)}$ will depend on the values z_i, and w_i. There are three cases. Case 1 reproduces the linear segments of $c^{(1)}$ far away from knots. It occurs for all z such that

$$z_{i-1} + w_{i-1} < z < z_i - w_i$$

then

$$c^{(3)}(z) = c_i + \beta_i(z - z_i).$$

Case 2 reproduces the parabolic sections of $c^{(2)}(w_i; z)$ near a knot. It occurs for all z such that

$$\max(z_{i-1} + w_{i-1}, z_i - w_i) < z < \min(z_{i+1} - w_{i+1}, z_i + w_i)$$

then

$$c^{(3)}(z) = c_i + \frac{w_i\alpha_i}{4} + \frac{(\beta_{i+1} + \beta_i)}{2}(z - z_i) + \frac{\alpha_i}{4w_i}(z - z_i)^2.$$

Case 3 is the smooth interpolation of two "overlapping" sections of case 2. It occurs for all z such that

$$z_{i+1} - w_{i+1} < z < z_i + w_i.$$

Note that every section of case 3 is sandwiched between two case 2 sections. Let $c^{(3)}(z)$ defined on each of these section be $c_l(z)$ and $c_r(z)$. Then

$$c^{(3)}(z) = c_l(z_l) + \gamma_l(z - z_l) + \frac{\gamma_r - \gamma_l}{2(z_r - z_l)}(z - z_l)^2$$

where

$$z_l = z_{i+1} - w_{i+1}$$
$$z_r = z_i - w_i$$
$$\gamma_l = \left.\frac{dc_l}{dz}\right|_{z_l}$$
$$\gamma_r = \left.\frac{dc_r}{dz}\right|_{z_r}$$

An algorithm to automatically generate the optimized widths for any depth grid has not been developed. Ray's default set of $\{w_i\}$ has been designed to work with sound–speed profiles on the depth grid of the Levitus data base.

Removing the bias created by smoothing

There is a serious side effect of the smoothing process described above. The resulting smooth function does not pass through the original data set,

$$c^{(3)}(z_i) = c_i + \frac{w_i \alpha_i}{4}$$
$$\neq c_i$$

If the widths of the smoothing regions w_i were much smaller than the spacing of the z_i, then this would be a minor problem since it would be a small correction over a small region of z. We want to make the widths as large as possible, so it is necessary to fix the discrepancy between the input data set and the smoothed profile in order to minimize the bias in the travel times of calculated rays.

It is always possible to find a new set of sound speeds $\{\tilde{c}_i\}$ which, when smoothed, provide a function that goes through the original data points. The new set of sound speeds can be determined by solving the following set of linear equations for $\{\tilde{c}_i\}$

$$c_i = \tilde{c}_i + \frac{w_i}{4}\left(\frac{\tilde{c}_{i+1} - \tilde{c}_i}{\Delta_i} - \frac{\tilde{c}_i - \tilde{c}_{i-1}}{\Delta_{i-1}}\right)$$

with $\Delta_i = z_{i+1} - z_i$. Ray does not solve this set of equations directly. Instead, it provides a method for obtaining an iterated solution of the form

$$\tilde{c}_i^{(n)} = \frac{4\Delta_i\Delta_{i-1}c_i + \epsilon w_i\left(\tilde{c}_{i+1}^{(n-1)}\Delta_{i-1} + \tilde{c}_{i-1}^{(n-1)}\Delta_i\right)}{4\Delta_i\Delta_{i-1} - \epsilon w_i(\Delta_i + \Delta_{i-1})}$$

with $\tilde{c}_i^{(0)} = c_i$. The user can input the number of iterations and the convergence factor ϵ. These two parameters allow some flexibility in how much debiasing Ray applies.

On each iteration the sound speed at any depth is only affected by its two nearest neighbors. Thus the number of iterations controls how local or global the debiasing will be. The convergence

factor controls how close Ray tries to get to the original sound speeds. If it is set to 1.0, then Ray will attempt to compensate for all of the smoothing. If it is set to 0.5 then Ray will only try to get rid of one half of the offset caused by the smoothing. For a few test profiles, ten iterations with ϵ set to 1.0 converges to the original sound speeds.

An example of a smoothed sound–speed profile with the bias removed is displayed in Figure 1.

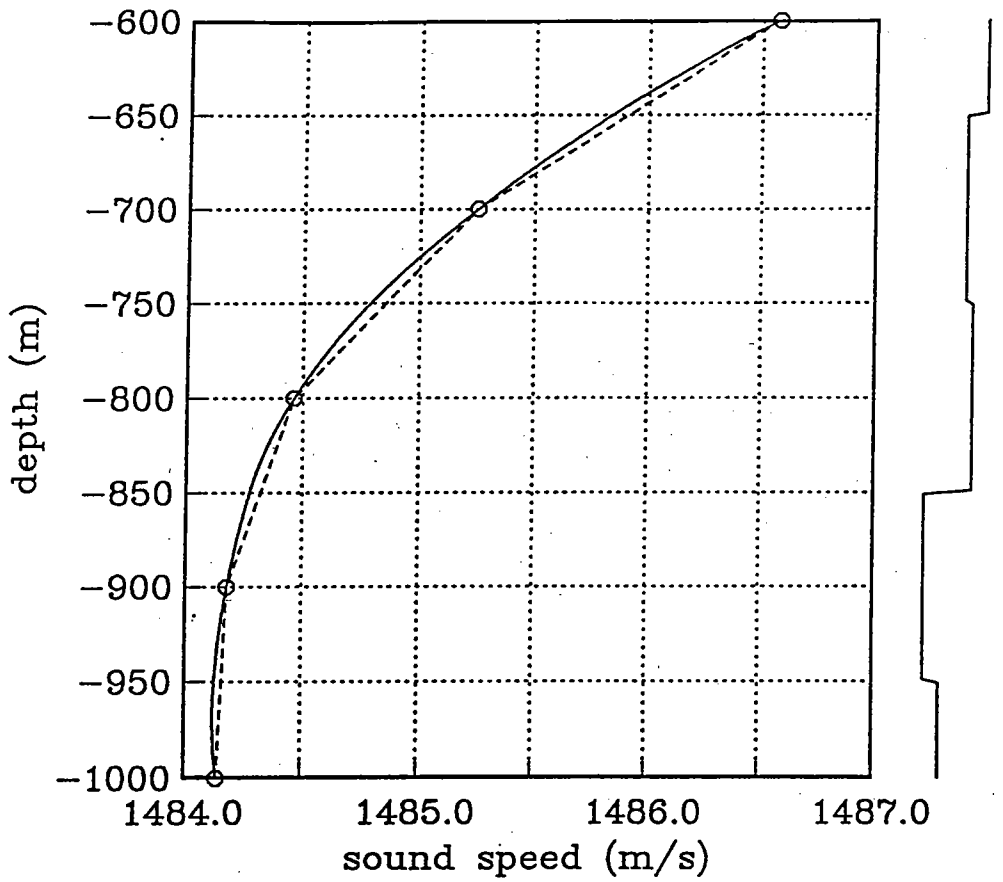


Figure 1. The solid line shows a section of a piecewise parabolic sound speed profile generated by Ray. The circles show the input sound speeds which were used by Ray to generate the profile. The curve to the right shows the piecewise constant second derivative of the solid–line profile, with sections centered on the input depths. In general there can be linear sections between the parabolic sections but none are shown in this example. The dashed linear segments show the constant gradient approximation used in circular arc ray tracing programs such as MPP and RDRYT.

Range Dependence

The way that Ray handles range dependence reflects the design philosophy of creating a simple model, consistent with the input data, that produces smooth wavefronts. Sound speeds are input as tables of sound speed versus depth at increasing ranges from the source. At each range the sound speed is smoothed and debiased as described above. Three different models of range dependence are provided.

The simplest model treats the ocean as a series of range independent sections. At each range where a sound–speed profile is defined the entire profile jumps abruptly to the new values. No compensation is made for obeying Snell's law at the interface. The rays that make up a wavefront pass through each interface at different depths. Since the amount the sound speed jumps varies with depth, this model introduces a modulation of the wavefront for each interface that is passed through. There are some regions of the ocean where this modulation does not produce serious problems but there are many regions where it is unacceptable. All attempts to eliminate this modulation by making some correction at the interface, such as obeying Snell's law, have failed. The corrections have changed the modulation but have not eliminated it.

The next level of sophistication lets the sound–speed profile vary linearly with range. Assume two adjacent profiles have been input at ranges r_j and r_{j+1}, then the sound speed at any intermediate range r will be

$$c(z,r) = c(r_j,z) + \frac{r - r_j}{r_{j+1} - r_j}\left(c(r_{j+1},z) - c(r_j,z)\right).$$

The equations of motion (see above) for a ray depend upon c, $\partial c/\partial z$ and $\partial c/\partial r$. In this model c and $\partial c/\partial z$ vary smoothly with range and depth while $\partial c/\partial r$ changes discontinuously at each interface. If this term is important in determining the path of a ray then the linear range dependent model will also contain modulations of the wavefront that depend upon the depths at which each ray passes through the interfaces.

Ray provides two implementations of the linear range dependent model in order to establish whether the contributions due to the $\partial c/\partial r$ term are significant. One of them keeps this term and the other one drops it. Using input from the Levitus data base, the differences between the wavefronts generated by these two implementations have been of the same scale as the numerical noise. This evidence has led us to conclude that the linear range dependent model is sufficient for the intended use of Ray.

If Ray is run with an environment where dropping the $\partial c/\partial r$ term produces a significant change in the wavefront then it is very likely that the linear range dependent model is inadequate for that environment even if the $\partial c/\partial r$ term is included.

Bathymetry and Bouncing

Version 1.0 of Ray provides several different options for dealing with bathymetry. The **soft** option keeps a record of the smallest distance to the bottom for each ray of a wavefront. Rays are still traced even if they go below the bottom. In these cases the distance-to-bottom becomes negative and the most negative value is recorded. The range where the smallest (or most negative) distance-to-bottom occurs is also recorded. The **absorbing** option terminates any ray that touches the bottom. The **reflecting** option reflects rays that hit the bottom.

The bathymetry data that Ray reads in can be smoothed in the same manner that the sound speeds are smoothed. Only one width parameter, **bath_smoothing**, may be input. The default value is 10 km. It is used for all bathymetry points whose nearest neighbor is at least 4 times greater than this width away. All bathymetry points that have a nearest neighbor closer than 4 times the bathymetry smoothing width have their smoothing width set to one quarter of the distance to the nearest neighbor. This scheme creates alternating sections of straight lines and parabolas. If the bathymetry smoothing width is set to 0 then no smoothing of the bathymetry is performed and Ray uses a bottom made up of straight lines connecting the input bathymetry points. No attempt has been made to "debias" the bathymetry, although the bathymetry points can be debiased before being passed on to Ray.

The heart of a bouncing algorithm consists of finding exactly where a ray path first intersects a boundary. Ray uses the following approach to find these intersection points. For every integration step taken, a quick check is performed to determine if the ray might have possibly crossed a boundary. Let r_1 and r_2 denote the range at the beginning and end of a step. If the possibility of crossing exists then the depth of this short section of the ray path and the depth of the boundary are both parameterized as parabolic in range and the difference of these two parabolas is taken to give the distance from the bottom as a function $dr = r_1 - r$,

$$\begin{aligned} z_{\text{ray}} - z_{\text{bottom}} &= z_{\text{miss}} \\ &\approx z_0 + dr\, z_1 + dr^2\, z_2. \end{aligned}$$

We assume that if the integration step size has converged then the first crossing of the boundary will occur either near the smallest positive real root of this equation or near the minimum of this equation if it has no real roots. In either case we take one integration step to the range $r_g^{(1)} = r_1 + r_{\text{root}}^{(1)}$ which is a rough guess of where the first intersection will occur. The expansion above is then repeated, this time expanding about $r_g^{(1)}$. If no real roots are found here then we assume that the ray does not cross the boundary at this step. If real roots are found then our next guess for the range where the first intersection occurs is $r_g^{(2)} = r_g^{(1)} + r_{\text{root}}^{(2)}$, which is found by adding the root of the new parameterization with the smallest absolute value to $r_g^{(1)}$. This process is repeated until either no real roots are found between r_1 and r_2 or $|z_{\text{miss}}| < z_{\text{tol}}$. The default

value for z_{tol} is 10^{-6} m. If no roots are found then we did not hit the boundary in this step. If the tolerance condition is met in the i^{th} iteration then we consider that the ray hit the boundary in this step at the range $r_g^{(i)}$ and the appropriate action is taken depending on the type of bathymetry requested.

3. Program Input

Invoking Ray with no arguments will produce an output similar to

```
/************************ Ray ************************/
/*
 *    Version 1.00 Friday, November 13, 1992, 3:33 pm
 *    Copyright (C) 1992,  Woods Hole Oceanographic Institution.
 *    Use -license flag for use, copying and distribution conditions.
 */

usage:
    ray initfile.ray [cmnd1] [cmnd2] ... [cmndN] [flags]
where:
   initfile.ray       is a ray initialization file
   cmnd1 ... cmndN    are command line parameters used in the
                      initialization file via $1$ ... $N$
flags:
   -d    (debug)     send some debugging info to stdout
   -l    (license)   display the license agreement
   -m    (makeinit)  generate an initialization file
   -h    (help)      send  detailed help to stdout
   -p    (parse)     send parsed init file to stdout
   -v    (verbose)   send information to stdout as we work
```

which lists all of Ray's command line options. Running Ray with the `-makeinit` option as in

```
Ray -m
```

causes Ray to print a prototype initialization file on the standard output. Capturing and editing this file is an easy way to quickly operate Ray.

The initialization file can then be sent back into Ray by specifying its name as the first command line parameter as in

```
Ray initialize.ray ,
```

which will run the Ray program with "initialize.ray" as the initialization file. It is always the first file read by Ray. Its format, structure, and syntax are described in this section. The formats of the other input files are described in an appendix.

Initialization File Format

The initialization file format will be familiar to people who work with the C programming language. All text enclosed between /* ...*/ is a comment and is ignored. All of the initialization parameters are organized into groups. The parameters within a group can be specified in two different ways. One way is a single line (or statement) for each parameter. For example,

```
model integration = rk_2;
model range_depend = grad_z;
```

will set the integration routine to be **rk_2** and the range dependence to be of the type **grad_z**. More information on exactly what these mean will follow below. Notice the semicolon at the end of each statement. The second method of specifying parameters within a group,

```
model {
    integration = rk_2;
    range_depend = grad_z;
    ...
};
```

sets these two parameters exactly like the example given previously. This method saves on redundant typing and encourages users to put all of the parameters within a single group together. There are currently six types of parameters that may be specified: string, dimensioned numerical, array of dimensioned numerical, dimensionless numerical, choice, and flag.

String Parameters

String parameters are used for specifying filenames. The name must always be inside a pair of double quotes, as in

```
input prof_file = "test1.ssp";
```

Dimensioned Numerical Parameters

Dimensioned numerical parameters are input as a number followed by units in parentheses. For example

```
receiver range = 1000.0 (km);
```

specifies that the receiver is at a range of 1,000 kilometers. The inclusion of units is not optional but choices of units are available. Two kinds of units are used, lengths and angles. Internally, all angles are in radians and all lengths are in meters. Table 1 lists valid length and angular units.

Table 1. Valid length and angular units.

unit	name	conversion to meters
(m)	meter	1.0e0
(km)	kilometer	1.0e3
(Nmi)	nautical mile	1.852e3
(ft)	foot	3.048e-1
(furlong)	furlong	2.0117e2
(parsec)	parsec	3.804e16
(cm)	centimeter	1.0e-2
(mm)	millimeter	1.0e-3
(Mm)	megameter	1.0e6
(lightyear)	lightyear	9.46e15
(angstrom)	angstrom	1.0e-10

unit	name	conversion to radians
(radians)	radians	1.0e0
(degrees)	degrees	3.14159265358979 / 180.0

Array of Dimensioned Numerical Parameters

An array of dimensioned numerical parameters is specified like a single dimensioned numerical parameter except the single number is replaced by a left brace, a list of numbers and a closing right brace as in

```
angles {
    specific =
        {1.0, 2.0, 3.0, 4.0, 5.0  } (degrees);
}
```

Note that the list of numbers can be delimited by either whitespace characters (which consist of space, tab, linefeed, and carriage return) or commas or both.

Dimensionless Numerical Parameters

Dimensionless numerical parameters are similar to dimensioned numerical parameters but they must not have any unit specification, even an empty "()" is not allowed. For example,

```
model margins = 100;
```

is a valid statement.

Choice Parameters

Choice parameters give the user a specific range of choices such as

```
model integration = rk_2;
```

which will set the integration routine to be `rk_2`. Other choices for this parameter are `rk_23` and `rk_4`. If a choice parameter is misspelled or otherwise inappropriate, then an error message will be generated showing where the problem occurred and what the valid choices are.

Flag Parameters

Flag parameters look similar to choice parameters. A flag parameter is set by mentioning its name as in:

```
output wavefront;
```

which will ensure that all of the wavefront variables for this run are put into the output file.

Initialization Parameters by Group

There are seven groups of parameters that are specified in the initialization file. They are: `input`, `output`, `source`, `receiver`, `angles`, `paths`, and `model`. All of the initialization parameters are described below, ordered by group.

Input Group

The input group specifies the additional files that will be read in by Ray. For example

```
input  {
    prof_file = "test1.ssp";
    bath_file = "test1.bth";
};
```

specifies that the file "test1.ssp" will be read in as a profile file containing sound–speed profiles, and the file "test1.bth" will be read in as a bathymetry file containing bathymetry. In order to maintain compatibility with existing software, the profiles and bathymetry may be specified together in a single MPP file with the line

```
input mpp_file = "test1.mpp";
```

If an `mpp_file` is specified, then it is an error to specify either a `prof_file` or a `bath_file`. The profile and bathymetry file formats are detailed in appendices. The `mpp_file` option is only included to maintain compatibility with existing software. This file format is not recommended and is not described in this report.

Output Group

The output group specifies the name of the output file generated by Ray, and what variables will be included in this file. The line

```
output mat_file = "test1.mat";
```

tells Ray to name the output file "test1.mat". The user can tailor what will be included in the output file with the following flag parameters,

output	*effect*
.....	.....
initialization;	save initialization parameters
filenames;	save filenames
date;	save the time and date of the run
sound_speeds;	save the sound–speed tables
bathymetry;	save the bathymetry table
paths;	save path information along each ray
wavefront;	save all ray parameters at receiver range
everything;	save all of the above
environment_only;	don't trace rays

If no **output** flags are specified then Ray will assume everything should be saved. To save the paths the user must explicitly set a **paths fixed_dr** or a **paths steps_per** in addition to the output flag. This is a safety feature to prevent accidentally filling a disk with detailed path information from thousands of rays. If **environment_only** is chosen then no rays will be traced regardless of the state of the other output flags.

Source Group

The source group specifies the depth and range of the source. The lines

```
source  {
    depth = 600.00 (m);
    range = 1000 (km);
}
```

will put the source 600 m below the surface at a range of 1,000 km

Receiver Group

The receiver group specifies the depth and range of the receiver. The depth of the receiver is not used by Ray Version 1.0. For example,

```
receiver  {
    depth = 1000.0 (m);
    range = 3000.0 (km);
}
```

specifies a receiver at a depth of 1000 m and a range of 3000 km.

Angle Group

The angle group contains all of the information needed to specify the launch angles of all the rays that will be traced. There are two ways to specify these angles, one is to provide two extreme angles and the total number of (equal spaced) angles as in

```
angles  {
    first = 15 (degrees);
    last = -15 (degrees);
    number = 3;
}
```

which tells Ray to shoot rays at the angles 15, 0, and -15 degrees. The user may alternatively specify specific angles to shoot as in

```
angles specific =  {
    1.0,
    2.0,
    3.0,
}(degrees) ;
```

which tells Ray to shoot three rays at initial angles of 1, 2, and 3 degrees. It is an error to specify angles in both formats.

Paths Group

The parameters in the paths group allow the user to save information about each ray as it is traced. For example

```
paths  {
    min_range = 0.0 (km)
    max_range = 1000 (km)
    fixed_dr = 1 (km)
}
```

will save path information for every ray as it is traced, every km over the region from 0 to 1,000 km. If the **min_range** and **max_range** are not specified (or if they are set to zero) then the path region is set to the entire region between the source and the receiver. If **steps_per** is used instead of **fixed_dr** then Ray will output information along each ray every **steps_per** steps. Since the step size can vary with depth, the range spacing of each path will vary. The information that is stored at points along a path can be tailored by specifying the following **paths columns** flags:

`paths columns`	*contents*
.....	.....
`range;`	range of the point
`depth;`	depth of the point
`time;`	time it took to get to the point
`angle;`	angle of the ray at the point
`speed;`	speed of sound at the point
`grad;`	∂ c/∂z at the point
`top_bounces;`	number of top bounces
`bot_bounces;`	number of bottom bounces
`ray_number;`	which ray the point belongs to
`everything;`	all of the above

If no columns are specified then all the columns are used. In addition if

```
paths include bounces;
```

is specified then every point where a ray bounces is included in the path in addition to all of the points generated by the `fixed_dr` specification or the `steps_per` specification. In order to only save the bounce points set `fixed_dr` to a range that is greater than the separation between the source and the receiver.

Model Group

The model group of parameters specify how Ray will model the ocean. The `model integration` choice controls which integration routine is used. The options are

`model integration =`	*effect*
.....	.....
`rk_2;`	2nd order Runge Kutta
`rk_23;`	2nd – 3rd order Runge Kutta
`rk_4;`	4th order Runge Kutta

The `model range_depend` choice controls how Ray interpolates between different sound–speed profiles. There are three options:

`model range_depend =`	*effect*
.....	.....
`none;`	no interpolation
`grad_z;`	interpolate c and $\partial c/\partial z$ but ignore $\partial c/\partial r$
`full;`	interpolate c and $\partial c/\partial z$ and include $\partial c/\partial r$

The `model bathymetry =` choice controls how Ray deals with bathymetry.

`model bathymetry =`	*effect*
.....	.....
`none;`	flat bottom, no bathymetry
`soft;`	pass through bottom
`absorbing;`	stop at bottom
`reflecting;`	reflect off bottom

The `soft` choice records the closest approach to the bottom (or the deepest descent into the bottom) for each ray. It also records the range where this closest approach occurs. The `absorbing` choice terminates any ray that touches the bottom. The `reflecting` choice lets rays bounce off of the bottom.

Within the `model` group is a subgroup that controls the size of the range step taken by the integration routines. Ray provides a facility for a depth dependent step size. The shape of this function is determined by the `prof_smoothing` widths which are described below. The overall size of the steps can be controlled by setting the step size multiplier, such as

```
model range_step multiplier = 0.5;
```

which halves all of the range steps. The maximum and minimum range step can also be controlled. This process occurs after the multiplier has been applied so that the units used for the minimum and maximum are not scaled by the multiplier. The lines

```
model range_step  {
    max = 10 (m);
    min = 10 (m);
}
```

will set the range step size to 10 m for all depths regardless of the value of the multiplier. The maximum and minimum can also be set to different values. If the maximum is less than the minimum, then the step size is set to the minimum.

There are two parameters in the `model` group that control how close Ray stays to the input sound–speed profiles. These are dimensionless numerical parameters and are set as in the following example

```
model debias  {
    factor = 1.0;
    iteration = 10;
}
```

A detailed description of what these two parameters do is contained in the Computational Algorithm section of this report.

There are six numerical parameters in the `model` group. these parameters and their default values are:

```
model  {
    prof_smoothing =  {
        5, 5, 5, 5, 5, 20, 12, 13, 12, 25, 25, 25, 50,
```

```
        50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
        125, 125, 250, 250, 250, 250, 250, 250, 250
    }(m);
    bath_smoothing = 10 (km);
    bottom_depth = 6000 (m);
    earth_radius = 6378.137 (m);
    z_tolerance = 1e-06 (m);
    margins = 100 ;
}
```

The `prof_smoothing` array contains the smoothing widths for the sound–speed profiles as discussed in the Algorithms section. The `bath_smoothing` parameter contains the single width used for smoothing the bathymetry. The `bottom_depth` parameter controls the absolute maximum depth that can be used in a ray trace. If there is bathymetry specified that is deeper than `bottom_depth` then `bottom_depth` is automatically set to the deepest bathymetry point. The `earth_radius` parameter is used for the spherical earth correction. If it is set to 0 then no spherical earth correction is used. The `z_tolerance` parameter controls how close a ray need get to a boundary before it bounces off of it as described in the Bathymetry and Bouncing section. It is currently set to 10^{-6} m. The `margins` parameter controls how much extra depth is allowed over the top and under the bottom.

The default values of these numerical parameters should work fine in most situations. If they are set incorrectly spurious results may be obtained.

Command Line Substitution

Often the situation arises when several ray traces need to be performed which differ in only one or two parameters. To facilitate work on such problems, Ray has the ability to do command line substitution in the initialization file. This allows repeated use of the same initialization file for a number of different ray traces. Everywhere in the initialization file where a `$1$` occurs, the second command line parameter is substituted for the `$1$`. Everywhere that a `$2$` is encountered, the third command line parameter is substituted for every `$2$`, and so on. For example suppose the "init.ray" contains the lines

```
input prof_file = "$1$.ssp";
input bath_file = "$1$.bth";
model integration = $2$;
```

and then suppose that Ray is invoked by

```
Ray init.ray run17 grad_z
```

then this will have the same effect as if the initialization file contained

```
input prof_file = "run17.ssp";
input bath_file = "run17.bth";
model integration = grad_z;
```

Note that although replaceable parameters are allowed to split a string (as in `"$1$.prf"` above) it is not valid to try to split up a word that is not in quotes with a replaceable parameter. For

example, the line

```
model integration rk_$1$;
```

is not valid and will generate several error messages.

Although Ray has a large number of inputs parameters which make it adaptable to a variety of needs, most users will never want to access all of these parameters since Ray has a reasonable set of default values. However, There is some information that needs to be included in the initialization file in order for Ray to run. The following listing shows a very short initialization file which contains all of the needed information.

```
/**** Minimum information initialization file ***/

input prof_file =  "test1.ssp";
output mat_file =  "test1.mat";
source depth    =   500 (m);
receiver range  =   220 (km);
angles  first   =    15 (degrees);
angles   last   =   -15 (degrees);
angles number   =   500;
```

If a bathymetric ray trace is desired then add the line

```
input bath_file = "fname.bth";
```

and Ray will read in the bathymetry information from this file and set `model bathymetry = reflecting;`. If path information is desired then add a line such as

```
paths fixed_dr = 1000 (m);
```

which will cause Ray to save information along the path of each ray every 1000 m.

4. Program Operation

Ray is written in the ANSI C language. The source code is split up into several files (which are called "translation units" in C jargon). All of these files have a ".c" extension. Many have an associated header file with the same name and a ".h" extension. A general outline of what Ray does when it runs follows below. Each major section of the outline includes the name of the translation unit which is primarily in charge of that section.

Outline

I. Initialization: **initray.c**
   a. Print banner and copywrite notice.
   b. Read command line parameters. Exit here if there are unknown command line parameters.
   c. Open and read initialization file (there are errors in initialization file then print error messages and exit).
   d. Open appropriate MPP, profile and bathymetry files. Open the output file.
   e. Save all initialization parameters, start time, filenames in the output file.

II. Read auxiliary input files: **mppio.c**
   a. Read MPP file containing sound–speed profiles and bathymetry.
   b. Read profiles file.
   c. Read bathymetry file.

III. Pre-process the environment: **preray.c**
   a. Get maximum number of sound–speed depths.
   b. Make a table of widths.
   c. Make idex table.
   d. Make rstep table.
   e. Debias the sound speeds.
   f. Make sound–speed tables.
   g. Make speed at receiver table.
   h. Make bathymetry table.
   i. Save environment (if requested to do so).

IV. Do the ray trace: **ray.c**
   a. For every angle: shoot one ray.
   b. Save wavefront information.

Program Function by Translation Unit

helpray.c Contains the text of Ray's extensive help documentation and a little function to print it out.

initray.c Prints out the banner and copywrite notice. Reads and processes command line parameters. If any errors occur (such as an unrecognized command line option) the usage text is displayed along with an error message. Reads and processes the initialization file. If any

errors occur while reading the initialization file, each offending line is printed out along with an error message showing where in the line the error occurred and what the error was. The total number of errors is displayed and then the program exits. If the initialization file is parsed successfully then its contents are checked for ambiguities. If any ambiguities occurred then all ambiguities in the initialization file are reported and the program exits. If no ambiguities occurred then we open the other input files and the output file.

mppio.c Reads MPP, profile and bathymetry files. Puts all of this information into a single `mpp` structure. Since we use **reada.c** (see below) to read ASCII files, comments may be freely placed throughout these files be inclosing them in `/* ...*/`.

outray.c Contains functions to write variables to the output file in binary MatLab format. Ray has one machine dependent parameter. It is called `Mat_mach_num` and is located in **outray.h**. There is a nearby comment which explains what to set this parameter to according to which machine Ray will be compiled on. If it is set incorrectly, Ray will run without errors but MatLab will be unable to read Ray's output.

preray.c Preprocesses the environment. Constructs an index table and tables of sound–speed parameterizations, bathymetry parameterizations and automatic rangestep parameterizations. All of these functions are accessed through the function *preprocess()*.

ray.c Does the actual ray tracing. Contains all of the functions for integrating the equations of motion and bouncing rays off of the surface and the (flat) bottom. Also contains the function *main()* which is the main routine for ray.

reada.c Reads ascii files and splits them into tokens. A token is defined as a contiguous series of non-whitespace characters separated by whitespace characters (space, tab, linefeed, and carriage return) and delimiter characters (`= , / " { } ( ) ; $`). Each delimiter character is a token. Strings and comments are handled appropriately. All text between a `/*` and a `*/` is ignored. All text between pairs of double quotes (`" ... "`) is treated as a string. Comments may be nested.

utils.c Contains small functions and macros used by many of the other translation units. The macros are in the file **utils.h**.

version.c Contains the version number and version date.

Detailed Outline of Tracing One Ray

Ray uses two different structures for tracing rays. The structure `rinfos` contains all of the information about a ray that is used in creating a wavefront. The following fragment from **ray.h** details all of the components of the `rinfos` structure.

```
struct rinfos {          /*  all  start, end info about a single ray  */
double    sangle,        /*  angle of ray at source                   */
          sdepth,        /*  depth of ray at source                   */
          srange,        /*  range of the source                      */
          rangle,        /*  angle of ray at receiver                 */
```