MICROLENS: Short Description of an Optimized Ray-Shooting Program

Introduction

This is a brief description of the structure of the microlensing program that I developed from 1988 to 1990. A more complete overview on the program can be found in my thesis ("Gravitational Microlensing", available as preprint MPA550 (1990), Max-Planck-Institut für Astrophysik, Garching), especially Chapters 2 and 3 and Appendix A. The description here concentrates on the way the program is coded. A sketch of the dependences of different subroutines on each other can be obtained on request.

Some general remarks at the beginning: before starting the program one should get a feeling what it is doing, but one should not attempt to understand each programming detail. Especially the routines to set up the cell/lens structure may look quite obsolete at first glance. Whenver one changes something and it does not work as expected, one will *have* to understand this part of the program anyway. I strongly recommend checking the output *graphically*, either directly on the screen of a workstation, or on a plotted output, because this is a very nice way to see possible errors (cf. Section 2.4 in thesis). One last remark: it is possible/likely that there are still bugs in the program. I would appreciate being informed, if anybody using the program finds an error.

Basic Structure of the Program

The program combines the backward-ray-tracing method with a hierarchical tree code. The implementation makes use of two more steps of improvement, the fact that the same lens/cell structure is used by a bunch of light rays inside a small area, and that for a few rays in an even smaller area the contribution of the cells is not determined directly any more, but interpolated.

The program in its current form (October 1991) consists of 18 `FORTRAN` files, 2 of which contain more than one subroutine (`massf.f` and `output.f`). Altogether these routines contain about 2500 lines of code; this number includes comments. The actual number of `FORTRAN` lines is about 1400. Here is a list of all `FORTRAN` files (number in front of filename gives number of lines):

```
250 main.f
 54 detko.f
254 input.f
100 massf.f
 50 niveau.f
100 numstar.f
385 output.f
130 pos.f
 56 poslc.f
 65 rands.f
100 raycount.f
```

```
 228 setcell.f
 100 setmult.f
  90 setstar.f
 193 shootcl.f
 130 shootfix.f
 125 shotayco.f
  90 sort.f
2500 total
```

The program works essentially in two parts: first the positions and masses of the (micro-)lenses are set up, and subsequently the lenses are put into square cells of different sidelength. This part usually takes only a tiny fraction of the CPU-time. Once this set up is completed, the ray-shooting starts: light rays are shot in a regular grid backwards from the observer through the lens plane. Here the gravitational deflection of each ray due to the lenses is determined. Then the deflected rays are followed to the source plane, where they are collected in a square array, containing typically `IPIX`$^2 = 512^2$ or 1024^2 small square pixels. When the calculation is finished, i.e. after a very large number of rays was shot, the array with the square pixels contains the number of light rays that hit each pixel. This array can be directly written out (it is called `CONUF007` in the current implementation). In the current version this `write`-statement (in subroutine `output.f`) is commented out. A more convenient output file is called `IRIS007`. In this file the number of rays is already converted in astronomical magnitudes, relative to the theoretical average magnification. Here the average magnification always gets a value of 1024, magnification by one magnitude increases by 256, demagnification by half a magnitude would thus correspond to a value of 896. This file is unformatted and contains the array with `IPIX`^{2} numbers in the format `INTEGER*2`. This file is quite handy for graphic display of the magnification patterns. Lightcurves can be obtained as cuts through these magnification patterns, the data have to be convolved with the appropriate source profile. All length values in the program are given in Einstein ring radii of unit mass, in the lens or in the source plane, respectively.

The program in its current version uses three CONVEX system routines. They are not essential for the program. They are subroutines `idate, time, cputime`. The first two are called twice, once in subroutine `input.f` and once in in subroutine `output.f`. They return the day of the year and the time of the day, respectively. All this is used for the output file `dat.007`, see below (all output files contain a three-digit job number; here I will always use "007" as a job number). So one can either comment out these statements, or use the appropriate routines of the local computing system. The routine `cputime` returns the cputime that was used from the start of the program. It is called many times in routines `main.f, setstar.f, setcell.f`, in order to get an overview of the time used in different parts of the program. These times are also printed out in `dat.007`, see below. Again, these calls are not essential, they are commented out in the current version of the program; however, these values of the CPU-time used in different parts of the program turned out to be quite useful, so I recommend substituting `cputime` by a locally available routine. There are different `stop`-statements in many parts of the program. Usually they are accompanied by a number and a `write`-statement, that writes

on unit 44. These `stop`-statements indicate that something went wrong, the accompanying `write`-statements usually tell where it happened.

One of the advantages of the program is that it can use a large number of lenses. Calculations were done with up to 10^6 lenses. However, an essential feature for very large numbers of lenses is the possibility to define eight-byte-integers (i.e., a `FORTRAN`-statement `integer*8`, or an equivalent compiler option that defines integer numbers by default to be eight bytes big). These features are available on, e.g. CONVEX or CRAY, they are, however, not available on a SUN SPARCstation IPC. The program still works, but only for smaller numbers of lenses. The problem is the way the position of a lens is identified (see below in the description of subroutine `numstar.f`). With `integer*8` the variable `levelmax` can be as big as 30, whereas in the other case it cannot be larger than 15 (because positive integer numbers bigger than 4^{15} cannot be used with 4 bytes per integer!). The number of lenses that still can be used with 4 bytes per integer is of the order of 20000; it varies a little, the limiting fact is the distance of the two closest lenses, which should be larger than the side length of the smallest cells used (which are in level `levelmax`, the sidelength is $2 \cdot$`rstars`$/2^{\text{levelmax}}$). In practice one should play around with the input seed number for the random number generator, `arand`, if the program stops for a lens number close to the limit given above, because for a different realization of lens positions the smallest distance between two lenses will change a little.

Description of Input Data

A typical input file (that is read by the subroutine `input.f`) is called `input` and has the form

```
   0.345 arand (random seed number; ex:  0.345)
       0 debug (parameter:  0 - standard run)
   0.200 sigmas (matter in compact objects)
   0.000 sigmac (continously distributed matter)
   0.000 gamma (external shear)
   0.600 eps (accuracy parameter)
      10 nray (number of rays per row in level 1)
1.000000 minmass (lower cut off in mass spectrum)
1.000000 maxmass (upper cut off in mass spectrum)
  -2.350 power (index in mass spectrum)
  10.000 pixmax0 (ex.:  10.000)
 -10.000 pixminx (ex.:  -2.000)
 -10.000 pixminy (ex.:  -2.000)
  20.000 pixdif (ex.:  4.000)
   0.050 fracpixd (ex.:  0.020, 0.050, 0.250 ...., 2.000)
       0 iwrite (iwrite >0:  write these values on fort.99)
```

Here is a description of the input parameters:

`arand` - is the seed number for the random number generator in subroutine `rands.f`. It is used for the x- and y-positions of the lenses and for their masses, if a power law mass spectrum is used. Note that the same input number `arand` will

guarantee the same positions and masses of stars in different runs (if nothing else has been changed); typical values: `0 < arand < 1` .

`debug` - the debug parameter is not effective in the current version; however, it turned out to be very useful for all kinds of changes, or `write`-statements. Is is passed to each subroutine via a `common`-statement, so it can be easily referred to (to, e.g., write down position of stars, or positions of rays, if `debug` is equal to a specified value); typical values: `debug = 0,1,2,...`

`sigmas` - surface mass density in *compact* objects, e.g. stars (in units of the critical surface mass density); typical values: `sigmas = 0.2, 0.85, 2.7`.

`sigmac` - surface mass density in *continuously* distributed matter (in units of the critical surface mass density); typical values: `sigmac = 0.0, 0.4, 1.0, 2.3`.

`gamma` - (normalized) external shear; typical values: `gamma = 0.0, 0.24, 1.3, -0.7`.

`eps` - accuracy parameter; this value determines whether to include a certain cell into the determination of the deflection angle or whether to use the four subcells (and so on). If the ratio between the sidelength of the cell and the distance between ray and center of mass of the cell is smaller than `eps`, then this cell is used. If this ratio is bigger than `eps`, then this cell is resolved into four subcells or into individual lenses, if the lowest level has been reached already (more detailed desription in thesis, Section 2.1).

`nray` - this parameter determines the number of rays to be shot per column and per row in the highest shooting hierarchy (in the case of no external shear the shooting takes place on a square grid, in which the number of rows is equal to the number of columns. If external shear is not equal to zero, `nray` is the number of rays along the larger side of the rectangle). In other words: for a case with no external shear, in which shooting takes place in a square, the total number of rays to be shot is $\texttt{nray}^2 \cdot \texttt{(factor1-1)}^2 \cdot \texttt{(factor2)}^2$); the number $\texttt{(factor1-1)}^2$ indicates the numbers of rays shot in level 2 for each ray in level 1, and the number $\texttt{(factor2)}^2$ gives the numbers of rays shot in level 3 (those that are collected in pixels) per each ray in level 2. Typically `factor1` is equal to 11, and `factor2` is equal to 10, which means the total number of rays shot in level 3 is $\texttt{nray}^2 \cdot 10^4$. For test runs values of `nray = 5, ..., 20` are appropriate (which result typically in about 2 to 5 rays per pixel), for production runs numbers between `nray = 50` and `nray = 250` were used.

`minmass` - lower cut-off in mass, if a power spectrum is used (it is in *unit* mass, which conveniently is interpreted as 1 $M_\odot$); typical values: `minmass = 1.0, 0.2, 0.01, 0.001`.

`maxmass` - upper cut-off in mass, if a power spectrum is used; typical values: `minmass = 1.0, 10.0`.

`power` - power index of mass spectrum; for Salpeter mass spectrum `power = -2.35`.

`pixmax0` - half side length of the largest magnification pattern that is to be calculated for a given star field (in units of Einstein ring radii projected to source plane, for lens of unit mass). This value has to be kept constant for *zooming* calculations, in which some parts of the magnification pattern can be seen in higher resolution;

typical values: `pixmax0 = 50.0, 10.0`.

`pixminx` - x-coordinate of lower left corner of magnification pattern (in units of Einstein ring radii in source plane). This value can be changed for *zooming* calculations; typical values: `pixminx = -50.0, -10.0, -0.32`.

`pixminy` - y-coordinate of lower left corner of magnification pattern (in units of Einstein ring radii in source plane). This value can be changed for *zooming* calculations; typical values: `pixminy = -50.0, -10.0, -0.32`.

`pixdif` - side length of magnification pattern (in units of Einstein ring radii in source plane). This value should be changed for *zooming* calculations; typical values: `pixminy = 100.0, 20.0, 4.0, 0.8`.

`fracpixd` - fraction of the length of the shooting square, that has to be added on either side, relative to the rectangle that would be mapped onto the magnification pattern if there was only smooth matter. This value should be of the order of one or two Einstein ring radii, i.e. for `pixmax0 = 50` the value of `fracpixd` should be about 0.02, for smaller `pixmax0` it has to be accordingly bigger. This value has to be adjusted experimentally. If it is too big, very many rays are wasted, they do not reach the region of the magnification pattern. If it is too small, there are regions in the magnification pattern, close to the boundary that are devoid of rays, and therefore give wrong results (light curves, probability distributions, ...). Fortunately this can be studied very easily just by looking at the magnification patterns.

`iwrite` - control variable to write out what was read in in subroutine `input.f`; typical values: `iwrite` $= 0$: nothing is written; `iwrite` $\neq 0$: input data are written on file `fort.99`.

Description of Individual Subroutines

In all subroutines and functions the statement `implicit none` is used, which means each variable has to be defined explicitly. The `FORTRAN` default definitions (e.g., all variables starting with letters `i-n` are integers) are **not** valid! I tried to pass the variables by name, and to minimize the number of `common`-statements, but I did not succeed in avoiding them completely. I furthermore attempted to use a unified header for each routine which defines in one sentence the purpose of that routine, describes the input and output variables, and lists the subroutines/routines that call this routine and those are are called from this routine. However, as this changed frequently in the process of developing the code, it is not guaranteed that everyting is up to date. Now I will describe the purpose of every subroutine/function:

`main.f:`

The main program basically calls once at the beginning the routines that read input, do some inititializations, set up the lens field, determine the lens/cell structure and the higher multipole moments of the cells: `input.f, setstar.f, numstar.f, setcell.f, setmult.f`. Then `main.f` calls repeatedly the shooting routines on different levels, after determining the x- and y-coordinates of the rays on a rectangular grid: `shootcl.f,`

shootfix.f, detko.f, shotayco.f. After finishing shooting main.f calls output.f, in which all the writing is done.

detko.f:

Determines the coefficients phi(1), ..., phi(8), which are used for the bidimensional interpolation of the deflection angle due to the lenses that are bunched together in cells in subroutine shotayco.f. This interpolation happens only for shooting on the lowest level, in which rays are "really" shot and collected in pixels. Details can be found in my thesis, especially Section 2.3 and Appendix A.2; there, the coefficients phi(1), ..., phi(8) are called $\Psi_1, \Psi_2, \Psi_{11}, \ldots, \Psi_{1112}$.

input.f:

Reads the values of the 16 input parameters (see above) from file input. Reads furthermore the number of the current job from file jobnum. This number is then increased by 1, and the file is updated (initially the file jobnum should contain the three digits 001 for the first job). Then routine input.f initializes some values and determines some numbers, e.g. the average magnification, the radius of the circle in which lenses should be distributed, the number of lenses, the distances of adjacent rays in levels 1, 2, and 3, and more.

double precision function massf.f:

Sets masses of the lenses according to the power law mass function $f(m)dm = m^{power}dm$, if input parameters minmass < maxmass (if minmass = maxmass, then all lenses have the same mass). The file mass.f contains one more double precision function, namely massav. This function determines the average mass $< m >$, which is used for the determination of the size of the lens circle (in subroutine input.f).

integer function niveau.f:

Calculates level of the cell, whose index is passed as an argument. Example: cells with index $1 \leq$ cellnum ≤ 4 are in level 1, cells with index $4+1 \leq$ cellnum $\leq 4+16$ are in level 2, cells with index $16+4+1 \leq$ cellnum $\leq 4+16+64$ are in level 3, and so on (details in thesis, Section 2.1).

subroutine numstar.f:

Determines position of each lens in each level, i.e. the quadrant the lens is in at that level: upper left quadrant gets index 0, upper right quadrant gets index 1, lower left quadrant gets index 2 and lower right quadrant gets index 3. This number is then multiplied with four to the power (levelmax-level). Then in the next level the position is again investigated for the respective quadrant, its contribution is added to the value above, and so on. At the end the value is put into the vector indlens(i) for the according lens number i; it contains, in compressed form, the position of each lens in each level. After attaching these numbers to each lens, the lenses are sorted according to its value. Example: consider a square with side length 1.0, ranging from (0.0,0.0) to (1.0,1.0). Assume the value of levelmax to be 3 (in practice it is 15 or 30). Lens position (0.87,0.52) would get a value

"1" for the first level in the hierarchy, because it is in the top right quadrant. This number is multiplied with $4^{3-1} = 16$, because it is level 1. Then a square with side length 0.5 is considered, ranging from (0.5,0.5) to (1.0,1.0). Here the lens in the lower right quadrant, i.e. it gets a value $3 \cdot 4^{3-2} = 12$ for this level. In the third level the square with side length 0.25 ranges from (0.75,0.5) to (1.0,0.75). Here the lens is in the lower left quadrant and gets a value $2 \cdot 4^0 = 2$. Up to the third level the value of `indlens` would therefore be $16 + 12 + 2 = 30$. This number contains all information about the relevant locations in the different levels.

`subroutine output.f:`

Writes data on output files `dat.007, CONUF007, IRIS007`. In file `dat.007` there are important numbers describing the current job: number of lenses used (`nlens`), number of cells that contain at least two lenses (`ncell`), average number of lenses and cells used per ray (`avlens, avcell`), cputime (in seconds) per ray, average number of rays per pixel (`rayperpi`), number of rays corresponding to magnification one (`rayamp1`), numerical average magnification (`ampav`), theoretical average magnification (`ampth`), total number of rays shot (`rayshot)`. Furthermore all input data (surface mass density, shear, side length and position of magnification pattern, ...) are printed in file `dat.007` , and many more useful numbers. An example of an output file is shown below. The file `CONUF007` contains the actual numbers of rays for each `IPIX`^{2} pixels (in the current version this output is commented out). This data is converted into astronomical magnitudes Δm, relative to the average magnification, and printed out in file `IRIS007`. Here average magnification gets the value 1024, and one magnitude up or down is +256 or -256. The magnification pattern is just a graphical display of this file. The transformation of the ray-per-pixel data into these magnification (or "color") data is being done by the subroutines `calcpix` and `calcpix1`, which are also contained in the file `output.f`.

`integer function pos.f:`

This routine is used in the process of determining which cells or lenses should be used for a certain light ray in `shootcl.f`. It determines – for a certain index – if there exists a lens with this index (then the function value is negative, and its absolute value gives the position in the vector `lensdata`), or a cell with this index (positive function value, which gives position of data in `cell`, or neither one (function value zero). This function is usually called for all four quadrants of a non-empty cell, and subsequently for all non-empty subcells, and so on.

`integer function poslc.f:`

Is called from function `pos` in order to actually determine the position of an existing cell or lens with index `index` in the respective data vectors.

`subroutine rands.f:`

Generates a random number `ar`; for the same seed input numbers (`arand, cr1, cr2)` the same sequence of random numbers is produced.

`subroutine raycount.f:`

Converts `double precision` source plane coordinates of rays into `(integer)` pixel numbers, and adds one ray to the according pixel. This is done twice, for the actual pixel field with `IPIX`^{2} pixels, and for the test field with twice the side length, but only `IPIX1`^{2} pixels (usually `IPIX` = 500 or 1024, whereas `IPIX1 = 100`). This second field can be used to check if the main field should have any problems at the boundaries, see Section 2.4 in thesis.

`subroutine setcell.f:`

Starting at level 1, for each subcell (i.e., each quadrant) the total mass, and the center of mass is determined by summing over all lenses that are in this subcell. This is iteratively done for each subcell, unless it contains zero (then it is ignored) or one lens. In the latter case this lens gets a new index, that is the index (in vector `indlens`) of that cell, in which the lens is single. This is done by comparing two neighbour lenses in old array `indlens()` (lenses that had similar indexes to begin with, are in the same quadrants in levels 1, 2, 3, ...; a slight difference in the index results from the highest levels). After each lens has got a new value, the lenses are sorted again, according to the new indexes in `indlens()`.

`subroutine setmult.f:`

Determines multipole moments (quadrupole, octopole, ..., 64-pole) of all cells that contain more than one lens. This routine is called only once, when all lenses and cells are set up.

`subroutine setstar.f:`

Attaches two random position coordinates and one mass coordinate to each lens. Lenses are distributed inside a circle with radius `rstars`.

`subroutine shootcl.f:`

Determines the lenses that have to be included directly in the determination of the deflection angle, and those that can be bunched together in cells. The sizes of the cells increase with increasing distance to the light ray. This is done for each ray in highest shooting level. No actual deflection angle is calculated here, only the cell/lens-structure is passed to the main program.

`subroutine shootfix.f:`

Determines deflection angle due to lenses that are bunched together in cells. This is done for an array of `factor1`^{2} rays in level 2. These deflection angles are then passed to rays in level 3, where the contributions of the cells are being interpolated.

`subroutine shotayco.f:`

Actual determination of deflection angle for "real" rays in level 3. The contribution of the cells is being interpolated, the contribution of the lenses is calculated directly for each ray here. Then the subroutine `raycount.f` converts the ray coordinates into pixels and adds up the rays.

`subroutine sort.f:`

Sorts cells or lenses according to the `index` of each cell/lens (uses a slightly changed routine from the *Numerical Recipes*).

Benchmarks

Here I give the CPU-time used by the microlensing program on two different machines (SUN SPARCstation IPC and CONVEX 210) for various types of optimization. Only the optimization level `-O2` on the CONVEX uses vectorization. It is easy to see how much this speeded up the calculation. The input file used for this benchmarks is the same as is shown above. For values of the surface mass density closer to unity this speed up is even higher An example output file (that for the optimization `-O2` on the CONVEX) is shown below.

CPU:	SPARCstation IPC:	CONVEX 210:	CONVEX 3440:
no optimization:	1213.8 sec	234.4 sec	
optimization -O0:	—	133.0 sec	
optimization -O1:	1131.9 sec	104.3 sec	91.3
optimization -O2:	278.4 sec	41.6 sec	33.8 sec

Example output file

Here I present a typical output file `dat.007`. It was obtained with the input file as given above, run on a CONVEX 220, compiled with optimization level `-O2`:

```
|                 1 jobnu:      number of job
|                 date and time of start:          1.11.91    11:05:15
|                 date and time of stop:          1.11.91    11:06:02
|CPU-time:              41.41 sec =       0.69 min =     0.01 h
|
| important parameters:
|                163 nlens:      total number of lenses within rstars
|                121 ncell:      total number of cells
|              74.23% ncell/nlens:  ratio in percent
|             18.917 avlens:      average number of lenses used per ray
|             31.331 avcell:      average number of cells  used per ray
|           0.000034 CPU-time/ray:  shooting time (CPU-sec) per ray
|              3.035 rayperpi:      average number of rays per pixel
|                 81 pixhigh:      highest number of rays per pixel
|                  0 pixlow:      lowest number of rays per pixel
|              2.116 rayamp1:        number of rays for amplification 1
|              1.435 ampav:        (numerical)   average amplification
|              1.562 ampth:        (theoretical) average amplification
```

```
|            1210000 rayshot:      total number of rays shot in level 3
|             758838 raysarr:      number of rays arrived in square
|             451162 rayslost:      number of lost rays
|
| INPUT parameters:
|              0.345 arand:      input for random number generator
|                  0 debug:      parameter for debugging the program
|              0.200 sigmas:    surface mass density in compact objects
|              0.000 sigmac:    surface mass density in compact objects
|              0.000 gamma:      global shear
|              0.600 eps:          accuracy parameter, (0 <= eps <= 1)
|                 10 nray:        number of rays per row in level 1
|              1.000 minmass:  lower cutoff of mass spectrum
|              1.000 maxmass:  upper cutoff of mass spectrum
|             -2.350 power:    exponent of mass spectrum (Salpeter:  2.35)
|             10.000 pixmax0:  size of field for distribution of stars
|            -10.000 pixminx:  left border of receiving field
|            -10.000 pixminy:  lower border of receiving field
|             20.000 pixdif:    size of receiving field
|              0.050 fracpixd:    fraction added
|
|                500 ipix:        size of pixel matrix  IRISxxx
|                 11 factor1:  linear multiplier from level 1 to level 2
|                 10 factor2:  linear multiplier from level 2 to level 3
|
| OUTPUT parameters:
|              1.000 boa:          b / a  :          (1-gamma) / (1+gamma)
|              1.000 bmsoams:  b-s/a-s:  (1-gamma-sigma)/(1+gamma-sigma)
|              1.000 massav:    average mass of lenses (in solar masses)
|            163.000 masstot:  total mass in all lenses (in solar masses)
|             27.500 raydif:    length of shooting region (level 1)
|                 -5 jxmin:      number of leftmost  ray in level 1
|                  5 jxmax:      number of rightmost ray in level 1
|                 -5 jymin:      number of lowest    ray in level 1
|                  5 jymax:      number of highest   ray in level 1
|            -13.750 rayminx  coordinate of leftmost  ray in level 1
|             13.750 raymaxx  coordinate of rightmost ray in level 1
|            -13.750 rayminy  coordinate of lowest    ray in level 1
|             13.750 raymaxy  coordinate of highest   ray in level 1
|           2.750000 raydist0:  distance of adjacent rays in level 1
|           0.275000 raydist1:  distance of adjacent rays in level 2
|           0.027500 raydist2:  distance of adjacent rays in level 3
|           0.040000 pixlen(1):  pixel length in units of Einstein radii
|           0.275000 pixlen(2):  pixel length in lens array PIXLENS
```

```
|            28.517 rstars:        radius of circle that comprises stars
|         15.111250 xur:           coordinates of leftmost ray in level 3
|         15.111250 yur:           coordinates of lowest   ray in level 3
|           1210000 rayth:         number of rays expected to be shot
|            50.248 avtot:         average # of cells+lenses used per ray
|          0.308270 avtot/nlens:   avtot divided by total # of lenses
|                 8 highle:         highest level used for cell hierarchy
|             57568 indlens(nlens) number of highest lens
|
| CPU time used for different parts of the program:
|             0.013 times(1):      time in CPU-sec for subroutine setstar
|                       0.008 times(11):  CPU-sec for subroutine rands
|             0.013 times(2):      time in CPU-sec for subroutine numstar
|             0.032 times(3):      time in CPU-sec for subroutine setcell
|                       0.009 times(12):  CPU-sec for subroutine setmult
|                       0.007 times(13):  CPU-sec for     setup
|                       0.006 times(14):  CPU-sec for sorting lenses
|                       0.010 times(15):  CPU-sec for sorting cells
|            41.333 times(7):      time in CPU-sec for    shooting
|                       0.338 times(4):  CPU-sec for shootcl  ( 0.82%)
|                       3.964 times(5):  CPU-sec for shootfix ( 9.59%)
|                      36.662 times(6):  CPU-sec for shotayco (88.70%)
|            41.415 times(8):      time in CPU-sec whole run
|
|       163 lenses   after sorting:
|        number               index      x-coord       y-coord      mass
|             1                  22    -15.815690     21.515290     1.00
|             2                  29    -23.003109      7.916561     1.00
|             3                  33     -8.932694      8.253564     1.00
|             4                  34     -2.073513     10.981506     1.00
|             5                  49     19.946109      9.050930     1.00
|
|           159               17584     -0.965095    -23.090911     1.00
|           160               18478      6.018271    -11.913793     1.00
|           161               18479      5.700723    -12.256803     1.00
|           162               57566    -24.507412    -12.944537     1.00
|           163               57568    -24.699485    -13.281454     1.00
|
|       121  cells   after sorting:
|        number               index      x-coord       y-coord      mass
|             1                  1    -12.840681     10.536490    39.00
|             2                  2     14.311249     11.533470    40.00
|             3                  3    -13.289734    -13.327468    45.00
|             4                  4     12.266782    -12.225731    39.00
```

```
|            5                   5    -17.057504     18.314218     5.00
|
|          117                2342     -2.009017      6.829369     2.00
|          118                3597    -24.603448    -13.112996     2.00
|          119                4395     -1.341597    -22.965279     2.00
|          120                4619      5.859497    -12.085298     2.00
|          121               14391    -24.603448    -13.112996     2.00
|
| distribution of cells and lenses in different levels:
|             level              cells                lenses
|                 1                  4                    0
|                 2                 16                    0
|                 3                 44                    8
|                 4                 36                   70
|                 5                 16                   51
|                 6                  4                   26
|                 7                  1                    6
|                 8                  0                    2
```

We will not comment on every variable printed out, just a few general remarks: The header contains the job number, date and time of start and end of program, and the CPU-time used (these values are obtained with CONVEX system routines `idate, time, cputime`, which should be replaced by the appropriate routines in the local computer system). The group of *important* parameters is explained partly in the description of the subroutine `output.f` above. The next group of parameters if just the numbers that were read in from the input file `input`. Then the values of different parameters is shown, like `boa` which gives the ratio (1-`gamma`)/(1+`gamma`), or positions (in integer numbers or real coordinates of lower left or upper right rays), and so on. Furthermore the total mass of all lenses `masstot`, the radius of the lens circle `rstars`, the average number of (lenses+cells) used per ray `avtot`, the highest level used `highle` and the highest index of any lens `indlens(nlens)` are shown. The next section displays the CPU times used for different parts of the program. The following two parts show – as examples – the five lowest and five highest indexes for lenses and cells, respectively, with masses and coordinates. The last part in the output file `dat.007` gives the number of lenses and cells in the different levels. The former indicates those lenses, that are single in a cell at this level, the latter indicates those cells, that have more than one lens in it at this level. Clearly in the first level all four cells are occupied with more than one lens, and no lens is single in a cell. And in the last level there are two lenses single, which form the cell that is indicated in the second to last level.

Joachim Wambsganss
Princeton University Observatory
Peyton Hall
Princeton, NJ 08544, USA
e-mail: jkw@astro.princeton.edu

October 31, 1991