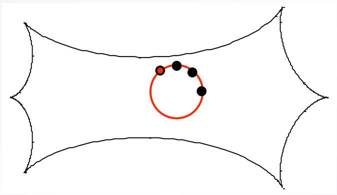
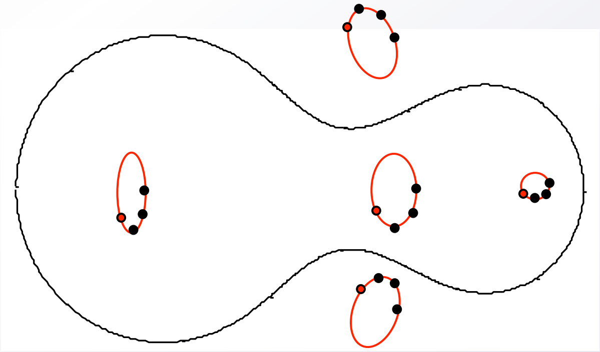
Light curve computation in binary microlensing



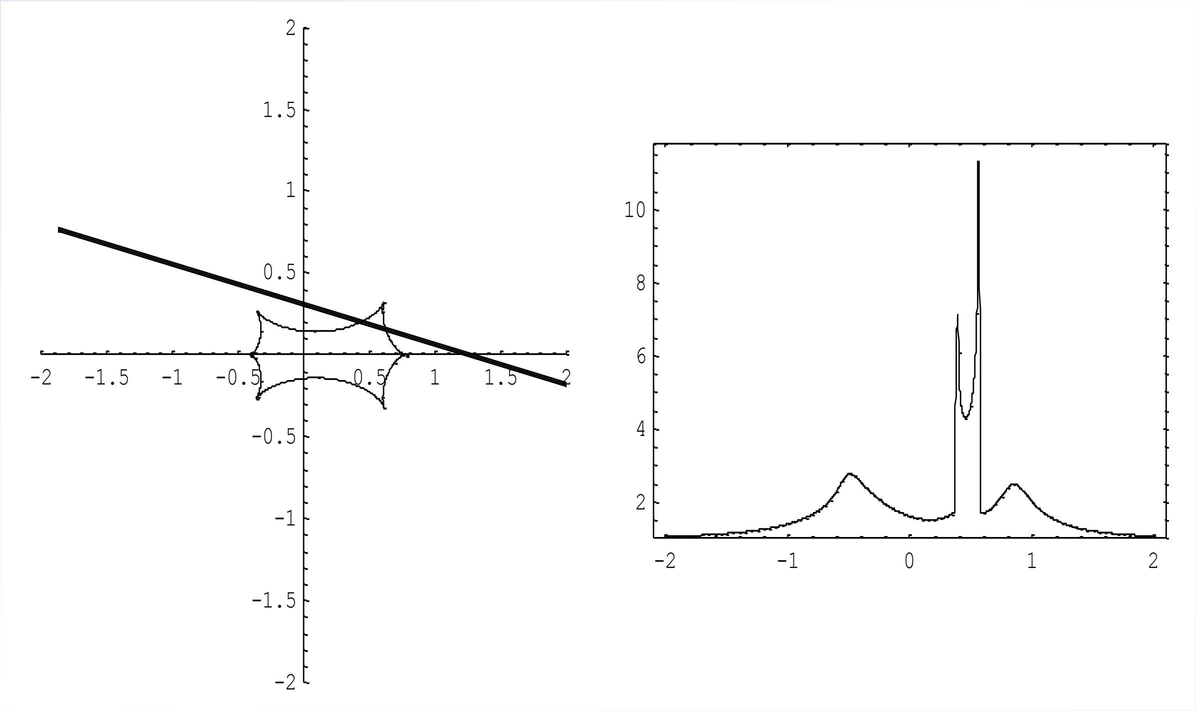


Valerio Bozza

University of Salerno, Italy

Our goal

- Our purpose is to calculate microlensing light curves

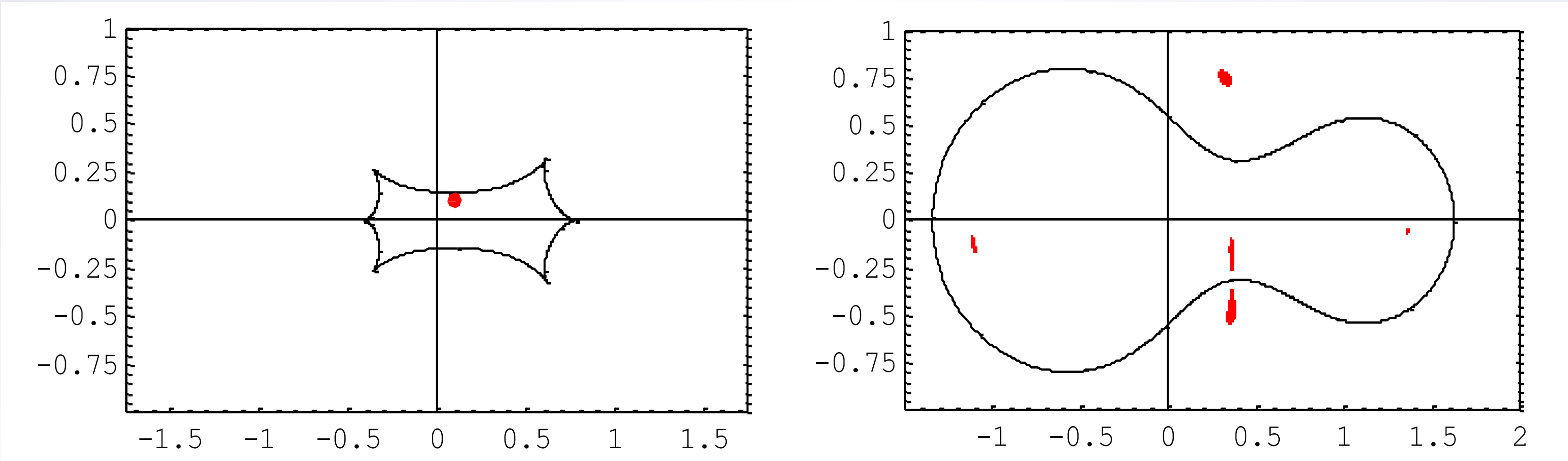


- … that is the magnification of a given source that passes behind a given binary lens.

Our goal

- For a given source position and size and for a given lens model, we need to calculate the magnification factor

$$\mu = \frac{\sum_I A_I}{A_S}$$



- We must find the images and calculate their area.

Summary

- In order to reach this goal, we will take several steps:

  - Point-source magnification (Solving the lens equation)
  - Finite-source approximations (Quadrupole and Hexadecapole)

- Full calculations
  - Inverse ray-shooting
  - Contour integration
    - VBBinaryLensing

Lens equation

- Let us use complex notations (Witt 1990), in a frame centered on the lower mass object:

$$\zeta = z - \frac{m_1}{\bar{z} - s} - \frac{m_2}{\bar{z}}$$

$$m_1 = \frac{1}{1+q}$$

$$m_2 = \frac{q}{1+q}$$

- $\bar{z}$ can be eliminated using the conjugate equation.
- We end up with a fifth order polynomial equation

$$p(z) = \sum_{i=0}^{5} c_i z^i = 0$$

$$\begin{aligned}
c_0 &= s^2 \zeta m_2^2 \\
c_1 &= -s\, m_2 \left(2\zeta + s\left(-1 + s\zeta - 2\zeta\bar{\zeta} + m_2\right)\right) \\
c_2 &= \zeta - s^3 \zeta\bar{\zeta} + s\left(-1 + m_2 - 2\zeta\bar{\zeta}\,(1 + m_2)\right) + s^2\left(\bar{\zeta} - 2\bar{\zeta} m_2 + \zeta\left(1 + \bar{\zeta}^2 + m_2\right)\right) \\
c_3 &= s^3 \bar{\zeta} + 2\zeta\bar{\zeta} + s^2\left(-1 + 2\zeta\bar{\zeta} - \bar{\zeta}^2 + m_2\right) - s\left(\zeta + 2\zeta\bar{\zeta}^2 - 2\bar{\zeta} m_2\right) \\
c_4 &= \bar{\zeta}\left(-1 + 2s\bar{\zeta} + \zeta\bar{\zeta}\right) - s\left(-1 + 2s\bar{\zeta} + \zeta\bar{\zeta} + m_2\right) \\
c_5 &= \left(s - \bar{\zeta}\right)\bar{\zeta}
\end{aligned}$$

Finding the roots

- Starting from an arbitrary initial condition z_0, we can find a root of a nth degree polynomial using Laguerre's method:

$$z_{k+1} = z_k - \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}} \qquad G = \frac{p'(z_k)}{p(z_k)}; \qquad H = G^2 - \frac{p''(z_k)}{p(z_k)}$$

- Once we have the first root z_1, we can divide the original polynomial by $(z-z_1)$ and find the next root.
- After all roots have been found on the reduced polynomials, they must be "polished" using the original full polynomial.
- Numerical Recipes implements this algorithm by `zroots` and `laguer` (Press et al.)
- An optimized root finding algorithm was published by Skowron and Gould (2012).

http://www.astrouw.edu.pl/~jskowron/cmplx_roots_sg

Point-source magnification

- Not all roots of p(z) are images.
  They must be checked with the original lens equation.
- When the source is outside the caustics, two roots are spurious and must be discarded.

- For each image we can calculate the magnification by the inverse Jacobian

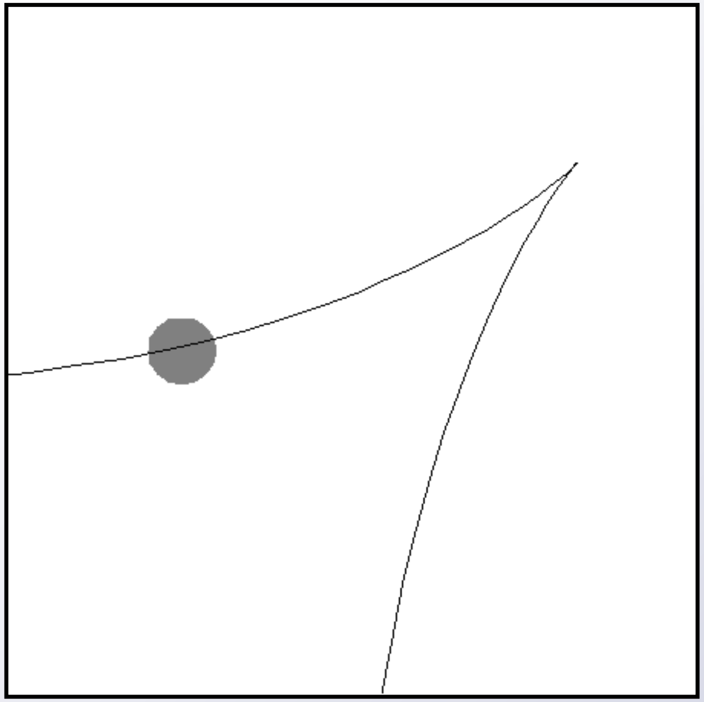
$$\mu_I = J^{-1}(z_I) = \frac{1}{\left|1 - \left|\frac{\partial \bar{\xi}}{\partial z}\right|^2\right|} \qquad \frac{\partial \bar{\xi}}{\partial z} = \frac{m_1}{(z_I - s)^2} + \frac{m_2}{z_I^2}$$

- The magnification of a point-source by a binary lens is then

$$\mu = \sum_I \mu_I$$

Finite-source effect

- We know that binary lenses have extended caustics where the magnification diverges.
- Finite source effects show up much more often than in the single lens case.



- Direct integration in the source plane is extremely unstable due to divergences.

$$\mu_{FS} = \int_{source} \mu_{PS} d^2 y$$

- Alternative algorithms needed.

Quadrupole and Hexadecapole

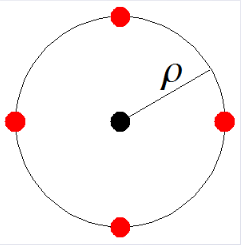
- Far from caustics, we can Taylor expand the magnification and take limb darkening into account

  (Pejcha & Heyrovsky 2007; Gould 2008; Cassan 2017)

$$A_{FS} = A_0 + \frac{A_2 \rho^2}{2}\left(1 - \frac{1}{5}\Gamma\right) + \frac{A_4 \rho^4}{3}\left(1 - \frac{11}{35}\Gamma\right) + \ldots$$

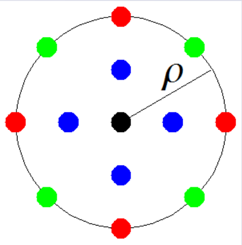
- The coefficients can be obtained by averaging the magnification calculated on few points on the boundary:

$$A_{\rho,+} = \tfrac{1}{4}\sum_{j=0}^{3} A\left[\rho\cos\left(j\pi/2\right), \rho\sin\left(j\pi/2\right)\right] - A_0$$



- Quadrupole: $A_2 \rho^2 = A_{\rho,+}$

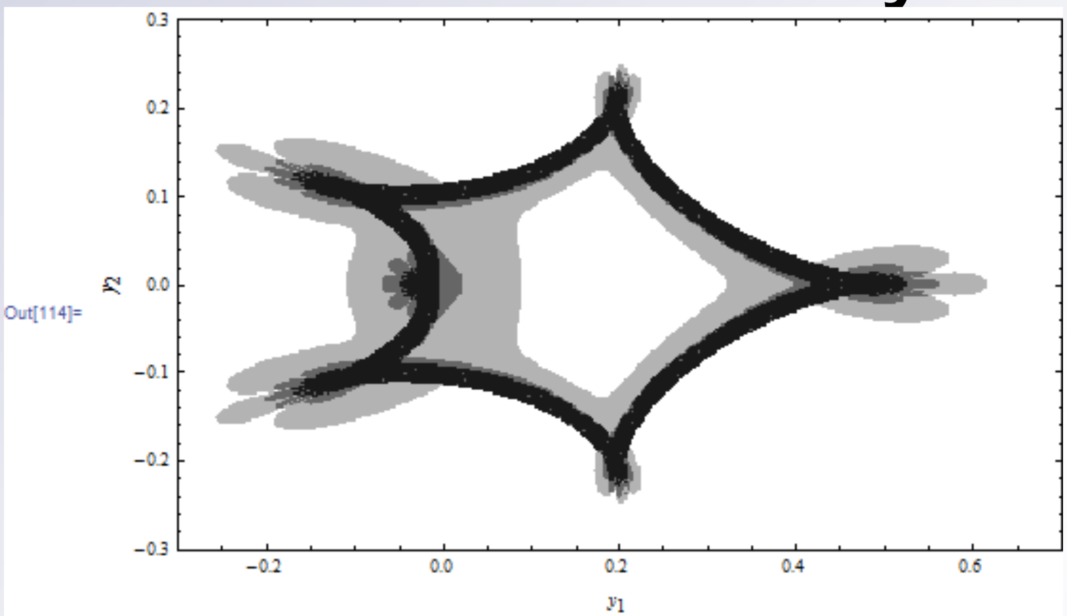
- Hexadecapole:



$$A_2 \rho^2 = \frac{16 A_{\rho/2,+} - A_{\rho,+}}{3}; \quad A_4 \rho^4 = \frac{A_{\rho,+} + A_{\rho,\times}}{2} - A_2 \rho^2$$

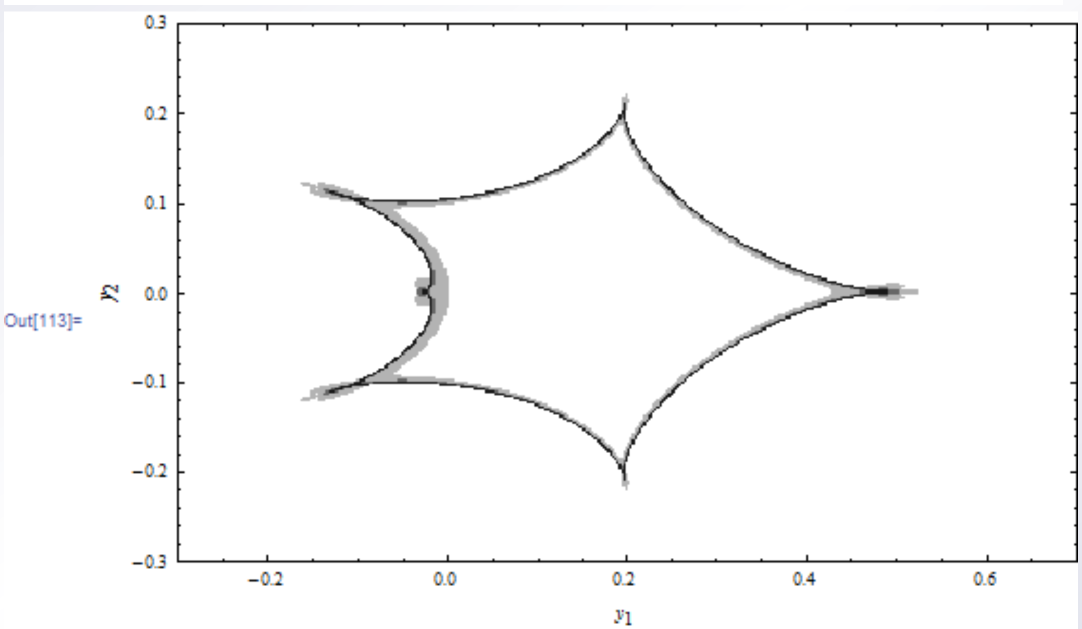
2. Finite-source approximations

Validity range



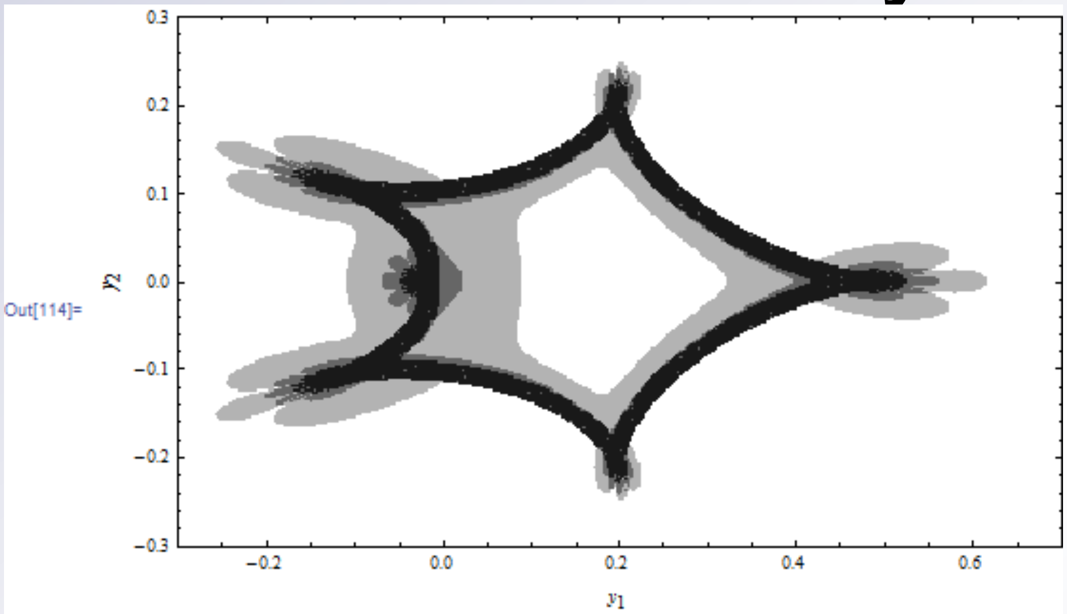
ρ = 0.01
Accuracy = 0.01

- ☐ Point-source valid
- Quadrupole valid
- Hexadecapole valid
- Full calculation needed



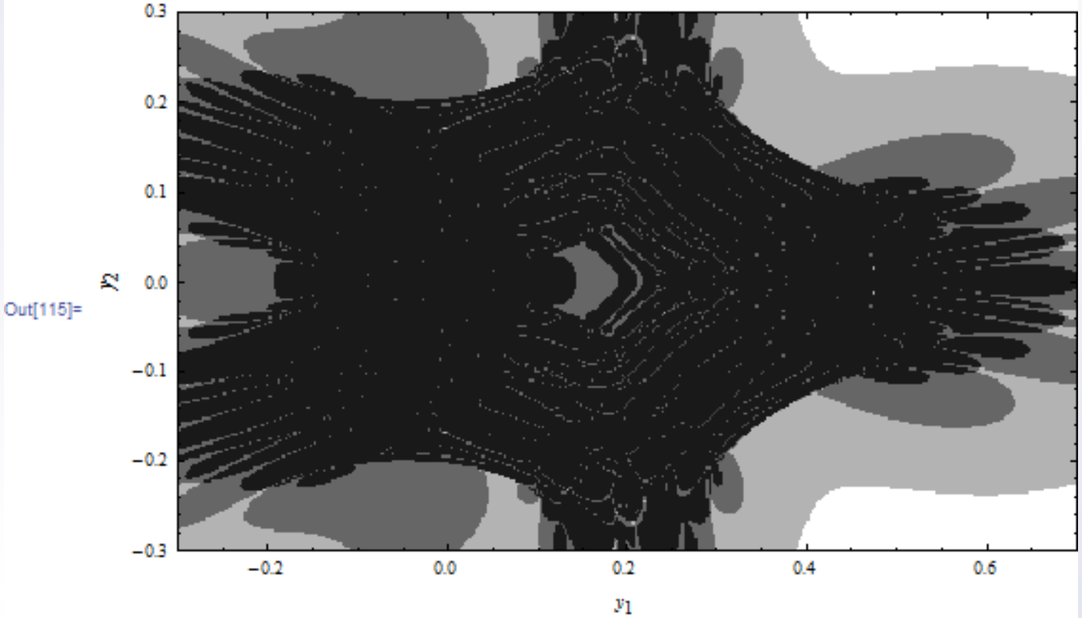
ρ = 0.001
Accuracy = 0.01

Validity range



ρ = 0.01
Accuracy = 0.01

- ☐ Point-source valid
- Quadrupole valid
- Hexadecapole valid
- Full calculation needed

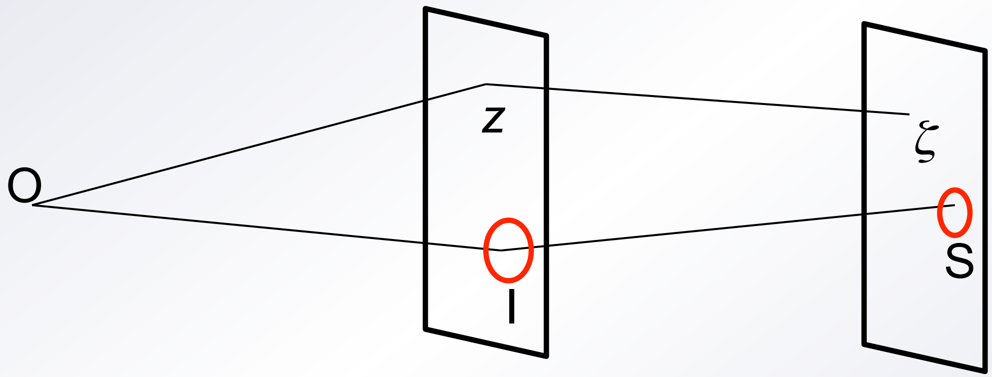


ρ = 0.1
Accuracy = 0.01

Inverse ray shooting

- For each point in the lens plane z, the lens map gives the position ζ in which a source should lie in order to have an image in z.

$$\zeta = z - \frac{m_1}{\bar{z} - s} - \frac{m_2}{\bar{z}}$$



- By scanning the whole lens plane, we can find all images.
- The area of the images is proportional to the number of rays landing at the source.
- Limb darkening obtained by weighing rays by the source brightness at landing point.

- Every ray requires little computation.
- Large numbers of rays needed to be accurate.

Magnification maps

- A uniform scansion of the lens plane results in a magnification map in the lens plane. (Wambsganns 1992, 1997)

- This can be re-used for any source trajectories on the same lens model.
- A broad search in the parameter space is cheap for fixed s and q.

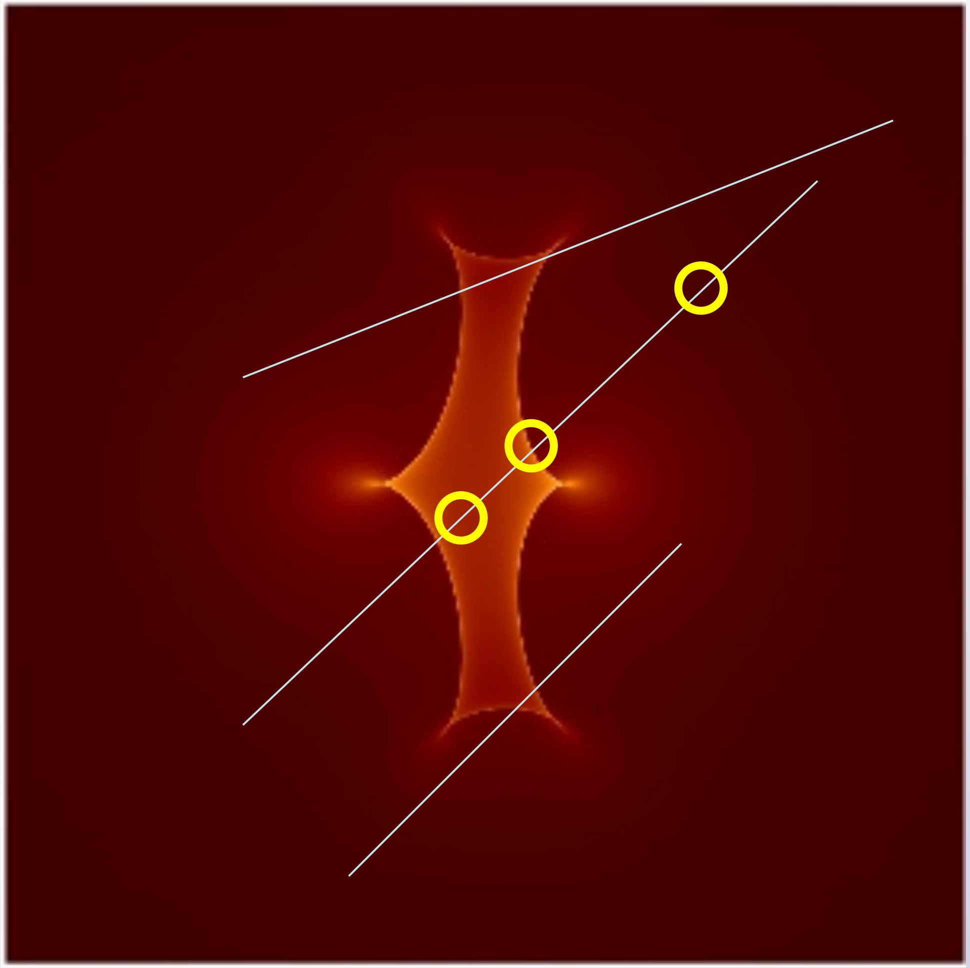
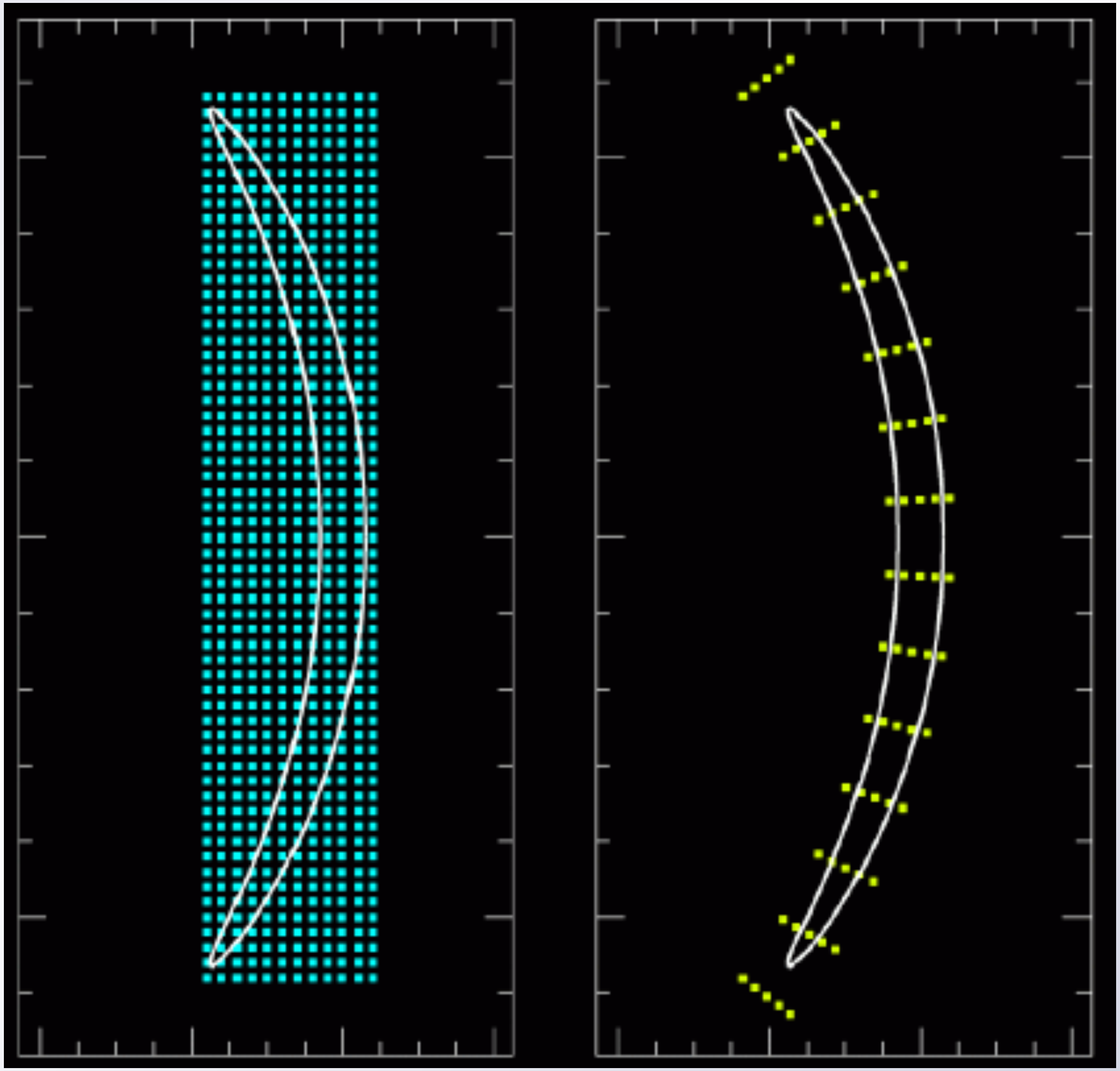


Image-centered ray-shooting

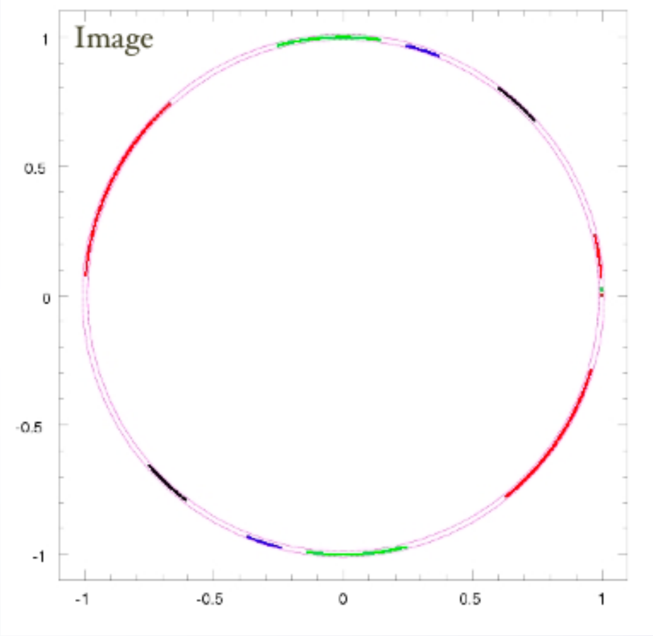
- First solve the lens equation for the center of the source.
- Then shoot rays around to get all the images (Bennett & Rhie 1996; Bennett 2010)

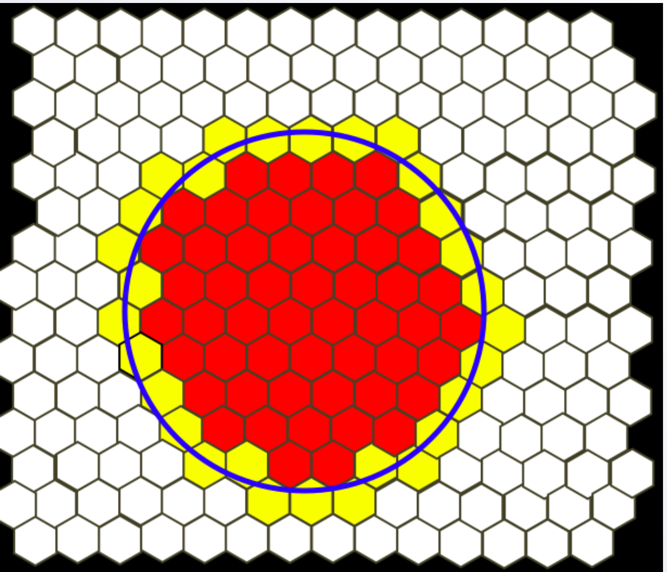
- Polar coordinates help diminish the number of rays.



Contours for driving ray-shooting

- For high-magnification events, ray shooting can be limited to an annulus around the Einstein ring (Dong et al. 2006)
- Otherwise, the regions in which to shoot rays can be defined by the boundaries of the images of a circle larger than the source.





- Light rays can be collected in hexagonal pixels
- Check pixels instead of rays (Dong et al. 2009)

Inverse-ray-shooting on GPUs

- The single ray shot is simple enough to be parallelized on GPUs.
- Joe Ling (NZ) has developed a fast working code for inverse-ray shooting on GPUs.

- Huge magnification maps can be generated quite rapidly.
- However, if they are not re-used, still we need image-centered shooting.

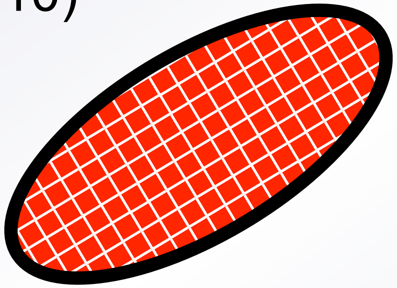
Inverse-ray-shooting: pros and cons

Pros:
- Individual rays require few operations
- Can be implemented on GPUs
- Magnification maps can be re-used
- Incorporates limb darkening

Cons:
- Large number of rays (scales as the area)
- Denser sampling required for smaller sources
- For non-static lenses, maps cannot be re-used

Contour Integration Concept

- <u>Contour integration concept</u>:
  The area enclosed in a curve is expressed by a simple contour integral on the boundary.
  (Schramm & Kayser, 1987; Dominik 1995; Gould & Gaucherel 1997; Dominik 1998; VB 2010)



- We only need to find the boundaries of the images
- A surface integral becomes a one-dimensional integral
- In principle this is much faster and very elegant
- In practice, a lot of work is required to keep everything under control.

Green's Theorem

- Green's theorem: $$\oint_{\partial I}\left(L_1 dx_1 + L_2 dx_2\right) = \iint_I dx_1 dx_2 \left(\frac{\partial L_2}{dx_1} - \frac{\partial L_1}{dx_2}\right)$$

  Note 1: ∂I is the counterclockwise boundary of I.
  Note 2: Green's theorem is the two 2-d specification of Stokes' theorem.

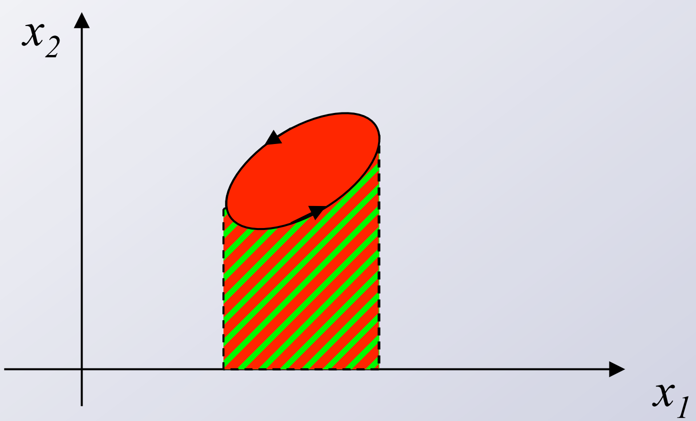
- If we want the area of the domain I we must choose

$$\left(\frac{\partial L_2}{dx_1} - \frac{\partial L_1}{dx_2}\right) = 1$$

- Possible choices for (L_1, L_2) are $(-x_2, 0)$, $(0, x_1)$, $(-x_2, x_1)/2$.

- Then the line integral takes the equivalent forms

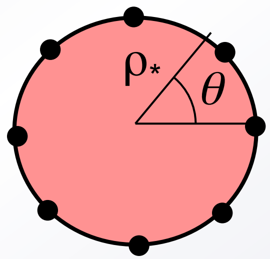
$$A = -\oint_{\partial I} x_2 dx_1 = \oint_{\partial I} x_1 dx_2 = \tfrac{1}{2}\oint_{\partial I} \mathbf{x} \wedge \mathbf{dx}$$



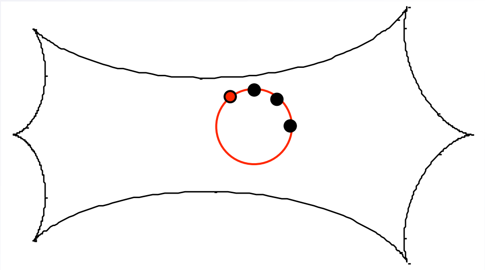
From source to image boundaries

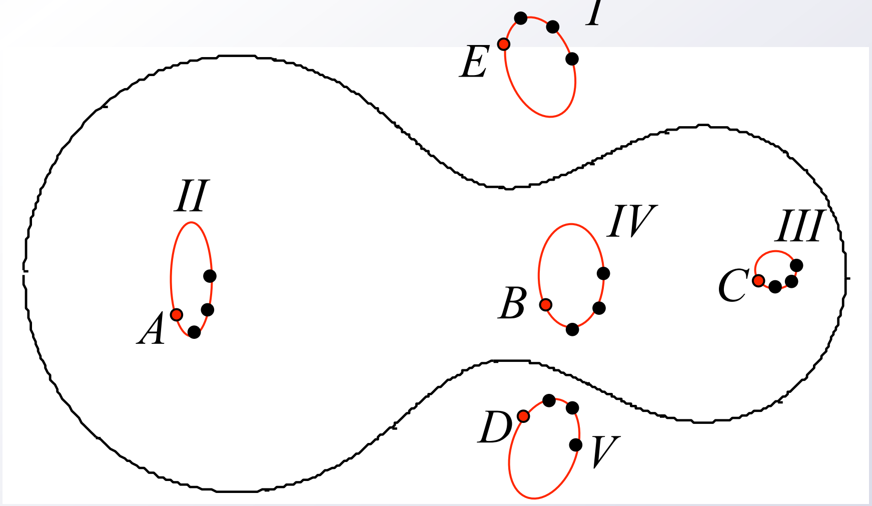
- Parameterization of the source boundary:

$$\mathbf{y} = \mathbf{y}_0 + \rho_* \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \quad \Leftrightarrow \quad \zeta = \zeta_0 + \rho_* e^{i\theta}$$



- After inversion of the lens equation, for each θ_i we get 3 or 5 points $z_{i,I}$ lying on the boundaries of the images.





- We need to associate the roots $z_{i,I}$ at step i with the roots $z_{i-1,I}$ of the step $i-1$.

Reconstruction of image boundaries

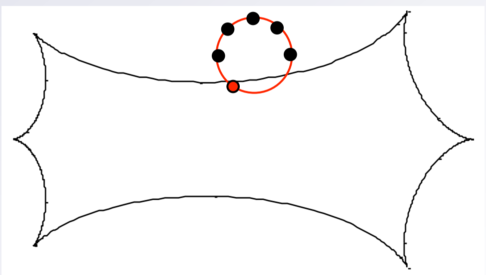
- We need to associate the roots $\mathbf{x}_{i,I}$ at step i with the roots $\mathbf{x}_{i-1,I}$ of the step i -1.

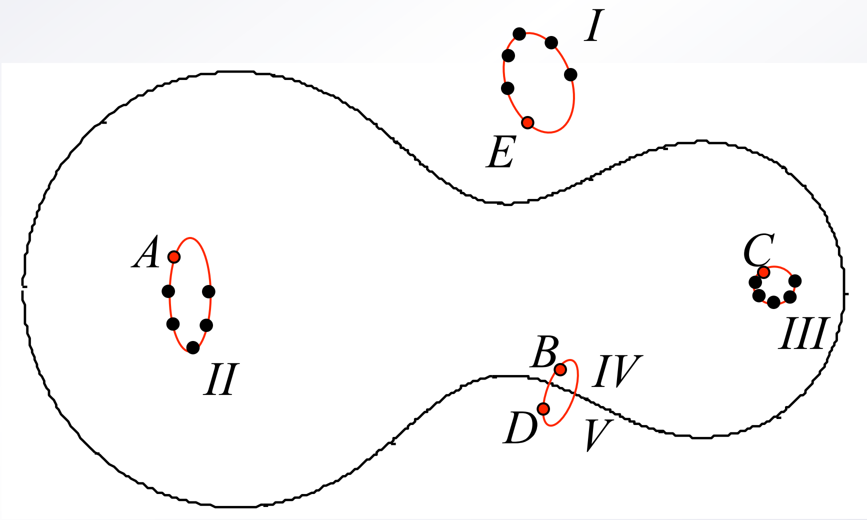
- The simplest way is to use the least distance criterium.
- Only same parity solutions can be associated.

$$
\begin{array}{ccccc}
\mathbf{x}_{i-2,I} & \mathbf{x}_{i-2,II} & \mathbf{x}_{i-2,III} & \mathbf{x}_{i-2,IV} & \mathbf{x}_{i-2,V} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\mathbf{x}_{i-1,I} & \mathbf{x}_{i-1,II} & \mathbf{x}_{i-1,III} & \mathbf{x}_{i-1,IV} & \mathbf{x}_{i-1,V} \\
\downarrow & \searrow & \swarrow & \searrow & \swarrow \\
\mathbf{x}_{i,A} & \mathbf{x}_{i,B} & \mathbf{x}_{i,C} & \mathbf{x}_{i,D} & \mathbf{x}_{i,E}
\end{array}
$$

Reconstruction of image boundaries

- If two new images are created at step i, we can recognize them as the last two unmatched roots.





- The same can be done at destruction of two images.
- We must keep track of pairing between image boundaries when they are created or destroyed (see next).

Reconstruction of image boundaries

- If two new images are created at step i, we can recognize them as the last two unmatched roots.

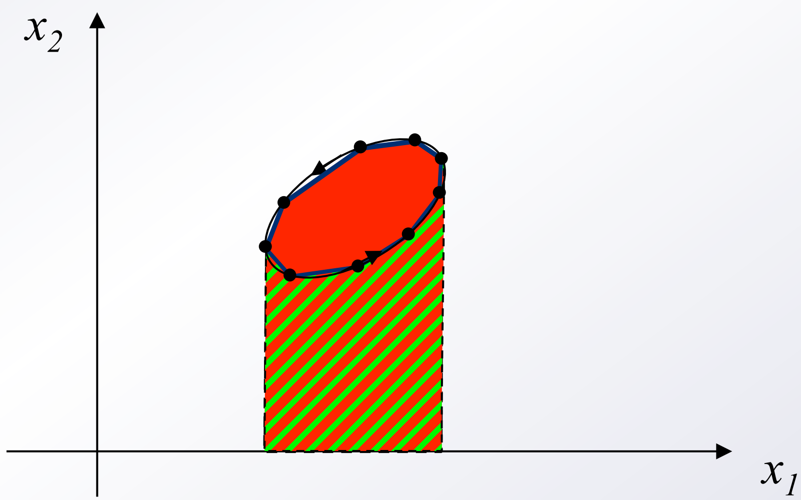
$$
\begin{tikzcd}
\mathbf{x}_{i-2,I} & \mathbf{x}_{i-2,II} & \mathbf{x}_{i-2,III} & & \\
\vdots & \vdots & \vdots & & \\
\mathbf{x}_{i-1,I} \arrow[d] & \mathbf{x}_{i-1,II} \arrow[drr] & \mathbf{x}_{i-1,III} \arrow[dl] & & \\
\mathbf{x}_{i,A} & \mathbf{x}_{i,B} & \mathbf{x}_{i,C} \arrow[rr, bend left, dash] & \mathbf{x}_{i,D} & \mathbf{x}_{i,E}
\end{tikzcd}
$$

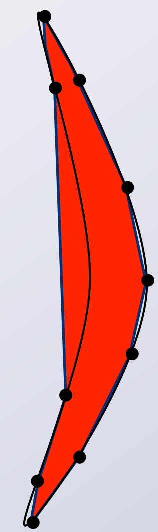
- The same can be done at destruction of two images.
- We must keep track of pairing between image boundaries when they are created or destroyed (see next).

Contour integration by polygonal

- The trapezium approximation gives the area of the polygonal defined by our image boundary sample

$$A = \tfrac{1}{2}\oint_{\partial I} \mathbf{x} \wedge \mathbf{dx} \cong \frac{1}{4}\sum_{i=0}^{n-1}\left(\mathbf{x}_{i+1} + \mathbf{x}_i\right) \wedge \left(\mathbf{x}_{i+1} - \mathbf{x}_i\right) = \frac{1}{2}\sum_{i=0}^{n-1}\mathbf{x}_i \wedge \mathbf{x}_{i+1}$$



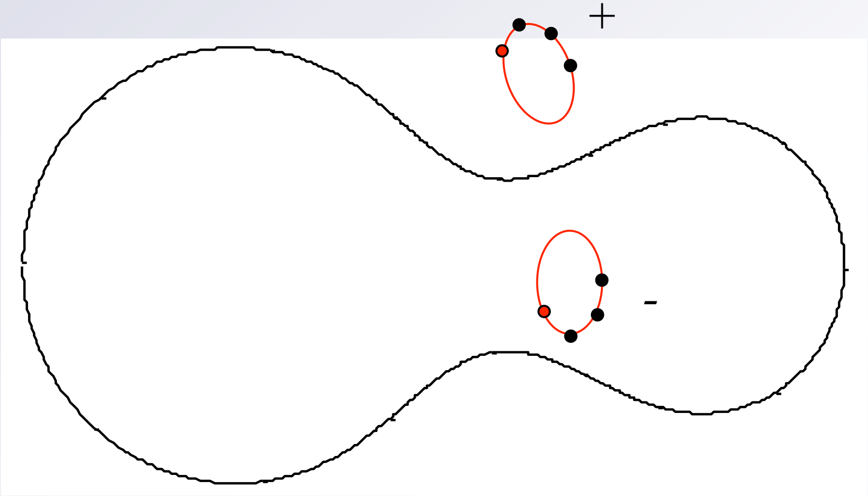


- Typically, the area is underestimated.
- … with some exceptions.

Contour integration by polygonal

- We must multiply the contour integrals by the parities of the boundaries:

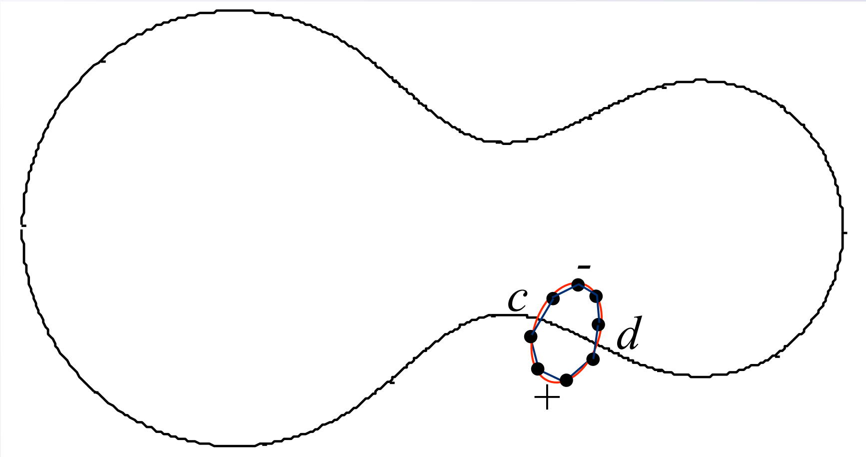
$$A_I = \frac{1}{2} p_I \sum_{i=0}^{n-1} \mathbf{x}_i \wedge \mathbf{x}_{i+1}$$



- For creation/destruction cases, we need to add a connection term.

$$\frac{1}{2}\left(\mathbf{x}_{\text{first},-} \wedge \mathbf{x}_{\text{first},+}\right)$$ for creation

$$\frac{1}{2}\left(\mathbf{x}_{\text{last},+} \wedge \mathbf{x}_{\text{last},-}\right)$$ for destruction



Summing up…

Steps in contour integration:

- Run a root finder routine for each point in the source boundary.
- At each step you must put the roots in the correct image boundary (least distance criterium) and keep track of created and destroyed pairs.
- Calculate the contour integral by polygonal approximation for each boundary.
- Sum up the contour integrals with the correct parity and add a connection term for each creation/destruction.

Order of the error

Let us estimate the order of the error

- At each step, the contribution of the interval $\Delta\theta$ to the contour integral is

$$\Delta A_I = \frac{1}{2}\int_{I(\Delta\theta)} \mathbf{x}_I \wedge d\mathbf{x}_I = \frac{1}{2}\int_{\theta_i}^{\theta_i+\Delta\theta} \mathbf{x}_I \wedge \mathbf{x}'_I d\theta$$

- The trapezium approximation is actually

$$\Delta A_I^{(t)} = \frac{1}{2}\mathbf{x}_I(\theta_i) \wedge \mathbf{x}_I(\theta_i + \Delta\theta)$$

- Expanding in powers of $\Delta\theta$, the difference is of third order

$$\Delta A_I^{(t)} - \Delta A_I = O\left(\Delta\theta^3\right)$$

Parabolic correction

- We can increase the accuracy without adding new points to the boundary.
  (VB, MNRAS 1365, 2966 (2010))

- If we add the following correction to the trapezium

$$\Delta A_I^{(p)} = \frac{1}{24}\left[\left(\mathbf{x}_I' \wedge \mathbf{x}_I''\right)\Big|_{\theta_i} + \left(\mathbf{x}_I' \wedge \mathbf{x}_I''\right)\Big|_{\theta_i+\Delta\theta}\right]\Delta\theta^3$$

… the residual is of fifth order

$$\Delta A_I^{(t)} + \Delta A_I^{(p)} - \Delta A_I = O\left(\Delta\theta^5\right)$$

- The wedge products of the derivatives can be calculated analytically using the lens map.
- Similar parabolic corrections can be introduced for creation/ destruction terms.

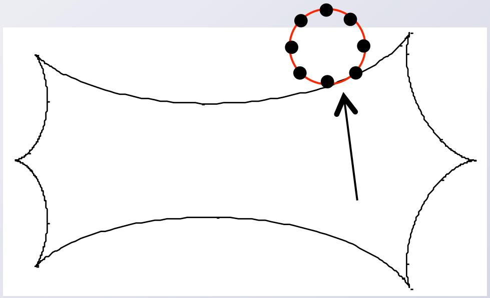
Error control

- In all numerical computations it is fundamental to have an estimate of the errors.
- The error estimators must be reliable but also cheap.

$$E_{I,i,1} = \frac{1}{48}\left| \left(\mathbf{x}'_I \wedge \mathbf{x}''_I\right)\Big|_{\theta_i} - \left(\mathbf{x}'_I \wedge \mathbf{x}''_I\right)\Big|_{\theta_i+\Delta\theta} \right| \Delta\theta^3$$

$$E_{I,i,2} = \frac{3}{2}\left| \Delta A_I^{(p)} \left(\frac{\Delta\tilde{\theta}^2}{\Delta\theta^2} - 1 \right) \right|$$

$$E_{I,i,3} = \frac{1}{10}\left| \Delta A_I^{(p)} \right| \Delta\theta^2$$



- These work in a complementary way and are combinations of quantities already calculated.
- Similar estimators can be introduced for creation/destruction terms and to unveil “hidden” images.

Error estimate at step i

- Our error estimate for the step i is thus

$$E_i = \sum_I \left(E_{I,i,1} + E_{I,i,2} + E_{I,i,3} \right)$$

- If creation/destruction occurs at step i we add

$$E_i += E_1^{(c)} + E_2^{(c)} + E_3^{(c)}$$

- The total error in the area of all images is $$E = \sum_{i=0}^{n-1} E_i$$

- At this point we are able to check if we have reached the target accuracy δμ in the magnification:

$$\frac{E}{\pi \rho_*^2} < \delta\mu$$

- If not, we must increase the sampling.

Optimal sampling

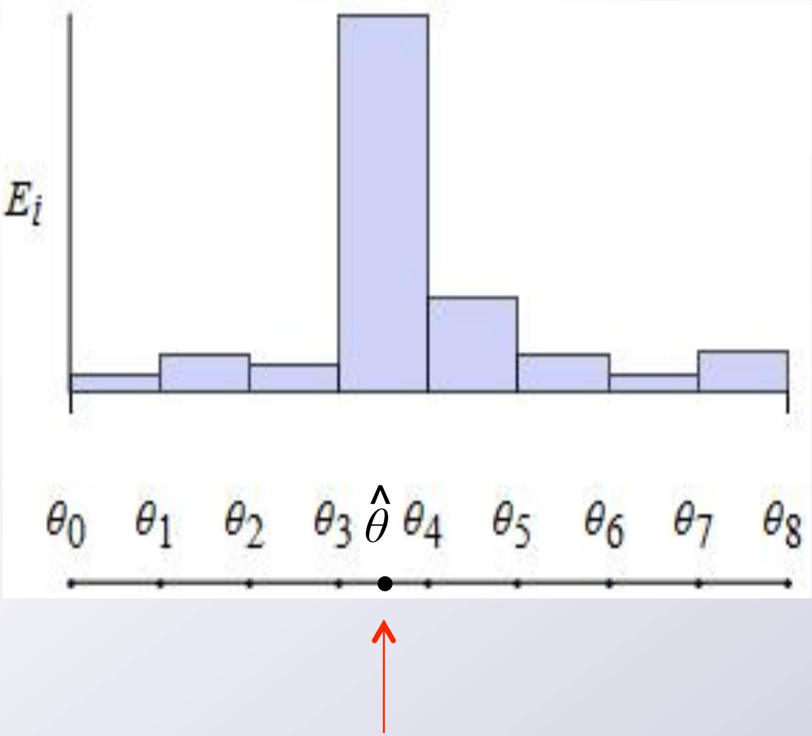
- We can pick the interval with the largest error

$$\text{Let} \quad \hat{i} : E_i \leq E_{\hat{i}} \quad \forall i$$

- … and add another point in the sample in the middle of this interval:

$$\hat{\theta} = \frac{\theta_{\hat{i}} + \theta_{\hat{i}+1}}{2}$$

- Then we just need to recalculate the contour integral and the error estimators in the new sub-intervals.
- In this way, sampling is increased only where needed, avoiding useless calculations.



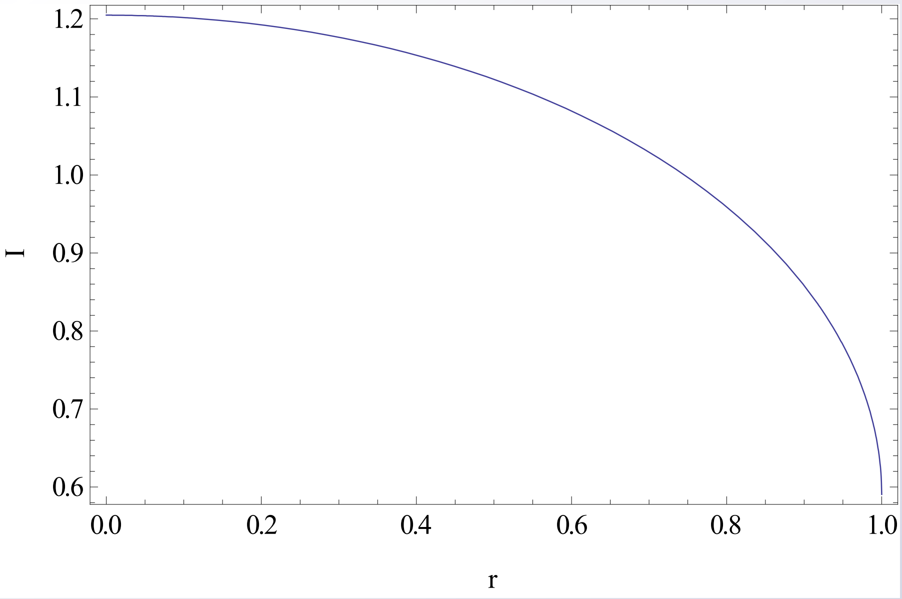
Limb darkening

- Up to now, we have assumed a uniform brightness source.
- However, physical stars have a limb-darkened profile, e.g. Milne's linear law

$$f(r) = \frac{1}{1-a/3}\left[1-a\left(1-\sqrt{1-r^2}\right)\right] \text{ with } r = \frac{\rho}{\rho_*}$$

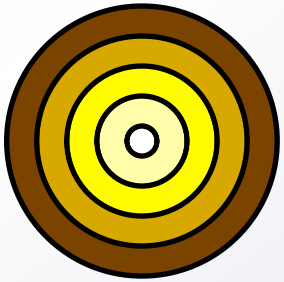
- In general, the source profile is a function f(r), normalized in such a way that

$$\int_0^1 2rf(r)dr = 1$$



Limb darkening

- In order to account for limb darkening with contour integration we may divide the source in annuli.



- Each source annulus is magnified by microlensing. The exact contribution to the total amplification is

$$M_i = \frac{1}{\pi} \int_{r_{i-1}}^{r_i} dr\, 2r f(r) \int_0^{2\pi} \mu(r,\varphi) d\varphi$$

- In each annulus we instead use a uniform brightness given by the limb-darkened profile averaged over the annulus

$$f_i = \frac{F(r_i) - F(r_{i-1})}{r_i^2 - r_{i-1}^2} \quad \text{with} \quad F(r) = \int_0^r dr' 2r' f(r')$$

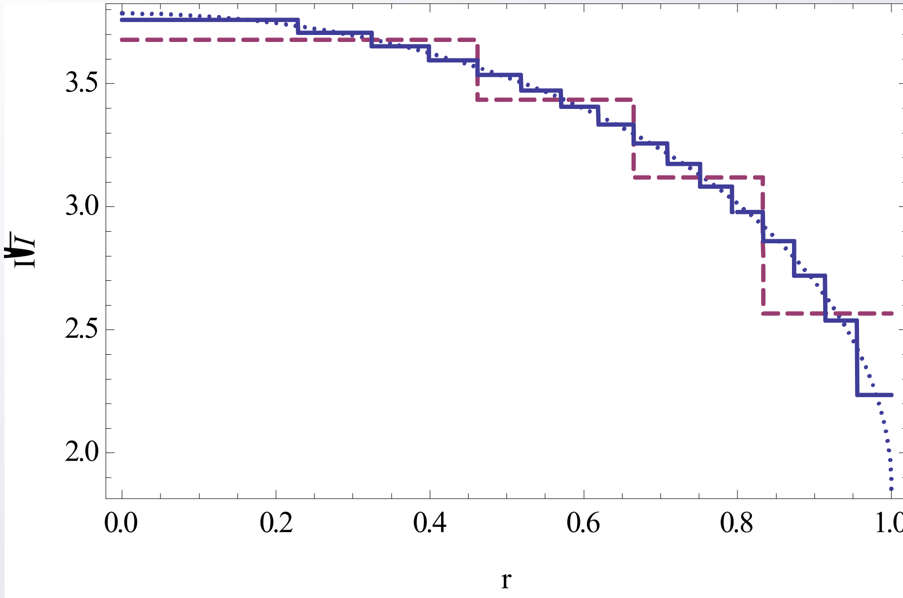
$$\widetilde{M}_i = f_i \left[\mu_i r_i^2 - \mu_{i-1} r_{i-1}^2 \right] \quad \text{where} \quad \mu_i = \frac{1}{\pi r_i^2} \int_{r_{i-1}}^{r_i} dr\, 2r \int_0^{2\pi} \mu(r,\varphi) d\varphi$$

Sampling the source profile

- Error estimators can be introduced also for limb darkening. They are then used to drive the profile sampling.
- We start with the two extremal annuli: the boundary (r=1) and the center (r=0).
- The new circle is put at an intermediate radius $\bar{r}$ so that the two new annuli give the same contribution to the source luminosity:

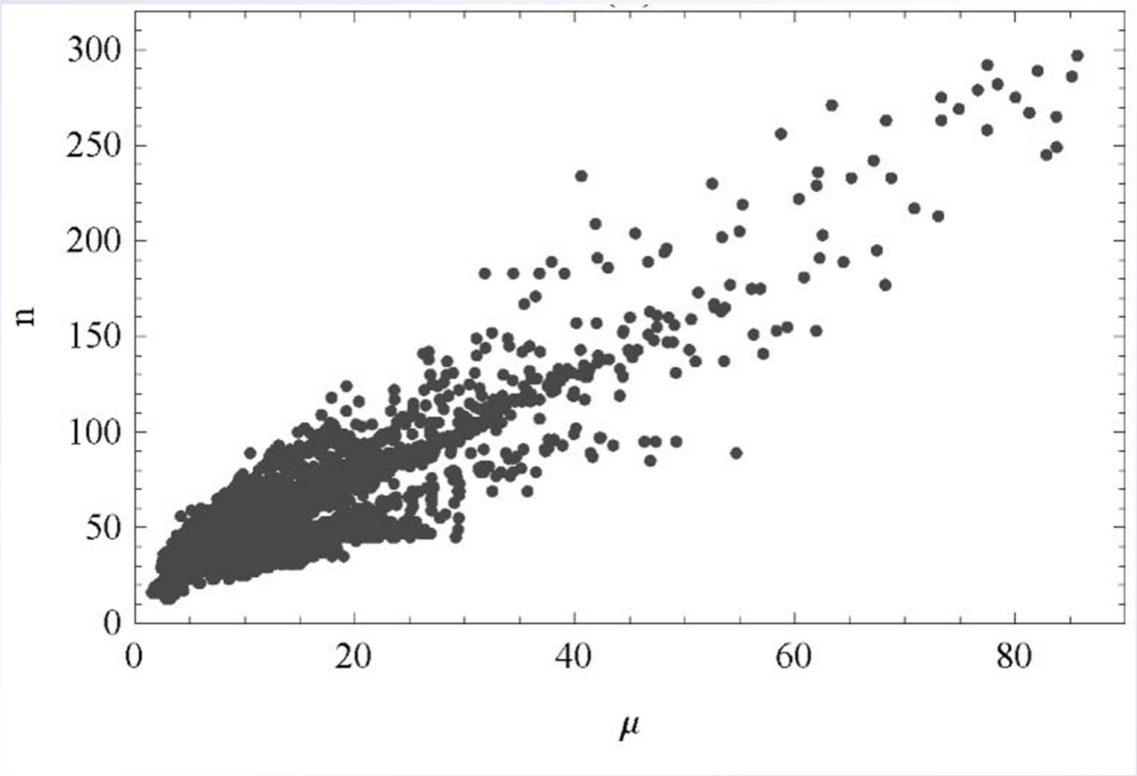
$$F(r_j) - F(\bar{r}) = F(\bar{r}) - F(r_{j-1})$$

- We keep introducing annuli until the total error falls below the target accuracy.



Testing

- This is a scatter plot of number of sampling points vs magnification (target accuracy is 10^{-2}).



Testing

- Summing up, at $\delta\mu=10^{-2}$ we get
  - a speed-up of 4 thanks to parabolic correction
  - a speed-up ranging from 3 to 20 thanks to optimal sampling
  - a slow-down from 2 to 10 if we include limb darkening
- No redundant calculation thanks to error estimators!

VBBinaryLensing

- **VBBinaryLensing** is a code for the calculation of **microlensing light curves** based on advanced contour integration (VB, MNRAS 1365, 2966 (2010)) .
    - Point-Source Point-Lens
    - Extended Source Point-Lens
    - Binary Source Point-Lens
    - Extended Source Binary Lens
- **Higher order effects** implemented:
    - Linear limb darkening
    - Annual and space parallax
    - Circular orbital motion
- C++ library.
- Tested on Windows, Linux, Mac OS.
- Importable in Python.
- Source code is public (but no specific standard has been adopted!).

Release of VBBinaryLensing

- **VBBinaryLensing** is available at *http://www.fisica.unisa.it/GravitationAstrophysics/VBBinaryLensing.htm*.
- The **zip file** contains:
  - readmeVB.txt — Generic introductory information
  - VBBinaryLensingLibrary.h — C++ header
  - VBBinaryLensingLibrary.cpp — C++ source
  - main.cpp — Sample code with examples and instructions.
  - Makefile.dat — Example of a makefile (courtesy of Zhu)
  - howtopython.txt — Instructions for wrapping the library in Python code (courtesy of Hundertmark)
  - OB151212coords.txt — Sample coordinate file for an event (used in the examples in main.cpp)
  - satellite1.txt — Table for the positions of a satellite for space parallax calculation (Spitzer)
  - satellite2.txt — Same for Kepler.

Example of use

```
#include <stdio.h>
#include “VBBinaryLensingLibrary.h”


int main()
{
   VBBinaryLensing VBBL;

   double Mag,s,q,y1,y2,Rs,accuracy;
   s=0.8; //separation
   q=0.1; // mass ratio
   y1=0.01; // source position
   y2=0.3;
   Rs=0.01; // source radius

   accuracy=1.e-2; // Required accuracy of the result
   Mag=VBBL.BinaryMag(s,q,y1,y2,Rs,accuracy);

   printf("Magnification = %lf\n",Mag);

   return 0;
}
```

Contour integration: pros and cons

- Contour integration is a very elegant way to calculate the microlensing magnification.

Cons:
- Complicated!
- Limb darkening comes at a substantial cost.

Pros:
- 1-d integration instead of 2-d integration (much faster!)
- Faster on small sources
- Only public code available, with large feedback from the community.

Microlensing computation: outlook

- The huge data flow from WFIRST will require very high performance for microlensing calculations.
- Multiple systems likely to show up.

- Inverse-ray-shooting has already been implemented for triple and multiple lenses.
- Contour integration has never been tried beyond binary lensing so far.

- There is still room for optimizations, speed-up, parallelization on different codes.
- New ideas are always welcome.