Ciencia y Tecnología **23** No.1-2(1999) 1-92

Visualización Científica de Lentes Gravitacionales

Marcelo Magallón
Laboratorio de Investigaciones Astrofísicas
Escuela de Física
Universidad de Costa Rica, San José, Costa Rica

Abstract: Concepts related to gravitational lenses are discussed and applied to develope an interactive visualization tool that allow us to investigate them. Optimization strategies were performed to elaborate the tool. Some results obtained from the application of the tool are shown.

Subject headings: Gravitational lenses, optimization technics, visualization tools.

Resumen: Se exponen los conceptos y resultados asociados al fenómeno de lentes gravitacionales. Estos conceptos son aplicados al desarrollo de una herramienta interactiva que permite el estudio del mismo. Se exponen las estratégias de optimización empleadas en el desarrollo de esta herramienta y se muestran algunos resultados que se obtienen al utilizarla.

Descriptores: Lentes gravitacionales, técnicas de optimización, herramientas de visualización

1. Introducción

La *visualización científica* es un área activa y vital de la investigación, enseñanza y desarrollo en la actualidad [NHM97], y ha venido desarrollándose a un ritmo acelerado gracias a los grandes avances que se han dado en la industria de la computación. El éxito que la visualización científica ha tenido se debe primordialmente a la solidez de la premisa detrás de ella: la idea básica de utilizar gráficos generados por computadora para obtener información y entendimiento sobre la *data* (geometría) y las relaciones (topología). Este es un concepto en extremo simple, pero muy importante, el cual está teniendo un profundo y difundido impacto en la metodología de la ciencia y la ingeniería. También es de señalar que no es sino hasta hace relativamente poco tiempo que la tecnología involucrada ha bajado su costo hasta niveles que permiten su utilización más generalizada.

El estudio de lentes gravitacionales, y específicamente, de *micro lentes gravitacionales*, han probado en los últimos años ser uno de los campos más prometedores en la determinación de la cantidad y distribución espacial de materia oscura [NB98], utilizando para esto algunos métodos de visualización. Además, mediante el fenómeno de lentes gravitacionales se puede obtener información en un gran rango de escalas de distancia. Si bien es cierto que este fenómeno ha contribuido históricamente a la verificación de la Relatividad General, ahora gracias a los avances en instrumentación y al desarrollo teórico, el tema se ha ubicado en un lugar importante dentro de la cosmología moderna. Mediante el mismo es posible comparar las cantidades totales de materia con las de materia luminosa, e imponer así nuevos límites sobre la densidad media de materia en el Universo. Para tal efecto, es importante y necesario el correcto *modelado* de lentes gravitacionales. Por modelado se entiende la determinación de la configuración de la lente (tipo, densidad superficial de materia, disposición espacial, etc), el tipo de objeto que constituye la fuente, las distancias involucradas (fuente-lente, lente-observador, fuente-observador).

Uno de los problemas importantes con los cuales se enfrenta el investigador es que el fenómeno de lentes gravitacionales es, si se le compara con, v.g., *gamma-ray bursts* (GRB), poco frecuente, y generalmente el proceso de reducción de datos y posterior estudio para la determinación de un evento de lente gravitacional lleva gran cantidad de tiempo. Sin embargo, existen proyectos activos para el análisis sistemático y continuo de candidatos para lentes gravitacionales, en particular CLASS [1], OGLE [2], CASTLE [3].

Gracias a la disponibilidad de computadoras cada vez más potentes con cantidades de memoria interna y externa que van en aumento, es posible realizar simulaciones *dinámicas* de eventos de microlentes. El problema que se encuentra aquí es que típicamente no hay suficientes ligaduras para determinar la solución unívocamente y quedan varios (¿muchos?) parámetros libres.

El problema con el que se trabaja es entonces multidimensional en más de un sentido:

1. involucra cálculos en dos y tres dimensiones

2. la cantidad de parámetros libres que aumentan la dimensionalidad del mismo.

---

[1] `http://dept.physics.upenn.edu/%7Emyers/class.html`
[2] `http://www.astro.princeton.edu/%7Eogle/`
[3] `http://cfa-www.harvard.edu/glensdata/`

Bajo estas condiciones y con la finalidad de investigar la influencia de los parámetros en la problemática de lentes gravitacionales, es que se utiliza la visualización y determinar así el efecto que uno o varios de estos puedan tener sobre la imagen que se obtiene. Por supuesto, todo esto tiene sentido siempre y cuando se puedan salvar y reproducir las simulaciones realizadas, para compararlas con la realidad física del fenómeno.

2. Mecanismos Físicos

El mecanismo físico involucrado en el fenómeno de lentes gravitacionales es conocido desde los inicios de la Relatividad General [Ein11]. Una masa desviará los fotones que pasen en su cercanías, respecto a la trayectoria no perturbada, un ángulo por unidad de longitud, $\frac{\delta\alpha}{\delta s}$, dado por:

$$\frac{\delta \boldsymbol{\alpha}}{\delta s} = -2\nabla_x \frac{\Phi}{c^2} \,, \tag{1}$$

donde la derivada espacial se toma en un plano ortogonal a la trayectoria del fotón y Φ es el potencial Newtoniano.

2.1 Aproximaciones

En las aplicaciones de importancia astrofísica, el ángulo total de desviación es como máximo del orden de un minuto de arco. Además, el desvío ocurre en las proximidades de la masa considerada, esto quiere decir que es posible considerar las trayectorias en una región suficientemente lejana como líneas rectas (aproximación de Born).

Adicionalmente, la desviación ocurre en una pequeña parte de la trayectoria total del rayo considerado, por lo que se puede aproximar la región de desviación mediante un plano. De esta manera, se describe el efecto de lente como si ocurriese en forma instantánea en el momento que un rayo de luz cruza este plano (aproximación de lente delgada).

En la figura 1 se muestra la geometría de una lente gravitacional para el caso de una masa puntual ubicada a una distancia D_d del observador, con una fuente a una distancia D_s. El eje OD se denomina *eje óptico*. Se indican la separación angular entre la fuente S y la lente D ($\boldsymbol{\beta}$), el ángulo de desviación del rayo de luz ($\hat{\boldsymbol{\alpha}}$) y la posición angular de la fuente ($\boldsymbol{\theta}$). De aquí se obtiene directamente:

$$D_s\boldsymbol{\theta} = D_s\boldsymbol{\beta} + D_{ds}\hat{\boldsymbol{\alpha}}$$

$$\boldsymbol{\beta} = \boldsymbol{\theta} - \frac{D_{ds}}{D_s}\hat{\boldsymbol{\alpha}} = \boldsymbol{\theta} - \boldsymbol{\alpha} \,, \tag{2}$$

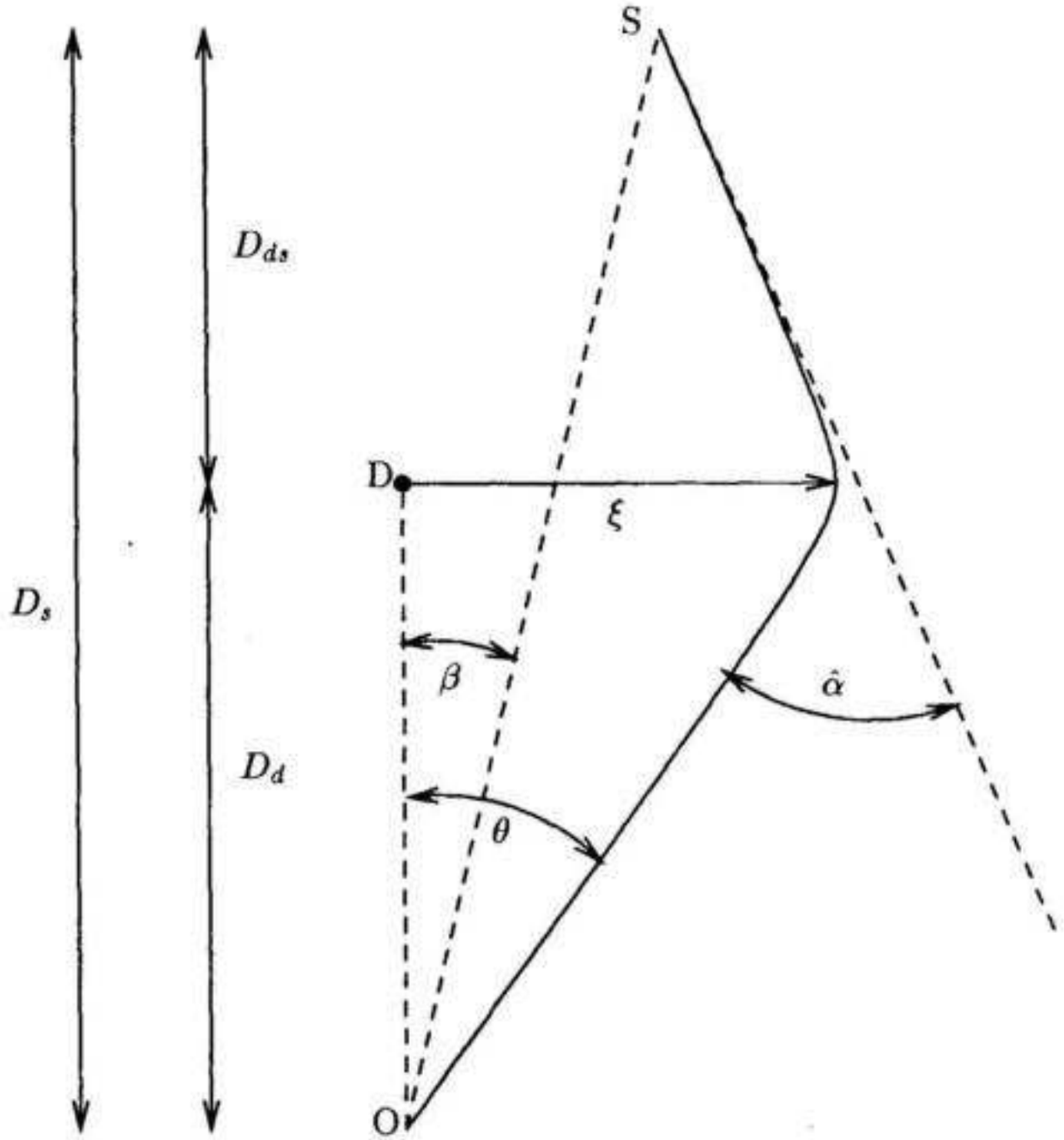

Figura 1: Geometría de una lente gravitacional para el caso de una masa puntual.

donde $\alpha = \frac{D_{ds}}{D_s}\hat{\alpha}$. Si $\xi = D_d\theta$ describe la separación entre el rayo y el eje OD, y $\eta = D_s\beta$ es la distancia de la fuente a este eje, se tiene:

$$\eta = \frac{D_s}{D_d}\xi - D_{ds}\hat{\alpha}(\xi) \, . \tag{3}$$

Esto es, conociendo la posición de las imágenes (ξ) es posible derivar directamente su posición en la fuente si se conocen la distribución de materia $\hat{\alpha}(\xi)$ y las cantidades D_{ds} y $\frac{D_s}{D_d}$. Es de notar que *no* es necesario conocer la distancia D_d ó D_s, sólo la relación $\frac{D_s}{D_d}$.

2.2 Masa puntual

El potencial para una masa puntual M es

$$\Phi(r) = -\frac{GM}{r} \, .$$

Se considera un parámetro de impacto ξ, junto con x la abscisa al punto de la trayectoria que está más cerca de la lente, el potencial está dado por:

$$\Phi(x) = -\frac{GM}{\sqrt{\xi^2 + x^2}} \, . \tag{4}$$

2.2.1 Multiplicidad de imágenes

Introduciendo el potencial 4 en la ecuación 1, se tiene:

$$\frac{\delta\alpha}{\delta x} = -2\frac{GMx}{c^2(\xi^2 + x^2)^{3/2}} \, .$$

El ángulo total de desviación se obtiene al integrar respecto a x sobre toda la trayectoria,

$$\hat{\alpha} = \frac{4GM}{c^2\xi} = \frac{2R_S}{\xi} \, ,$$

donde $R_S = 2GM/c^2$ es el *radio de Schwarzschild.* De aquí, la ecuación 2 se puede escribir como:

$$\beta = \theta - \alpha_0^2\frac{\theta}{\theta^2} \, ,$$

donde $\alpha_0 \equiv \sqrt{2R_S\frac{D_{ds}}{D_dD_s}}$ se denomina *ángulo característico.* Si se observa una imagen en $\theta = \alpha_0$, la lente y la fuente están alineadas ($\beta = 0$). Debido a que hay simetría respecto al eje óptico, la

imagen observada tiene forma anular, y se denomina comúnmente *anillo de Einstein*. El llamado *radio de Einstein*, $R_E \equiv \sqrt{2R_S \frac{D_d D_{ds}}{D_s}}$, es el equivalente *espacial* de la distancia *angular* α_0. Si la alineación no se da, se producen *dos imágenes* en las posiciones:

$$\theta_{1,2} = \frac{1}{2}\left(\beta \pm \sqrt{4\alpha_0^2 + \beta^2}\right) , \tag{5}$$

con una separación $\Delta\theta = \sqrt{4\alpha_0^2 + \beta^2} \geq 2\alpha_0$.

2.2.2 Amplificación

Si se considera una estrella de *una masa solar* ($R_S \approx 3$km) en el halo galáctico (distancia ≈ 30kpc), esta tiene un tamaño aparente del orden de 10^{-9} segundos de arco. El tamaño del anillo de Einstein correspondiente es del orden de 10^{-5} segundos de arco (la aproximación de Born es válida), por lo que la separación angular de las imágenes es muy pequeña para ser observada. En razón de lo anterior, la detección del efecto de lente gravitacional debido a estrellas debe realizarse observando efectos de amplificación.

Si se definen $\tilde{\boldsymbol{\theta}} \equiv \frac{1}{\alpha_0}\boldsymbol{\theta}$ y $\tilde{\boldsymbol{\beta}} \equiv \frac{1}{\alpha_0}\boldsymbol{\beta}$, la ecuación 2 se escribe como

$$\tilde{\beta} = \tilde{\theta} - \frac{1}{\tilde{\theta}} ,$$

y se tiene

$$\tilde{\theta}_\pm = \frac{1}{2}\left(\tilde{\beta} \pm \sqrt{4 + \tilde{\beta}^2}\right)$$

Si ahora se considera una fuente infinitesimal con un ángulo sólido normalizado $\tilde{\beta}\Delta\tilde{\beta}\Delta\varphi$, la imagen tiene un ángulo sólido normalizado $\tilde{\theta}_\pm \Delta\tilde{\theta}_\pm \Delta\varphi$ (por simetría, $\Delta\varphi$ es el mismo para ambos casos), donde

$$\Delta\tilde{\theta}_\pm = \frac{1}{2}\left(1 \pm \frac{\tilde{\beta}}{\sqrt{\tilde{\beta}^2 + 4}}\right)\Delta\tilde{\beta} .$$

El aumento es:

$$\mu = \left|\frac{\tilde{\theta}\Delta\tilde{\theta}}{\tilde{\beta}\Delta\tilde{\beta}}\right| ,$$

$$\mu_\pm = \frac{1}{4}\left[\left(1 \pm \frac{\tilde{\beta}}{\sqrt{\tilde{\beta}^2 + 4}}\right)\left(1 \pm \frac{\sqrt{\tilde{\beta}^2 + 4}}{\tilde{\beta}}\right)\right] .$$

Y el aumento total es

$$\mu_+ + \mu_- = \frac{\tilde{\beta}^2 + 2}{\tilde{\beta}\sqrt{\tilde{\beta}^2 + 4}} ,$$

cantidad que es siempre mayor que 1. Para el caso donde la fuente se ubica sobre el radio de Einstein, $\tilde{\beta} = 1$, así $\mu = 1,34$. Si la lente y la fuente se *mueven* una respecto a la otra, existirá una *variación temporal* inducida por la lente, la cual puede ser detectable [Got81]. Este tipo de variabilidad se denomina *microlente gravitacional*, y fue observada por primera vez en QSO 2237+0305 [IHC+89].

2.3 Galaxias

El fenómeno de lente gravitacional en masas puntuales se puede resolver en forma directa debido a su simplicidad, pero si se desea estudiar el caso de distribuciones de materia el problema se complica considerablemente. Usualmente se realiza una parametrización del problema, aunque algunos autores [ASW98a] [ASW98b] han realizado reconstrucciones no paramétricas de las distribuciones de materia.

2.3.1 Aproximación de lente delgada

En la figura 2 se ilustra la desviación de un rayo de luz por una masa puntual, la cual ocurre dentro de una región $\Delta x \approx \pm\xi$ alrededor del punto de máximo acercamiento. Δx es, en las situaciones de interés astrofísico, mucho más pequeño que D_d y D_{ds}. Por tanto, la lente puede ser considerada como *delgada* en comparación con la longitud total de la trayectoria rayo de luz. Esto permite *proyectar la distribución de materia de la lente* a lo largo de la línea de visión y reemplazarla con una *hoja de materia* ortogonal a la línea de visión. El plano donde se ubica esta hoja de materia se denomina *plano de la lente*. La hoja de materia se caracteriza por una densidad superficial de materia

$$\Sigma(\boldsymbol{\xi}) = \int \rho(\boldsymbol{\xi}, x) dx , \tag{6}$$

donde $\boldsymbol{\xi}$ es un vector bidimensional en el plano de la lente y $\rho(\boldsymbol{\xi}, x)$ es la densidad volumétrica. El ángulo de desviación en la posición $\boldsymbol{\xi}$ es la suma de las desviaciones debidas a todos los elementos de materia en el plano de la lente

$$\hat{\boldsymbol{\alpha}}(\boldsymbol{\xi}) = \frac{4G}{c^2} \int \frac{\boldsymbol{\xi} - \boldsymbol{\xi}'}{|\boldsymbol{\xi} - \boldsymbol{\xi}'|^2} \Sigma(\boldsymbol{\xi}') d\boldsymbol{\xi}' . \tag{7}$$

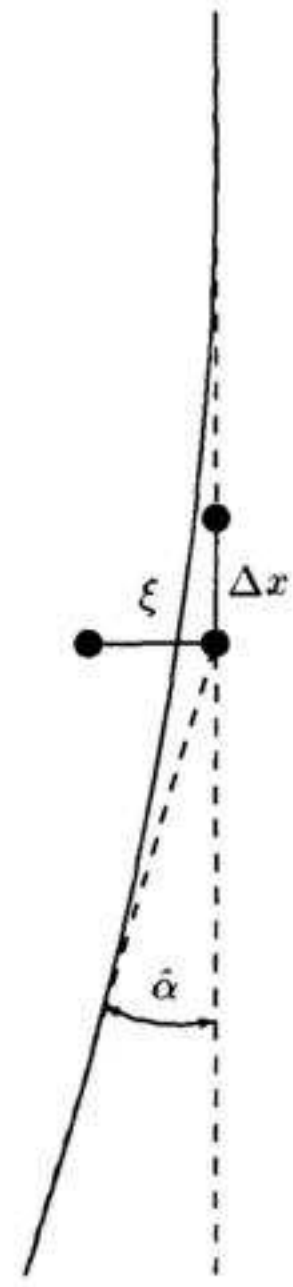

Figura 2: Desviación de la luz por una masa puntual. El rayo tiene un parámetro de impacto ξ y se desvía un ángulo $\hat{\alpha}$. La mayor parte de la desviación ocurre en una región $\Delta x \approx \pm\xi$ alrededor del punto de máximo acercamiento.

donde $\boldsymbol{\xi}$ es un vector bidimensional en el plano de la lente y $\rho(\boldsymbol{\xi}, x)$ es la densidad volumétrica. El ángulo de desviación en la posición $\boldsymbol{\xi}$ es la suma de las desviaciones debidas a todos los elementos de materia en el plano de la lente

$$\hat{\boldsymbol{\alpha}}(\boldsymbol{\xi}) = \frac{4G}{c^2} \int \frac{\boldsymbol{\xi} - \boldsymbol{\xi}'}{|\boldsymbol{\xi} - \boldsymbol{\xi}'|^2} \Sigma(\boldsymbol{\xi}') d\boldsymbol{\xi}' \,. \tag{7}$$

En general, la desviación es un vector bidimensional. Para el caso en el que existe simetría axial en la distribución de materia, 7 se reduce a

$$\hat{\alpha}(\xi) = \frac{4GM(\xi)}{c^2\xi} ,$$

donde ξ es la distancia hasta el centro de la lente y $M(\xi)$ es la masa total encerrada en un círculo de radio ξ

$$M(\xi) = 2\pi \int_0^{\xi} \Sigma(\xi')\xi' d\xi' .$$

De la misma manera puede definirse un potencial gravitacional bidimensional

$$\psi(\boldsymbol{\theta}) = \frac{D_{ds}}{D_d D_s}\frac{2}{c^2}\int \Phi(D_d\boldsymbol{\theta}, z)dz .$$

La derivada con respecto a $\boldsymbol{\theta}$ da el ángulo de desviación

$$\hat{\boldsymbol{\alpha}} = \frac{D_s}{D_{ds}}\nabla_{\boldsymbol{\theta}}\psi = \frac{2}{c^2}\int \nabla_{\boldsymbol{\xi}}\Phi(\boldsymbol{\xi}, z)dz , \tag{8}$$

mientras que el laplaciano se relaciona con la densidad superficial de materia

$$\nabla^2_{\boldsymbol{\theta}}\psi = \frac{2}{c^2}\frac{D_d D_{ds}}{D_s}\int \nabla^2_{\boldsymbol{\xi}}\Phi dz = \frac{8\pi G}{c^2}\frac{D_d D_{ds}}{D_s}\Sigma \equiv 2\kappa(\boldsymbol{\theta}) = 2\frac{\Sigma}{\Sigma_{\mathrm{cr}}} ,$$

donde κ se denomina *convergencia de la lente*, y Σ_{cr} es la densidad superficial crítica a la cual una hoja uniforme de materia focalizaría toda la radiación proveniente de la fuente en el observador [TOG84]. Debido a que se cumple la relación $\nabla^2_{\boldsymbol{\theta}}\psi = 2\kappa$, si κ es una función suave y decrece en infinito más rápido que $1/|\theta|^2$, el potencial efectivo se puede escribir como

$$\psi(\boldsymbol{\theta}) = \frac{1}{\pi}\int \kappa(\boldsymbol{\theta}')\ln|\boldsymbol{\theta} - \boldsymbol{\theta}'|d\boldsymbol{\theta}' ,$$

y el ángulo de desviación

$$\hat{\boldsymbol{\alpha}}(\boldsymbol{\theta}) = \frac{D_s}{D_{ds}}\nabla\psi = \frac{1}{\pi}\frac{D_s}{D_{ds}}\int \kappa(\boldsymbol{\theta})\frac{\boldsymbol{\theta} - \boldsymbol{\theta}'}{|\boldsymbol{\theta} - \boldsymbol{\theta}'|^2}d\boldsymbol{\xi}' .$$

Las propiedades de la aplicación $\boldsymbol{\theta} \mapsto \boldsymbol{\beta}$ de la lente están descritas por la matriz jacobiana

$$\mathcal{A} \equiv \frac{\partial\boldsymbol{\beta}}{\partial\boldsymbol{\theta}} = \left(\delta_{ij} - \frac{\partial\alpha_i(\boldsymbol{\theta})}{\partial\theta_j}\right) = \left(\delta_{ij} - \frac{\partial^2\psi(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\right) . \tag{9}$$

En particular, la amplificación μ está dada por

$$\mu = |\det\mathcal{A}|^{-1} . \tag{10}$$

Las regiones en el plano de la lente en las cuales el determinante de la matriz $\mathcal{A}$ tiene signos opuestos, están separadas por curvas en las cuales este determinante se anula, y son llamadas *curvas críticas*. De 10 se ve que la magnificación diverge en estas curvas. Debido a que la magnificación total es la media sobre toda la fuente, la magnificación total no diverge, es decir, no se forman imágenes infinitamente brillantes. Para el caso de fuentes puntuales, el problema debe ser tratado utilizando óptica ondulatoria, de manera que se tomen en cuenta efectos de difracción e interferencia. Al hacer esto, se encuentra que la magnificación tampoco diverge en dicho caso [SEF92].

Al transportar las curvas críticas al plano de la lente se obtienen las *cáusticas*. Cualquier punto de la fuente que se ubique sobre una de estas experimentará una magnificación (formalmente) infinita. Más importante es el hecho que el número de imágenes cambia en dos cuando la fuente cruza una caústica.

La matriz hessiana de ψ, denotada por conveniencia como

$$\psi_{ij} \equiv \frac{\partial^2 \psi}{\partial\theta_i \partial\theta_j} ,$$

describe cuánto se desvía de la identidad la aplicación de la lente. La convergencia se escribe como

$$\kappa = \frac{1}{2}(\psi_{11} + \psi_{22}) = \frac{1}{2}\,\mathrm{tr}\,\psi_{ij} . \tag{11}$$

Además, se define el *tensor de cizalla* como

$$\begin{aligned} \gamma_1(\boldsymbol{\theta}) &= \frac{1}{2}(\psi_{11} - \psi_{22}) \equiv \gamma(\boldsymbol{\theta}) \cos\left[2\phi(\boldsymbol{\theta})\right] \\ \gamma_2(\boldsymbol{\theta}) &= \psi_{12} = \psi_{21} \equiv \gamma(\boldsymbol{\theta}) \sin\left[2\phi(\boldsymbol{\theta})\right] . \end{aligned} \tag{12}$$

De esta forma la matriz jacobiana se puede escribir como

$$\begin{aligned} \mathcal{A} &= \begin{pmatrix} 1-\kappa-\gamma_1 & -\gamma_2 \\ -\gamma_2 & 1-\kappa+\gamma_1 \end{pmatrix} \\ &= (1-\kappa)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \gamma \begin{pmatrix} \cos 2\phi & \sin 2\phi \\ \sin 2\phi & -\cos 2\phi \end{pmatrix} . \end{aligned}$$

Así, se ve como la convergencia induce un *enfoque convergente* de los rayos de luz, produciendo un *aumento isotrópico de la fuente*. La cizalla introduce *anisotropía* (astigmatismo) en la aplicación: $\gamma = \sqrt{\gamma_1^2 + \gamma_2^2}$ describe la magnitud de la cizalla; ϕ describe su orientación. Siguiendo 10, la amplificación es:

$$\mu = \frac{1}{(1-\kappa)^2 - \gamma^2} .$$

2.3.2 Principio de Fermat

Las propiedades de los modelos de lente gravitacional son fácilmente visualizables mediante la aplicación del principio de Fermat [BN86]. Si en la figura 1 se consideran todas las trayectorias a partir de una fuente S hasta un observador O, entonces para cada una de ellas se tiene una integral de línea la cual mide el tiempo que toma al rayo llegar a O a partir de S. El principio de Fermat establece que este tiempo es extremo cuando la trayectoria considerada corresponde a una trayectoria real. Para la derivación del principio de Fermat en el contexto de lentes gravitacionales, se supone la existencia de distancias angulares que relacionan distancias propias en la fuente con el ángulo que subtienden en el observador. Lo anterior es cierto en modelos cosmológicos homogéneos. Adicionalmente, es necesario que el potencial sea estacionario durante el tiempo que le toma al rayo de luz cruzarlo.

De 2 y 8 se tiene

$$(\boldsymbol{\theta} - \boldsymbol{\beta}) - \nabla_{\boldsymbol{\theta}} \psi = 0 \,,$$

que puede ser escrita como

$$\nabla_{\boldsymbol{\theta}} \left[\frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\beta})^2 - \psi(\boldsymbol{\theta}) \right] = 0 \,. \tag{13}$$

La función entre corchetes describe el *atraso temporal total* para el rayo yendo de S a O. En unidades no reducidas, esta función se expresa como

$$\tau(\boldsymbol{\theta}, \boldsymbol{\beta}) = \frac{1 + z_d}{c} \frac{D_d D_s}{D_{ds}} \left[\frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\beta})^2 - \psi(\boldsymbol{\theta}) \right] = t_{\mathrm{geom}} + t_{\mathrm{grav}} \,, \tag{14}$$

donde $(1+z_d)/c$ garantiza que el tiempo corresponde al tiempo medido por un observador en O. La contribución geométrica (t_{geom}) es debida a la *distancia extra* que el rayo de luz debe viajar debido a la desviación respecto a la trayectoria no perturbada. La segunda contribución (t_{grav}), el atraso temporal gravitacional local, se debe a la *presencia de la materia* que ocasiona la desviación, fenómeno que ha sido estudiado en la vecindad del sistema solar mediante mediciones de radar [Sha64]. De esta forma, 13 es la expresión del *principio de Fermat* para lentes gravitacionales.

2.4 Cúmulos galácticos y estructura a gran escala como lentes gravitacionales

En cúmulos galácticos se tienen dos tipos distintos de fenómeno de lente gravitacional

1. Se producen *arcos gigantes* cuando una galaxia de fondo se alinea con las cáusticas de cúmulos ricos con una población condensada centralmente. Este fenómeno se puede estudiar con modelos paramétricos.

2. Todos los cúmulos producen (grandes cantidades de) imágenes débilmente distorsionadas de galaxias de fondo. Las imágenes se llaman *arclets* y el fenómeno como tal *lentes gravitacionales débiles*. De este tipo de lentes se pueden derivar las distribuciones de materia en el cúmulo mediante modelos no paramétricos.

2.4.1 Lentes gravitacionales fuertes en cúmulos

Para el caso de lentes gravitacionales fuertes, donde se observan múltiples imágenes, se requiere que la densidad de masa superficial en *algún lugar de la lente* sea mayor que la densidad de masa superficial crítica de forma que se produzca una o más cáusticas . En el caso en que la galaxia de fondo se ubique sobre el punto de quiebre de una cáustica, se producen arcos gigantes pues se mezclan *tres* imágenes de la misma fuente en una pequeña región. Si la galaxia se ubica sobre los así llamados *labios* o en cáusticas de *cúspide a cúspide* también se forman arcos grandes. Por otra parte, si la galaxia se ubica sobre una región de pliegue se forman dos imágenes en lugar de tres, por tanto los arcos son más pequeños.

La ubicación del arco respecto al cúmulo permite dar un estimado de la *masa proyectada* contenida dentro del círculo que define el arco . En el caso de una lente con simetría circular, el promedio de la densidad superficial de masa *dentro* de la curva crítica tangente es igual a la densidad crítica de masa $\Sigma_{\rm cr}$. Así, el radio $\theta_{\rm arco}$ del círculo trazado por el arco da un estimado de $\theta_{\rm E}$, el radio de Einstein del cúmulo. Por tanto, se tiene la relación

$$\langle \Sigma(\theta_{\rm arco}) \rangle \approx \langle \Sigma(\theta_{\rm E}) \rangle = \Sigma_{\rm cr} \ ,$$

y se tiene una masa total encerrada de

$$M(\theta) = \Sigma_{\rm cr} \pi (D_d \theta)^2 \ .$$

Tomando un modelo isotérmico para la distribución de materia, se puede dar un estimado para la velocidad de dispersión del cúmulo.

Los estimados anteriores se basan en suposiciones simples respecto a la distribución de materia en el cúmulo. Se pueden mejorar estos estimados utilizando modelos paramétricos y realizando ajustes a las observaciones.

2.4.2 Lentes gravitacionales débiles

Si la fuente no se ubica en las cercanías de un punto de quiebre de las cáusticas, el efecto de lente gravitacional se ve reducido y el resultado son los llamados *arclets*. La población de galaxias azules distantes alcanza densidades espaciales de 50 a 100 galaxias por minuto de arco cuadrado

para magnitudes débiles [Tys88], así, cada cúmulo tiene del orden de 50 a 100 *arclets* en la misma área, los cuales han sido observados [FPM+88]

Kaiser y Squires [KS93], motivados por Tyson, Valdes y Wenk [TWV90], desarrollaron un método sistemático no paramétrico para convertir las elipticidades observadas de los *arclets* en un mapa de densidad superficial de masa $\Sigma(\boldsymbol{\theta})$ del cúmulo que actúa como lente. Posteriormente Seitz y Schneider [SS95b] refinaron y generalizaron este método. El método se basa en que la convergencia (11) y la cizalla (12) son combinaciones lineales de las segundas derivadas del potencial efectivo $\psi(\boldsymbol{\theta})$, por tanto existe una relación matemática que conecta a ambas. Se utiliza la elipticidad de las galaxias de fondo para dar un estimado del valor de $\gamma_1(\boldsymbol{\theta})$ y $\gamma_2(\boldsymbol{\theta})$, de donde se obtiene un valor para $\kappa(\boldsymbol{\theta})$, de lo cual se infiere que $\Sigma(\boldsymbol{\theta}) = \Sigma_{cr}\kappa(\boldsymbol{\theta})$.

Si se toma la transformada de Fourier en 11 y 12 se tiene

$$\begin{aligned}
\tilde{\kappa}(\boldsymbol{k}) &= -\frac{1}{2}(k_1^2 + k_2^2)\tilde{\psi}(\boldsymbol{k}) \\
\tilde{\gamma}_1(\boldsymbol{k}) &= -\frac{1}{2}(k_1^2 - k_2^2)\tilde{\psi}(\boldsymbol{k}) \\
\tilde{\gamma}_2(\boldsymbol{k}) &= -\frac{1}{2}k_1 k_2 \tilde{\psi}(\boldsymbol{k}) \,,
\end{aligned}$$

donde $\boldsymbol{k}$ es el vector de onda bidimensional conjugado de $\boldsymbol{\theta}$. En espacio de Fourier se tiene

$$\begin{pmatrix} \hat{\gamma}_1 \\ \hat{\gamma}_2 \end{pmatrix} = \frac{1}{k^2}\begin{pmatrix} k_1^2 - k_2^2 \\ 2k_1 k_2 \end{pmatrix}\hat{\kappa} \,,$$

que puede ser resuelto para $\hat{\kappa}(\boldsymbol{k})$, que se transforma al espacio real para obtener la densidad superficial de materia.

Combinando los dos componentes de la cizalla en un solo número complejo $\gamma = \gamma_1 + \mathrm{i}\gamma_2$, esta cantidad se puede expresar como

$$\gamma(\theta) = \frac{1}{\pi}\int \mathcal{D}(\boldsymbol{\theta} - \boldsymbol{\theta}')\kappa(\boldsymbol{\theta}')d\boldsymbol{\theta} \,, \tag{15}$$

donde

$$\mathcal{D} = \frac{\theta_1^2 - \theta_2^2 + 2\mathrm{i}\theta_1\theta_2}{|\boldsymbol{\theta}|^4} \,,$$

invirtiendo 15

$$\kappa(\theta) = \frac{1}{\pi}\int \mathrm{Re}[\mathcal{D}(\boldsymbol{\theta} - \boldsymbol{\theta}')\gamma^*(\boldsymbol{\theta}')]d\boldsymbol{\theta}' \,.$$

Seitz y Schneider demostraron [SS95a] que no es posible medir *directamente* la cizalla a partir de observaciones locales de distorsión en las imágenes, y en su lugar utilizan otra cantidad, la distorsión compleja

$$\delta = \frac{2\gamma(1-\kappa)}{(1-\kappa)^2 + |\gamma|^2} \,,$$

que es una combinación de la convergencia y la cizalla.

3. Visualización

El problema a visualizar lo define la ecuación 2: encontrar la posición $\boldsymbol{\theta}$ de todas las imágenes para un posición dada $\boldsymbol{\beta}$ de la fuente, en otras palabras, invertir la ecuación. Si bien algunos casos particulares se pueden resolver analíticamente, como en 5, en general esto no es posible pues se deben encontrar todas las raíces de un sistema bidimensional de ecuaciones (no lineales) donde no se sabe *a priori* cual es el número de imágenes que se tiene para una posición dada de la fuente.

Si se consideran algunos casos simples de lentes gravitacionales, como aquellos con simetría axial, se pueden entender mejor las características del problema general. Los casos con simetría axial, son aquellos en las que la distribución de materia tiene una densidad de masa volumétrica que es invariante con respecto a rotaciones alrededor del *eje óptico*. La densidad superficial de materia correspondiente es también invariante ante rotaciones con respecto al centro de masa. Si bien es cierto que los modelos con simetría axial *no tienen* un gran número de aplicaciones astrofísicas, estos son utilizados con frecuencia para dar una primera aproximación a casos observados o como componentes elementales en aplicaciones estadísticas [BMM99], [RH94], [Fug89].

3.1 Planteamiento general

En el caso general, si se consideran 3 y 7, se tiene una aplicación completa $\boldsymbol{\xi} \mapsto \boldsymbol{\eta}$. Si se consideran las cantidades sin unidades [SEF92]

$$\boldsymbol{x} = \frac{\boldsymbol{\xi}}{\xi_0} \; ; \; \boldsymbol{y} = \frac{\boldsymbol{\eta}}{\eta_0} \, , \tag{16}$$

donde ξ_0 es una escala arbitraria y $\eta_0 = \xi_0 D_s / D_d$, la ecuación de lentes se puede escribir como

$$\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{\alpha}(\boldsymbol{x}) \, , \tag{17}$$

donde

$$\boldsymbol{\alpha}(\boldsymbol{x}) = \frac{1}{\pi} \int \frac{\boldsymbol{x} - \boldsymbol{x}'}{|\boldsymbol{x} - \boldsymbol{x}'|^2} \kappa(\boldsymbol{x}') d\boldsymbol{x}' \, , \tag{18}$$

es el ángulo de deflexión reducido. Este ángulo se puede expresar como $\boldsymbol{\alpha} = \nabla \psi$, donde

$$\psi(\boldsymbol{x}) = \frac{1}{\pi} \int \kappa(\boldsymbol{x}') \ln |\boldsymbol{x} - \boldsymbol{x}'| d\boldsymbol{x}' \, .$$

Además, la función

$$\phi(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{y})^2 - \psi(\boldsymbol{x}) \, , \tag{19}$$

$$(1 - \kappa)^2 + |\gamma|^2 \, ,$$

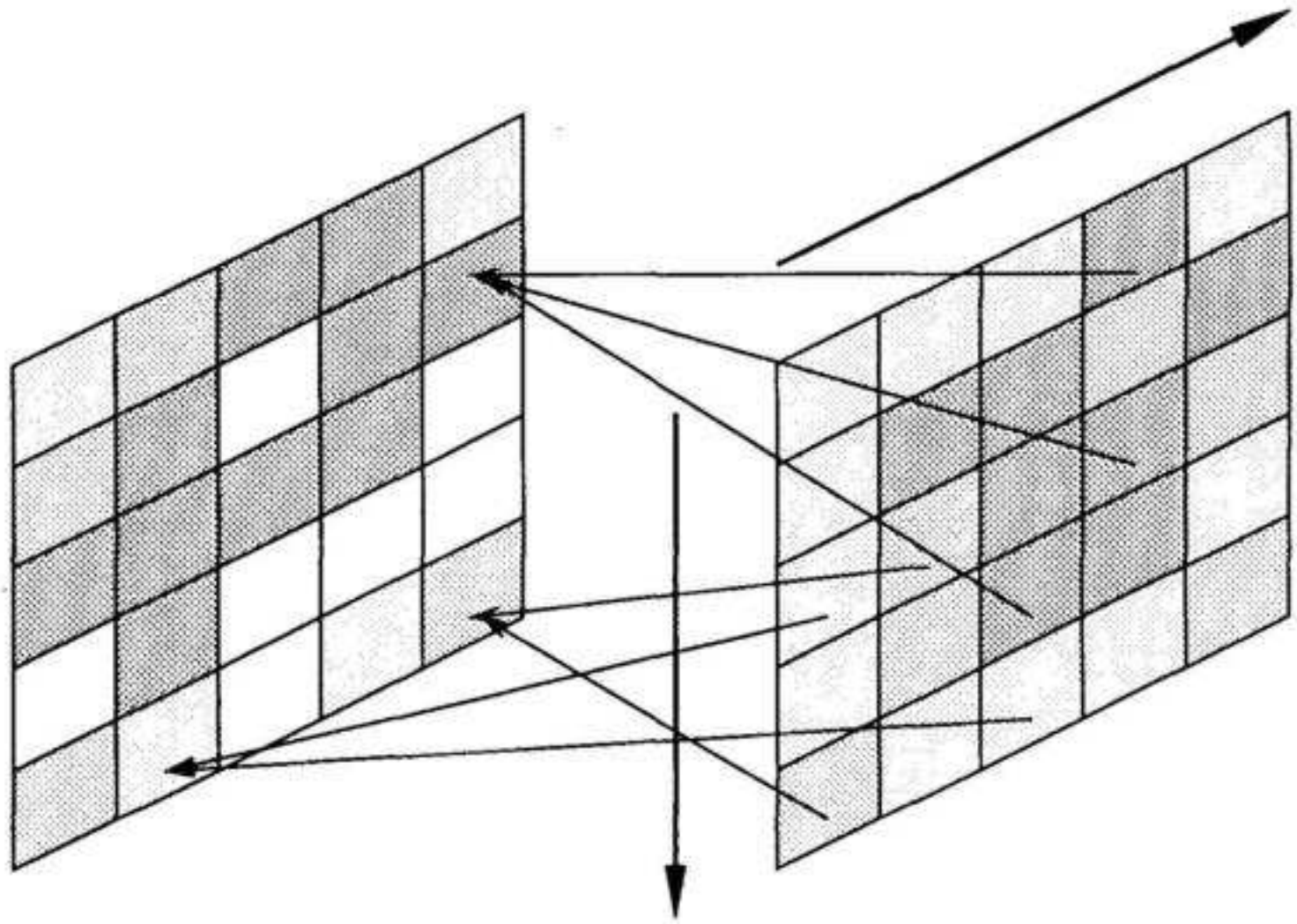

Figura 3: Construcción de imágenes a partir de la ecuación de lentes.

está relacionada con el atraso temporal tal como en 14.

La matriz jacobiana se expresa como

$$\mathcal{A}(\boldsymbol{x}) = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \ , \ A_{ij} = \frac{\partial y_i}{\partial x_j} \ , \tag{20}$$

y la magnificación es entonces

$$\mu = \left(\frac{\partial y_1}{\partial x_1} \frac{\partial y_2}{\partial x_2} - \frac{\partial y_1}{\partial x_2} \frac{\partial y_2}{\partial x_1} \right)^{-1} . \tag{21}$$

3.2 Construcción de imágenes

La ecuación 17 provee un método directo para la construcción de las imágenes producidas por una lente si se cuenta con el ángulo de deflexión 18. Recordando que en 17, $\boldsymbol{x}$ es la posición en el plano de la lente y $\boldsymbol{y}$ es la posición en el plano de la fuente, la construcción de las imágenes

se puede realizar como se indica en la figura 3: se recorre todo el plano de la lente, calculando la posición correspondiente en el plano de la fuente mediante 17. Es necesario hacer hincapié en que este cálculo se puede hacer en dos pasos: primero 18 para calcular el lado derecho de 17, y luego asociar la posición calculada $\boldsymbol{y}$ con un punto de la imagen particular con la que se está trabajando.

3.2.1 Matriz jacobiana

En diferencias finitas hacia adelante la matriz jacobiana 20 se expresa como

$$\mathcal{A}_{ij} = \begin{pmatrix} \frac{y_1^{i+1,j} - y_1^{i,j}}{h_1} & \frac{y_1^{i,j+1} - y_1^{i,j}}{h_1} \\ \frac{y_2^{i,j+1} - y_2^{i,j}}{h_2} & \frac{y_2^{i+1,j} - y_2^{i,j}}{h_2} \end{pmatrix}, \tag{22}$$

donde $y_n^{i,j} \equiv y_n(x_1^i, x_2^j)$, $x_n^i \equiv x_n^0 + ih_n$. Otra vez, es de notar que esto no depende de la posición de la fuente, sino únicamente de las propiedades de la aplicación $\boldsymbol{x} \mapsto \boldsymbol{y}$.

El determinante de la matriz $\mathcal{A}$ es:

$$\det \mathcal{A} = \frac{(y_1^{i+1,j} - y_1^{i,j})(y_2^{i,j+1} - y_2^{i,j}) - (y_1^{i,j+1} - y_1^{i,j})(y_2^{i+1,j} - y_2^{i,j})}{h_1 h_2}. \tag{23}$$

3.2.2 Magnificación

En diferencias finitas hacia adelante la magnificación 21 se expresa como

$$\mu = \frac{h_1 h_2}{(y_1^{i+1,j} - y_1^{i,j})(y_2^{i,j+1} - y_2^{i,j}) - (y_1^{i,j+1} - y_1^{i,j})(y_2^{i+1,j} - y_2^{i,j})}. \tag{24}$$

La función obtenida de esta forma *no es* la magnificación total de la imagen respecto a la fuente, sino la magnificación infinitesimal para *cada* punto de la lente.

3.2.3 Retraso temporal

La ecuación 19 está relacionada con el retraso temporal en cada punto de la lente. Esto, superpuesto con las imágenes de la fuente, permite explorar la distribución de masa en la lente.

3.2.4 Curvas críticas

Tal como se señaló en , a partir de la matriz jacobiana se pueden definir las curvas críticas como aquellas donde se cumple $\det \mathcal{A} = 0$. Para conseguir una representación gráfica de las curvas críticas se puede considerar la función $f(\boldsymbol{x}) = \det \mathcal{A}$ y trazar los contornos correspondientes a $f(\boldsymbol{x}) = 0$ utilizando una rutina estándar.

3.2.5 Curvas caústicas

El mismo método empleado en la sección anterior para las curvas críticas, se puede utilizar para obtener una representación de las curvas caústicas, con la salvedad de que se deben encontrar las regiones $\det \mathcal{A} = 0$ en el *plano de la lente* y utilizar la aplicación $\boldsymbol{x} \mapsto \boldsymbol{y}$ para encontrar las correspondientes curvas en el *plano de la fuente.*

3.3 Aplicación

En la figura 4 se muestra la interfaz completa de la aplicación para la visualización de lentes gravitacionales. Comenzando en la esquina superior izquierda, en sentido horario: la ventana principal de control, donde se puede ver ; el control de parámetros; la generación de imágenes, donde se muestran también las curvas isócronas; la ventana de magnificación, donde el blanco indica $\mu = 1$, el azul $\mu = 0$ y el rojo $\mu \to \infty$.

La aplicación está diseñada con el objetivo de permitir al usuario variar interactivamente los parámetros que definen cada modelo. Debido a esto, se prefieren las técnicas que permiten obtener un menor tiempo de respuesta sobre aquellas que minimizan el consumo de recursos en el sistema. En la figura 5 se muestra el diagrama de flujo para el cálculo de los datos.

Se han dividido los parámetros con los que opera el algoritmo en dos: aquellos que afectan directamente la imagen generada (v.g., posición y tamaño de la fuente) y aquellos que sólo afectan la aplicación $\boldsymbol{x} \mapsto \boldsymbol{y}$. Todos los parámetros de este último tipo se encuentran agrupados en una sola ventana como se muestra en la figura 6, desde la cual se puede seleccionar el modelo a utilizar, así como variar los parámetros de escala, convergencia y cizalla, además de aquellos que son específicos para cada modelo. Cuando se modifica cualquiera de estos parámetros, se altera el valor de la marca **needs_remake** dentro de la estructura **LensMap**, de tipo **map** que está definido como:

```
typedef struct {
        int w, h;
        int step;
        int needs_remake;
```

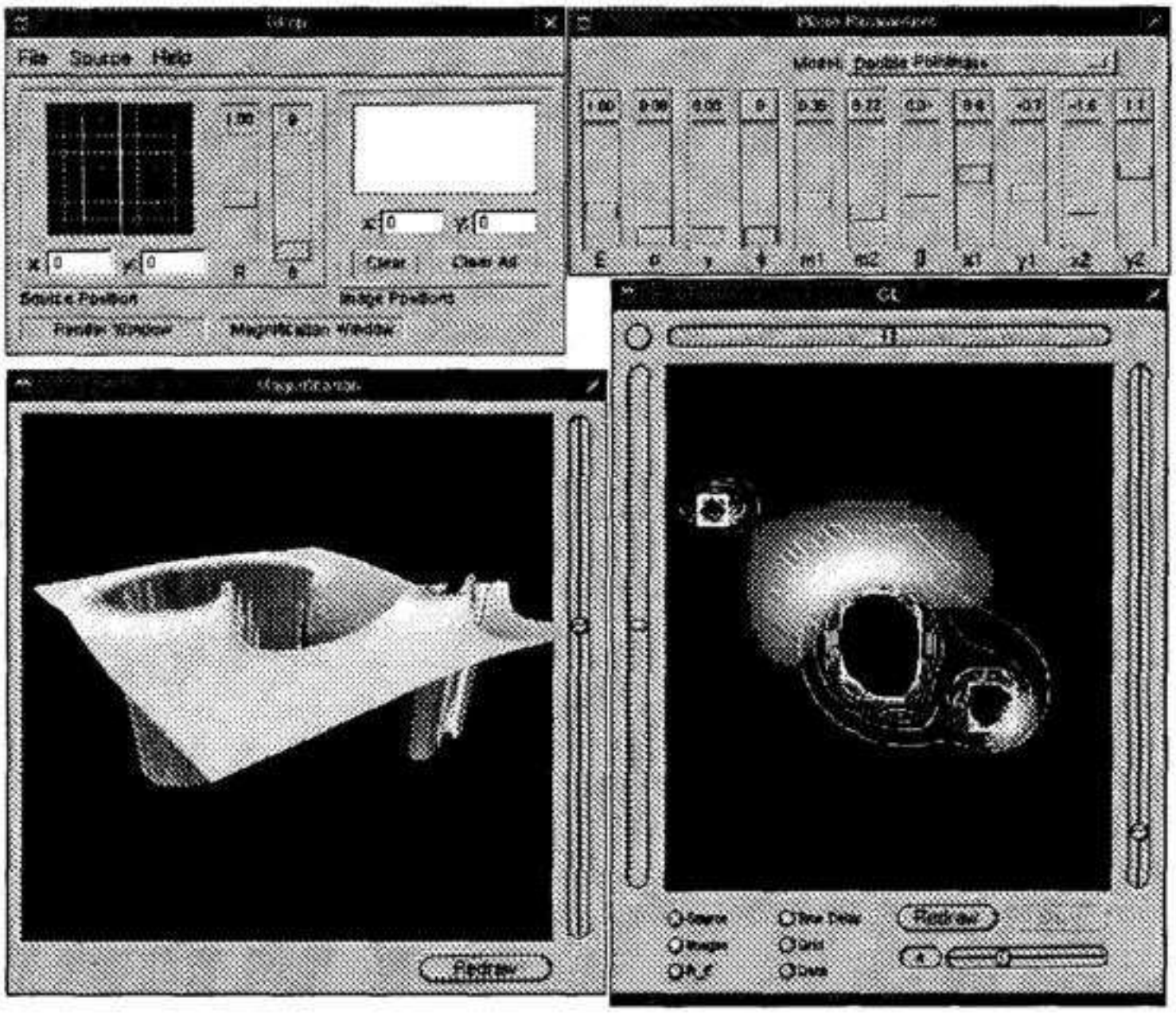

Figura 4: Aplicación para la visualización de lentes gravitacionales. Comenzando en la esquina superior izquierda, en sentido horario: ventana de control; parámetros; imágenes; magnificación

ventana como se muestra en la figura 6, desde la cual se puede seleccionar el modelo a utilizar, así como variar los parámetros de escala, convergencia y cizalla, además de aquellos que son específicos para cada modelo. Cuando se modifica cualquiera de estos parámetros, se altera el valor de la marca **needs_remake** dentro de la estructura **LensMap**, de tipo **map** que está definido como:

```
typedef struct {
        int w, h;
        int step;
        int needs_remake;
```

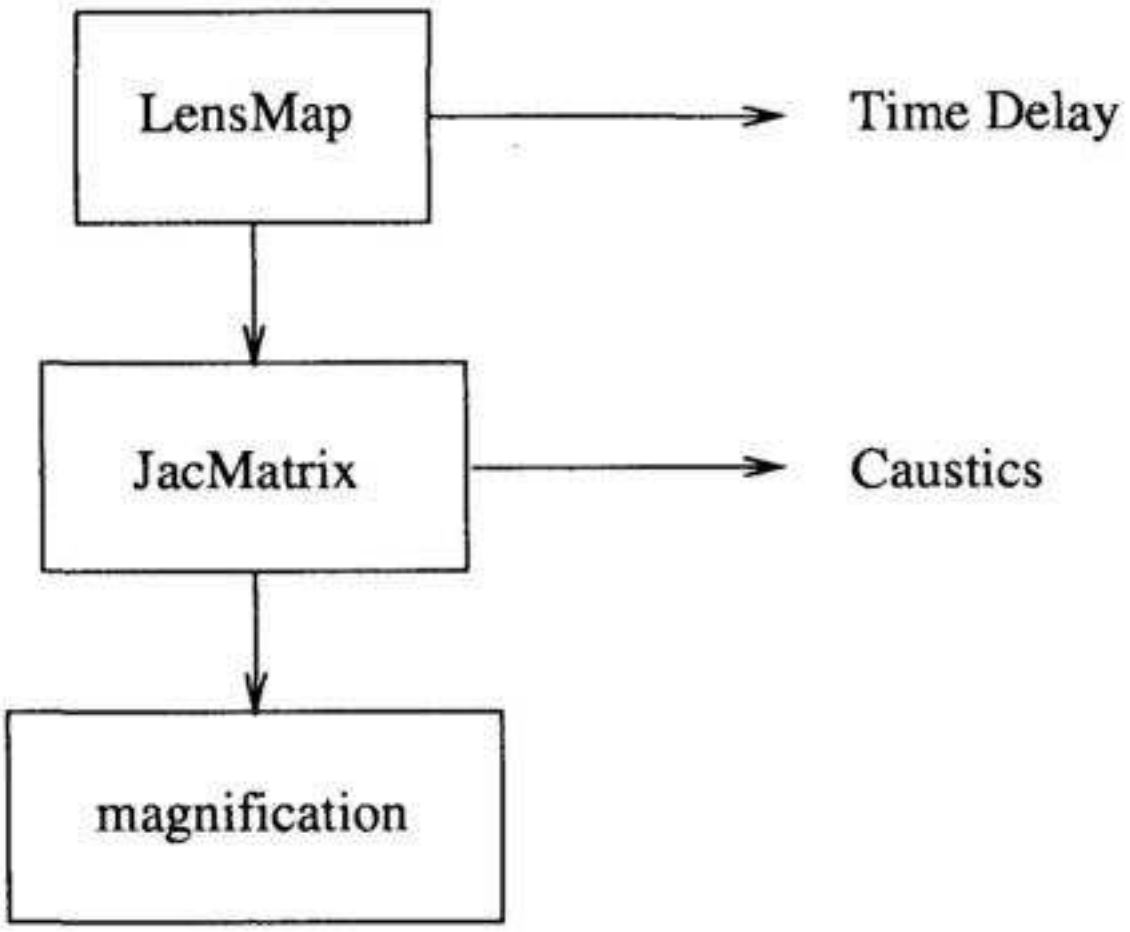

Figura 5: Diagrama de flujo para el cálculo de los datos a partir de los cuales se producen las imágenes. Los nombres corresponden a los nombres de las estructuras internas utilizadas.

```
        struct {
                double x1, x2;
                double y1, y2;
        } ** Map;
} map;
```

`w` y `h` corresponden al *ancho* y *alto* de la imagen con la que se desea trabajar. `step` es el intervalo en el que se calculan las imágenes; para `step=1`, se calculan todos los pixeles de la imagen, para `step=2`, se calculan uno de por medio, y así sucesivamente. El número de cálculos que hay que hacer para obtener una imagen es $w \times h/step^2$ (v.g. si se trabaja con `step=4` hay que hacer 16 veces *menos* cálculos que trabajando con `step=1`). En la estructura también se guardan los valores de x y y en un arreglo bidimensional. El objetivo de guardar x es evitar en algunas circunstancias[4] la realización operaciones de punto flotante, además de servir para la contención de errores derivados de operaciones de la forma a/b, donde ambos a y b son números enteros.

Para la matriz jacobiana existe una estructura similar, `JacMatrix`, de tipo `jacobian_matrix`:

```
typedef struct {
        int w, h;
        int step;
        int needs_remake;
```

[4]En particular, se evitan todos cálculos de la forma $x_i = x_0 + i \times \Delta x$.

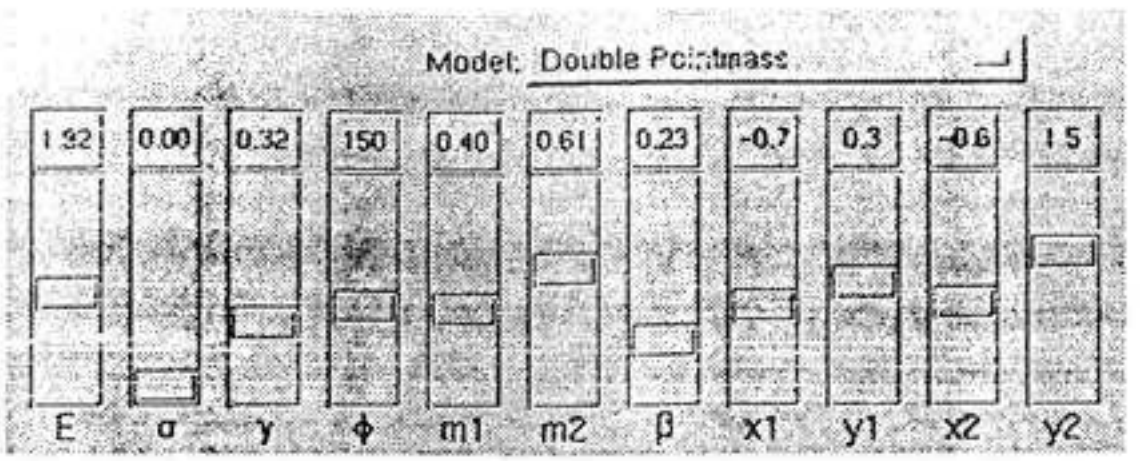

Figura 6: Ventana para modificación interactiva de parámetros de la lente.

```
        double ** A;
} jacobian_matrix;
```

Los primeros cuatro miembros de la estructura tienen el mismo significado que en `map`. `jacobian_matrix` corresponde a 23; la marca `needs_remake` se coloca cada vez que se calcula `LensMap`.

Las imágenes son construidas utilizando *puntos* (en el sentido de OpenGL, [KF96], p. 68), y son almacenadas como una *lista* ([KF96], p. 267). Lo anterior quiere decir que siempre que es posible, se elimina el proceso de conversión de coordenadas de mundo a coordenadas en el *frame buffer*. En general, cada imagen se crea según:

```
¿ha cambiando el campo vectorial o escalar correspondiente? {
        algunos cálculos previos;

        if (!(glIsList (lista))) // ¿existe la lista?
                lista = glGenLists (1);

        glNewList (lista, GL_COMPILE_AND_EXECUTE);
        instrucciones para crear la imagen;
        glEndList ();
} else
        glCallList (lista);
```

[4] En particular, se evitan todos cálculos de la forma $x_i = x_0 + i \times \Delta x$.

4. Resultados

Con la aplicación desarrollada es posible visualizar diferentes características de lentes gravitacionales mediante un esquema paramétrico. El diseño es tal que *añadir* modelos es una tarea simple. Siguiendo a [Fru98], se han introducido los siguientes modelos:

- *Chang-Refsdal*
- *Esfera isotérmica singular*
- *Esfera isotérmica no singular*
- *Esfera transparente*
- *Elíptico*
- *King*
- *King truncado*
- *Hubble*
- *De Vacouleur*
- *Espiral*
- *Multipolar*
- *Rotacional*
- *Masa puntual doble*

4.1 Imágenes estáticas

En la figura 7 se muestra la aplicación de un modelo de *Chang-Refsdal* para dos posiciones diferentes de la fuente. Se pueden observar las curvas críticas (curva exterior, color naranja en el gráfico original) y las curvas caústicas (curva interior, color celeste en el gráfico original). Además se indica la posición de la fuente mediante círculos concéntricos, donde se utiliza una gradación tonal para diferenciar las distintas partes: más intenso hacia el interior, más tenue hacia el exterior. Así es posible *ver* como se modifican diferentes regiones de la fuente bajo un modelo particular. En la figura de arriba, a la derecha, ha ocurrido la combinación de tres imágenes en un solo un arco extendido. Las tres imágenes se pueden apreciar en la figura inferior izquierda.

En la figura 8 se muestra un modelo de masa puntual doble. Se encuentran representadas las curvas críticas (suaves) y caústicas (con picos). Además se presentan las curvas isócronas.

En la figura 9 se muestra el resultado de aplicar un modelo de masa puntual sobre una placa de M45. Se observa la distorsión sufrida por las imágenes de las estrellas cercanas al eje óptico, así como múltiples imágenes de algunos de los componentes. Se ve además que componentes más pequeños exhiben elongación transversal.

En la figura 10 se presenta un campo de "galaxias" (izquierda) y la correspondiente imagen (derecha). Para producir esta imagen se ha utilizado un modelo de Chang–Refsdal con contribuciones externas de cizalla y convergencia (§). Los múltiples arcos tienen su origen en *varias* de las fuentes; no es posible lograr este tipo de estructura utilizando una sola fuente extendida. La asimetría en la localización de los centros de curvatura de estos arcos se logra introduciendo contribuciones externas. Se observa además que si bien no hay fuentes elípticas, varias de las imágenes exhiben elipticidad. Esta figura reproduce las *características generales* de los sistemas de arcos gigantes como Abell 370 y Abell 2218.

4.2 Animaciones

Además de las imágenes estáticas, el programa desarrollado puede ser utilizado para la generación de animaciones. Utilizando el *script* `anim.pl`, cuyo listado se encuentra en el apartado se pueden generar secuencias de imágenes en las cuales se puede variar un número arbitrario de parámetros. Por ejemplo, variando la posición de la fuente se pueden reproducir un escenario similar al de eventos de microlente. Por otro lado, variando alguno de los parámetros (v.g., cizalla, elipticidad, radio del núcleo, ...), se puede observar su efecto sobre el desarrollo de curvas críticas o la magnificación de las imágenes. Se ha producido un vídeo, disponible en VHS o en formato digital como MPEG, donde se pueden apreciar algunas de estas animaciones.

5. Conclusiones y trabajo futuro

El programa desarrollado tiene un valor didáctico, como herramienta para demostrar algunos efectos (sorprendentes) derivados de la Teoría de la Relatividad General. Además tiene valor como herramienta de trabajo, ya que en conjunto con otros programas puede ser utilizado para entender la dinámica asociada con el fenómeno de lentes gravitacionales. Debido a la forma en la cual está escrito, la tarea de modificar los modelos o añadir otros nuevos es muy simple, permitiendo al investigador concentrarse en las características físicas del problema y no los problemas computacionales. El programa desarrollado sirve además otro propósito, a saber, constituye una base para la creación de aplicaciones interactivas para el estudio de fenómenos físicos, ya sea en el campo de magnetohidrodinámica, relatividad general, teoría cuántica u otros.

Al desarrollar esta aplicación se buscó que fuese rápida, pues no es importante solamente que el investigador pueda *ver* el efecto que un parámetro tiene sobre el sistema completo, sino también la forma en la que el sistema responde a la *modificación* de dicho parámetro. Sin utilizar ningún tipo de aceleración gráfica en *hardware*, el programa permite variar interactivamente los parámetros y apreciar en forma casi instantánea las modificaciones en las imágenes, las curvas de retraso temporal, las curvas críticas y las cáusticas. Si se aumenta la resolución del cálculo, se pierde esta capacidad pues el número de operaciones que se deben realizar aumenta en forma cuadrática, pero aún en las peores condiciones las imágenes se generan en aproximadamente un segundo (tiempo

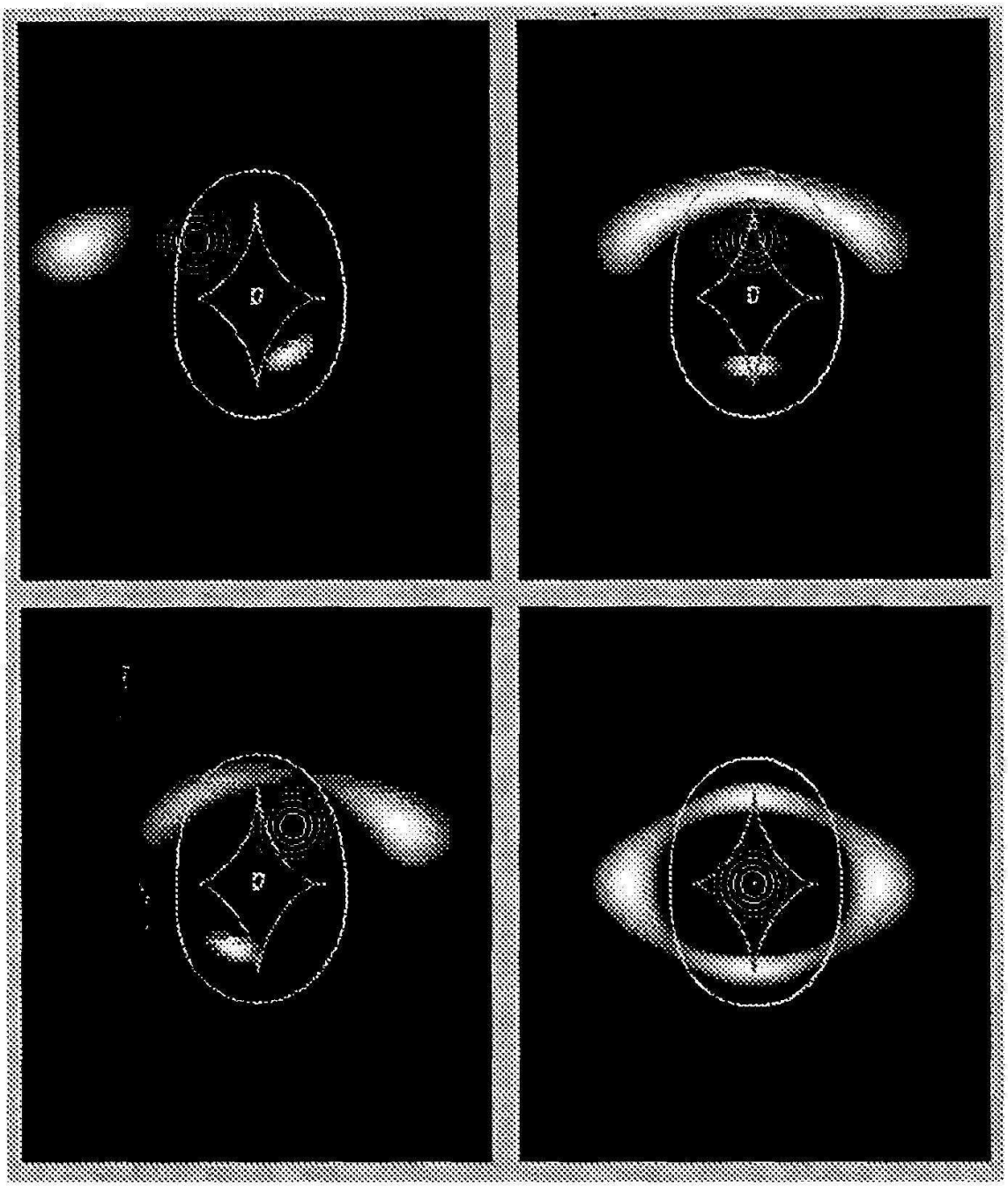

Figura 7: Imágenes producidas por un modelo de *Chang-Refsdal.* Se ha utilizado una escala $E = 0,34$, convergencia $\kappa = 0,85$ y cizalla $\gamma = 0,36$. Arriba a la izquierda, la fuente está en $(-0,5;\ 0,5)$. Arriba a la derecha, la fuente está en $(0;\ 0,5)$. Abajo a la izquierda, $(0,3;\ 0,5)$. Abajo a la derecha, la fuente se ha ubicado en $(0,0;\ 0,0)$, y se puede apreciar el efecto de la cizalla. Se muestra la imagen distorsionada de la fuente así como la posición de la misma y las curvas caústicas.

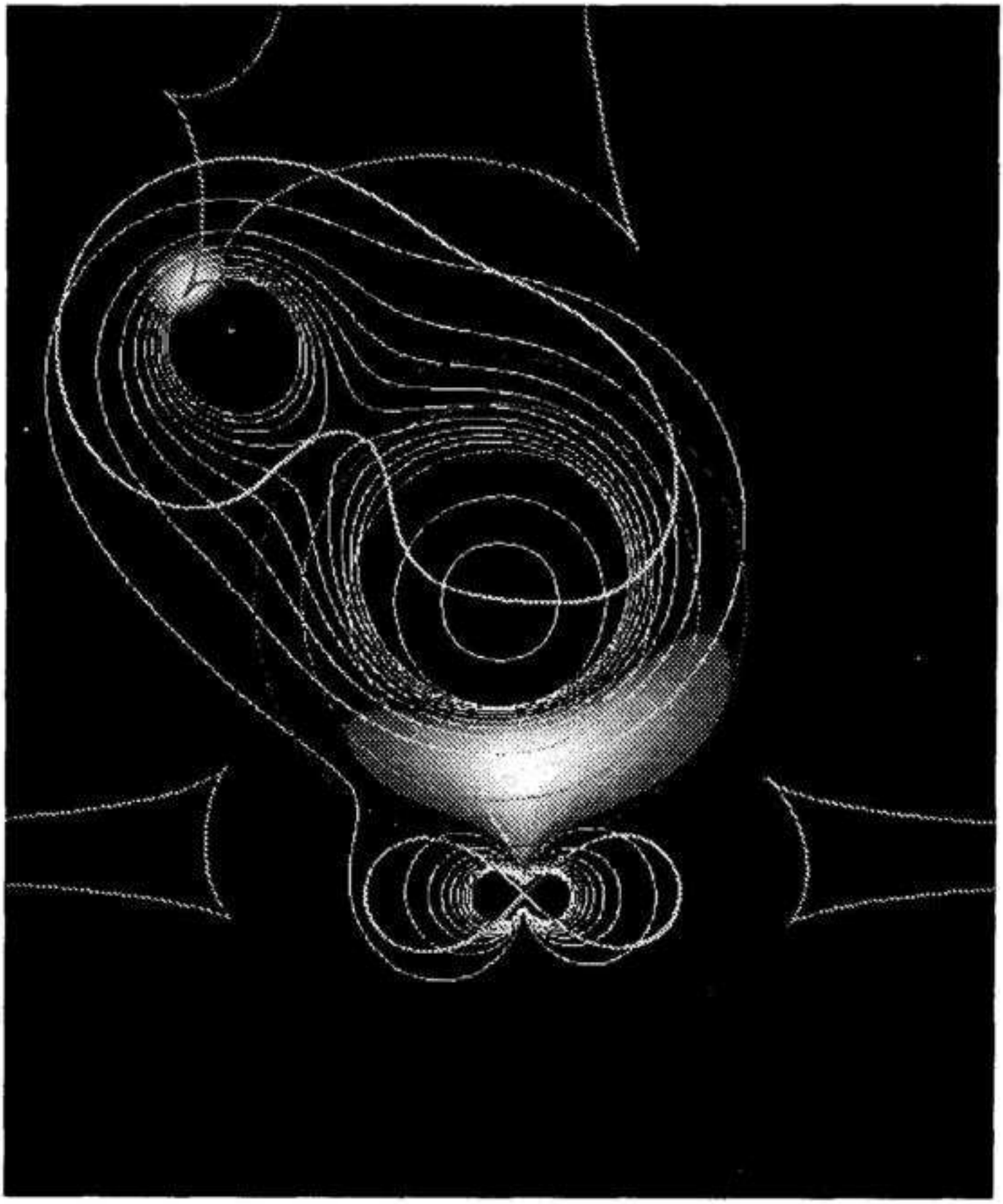

Figura 8: Imágenes producidas por un modelo de *masa puntual doble.* Se ha utilizado una escala $E = 1,00$. $m_1 = 0,49$ en $(-1,1;\ 1.1)$, $m_2 = 0,39$ en $(0,1;\ -1.2)$.

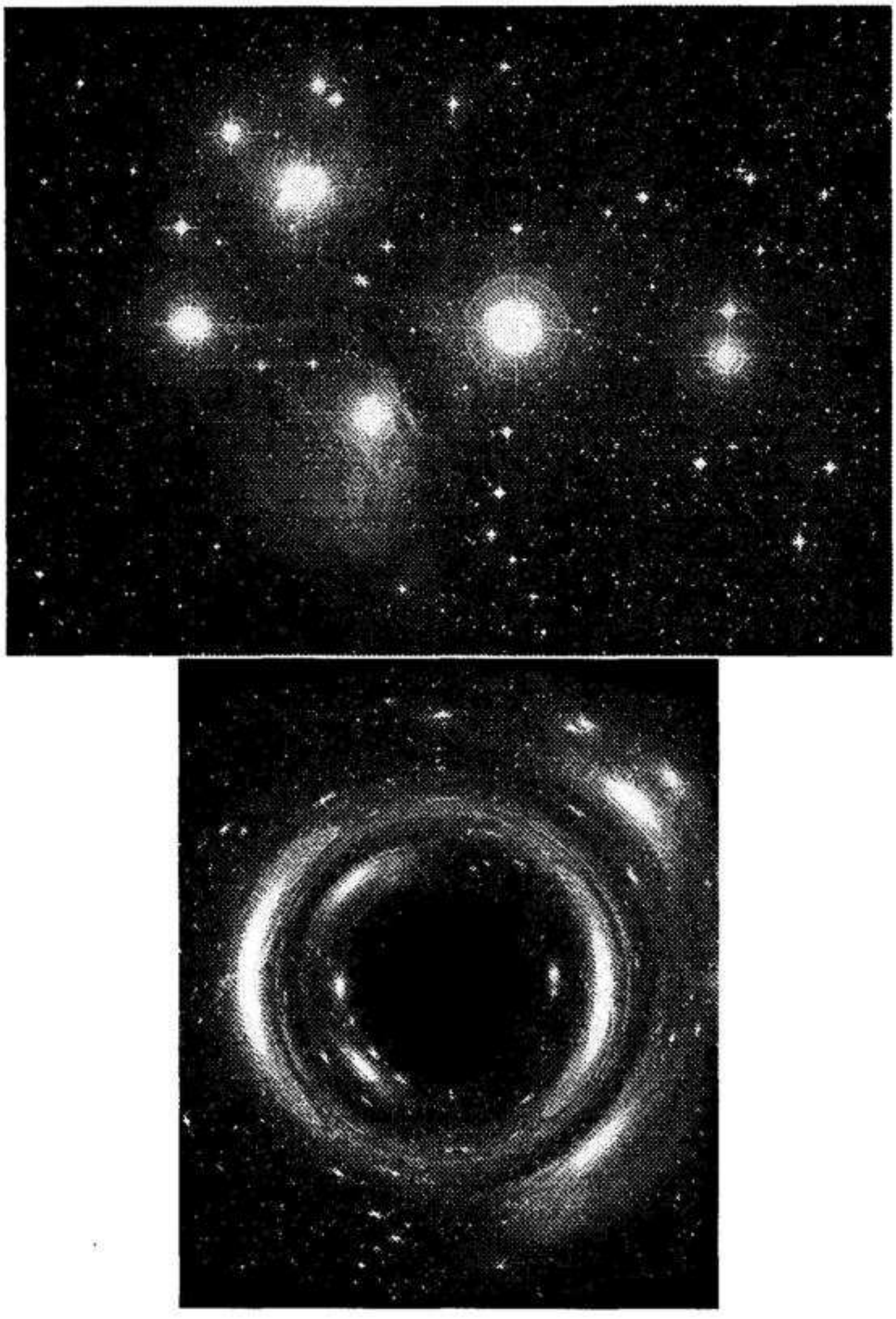

Figura 9: M45 (Pléyades) visto a través de una masa puntual. Arriba, imagen original. Abajo, imagen producida por el modelo con una escala $E = 1,5$. *Fotografía ©Royal Observatory Edinburgh/Anglo-Australian Observatory. Placas UK Schmidt por David Malin*

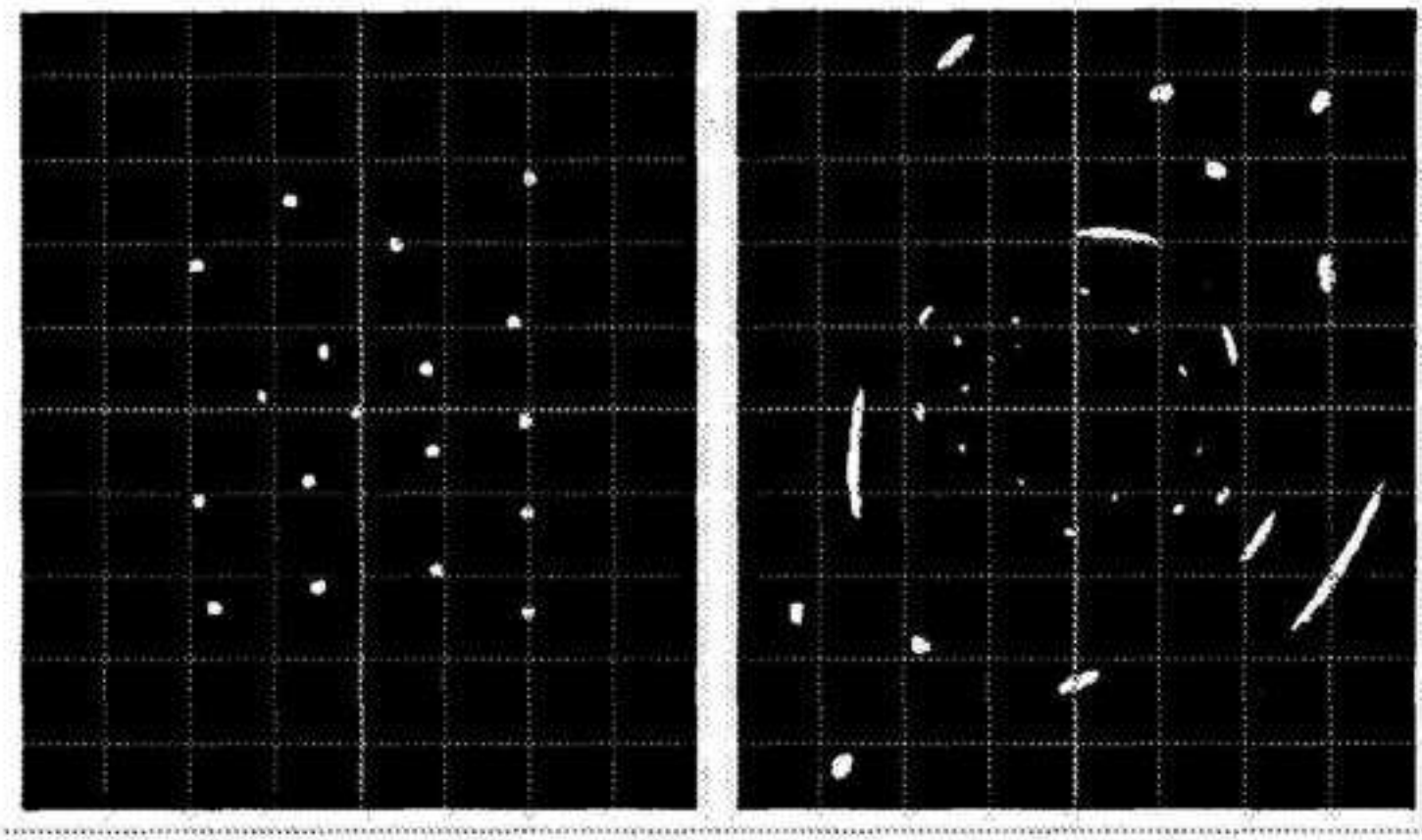

Figura 10: Campo de "galaxias" vistas a través de una lente correspondiente a una masa puntual con contribuciones externas. .

de pared). Para una de las animaciones del vídeo realizado, la cual tiene aproximadamente 30 segundos de duración, se generaron 800 cuadros con el máximo detalle permitido, lo cual requirió de aproximadamente 1000 segundos para calcular, además de aproximadamente 1800 segundos más para la compresión de los cuadros y su conversión a formato MPEG.

El tipo de optimizaciones utilizadas involucran en algunos casos la eliminación de divisiones superfluas, pero son en su mayoría la implementación de algún tipo de *cache* donde se almacenan resultados para su posterior reutilización, si es posible. Esto impone un costo desde el punto de vista de la utilización de recursos, habiéndose observado bajo condiciones extremas un requerimiento de aproximadamente 20 *megabytes* de memoria por parte de la aplicación, más lo que se requiera para el sistema X11 y el sistema operativo. Si se desea preservar la velocidad de la aplicación es improbable que se puedan bajar significativamente los requisitos de memoria. Es necesario señalar que la estrategia de optimización empleada funciona pues se conoce a priori la dimensión del mapa de la lente (*LensMap* en la sección , página 17), a saber, la dimensión de la imagen que se está calculando. Esta solución no se adapta bien al caso de lentes en múltiples planos, aunque el código se puede emplear sin introducir grandes complicaciones en situaciones de esa clase.

5.1 Trabajo futuro

Algunas cosas que se pueden hacer en forma inmediata:

- Implementación de nuevos modelos y refinamiento de los existentes.
- Cálculo de la magnificación total para la imagen.
- Cálculo de curvas de luz para eventos de microlente, partiendo de las secuencias de imágenes.

Cosas que es necesario hacer:

- Implementación de modelos no paramétricos. En particular, el modelado interactivo de distribuciones de materia y el posterior cálculo del potencial de Fermat.
- Mejoramiento de las rutinas de contornos. Debido a las características propias de problema que se trata, las rutinas de contorno estándar tienen algunos problemas, en particular al lidiar con *labios* y *picos* en las caústicas.

Cosas que sería *deseable* hacer:

- Utilización de GTK+ como *toolkit* para la interfaz gráfica, pues ofrece una mayor flexibilidad.
- Conversión de todo el código a C (son pocas las características de C++ que son explotadas por el código actual)
- Limpieza general del código, en particular favorecer el uso de miembros en estructuras vs variables globales.

Posibles líneas de desarrollo de las cuales se beneficiarían también otros proyectos:

- Definición de una interfaz de programación (API) con una subsecuente separación total de la parte puramente visual de la parte puramente física–computacional para permitir la implementación de una *biblioteca compartida.*
- Definición de un API para la implementación de los modelos en forma de objetos dinámicos relocalizables, de manera que sea posible añadir y modificar modelos sin tocar el código de la aplicación.
- Implementación de los diversos cálculos en un esquema paralelo con un modelo amo–esclavo con su correspondiente protocolo de intercambio de datos. El código es inmediatamente paralelizable, pero no se prevé un aumento significativo de la velocidad de la aplicación pues una porción grande (del orden de 50%) del tiempo se utiliza en llevar la imagen hasta el *frame buffer*. Un esquema paralelo debe ir acompañado de la utilización de aceleración de OpenGL a nivel de *hardware*.

Modelo	$\psi(x)$	$\alpha(x)$
Masa puntual	$\ln\|x\|$	$\frac{1}{\|x\|}$
Esfera singular isotérmica	$\sigma^2\|x\|$	σ^2
Esfera no singular isotérmica	$\sigma\sqrt{x_c^2+x^2}$	$\sigma\frac{x}{\sqrt{x_c^2+x^2}}$
Hoja de densidad constante	$\frac{\kappa}{2}x^2$	$\kappa\|x\|$

Tabla 1: Modelos con simetría circular

5. Apéndices

Apéndice A

Modelos

A.1 Modelos con simetría circular

En la tabla se muestran los potenciales efectivos así como los ángulos de deflexión para cuatro modelos con simetría circular.

A.1.1 Esfera isotérmica singular

Un modelo simple para la distribución de masa en una galaxia supone que las estrellas y otros componentes se comportan como partículas en un gas ideal confinadas por un potencial gravitacional con simetría esférica. La ecuación de estado de las partículas es

$$p = \frac{\rho k T}{m} ,$$

donde ρ es la densidad de materia, m es la masa de las estrellas, k es la constante de Boltzmann y T la temperatura asociada. En equilibrio térmico la temperatura se relaciona con la velocidad unidimensional de dispersión σ_v según

$$\frac{1}{2} m \sigma_v^2 = kT .$$

Se supone que este gas estelar es isotérmico, así que σ_v es constante en la galaxia.

En equilibrio hisdrostático [Har88]

$$\frac{p'}{\rho} = -\frac{GM(r)}{r^2}$$
$$M'(r) = 4\pi r^2 \rho$$

donde $M(r)$ es la masa contenida en el interior de r y las primas denotan derivación con respecto a r. Una solución es

$$\rho(r) = \frac{\sigma_v^2}{2\pi G}\frac{1}{r^2} .$$

Para esta distribución, la masa contenida $M(r)$ aumenta $\propto r$, y por tanto, la velocidad de rotación de las partículas es

$$v_{\text{rot}}^2 = \frac{GM(r)}{r} = 2\sigma_v^2 .$$

Utilizando este resultado en 6, se obtiene:

$$\Sigma(\xi) = \frac{\sigma_v^2}{2G}\frac{1}{\xi}$$

y de 7 se tiene

$$\hat{\alpha} = 4\pi\frac{\sigma_v^2}{c^2} .$$

A.2 Modelos con simetría no circular

Los modelos con simetría circular son útiles para estudiar el fenómeno de lente gravitacional, pero su aplicabilidad es limitada para describir galaxias reales, pues estas no son *exactamente* simétricas. Así mismo, no solo hay que considerar la geometría de la masa que actúa como lente, sino también las contribuciones externas, que rompen la simetría [SEF92] [GN88].

A.2.1 Modelos de cuadrupolo

Las configuraciones de algunas galaxias se pueden estudiar utilizando un modelo con simetría axial si se añade una perturbacion. Al hacer una expansión en serie de Taylor, el ángulo de deflexión debido a la perturbación se puede escribir como [SEF92] [Fru98]

$$\boldsymbol{\alpha}_p(\boldsymbol{x}) = \boldsymbol{\alpha}_p(0) + \begin{pmatrix} \Gamma_1 & 0 \\ 0 & \Gamma_2 \end{pmatrix}\boldsymbol{x} = \boldsymbol{\alpha}_p(0) + \begin{pmatrix} \kappa_p + \gamma_p & 0 \\ 0 & \kappa_p - \gamma_p \end{pmatrix}\boldsymbol{x} .$$

$(\Gamma_1 + \Gamma_2)/2$ es la densidad superficial de masa local y $(\Gamma_1 - \Gamma_2)/2$ es la cizalla. La posición de la fuente está descrita por:

$$\boldsymbol{y} = \boldsymbol{\alpha}_p(0) + \left(\mathbf{A} - \frac{\alpha_A(x)}{x}\mathbf{I}\right)\cdot\boldsymbol{x} ,$$

donde

$$\mathrm{A} = \begin{pmatrix} 1-\kappa_p-\gamma_p & 0 \\ 0 & 1-\kappa_p+\gamma_p \end{pmatrix} .$$

A.2.2 Modelos elípticos

Los modelos elípticos derivan su importancia del hecho que muchas galaxias exhiben isofotas elípticas [SSB+98] [BC98]. Aún así, es necesario señalar que existen varios problemas asociados con la utilización de modelos elípticos y se han introducido modelos más elaborados [Sch94].

Considerando el caso de un potencial ψ que solo depende de una variable u de la forma $u = (1-\epsilon)x_1^2 + (1+\epsilon)x_2^2$, es decir, es constante en elipses, se tiene [SEF92]:

$$\boldsymbol{\alpha} = 2\psi' \begin{pmatrix} (1-\epsilon)x_1 \\ (1+\epsilon)x_2 \end{pmatrix} ,$$

y la ecuación de lentes se expresa como

$$y_1 = x_1 - 2(1-\epsilon)\psi'(u)x_1$$
$$y_2 = x_2 - 2(1+\epsilon)\psi'(u)x_2 .$$

La densidad superficial de masa es

$$\kappa = 2\psi' + 2(u+\epsilon v)\psi'' ,$$

donde $v = (1+\epsilon)x_2^2 - (1-\epsilon)x_1^2$. Un caso particular para ψ es

$$\psi(u) = \frac{\kappa_0}{2p}\left[(1+u)^p - 1\right] ,$$

y la densidad correspondiente es

$$\kappa = \kappa_0(1+u)^{p-2}\left[1 + pu - (1-p)\epsilon v\right] .$$

A.2.3 Contribuciones externas

El entorno de una galaxia se manifestará como una contribución en la convergencia y la cizalla. Se puede entonces asociar al entorno un potencial efectivo de la forma

$$\psi(\theta_1, \theta_2) = \frac{\kappa}{2}(\theta_1^2 + \theta_2^2) + \frac{\gamma}{2}(\theta_1^2 - \theta_2^2) ,$$

en el sistema de ejes principales de la cizalla externa. Aquí la convergencia κ y la cizalla γ son localmente independientes de θ. Esto tiene en general el mismo efecto que introducir elipticidad en la lente.

Apéndice B

Programas utilizados para la visualización

El programa se ejecuta bajo un ambiente tipo UNIX, en el sistema de ventanas X11. Para su desarrollo se utilizó el *toolkit* FLTK (*Fast Light Toolkit*) de *Bill Spitzak*. Este *toolkit* fue elegido por cuanto es rápido, demanda pocos recursos por parte del sistema, es fácil de utilizar y soporta directamente ventanas de OPENGL con *buffers* simples y dobles. Una desventaja es que está escrito en C++, lo cual puede presentar problemas debido más que todo a las características del compilador que se utilice. FLTK ofrece además una aplicación para desarrollo rápido de interfaces gráficas, `fluid`. Otros *toolkits* considerados incluyeron GTK+, que es visualmente atractivo y extremadamente fexible, además estar escrito en C. Cuando se comezó el desarrollo de la interfaz gráfica, la documentación de GTK+ todavía no estaba completa, no había disponible una aplicación para desarrollo de interfaces gráficas y el soporte de OPENGL no era completo.

B.1 Definiciones

B.1.1 Definiciones generales, `GL.H`

```
#ifndef _GL_H_INCLUDED
#define _GL_H_INCLUDED

#include <math.h>

#ifndef NULL
#define NULL (0)
#endif

#define GRAD (M_PI/180.)

typedef double GL_parms;

typedef struct {
        double x1, x2;
        double y1, y2;
} map_;

typedef struct {
        int w, h;
```

```
        int step;
        int needs_remake;
        map_ ** Map;
} map;

typedef struct {
        int w, h;
        int step;
        int needs_remake;
        double ** A;
} jacobian_matrix;

extern void lens_equation(int, int, double, double, double *, double *);
extern double potential(int, double, double);
extern void resize_map (int, int, int);
extern void remake_map (void);
extern int make_jacmatrix (void);
extern void make_map (void);
extern void switch_model (int, int);
extern void load_prm(const char *);

extern long switches;
extern int source_function;
extern int auto_save;
#endif
```

B.1.2 Definiciones panel de control, GLcp.H

```
#ifndef _GLcp_H_INCLUDED_
#define _GLcp_H_INCLUDED_

// Put ControlPanel-specific things here, but don't include any
// model-related stuff here, that goes in GL_models.H

#include "GL_Positioner.H"
#include <FL/Fl_Value_Input.H>
#include <FL/x.H>
#include <Imlib.h>// Imlib is used to read images

// Enumerations

enum _gl_source { // source figures
F_SOLID,
F_GRAYSCALE,
F_DATAFILE
};

enum _gl_switches { // switches
```

```
// These define different values for a mask

S_SOURCE = 0,
S_IMAGES = 1,
S_GRID = 2,
S_EINSTEIN = 3,
S_DATA = 4,
        S_DELAY = 5,
        S_CRITICAL = 6,
        NOF_SWITCHES = 7
};

enum _gl_menu_items { // items on the menu

MI_QUIT,
MI_OPEN,
MI_SAVE,
MI_CHOOSEDATA
};

// typedefs

typedef struct {
Positioner_Window *Positioner;
Fl_Value_Input *X;
Fl_Value_Input *Y;
} GL_Positioner_Box;

// Global variables

extern GL_Positioner_Box Source;

extern ImlibData *ImlibId;

extern char * ImageFilename;
extern char * GraphFilename;
extern ImlibData * ImlibId;
#endif // _GLcp_H_INCLUDED_
```

B.1.3 Imágenes, GL_Drawing_Area.H

```
#ifndef _GL_DRAWING_AREA_H_INCLUDED
#define _GL_DRAWING_AREA_H_INCLUDED

#include <GL.H>
#include <FL/Fl_Gl_Window.H>
#include <FL/gl.h>
```

```
class GL_Drawing_Area:public Fl_Gl_Window {
        void draw();
void draw_grid();
void draw_images();
        void draw_isocrones();
        void draw_critical ();
void draw_inverted_images();
        void draw_source(GLdouble x, GLdouble y, GLdouble r);
void resize(int X, int Y, int W, int H);
int handle(int);
GLdouble bound;
GLdouble xmin_, xmax_;
GLdouble ymin_, ymax_;
        GLdouble step_;
GLdouble panx, pany;
GLint mousex, mousey;
        map * LensMap;
        jacobian_matrix * JacMatrix;

public:
        GL_Drawing_Area(int x, int y, int w, int h, const char *L)
                :Fl_Gl_Window(x, y, w, h, L) {
        };
GL_Drawing_Area(int x, int y, int w, int h)
                :Fl_Gl_Window(x, y, w, h) {
};
        double xmin () const { return xmin_; }
        double xmax () const { return xmax_; }
        double ymin () const { return ymin_; }
        double ymax () const { return ymax_; }
        double step () const { return step_; }
        void step (double);
        void lensmap (map *);
        void jacmatrix (jacobian_matrix *);
        int save_image (char *);
};

#endif // _GL_DRAWING_AREA_H_INCLUDED
```

B.1.4 Parámetros para modelos, `GL_model_parameters.H`

```
#ifndef GL_MODEL_PARAMETERS_H
#define GL_MODEL_PARAMETERS_H

enum {
     M_PLANE1,
     NOF_PLANES          // the *number* of planes
};
```

```
enum {
    M_NONE,             /* this one is tricky, because it's kind
                         * of a waste to set this information,
                         * but it's easier to understand the
                         * program (read, actually) this way */
    M_CHANG_REFSDAL,
    M_SIS,
    M_NSIS,
    M_TRANSP_SPHERE,
    M_ELLIPTICAL,
    M_KING,
    M_TRUNC_KING,
    M_HUBBLE,
    M_DE_VACOULEUR,
    M_SPIRAL,
    M_MULTIPOLE,
    M_ROTATION_LENS,
    M_DOUBLE_LENS,
    NOF_MODELS          // the *number* of models including none
};

enum {
    P_R,                // Source "radius"
    P_THETA,            // Image orientation
    P_NOF_GNRL_PRMS
};

enum {
    P_E,                // Plane scale
    P_GAMMA,
    P_SIGMA,
    P_PHI,
    P_NOF_PLANE_PRMS
};

// enums for each model parameter

enum {
    P_CHANG_REFSDAL_NOF_PARAMETERS
};

enum {
    P_SIS_KAPPA,
    P_SIS_NOF_PARAMETERS
};
```

```
enum {
     P_NSIS_C,
     P_NSIS_NOF_PARAMETERS
};

enum {
     P_TRANSP_SPHERE_C,
     P_TRANSP_SPHERE_NOF_PARAMETERS
};

enum {
     P_ELLIPTICAL_C,
     P_ELLIPTICAL_KAPPA,
     P_ELLIPTICAL_EPSILON,
     P_ELLIPTICAL_ALPHA,
     P_ELLIPTICAL_NOF_PARAMETERS
};

enum {
     P_KING_C,
     P_KING_KAPPA,
     P_KING_NOF_PARAMETERS
};

enum {
     P_TRUNC_KING_C,
     P_TRUNC_KING_KAPPA,
     P_TRUNC_KING_NOF_PARAMETERS
};

enum {
     P_HUBBLE_KAPPA,
     P_HUBBLE_NOF_PARAMETERS
};

enum {
     P_DE_VACOULEUR_KAPPA,
     P_DE_VACOULEUR_NOF_PARAMETERS
};

enum {
     P_SPIRAL_KAPPA,
     P_SPIRAL_NOF_PARAMETERS
};

enum {
     P_MULTIPOLE_DX,
```

```
    P_MULTIPOLE_DY,
    P_MULTIPOLE_Q1,
    P_MULTIPOLE_Q2,
    P_MULTIPOLE_NOF_PARAMETERS
};

enum {
    P_ROTATION_SX,
    P_ROTATION_SY,
    P_ROTATION_LENS_NOF_PARAMETERS
};

enum {
    P_DOUBLE_LENS_M1,
    P_DOUBLE_LENS_M2,
    P_DOUBLE_LENS_BETA,
    P_DOUBLE_LENS_X1,
    P_DOUBLE_LENS_Y1,
    P_DOUBLE_LENS_X2,
    P_DOUBLE_LENS_Y2,
    P_DOUBLE_LENS_NOF_PARAMETERS
};

// find a cleaner way to do this... this is ugly!!!

const int MAX_NOF_PARAMETERS = P_DOUBLE_LENS_NOF_PARAMETERS;

#endif // GL_MODEL_PARAMETERS_H
```

B.1.5 Modelos, GL_models.H

```
#ifndef GL_MODELS_H
#define GL_MODELS_H

#include <FL/Fl.H>
#include <FL/Fl_Valuator.H>
#include <FL/Fl_Value_Slider.H>

#include "GL_model_parameters.H"

const int nof_parameters_model[NOF_MODELS] =
{
    0,// M_NONE
    P_CHANG_REFSDAL_NOF_PARAMETERS,
    P_SIS_NOF_PARAMETERS,
    P_NSIS_NOF_PARAMETERS,
    P_TRANSP_SPHERE_NOF_PARAMETERS,
    P_ELLIPTICAL_NOF_PARAMETERS,
```

```
     P_KING_NOF_PARAMETERS,
     P_TRUNC_KING_NOF_PARAMETERS,
     P_HUBBLE_NOF_PARAMETERS,
     P_DE_VACOULEUR_NOF_PARAMETERS,
     P_SPIRAL_NOF_PARAMETERS,
     P_MULTIPOLE_NOF_PARAMETERS,
     P_ROTATION_LENS_NOF_PARAMETERS,
     P_DOUBLE_LENS_NOF_PARAMETERS
};

/* all the parameters are doubles */
typedef double TParameterV;

/* the containers are all Fl_Value_Slider */
typedef Fl_Value_Slider TParameter;

typedef TParameter *PParameter;

/* a butt-ugly structure containing all the values of the relevant
 * parameters to be passed to the final calculation */

typedef struct {
     TParameterV general[P_NOF_GNRL_PRMS];
     struct {
          TParameterV prms[P_NOF_PLANE_PRMS];
          TParameterV mprms[MAX_NOF_PARAMETERS];
     } plane[NOF_PLANES];
} GL_ParametersV;

typedef struct {
     unsigned int number_of_parameters;
     PParameter *parameters;
} GL_Model;

typedef struct {
     PParameter parameters[P_NOF_PLANE_PRMS];
     GL_Model models[NOF_MODELS];
} GL_Plane;

typedef struct {
     PParameter parameters[P_NOF_GNRL_PRMS];
     GL_Plane planes[NOF_PLANES];
} GL_Parameters;

extern GL_Parameters parameters;
extern GL_ParametersV prms;
```

```
// This will hold the index to the previous model

extern int current_model[];

// prototypes

extern void init_model(int, int);
//void lens_equation(int, int, double, double, double *, double *);

#endif // GL_MODELS_H
```

B.2 Programa principal, GLcp.C

```
// $Id: GLcp.C,v 1.22 1999/09/20 02:50:26 mmagallo Exp $

// some global symbols, not UI specific
#include <GL.H>
// the interface stuff -- callbacks mostly
#include <GLcp_ui.H>
// control panel stuff: global variables and such
#include <GLcp.H>
// all the model related stuff (variables included)
#include <GL_models.H>

#include <malloc.h>

// Instantiate this here... it doesn't belong anywhere...

int current_model[NOF_PLANES];
char * ImageFilename = 0;
ImlibData * ImlibId = 0;

GL_Positioner_Box Source;

map LensMap = { 0, 0, 0, 0, 0 };
jacobian_matrix JacMatrix = { 0, 0, 0, 0 };

int main(int argc, char **argv)
{
int i, j;

// some initialization stuff

for (i = 0; i < NOF_PLANES; i++) {
```

```
for (j = 0; j < NOF_MODELS; j++) {
GL_Model *current =
    &(parameters.planes[i].models[j]);
current->number_of_parameters =
    nof_parameters_model[j];
if (current->number_of_parameters)
current->parameters =
    new PParameter[current ->
   number_of_parameters];
else
current->parameters = NULL;
}

current_model[i] = 0;
}

switches = 0;
source_function = F_SOLID;

make_panels();

fl_open_display();
ImlibId = Imlib_init_with_params(fl_display,
 (ImlibInitParams *) 0);

RenderWindow->size_range(370, 462, 740, 832, 0, 0, 1);

        GLgl_Window -> lensmap (&LensMap);
        GLgl_Window -> jacmatrix (&JacMatrix);
        GLmag_Window -> lensmap (&LensMap);
        GLmag_Window -> jacmatrix (&JacMatrix);
        GLgl_Window -> step (precision -> value ());

ControlPanel-> show ();
        RenderWindow -> show ();
        GLgl_Window -> show ();

(make_plane_cp(0)) -> show ();

        if (argc > 1) load_prm (argv[1]);

return Fl::run();
}

void pack_parameters(void);

void remake_map (void) {
```

```
        LensMap.needs_remake = 1;
        GLgl_Window->valid(0);
        GLgl_Window->redraw();
}

void resize_map (int w, int h, int step) {
        int wp, hp;

        if (LensMap.Map) {
                wp = LensMap.w / LensMap.step;

                for (int i = 0; i < wp; i++)
                        free (LensMap.Map[i]);
                free (LensMap.Map);
        }

        LensMap.w = w;
        LensMap.h = h;
        LensMap.step = step;

        wp = LensMap.w / LensMap.step;
        hp = LensMap.h / LensMap.step;

LensMap.Map = (map_ **) calloc (wp, sizeof(map_ *));
        for (int i = 0; i < wp; i++)
                LensMap.Map[i] = (map_ *) calloc (hp, sizeof(map_));

        LensMap.needs_remake = 1;
}

void make_map (void) {
        double x1min, x1max, x2min, x2max, x1step, x2step;
        double wp, hp;
        int i, j;

        if (LensMap.needs_remake) {
                if (LensMap.w != GLgl_Window -> w () ||
                    LensMap.h != GLgl_Window -> h () ||
                    LensMap.step != int (GLgl_Window -> step ())) {
                        resize_map (GLgl_Window -> w (),
                                    GLgl_Window -> h (),
                                    int(GLgl_Window -> step ()));
                }

                x1min = GLgl_Window -> xmin ();
                x1max = GLgl_Window -> xmax ();
                x2min = GLgl_Window -> ymin ();
```

```
x2max = GLgl_Window -> ymax ();

wp = LensMap.w / LensMap.step;
hp = LensMap.h / LensMap.step;

x1step = (x1max - x1min) / wp;
x2step = (x2max - x2min) / hp;

pack_parameters ();

for (i = 0; i < NOF_PLANES; i++) {
        init_model (i, current_model[i]);
}

for (i = 0; i < wp; i++) {
        for (j = 0; j < hp; j++) {
                LensMap.Map[i][j].x1 = x1min + x1step * i;
                LensMap.Map[i][j].x2 = x2min + x2step * j;

                lens_equation(current_model[M_PLANE1],
                              0,
                              LensMap.Map[i][j].x1,
                              LensMap.Map[i][j].x2,
                              &(LensMap.Map[i][j].y1),
                              &(LensMap.Map[i][j].y2));
                }
        }

        LensMap.needs_remake = 0;
        JacMatrix.needs_remake = 1;
    }
}

int make_jacmatrix (void)
{
        int wp, hp, k;
        double x1min, x1max, x2min, x2max, x1step, x2step;

        if (LensMap.needs_remake)
                make_map ();

        if (JacMatrix.needs_remake) {
                if (JacMatrix.w != LensMap.w ||
                    JacMatrix.h != LensMap.h ||
                    JacMatrix.step != LensMap.step) {
                        if (JacMatrix.A) {
                                wp = JacMatrix.w / JacMatrix.step;
```

```
                for (int i = 0; i < wp; i++)
                        delete JacMatrix.A[i];
                delete JacMatrix.A;
        }

        JacMatrix.w = LensMap.w;
        JacMatrix.h = LensMap.h;
        JacMatrix.step = LensMap.step;

        wp = JacMatrix.w / JacMatrix.step;
        hp = JacMatrix.h / JacMatrix.step;
        JacMatrix.A = new double*[wp];
        for (int i = 0; i < wp; i++)
                JacMatrix.A[i] = new double[hp];
}

wp = JacMatrix.w / JacMatrix.step;
hp = JacMatrix.h / JacMatrix.step;
k = JacMatrix.step;

x1min = GLgl_Window -> xmin ();
x1max = GLgl_Window -> xmax ();
x2min = GLgl_Window -> ymin ();
x2max = GLgl_Window -> ymax ();

x1step = (x1max - x1min)/wp;
x2step = (x2max - x2min)/hp;

for (int i = 1; i < wp; i++) {
        for (int j = 1; j < hp; j++) {
                JacMatrix.A[i][j] =
                        (
                         (LensMap.Map[i][j-1].y1 -
                          LensMap.Map[i-1][j-1].y1)
                         *
                         (LensMap.Map[i-1][j].y2 -
                          LensMap.Map[i-1][j-1].y2)
                         -
                         (LensMap.Map[i-1][j].y1 -
                          LensMap.Map[i-1][j-1].y1)
                         *
                         (LensMap.Map[i][j-1].y2 -
                          LensMap.Map[i-1][j-1].y2)
                        )/(x1step*x2step);
        }
}
return 1;
```

```
        } else
                return 0;
}

void pack_parameters()
{
int i, j;

for (i = 0; i < P_NOF_GNRL_PRMS; i++)
prms.general[i] = parameters.parameters[i]->value();

for (i = 0; i < NOF_PLANES; i++) {
for (j = 0; j < P_NOF_PLANE_PRMS; j++)
prms.plane[i].prms[j] =
    parameters.planes[i].parameters[j]->value();
for (j = 0; j < nof_parameters_model[current_model[i]];
     j++) prms.plane[i].mprms[j] =
    parameters.planes[i].
    models[current_model[i]].parameters[j]->
    value();
}
}

void load_prm(const char *c) {
        double dv;
        int di;
        char imagefilename[1024];
        char graphfilename[1024];
        FILE * pf;
        pf = fopen (c, "r");

        for (int i = 0; i < P_NOF_GNRL_PRMS; i++) {
                fscanf (pf, "%lf", &dv);
                parameters.parameters[i] -> value(dv);
        }

        for (int i = 0; i < NOF_PLANES; i++) {
                fscanf (pf, "%d: ", &di);
                for (int j = 0; j < P_NOF_PLANE_PRMS; j++) {
                        fscanf (pf, "%lf", &dv);
                        parameters.planes[i].parameters[j] ->
                                value(dv);
                }
                fscanf (pf, "%i", &di);
                switch_model (i, di); model_menu[i] -> value(di);

                for (int k = 1 ; k < NOF_MODELS; k++) {
```

```
                int dk;
                fscanf (pf, "%d:%d: ", &di, &dk);
                for (int j = 0;
                     j < nof_parameters_model[k];
                     j++) {
                        fscanf (pf, "%lf ", &dv);
                        parameters.planes[i].
                                models[k].
                                parameters[j] -> value(dv);
                }
        }
}
fscanf (pf, "%lf", &dv); Source.X -> value(dv);
fscanf (pf, "%lf", &dv); Source.Y -> value(dv);
Source.Positioner -> value (Source.X -> value(),
                            Source.Y -> value());
fscanf (pf, "%li", &switches);
for (int i = 0; i < NOF_SWITCHES; i++)
        glswitch[i] -> value (switches & (1 << i));
fscanf (pf, "%lf", &dv); precision -> value(dv);
GLgl_Window -> step (precision -> value());
fscanf (pf, "%lf", &dv); PanX -> value(dv);
fscanf (pf, "%lf", &dv); PanY -> value(dv);
fscanf (pf, "%lf", &dv); ZoomFactor -> value(dv);
fscanf (pf, "%i", &source_function);
        fscanf (pf, "%s", imagefilename);
        free (ImageFilename);
        ImageFilename = strdup (imagefilename);
        if (!feof(pf)) {
                fscanf (pf, "%i", &auto_save);
                fscanf (pf, "%s", graphfilename);
                free (GraphFilename);
                GraphFilename = strdup (graphfilename);
        }

        fclose (pf);
        remake_map ();
}
```

B.3 Imágenes, `GL_Drawing_Area.C`

```
#include <stream.h>
#include <math.h>
```

```
#include <GL_Drawing_Area.H>
#include <GL/glu.h>

#include <GL.H>
#include <GLcp.H>
#include <GLcp_ui.H>
#include <GL_models.H>

#include <plcont.h>

#define SLICES          (32)
#define SIZE_0          ((double)512.)
#define BOUND_0         ((double)2.)
#define STEP0           (0.5)

#define MAGIC_NUMBER (10.)

#define R_MIN           (0.05)
#define N_CIRCLES       (5)

int auto_save = 0;
char * GraphFilename = 0;

void draw_source(GLdouble, GLdouble, GLdouble);
void draw_rings(GLdouble, GLdouble, GLdouble);
void draw_pixel (const GLubyte *, double, double);
void colormap(float, float, GLfloat *);
void draw_photo(void);

void i2world (double, double, double *, double *, void *);
void i2source (double, double, double *, double *, void *);

void GL_Drawing_Area::step (double v) {
        step_ = v;
}

void GL_Drawing_Area::lensmap (map * v) {
        LensMap = v;
}

void GL_Drawing_Area::jacmatrix (jacobian_matrix * v) {
        JacMatrix = v;
}

void GL_Drawing_Area::draw()
{
if (!valid()) {
```

```
bound = ZoomFactor->value() * BOUND_0;
panx  = ZoomFactor->value() * PanX->value();
pany  = ZoomFactor->value() * PanY->value();
xmin_ = -bound + panx;
xmax_ = bound + panx;
ymin_ = -bound*h()/w() + pany;
ymax_ = bound*h()/w() + pany;
valid(1);
glLoadIdentity();
glViewport(0, 0, w(), h());
gluOrtho2D(xmin_, xmax_, ymin_, ymax_);
}
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);

if (switches & (1 << S_IMAGES))
// if (switches & (1 << S_INVERT))
// draw_inverted_images();
// else
                draw_images();

        if (switches & (1 << S_DELAY))
                draw_isocrones();

if (switches & (1 << S_CRITICAL))
                draw_critical();

if (switches & (1 << S_SOURCE))
draw_source(Source.X->value(),
    Source.Y->value(),
    parameters.parameters[P_R]->value());

if (switches & (1 << S_EINSTEIN))
draw_rings(0.0,
   0.0,
   parameters.planes[0].parameters[P_E]->value());

if (switches & (1 << S_GRID))
draw_grid();

        if (auto_save && GraphFilename) {
                save_image (GraphFilename);
                if (auto_save > 1) exit(0);
        }
}

int GL_Drawing_Area::handle(int event)
```

```
{
int event_x = Fl::event_x();
int event_y = Fl::event_y();

switch (event) {
case FL_PUSH:
mousex = event_x;
mousey = event_y;
return 1;
case FL_DRAG:
panx += (event_x - mousex) * 2. * bound / (double) w();
pany += (event_y - mousey) * 2. * bound / (double) h();
mousex = event_x;
mousey = event_y;
return 0;
case FL_RELEASE:
printf("%d %d\n", event_x, event_y);
return 1;
case FL_KEYBOARD:
// ... keypress, key is in Fl::event_key(), ascii in
// Fl::event_text()
return 1;
default:
// tell fltk that I don't understand other events
return 0;
}
return 0;
}

void GL_Drawing_Area::draw_source(GLdouble x, GLdouble y, GLdouble r) {
GLfloat color[3];
double dr;
int i;
GLdouble fx, fy;
ImlibImage *im;
GLdouble xi, yi;
GLint w = GLgl_Window->w();
GLdouble dp = step_ * 2. * bound / (double) w * 0.99; // FIXME!

GLUquadricObj *qobj = gluNewQuadric();
gluQuadricDrawStyle(qobj, (enum GLenum) GLU_SILHOUETTE);

glPushMatrix();
glRotated(parameters.parameters[P_THETA]->value(), 0., 0., 1.);
glTranslated(x, y, 0.);

switch (source_function) {
```

```
case F_SOLID:
glColor3f(1.0, 1.0, 1.0);
gluDisk(qobj, 0., r, SLICES, 1);
break;

case F_GRAYSCALE:
if (r <= R_MIN) {
glColor3f(1.0, 1.0, 1.0);
gluDisk(qobj, 0., R_MIN, SLICES, 1);
} else {
dr = (r - R_MIN) / (double) N_CIRCLES;

for (i = 0; i < N_CIRCLES; ++i, r -= dr) {
colormap(0, r, color);
glColor3fv(color);
gluDisk(qobj, 0., r, SLICES, 1);
}
}

break;

case F_DATAFILE: {
im = Imlib_load_image(ImlibId, ImageFilename);
fx = fy = -SIZE_0 / (2. * BOUND_0);
```

```
                float ps;
                glGetFloatv (GL_POINT_SIZE, &ps);
                glPointSize (double(step_));

glBegin(GL_POINTS);
for (xi = xmin_; xi <= xmax_; xi += dp) {
for (yi = ymin_; yi <= ymax_; yi += dp) {
int yp =
    (int) ((yi - y) * fy) +
    im->rgb_height / 2;
int xp =
    (int) ((xi - x) * fx) +
    im->rgb_width / 2;
if ((xp >= 0) && (xp < im->rgb_width)
    && (yp >= 0) && (yp < im->rgb_height)) {
draw_pixel (im->rgb_data +
    3 * (yp *
 im->rgb_width +
 xp),
                                                xi, yi);
}
}
```

```
}
glEnd();
                glPointSize (ps);

Imlib_destroy_image(ImlibId, im);

break;
        }

default:
cerr << "Not yet implemented!\n";
}

glPopMatrix();
gluDeleteQuadric(qobj);
}

void draw_rings(GLdouble x, GLdouble y, GLdouble r)
{
GLUquadricObj *qobj = gluNewQuadric();

glPushMatrix();
glColor3f(1.0, 1.0, 0.0);
gluQuadricDrawStyle(qobj, (enum GLenum) GLU_SILHOUETTE);
gluDisk(qobj, 0., r, 32, 1);
glPopMatrix();
gluDeleteQuadric(qobj);
}

void GL_Drawing_Area::draw_grid()
{
GLdouble i;
GLdouble step;
GLdouble Xi, Xf, Yi, Yf;
GLfloat OldLineWidth;

if (ZoomFactor->value() < 1.0)
step =
    STEP0 * ceil(ZoomFactor->value() * MAGIC_NUMBER) /
    MAGIC_NUMBER;
else
step = STEP0 * ceil(ZoomFactor->value());

Xi = step * floor(xmin_ / step);
Xf = step * ceil(xmax_ / step);

Yi = step * floor(ymin_ / step);
```

```
Yf = step * ceil(ymax_ / step);

glColor3d(0.45, 0.45, 0.45);

glBegin(GL_LINES);
for (i = Xi; i < Xf; i += step) {
glVertex2f(i, ymin_);
glVertex2f(i, ymax_);
}
for (i = Yi; i < Yf; i += step) {
glVertex2f(xmin_, i);
glVertex2f(xmax_, i);
}
glEnd();

glGetFloatv(GL_LINE_WIDTH, &OldLineWidth);
glLineWidth(2.0);
glBegin(GL_LINES);
glVertex2f(0., ymin_);
glVertex2f(0., ymax_);
glVertex2f(xmin_, 0.);
glVertex2f(xmax_, 0.);
glEnd();
glLineWidth(OldLineWidth);
}
```

```
void GL_Drawing_Area::resize(int X, int Y, int W, int H)
{
resize_map(W, H, int(step_));
Fl_Gl_Window::resize(X, Y, W, H);
}

void colormap(float x, float y, GLfloat * color)
{
GLfloat R2 = parameters.parameters[P_R]->value();
R2 *= R2;
color[0] = color[1] = color[2] =
    -0.75 * (x * x + y * y) / R2 + 1.0;
}

void GL_Drawing_Area::draw_images(void)
{
ImlibImage *im = 0;
double fx = 1.;
double fy = 1.;

float color[3];
```

```
        int i, j;
double x1, x2;
double x1bar, x2bar;
double y1, y2;
double dist;

double z1 = Source.X -> value ();
double z2 = Source.Y -> value ();

int w = this -> w ();
int h = this -> h ();
        int step = int (step_);
        int wp = w / step;
        int hp = h / step;

//        double x1step = (xmax_ - xmin_) / double (wp);
//        double x2step = (ymax_ - ymin_) / double (hp);

GL_Plane & plane = parameters.planes[M_PLANE1];

double cs = cos(plane.parameters[P_THETA] -> value() * GRAD);
double ss = sin(plane.parameters[P_THETA] -> value() * GRAD);

double R2 = parameters.parameters[P_R]->value();

R2 *= R2;

// pack_parameters();

glPointSize (step * M_SQRT2);

if (step == 1)
glEnable(GL_POINT_SMOOTH);
else
glDisable(GL_POINT_SMOOTH);

if (source_function == F_DATAFILE) {
im = Imlib_load_image(ImlibId, ImageFilename);
fx = fy = -SIZE_0 / (2. * BOUND_0);
}
glColor3f(1.0, 1.0, 1.0);

        make_map ();

glBegin(GL_POINTS);
for (i = 0; i < wp; i++) {
```

```
for (j = 0; j < hp; j++) {
                        x1 = LensMap->Map[i][j].x1;
                        x2 = LensMap->Map[i][j].x2;

x1bar = x1 * cs + x2 * ss;
x2bar = -x1 * ss + x2 * cs;

/* FIXME!!! this is no longer lens_equation but something that figures
 * out which coordinate use from the lens map and apply that.
 *
 *                      lens_equation(current_model[M_PLANE1], 0, x1bar,
 *                                    x2bar, &y1, &y2);
 */

#if 0
y1 = x1bar;
y2 = x2bar;
#else
                        y1 = LensMap->Map[i][j].y1;
                        y2 = LensMap->Map[i][j].y2;
#endif

y1 -= z1;
y2 -= z2;
```

```
if (source_function == F_DATAFILE) {
int xp =
    (int) ((y1) * fx) + im->rgb_width / 2;
int yp =
    (int) ((y2) * fy) + im->rgb_height / 2;

if ((xp >= 0) && (xp < im->rgb_width) &&
    (yp >= 0) && (yp < im->rgb_height)) {
                                    // i'm dead sure there's a better
                                    // way to do this, but wth, long
                                    // live the kludge!
draw_pixel (im->rgb_data +
    3 * (yp *
 im->rgb_width +
 xp),
                                                    x1, x2);
}
} else {
dist = y1 * y1 + y2 * y2;

if (dist <= R2) {
if (source_function == F_GRAYSCALE) {
```

```
colormap(y1, y2, color);
glColor3fv(color);
}
glVertex2d(x1, x2);
}
}
}
glEnd();

if (source_function == F_DATAFILE)
Imlib_destroy_image(ImlibId, im);
}

void GL_Drawing_Area::draw_isocrones (void) {
        double clevel[10];
        const int k = 3;
        int step = int (step_)*k;
int wp = this -> w () / step;
int hp = this -> h () / step;
        double x1step = (xmax_ - xmin_) / double (this -> w ());
        double x2step = (ymax_ - ymin_) / double (this -> h ());
        double min, max;

        double prm[] = { x1step, x2step, xmin_, ymin_, step };

        double ** Potential = new double*[wp];
        for (int i = 0; i < wp; i++)
             Potential[i] = new double[hp];

        make_map ();

        min = max =
                0.5 *
                ((LensMap->Map[0][0].x1 - LensMap->Map[0][0].y1) *
                 (LensMap->Map[0][0].x1 - LensMap->Map[0][0].y1)
                 +
                 (LensMap->Map[0][0].x2 - LensMap->Map[0][0].y2) *
                 (LensMap->Map[0][0].x2 - LensMap->Map[0][0].y2))
                - potential (current_model[M_PLANE1],
                             LensMap->Map[0][0].x1,
                             LensMap->Map[0][0].x2);

for (int i = 0; i < wp; i++) {
for (int j = 0; j < hp; j++) {
                        Potential[i][j] =
                                0.5 *
```

```
                        ((LensMap->Map[i*k][j*k].x1 -
                          LensMap->Map[i*k][j*k].y1) *
                         (LensMap->Map[i*k][j*k].x1 -
                          LensMap->Map[i*k][j*k].y1)
                         +
                         (LensMap->Map[i*k][j*k].x2 -
                          LensMap->Map[i*k][j*k].y2) *
                         (LensMap->Map[i*k][j*k].x2 -
                          LensMap->Map[i*k][j*k].y2))
                        -
                        potential (current_model[M_PLANE1],
                                   LensMap->Map[i*k][j*k].x1,
                                   LensMap->Map[i*k][j*k].x2);

                if (Potential[i][j] < min)
                        min = Potential[i][j];
                else if (Potential[i][j] > max)
                        max = Potential[i][j];
        }
}

if (max > 2.) max = 2.;

for (int i = 0; i < 10; i++)
        clevel[i] = min + (max - min)/10.*i;

        glColor3f (1., 1., 0.);
        plcont ((double **)Potential,
                wp, hp, 1, wp, 1, hp,
                clevel, 10,
                i2world, prm);

        for (int i = 0; i < wp; i++)
             delete Potential[i];
        delete Potential;

}

void GL_Drawing_Area::draw_critical (void) {
        double clevel[] = { 0. };
int wp = w () / int(step_);
int hp = h () / int(step_);
        double x1step = (xmax_ - xmin_) / double (w());
        double x2step = (ymax_ - ymin_) / double (h());
        double prm[] = { x1step, x2step, xmin_, ymin_, step_ };
        double lw;
        glGetDoublev (GL_LINE_WIDTH, &lw);
```

```
        make_jacmatrix ();

        glColor3f (1., .6, 0.);
        glLineWidth (2.);
        plcont (JacMatrix -> A,
                wp, hp,
                2, wp,
                2, hp,
                clevel, 1,
                i2world, prm);
        glLineWidth (lw);

        glColor3f (0., .6, 1.);
        glLineWidth (2.);
        plcont (JacMatrix -> A,
                wp, hp,
                2, wp,
                2, hp,
                clevel, 1,
                i2source, (void *)LensMap);
        glLineWidth (lw);
}

void GL_Drawing_Area::draw_inverted_images(void) {
int i;
ImlibImage *im = 0;
GLdouble fx = 1.;
GLdouble fy = 1.;

GLfloat color[3];

GLdouble x1, x2;
GLdouble x1bar, x2bar;
GLdouble y1, y2;
GLdouble dist;

GLdouble z1 = Source.X->value();
GLdouble z2 = Source.Y->value();

GLint w = GLgl_Window->w();

GL_Plane & plane = parameters.planes[M_PLANE1];

GLdouble cs = cos(plane.parameters[P_THETA]->value() * GRAD);
GLdouble ss = sin(plane.parameters[P_THETA]->value() * GRAD);
```

```
GLdouble dp = step_ * 2. * bound / (double) w;

GLdouble R2 = parameters.parameters[P_R]->value();

R2 *= R2;

for (i = 0; i < NOF_PLANES; i++) {
init_model(i, current_model[i]);
}

glPointSize(step_ * M_SQRT2);

if (int(step_) == 1)
glEnable(GL_POINT_SMOOTH);
else
glDisable(GL_POINT_SMOOTH);

if (source_function == F_DATAFILE) {
im = Imlib_load_image(ImlibId, ImageFilename);
fx = fy = -SIZE_0 / (2. * BOUND_0);
}
glColor3f(1.0, 1.0, 1.0);

glPushMatrix();

glBegin(GL_POINTS);
for (x1 = xmin_; x1 <= xmax_; x1 += dp) {
for (x2 = ymin_; x2 <= ymax_; x2 += dp) {

x1bar = x1 * cs + x2 * ss;
x2bar = -x1 * ss + x2 * cs;

lens_equation(current_model[0], 0, x1bar, x2bar,
     &y1, &y2); // FIX ME!!

// y1 = x1bar;
// y2 = x2bar;

//y1 += z1;
//y2 += z2;

if (source_function == F_DATAFILE) {
int xp =
    (int) ((x1 - z1) * fx) +
    im->rgb_width / 2;
int yp =
    (int) ((x2 - z2) * fy) +
```

```
    im->rgb_height / 2;

if ((xp >= 0) && (xp < im->rgb_width) &&
    (yp >= 0) && (yp < im->rgb_height)) {
                                        draw_pixel (im->rgb_data +
    3 * (yp *
 im->rgb_width +
 xp),
                                                    y1, y2);
}
} else {
dist =
    (x1 - z1) * (x1 - z1) + (x2 -
     z2) * (x2 -
    z2);

if (dist <= R2) {
if (source_function == F_GRAYSCALE) {
colormap(x1 - z1, x2 - z2,
 color);
glColor3fv(color);
}
glVertex2d(y1, y2);
}
}
}
glEnd();

glPopMatrix();

if (source_function == F_DATAFILE)
Imlib_destroy_image(ImlibId, im);
}

int GL_Drawing_Area::save_image(char * filename) {
        // there's some alignment wonky honky going on here, for some reason
        // ReadPixels won't work if the image is w x h
        if (!filename) return 0;
        ImlibImage * image =
                Imlib_create_image_from_drawable
                (ImlibId, RootWindow(fl_display,fl_screen), 0,
                 0, 0,
                 w()+1, h());
        glReadPixels (0, 0, w(), h(),
                      GL_RGB, GL_UNSIGNED_BYTE,
                     (GLvoid *)image->rgb_data);
```

```
        Imlib_flip_image_vertical (ImlibId, image);
        Imlib_crop_image (ImlibId, image, 0, 0, w(), h());
        Imlib_render (ImlibId, image,
                      image->rgb_width,
                      image->rgb_height);
        Imlib_save_image (ImlibId, image, filename, 0);
        Imlib_kill_image (ImlibId, image);
        return 1;
}

#if 0
void draw_photo(void)
{
glBegin(GL_POINTS);
for (x1 = -x0; x1 <= x0; x1 += dpx) {
for (x2 = -x0; x2 <= x0; x2 += dpy) {
x1bar = x1 * cs + x2 * ss;
x2bar = -x1 * ss + x2 * cs;
lens_equation(x1bar, x2bar, &y1, &y2);

y1 -= z1;
y2 -= z2;

if ((y1 <= -x0) || (y1 >= x0) || (y2 <= -x0)
    || (y2 >= x0)) {
glColor3f(0.0, 0.0, 0.0);
} else {
glColor3ubv(im->rgb_data +
    3 * ((int) ((-y2 + x0) * fy) *
 im->rgb_width +
 (int) ((y1 + x0) * fx)));
}

glVertex2d(x1, x2);
}
}
glEnd();
}

#endif

void i2world (double x, double y,
              double *tx, double *ty,
              void * pltr_data) {
        double * data = (double *) pltr_data;
        *tx = data[0] * x * data[4] + data[2];
        *ty = data[1] * y * data[4] + data[3];
```

```
}

void i2source (double x, double y,
               double *tx, double *ty,
               void * pltr_data) {
        map * LensMap = (map *) pltr_data;
        int i = int(x);
        int j = int(y);
        * tx = LensMap -> Map[i][j].y1;
        * ty = LensMap -> Map[i][j].y2;
}

void draw_pixel (const GLubyte * colorv, double x, double y) {
     if (*colorv || *(colorv + 1) || *(colorv + 2)) {
          glColor3ubv(colorv);
          glVertex2d(x, y);
     }
}
```

B.4 Magnificación, **Mag_Drawing_Area.C**

```
#include <stream.h>
#include <math.h>

#include <Mag_Drawing_Area.H>

#include <GL.H>
#include <GLcp.H>
#include <GLcp_ui.H>
#include <GL/glu.h>
#include "trackball.h"

void colormap (double);
void vertex (double, double, double);

void Mag_Drawing_Area::lensmap (map * v) {
        LensMap = v;
}

void Mag_Drawing_Area::jacmatrix (jacobian_matrix * v) {
        JacMatrix = v;
}

void Mag_Drawing_Area::draw()
```

```
{
        float m[4][4];

if (!valid()) {
xmin_ = -2.; xmax_ =  2.;
ymin_ = -2.; ymax_ =  2.;
                z_ = -5.;
                trackball(curquat, 0.0, 0.0, 0.0, 0.0);

                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();
                gluPerspective(60, 1., .01, -2.);
                glEnable (GL_DEPTH_TEST);
                glEnable (GL_AUTO_NORMAL);
                glEnable (GL_NORMALIZE);
                glShadeModel(GL_FLAT);

valid(1);
}

        glClearColor(0., 0., 0., 0.);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glPushMatrix ();
        build_rotmatrix(m, curquat);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef(0, 0, z_);
        glMultMatrixf(&(m[0][0]));
        glRotatef(180, 0, 0, 1);

        magnification_map ();

        glPopMatrix ();
}

int Mag_Drawing_Area::handle(int event)
{
int event_x = Fl::event_x();
int event_y = Fl::event_y();

switch (event) {
case FL_PUSH:
mousex_ = event_x;
mousey_ = event_y;
return 1;
case FL_DRAG:
```

```
                trackball(lastquat,
                        (2.*mousex_ - w()) / w(),
                        (h() - 2.*mousey_) / h(),
                        (2.*event_x - w()) / w(),
                        (h() - 2.*event_y) / h());
                add_quats(lastquat, curquat, curquat);
mousex_ = event_x;
mousey_ = event_y;
                redraw();
return 0;
case FL_RELEASE:
return 1;
case FL_KEYBOARD:
return 1;
default:
                return 0; //Fl_Gl_Window::handle (event);
}

        return 0;
}

void Mag_Drawing_Area::magnification_map () {
        double ** magnification;

        if (make_jacmatrix ()) {
        int & w = JacMatrix -> w;
        int & h = JacMatrix -> h;
        int & k = JacMatrix -> step;

        int wp = w / k;
        int hp = h / k;

        magnification = new double*[wp];
        for (int i = 0; i < wp; i++)
                magnification[i] = new double[hp];

        for (int i = 1; i < wp; i++)
                for (int j = 1; j < hp; j++)
                        magnification[i][j] =
                                1. / JacMatrix -> A[i][j];

        if (!(glIsList (maglist))) {
                maglist = glGenLists (1);
        }

        glNewList (maglist, GL_COMPILE_AND_EXECUTE);
        glBegin(GL_QUADS);
```

```
for (int i = 1 ; i < wp - 1 ; i++) {
        for (int j = 1 ; j < hp - 1 ; j++) {
                colormap (magnification[i][j]);
                vertex (LensMap -> Map[i][j].x1,
                        LensMap -> Map[i][j].x2,
                        magnification[i][j]);

                colormap(magnification[i+1][j]);
                vertex (LensMap -> Map[i+1][j].x1,
                        LensMap -> Map[i+1][j].x2,
                        magnification[i+1][j]);

                colormap(magnification[i+1][j+1]);
                vertex (LensMap -> Map[i+1][j+1].x1,
                        LensMap -> Map[i+1][j+1].x2,
                        magnification[i+1][j+1]);

                colormap(magnification[i][j+1]);
                vertex (LensMap -> Map[i][j+1].x1,
                        LensMap -> Map[i][j+1].x2,
                        magnification[i][j+1]);
        }
}
glEnd();
                glEndList ();

                for (int i = 0; i < wp; i++)
                        delete magnification[i];
                delete magnification;
        } else
                glCallList (maglist);
}

void colormap (double z) {
        if (z < 0) z = -z;
        if (z > 1.) {
                glColor3d (1., 1/z, 1/z);
        } else glColor4d (z, z, 1., .5);
}

void vertex (double x, double y, double z) {
        if (z > 2.) z = 2.;
        else if (z < -2.) z = -2.;
        glVertex3d(x, y, z);
}
```

B.5 Modelos, `GL_models.C`

```
#include <stream.h>
#include <GL.H>
#include <GL_models.H>
#include <math.h>

// Ok boys and girls, the name of the game is speed and speed can only
// be achieved by cutting slack... that's what the init functions are
// for (didn't you pay attention to your English teacher? never end a
// sentence with a preposition!)
//
// The idea is that you put all the static-same-for-every-point,
// calculations in the init function, do it ONCE (the calculation I
// mean, smartguy) and that's it. The other calculations, the lenghty
// ones, use the precomputed values.
//
// That was rule number one. Number two is NEVER perform a division
// unless necessary, that means, if the precomputed values are going
// to be used for example in this way:
//
//          butt_ugly_expression_here / some_constant
//
// you do:
//
//          isome_constant = 1./some_constant;
//
// in the init function, and you use:
//
//          butt_ugly_expression_here * isome_constant
//
// Taking into account that normally you'll end up evaluating the same
// function about 30k times. That's 30k divisions. 2 divisions is one
// too many. 30k is a shitload of divisions!
//
// And whatever the heck you do, PLEASE DOCUMENT IT! CVS allows you to
// go back and check older versions to find out where you screwed up;
// that's it. It's not a substitute for documentation.

#define tau  7.67
#define tau2 58.8289                    // tau^2
#define tau3 451.217663                 // tau^3
```

```
#define tau4 3460.83947521                  // tau^4
#define tau5 26544.6387748607               // tau^5
#define tau6 203597.379403181569            // tau^6
#define tau7 1561591.90002240263423         // tau^7
#define tau8 11977409.8731718282045441      // tau^8
#define etau 2143.081445247758682998888     // e^tau

typedef void (*Model_function) (double, double, double *, double *);
typedef double (*Potential_function) (double, double);
typedef void (*Model_init_function) (int);

void init_chang_refsdal(int);
void init_sis(int);
void init_nsis(int);
void init_transparent_sphere(int);
void init_elliptical(int);
void init_king(int);
void init_truncated_king(int);
void init_hubble(int);
void init_de_vaucouleur(int);
void init_spiral(int);
void init_multipole(int);
void init_rotation_lens(int);
void init_double_lens(int);

void chang_refsdal(double, double, double *, double *);
void sis(double, double, double *, double *);
void nsis(double, double, double *, double *);
void transparent_sphere(double, double, double *, double *);
void elliptical(double, double, double *, double *);
void king(double, double, double *, double *);
void truncated_king(double, double, double *, double *);
void hubble(double, double, double *, double *);
void de_vaucouleur(double, double, double *, double *);
void spiral(double, double, double *, double *);
void multipole(double, double, double *, double *);
void rotation_lens(double, double, double *, double *);
void double_lens(double, double, double *, double *);

double chang_refsdal_potential (double, double);

Model_function models[] = {
        NULL,                       // none
        chang_refsdal,
        sis,
        nsis,
        transparent_sphere,
```

```
        elliptical,
        king,
        truncated_king,
        hubble,
        de_vaucouleur,
        spiral,
        multipole,
        rotation_lens,
        double_lens
};

Potential_function potentials[] = {
        NULL,
        chang_refsdal_potential,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL
};

Model_init_function models_init[] = {
NULL,// none
        NULL,                   // init_chang_refsdal
        init_sis,
        init_nsis,
        init_transparent_sphere,
        init_elliptical,
        init_king,
        init_truncated_king,
        init_hubble,
        init_de_vaucouleur,
        init_spiral,
        NULL,                   // init_multipole
        NULL,                   // init_rotation_lens
        NULL                    // init_double_lens
};

// instantiate parameters here. It's only used here and in GLcp_ui.C
```

```
GL_Parameters parameters;
GL_ParametersV prms;

static double a[NOF_PLANES], b[NOF_PLANES], c[NOF_PLANES];
static double R_E, R_E2, iR_E, iR_E2;
static double R_C[NOF_PLANES], R_C2[NOF_PLANES], iR_C2[NOF_PLANES];
static double ir_E2;
static double A1[NOF_PLANES], A2[NOF_PLANES], D[NOF_PLANES];
static double Ep1[NOF_PLANES], Ep2[NOF_PLANES], delta[NOF_PLANES];
static int cp; // uglier!

void init_model(int current_plane, int current_model)
{
R_E = prms.plane[cp].prms[P_E];
R_E2 = R_E * R_E;

iR_E = 1. / R_E;
iR_E2 = 1. / R_E2;

cp = current_plane;

A1[cp] = 1. - prms.plane[cp].prms[P_SIGMA]
              - prms.plane[cp].prms[P_GAMMA]
              * cos(prms.plane[cp].prms[P_PHI] * GRAD);
A2[cp] = 1. - prms.plane[cp].prms[P_SIGMA]
              + prms.plane[cp].prms[P_GAMMA]
              * cos(prms.plane[cp].prms[P_PHI] * GRAD);
D[cp] = -prms.plane[cp].prms[P_GAMMA]
              * sin(prms.plane[cp].prms[P_PHI] * GRAD);

if (models_init[current_model])
models_init[current_model] (current_plane);
}

void lens_equation(int model, int plane,
   double x1, double x2,
   double *y1, double *y2)
{
*y1 = *y2 = 0.;

if (models[model])
models[model] (x1, x2, y1, y2);

*y1 += A1[plane] * x1 +  D[plane] * x2;
*y2 +=  D[plane] * x1 + A2[plane] * x2;
}
```

```
double potential(int model, double x1, double x2) {
        if (potentials[model])
                return potentials[model] (x1, x2);
        else
                return 0.;
}

void init_chang_refsdal(int) {
// nothing to do
}

void chang_refsdal(double x1, double x2, double *y1, double *y2) {
ir_E2 = R_E2 / (x1 * x1 + x2 * x2);
*y1 = -(x1 * ir_E2);
*y2 = -(x2 * ir_E2);
}

double chang_refsdal_potential (double x1, double x2) {
        return (log(iR_E*sqrt(x1*x1 + x2*x2)));
}

void init_double_lens(int) {
// nothing to do
}
```

```
void double_lens(double x1, double x2, double *y1, double *y2) {
double irs12, irs22;
double xd1, xd2;
double ird2;

double dx11, dx21, dx12, dx22;

dx11 = x1 - prms.plane[cp].mprms[P_DOUBLE_LENS_X1];
dx21 = x2 - prms.plane[cp].mprms[P_DOUBLE_LENS_Y1];
dx12 = x1 - prms.plane[cp].mprms[P_DOUBLE_LENS_X2];
dx22 = x2 - prms.plane[cp].mprms[P_DOUBLE_LENS_Y2];

irs12 = R_E2 / (dx11 * dx11 + dx21 * dx21);
irs22 = R_E2 / (dx12 * dx12 + dx22 * dx22);

a[cp] = prms.plane[cp].mprms[P_DOUBLE_LENS_BETA] * irs12;

xd1 =
    x1 - prms.plane[cp].mprms[P_DOUBLE_LENS_M1] * a[cp] * dx11;
xd2 =
    x2 - prms.plane[cp].mprms[P_DOUBLE_LENS_M1] * a[cp] * dx22;
```

```
ird2 = R_E2 / (xd1 * xd1 + xd2 * xd2);

*y1 =
    -(prms.plane[cp].mprms[P_DOUBLE_LENS_M1] * dx11 * irs12 -
      prms.plane[cp].mprms[P_DOUBLE_LENS_M2] *
      (xd1 - prms.plane[cp].mprms[P_DOUBLE_LENS_X2]) * irs22);
*y2 =
    -(prms.plane[cp].mprms[P_DOUBLE_LENS_M1] * dx21 * irs12 -
      prms.plane[cp].mprms[P_DOUBLE_LENS_M2] *
      (xd2 - prms.plane[cp].mprms[P_DOUBLE_LENS_Y2]) * ird2);

// CHECK ME: that's fishy -> irs22 but ird2?
//           also xd2 contains M_1 not M_2
}

void init_sis(int) {
a[cp] = 1.0 - 0.5 * prms.plane[cp].mprms[P_SIS_KAPPA];
}

void sis(double x1, double x2, double *y1, double *y2) {
double ir_E2 = R_E2 / (x1 * x1 + x2 * x2);

double rbeta = pow(ir_E2, a[cp]);

*y1 = -(rbeta * x1 * ir_E2);
*y2 = -(rbeta * x2 * ir_E2);
}

void init_nsis(int) {
R_C[cp] = prms.plane[cp].mprms[P_NSIS_C];
R_C2[cp] = R_C[cp] * R_C[cp];
}

void nsis(double x1, double x2, double *y1, double *y2) {
double s2 = R_C2[cp] + (x1 * x1) + (x2 * x2);
a[cp] = sqrt(s2) - R_C[cp];

*y1 = -(R_E * a[cp] * x1 / s2);
*y2 = -(R_E * a[cp] * x2 / s2);

// CHECK ME: Is that R_C2 or something else in s2?
}

void init_transparent_sphere(int) {
R_C[cp] = prms.plane[cp].mprms[P_TRANSP_SPHERE_C];
iR_C2[cp] = 1. / (R_C[cp] * R_C[cp]);
}
```

```
void transparent_sphere(double x1, double x2, double *y1, double *y2) {
double r2 = (x1 * x1) + (x2 * x2);
double ir_E2 = R_E2 / r2;

if (R_C[cp] > r2) {
double s2 = 1. - r2 * iR_C2[cp];
a[cp] = 1. - pow(s2, (3. / 2.));

*y1 = -(a[cp] * x1 * ir_E2);
*y2 = -(a[cp] * x2 * ir_E2);
} else {
*y1 = -(x1 * ir_E2);
*y2 = -(x2 * ir_E2);
}

// CHECK ME: R_C compared to r2
}

void init_elliptical(int) {
Ep1[cp] = 1. - prms.plane[cp].mprms[P_ELLIPTICAL_EPSILON];
Ep2[cp] = 1. + prms.plane[cp].mprms[P_ELLIPTICAL_EPSILON];
delta[cp] = prms.plane[cp].mprms[P_ELLIPTICAL_ALPHA] - 1.;

R_C2[cp] = prms.plane[cp].mprms[P_ELLIPTICAL_C];
R_C2[cp] *= R_C2[cp];
iR_C2[cp] = 1. / R_C2[cp];
}

void elliptical(double x1, double x2, double *y1, double *y2) {
double s2 = (R_C2[cp] + Ep1[cp] * x1 * x1 + Ep2[cp] * x2 * x2);
a[cp] = pow(s2 * iR_C2[cp], delta[cp]);

*y1 = -(prms.plane[cp].mprms[P_ELLIPTICAL_KAPPA] *
                Ep1[cp] * x1 * a[cp]);
*y2 = -(prms.plane[cp].mprms[P_ELLIPTICAL_KAPPA] *
                Ep2[cp] * x2 * a[cp]);

// CHECK ME: simplification too damm easy to be overlooked
}

void init_king(int) {
a[cp] = prms.plane[cp].mprms[P_KING_KAPPA] * R_E2;
}

void king(double x1, double x2, double *y1, double *y2) {
double r2 = (x1 * x1) + (x2 * x2);
```

```
double r_E2 = r2 * iR_E2;

b[cp] = log(1.0 + r_E2) / r2;

*y1 = -(a[cp] * x1 * b[cp]);
*y2 = -(a[cp] * x2 * b[cp]);
}

void init_truncated_king(int) {
a[cp] = prms.plane[cp].mprms[P_TRUNC_KING_KAPPA] * R_E2;

R_C[cp] = prms.plane[cp].mprms[P_TRUNC_KING_C];
b[cp] = 1. / (1. + R_C[cp] * R_C[cp] * iR_E2);
}

void truncated_king(double x1, double x2, double *y1, double *y2) {
double r2 = (x1 * x1) + (x2 * x2);
double r_E2 = r2 * iR_E2;
double ir2 = 1. / r2;

c[cp] = log(1.0 + r_E2) + r_E2 * b[cp] +
    4. * (1. - sqrt(1.0 + r_E2)) * sqrt(b[cp]);

*y1 = -(a[cp] * x1 * c[cp] * ir2);
*y2 = -(a[cp] * x2 * c[cp] * ir2);

// CHECK ME: fist c is possibly wrong, it may be
//           (log(1.0 + r_E2) + r_E2) * b;
}

void init_hubble(int) {
a[cp] = 2. * prms.plane[cp].mprms[P_HUBBLE_KAPPA] * R_E2;
}

void hubble(double x1, double x2, double *y1, double *y2) {
double r2 = (x1 * x1) + (x2 * x2);
double ir2 = 1. / r2;
double r_E = sqrt(r2) * iR_E;

b[cp] = log(1.0 + r_E) - r_E / (1.0 + r_E);

*y1 = -(a[cp] * x1 * b[cp] * ir2);
*y2 = -(a[cp] * x2 * b[cp] * ir2);
}

void init_de_vaucouleur(int) {
a[cp] = 8.0 * prms.plane[cp].mprms[P_DE_VACOULEUR_KAPPA] * R_E2;
```

```
}

void de_vaucouleur(double x1, double x2, double *y1, double *y2) {
double r2 = (x1 * x1) + (x2 * x2);
double ir2 = 1. / r2;
double r_E = sqrt(r2) * iR_E;

b[cp] =
    (5040. / tau8
     + 5040. * pow(r_E, 1. / 4.) / tau7
     + 2520. * sqrt(r_E) / tau6
     + 840. * pow(r_E, 3. / 4.) / tau5
     + 210. * r_E / tau4
     + 42. * pow(r_E, 5. / 4.) / tau3
     + 7. * pow(r_E, 6. / 4.) / tau2
     + pow(r_E, 7. / 4.) / tau)
    * exp(tau * (1.0 - pow(r_E, 1. / 4.)))
    - 5040.0 * etau / tau8;

*y1 = a[cp] * x1 * b[cp] * ir2;
*y2 = a[cp] * x2 * b[cp] * ir2;
}

void init_spiral(int) {
a[cp] = prms.plane[cp].mprms[P_SPIRAL_KAPPA] * R_E2;
}

void spiral(double x1, double x2, double *y1, double *y2) {
double r2 = (x1 * x1) + (x2 * x2);
double ir2 = 1. / r2;
double r_E = sqrt(r2) * iR_E;

b[cp] = 1.0 - exp(-r_E) * (r_E + 1.0);

*y1 = -(x1 * a[cp] * b[cp] * ir2);
*y2 = -(x2 * a[cp] * b[cp] * ir2);
}

void init_multipole(int) {
// nothing to do
}

void multipole(double x1, double x2, double *y1, double *y2) {
double x12 = x1 * x1;
double x22 = x2 * x2;

double dif = x22 - x12;
```

```
ir_E2 = R_E2 / (x12 + x22);
double ir_E4 = ir_E2 * ir_E2;
double i2r_E6 = 2. * ir_E2 * ir_E4;

a[cp] = x2 * (x22 - 3. * x12);
b[cp] = x1 * (3. * x22 - x12);
c[cp] = 2. * x1 * x2;

double dipol1 = ir_E4 * (
                prms.plane[cp].mprms[P_MULTIPOLE_DX] * dif
                - c[cp] * prms.plane[cp].mprms[P_MULTIPOLE_DY]
                );
double dipol2 = ir_E4 * (
                prms.plane[cp].mprms[P_MULTIPOLE_DY] * dif
                + c[cp] * prms.plane[cp].mprms[P_MULTIPOLE_DX]
                );

double quadp12 = i2r_E6 * (
                -prms.plane[cp].mprms[P_MULTIPOLE_Q1] * a[cp]
                + prms.plane[cp].mprms[P_MULTIPOLE_Q2] * b[cp]
                );
double quadp22 = i2r_E6 * (
                -prms.plane[cp].mprms[P_MULTIPOLE_Q1] * b[cp]
                + prms.plane[cp].mprms[P_MULTIPOLE_Q2] * a[cp]
                );

*y1 = -(x1 * ir_E2 + dipol1 + quadp12);
*y2 = -(x2 * ir_E2 - dipol2 - quadp22);

// CHECK ME: every x1 has one of this ir_E2; check if it's possible
//           to move them to x1. would save 6 *'s in this case
}

void init_rotation_lens(int) {
// nothing ... cut it out already
}

void rotation_lens(double x1, double x2, double *y1, double *y2) {
double ir_E2 = R_E2 / (x1 * x1 + x2 * x2);
double ir_E4 = ir_E2 * ir_E2;

a[cp] = prms.plane[cp].mprms[P_ROTATION_SX] * x2
    - prms.plane[cp].mprms[P_ROTATION_SY] * x1;

*y1 = -((x1 + prms.plane[cp].mprms[P_ROTATION_SY]) * ir_E2
+ (2.0 * a[cp] * x1) * ir_E4);
*y2 = -((x2 - prms.plane[cp].mprms[P_ROTATION_SX]) * ir_E2
```

```
+ (2.0 * a[cp] * x2) * ir_E4);
}
```

B.6 Interfaz gráfica, GLcp_ui.fl

```
# data file for the Fltk User Interface Designer (fluid)
version 1.00
header_name {.H}
code_name {.C}
gridx 10
gridy 10
snap 3
Function {make_panels()} {} {
  Fl_Window RenderWindow {
    label GL
    callback {ShowRenderWindow->value(0);
o->hide();
GLgl_Window->hide();}
    xywh {98 290 480 519} box UP_BOX resizable
    code0 {\#include <GLcp.H>} visible
  } {
  Fl_Group GLgl_Window {
    xywh {40 40 400 400} box DOWN_BOX color 0 resizable
    code0 {\#include <GL_Drawing_Area.H>}
    class GL_Drawing_Area
  } {}
  Fl_Group {} {
    xywh {40 450 420 60}
  } {
    Fl_Light_Button {glswitch[S_SOURCE]} {
      label Source
      user_data {1 << S_SOURCE} user_data_type long
      callback cb_switches
      xywh {40 450 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
    }
    Fl_Light_Button {glswitch[S_IMAGES]} {
      label Images
      user_data {1 << S_IMAGES} user_data_type long
      callback cb_switches
      xywh {40 470 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
    }
    Fl_Light_Button {glswitch[S_DATA]} {
      label Data
      user_data {1 << S_DATA} user_data_type long
```

```
  callback cb_switches
  xywh {140 490 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
}
Fl_Light_Button {glswitch[S_GRID]} {
  label Grid
  user_data {1 << S_GRID} user_data_type long
  callback cb_switches
  xywh {140 470 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
}
Fl_Light_Button {glswitch[S_EINSTEIN]} {
  label R_E
  user_data {1 << S_EINSTEIN} user_data_type long
  callback cb_switches
  xywh {40 490 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
}
Fl_Button {} {
  label Redraw
  callback {GLgl_Window->redraw();}
  xywh {350 450 90 20} box ROUND_UP_BOX
}
Fl_Light_Button {glswitch[S_DELAY]} {
  label {Time Delay}
  user_data {1 << S_DELAY} user_data_type long
  callback cb_switches
  xywh {140 450 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
  }
  Fl_Button {} {
    label Save
    callback {char * filename = fl_file_chooser("Save image...", 0, 0);
GLgl_Window -> save_image (filename);}
    xywh {350 480 90 20} box ROUND_UP_BOX
    code0 {\#include <FL/fl_file_chooser.h>}
  }
  Fl_Value_Slider precision {
    user_data 0 user_data_type long
    callback cb_precision
    xywh {240 480 100 20} type {Horz Knob} box ROUND_DOWN_BOX when 4
  minimum 1 maximum 10 step 1 value 4
  }
  Fl_Light_Button {glswitch[6]} {
    label Critical
    user_data {1 << S_CRITICAL} user_data_type long
    callback cb_switches selected
    xywh {240 450 90 20} box NO_BOX down_box ROUND_UP_BOX labelsize 10
  }
}
Fl_Slider PanY {
```

```
      callback {remake_map();}
      xywh {10 40 20 400} box ROUND_DOWN_BOX minimum 2 maximum -2
    }
    Fl_Slider PanX {
      callback {remake_map();}
      xywh {40 10 400 20} type Horizontal box ROUND_DOWN_BOX minimum -2 maximum 2
    }
    Fl_Button {} {
      callback {PanX->value(0.0);
PanY->value(0.0);
ZoomFactor->value(1.0);
remake_map();}
      xywh {10 10 20 20} box ROUND_UP_BOX
    }
    Fl_Slider ZoomFactor {
      callback {remake_map();}
      xywh {450 40 20 400} type {Vert Knob} box ROUND_DOWN_BOX minimum 10 maximum 0.1
    step 0.001 value 1
    }
  }
  Fl_Window ControlPanel {
    label GLcp
    xywh {568 184 419 239}
    code0 {\#include "GL.H"} visible
} {
  Fl_Menu_Bar {} {
    xywh {0 0 550 30}
    code0 {\#include "GL.H"}
  } {
    submenu {} {
      label File open
      xywh {0 0 100 20}
    } {
      menuitem {} {
        label Open
        user_data MI_OPEN user_data_type long
        callback cb_menu
        xywh {0 0 100 20} shortcut 0x8006f
      }
      menuitem {} {
        label Save
        user_data MI_SAVE user_data_type long
        callback cb_menu
        xywh {0 0 100 20} shortcut 0x80073
      }
      menuitem {} {
        label Quit
```

```
    user_data MI_QUIT user_data_type long
    callback cb_menu
    xywh {0 0 100 20} shortcut 0x80071
  }
}
submenu {} {
  label Source open
  xywh {0 0 100 20}
} {
  menuitem {} {
    label {Filled Circle}
    user_data F_SOLID user_data_type long
    callback cb_source_function
    xywh {10 10 100 20} type Radio value 1
  }
  menuitem {} {
    label Grayscale
    user_data F_GRAYSCALE user_data_type long
    callback cb_source_function
    xywh {10 10 100 20} type Radio
  }
  menuitem {} {
    label Datafile
    user_data F_DATAFILE user_data_type long
    callback cb_source_function
    xywh {10 10 100 20} type Radio
  }
  menuitem {} {
    label {Choose datafile}
    user_data MI_CHOOSEDATA user_data_type long
    callback cb_menu
    xywh {0 0 100 20}
  }
}
menuitem {} {
  label Help
  xywh {0 0 100 20} shortcut 0x68
}
}
Fl_Group {} {
  label {Source Position}
  xywh {10 40 230 150} box ENGRAVED_BOX labelsize 12 align 6
} {
  Fl_Group {Source.Positioner} {
    user_data {&Source}
    callback cb_source_positioner
    xywh {30 50 110 100} box DOWN_BOX color 0 align 0 when 1
```

```
  code0 {\#include "GL_Positioner.H"}
  code1 {o->xbounds(-4,4);}
  code2 {o->ybounds(-4,4);}
  code3 {o->step(0.01,0.01);}
  class Positioner_Window
} {}
Fl_Value_Input {Source.X} {
  label {x:}
  user_data {&Source}
  callback cb_source_input
  xywh {30 160 50 20} minimum -10 maximum 10 textsize 12
}
Fl_Value_Input {Source.Y} {
  label {y:}
  user_data {&Source}
  callback cb_source_input
  xywh {100 160 50 20} minimum -10 maximum 10 textsize 12
}
Fl_Value_Slider {parameters.parameters[P_R]} {
  label R
  callback {GLgl_Window->redraw();}
  xywh {160 50 30 120} box ENGRAVED_BOX minimum 2 maximum 0 step 0.01 value 1
  code0 {\#include <GL_models.H>}
}
  Fl_Value_Slider {parameters.parameters[P_THETA]} {
    label q
    callback {GLgl_Window->redraw();}
    xywh {200 50 30 120} box ENGRAVED_BOX labelfont 12 minimum 360 maximum 0 step 1
    code0 {\#include <GL_models.H>}
  }
}
Fl_Group {} {
  label {Image Positions}
  xywh {250 40 160 150} box ENGRAVED_BOX labelsize 12 align 6
} {
  Fl_Browser observed_images_list {
    user_data source_pos
    callback cb_images_locations_sel
    xywh {260 50 140 70} type Hold box ENGRAVED_BOX selection_color 49 labelsize 10
  textfont 4 textsize 12
  }
  Fl_Value_Input {source_pos[0]} {
    label {x:}
    user_data observed_images_list
    callback cb_image_pos_x
    xywh {280 130 50 20} minimum -10 maximum 10 textsize 12
  }
```

```
      Fl_Value_Input {source_pos[1]} {
        label {y:}
        user_data observed_images_list
        callback cb_image_pos_y
        xywh {350 130 50 20} minimum -10 maximum 10 textsize 12
      }
      Fl_Button {} {
        label Clear
        user_data observed_images_list user_data_type {void*}
        callback cb_image_pos_clear
        xywh {260 160 50 20} box ENGRAVED_BOX labelsize 12
      }
      Fl_Button {} {
        label {Clear All}
        user_data observed_images_list user_data_type {void*}
        callback cb_image_pos_clear_all
        xywh {320 160 80 20} box ENGRAVED_BOX labelsize 12
      }
    }
    Fl_Button ShowRenderWindow {
      label {Render Window}
      callback {if (o->value()) {
   RenderWindow->show();
   GLgl_Window->show();
} else {
   RenderWindow->hide();
   GLgl_Window->hide();
}}
      xywh {10 210 140 20} type Toggle box THIN_UP_BOX down_box THIN_DOWN_BOX

   shortcut 0x72
   value 1 labelsize 12
    }
    Fl_Button ShowMagnificationWindow {
      label {Magnification Window}
      callback {if (o->value()) {
   MagnificationWindow->show();
   GLmag_Window->show();
} else {
   MagnificationWindow->hide();
   GLmag_Window->hide();
}}
      xywh {160 210 140 20} type Toggle box THIN_UP_BOX down_box THIN_DOWN_BOX

   shortcut 0x72
   labelsize 12
    }
```

```
  }
  Fl_Window MagnificationWindow {
    label Magnification
    callback {ShowMagnificationWindow->value(0);
o->hide();}
    xywh {113 186 450 450} hide
  } {
    Fl_Group GLmag_Window {
      xywh {10 10 400 400} box DOWN_BOX color 0 selection_color 50
      code0 {\#include <Mag_Drawing_Area.H>}
      class Mag_Drawing_Area
    } {}
    Fl_Button {} {
      label Redraw
      callback {GLmag_Window->redraw();}
      xywh {310 420 100 20} box ROUND_UP_BOX
    }
    Fl_Slider MagZoom {
      callback {GLmag_Window -> zoom(o->value());
GLmag_Window -> redraw ();}
      xywh {420 10 20 400} type {Vert Knob} box ROUND_DOWN_BOX minimum 10 maximum 0
    step 0.001 value 6
    }
  }
}

Function {make_plane_cp(int plane)} {} {
  Fl_Window {} {
    label {Plane Parameters}
    xywh {298 280 449 179} when 0 hide
  } {
    Fl_Choice {model_menu[plane]} {
      label {Model:}
      user_data plane user_data_type long
      callback cb_select_model
      xywh {210 10 200 20} labelsize 12 textsize 12
    } {
      menuitem {} {
        label None
        xywh {10 10 100 20}
      }
      menuitem {} {
        label {Chang Refsdal}
        xywh {50 50 100 20} labelsize 12
      }
      menuitem {} {
        label {Singular Isothermal Sphere}
```

```
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label {Nonsingular Isothermal Sphere}
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label {Transparent Sphere}
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label Elliptical
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label King
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label {Truncated King}
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label Hubble
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label {De Vaucouler}
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label Spiral
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label Multipole
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label Rotation
    xywh {50 50 100 20} labelsize 12
  }
  menuitem {} {
    label {Double Pointmass}
    xywh {50 50 100 20} labelsize 12
  }
}
```

```
Fl_Value_Slider {parameters.planes[plane].parameters[P_E]} {
  label E
  callback {remake_map();}
  xywh {10 40 30 120} box ENGRAVED_BOX minimum 4 maximum 0 step 0.01 value 1
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].parameters[P_GAMMA]} {
  label g
  callback {remake_map();}
  xywh {90 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 1 maximum 0 step 0.01
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].parameters[P_SIGMA]} {
  label s
  callback {remake_map();}
  xywh {50 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 1 maximum 0 step 0.01
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].parameters[P_PHI]} {
  label f
  callback {remake_map();}
  xywh {130 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 360 maximum 0 step 1
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_SIS].parameters[P_SIS_KAPPA]} {
  label k
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_NSIS].parameters[P_NSIS_C]} {
  label C
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_TRANSP_SPHERE].
parameters[P_TRANSP_SPHERE_C]} {
  label C
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_ELLIPTICAL].
```

```
parameters[P_ELLIPTICAL_KAPPA]} {
  label k
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_ELLIPTICAL].
parameters[P_ELLIPTICAL_C]} {
  label C
  callback {remake_map();}
  xywh {210 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_ELLIPTICAL].
parameters[P_ELLIPTICAL_EPSILON]} {
  label e
  callback {remake_map();}
  xywh {250 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 1 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_ELLIPTICAL].
parameters[P_ELLIPTICAL_ALPHA]} {
  label a
  callback {remake_map();}
  xywh {290 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 1 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_KING].parameters[P_KING_KAPPA]} {
  label k
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_KING].parameters[P_KING_C]} {
  label C
  callback {remake_map();}
  xywh {210 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_TRUNC_KING].
parameters[P_TRUNC_KING_KAPPA]} {
  label k
  callback {remake_map();}
```

```
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_TRUNC_KING].
parameters[P_TRUNC_KING_C]} {
  label C
  callback {remake_map();}
  xywh {210 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_HUBBLE].
parameters[P_HUBBLE_KAPPA]} {
  label k
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01

hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DE_VACOULEUR].

parameters[P_DE_VACOULEUR_KAPPA]} {
  label k
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_SPIRAL].
parameters[P_SPIRAL_KAPPA]} {
  label k
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 2 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_MULTIPOLE].
parameters[P_MULTIPOLE_DX]} {
  label dx
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_MULTIPOLE].
parameters[P_MULTIPOLE_DY]} {
  label dy
```

```
  callback {remake_map();}
  xywh {210 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_MULTIPOLE].
parameters[P_MULTIPOLE_Q1]} {
  label Q1
  callback {remake_map();}
  xywh {250 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_MULTIPOLE].
parameters[P_MULTIPOLE_Q2]} {
  label Q2
  callback {remake_map();}
  xywh {290 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_ROTATION_LENS].
parameters[P_ROTATION_SX]} {
  label Sx
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_ROTATION_LENS].
parameters[P_ROTATION_SY]} {
  label Sy
  callback {remake_map();}
  xywh {210 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
parameters[P_DOUBLE_LENS_M1]} {
  label m1
  callback {remake_map();}
  xywh {170 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
parameters[P_DOUBLE_LENS_M2]} {
  label m2
  callback {remake_map();}
  xywh {210 40 30 120} box ENGRAVED_BOX minimum 1 maximum 0 step 0.01 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
```

```
parameters[P_DOUBLE_LENS_BETA]} {
  label b
  callback {remake_map();}
  xywh {250 40 30 120} box ENGRAVED_BOX labelfont 12 minimum 1 maximum 0 step 0.01
hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
parameters[P_DOUBLE_LENS_X1]} {
  label x1
  callback {remake_map();}
  xywh {290 40 30 120} box ENGRAVED_BOX minimum 4 maximum -4 step 0.1 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
parameters[P_DOUBLE_LENS_Y1]} {
  label y1
  callback {remake_map();}
  xywh {330 40 30 120} box ENGRAVED_BOX minimum 4 maximum -4 step 0.1 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
parameters[P_DOUBLE_LENS_X2]} {
  label x2
  callback {remake_map();}
  xywh {370 40 30 120} box ENGRAVED_BOX minimum 4 maximum -4 step 0.1 hide
  code0 {\#include <GL_models.H>}
}
Fl_Value_Slider {parameters.planes[plane].models[M_DOUBLE_LENS].
parameters[P_DOUBLE_LENS_Y2]} {
  label y2
  callback {remake_map();}
  xywh {410 40 30 120} box ENGRAVED_BOX minimum 4 maximum -4 step 0.1 hide
  code0 {\#include <GL_models.H>}
}
  }
}
```

B.7 Animaciones, `anim.pl`

Para la generación de animaciones se escribió un *script* que requiere varios argumentos:

- **plantilla** Un archivo de definición de parámetros donde los parámetros a modificar se han reemplazado con `prm0`, `prm1`, ...
- **valor inicial 0** Valor inicial para el parámetro 0 (`prm0`)
- **valor final 0** Valor final para el parámetro 0
- **paso 0** Pasó para el parámetro 0 (debe ser positivo)

...

- **valor inicial** *i* Valor inicial para el parámetro *i*
- **valor final** *i* Valor final para el parámetro *i*
- **paso** *i* Paso para el parámetro *i* (puede tener cualquier valor)

Al ejecutar el *script* como `anil.pl plantilla.gld`, este produce produce varios archivos `plantilla-nnn.gld`, con `nnn` iniciando en `001` y aumentando hasta donde sea necesario para que el *primer* parámetro alcance el valor final que se ha indicado. Para los demás parámetros *no* se utiliza el valor final de los mismos, solo su valor inicial y el paso correspondiente. Estos archivos pueden ser cargados mediante `GLcp plantilla-nnn.gld` para producir un gráfico `plantilla-nnn.jpg`, de acuerdo con las especificaciones de la plantilla original y el correspondiente valor de los parámetros que varían.

```
#!/usr/bin/perl -w
# (c) 1999 Marcelo Magallón <mmagallo@efis.ucr.ac.cr>
#
# This script is free software.  It may be freely redistributed and
# modified under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 2 of the
# License, or (at your option) any later version.

(my $base = my $template = shift @ARGV) =~ s/\.[^.]*$//;
my $ext = $&;
my @p_i;
my @p_f;
my @p_s;
my $n;

while (@ARGV) {
    push @p_i, shift @ARGV;
    push @p_f, shift @ARGV;
```

```
    push @p_s, shift @ARGV;
}

$n = 1;
while ($p_i[0] <= $p_f[0]) {
    open TEMPLATE, $template;
    open OUTFILE, ">$base-" . sprintf("%03d", $n) . "$ext";

    while (<TEMPLATE>) {
        for (my $i = 0; $i <= $#p_i; $i++) {
            $p = "prm$i";
            s/$p/$p_i[$i]/;
        }
        print OUTFILE;
    }
    print OUTFILE "\n2 $base-" . sprintf("%03d", $n) . ".jpg"

    close OUTFILE;
    close TEMPLATE;

    for (my $i = 0; $i <= $#p_i; $i++) {
        $p_i[$i] += $p_s[$i];
    }
    $n++;
}
```

Por ejemplo, si se tiene una plantilla de parámetros `cr.gld`, se puede ejecutar:

```
% anim.pl cr.gld 0 1 0.1
% for f in cr-???.gld ; do GLcp $f ; done
```

lo cual produce diez archivos: `cr-001.jpg`, ... , `cr-010.jpg`.

Para convertir los gráficos generados utilizando este *script* en una animación digital para su posterior reproducción, es posible emplear, por ejemplo, el codificador MPEG-1 de Berkeley [Pla95]. Para el ejemplo considerado, un archivo de parámetros `cr.param` como el siguiente es suficiente:

```
PATTERN                 IBBPBBPBBPBBPBB
OUTPUT                  anim.mpg
GOP_SIZE                30
SLICES_PER_FRAME        1
FORCE_ENCODE_LAST_FRAME

BASE_FILE_FORMAT        YUV
YUV_SIZE                335x400

INPUT_DIR
INPUT_CONVERT           convert * yuv:-

INPUT
cr-*.jpg                [001-010]
END_INPUT

PIXEL                   FULL
RANGE                   10
PSEARCH_ALG             EXHAUSTIVE
BSEARCH_ALG             CROSS2
REFERENCE_FRAME         DECODED
IQSCALE                 8
PQSCALE                 10
BQSCALE                 25
```

7. References

[ASW98a] ABDELSALAM, H. M., P. SAHA y L. L. R. WILLIAMS: *Non-parametric reconstruction of cluster mass distribution from strong lensing - Modelling Abell 370.* Monthly Notices of the Royal Astronomical Society, 294:734+, Marzo 1998.

[ASW98b] ABDELSALAM, H. M., P. SAHA y L. L. R. WILLIAMS: *Nonparametric Reconstruction of Abell 2218 from Combined Weak and Strong Lensing.* Astronomy Journal, 116:1541–1552, Octubre 1998.

[BC98] BUOTE, D. A. y C. R. CANIZARES: *X-ray isophote shapes and the mass of NGC 3923.* Monthly Notices of the Royal Astronomical Society, 298:811–823, Agosto 1998.

[BMM99] BLAIN, A. W., O. MOLLER y A. H. MALLER: *Statistical lensing by galactic discs.* Monthly Notices of the Royal Astronomical Society, 303:423–432, Febrero 1999.

[BN86] BLANDFORD, R. y R.ÑARAYAN: *Fermat's principle, caustics, and the classification of gravitational lens images.* Astrophysical Journal, 310:568–582, Noviembre 1986.

[Ein11] EINSTEIN, ALBERT: *Uber den Einfluss der Schwerkraft auf die Ausbreitung des Lichtes.* Annalen der Physik, 35:898–908, 1911.

[FPM+88] FORT, B., J. L. PRIEUR, G. MATHEZ, Y. MELLIER y G. SOUCAIL: *Faint distorted structures in the core of A 370 - Are they gravitationally lensed galaxies at z about 1?* Astronomy and Astrophysics, 200:L17–L20, Julio 1988.

[Fru98] FRUTOS, FRANCISCO: *Die interaktive Visualisierung von Gravitationslinsen.* Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften, Eberhard-Karls-Universität zu Tübingen, Alemania, 1998.

[Fug89] FUGMANN, W.: *Galaxies near distant quasars - Observational evidence for statistical gravitational lensing. II.* Astronomy and Astrophysics, 222:45–48, Setiembre 1989.

[GN88] GROSSMAN, S. A. y R.ÑARAYAN: *Arcs from gravitational lensing.* Astrophysical Journal Letters, 324:L37–L41, Enero 1988.

[Got81] GOTT III, J. R.: *Are heavy halos made of low mass stars - A gravitational lens test.* Astrophysical Journal, 243:140–146, Enero 1981.

[Hai98] HAITZLER, CARSTEN: *The Imlib Programmers Guide.* http://www.rasterman.com/imlib/, 1998.

[Har88] HARWIT, MARTIN: *Astrophysical concepts*, capítulo Stars, páginas 309–311. Astronomy and astrophysics library. Springer-Verlag, New York, 2da edición, 1988.

[IHC+89] IRWIN, M. J., P. C. HEWETT, R. T. CORRIGAN, R. I. JEDRZEJEWSKI y R. L. WEBSTER: *Photometric variations in the Q2237 + 0305 system - First detection of a microlensing event.* Astronomy Journal, 98:1989–1994, Diciembre 1989.

[KF96] KEMPF, RENATE y CHRIS FRAZIER (editores): *OpenGL reference manual: the official reference document to OpenGL, version 1.1.* Addison Wesley Developers Press, Reading, Massachusetts, 2da edición, 199 ..

[KS93] KAISER, N. y G. SQUIRES: *Mapping the dark matter with weak gravitational lensing.* Astrophysical Journal, 404:441–450, Febrero 1993.

[LF94] LEBRUN, MAURICE J. y GEOFF FURNISH: *The PLplot Plotting Library Programmer's Reference Manual Version 5.0.* Institute for Fusion Studies, University of Texas at Austin, 1994. http://emma.la.asu.edu/plplot/.

[NB98] NARAYAN, RAMESH y MATTHIAS BARTELMANN: *Lectures on Gravitational Lensing.* En *Formation of Structure in the Universe.* Cambridge Univ. Press, 1998. http://cfa-www.harvard.edu/~narayan/papers/JeruLect.ps.

[NHM97] NIELSON, GREGORY M., HANS HAGEN y HEINRICH MÜLLER (editores): *Scientific Visualization: overviews, methodologies and techniques.* IEEE Computer Society, Los Alamitos, California, 1997.

[Pla95] PLATEAU RESEARCH GROUP: *Berkeley MPEG-1 Video Encoder User's Guide.* Computer Science Division University of California, Berkeley, California, 1995. ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg/bmt1r1.tar.gz.

[RH94] RODRIGUES-WILLIAMS, L. L. y C. J. HOGAN: *Statistical association of QSO's with foreground galaxy clusters.* Astronomy Journal, 107:451–460, Febrero 1994.

[Sch94] SCHRAMM, T.: *A toolbox for general elliptical gravitational lenses.* Astronomy and Astrophysics, 284:44–50, Abril 1994.

[SEF92] SCHNEIDER, P., J. EHLERS y E. E. FALCO: *Gravitational Lenses*, volumen XIV de *Astronomy and Astrophysics Library.* Springer-Verlag, Berlin Heidelberg New York, 1992.

[SES99] SWEET, MICHAEL, CRAIG P. EARLS y BILL SPITZAK: *FLTK 1.0.4 Programming Manual.* http://www.fltk.org/, 1999. Revision 11.

[Sha64] SHAPIRO, I. I.: *Fourth test of general relativity.* Phys. Rev. Lett., 13:789–791, 1964.

[SS95a] SCHNEIDER, P. y C. SEITZ: *Steps towards nonlinear cluster inversion through gravitational distortions I. Basic considerations and circular clusters.* Astronomy and Astrophysics, 294:411–431, Febrero 1995.

[SS95b] SEITZ, C. y P. SCHNEIDER: *Steps towards nonlinear cluster inversion through gravitational distortions II. Generalization of the Kaiser and Squires method.* Astronomy and Astrophysics, 297:287+, Mayo 1995.

[SSB+98] SEITZ, S., R. P. SAGLIA, R. BENDER, U. HOPP, P. BELLONI y B. ZIEGLER: *The z=2.72 galaxy cB58: a gravitational fold arc lensed by the cluster MS1512+36.* Monthly Notices of the Royal Astronomical Society, 298:945–965, Agosto 1998.

[TOG84] TURNER, E. L., J. P. OSTRIKER y J. R. GOTT III: *The statistics of gravitational lenses - The distributions of image angular separations and lens redshifts.* Astrophysical Journal, 284:1–22, Setiembre 1984.

[TWV90] TYSON, J. A., R. A. WENK y F. VALDES: *Detection of systematic gravitational lens galaxy image alignments - Mapping dark matter in galaxy clusters.* Astrophysical Journal Letters, 349:L1–L4, Enero 1990.

[Tys88] TYSON, J. A.: *Deep CCD survey - Galaxy luminosity and color evolution.* Astronomy Journal, 96:1–23, Julio 1988.