Gravitational Lensing by Point Masses

Michael Papasimeon
29 October 1997

***Abstract*—This paper was written for an introductory undegraduate class in computational physics in 1997. The focus is on computational astrophysics; specifically on the computer simulation of gravitational lensing. The source code for the simulation written in FORTRAN77 and C can be found in the appendices.**

I. AIMS

THE general aim is to investigate the effect of gravitational lensing through the use of computer simulation. The specific aims are to:

- Investigate the effects of distance on gravitational lensing by solving the Dyer-Roeder equation for different cosmologies.
- Write a computer program which lenses an extended source image represented as a 2D grid.
- Investigate the critical and caustic curves for a number of different situations.

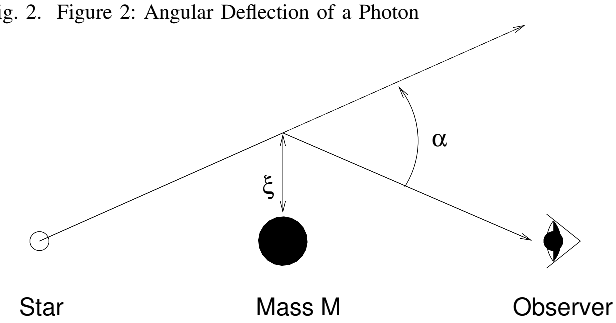
II. THEORY

According to Einstein's General Theory of Relativity, the path a light ray travels is deflected when it passes close to an object with a large enough gravitational field. The amount of angular deflection depends on the mass of the deflecting object, and the distance between the light and the object. The effect is known as gravitational lensing as the deflecting object acts very much like an optical lens, in that images of distant stellar objects can be magnified, distorted, focused and multiple images being created.

Fig. 1. Figure 1: Gravitational Lens in Abbell 2218

The effect has been observed many times using both ground and spaced based telesopes at both optical and radio wavelengths. The lensing objects are usually galaxies, but may also be other objects with large masses such as neutron stars and black holes.

Fig. 2. Figure 2: Angular Deflection of a Photon



Gravitational lensing has many applications in astronomy and astrophysics as the deflection angle of the light allows to calculate the mass of the deflecting object accurately. This technique allows the mass determination of non-lumnous astrophysical objects such as black holes. The main problem with observing gravitational lenses is that close alignment is needed between the Earth, the deflecting object and the source object to observe the effect. Therefore, many of the gravitational lenses observed involve distant galaxies and quasars as the source objects.

Figure 1 shows an example of a gravitational lens taken with the Hubble Space Telescope. The bright galaxy in the centre right of the image has graviationally lensed a galaxy which is located behind it. The large arcs around the deflecting galaxy are multiple images of a distant source galaxy.

A. The Lens Equation

Shown in figure 2, the angular deflection α of a gravitational lense is given by

$$\alpha = \frac{4GM}{c^2\xi} \tag{1}$$

where

- $M =$ mass of the deflecting object
- $G =$ gravitational constant
- $c =$ speed of light
- $\xi =$ impact radius of the incoming photon
- $\alpha =$ deflection angle

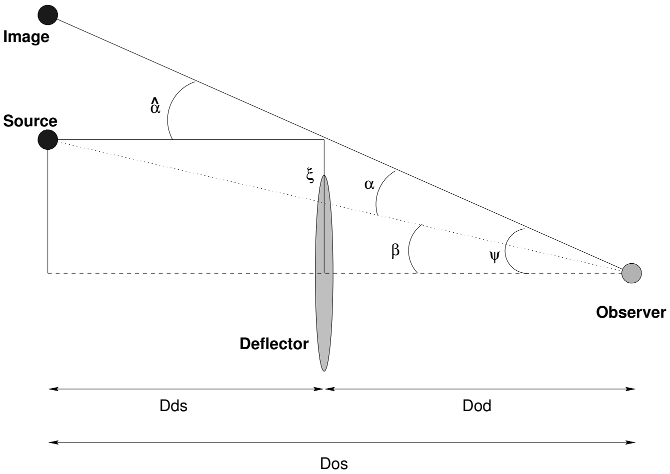
The deflecting mass may be viewed as an optical thin lens, made up of a two dimensional mass distribution.

$$M = \int_{R^2} \Sigma(\vec{\xi'}) \frac{\vec{\xi} - \vec{\xi'}}{|\vec{\xi} - \vec{\xi'}|^2} d^2\xi' \tag{2}$$

where

- $\Sigma =$ surface density
- $R^2 =$ surface area

Fig. 3. Figure 3: Setup for the lens equation



If the deflecting mass, is perfectly symmetric, the deflection angle becomes

$$\vec{\alpha} = \frac{4GM(<\xi)\vec{\xi}}{c^2|\vec{\xi}|^2} \tag{3}$$

Figure 3 below shows the setup for the lens equation. The lens equation is

$$\psi D_{os} + \alpha D_{ds} = \beta D_{os} \tag{4}$$

It is often the case in many gravitational lensing problems that the images form do not depend on the distances between source, observer and deflector directly. Rather a specific ratio of these distances is the quantity which needs to be considered. This is known as the effective distance D and is defined as:

$$D = \frac{D_{od} D_{ds}}{D_{os}} \tag{5}$$

B. Point Mass Lenses

Certain gravitational lenses may be approximated as point masses. These include large mass objects such as black holes and neutron stars. If a distant star is in perfect alignment with a point mass gravitational lens and an observer, the light from the distant star is lensed perfectly symmetrically forming a ring image of the star known as an Einstein ring. The radius of the Einstein ring is given by

$$\theta_E = \sqrt{\frac{4GM}{c^2}\frac{D_{ds}}{D_{od}D_{os}}} \tag{6}$$

If however there isn't a perfect alignment, for a point mass two images are produced for each point on the source plane. The angular positions at which these images is given by:

$$\theta_{\pm} = \frac{\xi}{D_{od}} = \frac{\beta}{2} \pm \sqrt{\beta^2 + 4\theta_E^2} \tag{7}$$

where β is the angular position of the source as shown in figure 3. The magnification of each of the images is given by

$$\mu_{\pm} = \frac{1}{4}\left[\frac{y}{\sqrt{y^2+4}} + \frac{\sqrt{y^2+4}}{y} \pm 2\right] \tag{8}$$

where the source and image angles have been scaled such that $y = \psi/\alpha_0$ and $x = \alpha/\alpha_0$.

Using this notation the lens equation can be rewritten as the scaled lens equation given by

$$\vec{y} = \vec{x} - \vec{\alpha}(\vec{x}). \tag{9}$$

The vector notation used represents the coordinate in a cartesian plane. Therefore

- $\vec{y} = (y_1, y_2)$ is a coordinate in the source plane and
- $\vec{x} = (x_1, x_2)$ is a coordinate in the deflecting plane.

This can also be written using complex number notation. A position in deflecting plane can be denoted as $z = x_1 + ix_2$, and a position in the source plane as $z_s = y_1 + iy_2$.

The magnification is given by:

$$\mu(\vec{x}) = \frac{1}{detA(\vec{x})} \tag{10}$$

where $detA(\vec{x})$ is the Jacobian determinant of the Hessian matrix given by

$$A(\vec{x}) = \frac{\partial \vec{y}}{\partial \vec{x}} \tag{11}$$

C. Critical Curves and Caustics

Critical curves are the set of all points in the deflection plane where $A(\vec{x}) = 0$. The corresponding curves in the source plane (obtained from the lens equation) are known as caustics. Critical curves are where the gravitational lens infinitely magnifies the light passing through that point. Source plane caustics are the pre-image of the critical curves. Any light emitted near a caustic will be greatly magnified as it passes through the lens.

D. The Chang-Refsdal Lens

The Chang-Refsdal lens model describes gravitational lensing using a modification of the point mass model. This model says that when a source crosses a fold caustic the lensing is due to a point mass but with an additional external shear applied. The corresponding lens equation for the Chang-Refsdal model is:

$$\vec{y} = \begin{bmatrix} 1+\gamma & 0 \\ 0 & 1-\gamma \end{bmatrix} \vec{x} - \frac{\vec{x}}{|\vec{x}|^2} \tag{12}$$

This can also be written in the complex notation as $z_s = z + \gamma\bar{z} - \frac{\epsilon}{\bar{z}}$, where

- γ is a constant determining the amount shear.
- $\epsilon = (\frac{\kappa_s}{1-\kappa_c})$, where κ_s is the density of compact objects such as stars and κ_c is the surface density of continuously distributed matter.

III. The Effects of Distance on Gravitational Lensing

A. Method

The Dyer-Roeder equation is given by:

$$(z+1)(\Omega z+1)\frac{d^2D}{dz^2} + \left(\frac{7}{2}\Omega z + \frac{1}{2}\Omega + 3\right)\frac{dD}{dz} + \frac{3}{2}\tilde{\alpha}\Omega D = 0 \tag{13}$$

This relates the angular diameter distance of a lensing system with the redshift z of the source object.

- Ω is the ratio of the mean mass density to the critical density of the universe and
- $\tilde{\alpha}$ is the clumpiness parameter, which determines the amount of matter between the source and the observer.

The initial conditions of the Dyer-Roeder equation are:

$$D_{ii} = 0 \tag{14}$$

$$\left[\frac{dD_{ij}}{dz}\right]_{z_j=z_i} = \frac{\text{sgn}(z_j - z_i)}{(z_i+1)^2\sqrt{\Omega z_i + 1}} \tag{15}$$

The Dyer-Roeder equation needs to be solved for three different cases.

1) $\Omega = 0$ (D_I)
2) $\Omega = 1$, $\tilde{\alpha} = 1$ (D_{II})
3) $\Omega = 1$, $\tilde{\alpha} = 0$ (D_{III})

This leads to three different equations which need to be solved.

$$(z+1)\frac{d^2D}{dz^2} + 3\frac{dD}{dz} = 0 \tag{16}$$

$$(z+1)^2\frac{d^2D}{dz^2} + \frac{7}{2}(z+1)\frac{dD}{dz} + \frac{3}{2}D = 0 \tag{17}$$

$$(z+1)^2\frac{d^2D}{dz^2} + \frac{7}{2}(z+1)\frac{dD}{dz} = 0 \tag{18}$$

A FORTRAN program (see Appendix B – `lense.f`) was written to numerically solve these equations. The algorithm used to solve these equation numerically was the *Runge-Kutta-Nyström* [1] method. Is method is a fourth order algorithm which is a general form of the standard *Runge-Kutta* method, used for solving second order ordinary differential equations.

The equations were integrated from $z = 0$ to $z = 10$. Also, the dimming factor $(D_{II}/D_{III})^2$ was determined and plotted as a function of redshift z.

Fig. 4. Plot of angular diameter distance against source redshift

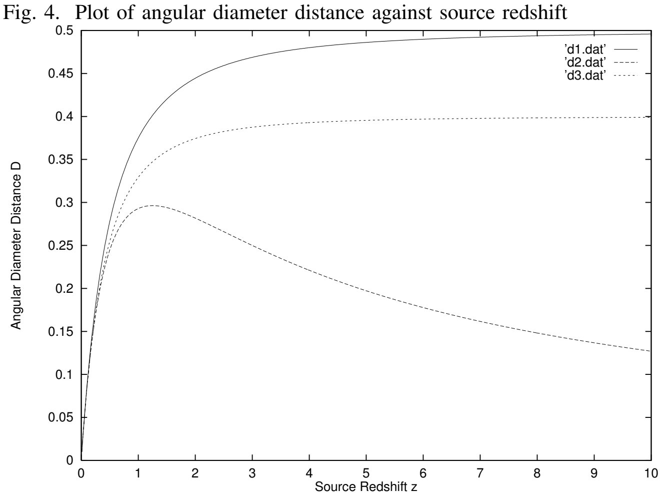
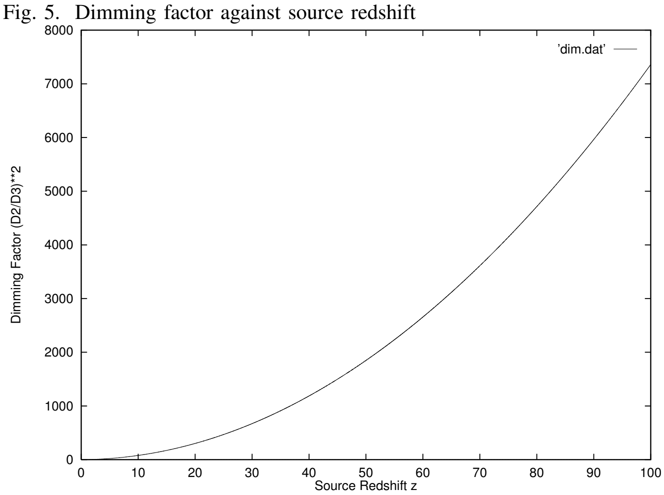


Fig. 5. Dimming factor against source redshift



B. Results and Discussion

The plot below shows the numerical result of solving the Dyer-Roeder equation for the three different cosmologies resulting in three solutions for the angular diameter distance D.

- D_I `d1.dat` (The top curve)
- D_{II} `d2.dat` (The bottom curve)
- D_{III} `d3.dat` (The centre curve)

According to the big bang model of the universe, objects with large redshifts are further away. Hence the results of solving the Dyer-Roeder equation for the cases (1 and 3) when the "clumpiness" parameter $\tilde{\alpha}$ is ignored the angular diameter distance D increases as the redshift increases. In both cases we get asymptoting values:

$$\lim_{z\to\infty} D_I(z) = 0.5 \tag{19}$$

$$\lim_{z\to\infty} D_{II}(z) = 0.4 \tag{20}$$

In both these cases where $\tilde{\alpha}$ is not in the equation, the lens is the only matter along the line of sight to the source and hence we get an increase in the angular diameter distance as the redshift increases. Of particular interest is that in case I ($\Omega = 0$), the value of D increases more rapidly than in case III ($\Omega = 1$). Hence we see that the density of the universe plays influences the results of any models chosen for gravitational lensing. Since the mean and critical densities of the universe are unknown (especially since it is believed that most of the matter of the universe is dark matter), there are limits to how accurately we can determined parameters from gravitational lensing observations.

The second case is perhaps the most interesting of the three cosmologies in that we have a non-zero contribution from the "clumpiness" parameter $\tilde{\alpha}$. As a result is also produces the most interesting result as we have a maxiumum in D at $z \simeq 1$. In this case where $\tilde{\alpha} = 1$, the gravitational lense make only a minor contribution to the total amount of smoothly distributed matter in the line of sight of the observer and the source. In this situation the matter is smootly distributed and we get gravitational lens being the matter between source and observer. We can then think of this as the light passing through a material of a different refractive index as in electrodynamics.

[1] See the code in Appendix B for the exact algorithm. This algorithm was obtained from Chapter 20 (Numerical Methods for Differential Equations) from *Advanced Engineering Mathematics* by Erwin Kreyszig.

The plot above shows the dimming factor $(D_{II}/D_{III})^2$ plotted against source redshift z. In case II has a smoothly distributed matter distribution meaning that we see more and brigter images than in case III where the "clumpiness" parameter results in less light reaching the observer. As a result, a case III universe appears to be dimmer than a case II universe. The dimming ratio of factor is $(D_{II}/D_{III})^2$ and is plotted against red shift in the plot above. As we can see in the plot the higher the redshift the more dimness in a case III universe.

IV. Lensing an Extended Source by a Point Mass

A. Method

A program was written to simulate the gravitational lensing of a two dimensional grid. Two make the results of the simulation more interesting the two dimensional grid chosen was represented as a two dimensional array of integers ranging from 0 to 255. The value at each array element represents an intensity value. As a result the grid represents a two dimensional Portable Grey Map (PGM) image. The program was written to accept the name of a PGM file on the command line, load the image into memory,, apply the gravitational lensing calculations and output a new "lensed" PGM image to standard output. The program also takes the mass M (in kg) of the lensing body, the effective distance D specified in equation 5, and the (x, y) location of the lensing body. The details of the lensing algorithm can be found in Algorith 1.

Algorithm 1 Lensing Algorithm

```
procedure LENSE
    img ← load_image()
    x_c, y_c ← find_centre(img)
    θ_E ← calc_einstein_radius()
    for c ∈ img.coordinates do
        b_x, b_y ← calc_location_lensing_body()
        β ← impact_radius()
        calc_angle_for_each_quadrant()
        θ_± ← calc_new_angles()
        c ← calc_new_deflected_coordinate()
    end for
    save_image(img)
end procedure
```

A number of different images were used in the simulation. Two were used for testing purposes and to observe the effect, and three were images of astronomical interest. The images used include:

- A human face
- The starship Enterprise
- The Andromeda Galaxy
- The Milky Way
- The Pleiades Star Cluster

The program used for the simulation called `image.c` can be found in Appendix C. The programming language used in this case was C instead of FORTRAN, for the following reasons:

- Using a C `struct` was appropriate in representing the data stored in a single PGM image.

Fig. 6. Human Face

- Image files of varying widths and heights were used, and therefore the program made used of dynamic memory allocation to allocate only the memory that was required to store the images.

In all cases the distance ratio used was $D = 1$ and just the mass was varied. This is because when calculating the Einstein radius, the quantity DM appears as follows:

$$\theta_E = \sqrt{\frac{4GMD}{c^2}} \tag{21}$$

B. Results and Discussion

1) A Human Face: An image of a face was used to first test the program. The image on the left is the original, and the image on the right is one that has been gravitationally lensed by a point mass in the lower right hand corner of mass, $M = 5 \times 10^{30}$ kg.

Apart from the obvious distortion in the lensed image, of particular interest is the formation of two images, one small and one large. The smaller image is inside the Einstein radius of the lense.

2) The Enterprise: The three images below are of the starship Enterprise from Star Trek.

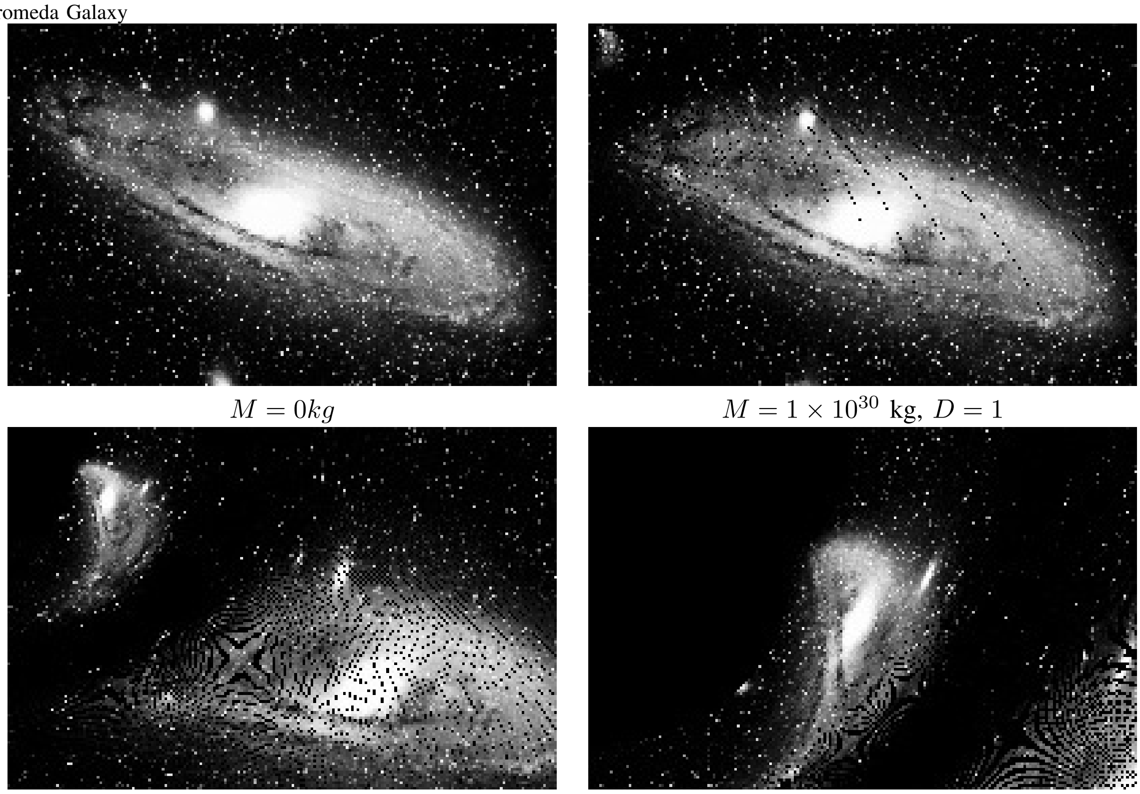
- The image on the left is the original
- The center image has a gravitational lense of mass $M = 2 \times 10^{30}$ kg with the lense placed in the centre.
- The image on the right has a gravitational lense of mass $M = 1 \times 10^{31}$ in the lower right hand corner.

With the lense in the center of the image it is more difficult to see the two images. The image on the right resulting from a larger mass (and with the lens in the bottom right hand corner) clearly shows both the images and large distortions. In many cases of actual observations, the images inside the Einstein radius are two small to be resolved by even the most powerful telescopes.

3) The Andromeda Galaxy: This image is that of the Andromeda galaxy. A number different lenses of varying masses were used. In all the cases the lensing object is located in the upper left hand corner of the image [position $(0, 0)$].

As the mass of the lense is increased, the amount of distortion in the resulting image is increased. The secondary image is also clearly visible.

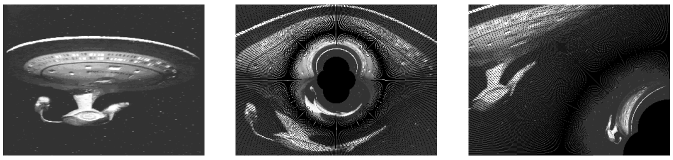
Fig. 8. Andromeda Galaxy



$M = 0kg$ $M = 1 \times 10^{30}$ kg, $D = 1$

$M = 1 \times 10^{31}$ kg, $D = 1$ $M = 4 \times 10^{31}$ kg, $D = 1$

Fig. 7. Enterprise



4) The Milky Way: The image on left is looking towards the centre of the Milky Way in the infrared. The image on the right is gravitationally lensed by a $M = 3 \times 10^{30}$ kg mass. The lense is located slightly left and above from the center. This image of the Milky Way allows us to see the distortion of the image within the Einstein radius much easier.

5) The Pleiades Star Cluster: The image on the left is that of the Pleiades star cluster. The image on the right is gravitationally lensed by a $M = 2 \times 10^{30}$ kg mass.

V. Caustics for the Chang-Refsdal Lens

A. Method

A FORTRAN program was written (see Appendix A – `caustics.f`) which calculated critical curves and caustics for given values of γ and ϵ. As stated earlier, critical curves are obtained when Jacobian determinant is zero ($\det A = 0$) This gives:

$$\det A = 1 - \left(\gamma + \frac{\epsilon}{\bar{z}^2}\right)\left(\gamma + \frac{\epsilon}{\bar{z}^2}\right) = 0 \tag{22}$$

. Using complex polar coordinates letting $z = x\cos\phi + ix\sin\phi$ we get

$$x^4(1-\gamma^2) - 2\gamma\epsilon x^2(\cos^2\phi - \sin^2\phi) - 1 = 0 \tag{23}$$

The equation can be parameterised by letting:

$$\lambda = \cos^2\phi - \sin^2\phi$$

$$u = x^2$$

giving

$$u^2(1-\gamma^2) - 2\gamma\epsilon\lambda u - 1 = 0. \tag{24}$$

Solving this quadratic we obtain:

$$u = \frac{\gamma\epsilon\lambda \pm \sqrt{\gamma^2(\lambda^2-1)+1}}{(1-\gamma^2)} \tag{25}$$

For a given values of γ, ϵ and for $0 < \phi < 2\pi$ the program calculates u and then x. The (x, y) coordinates for the caustic curve is then given by $(x\cos\phi, x\sin\phi)$. The corresponding caustics are given by:

$$y_1 = \left[(1+\gamma)x - \frac{\epsilon}{x}\right]\sqrt{\frac{1+\lambda}{2}} \tag{26}$$

$$y_2 = \left[(1-\gamma)x - \frac{\epsilon}{x}\right]\sqrt{\frac{1-\lambda}{2}} \tag{27}$$

In the program a value of $\epsilon = 0.5$ was chosen. Values of gamma were chosen for the four representative regions in which caustics are found. The four regions are:

- $\gamma < -1$

Fig. 9. Milky Way

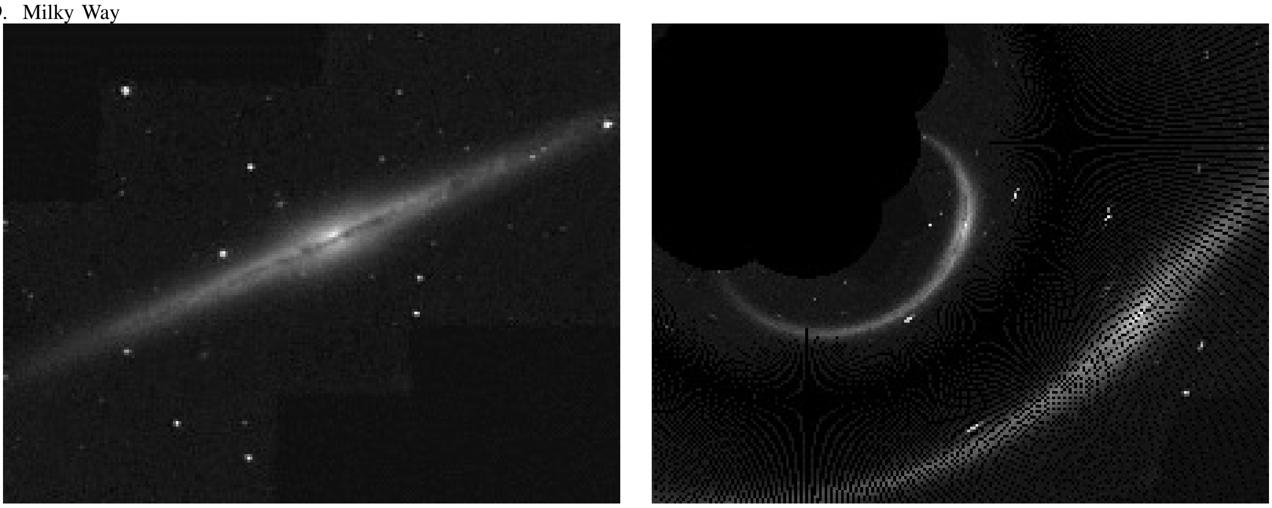
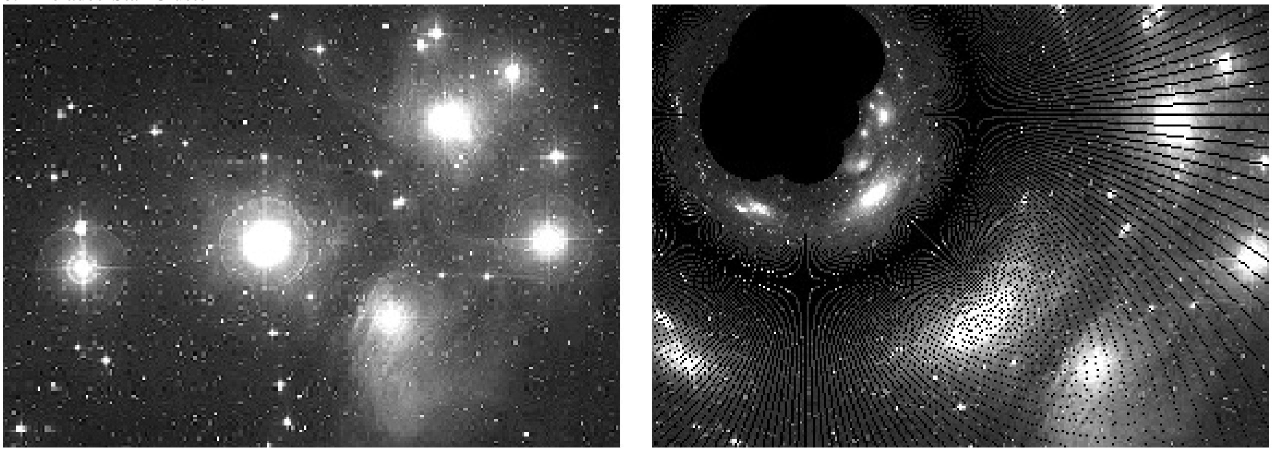


Fig. 10. Pleiades Star Cluster



- $-1 < \gamma < 0$
- $0 < \gamma < 1$
- $\gamma > 1$

The four values of γ selected are:

- $\gamma = -1.3$
- $\gamma = -0, 4$
- $\gamma = 0.8$
- $\gamma = 1.6$

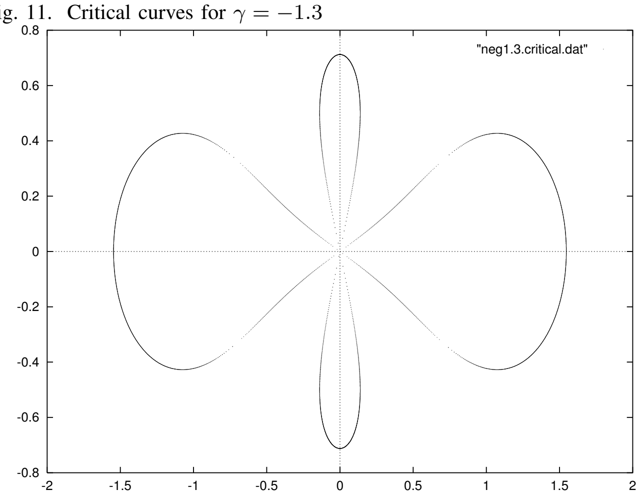
B. Results

1) $\gamma = -\mathbf{1.3}$: For $\gamma = -1.3$, the critical curves are shown in Figure 11 and the caustics shwon in in Figure 12.

2) $\gamma = -\mathbf{0.4}$:

3) $\gamma = \mathbf{0.8}$:

Fig. 11. Critical curves for $\gamma = -1.3$



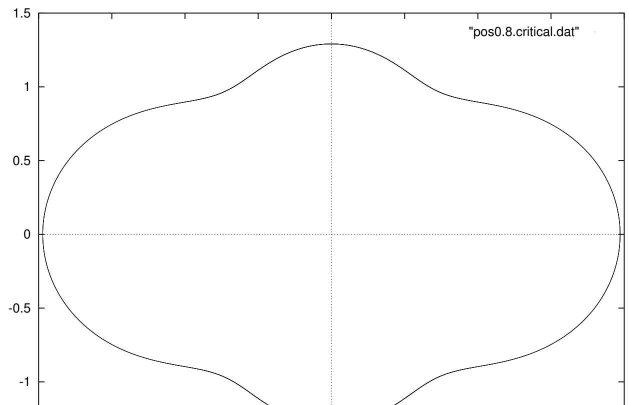


Fig. 12. Caustics for $\gamma = -1.3$

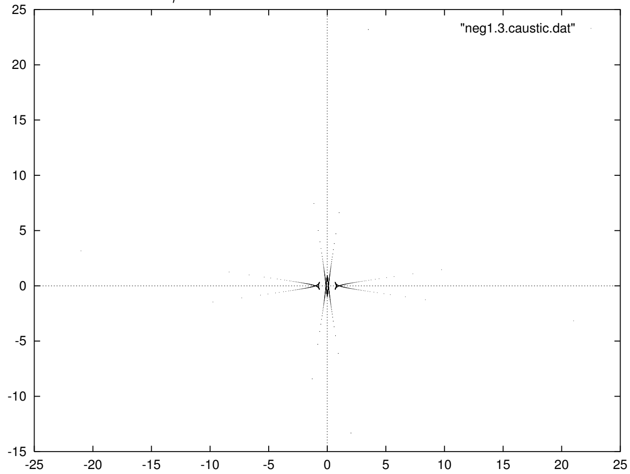


Fig. 13. Critical curves for $\gamma = -0.4$

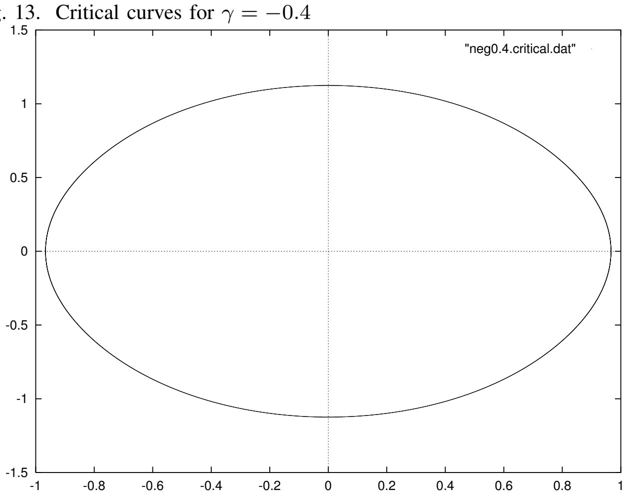
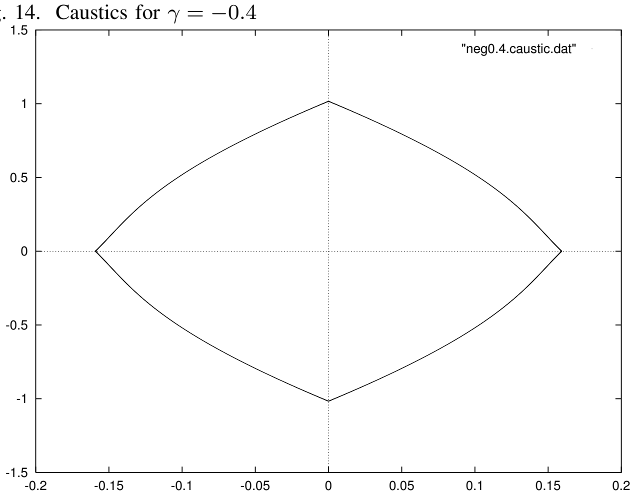
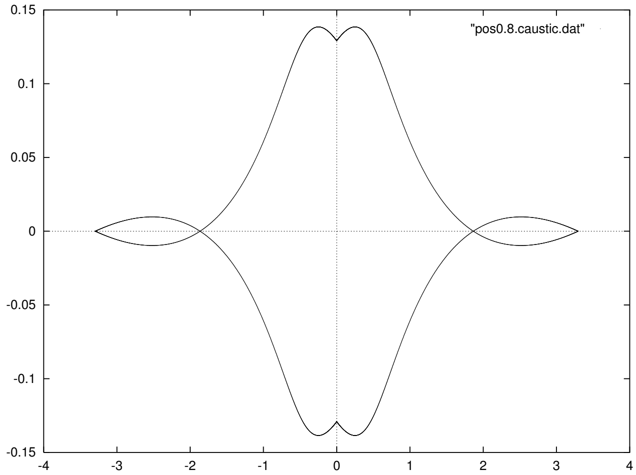


Fig. 14. Caustics for $\gamma = -0.4$

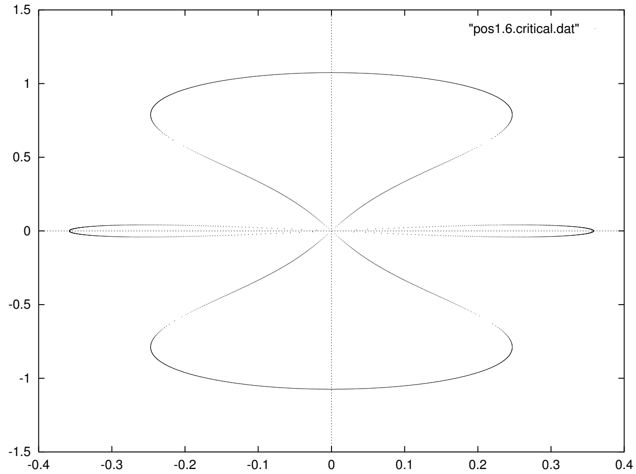


Critical curves for $\gamma = 0.8$

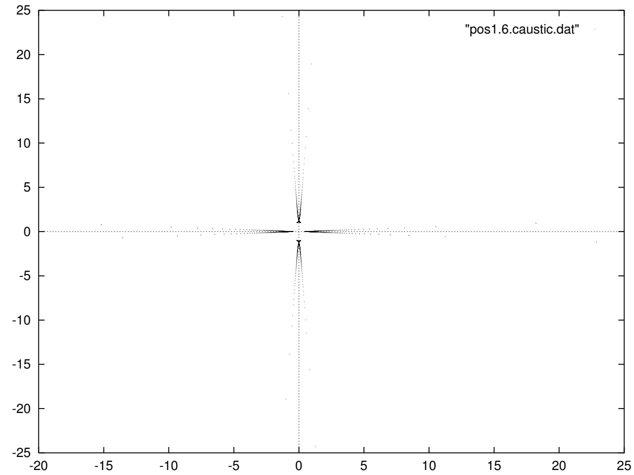


Caustic for $\gamma = 0.8$

4) $\gamma = \mathbf{1.6}$*:*



Critical curves for $\gamma = 1.6$

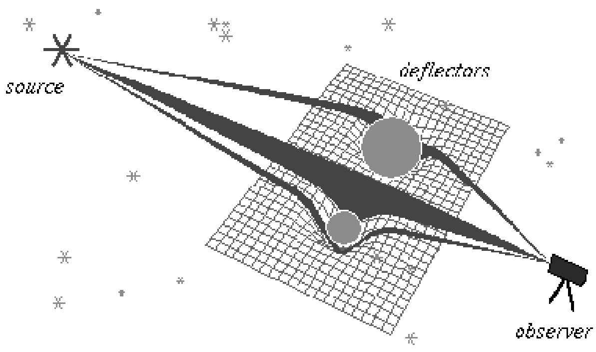


Caustics for $\gamma = 1.6$

C. Discussion

As can be seen from the diagrams in the previouse section, the caustic curves with values of γ ranging between -1 and 1

Fig. 15. Gravitational Lensing by a Binary Star System



are non overlapping curves, whereas the others overlap in a "petal" pattern.

The caustics can be used to determine light curves for background sources since they mark where intensity increases in the resulting gravitationally lensed image occur.

The use of critical lines and caustics are even more important when modelling graviational lenses especially when both the source and the lens are more complicated objects such as galaxies. For example, the diagram above shows the representation of gravitational lensing for a distant star by a binary star system. As can be seen from the diagram, a simple point mass model is not sufficient to handle a system such as this, and hence more detailed models such as the Chang-Refsdal lens model are needed to simulate this situation so that it we can accurate comparisons with observations.

APPENDIX
CODE LISTING: CAUSTICS.F

```
c----------------------------------------------------------

      program main
            implicit none
            real*8 phi
            real*8 gamma, lambda, epsilon
            real*8 a, b, c
            real*8 u1, u2, x1, x2, x3, x4
            real*8 y1, y2
            real*8 causticY1, causticY2
            real*8 twopi

            open(unit=1, file='critical_line.dat', status='unknown')
            open(unit=2, file='caustic_curve.dat', status='unknown')

            twopi = 2*4*atan(1.0d0)
            epsilon = 0.5d0

            write(*,*)'Enter Gamma: '
            read(*,*)gamma

            do while(phi .le. twopi)
                  lambda = cos(phi)**2.0d0 - sin(phi)**2.0d0
                  a = gamma*epsilon*lambda
                  b = sqrt((gamma**2.0d0)*(lambda**2.0d0-1.0d0)+1.0d0)
                  c = 1 - gamma**2.0d0

                  u1 = (a + b)/c
                  u2 = (a - b)/c

                  if (u1 .gt. 0.0d0) then
                        x1 = sqrt(u1)
                        x2 = -sqrt(u1)
                        write(1,*) x1*cos(phi), x1*sin(phi)
                        write(1,*) x2*cos(phi), x2*sin(phi)

                        y1 = causticY1(gamma, epsilon, x1, lambda)
                        y2 = causticY2(gamma, epsilon, x1, lambda)
                        write(2,*) y1*cos(phi), y2*sin(phi)

                        y1 = causticY1(gamma, epsilon, x2, lambda)
                        y2 = causticY2(gamma, epsilon, x2, lambda)
                        write(2,*) y1*cos(phi), y2*sin(phi)
                  end if

                  if (u2 .gt. 0.0d0) then
                        x3 = sqrt(u2)
                        x4 = sqrt(u2)
                        write(1,*) x3*cos(phi), x3*sin(phi)
                        write(1,*) x4*cos(phi), x4*sin(phi)

                        y1 = causticY1(gamma, epsilon, x3, lambda)
                        y2 = causticY2(gamma, epsilon, x3, lambda)
                        write(2,*) y1*cos(phi), y2*sin(phi)

                        y1 = causticY1(gamma, epsilon, x4, lambda)
                        y2 = causticY2(gamma, epsilon, x4, lambda)
                        write(2,*) y1*cos(phi), y2*sin(phi)
                  endif

                  phi = phi + 0.0001
            end do

            close(unit=1)
```

```
        close(unit=2)
      end

c--------------------------------------------------------

      double precision function causticY1(gamma, epsilon, x, lambda)
        implicit none
        real*8 gamma, epsilon, x, lambda
        real*8 cosphi

        cosphi = sqrt((1 + lambda)/2.0d0)
        causticY1 = ((1 + gamma)*x - epsilon/x)*cosphi
        return
      end

      double precision function causticY2(gamma, epsilon, x, lambda)
        implicit none
        real*8 gamma, epsilon, x, lambda
        real*8 sinphi

        sinphi = sqrt((1 - lambda)/2.0d0)
        causticY2 = ((1 - gamma)*x - epsilon/x)*sinphi
        return
      end
```

Appendix
Code Listing: lense.f

```
      program main
        implicit none

        call runge_kutta_nystrom(0.01d0, 10000, 0.0d0)
      end

      subroutine runge_kutta_nystrom(h, N, x0)
        implicit none
        real*8 h, x0
        real*8 k1, k2, k3, k4, K, L
        real*8 x, y, yd, f
        real*8 i, hon2
        integer N

        hon2 = h/2.0d0
        x = x0
        y = 0.0d0
        yd = 1.0d0

        do i = 0, N-1, +1
          k1 = hon2*f(x, y, yd)

          K = hon2*(yd + k1/2.0d0)
          k2 = hon2*f(x + hon2, y + K, yd + k1)

          k3 = hon2*f(x + hon2, y + K, yd + k2)

          L = h*(yd + k3)
          k4 = hon2*f(x + h, y + L, yd + 2.0d0*k3)

          x = x + h
          y = y + h*(yd + (k1 + k2 + k3)/3.0d0 )

          yd = yd + (k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/3.0d0

          write(*,*)x, y
        end do

      end
```

```
      double precision function f(x, y, yd)
            implicit none
            real*8 f2, f3
            real*8 x, y, yd

            f = (f2(x,y,yd)/f3(x,y,yd))**2.0d0
            return
      end

      double precision function f1(x, y, yd)
            implicit none
            real*8 x, y, yd

            f1 = (-3.0d0/(x + 1))*yd
            return
      end

       double precision function f2(x, y, yd)
            implicit none
            real*8 x, y, yd

            f2 = ((-7.0d0/2.0d0)*(x+1)*yd-3.0d0*y/2.0d0)/((x+1)**2.0d0)
            return
      end

       double precision function f3(x, y, yd)
            implicit none
            real*8 x, y, yd

            f3 = ((-7.0d0/2.0d0)*(x+1)*yd)/((x+1)**2.0d0)
            return
      end
```

APPENDIX
CODE LISTING: IMAGE.C

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

#define MIN_GREY_VAL 0
#define MAX_GREY_VAL 255

#define G 6.67E-11 /* Gravitational Constant */
#define C2 9.0E16 /* Speed of light squared */

typedef struct
{
    int width;
    int height;
    int maxGreyVal;
    int **pixels;
} PGM;

PGM* readPGM(char* filename);
void outputPGM(PGM *pgm);
PGM* gravitationalLense(PGM* pgm, float D, float M);
void* memCalloc(size_t n, size_t size);
void* memAlloc(size_t size);

int main(int argc, char **argv)
{
    PGM* image = (PGM*)memAlloc(sizeof(PGM));
    float D = 0.0, M = 0.0;

    if (argc == 4)
```

```
    {
        image = readPGM(argv[1]);
        D = atof(argv[2]);
        M = atof(argv[3]);
        image = gravitationalLense(image, D, M);
        outputPGM(image);
    }
    else
    {
        fprintf(stderr,"Usage: image_file.pgm D M\n");
    }
    return(0);
}

PGM* readPGM(char* filename)
{
    PGM* image = (PGM*)memAlloc(sizeof(PGM));
    char header[10];
    int i = 0, j = 0;
    FILE* fp;

    if ( (fp = fopen(filename, "r")) != (FILE*)NULL)
    {
        fscanf(fp, "%s", header);
        fscanf(fp, "%d %d", &(image->width), &(image->height));
        fscanf(fp, "%d", &(image->maxGreyVal));

        image->pixels = (int**)memCalloc(image->height, sizeof(int*));

        for (i = 0; i < image->height; i++)
        {
            image->pixels[i] = (int*)memCalloc(image->width, sizeof(int));

            for (j = 0; j < image->width; j++)
                fscanf(fp, "%d", &(image->pixels[i][j]));
        }

        fclose(fp);
    }
    else
    {
        fprintf(stderr,"Unable to open file\n");
        exit(1);
    }

    return(image);
}

PGM* gravitationalLense(PGM* pgm, float D, float M)
{
    PGM* image = (PGM*)memAlloc(sizeof(PGM));
    int i = 0, j = 0;
    float xc = 0.0, yc = 0.0;
    float xnew = 0.0, ynew = 0.0;
    float xp = 0.0, yp = 0.0;
    float xm = 0.0, ym = 0.0;
    int xi = 0, yi = 0;
    float beta = 0.0, psi = 0.0, phi = 0.0, angle = 0.0;
    float thetaE = 0.0, thetaPlus = 0.0, thetaMinus = 0.0;

    image->width = pgm->width;
    image->height = pgm->height;
    image->maxGreyVal = pgm->maxGreyVal;

    image->pixels = (int**)memCalloc(image->height, sizeof(int*));

    for (i = 0; i < image->height; i++)
    {
```

```
        image->pixels[i] = (int*)memCalloc(image->width, sizeof(int));

        for (j = 0; j < image->width; j++)
            image->pixels[i][j] = MIN_GREY_VAL;
    }
        /* Calculate center of image */

    xc = (image->width)/2.0;
    yc = (image->height)/2.0;

        /* Calculate Einstein Ring Radius */

    thetaE = sqrt(4*G*M*D/C2);

        /* Calculate image position for each
           pixel in the source */

    for (i = 0; i < image->height; i++)
    {
        for (j = 0; j < image->width; j++)
        {
            xnew = j - xc ; /*j - xc;*/
            ynew = -i + yc; /*-i + yc;*/

            beta = sqrt(xnew*xnew + ynew*ynew);

            if (xnew)
            {
                angle = atan(ynew/xnew);
            }
            else
            {
                if (ynew > 0.0)
                    angle = M_PI/2.0;
                else
                    angle = -M_PI/2.0;
            }
            angle = atan(ynew/xnew);

                /* Get correct angle for all quadrants */

            if (xnew > 0.0)
            {
                phi = angle;
                psi = angle + M_PI;
            }
            else if (xnew < 0.0)
            {
                phi = angle + M_PI;
                psi = angle;
            }

            if (xnew == 0.0)
            {
                if (ynew > 0.0)
                {
                    phi = M_PI/2.0;
                    psi = 3*M_PI/2.0;
                }
                else
                {
                    phi = 3*M_PI/2.0;
                    psi = M_PI/2.0;
                }
            }

            if (ynew == 0.0)
            {
```

```c
                if (xnew > 0.0)
                {
                    phi = 0.0;
                    psi = M_PI;
                }
                else
                {
                    phi = M_PI;
                    psi = 0.0;
                }
            }

                /* Calculate thetaPlus and thetaMinus */

            thetaPlus = 0.5*(beta + sqrt(beta*beta + 4*thetaE*thetaE));
            xp = thetaPlus*cos(phi);
            yp = thetaPlus*sin(phi);

            xi = (int)(xp + xc);
            yi = -(int)(yp - yc);

            if ((xi >= 0 && xi < image->width) && (yi >= 0 && yi < image->height))
            {
                image->pixels[yi][xi] = pgm->pixels[i][j];
            }

            thetaMinus = 0.5*(beta - sqrt(beta*beta + 4*thetaE*thetaE));
            xm = thetaMinus*cos(psi);
            ym = thetaMinus*sin(psi);

            xi = (int)(xm + xc);
            yi = -(int)(ym - yc);

            if ((xi >= 0 && xi < image->width) && (yi >= 0 && yi < image->height))
            {
                image->pixels[yi][xi] = pgm->pixels[i][j];
            }

        }
    }

    return(image);
}

void outputPGM(PGM *pgm)
{
    int i = 0, j = 0;
    int pixelCount = 0;

    printf("P2\n");
    printf("%d␣%d\n", pgm->width, pgm->height);
    printf("%d␣\n", pgm->maxGreyVal);

    for (i = 0; i < pgm->height; i++)
    {
        for (j = 0; j < pgm->width; j++)
        {
            printf("%d␣", pgm->pixels[i][j]);
            if (pixelCount++ >= 10)
            {
                printf("\n");
                pixelCount = 0;
            }
        }

        if (pixelCount != 0)
        {
            printf("\n");
```

```
            pixelCount = 0;
        }
    }
}

void* memCalloc(size_t n, size_t size)
{
    void* tmp;

    tmp = calloc(n, size);

    if (tmp == (void*)NULL)
    {
        fprintf(stderr,"Unable to allocate memory...Exiting\n");
        exit(1);
    }

    return(tmp);
}

void* memAlloc(size_t size)
{
    return(memCalloc(1, size));
}
```